

# Eliminación de ruido en imágenes con wavelets.

Análisis de señales

Grupo E: Alejandra Venegas, Rebeca Company, Marta Medina, Alejandro Cornelio y Ilia Zhigarev.

2024-12-29

## Índice

<b>1</b>	<b>Introducción</b>	<b>1</b>
<b>2</b>	<b>Fundamento teórico: eliminación de ruido con wavelets</b>	<b>1</b>
<b>3</b>	<b>Funciones de programación empleadas</b>	<b>2</b>
<b>4</b>	<b>Desarrollo y resultados</b>	<b>2</b>
4.1	Inclusión de ruido sintético en las imágenes . . . . .	3
4.2	Función imwd . . . . .	5
4.3	Función denoise... . . . .	5
<b>5</b>	<b>Conclusiones</b>	<b>5</b>

## 1 Introducción

En el campo del análisis de señales, uno de los problemas más relevantes es la eliminación de ruido en imágenes digitales. Las señales, que representan variaciones de magnitudes en el espacio y/o tiempo, a menudo están contaminadas por ruido debido a interferencias en los procesos de captura o transmisión. Este ruido, que puede existir de formas muy diversas como ruido gaussiano, artefactos asociados a frecuencias altas o bajas, o efectos multiplicativos, complica la extracción de características significativas.

El uso de wavelets es una herramienta clave para abordar este problema, el cual permite descomponer señales en sus componentes espaciales y frecuenciales de forma eficiente. Este método ofrece la posibilidad de adaptarse a las características particulares de cada tipo de ruido.

En este trabajo se propone explorar la capacidad de los wavelets para la eliminación de ruido en imágenes digitales. Para ello, se generarán y analizarán diferentes tipos de ruido artificial, evaluando su complejidad para su atenuación y determinando los parámetros más eficaces de los wavelets.

## 2 Fundamento teórico: eliminación de ruido con wavelets

Los wavelets son funciones matemáticas que permiten descomponer una señal en componentes de diferentes escalas, lo que resulta útil para identificar y procesar características específicas. Este enfoque es especialmente

relevante en la eliminación de ruido, donde las frecuencias indeseadas pueden ser separadas y reducidas sin afectar significativamente las características principales de la señal original. A diferencia de la Transformada de Fourier, que opera globalmente y no ofrece información sobre la localización temporal de los eventos, los wavelets permiten un análisis localizado, facilitando la identificación de patrones y anomalías en los datos.

El proceso de reducción de ruido con wavelets generalmente incluye tres etapas principales: la descomposición de la señal utilizando una wavelet madre, la modificación de los coeficientes wavelet mediante técnicas de umbral, y la reconstrucción de la señal. La selección de la wavelet madre adecuada y los parámetros de umbral son aspectos críticos que deben adaptarse a las características específicas del ruido y la señal.

Además, los wavelets ofrecen un enfoque multi-resolución, permitiendo una representación detallada de los componentes de alta frecuencia, asociados frecuentemente con el ruido, mientras preservan las estructuras globales de baja frecuencia. Esta característica hace que los wavelets sean especialmente útiles en aplicaciones donde la precisión y la integridad de los datos son esenciales, como en imágenes médicas, procesamiento de audio o análisis de datos científicos.

En este trabajo, se emplearán wavelets como herramienta principal para eliminar diferentes tipos de ruido sintético en imágenes.

### 3 Funciones de programacion empleadas

### 4 Desarrollo y resultados

Antes de comenzar, instalamos y/o cargamos todos los paquetes requeridos.

```
if (!require("pacman")) install.packages("pacman")
library(pacman)
p_load(imager, wavethresh, ggplot2, dplyr, SpatialPack, waveslim, EBImage, stringr, jpeg)
```

Comenzamos cargando y visualizando las fotografías a emplear.

```
images_path <- list.files("./fotos", full.names = TRUE)

nombres_images <- str_remove_all(string = str_remove_all(string = images_path, pattern = "./fotos/"), pattern = ".jpg")

images <- lapply(images_path, readJPEG) # Cargamos las imágenes

# Nota: cambie de load.image a readJPEG pq asi ya no hace falta usar el drop para quitar lo de cimg, save
names(images) <- nombres_images

# Rotamos algunas de las fotos para una visualización más uniforme
fotos_a_girar <- c("1", "2", "3", "4")
images_rotadas <- lapply(images[fotos_a_girar], aperm, perm = c(2, 1, 3))

for (i in fotos_a_girar) {
  images_rotadas[[i]] <- images_rotadas[[i]][dim(images_rotadas[[i]])[1]:1, , ]
}

images[fotos_a_girar] <- images_rotadas
rm(images_rotadas)
rm(fotos_a_girar)
```

```

# # Visualizamos las imagenes originales
#
# par(mfrow = c(2, 3), mar = c(1, 1, 1, 1))
#
# for (img in nombres_imagenes) {
#   display(Image(images[[img]]), colormode = "Color", method = "r")
# }

```

## 4.1 Inclusión de ruido sintético en las imágenes

```

# Definición de tipos de ruido

NOISE_TYPES <- list(
  gaussian = list(
    generator = function(channel, params) {
      # Desviación estándar del ruido con un valor predeterminado
      noise_std_dev <- params$std_dev %||% 0.5

      # Generación de ruido gaussiano
      noise <- array(
        rnorm(length(channel), mean = 0, sd = noise_std_dev),
        dim = dim(channel)
      )

      # Asegurar que los valores estén entre 0 y 1
      pmax(0, pmin(1, channel + noise))
    }
  ),
  sinusoidal_high = list(
    generator = function(channel, params) {
      # Frecuencia y amplitud del ruido sinusoidal de alta frecuencia
      frequency <- params$frequency %||% 25
      amplitude <- params$amplitude %||% 0.2

      # Generación de ruido sinusoidal
      height <- dim(channel)[1]
      width <- dim(channel)[2]
      x <- seq(0, 2 * pi, length.out = width)
      y <- seq(0, 2 * pi, length.out = height)
      noise_grid <- outer(sin(x * frequency), sin(y * frequency))

      # Aplicar el ruido
      noise <- array(noise_grid * amplitude, dim = dim(channel))
      pmax(0, pmin(1, channel + noise))
    }
  ),
  sinusoidal_low = list(
    generator = function(channel, params) {
      # Frecuencia y amplitud del ruido sinusoidal de baja frecuencia
      frequency <- params$frequency %||% 2
      amplitude <- params$amplitude %||% 0.2
    }
  )
)

```

```

# Generación de ruido sinusoidal
height <- dim(channel)[1]
width <- dim(channel)[2]
x <- seq(0, 2 * pi, length.out = width)
y <- seq(0, 2 * pi, length.out = height)
noise_grid <- outer(sin(x * frequency), sin(y * frequency))

# Aplicar el ruido
noise <- array(noise_grid * amplitud, dim = dim(channel))
pmax(0, pmin(1, channel + noise))
}
),
salt_pepper = list(
  generator = function(channel, params) {
    # Proporción de píxeles afectados por el ruido de sal y pimienta
    epsilon <- params$epsilon %||% 0.2

    # Generación de ruido
    noise <- matrix(sample(c(0, 1, NA), length(channel), replace = TRUE, prob = c(epsilon / 2, epsilon / 2, epsilon / 2)),
      nrow = dim(channel)[1], ncol = dim(channel)[2]
    )
    channel[!is.na(noise)] <- noise[!is.na(noise)]
    channel
  }
),
gamma = list(
  generator = function(channel, params) {
    # Ruido multiplicativo gamma con parámetro de dispersión
    looks <- params$looks %||% 2
    noise <- array(rgamma(length(channel), shape = looks, scale = 1 / looks), dim = dim(channel))
    pmax(0, pmin(1, channel * noise))
  }
),
uniform_multiplicative = list(
  generator = function(channel, params) {
    # Ruido multiplicativo uniforme
    looks <- params$looks %||% 2
    noise_channel <- SpatialPack::imnoise(
      img = channel,
      type = "speckle",
      looks = looks
    )
    pmax(0, pmin(1, noise_channel))
  }
)
)

# Función para añadir ruido a una imagen
add_noise_to_image <- function(image_name, noise_type, noise_params = list(), plot = FALSE) {
  # Verificar si la imagen existe en la lista
  if (!image_name %in% names(images)) {
    stop("La imagen con este nombre no se encuentra en la lista 'images'")
  }
}

```

```

# Verificar el tipo de ruido
if (!noise_type %in% names(NOISE_TYPES)) {
  stop(
    "El tipo de ruido es desconocido. Tipos disponibles: ",
    paste(names(NOISE_TYPES), collapse = ", ")
  )
}

# Obtener la imagen original de la lista
original_image <- images[[image_name]]

# Convertir la imagen a un array si es necesario
image_array <- as.array(original_image)

# Aplicar ruido a cada canal
noisy_channels <- lapply(1:3, function(i) {
  channel <- image_array[, , i]
  NOISE_TYPES[[noise_type]]$generator(channel, noise_params)
})

# Crear la imagen con ruido
noisy_image_array <- array(
  unlist(noisy_channels),
  dim = dim(image_array)
)

# Visualizar si se ha indicado
if (plot == TRUE){
  layout(matrix(1:2, 1, 2))
  plot(Image(original_image, colormode = "Color"))
  title("Original")
  plot(Image(noisy_image_array, colormode = "Color"))
  title(paste("Ruido:", noise_type))}

return(noisy_image_array)
}

# Aplicar los diferentes tipos de ruido a cada imagen
# add_noise_to_image("1", "gaussian", list(std_dev = 0.3))
# add_noise_to_image("2", "sinusoidal_high", list(frequency = 25, amplitude = 0.2))
# add_noise_to_image("3", "sinusoidal_low", list(frequency = 2, amplitude = 0.2))
# add_noise_to_image("4", "salt_pepper", list(epsilon = 0.1))
# add_noise_to_image("5", "gamma", list(looks = 2))
# add_noise_to_image("5", "uniform_multiplicative", list(looks = 2))

```

## 4.2 Función imwd

## 4.3 Función denoise...

# 5 Conclusiones