

# Compresión de imágenes y vídeos vía wavelets

Análisis de señales. Máster en Ciencia de Datos.

Adrián Lara, Jesús Martínez, Miguel Muñoz, Samuel Ortega, Álex Riera y Pablo Vicente

2023-11-24

## Contents

|   |           |
|---|-----------|
| <b>1 Abstract</b>   | <b>1</b>  |
| <b>2 Contenidos teóricos</b>  | <b>1</b>  |
| <b>3 Procedimiento práctico</b>   | <b>3</b>  |
| 3.1 Búsqueda de referencias o trabajos similares . . . . .                        | 3         |
| 3.2 Compresión de imágenes en R . . . . .   | 4         |
| 3.3 Compresión de imágenes en Python . . . . .                                    | 5         |
| 3.3.1 Compresión manual de imágenes BW single-level . . . . .                     | 6         |
| 3.3.2 Compresión de imágenes RGB single-level con PyWavelets . . . . .            | 6         |
| 3.3.2.1 Función diseñada para el análisis y su aplicación . . . . .               | 6         |
| 3.3.2.2 Comparación entre distintos wavelets, threshold values y modos . . . . .  | 8         |
| 3.3.2.3 Búsqueda de un threshold adecuado para un porcentaje de reducción deseado | 9         |
| 3.3.3 Descomposición multivariante de imágenes . . . . .                          | 10        |
| 3.3.3.1 Nivel de descomposición máximo . . . . .                                  | 13        |
| 3.3.3.2 Métodos de comparación de la imagen original y la comprimida . . . . .    | 13        |
| 3.4 Compresión de vídeo utilizando Wavelets . . . . .                             | 15        |
| 3.4.1 Separar el vídeo en frames . . . . .  | 15        |
| 3.4.2 Aplicación de la transformada para cada frame . . . . .                     | 16        |
| 3.4.3 Volver a unir los frames para obtener el vídeo comprimido . . . . .         | 16        |
| <b>4 Producto final: Wavelet Compressor</b>                                       | <b>16</b> |
| <b>5 Conclusiones</b>   | <b>17</b> |

## 1 Abstract

En este proyecto se presenta un estudio sobre la compresión de imágenes y vídeo utilizando wavelets. Se comienza con una revisión teórica de los fundamentos de los wavelets y su aplicación en la compresión de imágenes. Posteriormente, se extiende el estudio a la compresión de vídeo, donde se analiza la aplicación de los wavelets en la compresión de secuencias de imágenes.

## 2 Contenidos teóricos

Como todo buen trabajo, tiene un motivo de interés de estudio detrás. En nuestro caso, nos centramos en la compresión de imágenes y vídeos.

Lo primero de todo, ¿qué es una imagen?

Una imagen puede considerarse una señal bidimensional en la que cada píxel representa valores de intensidad lumínica o de color en una ubicación específica. Es decir, cada punto de la imagen es un píxel que contiene información sobre su color y posición.

Antes de adentrarnos en el estudio de cómo comprimir vídeos, es fundamental comprender cómo se comprime una imagen. Esencialmente, se trata de representar la información visual de una manera más eficiente, reduciendo la cantidad de datos necesarios para almacenar o transmitir la imagen sin comprometer significativamente su calidad perceptible.

La compresión de imágenes juega un papel crucial en una amplia gama de aplicaciones, desde el almacenamiento eficiente de datos visuales hasta la transmisión rápida y efectiva de información gráfica en diversas plataformas y dispositivos.

En nuestro caso, la compresión de imágenes hará uso de las transformadas wavelet y sus diversas familias.

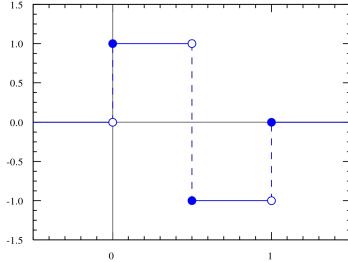
Una wavelet u ondícula es una pequeña onda cuya energía está concentrada en un período de tiempo limitado. Su forma es la de una onda característica que oscila, lo que permite realizar análisis temporales y frecuenciales. Esta herramienta resulta de gran utilidad en el estudio de fenómenos que varían en el tiempo. La wavelet viene definida por la siguiente ecuación:

$$W_\psi(s, \tau) = \frac{1}{\sqrt{s}} \int_{-\infty}^{\infty} x(t) \cdot \psi\left(\frac{t-\tau}{s}\right) dt$$

Los coeficientes  $W_\psi(s, \tau)$  representan los resultados obtenidos al aplicar la transformada. La señal a la que se le aplica esta transformada se denota como  $x(t)$ , y  $\psi\left(\frac{t-\tau}{s}\right)$  representa la wavelet madre. Esta wavelet madre puede tomar diversas formas según la familia de wavelets que se elija para la transformación.

Algunas de las familias de más interés son:

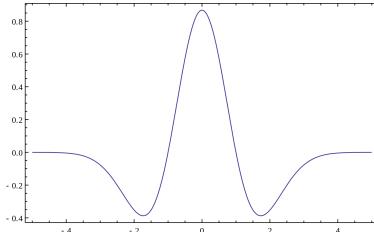
- Haar: La transformada de wavelets Haar es la primera que se propuso y es la más sencilla y simple de todas las familias que existen. Estas son funciones discontinuas y se parece a una función escalonada. Una visión de esta se observa en la Figura 1.



**Figure 1.** Wavelet de Haar

- Daubechies: Son una distinguida familia de wavelets ortogonales ampliamente empleadas en el procesamiento de señales y la compresión de datos. Las wavelets de Daubechies se destacan por su habilidad para lograr una compresión significativa de datos, conservando al mismo tiempo una representación precisa de la señal original. Esta capacidad de compresión es una de las características más relevantes y valiosas de las wavelets de Daubechies en diversas aplicaciones.
- Coiflet: Las wavelets Coiflet destacan por su notable propiedad de simetría. Una de sus características más sobresalientes es su habilidad para equilibrar la suavidad de una señal con la capacidad de detectar cambios abruptos en ella. En términos generales, estas wavelets comparten similitudes fundamentales con las Daubechies, con la distinción principal de su simetría, que es la marca distintiva de las Coiflet.
- Mexican Hat (Ricker): Este tipo de wavelet, también conocida como la wavelet de Ricker o “Mexican Hat”, se deriva de una función que es proporcional a la segunda derivada de una función de densidad

de probabilidad gaussiana. A diferencia de algunas otras wavelets, no posee una función de escala. Su singularidad radica en su habilidad para detectar y analizar cambios abruptos en una señal, así como para representar tanto componentes de alta frecuencia como de baja frecuencia con precisión y detalle. Una visión de esta se observa en la Figura 2.



**Figure 2.** Wavelet de Mexican Hat

- Symlets: Esta familia de wavelets guarda similitudes con las wavelets de Daubechies, compartiendo la capacidad de detectar cambios bruscos en señales. No obstante, las wavelets Symlets presentan una leve dosis adicional de asimetría. Esta diferencia en asimetría les confiere una mayor capacidad de representación en comparación con la familia Daubechies, especialmente al analizar ciertos tipos de señales.
- Gabor: Las wavelets Gabor, también conocidas como filtros Gabor, poseen la notable capacidad de capturar tanto las características de frecuencia en una señal como su localización precisa en el tiempo. Esta habilidad las convierte en herramientas sumamente útiles en la representación de texturas en imágenes, ya que tienen la capacidad única de detectar y representar las características de diversas escalas y orientaciones.
- Biortogonal: Esta familia de wavelets exhibe una notable propiedad: la fase lineal, esencial para la reconstrucción precisa de señales e imágenes. Lo particular de estas wavelets es su enfoque dual, utilizando dos wavelets, una para la descomposición y otra para la reconstrucción, en lugar de depender de una sola wavelet, como suele ser el caso en otras familias.

### 3 Procedimiento práctico

#### 3.1 Búsqueda de referencias o trabajos similares

Previo a la realización de cualquier actividad práctica, se cree conveniente realizar una pequeña revisión bibliográfica con el fin de poder tener una idea de las posibilidades que ofrecen los wavelets. En este rumbo, encontramos los siguientes trabajos:

- [Familias de wavelets en Matlab](#)
- [Introducción a los wavelets en Python](#)
- [Conceptos básicos de procesado de señales](#)
- [Paso de la FT a la WT](#)
- [Compresión de imágenes usando wavelets](#)
- [Esquemas descomposición transformada wavelets](#)

Una vez tenemos una idea preliminar sobre trabajo que vamos a realizar, las posibilidades dentro del marco de los wavelets y una serie de resultados, comenzamos a trabajar de forma práctica.

### 3.2 Compresión de imágenes en R

Como primer entorno de trabajo, escogimos RStudio, dado que la capacidad del software R para el procesado de señales es bastante elevada. Durante dos semanas estuvimos explorando los distintos paquetes en R que permitían realizar una transformada de wavelets, tanto discreta como continua, y haciendo pruebas con la imagen “Lena.png”, comúnmente utilizada en proyectos de tratamiento de imágenes.

```
library(imager) # Librería para importar imágenes y poder procesarlas
library(wavelets) # Librería para la transformación por wavelets

image_path <- "./data/Lena.png"
my_image <- load.image(image_path)

# Si la imagen está en RGB, la querremos convertir a escala de grises por simplicidad
#my_image <- grayscale(my_image)

# Aplicamos DWT
coeffs <- dwt.2d(my_image, wf = "haar")

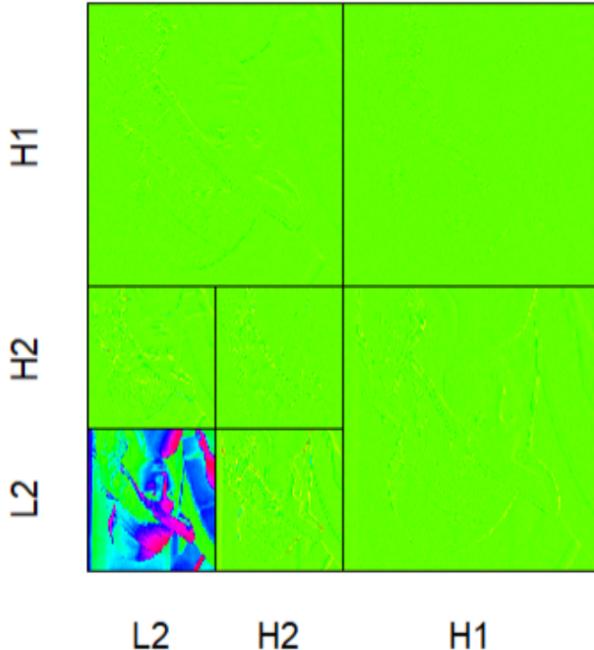
# Reconstruimos la imagen usando IDWT
reconstructed_image <- idwt.2d(coeffs)
compressed_image <- as.cimg(reconstructed_image)

# Guardamos la imagen
#save.image(compressed_image, "./data/compressed_image.jpg")

plot(my_image)
plot(compressed_image)
```



**Figure 3.** Comparación entre la imagen de Lena original y modificada en R en BW.



**Figure 4.** Visualización de los niveles de la imagen de Lena en R.

Como podemos observar, el código previo nos descompone la imagen en distintas capas, según el valor de los coeficientes de la imagen, y posteriormente reconstruye la imagen original. Este primer código nos permitió comprender el funcionamiento de la transformada de wavelets y cómo, al manipular las capas que conservamos de la descomposición, podemos influir en la compresión de la imagen (quitando detalle, suavizando, quitando ruido...). Pese a esto, no terminamos de hallar métodos para manipular las capas dentro de los objetos de tipo `dwt.2d`, que son los nativos de esta librería. Por ello, decidimos trasladar el proyecto a Python, ya que bajo nuestro punto de vista posee un mayor número de librerías con las que poder experimentar y trabajar.

### 3.3 Compresión de imágenes en Python

Pasamos todo el desarrollo práctico del proyecto a Python.

La compresión de imágenes por wavelets es un proceso que utiliza la transformada wavelet discreta 2D para descomponer una imagen en diferentes frecuencias y direcciones. La transformada wavelet discreta 2D se realiza utilizando una familia de wavelets, como `haar`, `db`, `sym` o `coif`. Los coeficientes resultantes se comprimen aplicando un determinado `threshold` con un `mode` particular, permitiendo entonces reconstruir la imagen comprimida. Para establecer un wavelet u otro se utilizará el parámetro `wavelet`, destacando `haar` y `db2`.

El proceso de compresión de imágenes por wavelets implica los siguientes pasos:

1. Cargar la imagen original.
2. Realizar una transformada wavelet discreta 2D en la imagen utilizando un wavelet específico.
3. Comprimir los coeficientes resultantes a través de un determinado `threshold` y con algún `mode`.
4. Reconstruir la imagen utilizando los coeficientes comprimidos.
5. Visualizar la imagen original y la imagen comprimida, comparando su tamaño.

La compresión de imágenes por wavelets puede ser utilizada para reducir el tamaño de las imágenes digitales sin perder demasiada información visual. Sin embargo, es importante tener en cuenta que la calidad de la imagen comprimida depende del valor del umbral utilizado y de la familia de wavelets seleccionada.

### 3.3.1 Compresión manual de imágenes BW single-level

En primer lugar, implementamos wavelets discretas definiendo las transformaciones directas e inversas como la convolución de las imágenes con los coeficientes correspondientes.

Aplicando la transformación directa sobre la imagen, obtenemos los coeficientes de aproximación y detalle. La compresión se realiza filtrando los coeficientes de detalle con un valor umbral, definido como un porcentaje del valor máximo en la imagen.



**Figure 5.** Compresión manual de Lena con Coiflet 1 y umbral = 0.1.

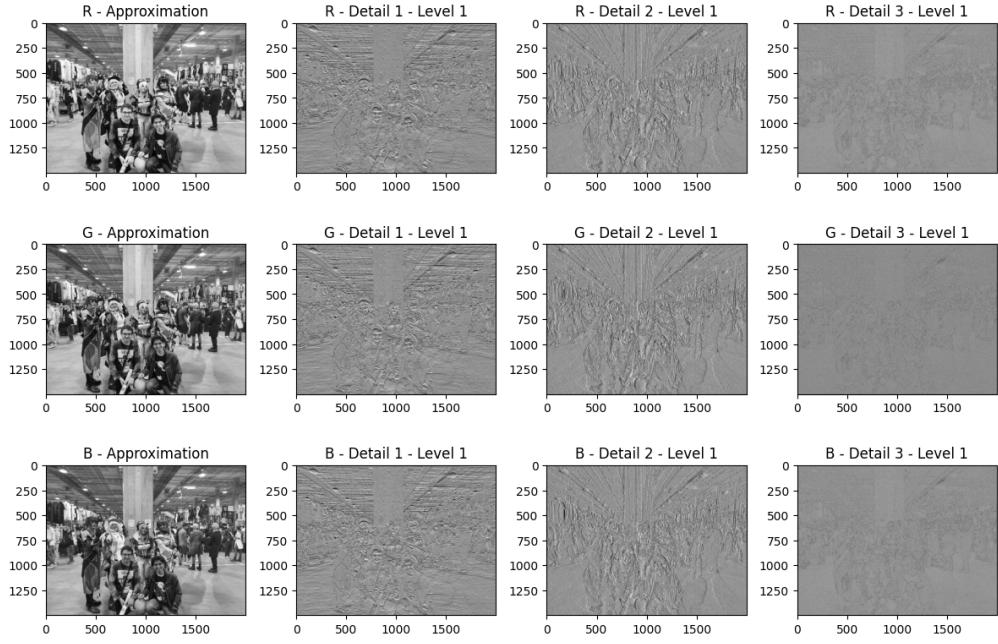
### 3.3.2 Compresión de imágenes RGB single-level con PyWavelets

Para la realización de la Transformada Discreta Wavelet 2D, así como de su inversa, lo más cómodo en Python es hacer uso de la librería [PyWavelets](#), la cual puede instalarse haciendo uso de `pip install PyWavelets`.

#### 3.3.2.1 Función diseñada para el análisis y su aplicación

Una función fue diseñada (ver [Memoria Wavelets.ipynb](#)) para realizar esta a nivel único (`single level`), de tal forma que se puede analizar rápidamente una imagen comprimida por la transformación wavelet para un determinado nivel de aproximación y detalle. Esto se hizo principalmente gracias a las funciones `wavedec2` e `idwt2` del módulo de [PyWavelets](#). Nuestra función `compress_image` a su vez permite trabajar tanto con imágenes en escala de grises como en RGB. Posee la capacidad de elegir entre los distintos tipos de `wavelet`, [modos de thresholding](#) o valor de `threshold`.

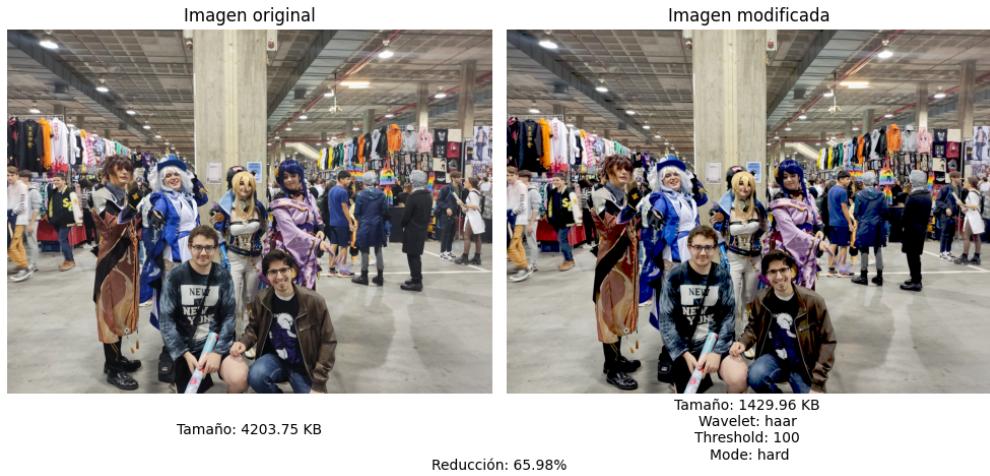
La aplicación de `compress_image` a una imagen (presente en `./data/genshin.jpg`) muestra lo siguiente:



**Figure 6.** Visualización de la aproximación y de los detalles horizontales, verticales y diagonales en una fotografía en RGB.

En la Figura 6 se aprecia una representación en escala de grises de lo que capturan nuestros coeficientes Wavelet para cada uno de los canales de la imagen: Red, Green y Blue. En este caso el análisis se hace para el primer nivel, mostrando los detalles horizontales, verticales y diagonales, en ese orden.

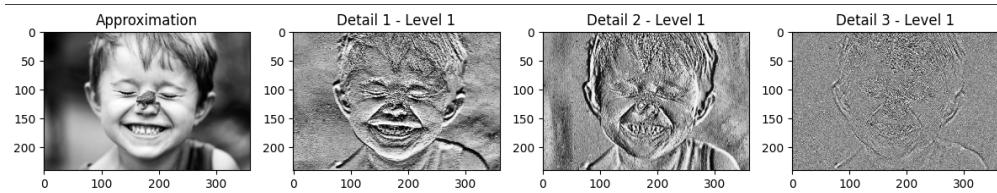
Por otra parte, se tiene una compración entre la imagen original y la comprimida.



**Figure 7.** Comparación entre la imagen original y modificada en el caso RGB.

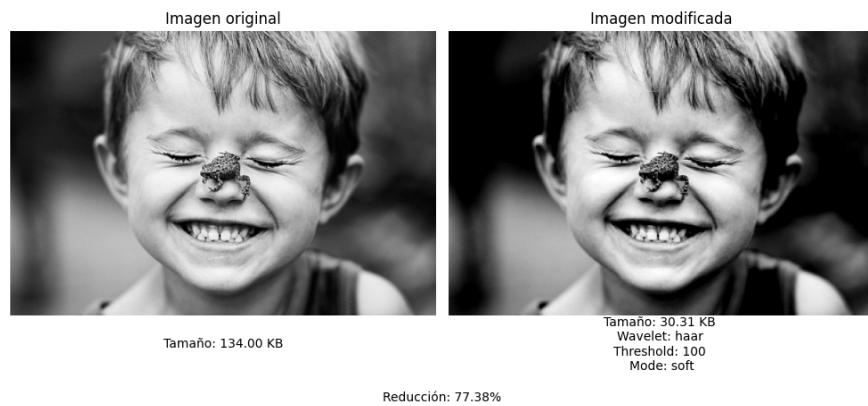
Apreciamos en la Figura 7 que nuestra compresión ha utilizado el Wavelet **haar**, un **threshold** de 100 y con un modo **hard**, obteniendo una reducción del 65.98 % del tamaño inicial.

Un ejemplo en blanco y negro también es posible realizarlo, usando la imagen presente en `./data/nino.jpg`.



**Figure 8.** Visualización de la aproximación y de los detalles horizontales, verticales y diagonales en una fotografía en escala de grises.

En la Figura 8 pueden apreciarse claramente los detalles según la dirección, resultando muy interesante a nivel teórico. La comparación entre la imagen original y comprimida se muestra a continuación.



**Figure 9.** Comparación entre imagen original y modificada en el caso de escala de grises.

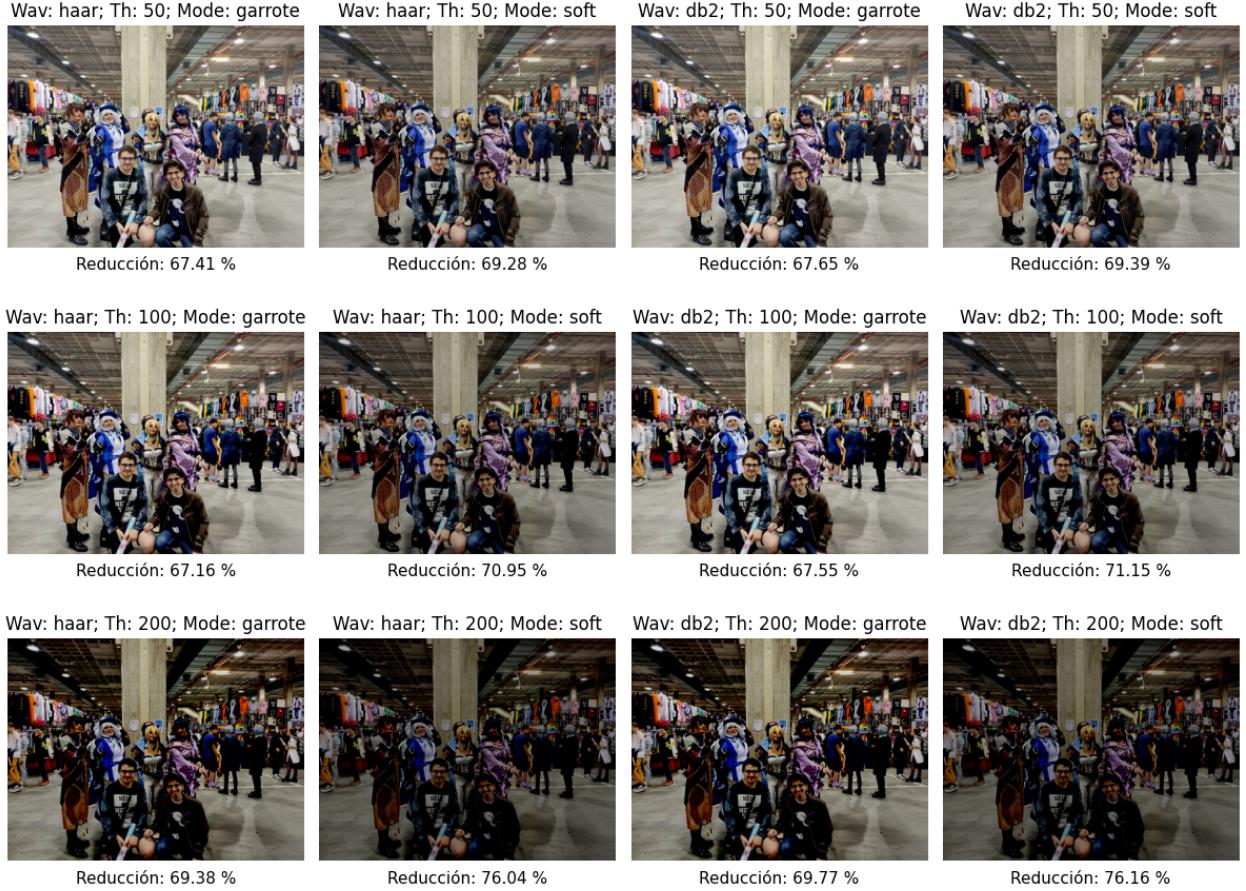
La reducción en tamaño en este caso es del 77.38 %. Nótese que se ha utilizado en este caso el modo de thresholding **soft**, lo que permite en la mayoría de los casos una reducción de ruido sin sacrificar demasiados detalles importantes, tal y como se aprecia en este caso.

**Nota:** los detalles han sido en todos los casos ecualizados utilizando la función `exposure.equalize_hist()` del módulo `exposure` de la biblioteca `scikit-learn`. Esto es muy útil pues permite el resalte de los detalles para que sean visibles al lector.

**Aplicación de la función:** en el código se ha añadido una forma de comprimir una carpeta completa de imágenes con la correspondiente creación de otra comprimida. ¡Esto permite al usuario reducir el tamaño de sus imágenes si no quiere usar un `.zip`!

### 3.3.2.2 Comparación entre distintos wavelets, threshold values y modos

Para hacer una comparación entre lo que se obtiene con distintos wavelets, valores de umbral y modos de aplicación de esos umbrales lo único que se tuvo que hacer fue iterar llamando a la función `compress_image` con los distintos valores deseados. Un ejemplo de ello se muestra en la Figura 10.

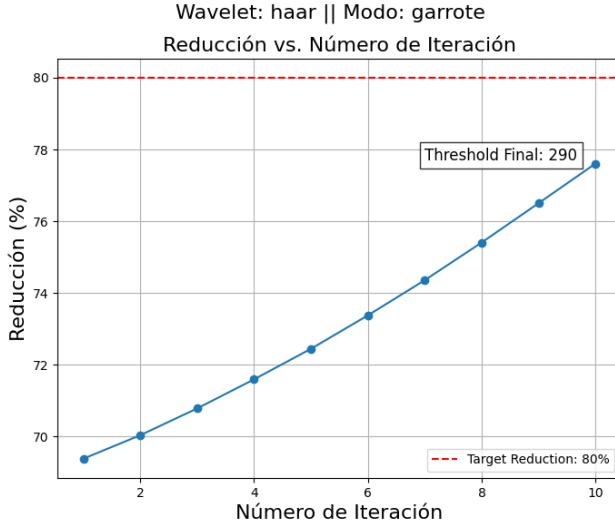


**Figure 10.** Comparación entre distintos wavelets, valores umbral y modos de threshold.

Puede observarse una cierta tendencia del modo **soft** a oscurecer la imagen en un intento de suavizar los coeficientes. El modo **garrote**, por otra parte, tiende a ser algo agresivo como **hard** para valores de coeficientes altos (los deja prácticamente igual). Sin embargo, este representa una transición entre **soft** y **hard** al comportarse de manera similar a **soft** para valores bajos. Es inmediato también apreciar cómo un aumento en el threshold acompaña un incremento en la reducción del tamaño de la imagen. De la misma forma, puede verse una ligera mayor compresión de la imagen para la Wavelet **db2** frente a la **haar**.

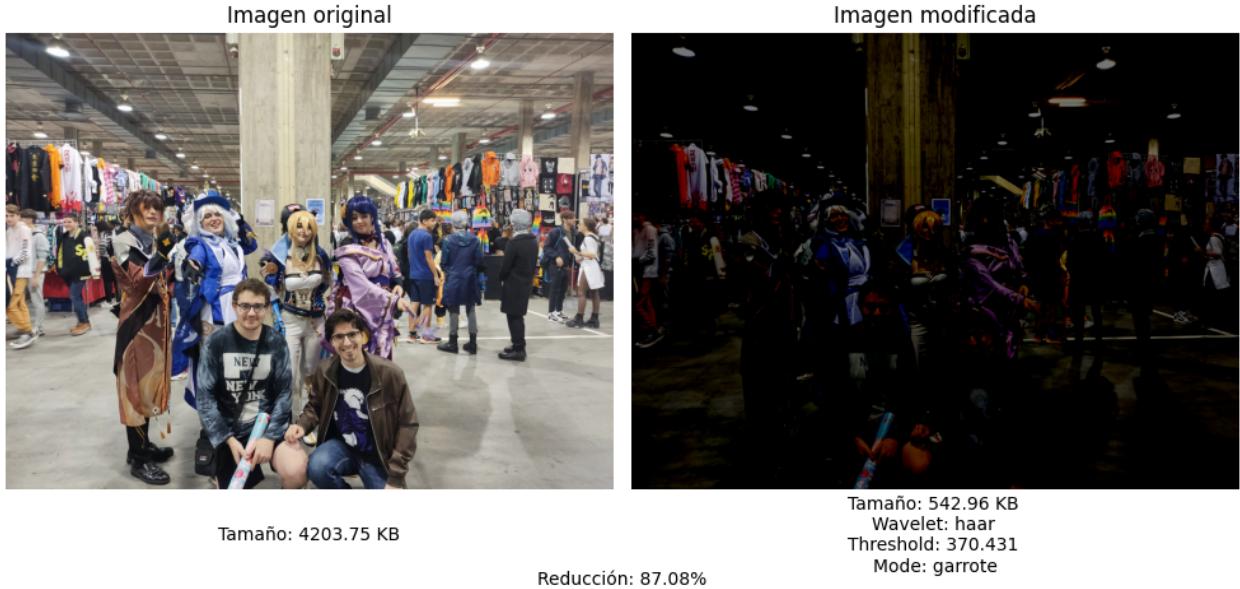
### 3.3.2.3 Búsqueda de un threshold adecuado para un porcentaje de reducción deseado

La función propia diseñada llamada `find_optimal_threshold` nos permite que dada una imagen, una compresión buscada con una familia y modo de thresholding fijo, obtener el threshold adecuado para un porcentaje de compresión deseado. Esta función básicamente itera de forma optimizada (con un paso adaptativo) sobre las distintas imágenes que se generarían para un threshold dado (ver código). El proceso iterativo para la imagen `"/data/genshin.jpg"` se muestra en la siguiente Figura.



**Figure 11.** Representación del proceso iterativo en la búsqueda de la reducción deseada.

Observamos que el *threshold final* que nos ofrece esta función sería 290. La razón por la que no se llega estrictamente a la reducción deseada es por una tolerancia implementada para evitar un bucle muy largo. La imagen con esta compresión se muestra a continuación.



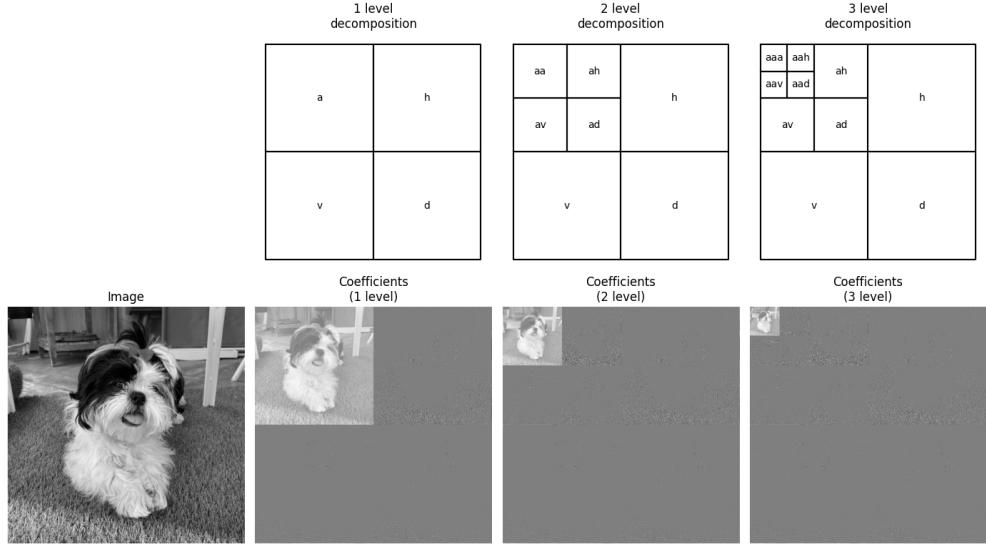
**Figure 12.** Imagen obtenida tras la búsqueda de la reducción deseada.

La calidad de la imagen obtenida no alcanza los estándares deseados. Esto subraya la variabilidad significativa asociada con el tema de la compresión, la cual depende en gran medida de las preferencias del usuario y de lo que este desee obtener.

### 3.3.3 Descomposición multinivel de imágenes

La librería PyWavelets incorpora numerosas funciones para aplicar transformaciones wavelet a señales n-dimensionales. En este trabajo, ya que nos centramos en imágenes (señales bidimensionales), empleamos

principalmente las funciones `dwt2` e `idwt2`, que nos permiten obtener la descomposición wavelet de nuestras imágenes y reconstruirlas a partir de los coeficientes de la transformación. Además de estas funciones, PyWavelets también incorpora las funciones `wavedec2` y `waverec2` para el análisis multirresolución de las imágenes aplicando el algoritmo de Mallat. En este algoritmo, un nuevo nivel de descomposición se logra al aplicar de nuevo la transformación wavelet sobre los coeficientes de aproximación el nivel anterior.



**Figure 13.** Representación esquemática de la descomposición de una imagen en distintos niveles aplicando el algoritmo de Mallat.

El algoritmo de Mallat para señales bidimensionales, igual que en el caso unidimensional, emplea bancos de filtros para producir la transformada wavelet discreta. Estos bancos de filtros son esencialmente filtros de pasa baja y de pasa alta, y pueden ser distintos (dependiendo del tipo de wavelet) en los procesos de descomposición y recomposición.

### Descomposición multinivel de una señal 2D

En la figura 14 se muestra un esquema del algoritmo de descomposición de una señal bidimensional mediante el método de Mallat. Partimos de la subseñal de aproximación del nivel anterior (que en el caso del nivel 0 se corresponde con la señal orginal) y la descomponemos en otras subseñales de aproximación y detalles.

- En primer lugar, las filas de la señal de aproximación del nivel anterior (en la primera iteración la imagen o señal bidimensional original) se convolucionan con los filtros de pasa alta y de pasa baja por separado, dando dos señales distintas.
- Posteriormente, las columnas de estas señales son submuestreadas por un factor dos, quedándonos solo con la mitad de las columnas.
- Después, a cada una de estas dos señales se les vuelven a aplicar los dos filtros de descomposición por separado, dando lugar a cuatro subseñales distintas.
- Cada una de estas subseñales es de nuevo submuestreada, esta vez en las filas, de modo que nos quedamos con la mitad de las filas. Estas cuatro subseñales constituyen ahora las señales de aproximación y detalles horizontal, vertical y diagonal del siguiente nivel. La señal de aproximación es aquella que ha sido convolucionada con el filtro de pasa baja dos veces, la subseñal de detalles horizontales primero recibe un filtro de pasa baja y luego un filtro de pasa alta, mientras que la de detalles verticales es al contrario. Por último, la subseñal de detalles diagonales es aquella que ha recibido dos veces el filtro de pasa alta en el proceso de su creación.

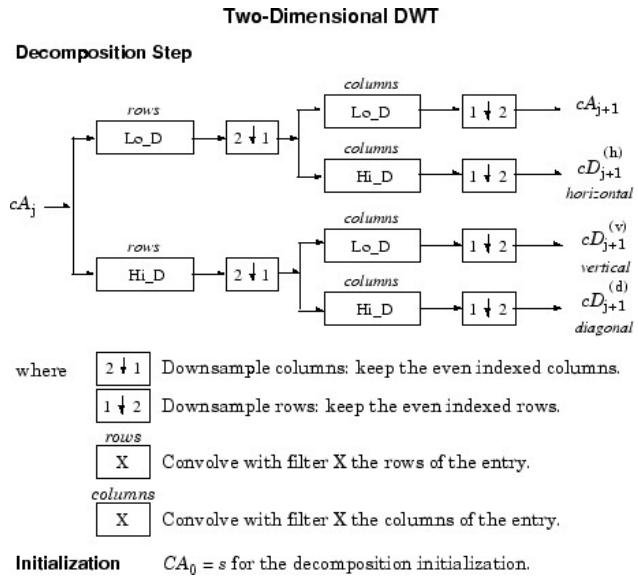
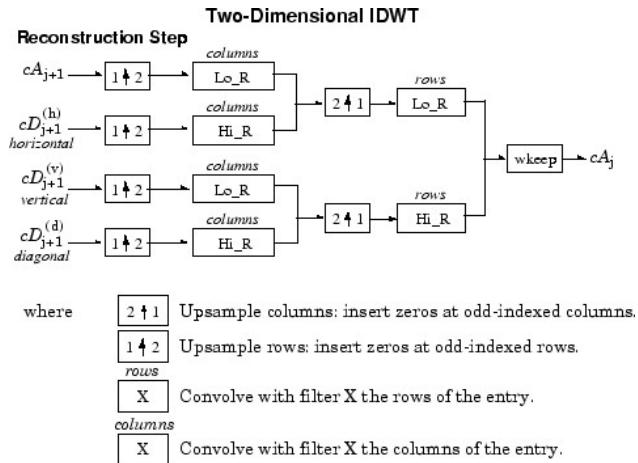


Figure 14. Esquema del algoritmo de Mallat para la descomposición multinivel de señales 2D.

### Recomposición de una señal 2D descompuesta en niveles

En la figura 15 se muestra un esquema del algoritmo de recomposición de una señal bidimensional mediante el método de Mallat. Partimos de las subseñales de aproximación y detalles de un nivel superior para recuperar a partir de ellas la subseñal de aproximación del nivel anterior (que en el nivel 0 se correspondería con la señal original)

- En primer lugar se aplica un muestreo ascendente a las filas de las subseñales de aproximación y detalle, añadiendo ceros a las columnas impares.
- Posteriormente, a las columnas de las señales obtenidas de las subseñales de aproximación y detalles verticales se les aplica el filtro de pasa baja de reconstrucción y a las columnas de las otras dos el filtro de pasa alta de reconstrucción.
- Las señales obtenidas de la subseñal de aproximación y de detalles horizontales se suman, así como las señales obtenidas a partir de los detalles verticales y diagonales, por lo que ahora contamos con dos señales distintas.
- A estas dos señales se les vuelve a aplicar un muestreo ascendente, esta vez sobre las columnas, insertando columnas de ceros en los índices impares.
- La señal obtenida de la suma de aproximaciones y detalles horizontales es convolucionada con el filtro de pasa baja de reconstrucción y la otra, obtenida a partir de los detalles verticales y diagonales, se convoluciona con el filtro de pasa alta de recuperación.
- Estas dos últimas señales obtenidas de la convolución se añaden para reconstruir la subseñal de aproximación del nivel anterior.



**Figure 15.** Esquema del algoritmo de Mallat para la recomposición multinivel de señales 2D.

### 3.3.3.1 Nivel de descomposición máximo

La función `dwt_max_level` de la librería PyWavelets devuelve el nivel máximo de descomposición en que podemos descomponer una imagen de un determinado tamaño (por ejemplo utilizando la función `wavedec2`) con una determinada familia de wavelets. Esta función es en realidad una generalización para señales n-dimensionales de la función `dwt_max_level`, la cual determina el nivel máximo para la descomposición de una señal unidimensional.

La función `dwt_max_level` calcula el nivel máximo de descomposición como el nivel donde hay al menos un coeficiente que no se ve afectado por los efectos de borde. Los efectos de borde son problemas que pueden surgir cuando se aplica una transformada wavelet en los bordes de una señal (en este caso la imagen) debido a la naturaleza finita de la señal y al hecho que las funciones wavelet requieren de un número mínimo de muestras para calcularse adecuadamente. También se puede interpretar este nivel máximo como si la descomposición parase cuando la señal se vuelve más pequeña que la FIR (finite impulse response) del filtro asociado a la wavelet. La expresión analítica de este nivel viene dada por:

$$max\_level = \log_2 \left[ \frac{data\_len}{filter\_len - 1} \right]$$

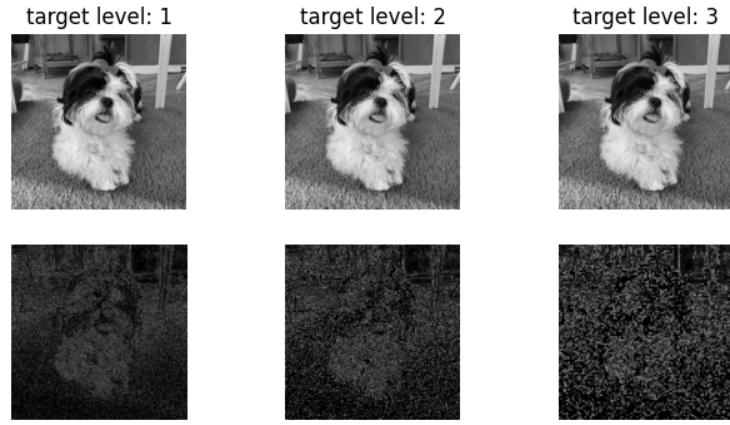
donde `data_len` y `filter_len` son los tamaños de la señal original y el filtro, respectivamente.

En el caso de señales n-dimensionales, el nivel máximo de descomposición es el mínimo de los niveles máximos calculados para todos los n ejes. Por ejemplo, el nivel de descomposición máximo para la imagen mostrada en la Figura 13 es 9.

En la Figura 16 podemos ver una imagen comprimida descartando el 50% de los coeficientes de detalle de los niveles 1, 2, 3. Vemos que cuando aproximamos a cero los coeficientes de detalle de un nivel más alto la diferencia de la imagen comprimida con la imagen original afecta a zonas distintas. Esto se debe a que en cada nivel superior los coeficientes de detalle se obtienen a partir de los coeficientes de aproximación del nivel anterior y se seleccionan diferentes bandas de frecuencias (diferentes detalles).

### 3.3.3.2 Métodos de comparación de la imagen original y la comprimida

Una vez reaizada la compresión, resulta siempre interesante comparar la imagen comprimida con la imagen original, ya que típicamente queremos mantener un equilibrio entre la compresión de la imagen y su calidad, de manera que se parezca lo máximo posible a la imagen original.



**Figure 16.** Imagen comprimida (fila superior) anulando la mitad de los coeficientes de detalle de distintos niveles de descomposición y su diferencia con la imagen original (fila inferior),

Como ya hemos comentado anteriormente, para comprimir la imagen por el método de las wavelet, dividimos la información de la imagen en subseñales de aproximación y detalle aplicando a la imagen una (o varias) transformaciones wavelet. Los coeficientes de aproximación proporcionan la tendencia general de los píxeles mientras que los coeficientes de detalle proporcionan los cambios o detalles en las direcciones horizontal, vertical y diagonal de la imagen. Para comprimir la imagen establecemos un umbral o “threshold”, que no es más que un valor de los coeficientes de detalle por debajo del cual los coeficientes son aproximados a cero. Cuanto mayor sea el número de ceros, mayor podrá ser la compresión alcanzada. Sin embargo, al hacer cero estos coeficientes se pierde cierta cantidad de información de la imagen original, que será mayor cuantos más coeficientes se aproximen a cero, por lo que se debe buscar un balance entre ambos. Por ello, para cuantificar la compresión de la imagen y su semejanza con la imagen original podemos emplear diversos métodos:

- **Ratio de compresión**

El ratio de compresión es el ratio entre el tamaño de la imagen original y la comprimida. Esre ratio es un indicador cuánto se ha comprimido una imagen en particular. Generalmente, cuanto mayor es el ratio de compresión peor es la calidad de la imagen.

- **Energía perdida**

En el contexto del procesado de señales, la energía de una imagen es proporcional a la suma del cuadrado de los valores sus píxeles y es una medida de la información presente en la imagen. Por tanto, la pérdida de información puede cuantificarse en forma de la pérdida de energía de la imagen en tanto por cien (energy loss).

- **Error cuadrático medio**

Otra manera de cuantificar la perdida de información o la diferencia entre la imagen original y la comprimida es mediante el error cuadrático medio, es decir, el error acumulado en cada pixel correspondiente a la imagen original y su análogo en la imagen comprimida al cuadrado.

- **Índice de Medida de Similitud Estructural (SSIM)**

Este índice también sirve para determinar la similitud de dos imágenes. En particular mide la similitud estructural entre ambas, teniendo en cuenta la luminancia, contraste y estructura. Toma valores entre -1 y 1 siendo 1 el índice para imágenes idénticas, por lo que valores próximos a 1 indican alta similitud. La obtención de este índice es más complejo por lo que podemos usar librerías externas para calcularlo.



**Figure 17.** Comparación (de izquierda a derecha) de la imagen original, la imagen comprimida y la diferencia entre ambas junto con el ratio de compresión (CP) de la imagen comprimida, el error cuadrático medio (MSE) y el índice de medida de similitud estructural (SSIM).

En la Figura 17 se presenta un ejemplo de cómo podemos comparar una imagen comprimida mediante transformada wavelet discreta de Haar descartando el 20% de los coeficientes de detalle del primer nivel con la imagen original. El ratio de compresión, mayor que uno, indica que la imagen comprimida tiene menor tamaño que la original. El error cuadrático medio mayor que cero nos indica que se ha perdido cierta información de la imagen original y el índice de medida de similitud estructural, próximo a uno nos indica que ambas imágenes son muy similares.

### 3.4 Compresión de vídeo utilizando Wavelets

La compresión de vídeo utilizando wavelets sigue la evolución natural de nuestro proyecto donde hemos iniciado aplicando esta transformada para una imagen y ahora lo hacemos para múltiples imágenes ya que un vídeo no es más que una sucesión de imágenes o frames consecutivos uno detrás de otro. Por este motivo se nos ocurrió la forma más intuitiva de comprimir un vídeo mediante wavelets que se resume en lo siguiente:

1. Separar el vídeo en frames
2. Aplicación de la transformada para cada frame
3. Volver a unir los frames para obtener el vídeo comprimido

A continuación nos dedicaremos a explicar cada paso del proceso de forma detallada.

#### 3.4.1 Separar el vídeo en frames

Para ilustrar todo el proceso utilizamos el archivo llamado **baile.mp4**, aunque se podría utilizar cualquier archivo de video de tipo **.mp4**. El códec del vídeo es **H264**, *veremos que es importante después*, el peso del archivo es de **1.17 MB**. Nuestro objetivo es tratar de comprimir el vídeo lo máximo posible tratando de comprometer la calidad lo mínimo posible.

Mediante el paquete **open-cv** de Python, podemos dividir el vídeo en frames, en este caso, el vídeo se convierte en 344 frames de peso **7,45 MB**, esto podría parecer poco intuitivo, lo lógico es pensar que si dividimos un video en frames, la suma de estos debería ser igual al peso del video. Sin embargo, a los vídeos se les aplican códecs que comprimen el tamaño de los frames sin apenas influir en la calidad. Por lo tanto, al obtener los frames individuales, se descomprimen y pesan más que este. Esto nos indica que cuando queramos comparar el vídeo original con el que se le ha aplicado la transformación, deben estar en el mismo códec, en este caso **H264**.

Los frames resultantes se almacenan en una carpeta en el directorio **./data/frames\_video\_original**. El tipo de archivo de las imágenes es **JPEG**.

### 3.4.2 Aplicación de la transformada para cada frame

Para aplicar la transformada, como para otras partes del proyecto, se utiliza el paquete **pywavelets** entre otros. En primer lugar creamos una nueva carpeta para almacenar los que serán los frames transformados, la ruta es `./data/frames_modificados`, luego de forma iterativa vamos aplicando el siguiente proceso a cada frame:

1. Aplicamos la transformada wavelet, en este caso daubechies, `db4` en `python`, y cuantos niveles de descomposición en aproximación y detalles queremos, para el ejemplo hemos utilizado 3. No ha sido necesario utilizar los frames en escala de grises dado que pywavelets opera correctamente con imágenes a color.
2. Se obtienen los coeficientes de aproximación y los de detalles. De estos, solo nos quedamos con el último coeficiente de aproximación y con los coeficientes de detalle en el mismo nivel y por encima de los del coeficiente de aproximación.
3. Creamos un threshold adaptativo que consiste en utilizar un percentil como límite, todo valor por debajo de ese percentil se le asigna el valor 0, mientras que aquellos valores por encima del umbral no sufren cambios. Estos son los valores de los coeficientes que mayor información aportan sobre la imagen. Le aplicamos el umbral a los coeficientes de detalle que están en el mismo nivel que los de aproximación mientras que el resto de detalles no se les aplica dicho threshold.
4. Realizamos la transformada wavelet inversa con los nuevos coeficientes de aproximación y detalle que hemos obtenido, reconstruyendo así la imagen ya transformada, aplicando la wavelet daubechies.
5. Guardamos los frames dentro de la carpeta directorio.



(a) Primer frame del vídeo original.



(b) Primer frame del vídeo reconstruido.

**Figure 18.** Comparación de frames en la compresión de vídeo.

### 3.4.3 Volver a unir los frames para obtener el vídeo comprimido

Una vez tenemos los frames transformados, realizamos el proceso de reconversión a video. Establecemos el directorio de salida del que será el vídeo resultante, `./data/video_salida.mp4`, hay que establecer el número de frames por segundo (fps) del vídeo, el estándar son 30 fps. La librería **open-cv** de Python, no permite utilizar el códec **H264**, por lo que inicialmente, el vídeo se codifica con el tipo **XVID** y luego mediante el uso de la librería **moviepy**, lo recodificamos a **H264** de nuevo. El resultado final se encuentra en `./data/video_convertido.mp4`, el peso del nuevo vídeo es de **655 KB** por lo que al final se aprecia una reducción significativa del tamaño del archivo. En cuanto a la pérdida de calidad, apenas se perciben cambios visuales entre los vídeos.

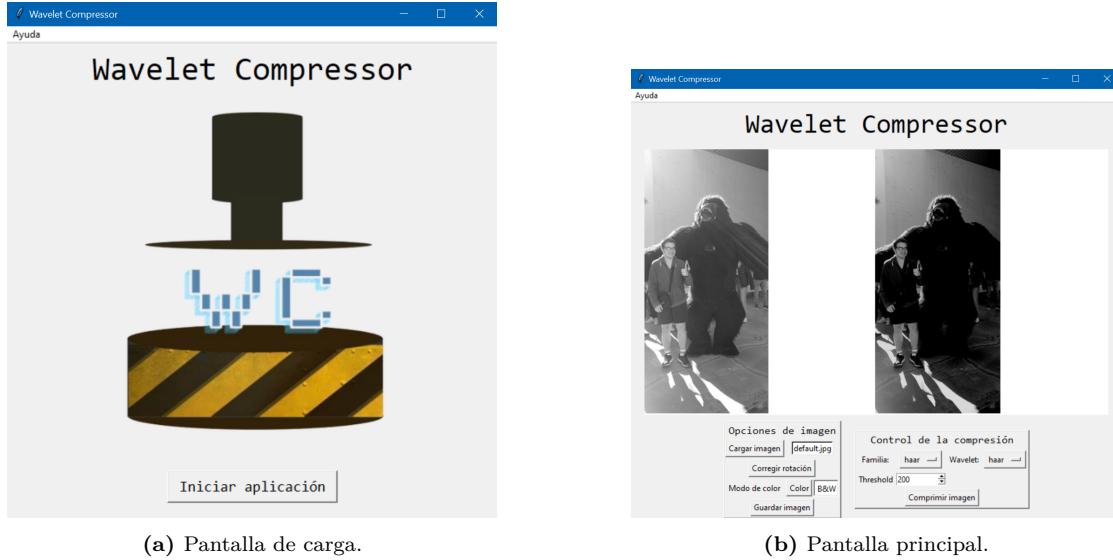
## 4 Producto final: Wavelet Compressor

La compresión de imágenes es directamente aplicable en el día a día. Por ello, decidimos unir los conocimientos desarrollados en esta memoria para crear una interfaz de usuario gráfica (GUI) que permita aplicar la compresión con wavelets de manera rápida y sencilla.

**Wavelet Compressor** está basada en la librería `tkinter` de Python. Esta librería es, a su vez, una interfaz de Tk para Python, escrita en el lenguaje de programación Tcl/Tk. Su implementación en Python es intuitiva: creamos una clase base, comúnmente llamada `root`, sobre la que vamos añadiendo distintos **Frames**. Dentro de los **Frames** podemos posicionar distintos elementos de entre una gran variedad: `canvas`, `Label`, `Button`, `Spinbox`, etc. Cada uno de estos elementos se ordena dentro de su `Frame` padre controlando su geometría con uno de los métodos disponibles: `pack()`, `grid()` o `place()`.

En concreto, el menú principal de nuestra GUI, representado en la Figura 19b, cuenta con los siguientes elementos principales:

- Display de imágenes: es un único `canvas` sobre el cual calculamos dónde y con qué tamaño han de ser representadas las imágenes.
- Panel de imagen: es un `Frame` que contiene distintos `Buttons` y `Labels`. Este panel permite cargar la imagen, corregir su rotación, cambiar entre color y BW y guardar la imagen comprimida.
- Panel de compresión: también es un `Frame`, contenedor de `Buttons`, `Labels` y un `Spinbox`. Estos controles modifican la familia, la wavelet, y el threshold de compresión.



**Figure 19.** Paneles principales de la GUI **Wavelet Compressor**.

## 5 Conclusiones

El presente trabajo ha explorado exhaustivamente la aplicación de técnicas de compresión utilizando wavelets en el ámbito de imágenes y videos, abordando tanto aspectos teóricos como prácticos. A continuación, se presentan las conclusiones más relevantes obtenidas a lo largo de la investigación:

El análisis teórico proporcionado en la sección de Contenidos Teóricos ha sentado las bases fundamentales para comprender el funcionamiento de las transformadas wavelet en la compresión de información visual. La comprensión de conceptos clave, como la descomposición multivariante y los métodos de comparación, ha permitido contextualizar de manera efectiva los procedimientos prácticos implementados.

El procedimiento práctico ha abordado la compresión de imágenes tanto en R como en Python, destacando la versatilidad y eficiencia de PyWavelets en la manipulación de imágenes en niveles de gris y color. La implementación de la descomposición multivariante ha demostrado ser una estrategia efectiva para la compresión sin pérdida de detalles significativos.

En el caso de la compresión de video, la metodología seguida, que implica la separación en frames, la aplicación

de la transformada a cada uno y la posterior reconstrucción, ha evidenciado la aplicabilidad de las wavelets en este contexto, logrando una reducción de tamaño sin comprometer de manera significativa la calidad visual.

La culminación de los esfuerzos prácticos se traduce en el desarrollo de el producto final: **Wavelet Compressor**. Esta herramienta integra las técnicas discutidas y aplicadas, ofreciendo una solución versátil para la compresión de imágenes mediante wavelets. Su diseño modular y la posibilidad de ajustar parámetros clave hacen de esta herramienta una opción valiosa para aquellos que buscan una compresión eficiente y adaptable.

En conclusión, la investigación ha revelado que la aplicación de wavelets en la compresión de imágenes y videos es una estrategia viable y efectiva. La comparación de distintos wavelets, valores de umbral y modos de compresión ha permitido identificar configuraciones óptimas para distintos casos de uso.

Este trabajo no solo contribuye al entendimiento teórico de las wavelets en el contexto de la compresión, sino que también proporciona una implementación práctica que puede ser aprovechada en diversos escenarios.