ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Electricidad y Computación

“ Un nuevo modelo BM3D-RNCA para mejorar la estimación de la imagen libre de ruido producida por el método BM3D”

Trabajo de Titulación Previo a la Obtención del Título de:

MAGISTER EN CIENCIAS DE LA COMPUTACIÓN

Presentado por:

STALIN FRANCIS QUINDE

Guayaquil Ecuador

2019

Agradecimiento

A mi familia, amigos y docentes de la maestría en Ciencias de la Computación de la ESPOL, por su apoyo y colaboración que me han brindado durante este proceso.

Dedicatoria

A mis hijos William, Iveth y Christopher esperando que esta meta alcanzada sirva de inspiración para luchar por sus propios sueños.

Stalin

Tribunal de graduación expresa

PhD. TITULAR	PhD. ALTERNO
PhD. TUTOR	PhD. CO-TUTOR

Declaración Expresa

"La responsabilidad del contenido de este Informe de Proyecto, me corresponde exclusivamente; y el patrimonio intelectual del mismo, a la ESCUELA SUPERIOR POLITÉCNICA DEL LITORIAL".

Art. 12 del Reglamento de Graduación

Stalin Adalberto Francis Quinde

Resumen

La estimación de una imagen que realiza el algoritmo BM3D, es un punto de partida para lograr que los dispositivos de captura de imágenes (DCI) imiten el funcionamiento de la vista y cerebro humano; la capacidad de aprender que tiene el algoritmo de una Red neuronal convolucional con topología autoencoder (RNCA), le permite reconstruir detalles poco o nada visibles en una imagen; gracias a estos algoritmos los DCI más allá de simplemente capturar una imagen actualmente son capaces de tomar decisiones sobre lo que se quiere observar o analizar en la imagen; sin embargo, estas herramientas de software que realizan estas interesantes e importantes funciones, tienen un rango de acción muy amplio por lo que están siendo constantemente evaluadas a fin de encontrar las mejores combinaciones y configuraciones que ayuden a cubrir todas las posibilidades de aplicación de la forma más productiva posible. El presente trabajo es un aporte más a la evaluación de estas herramientas donde primero se analizó el ruido como una de las razones más frecuentes por la que en una imagen capturada el contenido no puede ser identificado parcial o totalmente; luego se hizo una revisión de los métodos, técnicas y algoritmos creados y utilizados en el estado del arte del procesamiento de imágenes para recuperar estos contenidos poco o nada visibles; y finalmente, se creó un nuevo filtro que se lo llamo BM3D-RNCA el cual combinó dos exitosas técnicas y algoritmos de eliminación de ruido basado en conocimiento previo de la imagen para poderla restaurar. Los dos algoritmos fueron el Block matching 3D (BM3D) y la Red neuronal artificial convolucional autoencoder (RNCA), los resultados obtenidos de la evaluación a este nuevo filtro, indicaron a través del indicador PNSR, que la combinación de estos algoritmos logró un mejor resultado a los obtenidos individualmente en niveles altos de ruido, considerados así a imágenes afectadas con ruido blando Gaussiano aditivo(RBGA) que tiene desviación estandar de $\sigma > 40$ donde para $\sigma = 100$ se logró una pérdida de 50 % en un tiempo promedio máximo de 204.3 segundos, utilizando 400 epoch y 5 batch.

Palabras claves: Block matching 3D, Red Neuronal Convolucional autoencoder, Ruido aditivo blanco Gaussiano.

Abstract

The estimation made by the BM3D algorithm of an image is a starting point to achieve that photographic capture devices mimic the operation of the sight and human brain; the capacitance of the Neuronal Convolutional Network with topology autoencoder (RNCA) to train and use prior knowledge to rebuild little or nothing visible in an image, has made these devices can they go beyond simply capturing an image, making decisions about what wants to observe or analyze in the image; however these software tools who perform these interesting and important functions, have a very wide range of action broad so they are constantly being evaluated in order to find the best combination and configurations that help cover all the possibility of the form more productive as possible. The present work is one more contribution to the evaluation of these tools where noise was first analyzed as one of the most frequent reasons whereby in a captured image the content can not be identified as partial or totally; then a review of the methods, techniques and algorithms created was made and used in the state of the art of image processing to recover these little or no visible content; and finally, a new filter was created, baptized as BM3D-RNCA that combined two successful techniques and noise elimination algorithms based on prior knowledge of the image (Block Matching 3D and Neural Network artificial convolutional autoencoder) to restore it. The results obtained of the evaluation to this new filter, indicated through the PNSR indicator, that the The combination of these algorithms achieved a better result than those obtained individually. in high levels of noise, considered to be affected images with Additive Noise Gaussian white (RABG) that has standard deviation of $\sigma > 40$ where it was achieved a loss of 50 % in a time of maximum 204.3 seconds.

Keywords: Block matching 3D, Neural Network Convolutional autoencoder, Noise Gaussian white additive.

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1

Introducción

El cerebro utiliza su capacidad cognitiva para asociar detalles muy pequeños de una imagen borrosa a las cientos o miles de imágenes que están almacenadas en su memoria, esta capacidad humana es la que sigue motivando la búsqueda incansable para hallar filtros de eliminación de ruido más inteligentes.

En el estado del arte del procesamiento de imágenes, la eliminación total del ruido en una imagen, es uno de los objetivos aun pendientes de lograr; el cerebro humano es capaz de identificar objetos en escenarios muy difusos utilizando su capacidad cognitiva para asociar detalles muy pequeños de una imagen borrosa a las cientos o miles de imágenes que están guardadas en su memoria, esta capacidad humana es la que sigue motivando la búsqueda para hallar filtros de eliminación de ruido más inteligentes.

Uno de los algoritmos más destacados en el estado del arte del procesamiento de imágenes es el llamado BM3D, el cual aplica de forma secuencial la transformada Wavelet discreta (TWD) y el filtro de Wiener a una imagen con ruido. Este último filtro utiliza un enfoque estadístico, y asume un conocimiento de las propiedades espectrales de la imagen y del ruido, para con esta información estimar la imagen original. Esta combinación de filtros ha logrado buenos resultados para imágenes afectadas por un ruido aditivo blanco Gaussiano cuya distribución normal no supere la desviación estandar (σ) de 40 decibelios, ésta es la razón por lo que este algoritmo ha sido y aún sigue siendo utilizado como punto de partida en varias investigaciones del estado del arte del procesamiento de imágenes.

Como punto de partida, esta investigación utiliza el algoritmo BM3D, con el cual se estima la imagen libre de ruido, y dicha estimación es la imagen de entrada de la red neuronal convolucional con topología autoencoder (RNCA), la cual se encarga de extraer y mapear las características de imágenes similares a la imagen estimada, para reconstruir los bordes contornos y texturas que en el proceso de eliminación del ruido se

perdieron. Por este trabajo conjunto, entre las dos herramientas, es que el nuevo modelo toma el nombre de BM3D-RNCA. Finalmente se demostró la superioridad en cuanto a lograr un mejor resultado en la reducción de ruido con una distribución (σ) mayor, pero también se analizó la desventaja en cuanto al tiempo de procesamiento y la exigencia de mayores recursos computacionales.

En este capítulo introductorio se brinda el conocimiento preliminar necesario para poder comprender los elementos y la lógica detrás del algoritmo base BM3D [16], no se toca el tema del filtro de Wiener que es abordado de forma más profunda en el capítulo 3; también se abordan conceptos generales básicos sobre los tipos de redes neuronales artificiales y se profundiza en el capítulo 4 sobre las Redes Neuronales Convolucionales la cual va a ser utilizada en la investigación.

La salida del algoritmo BM3D servirá como entrada al modelo de red neuronal convolucional autoencoder (RNCA), estas dos herramientas con sus respectivas adaptaciones serán los dos elementos claves que permitirán a esta investigación construir una herramienta más efectiva que a más de reducir el ruido con densidades mayores a las logradas en el estado del arte $\sigma < 40$ [5], pueda recuperar las intensidades perdidas en el proceso tradicional de estimación.

1.1 Objetivo

Encontrar una nueva configuración y combinación BM3D-RNCA, que supere los resultados del algoritmo BM3D, sometiendo estos resultados a una red neuronal convolucional autoencoder (RNCA), y así aportar al estudio y evaluación permanente de estos dos algoritmos.

1.2 Trabajos relacionados

A más de este trabajo de investigación, abundante es el estado del arte que aporta a la evaluación y selección de métodos, técnicas y algoritmos para eliminar de forma efectiva y eficiente la señal de ruido producido por un dispositivo de captura de imágenes (DCI) de uso común. El fenómeno del ruido (*noise*) junto a la difuminación (*blur*) son las dos principales limitaciones que tienen los DCI para obtener una imagen clara y precisa según Antoni Buades, Bartomeu Coll y Jean-Michel Moel[3], estos autores hacen una revisión de algoritmos tradicionales para la eliminación del ruido y proponen un algoritmo nuevo basado en el método de la media no local (*Non Local Means*). De los

diferentes modelos de ruido existentes [19], uno de los modelos de ruido ampliamente tratados es el ruido aditivo blanco Gaussiano(AWGN), el cual es utilizado en muchas investigaciones que intentan dar solución a la eliminación de ruido para diferentes tipos de imágenes y métodos de captura, así Julie Delon y Antoine Houdard[8] han estudiado la eliminación de ruido basado en modelos estadísticos que utilizan la distribución de Gauss o función de distribución probabilística (PDF:)), la cual consiste en predecir el valor de un elemento de la imagen tal que se conozca la media y la varianza de la muestra; para optimizar los procesos, los cálculos se realizan dividiendo la imagen en parches, este modelo basado en parches también conocido como modelo no-local ha creado un nuevo paradigma en procesamiento de imágenes no solo para solucionar problemas de ruido sino también otros problemas de restauración como interpolación, edición y sintetización de imagen, los parches que se extraen de una imagen son grupos de pixeles tomadas de una pequeña área de la imagen por lo general de forma cuadrada y centradas en un pixel de referencia.

La investigación del presente trabajo tuvo como punto de partida el artículo [5] que describe de manera detallada los modelos matemáticos de la Transformada Discreta de Wavelet (DWT), Transformada Discreta de Fourier (DFT), Transformada Discreta del Coseno (DCT) y Wiener; todos esos filtro son utilizados de forma directa y alternativa para implementar el algoritmo BM3D, también en este artículo se explica la forma en que estos filtros interactúan para lograr la reducción del ruido. Varias investigaciones a más de presente trabajo se han valido del algoritmo BM3D para crear aplicaciones más efectivas adicionando otros recursos innovadores.

El trabajo que inspiró esta investigación, utiliza el algoritmo BM3D, para restaurar una imagen RGB, tomada en un ambiente de poca iluminación, pero con una imagen alineada tomada con una cámara infrarroja [30]. Esta investigación aprovechó la correlación entre las bandas de color RGB y el NIR y utilizó la información de la banda menos ruidosa NIR, para con la técnica del filtro colaborativo restaurar la textura en un alto rango espacial de frecuencia mientras mantiene el balance del color, superando a otros métodos en el estado del arte en términos de PSNR, calidad de textura y fidelidad del color. La fortaleza que tienen las bandas espectrales al proveer un rica representación de una escena también fue aprovechada por Angel Sappa et al en [1] donde utilizan una red neuronal convolucional con 2 canales para medir la similitud entre dos parches de imágenes de diferentes espectro, la salida de la red fue un valor escalar que indicaba la distancia entre los parches de entrada, esta red estuvo compuesta por una serie de capas de tipo convolución, ReLU y max-pooling que finalizaba con una capa lineal, a

este tipo de red se le llamo red métrica y esta investigación fue luego retomada [27] para reducir los costos de hardware comparando los nuevos resultado con los obtenidos [1] y con los obtenidos al aplicar la detección de características clásica llamada SIFT.

Parámetros exactos con los que trabajó el BM3D en un conjunto de imágenes reales es presentado en [5], en esta investigación se aplica a ruido con ($\sigma = 35$) obteniendo buenos resultados, también muestra los resultados aplicados a ruido con ($\sigma = 50$), y esto ya se considera un alto nivel de ruido donde se notó pocas distorsiones, en una prueba extrema se utilizó una imagen con un nivel de ruido ($\sigma = 100$) sobre una imagen de Lena, y aunque muchos de los detalles aparece borroso las rayas del vestido fueron fielmente restauradas. Otra de las innovaciones que se sumo al algoritmo BM3D fue la utilización del Análisis de Componentes Principales [7], aunque visiblemente los resultados no son tan expresivos, se puede ver que al calcular el PSNR, se obtiene una ligera mejora.

Una de las incursiones exitosas de mejorar la eliminación de ruido utilizando las redes neuronales fue el trabajo reciente realizado por Yi-Quing Wang [32] donde crea una pequeña red neuronal para utilizarla como complemento al algoritmo BM3D. Este modelo se lo llamó SSaNN (*Self-Similarity and Neural Networks*) y combina la efectividad del BM3D al trabajar con patrones estructurados a gran escala, con la efectividad que han demostrado las redes neuronales al trabajar bien con patrones de textura a pequeñas escalas.

Una arquitectura de red neuronal convolucional parecida a la utilizada en esta investigación para restaurar una imagen, ha sido propuesta en [24], esta red utiliza la técnica de interpolación bicubica obtener una imagen restaurada a la salida, y ha sido entrenada con imágenes capturadas en el espectro de luz diferente a la visible, las imágenes tomadas en el espectro de luz infrarojo proveen extra información a la imagen capturada en el espectro visible, en un entrenamiento se utilizaron 98 imágenes de 640×512 de resolución tomadas en la mañana y en la noche sumadas a 40 imágenes públicas de 640×480 de resolución, en total fueron 138 imágenes las cuales fueron rotadas y reflejadas vertical y horizontalmente para incrementar el número de imágenes de entrenamiento.

1.3 Estructura de la tesis

En lo que resta de este trabajo investigativo el contenido de este documento esta estructurado como se detalla a continuación.

Capítulo 2, presenta un enfoque claro de los conceptos más básicos utilizados para

desarrollar la investigación, comenzando con el objeto de estudio que es la imagen y siguiendo con todas las transformaciones a la cual esta es sometida en su estado más básico, aquí se presentan los modelos matemáticos que serán utilizados en todo el desarrollo del documento; también se aborda de manera detallada los fundamentos teóricos y matemáticos que describen el ruido blanco Gaussiano aditivo, y la métricas que permitirán medir su presencia o ausencia de la imagen tratada; finalmente se analiza el funcionamiento matemático y estadístico del filtro de Wiener, componente que hace la diferencia con los resultados obtenidos en los demás filtros de eliminación de ruido.

Capítulo 3, se da una interpretación personal del funcionamiento del algoritmo BM3D, resaltando la función que tiene la combinación de los dos filtros que la componen y analizando sus resultados con otros filtros que lo preceden.

Capítulo 4, estudia la arquitectura de las redes neuronales artificiales tratando de explicar su funcionamiento en forma general para profundizar en la red neuronal convolucional que es la que se utiliza en este investigación.

Capitulo 5, se utiliza la teoría desarrollada en los capitulo 4, 5 y 6 para de forma práctica construir el nuevo modelo BM3D-RNCA, describiendo muy minuciosamente su estructura y su funcionamiento;

Capítulo 6, describe y tabula los resultados de la evaluación de la nueva combinación de algoritmos BM3D-RNCA, junto a la evaluación de manera individual del BM3D y el RNCA, estos resultados también se muestra en cuadros comparativo y con gráficos de linea los PSNR obtenidos por tipo de algoritmo y por nivel de ruido.

Finalmente, se concluye esta investigación resaltando los buenos resultados obtenidos al combinar los algoritmos BM3D y RNCA, en comparación a los resultados obtenidos por el tradicional algoritmo BM3D y el algoritmo RNCA aplicados de forma individual, pero también se describe las desventajas en cuanto a la necesidad de recursos y tiempo al utilizar redes neuronales convolucionales.

CAPÍTULO 2

Procesamiento de imágenes

Este capítulo describe de una forma simple en que consiste el procesamiento, como se lo realiza y que beneficios tiene. Actualmente la necesidad de procesar las imágenes digitales proviene del interés de dos áreas de aplicación que son: 1) mejorar y ampliar la información gráfica para poder ser interpretada por los seres humanos a fin de mejorar la toma de decisiones; 2) obtener datos de la imagen para que una máquina autónoma pueda almacenarla, transmitirla y representarla.

En el procesamiento de una imagen el objeto de interés es la imagen y el procesamiento de esta consiste en cambios que se dan ya sea desde el momento en que es creada por el dispositivo de captura (ejemplo el ruido incorporado), como por manipulación directa aplicando los filtros nombrados en la introducción de este trabajo.

El resto de este capítulo está organizado de la siguiente manera: la sección 2.1 habla de la imagen, que es y cómo se representa matemáticamente y computacionalmente; la sección 2.2, describe el ruido en la imagen, su modelo matemático y como se combina con la imagen; la sección 2.3, describe la matriz kernel como un elemento básico utilizado en el filtrado de una imagen, y se da dos ejemplos simples pero prácticos del filtra de una imagen; la sección 2.4 describe la estructura y funcionalidad del filtro de la trasformada discreta Wavelet el cual es el primer filtro utilizado en el algoritmo BM3D; la sección 2.5 explica los fundamentos matemáticos el ruido aditivo blanco Gaussiano (AWGN), para comprender el por que de su utilización en el procesamiento de imágenes; finalmente las sección 2.6 describe el modelo matemático del filtro de Wiener, destacando sus buenos resultados alcanzados en la estimación de una imagen afectada por el AWGN.

2.1 La imagen

Visualmente una imagen se la percibe como un conjunto de colores derivados de 3 colores básicos (rojo, verde y azul), la intensidad de cada uno de estos colores básicos matemáticamente han sido valorados entre 0 y 255, lo que significa que una imagen puede ser representada por 256^3 colores distintos. Para manipular digitalmente una imagen tanto en pequeña como a gran escala, es necesario contar con un modelo matemático de la imagen que permita comprender su estructura y su comportamiento ante la aplicación de varias transformaciones.

Una imagen a color matemáticamente está representada por tres matrices, y cada matriz es un conjunto de valores Y que están dados en un rango de 0 a 255, donde 0 corresponde a la menor intensidad de color y 255 a la mayor intensidad de color. Para poder acceder a cada valor se considera que la imagen esté en el plano bidimensional (x,y), donde x,y $\subseteq \mathcal{X} \rightarrow \mathbb{R}$ es decir que si se quiere acceder al primer valor de la imagen se la encontrará en la posición (0,0).

La figura 2.1-(*a*) muestra una imagen como la percibe el ser humano con su sentido de la vista, y la figura 2.1-(*b*) muestra la imagen representada como una matriz que es como lo maneja el computador en su memoria.

(a) Imagen

63	61	59	59	61	65	70	74	68	64	60	59	60	61	60	59	58	57	57	59	63
60	60	60	61	62	64	66	68	64	61	58	58	59	59	58	57	59	61	62	62	62
58	59	61	63	63	63	61	61	64	63	61	61	62	63	61	60	56	59	62	60	57
59	60	62	62	62	60	58	57	62	62	62	63	64	64	63	62	57	59	60	59	56
62	62	61	61	60	59	58	58	58	59	60	60	60	60	60	60	64	62	60	61	63
64	63	62	61	60	60	60	60	60	61	62	62	61	60	61	63	65	62	59	60	63
63	63	63	63	63	63	62	62	65	66	67	65	63	63	65	68	64	62	61	60	60
62	63	64	66	66	65	64	63	66	66	66	64	62	62	65	68	66	67	67	64	60
60	62	62	60	60	63	64	61	66	62	59	60	65	66	61	56	64	64	64	62	61
62	65	66	64	63	65	63	59	65	63	61	63	66	67	65	62	63	63	63	62	61
64	67	69	67	66	66	63	59	59	59	60	61	62	63	64	64	61	62	62	62	62
65	68	68	66	66	67	65	61	57	59	61	60	59	59	62	64	60	60	61	61	62
65	66	65	62	62	66	67	65	62	64	66	64	61	60	63	67	61	60	60	61	62
65	66	64	60	60	65	67	67	65	66	66	63	60	59	62	64	62	60	59	60	62
65	67	65	62	62	65	66	64	65	65	63	61	60	60	61	63	64	61	58	59	62
66	68	68	65	64	65	64	61	69	67	65	64	64	65	66	66	66	62	58	59	62
58	62	66	68	66	65	66	67	68	68	64	61	62	65	64	60	63	61	59	57	57
61	63	66	68	68	67	67	67	68	68	65	62	63	67	66	62	64	63	61	60	59
64	64	65	68	70	70	68	66	67	68	66	63	64	67	67	63	66	65	64	63	62
65	64	64	67	70	70	67	64	65	67	66	63	64	66	66	63	65	65	65	64	64
64	63	63	66	69	69	66	62	62	65	65	63	62	64	64	62	63	64	64	64	63
63	63	64	66	67	67	65	63	61	65	66	63	61	63	63	61	62	62	62	62	62
62	65	67	68	67	66	65	65	61	66	68	65	62	62	60	62	62	61	61	61	61
63	66	70	70	67	65	66	67	63	67	69	66	63	63	62	61	62	62	61	61	61
67	63	63	69	72	69	66	66	69	69	66	62	63	65	62	56	64	63	61	59	58
62	59	60	65	66	63	61	62	68	70	68	64	64	67	65	61	65	65	64	63	62
63	63	65	68	67	64	64	66	65	68	68	66	64	65	66	65	65	65	65	65	65
66	68	71	71	69	66	68	70	63	66	68	65	62	61	63	64	61	62	63	64	63
64	67	69	68	65	64	66	67	65	67	68	66	63	60	61	64	60	61	62	63	63
65	68	69	66	64	66	66	66	69	68	68	68	66	62	62	65	63	64	64	64	63

(b) Representación como una matriz

Figura 2.1: (a) Imagen en escala de gris (b) Representación en forma de matriz, cada número representa un pixel y su valor representa la intensidad del color que va de 0 (negro) hasta 255 (blanco).

Cada valor y de la matriz puede obtenerse con una función que reciba una coordenada

$x \in X \to \mathbb{R}^2$, así el modelo matemático de la imagen quedará definido como:

$$y = f(x) \tag{2.1}$$

2.2 Ruido en una imagen

Los dispositivos de captura de imágenes de uso cotidiano aún no son capaces de obtener una imagen limpia que se puede representar con el modelo matemático de la ecuación (2.1), puesto que varios factores introducen imperfecciones que comúnmente se le llama ruido, estas se pueden producir ya sea al momento de la adquisición como al momento de la transmisión; el ruido afecta principalmente el desempeño visual y el análisis digital que se realiza de la señal o imagen.

Aunque el origen y la no estacionalidad del ruido es difícil de modelar, si se asume este ruido como estacionario y aditivo puede ser representado de la siguiente manera:

$$z(i) = y(i) + \sigma\varepsilon(i),\ i = 1, 2, \ldots, n-1 \tag{2.2}$$

donde $z(i)$ representa la señal con ruido, $y(i)$ representa la señal libre de ruido y ε es una variable aleatoria independiente que junto a σ representa la intensidad del ruido en $z(i)$. El ruido es modelado como una variable independiente que cambia según una función de densidad probabilística llamada también función de distribución normal o distribución Gaussiana $P(x)$ como se muestra en la ecuación 2.3.

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp^{\frac{-(x-\mu)^2}{2\sigma^2}} \tag{2.3}$$

donde $P(x)$ se distribuye a lo largo del dominio $x \in (-\infty, \infty)$ y concentra su mayor energía alrededor de un valor de referencia llamado media μ con un rango de *varianza* σ^2, como se puede observar en la figure 2.2.

El objetivo de modelar el ruido utilizando una función de distribución normal, se da porque se entiende que el ruido es una señal independiente que se reparte a través de toda la señal o imagen original con altas frecuencia y una estructura determinada que puede ser diferenciada de la señal o imagen original.

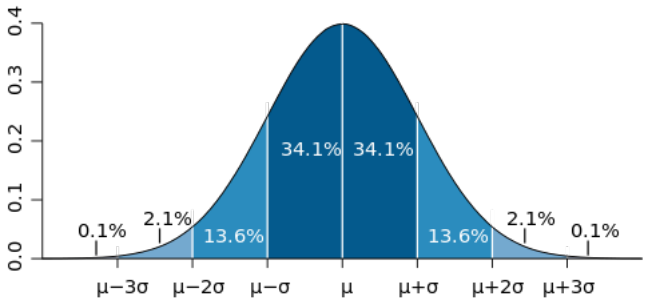


Figura 2.2: Función distribución normal o Gaussiana.

El efecto de aplicar un ruido a una imagen es simplemente una imagen de menor calidad que reduce la visibilidad del contenido. En la figure 2.3-(a) se muestra la imagen original, y en la figura 2.3-(b) se muestra la misma imagen afectada por un ruido de tipo Gaussiano, donde los detalles de la imagen no son legibles por efecto de los píxeles que no pertenecen a la imagen.

(a) Imagen sin ruido (b) Imagen con ruido

Figura 2.3: (a) Imagen original. (b) Imagen después de aplicar un ruido de tipo Gausiano.

2.3 Filtrando la imagen

Transformar una imagen con ruido para obtener mejoras significativas, involucra someter la imagen a un proceso de filtrado que consiste en alterar en la matriz que representa la imagen el valor de cada elemento (píxel), utilizando de forma iterativa e interactiva las cuatro operaciones básicas de suma, resta, multiplicación y división, contra otra matriz. Este proceso de filtrado toma el nombre de convolución y su modelo matemático es representado según la ecuación 2.4 que se muestra seguidamente.

$$(f * g)[t] = \int_{-\infty}^{\infty} f[\tau]g[t-\tau]d\tau \tag{2.4}$$

donde f representa la matriz de la imagen a procesar y g una pequeña matriz cuadrática llamada Kernel que barre la matriz de la imagen de izquierda a derecha y de arriba a

abajo, $(f * g)$ obtiene el peso ponderado de la función $f(\tau)$ en el momento t donde el peso es dado por $g(t - \tau)$ simplemente desplazada por la cantidad t. Como t cambia la función de peso acentúa diferentes partes de la función de entrada.

La ecuación de convolución es parte del modelo matemático de la imagen y describe el comportamiento que la imagen tiene ante este tipo de transformación; los cambios que se quieren lograr en la imagen dependen de los valores con que se configure la matriz kernel g.

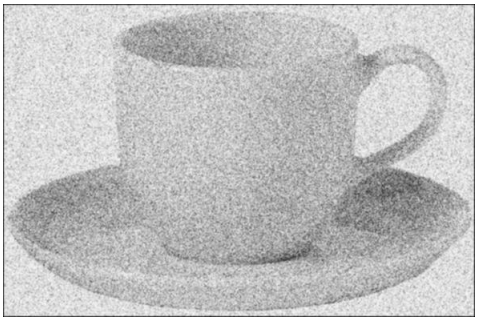
En el experimento de esta investigación, el procesamiento de las imágenes involucra aplicar la técnica del filtrado a fin de reducir o aumentar el valor de los pixeles según dos necesidades principales: 1) obtener información de la imagen o señal y 2) mejorar la visualización de la señal o imagen; pero en esta investigación específicamente se necesitó eliminar el ruido que corresponde a la necesidad de mejorar la visualización.

Uno de lo muchos esquemas para conformar una matriz que servirá de kernel g en el procesamiento de una imagen es el utilizado para lograr reducir el ruido de una imagen, en esta matriz los elementos deben ajustarse a la configuración de una distribución normal de Gauss como se muestra en la matriz 2.5.

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \frac{1}{16} \tag{2.5}$$

La matriz 2.5 cumple la función de un **filtro paso bajo** ya que permite el paso de las frecuencias más bajas y atenúa las frecuencias más altas, los resultados que se obtienen después de convolucionar la imagen con este kernel es la de eliminación de los pixeles atípicos o más pronunciados. En la figura 2.4- (a) se muestra la imagen con ruido donde se puede observar píxeles que no corresponden a la imagen (*píxeles con valores atípicos*), mientras que en la figura 2.4- (b) se observa la reducción de los píxeles atípicos.

(a) Imagen con ruido



(b) Imagen filtrada

Figura 2.4: (a) La imagen con ruido (b) Imagen después de aplicar el proceso de convolución utilizando un kernel con una distribución Gauss o filtro paso bajo.

Otra de las aplicaciones de los filtros es para detectar los bordes presentes en una imagen, para lo cual se utiliza un kernel de tipo **paso alto** con valores distribuidos como muestra la matriz (2.6). De estos dos kernel el de la derecha detecta los bordes verticales y el de la izquierda los bordes horizontales; en otras palabras, el proceso deja aquellos píxeles que mantienen una variación tanto vertical como horizontal con sus vecinos, y elimina aquellos que no cumplen con esta condición.

$$\begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \quad \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{2.6}$$

El efecto de aplicar estos dos kernels juntos es el que se presenta en la figura 2.5- (a) Imagen original sin ruido donde se puede visualizar los más pequeños detalles de su contenido, y la figura 2.5- (b) Imagen después de aplicar el proceso de convolución con las dos matrices 2.6 donde se observa cómo se resalta los bordes y se elimina los demás detalles.

(a) Imagen sin ruido

(b) Bordes de la imagen

Figura 2.5: (a) Imagen original; (b) Imagen después de aplicar el proceso de convolución con las dos matrices 2.6.

Estos filtros basados en kernels, trabajan en el dominio espacial, y en el caso del filtro de Gauss, que sirve para eliminar el ruido, tiene su desventaja en que también elimina detalles propios de la imagen como por ejemplo bordes y contornos resultando una imagen algo borrosa. El problema de la perdida de detalles ha sido y sigue siendo fuertemente investigada en el estado del arte del procesamiento de imágenes, buscando soluciones más efectivas de lo cual surge la transformada de Fourier que lleva la imagen a una representación de frecuencias. Es así que las más altas frecuencias que corresponden a la señal de ruido, pueden ser detectadas utilizando la tradicional transformada de Fourier, específicamente calculando la transformada discreta de Fourier(TDF).

Para la solución de los problemas reales utilizar TDF, no resulta tan efectivo, debido a que este método no brinda información temporal y esto resulta en que mucha información propia de la imagen sea eliminada, por tener valores de frecuencias muy parecidos a la información no deseada; es por eso que una solución alternativa para esta deficiencia es la utilización de la transformada WAVELET la cual ha brindado mejores resultados al conservar detalles de la imagen original.

2.4 Transformada discreta de Wavelet

Como se indicó en la sección anterior la Transformada Discreta de Wavelet (TDW) es una herramienta más efectiva que la Transformada de Fourier Discreta (DFT), la razón es que la transformada Wavelet es un método que mantiene la información tanto frecuencial como espacial al descomponer la señal en componente de alta y baja frecuencia.

La transformada Wavelet no utiliza las funciones base de seno y coseno que permanecen en todo el domino del tiempo, sino que hace uso de una familia de funciones $\psi \in L^2(\mathbb{R})$ llamada Orthonornal Wavelet. La idea es descomponer la señal principal en sub-señales que contengan los diferentes niveles de frecuencias de la imagen original. Las funciones Wavelet más comunes y que cumplen con la propiedad de ortogonalidad son: *daubechies*, *symlets*, *coiflets* y *discrete meyer* [26].

Para la descomposición a través de la transformada de Wavelet en esta investigación: primeramente, se escoge un tipo de Wavelet y el nivel de descomposición, un tipo de Wavelet utilizado para eliminar el ruido blanco Gaussiano es el biorthogonal 3.5 utilizado con un nivel N de 10. Después de la descomposición de la imagen se establece el valor límite para cada uno de los niveles. Finalmente es necesario reconstruir la imagen pero con los niveles previamente modificados, esto se lo realiza utilizando la inversa de la transformada Wavelet.

La transformada de Wavelet es representada matemáticamente con la siguiente sumatoria:

$$X(a,b) = \sum_{-\infty}^{\infty} x(t)\psi^*_{a,b}(t)dt \tag{2.7}$$

La descomposición que la transformada Wavelet realizada, crea imágenes de varias resoluciones, cada resolución es caracterizada por un coeficiente obtenido según la fórmula. El resultado del DWT es una descomposición en varios niveles, cada nivel es caracterizado por coeficientes de 'Aproximación' y de 'Detalles'.

$$D_1[n] = \sum_{k=\infty}^{\infty} h_d[k]x[2n-k] \tag{2.8}$$

$$A_1[n] = \sum_{k=\infty}^{\infty} l_d[k]x[2n-k] \tag{2.9}$$

donde n y k son los coeficientes en tiempo discreto y x presenta la señal [4].

La manera de remover ruido o reconstruir la señal original de una señal contaminada, en caso de una y dos dimensiones, usando los coeficientes Wavelet, que son resultado de la descomposición en el proceso de la transformación Wavelet, es eliminar los coeficientes pequeños que están asociados al ruido.

Después de eliminar los coeficiente que contienen el ruido la seña $x(t)$ es reconstruida con la siguiente ecuación:

$$x(t) = \sum_{m=1}^{L} \left[\sum_{k=-\infty}^{\infty} D_m(k)\psi_{m.k}(t) + \sum_{k=-\infty}^{\infty} A_l(k)\phi_{l,k}(t) \right] \tag{2.10}$$

2.4.1. Estimación del umbral señal original vs señal con ruido

Para eliminar el ruido, fue necesario seleccionar los coeficiente de Wavelet más pequeños que son los que están asociados con el ruido, así con los coeficientes restantes se puede reconstruir la imagen libre de ruido; para escoger los coeficientes que representan al ruido es necesario aplicar una técnica que determine este límite *thresholding*.

La umbralización Wavelet o método de comprensión son los más adecuados según Donoho y Johnstone [9], quienes han investigado sobre el nivel límite y sus tipos, ellos propusieron una estrategia no-lineal para definir los límites, según su investigación el

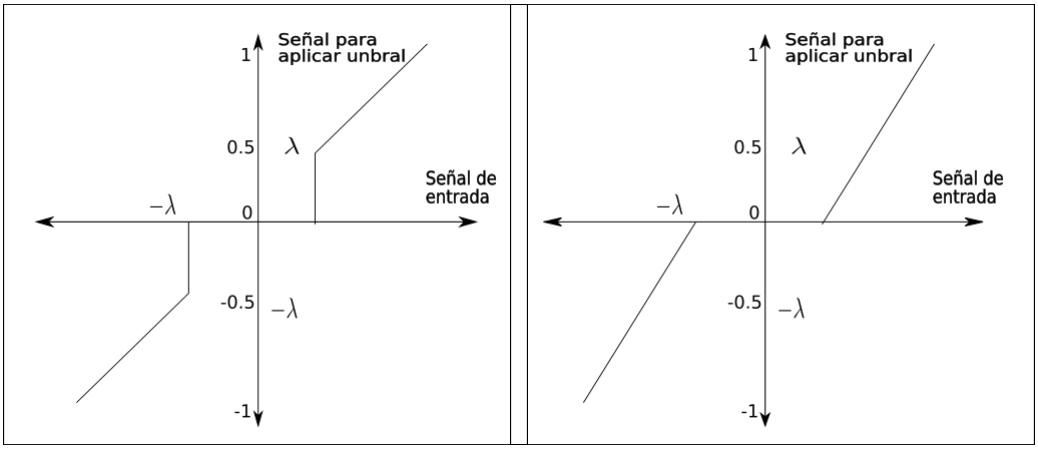
umbral puede ser aplicado implementando ya sea un método de umbralización hard o soft que también lo llaman contracción.

Cuando se hable de una umbralización hard, el coeficiente Wavelet bajo un valor dado son cambiado a cero (0), mientras que en una umbralización soft los coeficientes son reducidos hasta llevar a un valor del umbral definido. El valor del umbral es la estimación del nivel del ruido, el cual es generalmente calculado de la desviación standar de los coeficientes de detalle [9].

$$Hard\ threshold: \begin{cases} y = x, & \text{if } |x| > \lambda \\ y = 0, & \text{if } |x| < \lambda \end{cases} \tag{2.11}$$

$$Soft\ threshold: \begin{cases} y = sign(x), & |x| - \lambda \end{cases} \tag{2.12}$$

donde x es la señal de entrada y y es la señal después de aplicar la umbralización y λ es el valor del umbral; este valor es el que destruye, reduce o incrementa el valor del coeficiente Wavelet.



(a) (b)

Figura 2.6: (a) Hard thresholding; (b) Soft thresholding

Un punto importante para la utilización del método de umbralización es encontrar un valor aproximado para el umbral, y para este valor muchas han sido las propuestas cada una de las cuales requiere de la estimación del nivel ruido existente, la desviación standar de los valore de los datos pueden ser usados como estimadores, es así que Donoso y Johnstone [9] ha propuesto un buen estimador σ para la eliminación del ruido:

$$\sigma = \frac{median(d_{L-1,k})}{0{,}6745}, \quad k = 0, 1, \ldots, 2^{L-1} - 1 \tag{2.13}$$

donde L denota el número de niveles producido en la descomposición, y d son los valores de los coeficientes de detalle.

Con la estimación del ruido es posible calcular el valor del umbral λ, así los más conocidos algoritmos utilizados para esta selección son minimax, universal y rigorous sure propuestas también por Donoso y Johnstone [9].

En esta investigación se utilizará el universal que está dada por la fórmula:

$$\lambda_u = \sigma\sqrt{2\log(n)} \tag{2.14}$$

2.4.2. Ventajas de utilizar un arreglo 3D como una forma de contar con una muestra más grande

El método de unbralización de ruido universal ecuación (2.14) llamado también SQTWOLOG propuesto por Donoho y Johnstone, remueve el ruido de forma más eficiente mientras mayor sea el número de pixeles n que contengan la señal de ruido así si $z_1 \ldots .z_n$ representan los coeficientes Wavelet del ruido con $N(0,\sigma^2)$, entonces se puede expresar matemáticamente como:

$$\lim_{n\to\infty} P(z_i \leq \sigma\sqrt{2\log n}) = 1 \tag{2.15}$$

Esta expresión indica que la probabilidad de que el ruido sea reducido a cero es muy alta mientras más grande es la muestra, esto se debe a que el procedimiento umbralización universal está basado en un resultado asintótico, y por el contrario si la muestra es muy pequeña los resultados no son muy buenos [18].

2.5 Ruido aditivo blanco Gaussiano

En esta sección se brinda una explicación más profunda y detallada del ruido aditivo blanco Gaussiano introducido en la sección 2.2, con el objetivo de comprender el porque es utilizado ampliamente en el estado del arte del procesamiento de imágenes; se describirá su modelo matemático y su aplicación en señales e imágenes.

Ruido aditivo blanco Gaussiano es un modelo de ruido básico, su uso ha sido muy generalizado en el campo de la informática con la finalidad de imitar el efecto aleatorio natural del ruido que perturba las imágenes. Este tipo de ruido proviene de fuentes naturales tales como la vibración térmica de los átomos en los conductores que constituyen los dispositivos que capturan la imagen.

2.5.1. Modelo matemático

El modelo matemático que describe la imagen como un conjunto de intensidades de colores I_i descrito en la sección 2.1 tiene la forma descrita por la ecuación (2.1), donde $y(x) \equiv I_i$, es la intensidad de color correspondiente a una imagen ideal. En la vida real la mayoría de las imágenes capturadas siempre están afectadas por algún tipo de ruido, el cual altera varios de los valores I_i, a un nuevo valor Y_i; uno de los ruidos más frecuentes y que ha sido posible reproducir en el laboratorio es el Aditivo Blanco Gaussiano (AWGN) el cual de forma reducida se lo modela según la ecuación (2.16), donde Z es el valor independiente que se va a calcular según la función $\mathcal{N}$ de una media cero, y una varianza N :

$$Z_i = \mathcal{N}(0, N) \tag{2.16}$$

Tanto la señal del ruido como la señal de la imagen viajan por un mismo canal, cuya capacidad está dada por la ecuación (2.17), una imagen con ruido estará dado por una serie de valores de salida Y_i que dependen de la señal de la imagen real sumada a la señal del ruido.

$$Y_i = I_i + Z_i \tag{2.17}$$

El valor final de Z_i depende de la posición x y será calculado según la función $f(x|\mu, \sigma^2)$ mostrada en la siguiente ecuación:

$$f(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{2.18}$$

donde x es la intensidad de color, μ la media de las intensidades y σ es la desviación standar.

2.5.2. Medición del ruido en una imagen

Eliminar el ruido en su totalidad hasta ahora ha sido una tarea imposible cuando el ruido es producido de forma real, lo único que se ha logrado en el procesamiento de imágenes utilizando técnicas de eliminación de ruido es reducirlo a un nivel aceptable. Muchas veces los logros en reducción de ruido no se pueden percibir a simple vista por lo que se recurre a mediciones del nivel de ruido antes y después de aplicar la metodología de eliminación de ruido. La medición se la realiza a través de la Relación Señal Ruido de Pico PSNR (del ingles *Peak Signal-to-Noise Radio*), la unidad en que esta dado este parámetro es el decibeles (dB), como referencia la aplicación de un algoritmo

de eliminación de ruido con *block-matching* filtro 3D [5] ha logrado en una imagen con densidad de ruido de $\sigma = 35$ un nivel de eliminación de ruido de PSNR 31.21 dB.

El indicador PSNR resulta de la diferencia entre la imagen antes y después de aplicar el proceso de eliminación de ruido, por lo que primeramente es necesario calcular el error medio cuadrático [33] con la siguiente ecuación.

$$MSE = \frac{1}{MN} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} ||I(i,j) - K(i,j)||^2 \tag{2.19}$$

Donde $I(i,j)$ y $K(i,j)$ devuelven el valor de las intensidades de colores en las posiciones i, j de las imágenes a comparar; M corresponde a la cantidad de píxeles existente un una fila de la imagen y N la cantidad de píxeles existente en una columna de la imagen, MN es simplemente la cantidad de píxeles con que cuenta toda la imagen.

El valor MSE obtenido permite calcular el PSNR el cual es un valor en escala logarítmica calculado segun la siguiente ecuación.

$$PSNR = 10\dot{\log}_{10}\left(\frac{MAX_I^2}{MSE}\right) = 20\dot{\log}_{10}\left(\frac{MAX_I}{\sqrt{MSE}}\right) \tag{2.20}$$

En esta ecuación MAX corresponde al valor máximo que puede alcanzar un pixel, en este caso es 255.

2.6 Aporte del filtro de Wiener en la eliminación del ruido

Esta sección describe de la manera más simple posible el funcionamiento del filtro de Wiener y destaca el aporte de este filtro en los buenos resultados alcanzados en el filtrado de una imagen con alto nivel de ruido, logrando preservar aceptablemente características importantes como bordes, contornos y texturas; esto es posible gracias a su enfoque estadístico que mira a la señal de ruido y a la señal útil como procesos estocásticos posibles de predecir utilizando el conocimiento de las propiedades espectrales de las señales.

Es importante resaltar que en el algoritmo BM3D abordado en este trabajo de investigación y también en el nuevo algoritmo BM3D-RNCA propuesto, el filtro de Wiener se aplica para obtener una segunda estimación, por lo cual este filtro no utiliza como imagen de entrada la imagen original con ruido, sino la imagen estimada en una primera etapa aplicando un umbral duro (*hard-thresholding*) a los coeficientes obtenidos en la primera transformación [6].

2.6.1. Modelo matemático del filtro de Wiener

Una primera aproximación al entendimiento del modelo del filtro de Wiener es saber que el proceso consiste en minimizar el error cuadrático medio (MSE) entre una señal estimada $x[n]$, y una señal deseada $s[n]$; el filtro de Wiener estima la señal de ruido y la compara con la señal ruido causado por una señal Gaussiana estacionaria [17], ver figure 2.7.

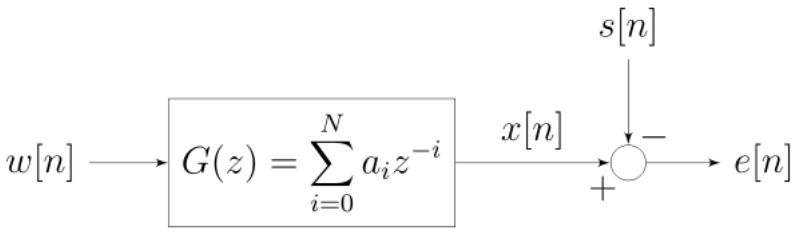


Figura 2.7: Filtro Wierner utilizando el esquema de respuesta impulso finito (FIR).

La solución del filtro de Wiener basada en Respuesta de Impulso Finito (FIR) no utiliza datos históricos ni futuros como otros tipos de soluciones dadas por este mismo filtro, en compensación la fortaleza de FIR es contar con una buena muestra de la señal de entrada $w[n]$ que permita establecer la correlación con la señal deseada $s[n]$ e ir construyendo la señal estimada $x[n]$.

El filtro de Wiener FIR, representado en la figura 2.7 como $G(z)$ construye cada elemento actual n de la señal estimada $x[n]$ utilizando todos los elementos previos y actual $n-i$ de la señal con ruido w, junto a los coeficientes a_i que representan la respuesta impulso del filtro, como muestra la ecuación (2.21).

$$x[n] = \sum_{i=0}^{N} a_i w[n-i] \tag{2.21}$$

Cada elemento de la señal estimada es comparada con su similar de la señal deseada a fin de encontrar el error residual $e[n]$, que servirá para calcular los coeficientes para la siguiente estimación según el modelo de la ecuación (2.22).

$$a_i = argminE[e^2[n]] \tag{2.22}$$

En la ecuación (2.21) es posible calcular la señal $x[n]$ aplicando un filtro de Wiener de orden N con coeficientes $\{a_0, \ldots, a_N\}$.

El error residual $e[n]$ es calculado como $e[n] = x[n] - s[n]$, y es la *función objetivo* cuya salida se necesita minimizar convirtiéndose así en la *función de costo* del filtro de Wiener, y aunque la función de costo puede ser planteada de varias maneras ($J = |e|$, $J = e^2$, o $J = E[e^2]$), la obtención de un valor esperado E[.], considera procesos estocásticos o aleatorios, indicando que cada elemento de la señal está relacionado con una probabilidad de ocurrencia.

Para una señal cada elemento $x[n]$ y $s[n]$ es un valor complejo, pero para simplificar el entendimiento se considerará que estas cantidades son reales, así el error cuadrático medio (MSE) puede ser escrito como:

$$\begin{aligned} E[e^2[n]] &= E[(x[n] - s[n])^2] \\ &= E[x^2[n]] + E[s^2[n]] - 2E[x[n]s[n]] \\ &= E\left[\left(\sum_{i=0}^{N} a_i w[n-i]\right)^2\right] + E\left[s^2[n]\right] - 2E\left[\sum_{i=0}^{N} a_i w[n-i]s[n]\right] \end{aligned} \tag{2.23}$$

Después de reducir la función de costo en función de los coeficientes a_i, se puede calcular los valores de a_i derivando e igualando a cero.

$$\frac{d}{da_i} E[e^2[n]] = 0 \tag{2.24}$$

2.6.2. Restricciones de filtro de Wiener

Algunas restricciones son necesarias aplicar a fin de poder llevar a efecto todos los cálculos necesarios para lograr una estimación significativa; 1) el filtro es lineal para que el tratamiento matemático resulte más sencillo , 2) el filtro debe ser discreto a fin que pueda ser implantado en hardware digital, 3) el filtro debe tener una condición de causalidad implicando que debe tener un inicio a partir de cero y 4) que la respuesta impulso del filtro sea finita y que permite que el filtro sea inherentemente estable.

A pesar de las restricciones consideradas, tanto el cálculo de los coeficientes Wiener, así como las características propias del filtro para obtener una buena estimación, presentan una carga computacional muy grande, ya que aplicando en imágenes la transformada de Fourier bidimensional utilizando el algoritmo de fila-columna [10], requiere N*M transformadas, es decir, para una imagen de 256*256 píxeles se requieren 64k transformadas de Fourier de una dimensión de 256^2 puntos.

CAPÍTULO 3

Filtro BM3D

La eliminación de ruido en imágenes fijas utilizando un efectivo filtro en el dominio 3D transformado, fue propuesto por primera vez por Kostadin Dabov y tres colaboradores más [5], este enfoque ampliamente conocido como BM3D, sigue siendo utilizado por varios investigadores como punto de partida para nuevos trabajos en el campo del procesamiento de imágenes; en varios artículos también se lo conoce con el nombre de filtro colaborativo [14] [30].

Este capítulo estudia el tratamiento que da Mark Lebrum (creador del algoritmo BM3D) a los parches 3D obtenidos de la imagen a estimar, para comenzar a comprender este proceso y su código, como punto de partida es necesario conocer los componentes del algoritmo y la relación entre ellos mediante el diagrama UML que se muestra en el *Anexo A*.

El resto del capítulo está organizado de la siguiente manera: en la sección 3.1 se describe el algoritmo BM3D y cómo éste es ejecutado, detallando los parámetros necesarios que deben de ser utilizados para obtener la estimación de una imagen. En la sección 3.2 se describe los principales y más importantes modelos matemáticos del cual se vale el algoritmo para obtener sus resultados. La sección 3.3 explica cuáles son las restricciones consideradas en el algoritmo para que los procesos se ejecuten los más rápido y eficiente posible. La sección 3.4 explica el rango de valores que cada parámetro con que se ejecuta el algoritmo puede tener, haciendo énfasis en la mejor combinación para el objetivo de esta investigación. La sección 3.5 describe las interfaces llamadas durante la ejecución, y finalmente las secciones 3.6 y 3.7 detallan los procesos llevados a cabo en la primera y segunda etapa de estimación de la imagen.

3.1 ¿Qué es el algoritmo BM3D?

El algoritmo BM3D también es conocido como filtro colaborativo [16], esto se debe a que combina de una manera muy sincronizada el trabajo de dos filtros ampliamente utilizados en el estado del arte, estos son: 1) la transformada discreta de Wavelet y 2) el filtro de Wiener, estos filtros son aplicados a una imagen con ruido de forma secuencial, y han logrado un buen resultado eliminando el ruido y manteniendo los detalles de textura, bordes y contornos inclusive cuando los valores de ruido son altos ($\sigma > 40$) según lo considerado en el estado del arte, e.g. [16].

Más allá de la aplicación de dos de las mejores transformaciones indicadas previamente, el alto rendimiento del algoritmo BM3D radica en aumentar la escasa representación del ruido en el dominio espacial, seleccionando similares parches cuadrados (2D patches) de regiones cercanas dentro de la imagen para convertirlas en un solo parche cúbico (3D patch) que posteriormente son comparados entre sí para extraer patrones de similitud que ayudan a obtener el estimado de un parche de referencia.

Este proceso del selección y agrupamiento de parches se puede comprender observando la Figura 3.1, aquí se describe gráficamente como después de seleccionar un parche de referencia (cuadro con marco más grueso) se busca parches similares cercanos a el (cuadros con marco menos grueso) y se los apila para formar el arreglo 3D de parches.

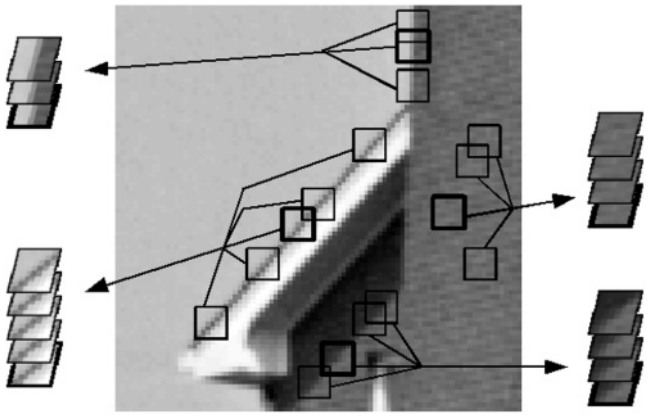


Figura 3.1: Proceso de selección y agrupamiento de parches para la estimación de una imagen con ruido, utilizando el algoritmo BM3D.

Pero el BM3D va más allá de simplemente selección y agrupar parches, el proceso completo de estimación requiere de varias subprocesos agrupados en dos etapas muy bien definidas como se indico al inicio de esta sección y se puede observar de forma

resumida en la Figura 3.2.

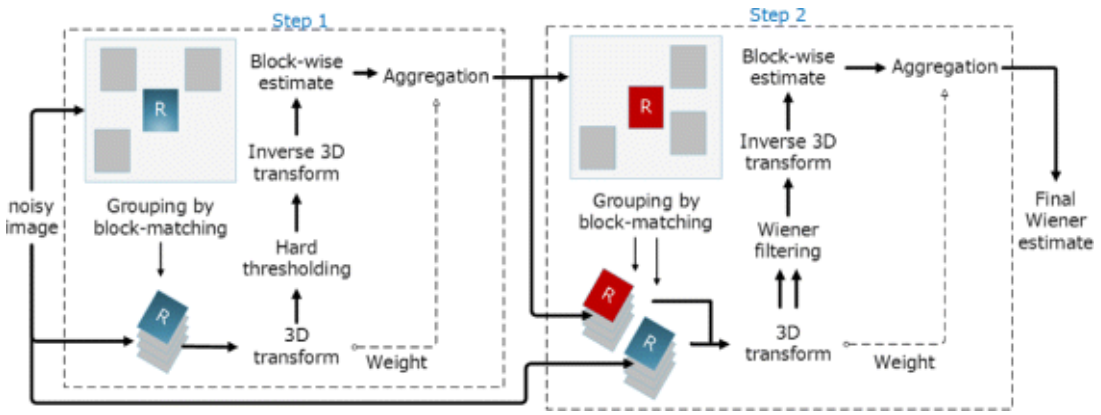


Figura 3.2: Etapas de estimación utilizando el modelo BM3D con el filtro Wiener [20].

Partiendo desde la imagen con ruido que en la figura anterior la designan con la letra R pero que en esta investigación utilizaremos la letra z, cada nombre descrito en la figura esta ligado a una función y cada función se rige por modelos matemáticos que son descritos en la siguiente sección.

3.2 Principales modelos matemáticos que utiliza el algoritmo BM3D

El primer modelo matemático considerado en el algoritmo BM3D, es el que permite obtener la imagen con ruido (z) (ecuación (3.1)) a partir de la imagen libre de ruido y y el modelo probabilístico de Gauss $\eta(\sigma)$; esta imagen con ruido será objeto de una serie de transformaciones hasta obtener la imagen estimada sin ruido $\hat{y}$.

$$z = y + \eta(\sigma) \tag{3.1}$$

donde el valor observado $z : X \to \mathbb{R}$, es el resultado de aplicar una señal de ruido η a la imagen fija sin ruido y; la señal de ruido η está en función de la desviación estandar σ.

Para lograr el efecto de ruido en la imagen original, se utiliza el modelo del ruido aditivo blanco Gausiano, el cual consiste en adicionar al canal de la imagen una señal que se distribuye probabilísticamente siguiendo un modelo de probabilidad normal o Gausiano esta señal de ruido centrado en $\mu = 0$ y con una varianza σ que varía según la necesidad.

$$P_G(z) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(z-\mu)^2}{2\sigma^2}} \tag{3.2}$$

En este modelo resalta la media (μ) y la desviación standar (σ) que van a determinar

la intensidad con que el ruido se va a presentar en la imagen.

La fase inicial donde el algoritmo obtiene la primera estimación de una imagen sin ruido $\hat{y}$, a partir de una imagen con ruido z, se utiliza el modelo matemático de la ecuación (3.3), este modelo describe la secuencia en que se aplica las transformaciones (DCT o Wavelet $bior1{,}5$) y filtro de tipo (ht) hasta obtener la estimación.

$$\widehat{Y}^{ht}_{S^{ht}_{x_R}} = \mathcal{T}^{ht^{-1}}_{3D}\left(\Upsilon\left(\mathcal{T}^{ht}_{3D}\left(Z_{S^{ht}_{x_R}}\right), \lambda_{thr3D}\sigma\sqrt{2log(N_1^2)}\right)\right) \tag{3.3}$$

En este modelo el conjunto de coordenadas $S^{ht}_{x_R}$ contiene las posiciones de los parches similares al parche ubicado en la posición de referencia x_R, todos los parches obtenidos pertenecen a una ventana al interior de la imagen con ruido z previamente transformada al dominio de la frecuencia y aplicado el respectivo filtrado (*hard thresholding*). Por limitaciones de recursos solo fueron 16 parches que se seleccionaron los cuáles se agruparon en un arreglo 3D representado por $\mathbb{Z}_{S^{ht}_{x_R}}$. Este arreglo 3D se lo trata como una sola imagen de $N \times N \times |S_{x_R}|$ píxeles al cual se lo somete a una transformación al dominio de la frecuencia $\mathcal{T}^{ht}_{3D}$ y se le aplica un filtro (*hard thresholding*) utilizando como parámetro del filtro el thresholding Universal $\sigma\sqrt{2log(N_1^2)}$, luego de lo cual se aplica la inversa de la transformada $\mathcal{T}^{ht^{-1}}_{3D}$, para finalmente obtener la primera estimación de la imagen $\widehat{Y}^{ht}_{S^{ht}_{x_R}}$.

Los parches Z_x de tamaño $N_1 \times N_1$ donde $N_1 = 8$ y cuya posición de su esquina superior izquierda dentro de la imagen z es x; en la ecuación (3.3) $S^{ht}_{x_R}$ representa una región de z alrededor de x_R, de ahí que los parches a procesar son señalados como $\mathbb{Z}_{S^{ht}_{x_R}}$.

Es importante enfatizar que para obtener las coordenadas $S^{ht}_{x_R}$ de los parches Z_x similares al parche de referencia Z_{x_R}, el criterio de selección entre todos los parches encontrados fue distancia de cada parche al de referencia el cual fue calculado aplicando primero un preprocesamiento que consistió en una transformación lineal a cada bloque 2D mostrado como $\mathcal{T}^{ht}_{2D}(Z_x)$ y luego un umbral Υ' de valor $\lambda_2 D\sigma$ [14, p. 28], esto permitió realizar una estimación base, después de lo cual se calcula la correlación (distancia cuadrática normalizada) entre cada bloque con el de referencia $d(Z_{x_R}, Z_x)$.

$$d(Z_{x_R}, Z_x) = \frac{||\Upsilon'\left(\tau^{ht}_{2D}(Z_{x_R}), \lambda_{thr2D}\sigma\sqrt{2log(N_1^2)}\right) - \Upsilon'\left(\tau^{ht}_{2D}(Z_x), \lambda_{thr2D}\sigma\sqrt{2log(N_1^2)}\right)||_2^2}{N_1^{ht^2}} \tag{3.4}$$

Solo son considerados los parches que tienen una distancia $d \leq \tau^{ht}_{match}$ de x_R, donde

τ_{match}^{ht} es la máxima distancia para el cual dos bloques son considerados similares [6, pag.5].

$$S_{x_R} = \{x \in \mathbb{X} | d(Z_{x_R}, Z_x) < \tau_{match}\} \tag{3.5}$$

En una segunda etapa del proceso *Block Matching* las lineas de código se vuelven a repetir solamente con una variación que consiste en que en vez de utilizar el filtro de Hard Threshold (HT) se utiliza el filtro de Wiener (WIE) [6, pag.6] según la ecuación 3.6 siguiente:

$$\widehat{Y}_{S_{x_R}^{wie}}^{wie} = \tau_{3D}^{wie^{-1}} \left(\mathbf{W}_{S_{x_R}^{wie}} \left(\tau_{3D}^{wie} \left(Z_{S_{x_R}^{wie}} \right) \right) \right) \tag{3.6}$$

3.3 Restricciones del algoritmo para reducir la complejidad

Por las limitaciones de recursos computacionales fue necesario establecer restricciones tanto en los valores de los parámetros como en algunas de las funcionalidades del algoritmo BM3D, esas restricciones fueron calculadas con la finalidad de obtener un óptimo resultado en un tiempo y con una utilización de memoria y procesador razonable.

1. La cantidad de parches similares a un parche de referencia S_{x_R}, dentro de una ventana $N \times N$ puede ser tan grande como $(N-1) \times (N-1)$, lo que demandaría gran cantidad de tiempo y uso de procesador como memoria, es por eso que una de las primeras restricciones a considerar es establecer el número máximo de bloques similares a los bloques de referencia que se van a tomar en cuenta, este valor lo guarda en la variable designada como N_2 [6], esta variable daría el tamaño al conjunto $S_{x_R \in X}$.

2. Los bloques similares se los busca solo en una ventana local $N \times N$ menor al tamaño de la imagen en cuyo centro estará el bloque de referencia, y no se toma en cuenta toda la imagen.

3. También se limita el número de bloques de referencia al procesar determinando el número de pasos a desplazarse para conseguir el siguiente bloque de referencia esto se lo establece con la variable $N_{step} > 1$.

A pesar de estas restricciones, el procesamiento de imágenes utilizando el algoritmo BM3D exige la ejecución de gran cantidad de operaciones matemáticas de forma interactiva e iterativa que demandan el uso del procesador permanentemente y por largos periodos de tiempo; por lo que es necesario aprovechar al máximo las actualizaciones

que se hacen a los nuevos procesadores para que puedan soportar este nivel de procesamiento. Una de estas mejoras ha sido la de compartir el proceso entre todos los núcleos disponibles en el equipo de trabajo, distribuyendo así la carga computacional de los avanzados algoritmos, BM3D utiliza la interfaz de programación aplicada (API) llamada openMP para aprovechar el multiproceso y la memoria compartida en múltiples plataformas, este API permite añadir concurrencia a los programas escritos en C, C++, sobre la base del modelo de ejecución fork-join como se muestra en la figura 3.3.

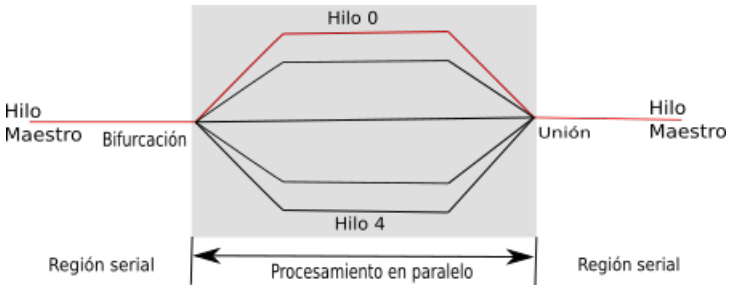


Figura 3.3: El procesamiento en paralelo es lo que caracteriza al API openMP

3.4 Ejecución parámetros y valores de parámetros utilizados en las interfaces del proceso de estimación BM3D

El algoritmo BM3D fue implementado en C++, y el punto de inicio para poder ejecutar todos los procesos hasta llegar a su culminación consiste en ejecutar un programa o función principal (main function), esta función principal requiere de un conjunto de parámetros que deben ser pasados vía linea de comandos utilizando el siguiente formato (BM3Ddnoising <Parametro1 Parametro2 . . . Parametro14>). Tanto el nombre del programa como sus argumentos constituye la interface de ejecución. Estos parámetros en un número de 14 deben ser brindados cumpliendo con ciertas especificaciones descritas en la tabla 3.1.

Tabla 3.1: *Parámetros iniciales para ejecutar el algoritmo BM3D*

Orden	Variable	Descripción
1	img	Cadena de caracteres que contiene el nombre del archivo de la imagen a procesar, la imagen debe estar en formato **png** y en el espacio de color RGB
2	sigma	Valor de tipo numérico que representa la intensidad del ruido que se aplica a la imagen original, se utiliza el ruido aditivo blanco Gausiano.
3	*img_noisy*	Cadena de caracteres que contiene el nombre con que se va a guardar la imagen con ruido.
4	*img_basic*	Cadena de caracteres que contiene el nombre con que se guarda el resultado de la primera estimación, esta es una estimación básica antes de la estimación final
5	*img_denoised*	Cadena de caracteres que contiene el nombre del archivo con que se guarda estimación final.
6	*img_diff*	Cadena de caracteres que contiene el nombre del archivo con que se guarda la imagen diferencia entre la imagen con ruido y la imagen final estimada.
7	*img_bias*	Cadena de caracteres que contiene el nombre del archivo con que se guarda la imagen estimada de la imagen sin ruido.
8	img_bias_diff	Cadena de caracteres que contiene el nombre del archivo con que se guarda la diferencia de la imagen sin ruido con su imagen estimada.
9	*compute_bias*	Valor de tipo lógico que le indica al algoritmo si es necesario calcular o no la estimación de la imagen sin ruido.
10	*tau_2D_hard*	Tipo de transformación que se va a aplicar DCT o BIOR
11	*useSD_1*	Valor de tipo lógico que le indica al algoritmo si es necesario utilizar la desviación standar en la primera estimación.
12	*tau_2D_wien*	Cadena de caracteres que contiene el nombre del tipo de transformación que se va a aplicar en la segunda face de la estimación (DCT o BIOR).
13	*useSD_2*	Valor de tipo lógico que le indica al algoritmo si es necesario utilizar la desviación standar para la segunda estimación.
14	*color_space*	Cadena de caracteres que contiene el nombre del espacio de color a utilizar.

Después que el algoritmo BM3D recibe los parámetros iniciales de la Tabla 3.1, estos son repartidos al resto de módulos que componen el algoritmo, así el módulo (bm3d_1st_step) que realiza la primera estimación de la imagen, necesita los parámetros indicados en la Tabla 3.2.

Tabla 3.2: *Variables que contienen los parámetros que utilizan los procesos del algoritmo BM3D para estimar la imagen.*

variable	valor	Descripción
σ	$0 < \sigma < 100$	Valor entero que indica la intensidad del ruido.
img_noisy	vector	Vector de una dimensión que contiene todos los píxeles de la imagen con ruido.
img_basic	vector	Vector de una dimensión que contiene o contendrá todos los píxeles de la primera estimación de la imagen.
width	> 0	Valor entero que contiene el ancho de la imagen a procesar.
height	> 0	Valor entero que contiene el alto de la imagen a procesar.
chnls	> 0	Valor entero que indica el número de canales que contiene la imagen, si la imagen es RGB chnls=3.
nHard,nWien,nHW	16	Valor entero que indica el tamaño de la ventana de búsqueda.
tau_2D_hard,tau_2D_wien,tau_2D	DCT/BIOR	Cadena de caracteres que indica el tipo de transformada lineal aplicada a los parches en 2D.
kHard,kWien,kHW	8	Valor entero que indica el tamaño del parche (sı tau_2D_hard="BIOR").
	8	Indica el tamaño del parche (si tau_2D_hard="DCT" y $\sigma < 40$).
	12	Indica el tamaño del parche (si tau_2D_hard="DCT" y $\sigma \geq 40$).
pHard,pWien,pHW	3	Valor entero que indica los espacios (pasos) que se recorre entre una ventada y otra.
NHard,NWien,NHW	16	Valor entero que indica la cantidad máxima de parches similares al parche de referencia escogidos para optimizar la carga computacional.
lambdaHard3D	2.7	Un valor decimal que indica el umbral o límite para eliminar coeficientes (Hard Thresholding).

3.5 Interfaces utilizadas en la estimación

En la ejecución del algoritmo BM3D, cada módulo hace llamadas a otros módulos a través de interfaces que requieren ciertos parámetros, el altoritmo escrito en C++ ejecuta inicialmente la función *main()* la cual divide el proceso en dos etapas; para la primera etapa utiliza la interface que llama como (bm3d_1st_step()) y cuyas llamadas a interfaces se muestra en la tabla 3.3.

Tabla 3.3: *Interfaces a las funciones que utiliza el algoritmo BM3D para la primera etapa del procesamiento de la imagen.*

Interface	Descripción
load_image(. . .)	Interface que llama al procedimiento para cargar en memoria la imagen, el cual debe tener un formato PNG, esta imagen es transformada en 3 arreglos del tamaño de la imagen y que representan los 3 canales que conforman el espacio de colores RGB.
add_nise(. . .)	Interface que llama al procedimiento que crea la imagen con ruido a partir de la imagen original libre de ruido, valiéndose del modelo probabilístico de la distribución de Gauss.
run_bm3d(. . .)	Interface que llama a los procedimientos que utilizan los modelos matemáticos necesarios para obtener la estimación de la imagen en dos etapas.
compute_psnr(. . .)	Interface que llama al procedimiento que calcula el índice de la Relación Señal a Ruido de Pico (en ingles: PSNR) basado en el cálculo de error cuadrático medio y la máxima energía posible de alcanzar por cada píxel.
save_image(. . .)	Interface que llama al procedimiento que permite guardar los resultados obtenidos como una imagen en formato RGB.

Cada una de estas interfaces utilizan procedimientos que también hacen llamados a otras interfaces, las cuáles en conjunto ejecutan el proceso de eliminación de ruido, así la función run_bm3d que es la que estima la imagen utiliza las interfaces que se muestran en la tabla 3.4.

La segunda etapa de eliminación de ruido y que permite obtener la estimación final del algoritmo BM3D, es ejecutada llamando al procedimiento (bm3d_2da_step()) la cual utiliza modelos e interfaces parecidas a la primera etapa con ligeros cambios.

Tabla 3.4: *Interfaces que llaman a las funciones que realizan la primera estimación BM3D.*

Interface	Descripción
estimate_sigma(. . .)	Estima el sigma en otros canales debido a que se cambió de RGB a YUV.
ind_initialize (. . .)	Interface que llama al procedimiento para obtener las posiciones de cada parche a procesar.
preProcess (. . .)	Interface que llama al procedimiento que calcula los coeficientes para aplicar el algoritmo Kaiser_window y obtener longitudes de ondas apareadas procesables por la librería FFTW.
bior15_coef(. . .)	Interface que llama al procedimiento que prepara un grupo de filtros necesarios para la transformación de la imagen al dominio de la frecuencia utilizando el algoritmo bior1.5.
precompute_BM(. . .)	Interface que llama al procedimiento que selecciona los parches similares utilizando el error cuadrático y un threhold definido por el usuario.
dct_2d_process (. . .)	Interface que llama al procedimiento para aplicar la transformada lineal de Fourier, utilizando la librería FFTW (*Fast Fourier Transform in the West*).
ht_filtering_hadamard (. . .)	Interface que llama al procedimiento para aplicar una transformada lineal basada en Hadamard al arreglo 3D y luego elimina el ruido utilizando el hard thresholding al arreglo 3D.
sd_weighting (. . .)	Interface que obtiene los pesos ponderados en 3D utilizando la desviación estandar.

3.6 Primera fase de estimación utilizando el filtro DCT

Utilizando una terminología estadística se puede considerar que los individuos de la población a procesar están conformados por los píxeles, aquí el atributo de interés a medir es la intensidad del color Z_x de cada píxel. Una muestra de esta población que es este caso se lo denomina como un parche $\mathbf{Z}_x$, es determinada con aquellos píxeles cuyo atributo de interés sean similares a un parche (o muestra) de referencia $\mathbf{Z}_{x_x}$. El proceso comienza con crear una matriz de diferencias **DIFF[]** con dimensiones igual al de la imagen, donde en vez de guardar la intensidad de color se guarda el cuadrado de la diferencia entre las intensidades de color del píxel que se encuentra dentro del parche de referencia con un píxel en la misma posición vertical pero horizontalmente en un rango de (-nHW,+nHW) del píxel de referencia $d = (Z_x - Z_{x_R})^2$, así cuando estas ventanas coinciden $d = (Z_{x_R} - Z_{x_R})^2 = 0$.

La sumatoria de la diferencia entre el error cuadrático del píxel de referencia con el píxel de la derecha y el error cuadrático del píxel de referencia con el píxel de la izquierda, de todos los píxeles del parche es lo que dará el criterio para poder compararlo con el threshold y poder seleccionar los **NHW** necesarios para crear el arreglo 3D, ecuación (3.7).

$$y = \sum_{i=0}^{x} [(Z_{x_{left}} - Z^{i}_{x_R}) - (Z^{i}_{x_R} - Z_{x_{right}})] \tag{3.7}$$

Después que el proceso *block matching* permitió ubicar los parches a procesar, es necesario realizar la primera transformación lineal a cada parche, para esto es necesario verificar que modelo utilizar en función de la selección que hizo el usuario al ejecutar el programa principal, pasado en los argumento 10 y 12 (argv[10],argv[12]), la cadena de caracteres "dct" si desea que se utilice el DCT (*Transformada Discreta del Coseno*) o la cadena "bior" si por el contrario desea utilizar la transformada Biorthogonal (Bior1.5) que pertenece a la familia DWT (*Transformada Discreta de Wavelet*), cualquiera de estas dos transformadas permite llevar a la imagen del dominio espacial al dominio de la frecuencia para así poder identificar las frecuencias más altas que caracterizan a la señal del ruido AWGN y poderlo eliminar aplicando un umbral (*hard Thresholding*) el cual consiste en eliminar los coeficientes de mayor valor. La transformada Bior1.5 según el estado del arte es más recomendable utilizar porque su ejecución demanda mucho menos recursos de hardware en comparación con las transformadas de Fourier como es el DCT [25].

Para utilizar la Transformada Discreta del Coseno (DCT), el algoritmo BM3D utiliza la librería creada en C llamada FFTW (*Fast Fourier Transform in the West*) la cual tiene la capacidad de calcular la Transformada Discreta de Fourier en una o más dimensiones con una complejidad de $\mathcal{O}(N \log N)$ hasta la edición de este trabajo la última versión era la 3.8.8 y estaba disponible bajo licencia *General Public Licence* (GLP).

Una de las fuertes razones para utilizar la librería FFTW, es porque ha sido optimizado para realizar el proceso de transformación en paralelo, paralelizando el código para plataformas que utilizan máquinas con SMP (Symetric Multi-Processing), la librería además soporta variedad de interfaces (POSI y openMP) para trabajar con diferentes tipos de hilos y utilizando la capacidad SIMD (*Simple instructions, multiple data*).

Básicamente, el problema del cálculo de DFT es obtener la secuencia X(n) de

longitud N según la fórmula siguiente:

$$X(k) = \sum_{n=0}^{N-1} x(n) W^{nk} \tag{3.8}$$

Donde $N = 2^a$ siendo $a = 1, 2, 3, \ldots$.

El factor $W = e^{\frac{-j2\pi}{N}} = cos\left(\frac{2\pi}{N}\right) - j\ sin\left(\frac{2\pi}{N}\right)$ denominado constante de rotación [15].

Por consiguiente:

$$W^n = cos\left(\frac{2\pi}{N}n\right) - j\ sin\left(\frac{2\pi}{N}n\right) \tag{3.9}$$

El *Fast Fourier Transform* (FFT) optimiza el algoritmo DFT, reduciendo el número de operaciones, esto lo logra separando la señal x(n) en dos secuencia para n's pares e impares y así aplicarle a cada subsecuencia un DFT por separado.

$$X(k) = \sum_{n=0}^{\frac{N}{2}-1} x(n) W^{nk} + \sum_{n=\frac{N}{2}}^{N-1} x(n) W^{nk} \tag{3.10}$$

Se puede observar que para el funcionamiento del FFTW es necesario contar con un número de muestras par por lo que para lograr obertener una muestras con estas características, se realiza un preprocesamiento donde utilizando la técnica de windoming a través del modelo de Kaiser window (ecuación (3.11)) se obtiene un coeficiente para obtener longitudes de ondas apareadas.

$$w_k = \begin{cases} \frac{I_0(\pi\alpha\sqrt{1-(\frac{2k}{n}-1)^2})}{I_0(\pi\alpha)} & si\ 0 \leq k \geq n \\ 0 & resto \end{cases} \tag{3.11}$$

Donde I_0 representa un modificador basado en la función de Bessel de orden cero, n es el tiempo de duración de la ventana, y α es un número real no negativo que determina la forma de la ventana como se muestra en la figura 3.4.

En la utilización de una transformada de la familia FFT para obtener los componentes de frecuencia de la imagen, se asume que se ha capturado un conjunto finito de datos provienen de un espectro continuo de una señal periódica y que el tiempo tomado ha logrado capturar un número entero de periodos, es decir que el inicio y el final de las muestras están conectados.

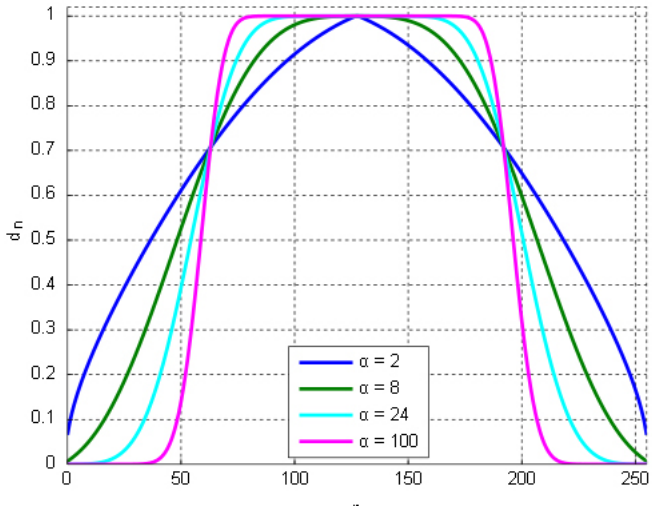


Figura 3.4: Ventana de Kaiser, problemas de windowing

La otra alternativa de transformación que soporta el algoritmo BM3D, es la Tranformada Biortogonal de la familia de Wavelet, esta utiliza el algoritmo Bior1.5 el cual necesita primeramente calcular los 4 filtros siguientes.

filtro de frecuencias baja para la transformación inicial

$$lpd = \frac{\sqrt{2}}{256}[3; -3; -22; 23; 128; 22; -22; -3; 3]$$

filtro para frecuencias altas para el inicio de la transformación

$$hpd = \frac{\sqrt{2}}{256}[0; 0; 0; 0; -1; 1; 0; 0; 0; 0]$$

filtro de frecuencias baja para la transformación inicial

$$lpr = \frac{\sqrt{2}}{2}[0; 0; 0; 0; 1; 1; 0; 0; 0; 0]$$

filtro para frecuencias altas para el inicio de la transformación

$$hpr = \frac{\sqrt{2}}{256}[3; 3; -22; -22; 128; -128; 22; -3; 3]$$

Después de obtener una primera estimación que lleva cada parche al dominio de la frecuencia utilizando las transformaciones anteriores, es necesario fusionar todos los parches similares en un arreglo 3D, este es un proceso sencillo que solo necesita un par de lazos que recorra una tabla de índices para tomar las ubicaciones de cada parche en sus 3 canales y ubicarlos en una tabla que guardará los índices agrupados por su similaridad.

A cada arreglo, se le puede aplicar el filtro Hard Threholding pero para ello primero se aplica la transformada de Hadamard, la cual es una variación del *Fast Fourier Transform* (FFT) utilizado por su mayor rapidez en vista de que se aplica a una muestra mucho mayor.

Se aplica la inversa de la transformada dependiendo si se utilizó la transformada discreta del coseno “DTC", o la transformada de Wavelet con la señal madre bi-ortogonal “bior"

Finalmente, en esta primera fase se reconstruye la imagen liberada de una parte del ruido, se la realiza, reconstruyendo cada píxel, utilizando los pesos ponderados calculados con parámetros obtenidos en el dominio de la frecuencia.

3.7 Segunda fase de estimación utilizando el filtro Wiener

En esta segunda fase del algoritmo BM3D, el proceso es muy similar al realizado en la primera fase, salvo por dos cambios. Primero se utiliza la estimación obtenida en la primera fase como la salida deseada para la nueva estimación de la imagen libre de ruido, y segundo se elimina el ruido utilizando el filtro de Wiener en lugar de aplicar el Hard thresholding.

Los procesos que se realizan para lograr la eliminación del ruido son:

1. Indexación de las ventanas de búsqueda a lo largo de toda la imagen.

   Se inicia el proceso estableciendo y registrando a lo ancho ($width$) y alto ($height$) de la imagen, las coordenadas (x, y) de todas las ventanas de tamaño $nWien \times nWien$ que se van a procesar individualmente según el proceso *block matching*; las ventanas están traslapadas, y sus esquinas están separadas 3 píxeles de la siguiente.

2. Se crear el arreglo 3D.

   Luego de obtener de aplicar la transformada se construye el arreglo 3D.

3. Se aplica el filtro Hadamar en el arreglo 3D.

4. Se aplica el filtro de Wiener

   Se aplica el filtro de Wiener, para lo cual primero se utiliza una variación de la transformada de Fourier llamado el filtro de Hadamard, para llevar los parches al dominio de la frecuencia; a esta imagen en el dominio de la frecuencia se le

aplica el algoritmo del filtro de Wiener; posteriormente se aplica la inversa de la transformada de Hadamard.

5. Se aplica los pesos ponderados.

   Los pesos ponderados se calculan de la misma forma que la primera etapa.

6. Reconstrucción de la imagen estimada.

Para la reconstrucción final de la imagen estimada en esta segunda etapa, los cálculos son similares a los utilizados en la primera etapa, la única diferencia es la ponderación se aplica a los píxeles estimados y no a los originales píxeles con ruido [20].

$$\forall Q \in P(P), \forall x \in Q, \begin{cases} v(x) = v(x) + w_P^{hard} u_{Q,P}^{hard}(x) \\ \delta(x) = \delta(x) + w_P^{hard} \end{cases} \tag{3.12}$$

donde se utilizan todos los píxeles x para estimar un numerador (v) y denominador (δ) que servirán para calcular el peso ponderado de cada uno de los píxeles de la imagen, $u_{Q,P}^{hard}(x)$ corresponde al estimado del valor del píxel x perteneciente al parche Q obtenido en el proceso del filtro colaborativo del parche de referencia P, w_P^{hard} es el resultado después de eliminar los coeficientes determinados por el hardthresh aplicado a los N coeficiente dentro del parche P.

$$w_P^{hard} = \begin{cases} (N_P^{hard})^{-1} & if\ N_P^{hard} \leq 1 \\ 1 & otherwise \end{cases} \tag{3.13}$$

CAPÍTULO 4

Redes neuronales artificiales – RNA

Este capítulo describe la arquitectura y funcionamiento de la Red Neuronal Artificial (RNA) para comprender de forma concreta las Redes Neuronales Convolucionales utilizadas en esta investigación. En su parte funcional, la red neuronal artificial esta creada para resolver principalmente dos problemas de la vida real que son: 1) ***predicción***, y 2) ***clasificación*** de información; la predicción y la clasificación de información son funciones que realizan un tratamiento especial de la información las cual consiste en buscar y seleccionar patrones de información histórica sobre un problema, para aplicarlos al procesamiento de información presente o futura sobre el problema.

De antemano se empieza indicando que existen varias topologías de redes neuronales artificiales entre las más populares están: *Multilayer Perceptron* (MLP), *Probabilistic Neural Network* (PNN), *Time delay Neural Network* (TDNN) y *Convolucional* (RNC); esta última es ampliamente utilizada en tareas como reconocimiento en imágenes, sistemas de reconocimiento de voz y procesamiento de lenguaje natural, por lo que es la herramienta ideal seleccionada para utilizarla en esta investigación.

En lo que resta de este capítulo se analiza el complejo proceso de búsqueda y selección de los patrones adecuados que permitan llegar a la más precisa solución del problema. Para este análisis se explicará los fundamentos matemáticas de dos procesos básicos que son: 1) ***regresión lineal*** y 2) ***regresión logística***; al final del capítulo se describe las características y funcionamiento de una variante de red neuronal artificial aplicable al procesamiento de imágenes llamada Red Neuronal Convolucional (RNC) con una topología llamada Auto-encoder, la cual se ajusta a la necesidad de esta investigación para cumplir con el objetivo propuesto en la sección 1.1.

El resto del capítulo está organizado de la siguiente manera: la sección 4.1 explica el funcionamiento básico de una red neuronal, esto es la regresión lineal y la regresión logística, la sección 4.2 detalla los modelos y estructura de la RNA, la sección 4.3

explica cómo se calcula el tamaño de una RNA, la sección 4.4 describe el modelo de la red neuronal convolucional (RNC) y su funcionamiento a través del framework Tiny-dnn escrito en C++, finalmente en esta misma sección se analizan dos parámetros que influyen en el tiempo y precisión con que se dan los resultados, estos son epoch y batch.

4.1 Bases fundamentales para el funcionamiento de una RNA

Específicamente esta investigación aprovecha la capacidad que tiene la red neuronal artificial de identificar y diferenciar los patrones que se encuentran en una matriz, esto lo hace ponderando cada elemento o grupo de elementos, para obtener los valores ponderados o más comúnmente conocidos como pesos; estos pesos se obtienen después de un largo proceso de interacción bidireccional entre unidades básicas de cálculo, que permiten transformar una entrada desconocida en una salida esperada.

Estos patrones basados en valores y ubicación dentro de la matriz son encontrados gracias al análisis de una gran cantidad de datos suministrados en un proceso inicial de entrenamiento y validación que le dan a la red la capacidad de inferir y clasificar sobre datos completamente nuevos. El procedimiento que permite este aprendizaje profundo se llama Gradiente Descendiente y es utilizado en todos los modelos de redes neuronales artificiales que el estado del arte a creado.

Los fundamentos conceptuales que han permitido el desarrollo de las redes neuronales artificiales, nacen de la llamada SIMPLE REGRESIÓN LINEAL que es explicado en la siguiente sección.

4.1.1. Regresión lineal

Este es un método de tipo estadístico que permite resumir y estudiar la relación entre dos variables continuas; una de las variables siempre denotada por la letra $\mathbf{x}$, considerada como la variable predictora, explicativa o matemáticamente la variable independiente; la otra variable denotada por la letra $\mathbf{y}$, se la conoce como la respuesta, salida, o matemáticamente hablando variable dependiente.

En esta investigación se va a considerar los nombres de PREDICTORA para la variable $\mathbf{x}$ y RESPUESTA para la variable $\mathbf{y}$, por ser estos los que más se ajustan a la aplicación del procesamiento de imágenes. En resumen, la regresión lineal consiste en obtener el segmento de línea que mejor se ajuste a la intersección en el plano cartesiano de cada par (x,y), y cuando se dice que se ajuste a la intersección significa que la distancia de

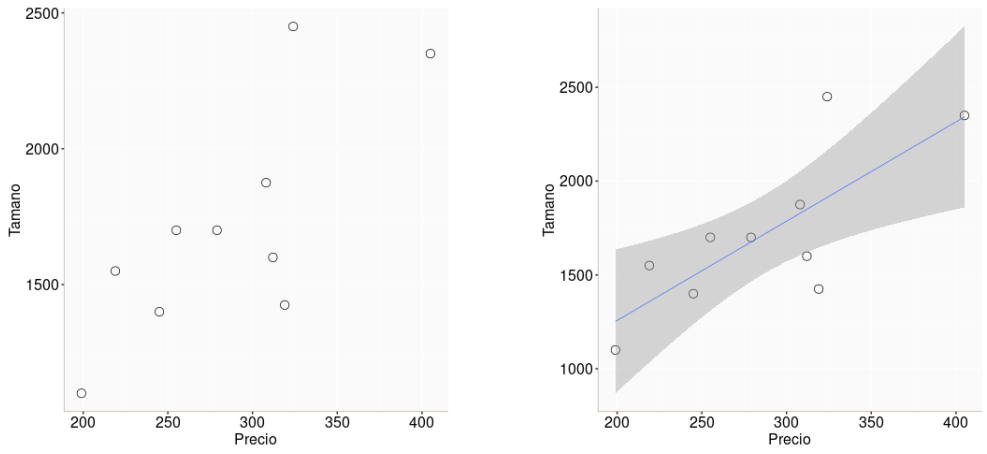
cada punto a la recta debe ser la menor posible de entre todas las posibles rectas que se puedan dibujar.

Un ejemplo clásico muy utilizado en la literatura para mostrar la funcionalidad del método simple regresión lineal es el caso de predecir el precio de una vivienda en función del tamaño que tenga esta (tabla 4.1).

Tabla 4.1: *Datos que muestran los precios a los que se ha vendido una vivienda de un tamaño determinado.*

#	**Precio** en $1000 (x)	**Tamaño** en $pies^2$ (y)
1	245	1400
2	312	1600
3	279	1700
4	308	1875
5	199	1100
6	219	1550
7	405	2350
8	324	2450
9	319	1425
10	255	1700

Para procesar estos datos utilizando el método de regresión lineal, se debe comprobar que todos los datos tengan una relación lineal, lo cual requiere de aplicar un método estadístico como el *t-estudiante* u otro, en este ejemplo en particular, los datos si muestras una correlación positiva que se aprecia a simple vista en la figura 4.1-(a).



(a) (b)

Figura 4.1: (a) Representación de los valores (precio, tamaño) en el plano cartesiano; (b) Representación del modelo lineal.

En este simple ejemplo muy común de una simple regresión lineal, se trabaja con una muestra de 10 datos, el tamaño es la variable PREDICTORA y el precio es la variable RESPUESTA, que de forma simplificada y según la notación matemática se la puede expresar en un par ordenado (x,y) respectivamente.

Como se puede observar en la figura 4.1-(b), la simple regresión lineal trata de resumir los datos ajustándolo a un patrón (una recta), tratando de que el costo (distancia acumulada entre el punto real y su respectiva proyección sobre la recta) sea el menor posible.

$$\hat{y} = b_0 + b_1 * x \tag{4.1}$$

Después de encontrar la recta, cada punto se puede encontrar solo con saber la distancia del punto estimado con el punto real.

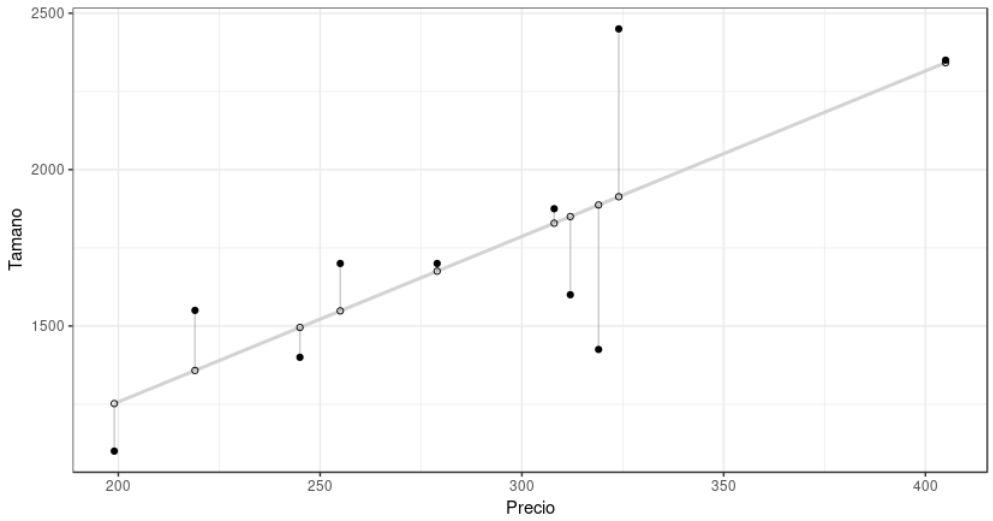


Figura 4.2: Regresión lineal: error de cada punto.

$$\hat{y} = b_0 + b_1 * x + \epsilon \tag{4.2}$$

Aunque la solución de esta ecuación para una simple regresión lineal consiste en hallar los coeficientes b_0 y b_1 la cual se puede hacer con una simple operación de matrices de la forma:

$$B = (X^T X)^{-1} X^T Y \tag{4.3}$$

Esta ecuación llamada Ecuación Normal y calculable computacionalmente cuando la muestra de los puntos es bastante limitada; no puede ser aplicada cuando se cuenta

con una muestra muy grande o cuando se pretende calcular una regresión lineal con múltiples variables predictoras, puesto que su complejidad es $\mathcal{O}(n^3)$ donde n es la cantidad de muestras, por lo que es necesario recurrir a otros métodos.

Una de estas metodologías es el llamado Gradiente descendente, el cual consiste en analizar el error entre la variable estimada $\widehat{y}$ y la variable real y como se muestra en la ecuación 4.4.

$$error = \sum(y - \hat{y})^2 \tag{4.4}$$

Si en la ecuación 4.4 se remplaza $\hat{y}$ por su respectiva ecuación de la recta 4.2 que se quiere encontrar, se obtendrá la ecuación 4.5:

$$error = \sum_{i=1}^{n}(y_i - (b_0 + b_1 * x_i)) \tag{4.5}$$

Donde i=1,2,3,. . . ,n, representa la cantidad de valores que toma la variable predictora x; en el ejemplo particular tomado en esta sección n=10.

En la ecuación (4.5) se observa dos variables desconocidas que son b_0 y b_1, las cuales pueden calcularse al resolver un sistema de 2 ecuaciones, estas ecuaciones pueden ser determinadas por la condición de que la diferencias $(y - \hat{y})$ debe tender a 0.

$$\frac{\partial}{\partial y} error = 0 \tag{4.6}$$

Puesto que el error depende de las dos variable b_0 y b_1, se puede utilizar las derivadas parciales, ecuación 4.7 y ecuación 4.8.

$$\frac{\partial error}{\partial b} = 0 \tag{4.7}$$

$$\frac{\partial error}{\partial w} = 0 \tag{4.8}$$

4.1.2. Regresión logística

Otra fuente de inspiración para las redes artificiales neuronales, es la regresión logística, que es también un método estadístico igual que la regresión lineal explicada en el apartado anterior, y que de igual manera permite encontrar la relación entre dos variables, una variable predictora y una variable de respuesta. En este caso la diferencia está en que la variable de respuesta es una variable dicotómica (lógica) o también en el estado del arte se la llama muy frecuentemente variable nominal, y a la variable

predictora se la llama variable de medida. La meta es determinar si la probabilidad de conseguir un valor en particular de una variable nominal está asociada con la variable de medida.

Un ejemplo bastante significativo esta dado por [28], el cual midió el tamaño de los granos de arena en 28 playas de Japón y observó la presencia o ausencia de arañas lobo marineras *Lycosa ishikariana* en cada playa, obteniendo la tabla 4.2 siguiente:

Tabla 4.2: *Simple regresión logística.*

#	Tamaño de Grano(mm) X	Araña Y
1	0.245	ausencia
2	0.247	ausencia
3	0.285	presencia
4	0.299	presencia
5	0.327	presencia
6	0.347	presencia
7	0.356	ausencia
8	0.36	presencia
9	0.363	ausencia
10	0.364	presencia
11	0.398	ausencia
12	0.4	presencia
13	0.409	ausencia
14	0.421	presencia
15	0.432	ausencia
16	0.473	presencia
17	0.509	presencia
18	0.529	presencia
19	0.561	ausencia
20	0.569	ausencia
21	0.594	presencia
22	0.638	presencia
23	0.656	presencia
24	0.816	presencia
25	0.853	presencia
26	0.938	presencia
27	1.036	presencia
28	1.045	presencia

Estadísticamente se calcula la probabilidad de que los cangrejos pertenezcan a la primera clase (de las dos clases presencia/ausencia) dependiendo del tamaño que tengan los granos de arena en la playa:

$$P(cangrejo = ausencia|Tamaño\ de\ granos\ de\ arena)$$

Una forma estandar de expresarlo es según la ecuación 4.9.

$$P(X) = P(Y = 1|X)g \tag{4.9}$$

La regresión logística lo que hace es llevar los datos a una dimensión donde estos pueden clasificarse en dos categorías; la forma matemática de hacer esto es utilizando un tipo de función llamada FUNCIÓN LOGÍSTICA.

$$y = \frac{e^{b_0+b_1x}}{1+e^{b_0+b_1x}} \tag{4.10}$$

4.2 Modelo de una red neuronal artificial

Las redes neuronales artificiales, computacionalmente están implementadas como una estructura de grafos acíclicos que almacenan valores (en sus conexiones) y procedimientos (en sus nodos); los procedimientos almacenados son de dos tipos, uno de tipo **regresión lineal** y el otro de tipo **gradiente descendente**. La regresión lineal crea un modelo en base a un conjunto de datos de entrenamiento que representan la experiencia E en una tarea T determinada [12], este modelo determina la relación entre los datos de entrada y la salida (o la solución) a fin de poder predecir soluciones a datos de entradas futuros pero perteneciente a la misma tarea T.

En su primera aproximación al modelo de una RNA comenzaría como una regresión lineal que matemáticamente se expresa según la ecuación (4.11).

$$\hat{y} = w^T x \tag{4.11}$$

Donde $w \in \mathbb{R}^n$ es un vector que contiene los **parámetros** que fueron seleccionados y asignados, producto del entrenamiento o experiencia E, y $\hat{y}$ representa una estimación de la salida o solución a la tarea T.

El modelo de la ecuación (4.11) simplemente lo que hace es transformar una entrada x en una salida $\hat{y}$ utilizando un conjunto de parámetros, que comunmente se les llama peso, esta sola ecuación está muy lejos de poder resolver problemas reales y tan complejos como los que actualmente resuelven herramientes de aprendizaje profundo (deep learning). Esta simple ecuación es ejecutada por la unidad de las redes neuronales al cual se les llama **Perceptrón** y gráficamente es representada como indica la figura 4.3.

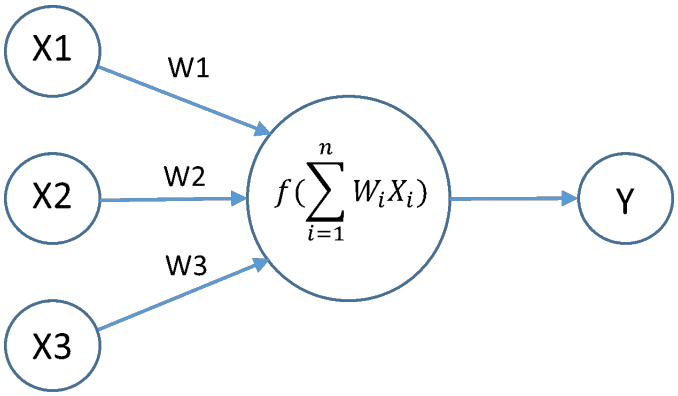


Figura 4.3: Perceptrón como componente unitario de las redes neuronales

La red neuronal artificial es la muestra más fiable de que la unión hace la fuerza, puesto que simples operación de multiplicación y suma repetida muchas veces es lo que le da a las redes neuronales artificiales la capacidad de resolver problemas complejos; en la figura 4.4 se puede observar la estructura de una RNA, aunque aún bastante simple comparada con las utilizadas en la solución de problemas reales.

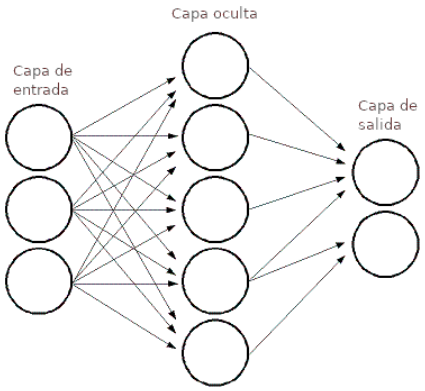


Figura 4.4: Capas completamente conectadas.

En este punto se observa que una red neuronal está formada por mínimo 3 capas; la primera que es la de entrada, esta formada por los nodos que reciben los valores con los cuales la red va a trabajar; la segunda capa se la llama capa oculta que aunque en este ejemplo se ha utilizado una, en aplicaciones reales su número es mayor y contiene los nodos que aplican un filtro utilizando una función llamada **Sigmoidea** que transforma junto con los pesos (llamados también parámetros) los datos de entrada; finalmente la capa de salida que también utiliza una función sigmoid para lograr los resultados deseados.

Una **función sigmoid** mostrada en la ecuación 4.12, es una función de activación que permite llevar el resultado de la multiplicación $w^T x$ a valores operables para la red,

así existen varias funciones de activación ya establecidas; una de las más comunes es mostrada en la Figura 4.5.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{4.12}$$

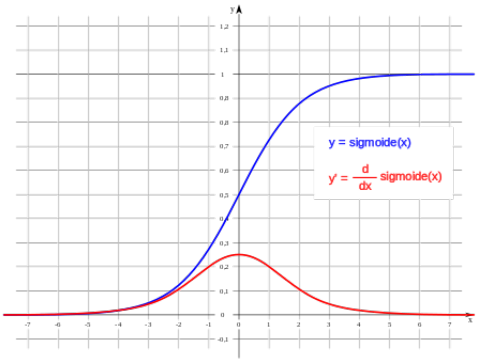


Figura 4.5: Función sigmoid y su derivada

Es así que en la siguiente red ya entrenada mostrada en la siguiente imagen podemos observar que, en la primera neurona de la capa oculta, su valor es el resultado de multiplicar la entrada con los pesos (1*0.8+1*0.2= 1) la función de activación sigmoid mostrado en la ecuación (4.12), transforma el valor de x=1 en la salida 0.73.

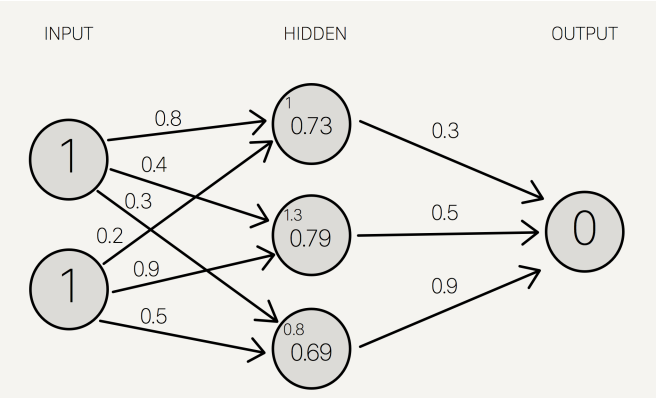


Figura 4.6: Simple red neuronal

Esta simple red neuronal arroja los siguiente valores:

Tabla 4.3: *Valores obtenidos de la red neuronal artificial para la salida XOR.*

x1	x2	salida real	salida deseada
0	0	0.77	0
0	1	1,13	1
1	0	1,13	1
1	1	0.77	0

4.3 Tamaño de una red neuronal

Claro esta que la red neuronal mostrada en la sección anterior esta muy lejos de solucionar un problema real, debido a que el aprendizaje seria muy limitado; las redes neuronales actualmente utilizadas en tareas como reconocimiento de lenguaje natural, tienen muchas más unidades de neuronas; ¿pero cómo se mide el tamaño de una red neuronal, existen dos maneras de medir el tamaño de una red neuronal, la primera es por la cantidad de unidades de neuronas, y la segunda es por la cantidad de parámetros; cuando se utiliza el número de unidades que posee la red, es importante indicar que la primer capa o capa 0 que son las de entradas no son tomadas en cuente, así la red anterior tendría 3+1= 4 neuronas o $[2 \times 3] + [3 \times 1] = 9$ pesos y $3 + 1 = 4$ bias.

4.4 Modelo de la red neuronal convolucional

El modelo de una red neuronal convolucional (RNC), ha demostrado ser excelente para el manejo de imágenes en lo que respecta principalmente a clasificación, [34]; gracias a las múltiples capas de convolución con las que pueden ser implementadas, es posible manejar gran cantidad de datos o imágenes con considerable estabilidad; varios son los modelos de arquitectura que se han creado para cubrir la diversidad de problemas en el procesamiento de imágenes que se presentan, entre algunas de ellas están *OxfordNet*, *Inceptron*, *letNet* (figura 4.7) y una de las primeras creada por los años 1990 fue ConvNet, ésta última ha demostrado excelente desempeño en tareas como clasificación de escritura a mano y detección de caras; muchas investigaciones en el estado del arte demuestran que este modelo aún se puede aplicar a muchos otros diferentes trabajos más, es por eso que la propuesta en esta investigación se centrará en el uso de *ConvNet* como alternativa para mejorar la salida del filtro BM3D. Para empezar las pruebas de su funcionamiento se utilizaron 4 capas ocultas, las dos primeras capas llamadas Convolucionales, la tercera fue la capa de reducción o *pooling* y la cuarta, la capa clasificadora, luego de esta última se encuentran las neuronas de salida que están totalmente conectadas a la capa anterior con una funcionalidad de igual manera como los tradicionales perceptrones múlti capas.

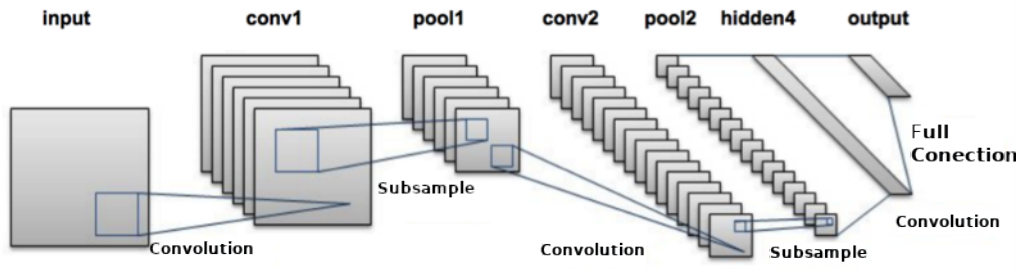


Figura 4.7: Red neuronal convolucional con arquitectura le-Net[11]

Los modelos de las redes neuronales convolucionales han sido implementados en varios framework principalmente basados en C++ y python así entre los más populares se tienen:

- TensorFlow : Google, Python
- Teano:Univ, Python
- Torch:C++, Lua, Python
- Tiny-dnn:C++
- Keras:Google, Python

4.4.1. Tiny-dnn una red neural implementada en C++

El framework Tiny-dnn creado utilizando el lenguaje C++14, implementa todos los modelos matemáticos que utiliza una red neuronal convolucional para poder procesar una imagen, todos estos modelos matemáticos están agrupados dentro de una clase llamada **network**, que utiliza dos formas de representar la arquitectura de la red; la primera llamada **secuencia** que consiste en utilizar la estructura *lista enlazada* donde cada capa tiene al menos una capa predecesora y una capa sucesora; y la segunda es llamada **grafos** donde cada nodo del grafo es una capa, y cada vértice mantiene el tensor y su gradiente, al momento de crear una instancia de la res se la puede crear de las siguientes dos maneras:

- Network (secuencial) mynet
- Network (graph) mynet

4.4.2. El problema de la compuerta XOR

Un ejemplo muy simple que permite comprender de manera muy didáctica la funcionalidad de este interesante framework implementado en C++, se da en la obtención de las salidas obtenidas por la compuerta XOR; el funcionamiento de esta compuerta consiste en que al recibir 2 entradas $(x1, x2)$ binarias se obtiene una salida (y)también binaria de acuerdo a la tabla 4.4 que se muestra seguidamente.

Tabla 4.4: *Entradas y salidas de la compuerta XOR.*

$x1$	$x2$	y
0	0	0
0	1	1
1	0	1
1	1	0

Lo que hace del funcionamiento de la compuerta XOR, especial y candidato para ser imitado utilizando una red neuronal artificial, es el hecho de que una misma respuesta tiene dos estimulaciones diferentes, así para la entrada (0, 0) el resultado es 0, pero si se proporciona una entrada de (1, 1) se observara que el resultado también sera 0. Esta aparente contradicción da indicios que se está tratando con variables de entrada y salida que tiene una relación no determinística, y cuando es así se debe buscar la existencia de una relación probabilística, trabajo conseguido por una estructura de red con multiples perceptrones como muestra la Figura 4.8 siguiente.

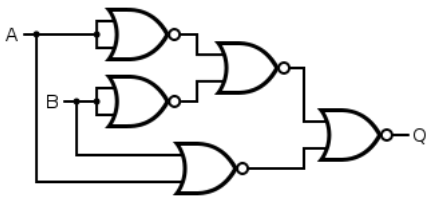


Figura 4.8: Estructura multiperceptron para imitar el funcionamiento de la compuerta XOR.

Código 4.1: Líneas de código utilizando el framework **tiny-dnn** para imitar el funcionamiento de la compuerta XOR.

```
1 #include "tiny_dnn/tiny_dnn.h"
2
3 using namespace tiny_dnn;
4 using namespace std;
5
```

```
6  int main(int argc, char *argv[]){network<sequential> net;
7      net << layers::fc(2,3) << activation::tanh()
8          << layers::fc(3,1) << activation::tanh();
9
10     vector<vec_t> input_data  {{0,0}, {0,1}, {1,0}, {1,1}
           };
11     vector<vec_t> desired_out {{0},   {1}, {1},     {0}
          };
12
13     size_t batch_size = 1;
14     size_t epochs = 1000;
15     gradient_descent opt;
16     net.fit<mse>(opt, input_data, desired_out, batch_size
          , epochs);
17
18     double loss = net.get_loss<mse>(input_data,
          desired_out);
19     cout << "mse : " << loss << endl;
20     net.save("xor_net");
21     return 0;
22 }
```

Las líneas de código en C++ mostradas, la librería tiny_dnn, para crear el modelo de red mostrada en la Figura 4.8, el cual simular el comportamiento estocástico de la compuerta XOR, en esta implementación solo se necesito una red con tres capas totalmente conectadas que son: 1) la capa de entrada, 2) la capa oculta y 3) la capa de salida. La librería tiny-dnn explícitamente implementa la capa oculta y la capa de salida, estas dos capas utilizan una función de activación la cual le indica al sistema neuronal si esa neurona debe o no debe ser tomada en cuenta para los cálculos de la siguiente capa o para la salida; en este ejemplo se utiliza la función de activación **tanh** que es una de las 12 funciones de activación que permite seleccionar este framework dependiendo de la aplicación que se haga.

Los valore utilizados en la linea de código 11 y 12 corresponden a las entradas y las salidas ideales de la compuerta XOR, como se pueden dar cuenta, para dos entradas diferentes existe una salida idéntica, lo que lo hace díficil de manipular determinísticamente.

Los pesos (weights) iniciales que el framework asigna son poco a poco actualizados en el proceso de gradiente descendente utilizando 1 bach con 1000 epochs, y utilizando como función de costo el error medio cuadrático (MSE).

Finalmente, los nuevos pesos son guardados en un archivo “xor_net” , para poder ser utilizados con datos de la prueba o verificación como muestra el siguiente código.

Código 4.2: Programa principal tiny-dnn que implementa el funcionamiento XOR

```
1
2  #include "tiny_dnn/tiny_dnn.h"
3  #include <cstdlib>
4
5  using namespace tiny_dnn;
6  using namespace std;
7
8  int main(int argc, char *argv[]){network<sequential> net;
9      net.load("xor_net");
10
11     vec_t in = {atof(argv[1]), atof(argv[2])};
12     vec_t result = net.predict(in);
13     cout << result.at(0) << endl;
14
15     return 0;
16 }
```

4.4.3. Red neuronal convolucional con Tiny-dnn

En el sencillo ejemplo de imitar el comportamiento XOR, se muestra un limitado funcionamiento del framework Tiny-dnn, puesto que la arquitectura de la red implementada solo se limita a una mlp (*multi layer peceptro*) con 3 capas totalmente concentradas. Para esta investigación y en si para el procesamiento de imágenes, es necesario utilizar mayor cantidad de capas con una mayor diversidad, para poder extraer mayores patrones de una gran cantidad de datos de entrenamiento.

Tiny-dnn soporta la utilización en sus capas ocultas de capas convolucionales y capas que realizan *downsamples* a más de las capas totalmente conectas como se muestra en la porción de código siguiente:

Código 4.3: *Porción de código en tiny-dnn que muestra como se implementa una red neuronal de 4 capas para procesar una imagen en escala de grises.*

```
1 // input: 32x32x1 (1024 dimensions) output:10
2     network<sequential> net;
3     net << convolutional_layer(32,32,5,1,6)    << tanh()
4         << average_pooling_layer(28,28,6,2)    << tanh()
5         << fully_connected_layer(14*14*6,120) << tanh()
6         << fully_connected_layer(120,10);
```

En esta porción de código se construye un modelo de red neuronal artificial convolucional, con 3 capas ocultas que utilizan la función de activación **tanh** para reducir la complejidad de su entrada. La primera capa procesa una imagen de dimensión 32*32x1 píxeles con una kernel de 5x5 a una salida de 28*28, la segunda capa recibe una imagen reducida a 28*28 y utilizando un kernel de 6*6 brinda una salida del 14*14, finalmente, una capa fullconnect que recibe 120 bit y lo transforma en 10 bit.

En otro caso para procesar una imagen a colores se utiliza el siguiente pedazo de código;

Código 4.4: *Porción de código en tiny-dnn que muestra como se implementa una red neuronal de 4 capas para procesar una imagen RGB.*

```
1 // input: 32x32x3 (1024 dimensions) output:40
2     network<sequential> net;
3     net << convolutional_layer(32,32,5,3,6)    << relu()
4         << average_pooling_layer(28,28,6,2)    << relu()
5         << fully_connected_layer(14*14*6,120) << tanh()
6         << fully_connected_layer(120,10);      << softmax
              ();
```

En esta otra porción de código se observó como Tiny-dnn permite utilizar varias funciones de activación a más de la *tanh*() para realizar una mejor clasificación.

Luego de haber creado el modelo de red neuronal Tiny-dnn, el siguiente paso es entrenar a la RED, y para eso es necesario contar con datos de entrada (*features*) y datos de salida (*label*), así como también definir el número de *batch*, y el número de *epochs* en que se va a dar este entrenamiento los cuales son asignados en la siguiente porción de código:

Código 4.5: *Porción de código en tiny-dnn que muestra cómo se alimenta a una red neuronal y como se comienza el proceso entrenamiento.*

```
1
2         std::vector<vec_t> input_data {{1,0},{0,2}};
3         std::vector<vec_t> desired_out{{2},  {1}};
4         size_t batch_size=1;
5         size_t epochs=30;
6
7         net.fit<mse>(opt,input_data, desired_out,
              batch_size,epochs);
```

Tiny-dnn necesita suministrar los datos de entrada, así como los datos de salida deseados, a través de una estructura vectorial **vec_t**, pero cuando los datos de entrada y salida corresponden a imágenes, la forma de llenar estas estructuras no es tan sencilla como los ejemplos antes desarrollados, para lo cual se recurre a herramientas o algoritmos especiales.

Código 4.6: *Líneas de código en tiny-dnn que muestra cómo se alimenta las imágenes a la red neuronal convolucional.*

```
1
2           #include <opencv2/imgcodecs.hpp>
3 #include <opencv2/imgproc.hpp>
4 #include <boost/foreach.hpp>
5 #include <boost/filesystem.hpp>
6 using namespace boost::filesystem;
7
8 // convert image to vec_t
9 void convert_image(const std::string& imagefilename,
10                    double scale,
11                    int w,
12                    int h,
13                    std::vector<vec_t>& data)
14 {auto img = cv::imread(imagefilename, cv::
      IMREAD_GRAYSCALE);
15     if (img.data == nullptr) return; // cannot open, or
          it's not an image
16
17     cv::Mat_<uint8_t> resized;
```

```
18      cv::resize(img, resized, cv::Size(w, h));
19      vec_t d;
20
21      std::transform(resized.begin(), resized.end(), std::
            back_inserter(d),
22                     [=](uint8_t c) {return c * scale; });
23      data.push_back(d);
24  }
25
26  // convert all images found in directory to vec_t
27  void convert_images(const std::string& directory,
28                      double scale,
29                      int w,
30                      int h,
31                      std::vector<vec_t>& data)
32  {path dpath(directory);
33
34      BOOST_FOREACH(const path& p,
35                    std::make_pair(directory_iterator(dpath
                          ), directory_iterator())) {if (
                          is_directory(p)) continue;
36          convert_image(p.string(), scale, w, h, data);
37      }
38  }
```

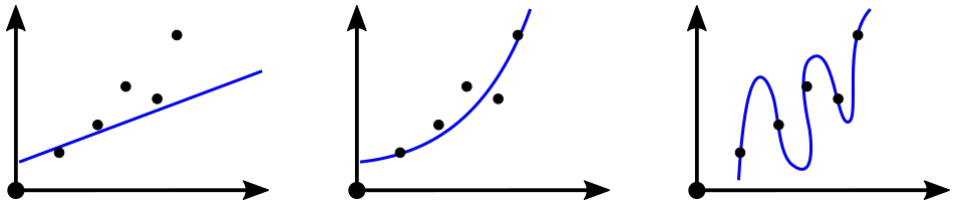
Estas líneas de código no solo permiten llevar una imagen a la estructura vec_t, sino que también permitió transformar todo un directorio de imágenes, que es el caso real para datos de entrenamiento.

4.4.4. Parámetros epoch y batch

Después de haber descrito claramente las características y funcionamiento de la red neuronal artificial y determinar la variante a utilizar para esta investigación, la cual fue la Red Neuronal Convolucional Autoencoder (RNCA) que fue implementada en el siguiente capítulo, existen dos parámetros que van a determinar el tiempo y la precisión con que vamos a obtener los resultados deseados.

Uno de estos parámetros es llamado **epoch** que tiene un valor que va desde 1 hasta

un valor muy alto, y corresponde a la cantidad de veces que los datos de entrada son procesados para actualizar los hiperparámetros; por ejemplo, si le damos a la red un *epoch=1*, significa que una sola vez vamos a procesar la información de entrada y una sola vez los hiperparámetros serán actualizados, lo que significa que se obtendrá una muy baja precisión en los resultados; por otra, parte dar un valor muy alto a este parámetro por ejemplo *epoch=1000* involucraría seguir procesando la información muchas veces más, aun después de haber llegado a encontrar los parámetros adecuados alejándose de la solución y convirtiendose en un sobreajuste (*overfiting*) lo que también puede perjudicar los resultados de predicción o clasificación según sea el caso, problema que se describe en la Figura 4.9 siguiente.



(a) *Underfitting* (b) *Correcto* (c) *Overfitting*

Figura 4.9: *(a) Bajo ajuste (Underfitting); (b) Ajuste correcto (Correcto); (c) Sobre ajuste (Overfiting).*

No hay que olvidar que el proceso de aprendizaje (J(w)) de la red neuronal es un arduo camino(Gradiente) en búsqueda de un estado óptimo (J_{min} (w)) dados por los pesos (w) mostrado en la Figura 4.10.

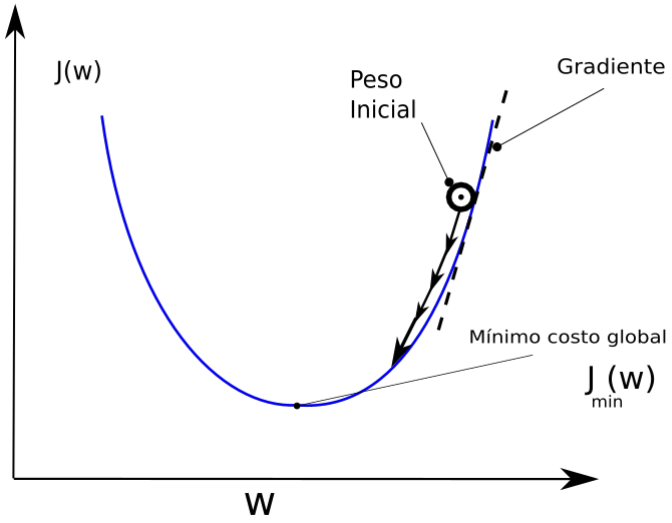


Figura 4.10: Efecto de las interacciones en el proceso de gradiente descendente.

Los valores del parámetro **batch** en cambio están en el rango de 1 a la cantidad

de imágenes de entrenamiento utilizadas, es decir, si se utiliza un batch=1, cada epoch va a ejecutarse procesando una imagen a la vez. El valor del batch utilizado tiene una impacto sobre el uso de la memoria, pues mientras más grande será el valor de batch más memoria se utilizará en el proceso.

CAPÍTULO 5

Implementación del filtro BM3D-RNCA

Este novedoso filtro BM3D-RNCA combina dos de los algoritmos que se han destacado en el estado del arte del procesamiento de imágenes, estos son: 1) el Block Matching 3D (BM3D) y 2) la red neuronal convolucional configurada para trabajar con una topología autoencoder (RNCA). Esta combinación logró recuperar de una imagen con ruido las características de borde, contornos, texturas e intensidades de colores de una manera más eficiente de lo que lo haría cada método de forma individual al menos en imágenes con un alto nivel de ruido $\sigma > 40$.

Para implementar esta nueva combinación de filtros se recurrió a líneas de código disponibles bajo licencia de software libre, así se accedió a códigos ya probados de los dos algoritmos antes mencionados, los códigos originales fueron creados en dos de los lenguajes comercialmente más utilizados. El algoritmo BM3D fue implementado en el lenguaje C++ el cual fue ejecutado sin problemas en esta investigación en la versión 8.3.1, y el segundo algoritmo de red neuronal convolucional fue desarrollado en *Python* y ejecutado en esta investigación en su versión 3.7,2; para el algoritmo BM3D se utilizó la implementación cuyo creador fue Marc Lebrun y cuyo código original puede encontrarse en [21], mientras que para el algoritmo CNN se utilizó la implementación creada por el equipo Google-brain de Google llamado TensorFlow [29], este último se eligió por la facilidad que ofrece en la creación y manipulación de diversas arquitecturas de redes neuronales de forma fácil utilizando un conjunto de librerías API's de más alto nivel llamadas kera, e implementada también en python.

El algoritmos, BM3D implementado en C++ y la red neuronal convolucional implementada en Python, fueron ejecutados en su face de prueba en un servidor con las siguientes características: procesador Intel ® Xeon® CPU E5-2640 @ 2.50 GHz, 2494 Mhz, 6 Core(s), 12 Logical Processor(s) con 96 GB de memoria principal, además el computador tiene incorporada una GPU (NVIDIA Tesla k20m), en cuanto al software se utilizó como Sistema Operativo Microsoft Windows Server 2012 R2 Datacenter.

Una de las restricciones para la fase de prueba fue en cuanto al tamaño de la imagen, que debía ser múltiplo de 32, para que se acople a la necesidades de la red neuronal. De esta forma, el máxima tamaño utilizado fue de 512, múltiplo de 32, por lo que la imagen se llevó al tamaño de 512×512. El filtro BM3D-RNCA regenera una imagen de tamaño 512×512 afectada con ruido de tipo Aditivo Blanco Gaussiano; esta imagen, que para efecto de esta investigación es referida como IO por ser la imagen objeto de este trabajo, fue tratada en una primera etapa por el algoritmo BM3D para reducir el ruido y obtener una imagen estimada de lo que posiblemente seria la imagen libre de ruido; mientras que en una segunda etapa, la imagen estimada pasa a convertirse en los datos de entrada de la red neuronal convolucional-autoencoder (RNCA), el cual consiguió a la salida una imagen recuperada muy parecida a la verdadera imagen libre de ruido.

Para lograr que la Red Neuronal Convolucional Autoencoder (RNCA) obtenga el conocimiento necesario que permita recuperar los bordes, contornos y texturas, se utilizaron 36 imágenes libre de ruido capturadas de la misma escena de la imagen objeto IO; estas imágenes fueron divididas en 29 (80 %) para utilizarlas como datos de entrenamiento y 7 (20 %) para datos de validación. Todas estas 36 imágenes en esta investigación fueron referidas como el conocimiento de la imagen objeto CIO_n, donde $n = 1, 2, \ldots, 36$; es importarte recalcar que estas imágenes son parecidas a la imagen objeto $CIO \simeq IO$, porque fueron capturadas en el mismo escenario.

En lo que queda del capítulo se explicará de forma detallada el proceso que en cada una de las siguientes etapas bien definidas se ejecuta hasta lograr la meta final que es liberar la imagen del ruido, se incluye la etapa de pre-procesamiento en la cual se consigue la imagen con ruido (imagen objeto IO) a partir de una imagen libre de ruido:

1. **BM3D y la estimación de la imagen:** La imagen objeto IO fue sometida al proceso de eliminación de ruido del algoritmo BM3D implementado por Marc Lebrun (cuyo funcionamiento se explicó en el capítulo 5), el cual arroja (después de aplicar un filtro intermedio y un filtro final) la imagen objeto estimada $\widehat{IO}$, y que aunque logra reducir en un gran nivel el ruido, su calidad no llega a compararse a la original.

2. **Super resolución de la imagen estimada:** La imagen estimada que se obtiene como salida del algoritmo BM3D, es tomada para alimentar la red neuronal convolucional de tipo autoencoder implementada utilizando el framework *tensorflow* con las APIs de *keras*, esta arquitectura mejora baja resolución $\widehat{IO}$ utilizando el

conocimiento obtenido de las CIO_n.

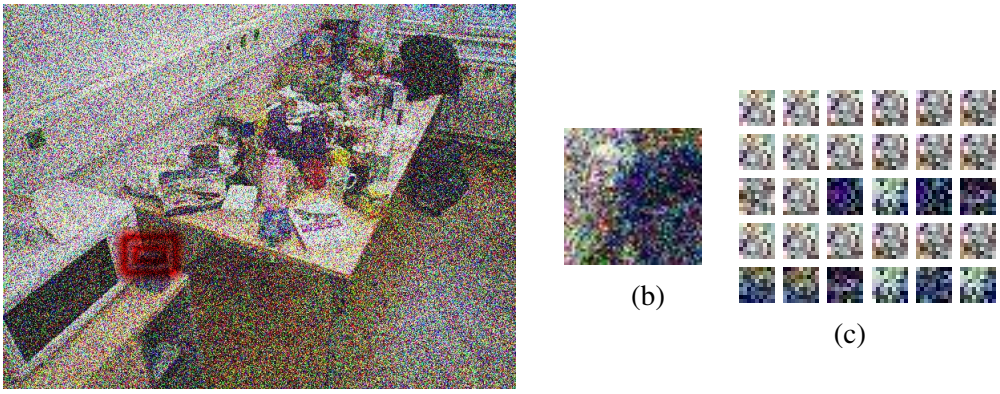
3. **Comparación de los resultados:** Se calcula el indicador PSNR, para cuatro productos obtenidos durante el proceso de estimación: 1) la imagen con ruido, 2) la imagen obtenida luego de aplicar la transformada discreta del coseno (TDC), que es parte del pre-procesamiento de algoritmo BM3D, 3) la imagen obtenida al final del algoritmo BM3D, lograda al aplicar el filtro de wiener y 4) la imagen con super-resolución obtenida por la red neuronal artificial autodecoder.

5.1 BM3D y la estimación de la imagen

En esta etapa se asume que ya se cuenta con la imagen dotada de ruido IO y se procede a estimar la imagen, la cual es realizada en dos sub-procesos, 1) se aplica la Transformada Discreta de Wavelet (DWT) aunque alternativamente también se puede escoger la Transformada Discreta del Coseno (DCT) según la disponibilidad de recursos, y 2) a la salida del algoritmo se aplica el filtro de Wiener para la estimación final de este algoritmo.

5.1.1. Primera estimación Transformada Discreta Wavelet(DWT)

Por limitaciones del DCT y DFT que se explicó en el capítulo 3 sección 3.6, para esta investigación se seleccionó la Transformada Discreta Wavelet utilizando la señal madre biortogonal llamada bior1.5. Para lograr esto se ejecuta el algoritmo pasando entre sus argumentos la palabra **‘‘BIOR”**, es importante recordar que esta transformada se aplica individualmente a arreglos 3D de bloques de tamaño 8×8 (ver figure 5.1-c) que se encontraron dentro de una ventana de dimensiones 40×40 (ver figura 5.1-b), por eso el proceso se llama filtro colaborativo, donde para reconstruir el parche estimado de forma rápida y sencilla se promedia los píxeles que se encuentran en la misma posición relativa. El problema de este filtro colaborativo utilizando simplemente el promedio entre píxeles es asumir que todos los bloques son muy similares, que si así fuera el caso la estimación fuera imparcial, pero en una imagen natural de baja resolución los bloques van a ser disimilares por lo que la estimación va a estar sesgada [6]; la ventana se desliza por lo ancho y alto de la imagen y al final del proceso se obtiene el primer producto a ser evaluado y comparado utilizando el indicador PSNR.



(a) (b) (c)

Figura 5.1: (a) Imagen objeto IO muestra ubicación de la ventana $n \times n$ utilizada, (b) ventana o región de tamaño $n \times n$, donde se busca los parches, (c) parches similares al parche central del referencia

.

5.1.2. Segunda estimación aplicando la transformada de Wiener

Por el inconveniente explicado en la sección anterior es necesario reforzar el filtro y para solucionar este inconveniente se utiliza el Filtro en la transformación del dominio Wiener con un apropiado umbral $\mathbf{W}_{S^{wie}_{x_R}}$, pero aun así los resultados no son buenos para ruidos bastante densos (AWG $\sigma > 25$), y esto se debe a que el filtro de Wiener no tiene la capacidad de ir aprendiendo del contexto de la imagen a medida que se pone en marcha el proceso de eliminación de ruido. Esta habilidad de aprendizaje puede ser brindada utilizando redes neuronales convolutivas (CNN) que con el creciente desarrollo de librerías y hardware para mejorar su rapidez, esta ya pueden ser aplicadas [2] e implementadas.

Al finalizar la ejecución del algoritmo BM3D se debe obtener una imagen estimada, que como se muestra en la Figura 5.2, aunque se rescata gran cantidad de bordes, contornos y texturas, la imagen se muestra con una baja resolución.

(a) (b)

Figura 5.2: (a) Imagen original libre ruido (b) Imagen objeto estimada $\widehat{IO}$ por el algoritmo BM3D ($\sigma = 60$)

Este procedimiento no solo se aplica a la imagen objeto IO sino que también a las 36 imágenes utilizadas por la red neuronal artificial (para aprender recuperar la resolución original de la imagen, a través de un largo proceso de entrenamiento y prueba explicado en el capítulo 4), también tendrán que ser estimadas por el algoritmo BM3D, a fin de obtener los elementos de entrada (input) y las salidas deseadas (label) que necesita los algoritmos de propagación hacia delante y propagación hacia atrás de la red para encontrar los parámetros que permitan llevar la imagen a una alta resolución.

5.2 Restauración de la imagen estimada

Después de obtener la imagen estimada $\widehat{IO}$ por el filtro BM3D, se observa en la Figura 5.2–(b) que esta imagen no alcanza el nivel de calidad visual de la imagen original libre de ruido, dando la apariencia de una versión de baja resolución (LR) de la imagen original.

La meta en esta etapa es transformar esta imagen de baja resolución (LR) (ver Figura 5.3-(a)) a su equivalente de alta resolución (HR) (ver Figura 5.3-(b)), y esto fue posible utilizando una red neuronal artificial que fue capaz de aprender patrones similares entre la imagen de baja resolución con la imagen de alta resolución a fin de poder interpolar los píxeles faltantes, este proceso se le explicó detalladamente en el capítulo 4.

(a) (b)

Figura 5.3: (a) la imagen estimada por el algoritmo BM3D con un $\sigma = 100$; (b) la imagen referencia IR libre de ruido a la cual se quiere llegar.

Muchas fueron las topologías de redes neuronales evaluadas en esta investigación llegando a la conclusión que para restituir la resolución de la imagen fue necesario utilizar una red Siames, propuesto por LeCum et al [13] la cual consiste en *utilizar dos arquitecturas del tipo LeNet acoplada como siamesas cuya función de pérdida trabajarán en sentido contrario*, esto le dará una característica de semi-supervisada.

Este tipo de redes es el que en esta investigación se utilizó para recuperar la alta resolución de la imagen, actualmente se la conoce como redes neuronales artificiales con topología autoencoder.

5.2.1. Modelo y arquitectura RNCA

El objetivo de la RNCA en su fase de entrenamiento es encontrar una función con los parámetros que mapee el espacio de alta dimensión que crea las imágenes de alta resolución deseada, a un espacio de menor dimensión de la imagen estimada por el filtro BM3D considerada de baja resolución. El mapa de la relación entre alta resolución y baja resolución servirá para llevar en la etapa de prueba la imagen de baja resolución a su equivalente de alta resolución.

Expresado de forma matemática, se cuenta con un vector $I = \{x_1, \ldots, x_P\}$ donde $X_i \in \mathbb{R}^D$, y se desea encontrar los parámetro de la función $G_w : R^D \longrightarrow R^d$ con $d << D$ donde la función debe cumplir con características de: 1) mantener una medición simple para el espacio de salida (tal como la distancia euclidiana) que aproxime la relación de los vecinos en el espacio de salida; 2) que el mapeo no esté restringido a la implementación de una simple medida de la distancia en el espacio de la entrada, sino que también pueda ser capaz de aprender complejas transformaciones, y 3) las

dos características anteriores deben ser cumplidas incluso para los patrones que tienen relación con los vecinos desconocidos.

La red neuronal aprenderá a interpolar los píxeles que faltan en la imagen de baja resolución para convertirse en una imagen de alta resolución, este aprendizaje lo logró determinando como cambian los píxeles entre las 36 imágenes libre de ruido CIO y su correspondiente imagen estimada $\widehat{CIO}$, por el algoritmo BM3D.

Para lograr este tipo de aprendizaje la red neuronal convolucional (ConvNet) con topología **autoencoder**, debe contener tres bloques con funciones bien definidas: 1) Extracción de características, 2) Mapeo y 3) Reconstrucción de la imagen. Las capas convolucionales de cada bloque son denotadas por $n \times Conv(s, d)$, donde las variables n, s y d indican el número de capas, tamaño del filtro y las dimensiones de las características respectivamente [22].

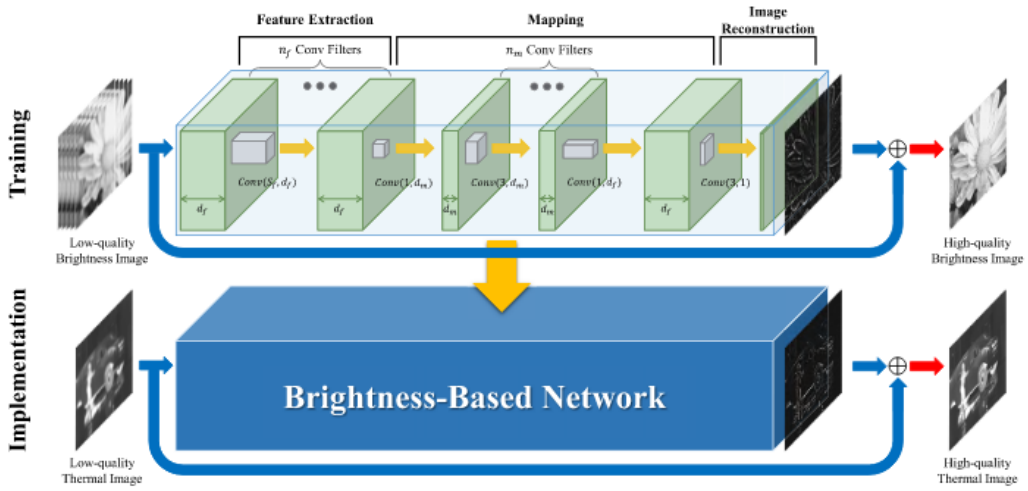


Figura 5.4: Estructura de la red neuronal convolucional con topología autoencoder propuesta para esta investigación (de [22]).

En esta topología de red la capa de salida es la que discrimina, dando paso a la interpolación de píxeles faltantes utilizando el aprendizaje de las imágenes libre de ruido analizadas en la fase de entrenamiento. A diferencia del problema de reconocimiento en escritura a mano alzada analizado en el capítulo 4, donde las opciones de salida se limitan a 10; en la red propuesta para mejorar la resolución de la imagen las salidas pueden ser en el peor de los casos 290625 que es la cantidad de bloques que se obtuvo de la imagen de baja resolución, pero como los bloques al llevarlos a una estructura más simple pueden repetirse o ser muy parecidos se podría considerar un valor algo menor.

El siguiente código en python muestra la implementación de la red neuronal convolucional (ConvNet) con topología autoencoder, se utilizan 14 capas; 5 para la etapa encoder, 2 para la etapa de mapeo y 7 para la etapa de decoder, las capas de convolución utilizan la función de activación 'relu' y en todas las capas el filtro del kernel actúan sin olgura.

Código 5.1: *Porción de código en Python que muestra la implementación de red neuronal convolucional autoencoder.*

```
1
2  autoencoder = Sequential()
3
4  autoencoder.add(Conv2D(16, (3,3), activation='relu',
       padding='same', input_shape=(img_size,img_size,1)))
5  autoencoder.add(MaxPooling2D((2,2), padding='same'))
6  autoencoder.add(Conv2D(8,(3,3), activation= 'relu',
       padding='same'))
7  autoencoder.add(MaxPooling2D((2,2), padding='same'))
8  autoencoder.add(Conv2D(8,(3,3), strides=(2,2), activation
       ='relu', padding='same'))
9  autoencoder.add(Flatten())
10 autoencoder.add(Reshape((16,16,8)))
11 autoencoder.add(Conv2D(8,(3,3), activation='relu',
       padding='same'))
12 autoencoder.add(UpSampling2D((2,2)))
13 autoencoder.add(Conv2D(8,(3,3), activation='relu',
       padding='same'))
14 autoencoder.add(UpSampling2D((2,2)))
15 autoencoder.add(Conv2D(16, (3,3), activation='relu',
       padding='same'))
16 autoencoder.add(UpSampling2D((2,2)))
17 autoencoder.add(Conv2D(1,(3,3),activation='sigmoid',
       padding='same'))
```

En resumen, la arquitectura de la red neuronal convolucional autoencoder implementado trabajando con imágenes de tamaño 512×512 generan en cada capa las siguientes cantidades de parámetros:

Tabla 5.1: *Arquitectura de la red neuronal convolucional - autoencoder.*

Capas	Número de parámetro	Padding	Forma de Activación	Tamaño de Activación
Input Image	0	-	(512,512,1)	262,144
Conv2D(f=3,s=1)	160	same	(512,512,16)	4,194,304
MaxPool	0		(256,256,16)	1,048,576
Conv2D(f=12,s=1)	1160	same	(256,256,8)	524,288
MaxPool	0		(128,128,8)	131,072
Conv2D(f=0,s=1)	584	same	(128,128,8	131,072
MaxPool(f=2,s=2)	0		(64,64,8)	32,768
flatten	0		(32,786)	
Conv2D	584	same	(64,64,8)	32,768
UpSampling2D	0		(128,128,8)	131,072
Conv2D	584	same	(128,128,8)	131,072
UpSampling2D	0		(256,256,8)	524,288
Conv2D	1168	same	(256,256,16)	1,048,576
UpSampling2D	-		(512,512,16)	4,194,304
Conv2D	145	same	(512,512,1)	262,144
	4,385			

5.2.2. Imágenes de entrenamiento y validación

El escenario real que se intenta reproducir es el de no contar con la imagen original libre de ruido (I), pero si con la imagen con ruido (IO).

(a)

(b)

Figura 5.5: (a) Imagen original libre de ruido, la cual no se dispone (b) Imagen objeto estimada $\widehat{IO}$, por el algoritmo BM3D ($\sigma = 60$)

También se cuenta con varias imágenes parecidas (CIO) tomadas en la misma escena que la imagen objeto original, pero con leves variaciones en la posición de la cámara de captura.

(a) (b)

(c) (d)

Figura 5.6: (a) imagen objetivo (IO), (b-c-d) imagenes utilizadas para entrenar la red

En la Figura 5.6, se observa la imagen objeto IO y 4 imágenes CIO que servirán para el aprendizaje de la red, se puede apreciar como las imágenes fueron tomadas desde otro ángulo y distancia a la imagen objeto IO, estas 3 imágenes sumadas a otras 3 más brindaron el conocimiento necesario a la red neuronal convolucional para lograr recuperar los píxeles perdidos en el proceso de estimación del algoritmo BM3D.

En total fueron 36 las imágenes similares a la imagen objeto que fueron utilizadas para el proceso de entrenamiento y de prueba que permitió el aprendizaje de la red neuronal convolucional autoencoder. Cada imagen proporciona un par $(A_{C_{x,y}}[\widehat{IR}], A_{C_{x,y}}[IR])$, que corresponde al par (*input, output*), estas serán las imágenes de entrenamiento y validación.

Por otra parte, la red neuronal convolucional necesitó saber cómo llevar la imagen estimada y de baja resolución a su equivalente sin ruido y con alta intensidad, para lo cual cada una de las imágenes de conocimiento CIO, también fueron sometidas al proceso de contaminación y posteriormente al algoritmo BM3D para obtener su valor estimado.

(a) (b)

Figura 5.7: (a) imagen sin ruido (CIO) que forma parte del conocimiento de la red, junto a (b) una imágenes con ruido (CIO) que también forma parte del conocimiento, para eliminar el ruido de la imagen objeto.

La Figura 5.8 muestra un ejemplo del input y del output (label) que fueron cargados para entrenar la red, y también muestra dos bloques extraídos tanto de la imagen estimada (*input*), como de la imagen sin ruido (*label*), donde se puede observar a simple vista que hay un patrón de similitud que la red poco a poco en su proceso de entrenamiento lo irá corroborando para colocar una ponderación y así poder inferir. Para este proceso la red extraerá las estructuras básicas de la imagen basándose en gradientes utilizando el proceso de convolución con los kernel ([[2.],[0.],[-1.]] y [2.; 0.; -1.]), lo que hace a los bloques comparables digitalmente.

(a) (b)

(c) (d) (e) (f)

Figura 5.8: (a) imagen estimada por el algoritmo BM3D, (b) imagen referencia IR libre de ruido, (c y e) áreas específicas de la imagen $A_{C_{x,y}}[\widehat{IR}]$, (d y f) $A_{C_{x,y}}[IR]$

De la misma manera varias van a ser las imágenes de referencia de donde los kernel

utilizados en las capas convolucionales extraerán patrones que en un proceso de propagación hacia delante y propagación hacia atrás irán ajustando durante las 30 imágenes de entrenamiento y 6 imágenes de prueba para poder llevar imágenes de baja resolución (LR) a su equivalente imágenes de alta resolución (HR).

Con la finalidad de entrenar la red neuronal convolucional de topología autoencoder para obtener los valores óptimos de los hiper-parámetros que permitan recuperar la alta resolución (HR) de la imagen estimada por el filtro BM3D, se utilizaron como salida (*label*) 36 imágenes producto de 6 imágenes originales libre de ruido (de tamaño 512×512) capturadas del mismo escenario que la imagen objeto, y que al rotar cada una de estas imágenes (90, 180 y 270 grados) y reflejarlas horizontal y verticalmente se obtuvieron las 30 imágenes adicionales. Todas estas fueron sometidas también a la estimación del filtro BM3D para ser utilizadas como entradas (input) de la red.

(a) Original (b) Reflejo horizontal (c) Rotada 90^o

(d) Reflejo vertical (e) Rotada 180^o (f) Rotada 270^o

Figura 5.9: De la imagen original se obtuvieron 5 imágenes adicionales al utilizar la transformación de rotación y reflejo.

Es importante indicar que todas las imágenes fueron obtenidas del sitio web perteneciente a la Universidad Tecnológica de Viena [31], las características de estas fotos es que capturan una misma ubicación en la cocina de un hogar variando la posición de la cámara en varios ángulos, con la finalidad que un mismo detalle de la imagen sea

representada desde diferentes perspectivas y que nuestra red neuronal puede reconocerla en cada una de las imágenes.

5.2.3. Formato para cargar de datos a la red

La red neuronal convolucional con topología autoencoder debe ser alimentada por las n imágenes estimadas ($\hat{y}_n$).

$$y_n = autoencoder(\hat{y}_n) \tag{5.1}$$

Para poder alimentar el modelo autoencoder creado con *keras*, fue necesario llevar todas las imágenes al formato de base de datos brindado por el estándar NIST, el cual consiste en llevar las imágenes a un formato binario que se desprenda del formato natural *jpg*, *png*.. etc. de las imágenes. En esta investigación se crea la base de datos MNIST con imágenes capturadas al interior de una casa, el código en C++ que transforma las imágenes a formato binario puede ser encontrado en [23].

La librería MNISTEN implementada en C++, funciona transformando todos los archivos de imágenes que se encuentran en un directorio agrupado por categorías, a dos archivos de formato binario los cuales guardan relación pues el uno contiene los valores que ingresarán a la red y el otro los *label* o valores con los cuales se comparará la salida de la red.

```
.\
|--Categoria1
|     |--a.jpg
|     |--b.jpg
+--Categoria2
|     |--c.jpg
|     |--d.jpg
```

Así, luego de procesar esta jerarquía de directorios y archivos utilizando las librerías *MNISTEN*, se obtienen los siguientes archivos que contienen las imágenes pero en binario, listos para ser procesados por la red neuronal.

```
prefix_train_images.idx3
prefix_train_labels.idx1
```

En la red neuronal convolucional con topología autoencoder, los datos de entrenamiento son las imágenes estimadas y las etiquetas (*labels*) serán las imágenes libres de ruido, por lo tanto, la etiquetación debe ser 1 a 1, el archivo de categoría (*label*) no se utiliza, cada imagen de un archivo de entrada tendrá su par en el archivo de salida.

Tabla 5.2: *Directorios con las imágenes de entrenamiento.*

```
.\                                  .\
|--Categoria1                       |--Categoria1
|     |--est_0000001.jpg            |     |--img_0000001.jpg
|     |--est_0000002.jpg            |     |--img_0000002.jpg
|     |--........                   |     |--........
|     |--........                   |     |--........
|     |--est_0000060.jpg            |     |--img_0000060.jpg
```

A la izquierda de la Tabla 5.2 se contó con un directorio de imágenes estimadas que fueron los datos de entrada al autoencoder, y a la derecha de la tabla se contó con las imágenes libres de ruido que fueron las etiquetas con las cuales se compararon los resultados obtenidos en el proceso de encadenamiento para ajustar los parámetros durante el entrenamiento. Estas imágenes agrupadas en directorios fueron procesadas por las librerías *MNIST* para producir los 4 archivos mostrados en la Tabla 5.3, de los cuales sólo se tomarán aquellos que contienen las imágenes en binario y los dos que contienen las etiquetas (*label*) se descartarán por no ser necesarias en esta topología.

Tabla 5.3: *Resultado de la librería MNIST, donde se elimina las salida (labels) porque todos las entradas (imputs) pertenecen a una sola categoría*

Archivo binario obtenido de un directorio con las imágenes estimadas del conocimiento ($\widehat{CIO}$)	Archivo binario obtenido de un directorio con las imágenes del conocimiento (CIO)
`prefix_train_i_images.idx3` ~~prefix_train_labels.idx1~~	`prefix_train_o_images.idx3` ~~prefix_train_labels.idx1~~

5.2.4. Campos receptivos de entrada y su carga

La primera capa de la red neuronal convolucional–autoencoder, la cual contiene los campos receptivos, receptan cada elemento de la matriz o matrices de las imágenes, que junto con la segunda capa extraen los primeros parámetros de la imagen que se va

ajustando a medida que el proceso de entrenamiento se va dando. La topología auto-encoder utilizada en esta investigación fue implementada utilizando las librerías keras de Python; el proceso de cargar las imágenes y normalizar cada píxel para adecuarlos al formato exigido por el framework Tensorflow se muestra en las siguientes líneas de código.

Código 5.2: *Porción de código en Python que muestra la carga de las imágenes de entrenamiento y prueba.*

```
1   import os
2   from IPython.display import Image, SVG
3   import matplotlib.pyplot as plt
4   import numpy as np
5   import keras
6
7
8   def extra_data(filename,num_images):
9       with gzip.open(filename) as bytestream:
10          bytestream.read(16)
11          buf=bytestream.read(512*512*num_images)
12          data = np.frombuffer(buf,dtype=np.uint8).astype
                (np.float32)
13          data=data.reshape(num_images,512,512)
14  return data
15
16  # Carta de las imagenes para entrenar la red
17
18  x_train_i=extract_data('prefix_i_train_images.idx3.gz'
        ,30)
19  x_train_o=extract_data('prefix_o_train_images.idx3.gz'
        ,30)
20
21  # Carta de las imagenes para probar la red
22
23  x_test_i=extract_data('prefix_i_test_images.idx3.gz',6)
24  x_test_o=extract_data('prefix_o_test_images.idx3.gz',6)
25
26  # Carga de las imagen objeto para predecer la imagen sin
        ruido
```

```
27
28  x_objeto_i=extract_data('prefix_i_objeto_images.idx3.gz'
        ,1)
29
30  #Normalizacion de cada pixel de la imagen
31
32  max_value=float(x_train_i.max())
33  x_train_i=x_train_i.astype('float32')/max_value
34  x_train_o=x_train_o.astype('float32')/max_value
35
36  x_test_i=x_test_i.astype('float32')/max_value
37  x_test_o=x_test_o.astype('float32')/max_value
```

Como se observa en el código anterior, de las 36 imágenes en formato binario que alimentan el conocimiento de la red neuronal; 22 imágenes (60 %) son utilizadas para el entrenamiento y 14 imágenes (40 %) se utilizan en la etapa de prueba de la red.

Antes de ejecutar el proceso de entrenamiento la red debe saber: 1) con qué algoritmo de optimización trabajar, 2) cómo se evaluarán las pérdidas para optimizar los parámetros en un el proceso de aprendizaje y finalmente, 3) definir la cantidad de epochs y batch que definirán la cantidad de interacciones y el tiempo a tomar hasta el aprendizaje total de la red.

Código 5.3: *Porción de código en Python que muestra la forma en que se entrena la RNCA.*

```
1  '''
2   Asignando funcion de optimizacion y de perdida
3  '''
4   autoencoder.compile(optimizer='adam', loss='
       binary_crossentropy')
5
6  '''
7  Ejecucion del proceso de aprendizaje
8  '''
9  autoencoder.fit(x_train_i, x_train_o, epochs=500,
       batch_size=5, shuffle=True, validation_data=(x_test_i,
       x_test_o))
```

Finalmente la predicción de la imagen sin ruido a partir de la image objeto (imagen estimada por BM3D) se realiza con el siguiente código:

Código 5.4: *Porción de código en Python que muestra la predicción aplicando el encoded y decoded.*

```
1 encoded_img=encoder.predict(x_objeto)
2 decoded_img=autoencoder.perdict(x_objeto)
```

Esta imagen es mostrada con el siguiente código:

Código 5.5: *Porción de código en Python que permite visualizar la imagen estimada.*

```
1   plt.figure(figsize=(400,400))
2   for i,image_idx in enumerate(randon_test_images):
3   plt.show()
```

5.2.5. Capa de salida y su optimización

Para la salida de la red neuronal fue necesario proveerla de dos funciones adicionales a más de la función de activación, estas funciones son: 1) la función de perdida para la cual se eligió *binary cross entropy* (BCE) y 2) una función de optimización(adam) de los parámetros el cual comienza en la salida y se propaga hacia la entrada en un proceso que se repite varias veces en el entrenamiento llamado encadenamiento hacia atrás.

La función de perdida *binary cross entropy* (BCE) se la eligió tomando en cuenta que normalmente los pixeles son normalizados en un rango entre 0 y 1 al dividir cada pixel para 255, esta función mide la perdida al comparar la salida estimada ($\widehat{y}$) con la salida deseada (y) de la forma $y * log(\widehat{y}) - (1 - y) * log(1 - \widehat{y})$.

CAPÍTULO 6

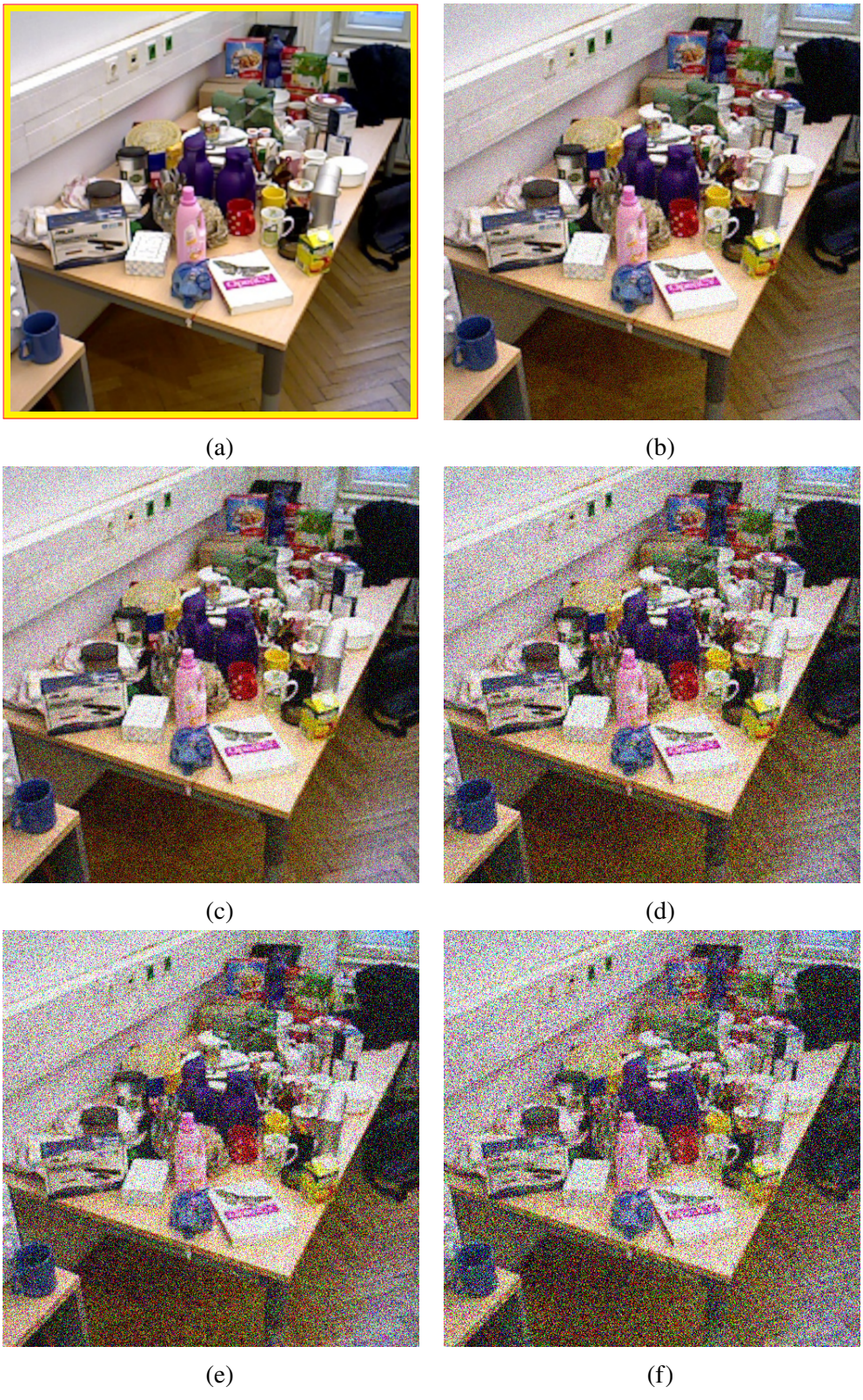
Evaluación del filtro BM3D-RNCA

En este capítulo se presentan y evalúan los resultados obtenidos en la implementación y prueba del nuevo filtro BM3D-RNCA el cual nace al combinar los algoritmos Block Matching 3D (BM3D) con la Red Neuronal Convolucional Autoencoder (RNCA). La estimación que se obtiene al final, es comparada con las estimaciones intermendias obtenidas al aplicar de forma individual los 3 filtros que forman parte del BM3D-RNCA, estos son la Transformada Discreta de Wavelet (DWT), el filtro de Wiener que es el resultado final del BMD3D, y la RNCA cada una aplicadas de forma aislada, el objetivo de todos estos filtros fue convertir una imagen afectada por ruido (denominada IO) en una imagen estimada (denominada $\widehat{IO}$) lo más parecida a la imagen original libre de ruido.

El ruido que se utilizó en esta investigación fue de tipo Aditivo Blanco Gaussiano (AWGN), y fue aplicado de forma inicial a una imagen original libre de ruido que fue capturada en un escenario al interior de una casa de hogar (*indoor*) Figura 6.1-(a) por una cámara de especificaciones desconocidas; la evaluación se realizó sometiendo la imagen a 5 niveles del tipo anteriormente indicado, variando el parámetro de varianza; $\sigma = 20$, $\sigma = 40$, $\sigma = 60$, $\sigma = 80$ y $\sigma = 100$ como se observa en la Figura 6.1.

Para medir el nivel de ruido existente en la imagen estimada se calculó la Proporción Máxima de Señal a Ruido (PSNR) utilizando un programa creado en python y que se puede encontrar en el repositorio GITHUB de esta investigación. Una de las restricción a este programa es que calcula el PSNR pero solo para las imágenes en escala de grises que para las necesidades comparativas de esta investigación son suficiente.

En la Figura 6.1 se observa cómo se presentan los diferentes niveles de ruido del tipo Aditivo Blanco Gaussian (AWGN) según su valor σ, aquí también se puede apreciar la diferencia con la imagen original; en la Tabla 6.1 se puede ver como varía el índice PSNR con el nivel de ruido aplicado.



(a) (b)

(c) (d)

(e) (f)

Figura 6.1: (a) Imagen original sin ruido ; (b) imagen con $\sigma = 20$; (c) imagen con $\sigma = 40$; (d) imagen con $\sigma = 60$; (e) imagen con $\sigma = 80$; (f) imagen con $\sigma = 100$.

Si comparamos la Figure 6.1-(a) con la Figure 6.1-(b), muchas veces no es fácil

apreciar de manera visual la diferencia entre los niveles de ruido, por lo que es necesario utilizar una apreciación numérica la cual es posible mediante la obtención del índice PSNR o el RMSE para los diferentes niveles de ruido mostrado en la siguiente tabla:

Tabla 6.1: *PSNR de la imagen con diferentes RBGA.*

Medidas	$\sigma = 20$	$\sigma = 40$	$\sigma = 60$	$\sigma = 80$	$\sigma = 100$
PSNR	25.9104dB	20.2398dB	17.1215dB	15.0813dB	13.6207dB
RMSE	166.739	615.319	1261.63	2018.12	2824.95

Para aplicar los diferentes niveles de ruido a cada una de las imágenes, también se utilizó el algoritmo BM3D, y cada una de las 5 veces que se ejecutó este algoritmo se varió el parámetro que alimentó a la variable sigma; así la primera ejecución se la realizó para obtener la imagen con nivel de ruido $\sigma = 20$ utilizando los siguientes parámetros:

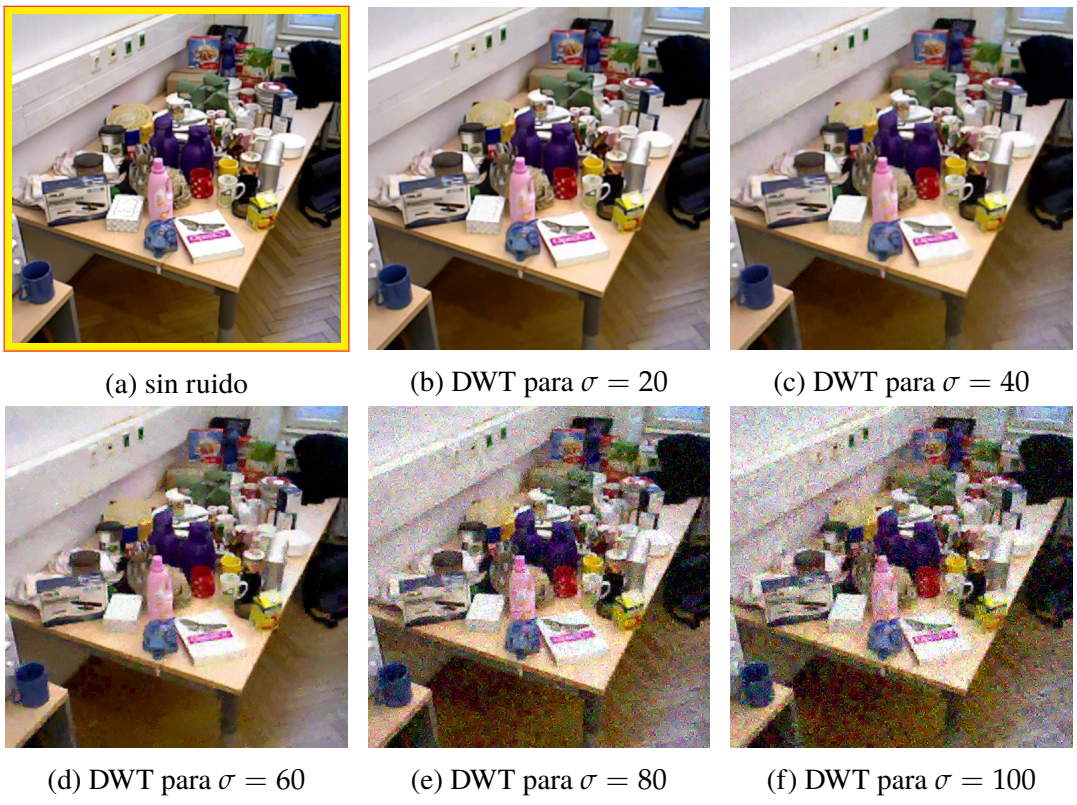
Tabla 6.2: *Parámetros iniciales para ejecutar el algoritmo BM3D.*

Orden	Variable	Valores
1	img	00000.png
2	sigma	20
3	img_noisy	00000_Noisy_20.png
4	img_basic	00000_Basic_20.png
5	$img\_denoised$	00000_Denoised_20.png
6	img_diff	00000_Diff_20.png
7	img_bias	00000_Bias_20.png
8	img_bias_diff	00000_DiffBias_20.png
9	$compute\_bias$	1
10	tau_2D_hard	bior
11	$useSD\_1$	0
12	tau_2D_wien	dct
13	$useSD\_2$	1
14	$color\_space$	rgb

Cada uno de estos parámetros fueron explicados en la Tabla 3.2 del capítulo 3 donde para la obtención de las imágenes con ruido no fue necesario probar la mejor combinación de valores a utilizar, pero para la estimación si fue necesario ya que por ejemplo la utilización del tipo de espacio de color influye en la calidad de la estimación.

6.1 Primera estimación

El algoritmo BM3D realiza una primera estimación aplicando una técnica en el dominio de la frecuencia llamada la Transformada Discreta del Wavelet (DWT), y cuyo resultado se observa visualmente en la Figura 6.2 y numéricamente en la Tabla 6.7.



(a) sin ruido (b) DWT para $\sigma = 20$ (c) DWT para $\sigma = 40$

(d) DWT para $\sigma = 60$ (e) DWT para $\sigma = 80$ (f) DWT para $\sigma = 100$

Figura 6.2: Primera estimación aplicando DWT; (a) Imagen original; (b) estimación para $\sigma = 20$; (c) estimación para $\sigma = 40$; (d) estimación para $\sigma = 60$; (e) estimación para $\sigma = 80$; (f) estimación para $\sigma = 100$

Tabla 6.4: *PSNR calculado en la primera estimación obtenida por el algoritmo BM3D donde se observa el trabajo realizado por el filtro DWT.*

Medidas	$\sigma = 20$	$\sigma = 40$	$\sigma = 60$	$\sigma = 80$	$\sigma = 100$
PSNR	34.9470dB	30.4696dB	26.5546dB	21.1796dB	19.6071dB
RMSE	20.8150	58.3607	143.754	495.591	711.818

6.2 Segunda y final estimación del algoritmo BM3D

La imagen estimada al finalizar el proceso del algoritmo BM3D es atribuido al filtro de Wiener, el cual como se indicó en el capítulo **??** utiliza el esquema de impulso finito(FIR).

(a) sin ruido (b) BM3D para $\sigma = 20$ (c) BM3D para $\sigma = 40$

(d) BM3D para $\sigma = 60$ (e) BM3D para $\sigma = 80$ (f) BM3D para $\sigma = 100$

Figura 6.3: Segunda y final estimación del algoritmo BM3D, aqui se aplica el filtro de Wiener; (a) imagen original; (b) estimación para $\sigma = 20$; (c) estimación para $\sigma = 40$; (d) estimación para $\sigma = 60$; (e) estimación para $\sigma = 80$; (f) estimación para $\sigma = 80$.

Tabla 6.5: *PSNR calculado en la segunda estimación obtenida del algoritmo BM3D.*

Medidas	$\sigma = 20$	$\sigma = 40$	$\sigma = 60$	$\sigma = 80$	$\sigma = 100$
PSNR	35.8439dB	31.7910dB	28.7892dB	23.9422dB	22.1432dB
RMSE	16.9311	43.0506	85.9318	262.335	396.9717

6.3 Tercera estimación de la imagen (BM3D-RNCA)

En la imagen con ruido $\sigma = 100$, estimada con filtro BM3D, Figura 6.3 (f) se puede observar que la estimación redujo las intensidades de colores comparada con la imagen original sin ruido, lo cual es mejorado utilizando el nuevo filtro BM3D-RNCA .

(a) sin ruido (b) BM3D-RNCA con $\sigma = 20$ (c) BM3D-RNCA de $\sigma = 40$

(d) BM3D-RNCA $\sigma = 60$ (e) BM3D-RNCA de $\sigma = 80$ (f) BM3D-RNCA de $\sigma = 100$

Figura 6.4: Estimación después de aplicar el nuevo filtro BM3D-RNCA; (a) Imagen original sin ruido; (b) estimación para $\sigma = 20$; (c) estimación para $\sigma = 40$; (d) estimación para $\sigma = 60$; (e) estimación para $\sigma = 80$; (f) estimación para $\sigma = 100$.

Tabla 6.6: *PSNR obtenido al aplicar el filtro BM3D-RNCA.*

Medidas	$\sigma = 20$	$\sigma = 40$	$\sigma = 60$	$\sigma = 80$	$\sigma = 100$
PSNR	37.1146dB	33.9734dB	31.0934dB	25.5534dB	25.4023dB
RMSE	6.50218	9.77333	13.7966	24.6975	30.9466

Utilizando el paquete *keras* de *python* se aplicó la red neuronal convolucional la cual se la entreno con los valores de *epoch* y *batch* necesarios para lograr una precisión de promedio 0.5.

6.3.1. Estimación con solo el RNCA

Para comprobar si tiene algún sentido la combinación BM3D con RNCA, se aplicó individualmente el algoritmo RNCA a la imagen objeto IO, y se obtuvieron los índices PSNR en los diferentes niveles de ruido.

(a) sin ruido (b) RNCA con $\sigma = 20$ (c) RNCA de $\sigma = 40$

(d) RNCA $\sigma = 60$ (e) RNCA de $\sigma = 80$ (f) RNCA de $\sigma = 100$

Figura 6.5: Estimación después de aplicar solo RNCA; (a) Imagen original sin ruido; (b) estimación para $\sigma = 20$; (c) estimación para $\sigma = 40$; (d) estimación para $\sigma = 60$; (e) estimación para $\sigma = 80$; (f) estimación para $\sigma = 100$.

La estimación se realizó buscando los parámetros de epoch y batch que permitieron obtener la misma precisión que el filtro BM3D-RNCA.

Tabla 6.7: *PSNR obtenidas al aplicar la RNCA, a la imagen con ruido.*

Medidas	$\sigma = 20$	$\sigma = 40$	$\sigma = 60$	$\sigma = 80$	$\sigma = 100$
PSNR	36.0134dB	32.1168dB	29.1514dB	24.8499dB	24.1019dB
RMSE	6.50218	9.77333	13.7966	24.6975	30.9466

6.4 Análisis comparativo de las estimaciones

En la Tabla 6.8 se muestran los resultados obtenidos al medir el índice PSNR arrojado por cada uno de los filtros para los diferentes niveles de ruido donde se puede observar que la eliminación del ruido ha conseguido mejores resultados utilizando la combinación BM3D-RNCA que con los 4 Filtro aplicados individualmente.

Tabla 6.8: *PSNR obtenidos por filtros y por niveles de ruido.*

σ	con ruido	DWT	BM3D-Wiener	RNCA	BM3D-RNCA
20	25.91dB	34.94dB	35.84dB	36.01dB	37.11dB
40	20.23dB	30.46dB	31.79dB	32.11dB	33.97dB
60	17.12dB	26.55dB	28.79dB	29.15dB	31.09dB
80	15.08dB	21.17dB	23.94dB	24.84dB	25.55dB
100	13.62dB	19.60dB	22.14dB	24.10dB	25.40dB

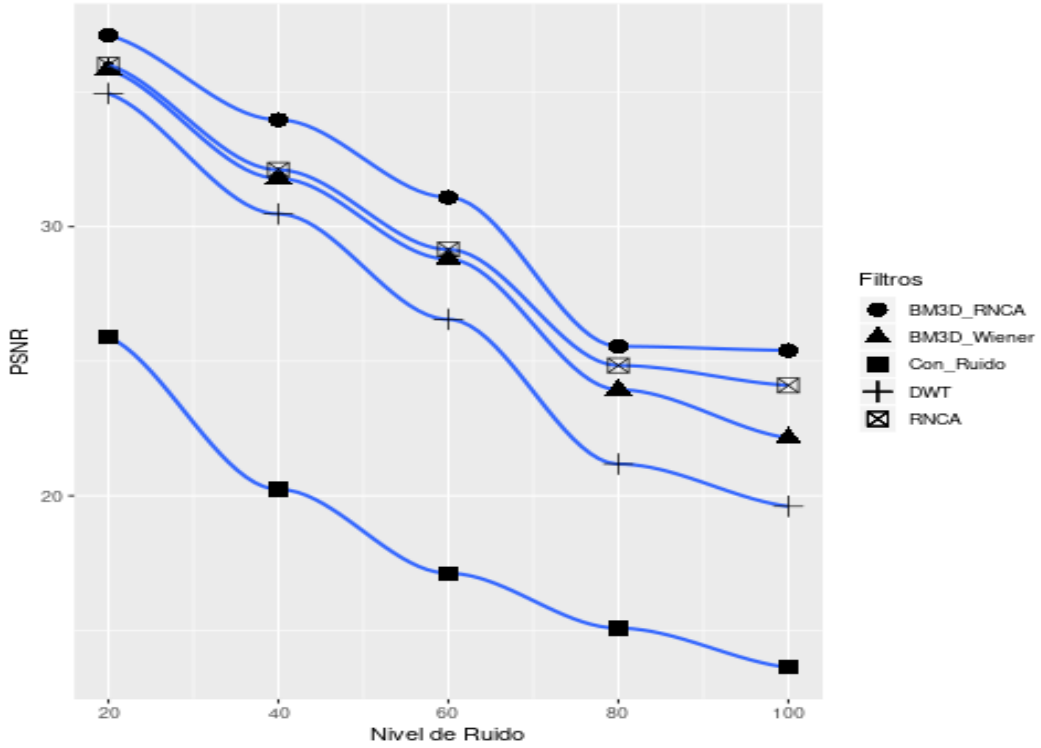
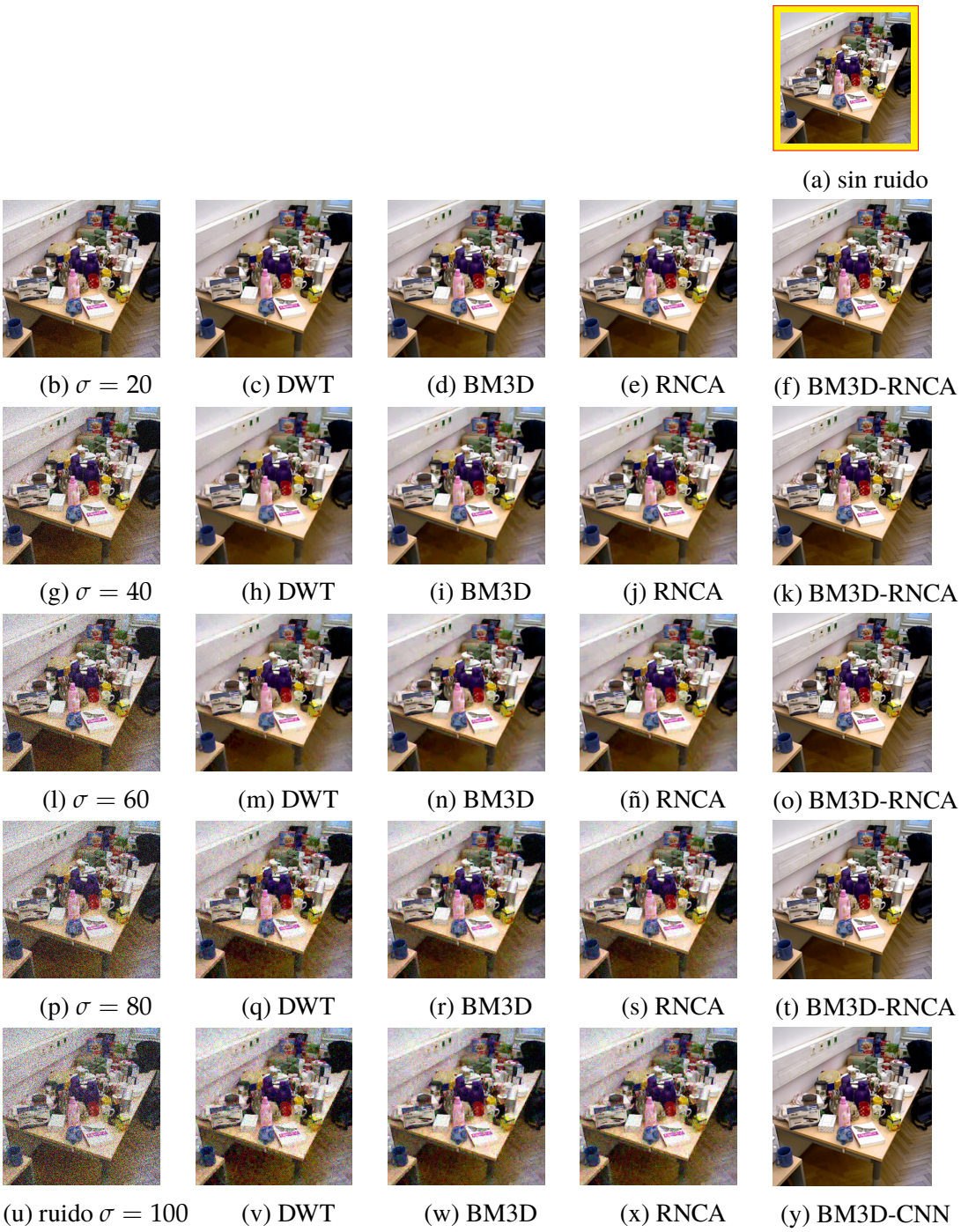


Figura 6.6: PSNR de los 5 filtros aplicados a la imagen objeto.

Los resultados también son visibles en las imágenes obtenidas.



(a) sin ruido

(b) $\sigma = 20$ (c) DWT (d) BM3D (e) RNCA (f) BM3D-RNCA

(g) $\sigma = 40$ (h) DWT (i) BM3D (j) RNCA (k) BM3D-RNCA

(l) $\sigma = 60$ (m) DWT (n) BM3D (ñ) RNCA (o) BM3D-RNCA

(p) $\sigma = 80$ (q) DWT (r) BM3D (s) RNCA (t) BM3D-RNCA

(u) ruido $\sigma = 100$ (v) DWT (w) BM3D (x) RNCA (y) BM3D-CNN

Figura 6.7: Resultados visuales de los filtros de eliminación de ruido para los diferentes niveles de ruido dado por el valor del σ.

Fue necesario comparar los resultados de la combinación BM3D-RNCA con los resultados del algoritmo RNCA ejecutado de forma individual, esto se dio suministrando la imagen con ruido IO sin haber sido previamente estimada por el algoritmo BM3D;

para los dos modelos, se mantuvo los valores constantes del parámetros batch en 2, y se jugo con los valores del parámetro epoch (entre 800 y 3000) para que las perdidas sean muy parecidas (≈ 0.5) a las obtenidas en el BM3D-RNCA, de esta manera se registro los parámetros PSNR arrojados para los diferentes niveles de ruido, y el tiempo tomado para entrenar y validar como se muestra en la siguiente Tabla 6.9.

Tabla 6.9: *Parámetros epoch y batch obtenidos en los diferentes niveles de ruido por la RNCA y la BM3D-RNCA.*

σ	RNCA				BM3D-RNCA				perdida
	epoch	batch	tiempo	PSNR	epoch	batch	tiempo	PSNR	
20	800	2	2h	36.01dB	400	2	1h	37.11dB	≈0.531
40	1000	2	3h	32.11dB	600	2	1.5h	33.97dB	≈0.529
60	1500	2	4h	29.15dB	800	2	2h	31.09dB	≈ 0.530
80	2000	2	5h	24.84dB	1000	2	3h	25.55dB	≈0.532
100	3000	2	7.5h	24.10dB	2000	2	5h	25.40dB	≈0.527

Los valores de *epoch* que se muestra en la Tabla 6.9 para cada nivel de ruido son los máximos posibles para obtener una pérdida del 0.5 después de estos valores no se obtienen valores más bajos en las perdidas, ya que con el entrenamiento brindado no es posible obtener mejores resultados.

CAPÍTULO 7

Conclusiones

En esta investigación el nuevo modelo de filtro que nace de la combinación **BM3D-RNCA**, arrojó mejoras en la eliminación del ruido y recuperación de una imagen, destacándose aquellos resultados para altos niveles de afectación con ruido blanco Gaussiano Aditivo (σ>40), donde para $\sigma = 100$ la combinación BM3D-RNCA obtuvo un PSNR=25.40 db mayor al obtenido al utilizar solo el algoritmo BM3D que obtuvo un PSNR=22.14 y también al obtenido por el algoritmo RNCA que obtuvo un PSNR=24.10..

Cinco son las características más importantes que se personalizaron para que el filtro de la red neuronal convolucional permita obtener los resultados deseados en esta investigación: 1) la topología autoencoder; 2) el número de capas implementadas para este propósito, las cuales fueron 14, distribuidas 7 en su etapa encoder, 2 en la etapa de mapeo y 5 en su etapa de decoder; 3) la cantidad de parámetros en las 15 capas donde se utilizaron 4,385 parámetro; 4) el tamaño de las imágenes tanto de entrenamiento, prueba y predicción fueron de 512x512; 5) el máximo valor del parámetro epoch fue de 3000, el número de lotes (batch) por cada epoch fue de 2.

En el experimento para el aprendizaje de la RNCA se utilizó **36** imágenes, de las cuales 30 fueron para el proceso de entrenamiento y 6 para el proceso de prueba y se crearon 5 escenarios distintos, cada escenario correspondió a un nivel de ruido Aditivo Blanco Gaussiano distinto ($\sigma = 20$, $\sigma = 40$, $\sigma = 60$, $\sigma = 80$,$\sigma = 100$), para cada nivel de ruido se crearon datos de entrada y datos de salida (*label*), los datos de entrada fueron la imágenes estimadas por el algoritmo BM3D y los label fueron las imágenes originales libre de ruido, en cada escenario se logró demostrar numéricamente con el indicador PSNR la ventaja que tuvo sumar un nivel más de procesamiento (RNCA) al ya exitoso filtro BM3D, más que todo en los niveles más altos de ruido ($\sigma > 40$), sacrificando tiempo y recursos.

Las pocas 36 imágenes con que se entrenó la RNCA arrojo perdidas muy altas de $\approx$ 0.5/1 obteniendo un nivel de precisión muy bajo 0.0078/1.

Finalmente, se pudo observar que aplicar la red neuronal luego de la estimación del BM3D, ayuda a reconstruir mejor la imagen, que si se aplica la red neuronal directamente a la imagen objeto (IO), la mejora se nota en cuanto al tiempo de procesamiento como en la calidad de los resultados.

Trabajo futuro

Una vez que se cuenta con los *shell script* que pemitieron preparar las imagenes para entrenar y validar la red neuronal, el siguiente trabajo consistirá en crear un set más grande de imágenes y experimentar con diferentes funciones de *optimización* y de *perdida* a fin de hallar parámetros que se adapten mejor a la arquitectura creada.

APÉNDICE A

Anexo I: Componentes del algoritmo BM3D

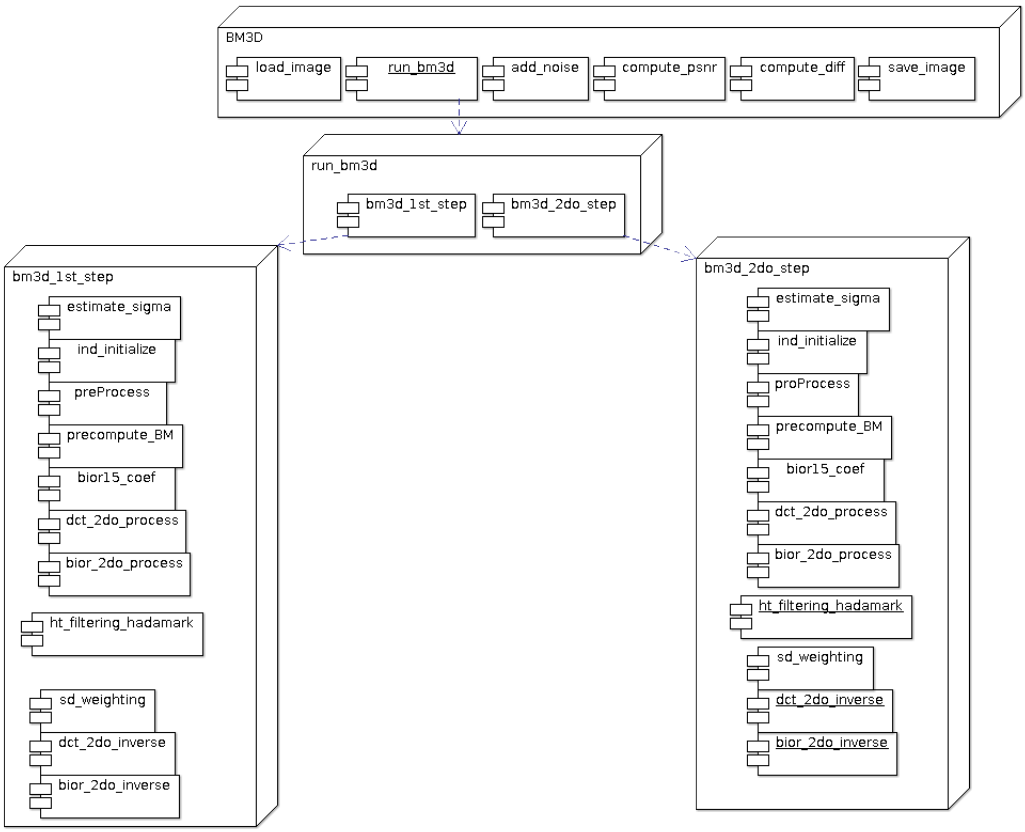


Figura A.1: Componentes del algoritmo BM3D

APÉNDICE B

Anexo II: Abreviaturas

Termino	Significado	Descripción
RNCA	Red Neuronal Convolucional Autoencoder	Algoritmo para la Red Neuronal Convolucional Autoencoder.
MSE	Error cuadrático medio (Mean Square Error)	
PSNR	Peak Signal Noise Ratio	
BM3D	Block Matching 3D	
IO	Imagen Objeto	La imagen con ruido que será estimada tanto por el algoritmo BM3D como por el algoritmo RNCA.
$\widehat{IO}$	Imagen Objeto Estimada	La imagen estimada después de haber sido procesada por el algoritmo BM3D.
Wiener	Wiener-Kolmogorov	Este filtro utiliza métodos estadísticos para eliminar el ruido de una señal.
RBGA	Ruido Blanco Gaussiano Aditivo	Ruido aplicado a la imagen que fue utilizada en la investigación, en ingles su nombre es Additive White Gaussian Noise (AWGN).
DCI	Dispositivos de captura de imagenes	Dispositivos tales como la cámaras fotográficas, smartphone, filmadoras, etc.
σ	Desviación estandar	Parámetro para el ruido blando Gaussiano aditivo

Tabla B.2: Abreviaturas utilizadas en el documento

Bibliografía

[1] Aguilera, C. A., Aguilera, F. J., Sappa, A. D., Aguilera, C., and Toledo, R. Learning cross-spectral similarity measures with deep convolutional neural networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)* (June 2016), pp. 267–275.

[2] Bodington, D. Learned convolutional denoising. *Stanford University* (2015).

[3] Buades, A., Coll, B., and Morel, J.-M. A review of image denoising algorithms, with a new. *A SIAM Interdisciplinary Journal, Society for Industrial and Applied Mathematics* (2010).

[4] BurganErgen. Signal and image nosing using wavelet transform, Abril 2012.

[5] Dabov, K., Foi, A., Katkovnik, V., and Egiazarian, K. Image denoising with block-matching and 3d filtering. *Proc. SPIE 6064* (2006), 606414–606414–12.

[6] Dabov, K., Foi, A., Katkovnik, V., and Egiazarian, K. Image denoising by sparse 3d transform-domain collaborative filtering. *IEEE TRANS. IMAGE PROCESS 16*, 8 (2007), 2080.

[7] Dabov, K., Foi, A., Katkovnik, V., and Egiazarian, K. BM3D Image Denoising with Shape-Adaptive Principal Component Analysis. In *SPARS'09 - Signal Processing with Adaptive Sparse Structured Representations* (Saint Malo, France, Apr. 2009), R. Gribonval, Ed., Inria Rennes - Bretagne Atlantique.

[8] Delon, J., and Houdard, A. Gaussian priors for image denoising. In *Denoising of Phothographic Images and Video: Fundamentals, Open Challenges and New Trends*, HAL, Ed. HAL, Mayo 2018.

[9] Donoho, D. L., and Johntone, L. M. Adaption to unknows via wavelet shrinkage. *American Statistical Asociation* (1995).

[10] Duhamel, P. . V. M. *Fast Fourier Transforms: A Tutorial Review and a State of the Art*. Digital Signal Processing Handbook, 1999, ch. 7.

[11] Dwyer, H. Deep learning with dataiku data science studio. `https://blog.dataiku.com/deep-learning-with-dss`, Agosto 2015.

[12] Goodfellow, I., Bengio, Y., and Courville, A. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[13] Hadsell, R., Chopra, S., and LeCun, Y. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)* (June 2006), vol. 2, pp. 1735–1742.

[14] Hasan, M. M. Adaptive edge-guided block-matching and 3d filtering (bm3d) image denoising algorithm. *The University of Western Ontario* (2014).

[15] Illezca, P. L. Transformada rápida de fourier fft. *Universidad Técnica Federico Santa María* (2005).

[16] An analysis and implementation of the bm3d image denoising method. `http://www.ipol.im/pub/art/2012/l-bm3d/`, 2012. Algoritmo BM3D.

[17] Jin, F., and Fieguth, P. Adaptive wiener filtering of noisy images and image sequences. *IEEE* (2003).

[18] Joy, J., Peter, S., and John, N. Denoising using soft thresholding. *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering 2*, 3 (March 2013), 6.

[19] Kaur, S. Noise types and various removal techniques. *International Journal of Advanced Research in Electronics and Communication Engineering* (2015).

[20] Lebrum, M. An analysis and implementation of the bm3d image denoising method. *Image Processing On Line* (2012).

[21] Lebrun, M. Bm3d image denoising. `https://github.com/gfacciol/bm3d/`, Septiembre 2011. Sitio github donde se puede encontrar la codificación en c++ del algoritmo BM3D.

[22] Lee, K., and S.Lee, J. L. Brightness-based convolutional neural network for thermal image enhancement. *IEEE Access* (2017).

[23] nyanp. convert image files to mnist idx format. `https://github.com/nyanp/mnisten/`, Enero 2016. Este es el código en c++ para llevar un conjunto de imágenes al formato de base de datos mnisten.

[24] Rivadeneira, R. E., Suarez, P. L., Sappa, A. D., and Vintimilla, B. X. Thermal image superresolution through deep convolutional neutral network. *International Conference on Image Analysis and Recognition ICIAR 2019* (2019).

[25] Singh, P., and Jindal, S. A comparative study of dct and dwt-spiht. *Internatioanl Journal of Computational Engineering and Management 15*, 2 (March 2012).

[26] Sridhar, S., Kumar, P. R., and K.V.Ramanaiah. Wavelet transform techniques for image compression - an evaluation. *I.J. Image, Graphics and Signal Processing* (2014).

[27] Suarez, P., Sappa, A., and Vintimilla, B. Cross-spectral image patch similarity using convolutional neural network. In *2016 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)* (05 2017), pp. 1–5.

[28] Suzuki, S., Tsurusaki, N., and Kodama, Y. Distribution of an endangered burrowing spider lycosa ishikarianain the san'in coast of honshu, japan (araneae: Lycosidae). *Acta Arachnologica 55*, 2 (2006), 79–86.

[29] team, G. B. Tensorflow. `https://www.tensorflow.org`, octubre 2018. Sitio Web oficial de tensorflow y keras.

[30] Timofte, H. H. R., and Gool, L. Make my day - high-fidelity color denoising with near-infrared. *IEEE* (2015).

[31] Tuw object instance recognition dataset. `https://repo.acin.tuwien.ac.at/tmp/permanent/dataset-index.php`, 2013. TUW Dataset,imagenes indoor.

[32] Wang, Y.-Q. Small Neural Networks can Denoise Image Textures Well: a Useful Complement to BM3D. *Image Processing On Line 6* (2016), 1–7.

[33] Wikipedia. Psnr. `https://es.wikipedia.org/wiki/PSNR`, 2017.

[34] Ziler, M. D., and Fergus, R. Visualizing and understanding convolutional network. *ArXiv:1311.2901v3* (2013).