Trabajo Fin de Máster
Máster en Ingeniería de Telecomunicación

Comparación de algoritmos de hashing para la identificación de anuncios en TV

Autor: Mónica Pérez Hernández
Tutor: José Ramón Cerquides Bueno

Dpto. Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2019

Trabajo Fin de Máster
Máster en Ingeniería de Telecomunicación

Comparación de algoritmos de hashing para la identificación de anuncios en TV

Autor:
Mónica Pérez Hernández

Tutor:
José Ramón Cerquides Bueno
Profesor Titular

Dpto. de Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2019

Trabajo Fin de Máster: Comparación de algoritmos de hashing para la identificación de anuncios en TV

Autor: Mónica Pérez Hernández

Tutor: José Ramón Cerquides Bueno

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2019

El Secretario del Tribunal

A mi familia

A mis amigos

A mis profesores

Agradecimientos

Ahora que mi etapa en la Universidad parece que toca a su fin, sólo me queda dar las gracias.

Gracias, a mis padres por haberme dado la oportunidad de salir de casa a estudiar esa carrera que me apasionaba desde que tuve edad de decidir qué quería hacer en la vida, por haberme animado a seguir estudiando y hacer el máster, por comprenderme y darme su apoyo incondicional en cada decisión que he tomado.

Gracias, a mis compañeros del máster por toda la ayuda que me han brindado durante estos dos cursos, las risas, los cafés del descansito y miles de cosas más que no caben en tan sólo un par de párrafos; en especial, gracias a Agustín, Lourdes, Paloma, Rocío, Luis, Sara y Fran, por todas las cervecitas post clase, el banco de apuntes compartidos, las notas de voz interminables explicando ejercicios, etc. porque además de todo lo aprendido, me llevo muchos amigos que espero seguir manteniendo en el futuro.

Gracias también a mis profesores, en especial a mi tutor, Ramón Cerquides, por haber confiado en mí para realizar este proyecto, por haberme ayudado a alcanzar mis metas.

En definitiva, gracias a todas y cada una de las personas con las que he compartido los años que he pasado en la Universidad, por haber hecho de ellos el buen recuerdo que son ahora.

Mónica Pérez Hernández

Sevilla, 2019

Resumen

En este documento se lleva a cabo un estudio sobre diferentes algoritmos que permiten tomar huellas de imágenes en base a sus características. Esto se extrapola para su uso en la comparación de archivos de vídeo, de modo que, tomando el hash del primer fotograma no negro de cada archivo, dicho hash sirva tanto para distinguir cada vídeo de los demás como para identificar un vídeo pese a modificaciones tales como adición de ruido o la reducción de la tasa de vídeo, entre otras.

Inicialmente, se toman una serie de archivos de vídeo de prueba, correspondientes a spots publicitarios y se realiza una serie de modificaciones sobre ellos: adición de ruido, reducción de la tasa de bit, reescalado… Una vez hecho esto, se calcula el hash de los fotogramas no negros de los vídeos originales y de los modificados con el objetivo de compararlos entre sí para establecer umbrales personalizados para cada algoritmo y evaluar sus prestaciones, además de calcular el coste computacional de cada uno de ellos.

Una vez que se tienen todos estos datos, se comparan los algoritmos y se evalúa cuáles de ellos ofrecen las características más atractivas.

Abstract

In this document, we study different algorithms that allow us to take fingerprints of images based on their features. Those algorithms can be used to compare video files by computing the hash of the first non-black frame of two different videos and checking the differences between the bits of their hashes. Comparations between hashes allow us both identify modified videos and detect different video files.

Initially, we take some televisión ads. Then, some modifications are aplied to those video files: noise addition, bit rate reduction, frame resize, etc. When it is done, we calculate the hash of every non-black frame of both original and modified videos files to compare them in order to check how similar they are: the Hamming distance between the hashes will allow us to evaluate the performance of every algorithm and their computational cost.

Once we have all of the data, we can compare the algorithms and choose the one who presents the most attractive features.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

"Thus we may have knowledge of the past but cannot control it; we may control the future but have no knowledge of it".

Claude Elwood Shannon

El desarrollo de programas informáticos que permitan o faciliten el reconocimiento de contenido audiovisual es cada vez más popular. Su interés radica en que pueden ser aplicados a numerosos ámbitos como pueden ser la detección de anuncios de televisión [1], la detección de distribución de contenido no autorizado en Internet [2], la detección de vídeos duplicados para purgar bases de datos, televisión interactiva [3], etc.

Una vez comprobado el éxito de este tipo de sistemas surge una nueva necesidad, que no es otra que la de mejorarlos. Uno de los posibles focos de mejora radica en implementar un algoritmo de extracción de huella que ofrezca las mejores prestaciones posibles con un coste computacional asequible, es decir, que la bondad del algoritmo no se vea penalizada por el tiempo de obtención de la huella o de comparación entre ellas,entre huellas, la capacidad de escalar, la granularidad de cada algoritmo, etc.

El hash, fingerprint o "huella" de un elemento se define como la secuencia de bits que resulta de aplicar un cierto algoritmo sobre un elemento de entrada y que tiene como propósito identificar unívocamente (dentro de lo posible) a dicho elemento. Dicho hash tendrá, para el algoritmo en cuestión, una longitud finita y fija, independientemente del tamaño de los datos de entrada; esto no podría ser de otra forma, ya que resulta más fácil comparar entre sí secuencias de la misma longitud.

1.1 Motivación

Existen numerosas técnicas y algoritmos para realizar el "fingerprinting", algunos más populares que otros. Por este motivo, resulta de especial interés compararlos entre sí, determiner cuáles son los puntos fuertes y menos fuertes de cada algoritmo para que nos sea más fácil escoger el algoritmo óptimo en función del ámbito de aplicación concreto.

De entre todas las características que definen un algoritmo de hashing o fingerprinting, destacan las siguientes:

- **Robustez**: implica la identificación correcta del contenido pese a adversidades como la compression, la distorsión debido a la presencia de ruido, etc.
- **Precisión**: cómo de baja es su probabilidad de error, es decir, si arroja el mínimo número possible de falsos negativos y falsos positivos.
- **Discriminabilidad**: capacidad para diferenciar dos huellas distintas.
- **Coste computacional y eficiencia**: tiempo y recursos que se consumen durante la obtención de la huella, además de espacio en memoria.
- **Escalabilidad**: capacidad de adaptarse al tamaño del Sistema: cuantos más elementos haya, más bits

se necesitarán en la longitude de la huella para poder garantizar precision.

- **Versatilidad**: capacidad para adaptarse a diferentes formatos de entrada.
- **Seguridad**: vulnerabilidad de la solución ante posibles manipulaciones.

La motivación de este trabajo será evaluar cada una de estas propiedades en los algoritmos que se propondrán más adelante, en pro de escoger el que obtenga mejores resultados en el cómputo global.

1.2 Objetivos

El objetivo de este documento es estudiar las características de varias técnicas de fingerprinting, evaluando su capacidad de detección y umbrales que garanticen las mejores prestaciones posibles para, finalmente, escoger el más apropiado para la detección de anuncios de television.

Por este motivo, dichas técnicas de fingerprinting se aplicarán sobre una serie de vídeos publicitarios emitidos en television en el año 2016.

1.3 Alcance

El alcance del proyecto consistirá en:

1. Identificar, en base a estudios y publicaciones recientes, los algoritmos de fingerprinting más empleados en el ámbito que nos ocupa
2. Someter dichos algoritmos a una serie de pruebas para evaluar sus propiedades.
3. Modelar el comportamiento de dichos algoritmos en su aplicación en la detección tanto de vídeos similares como de vídeos “diferentes” y, en base a cada modelo, establecer umbrales personalizados que optimicen su funcionamiento
4. Una vez se haya modelado y configurado cada algoritmo, escoger de forma argumentada y apoyándose en los resultados el que mejor se adapta al fin buscado: el reconocimiento de anuncios.

1.4 Estructura del documento

Para abarcar todo lo requerido por el alcance del proyecto, el presente documento se descompone en una serie de partes bien diferenciadas.

Comienza contextualizando el hashing, los antecedentes en el campo y los avances y aplicaciones de esta técnica más significativos hasta la fecha.

Seguidamente, se describen detalladamente los algoritmos de hashing escogidos para el estudio, aportando datos sobre sus principales características y bases teóricas de funcionamiento. Se aportarán, además, ejemplos gráficos que apoyen el marco teórico descrito.

Una vez contextualizados los algoritmos a analizar, se aportará información sobre el entorno de trabajo y las herramientas escogidas para la implementación de la batería de pruebas a la que van a ser sometidos dichos algoritmos.

Por último, se aborda el problema.

En primer lugar, se describe el diseño escogido para las pruebas: descripción de las modificaciones que se van a realizar sobre los vídeos de prueba, análisis que se van a realizar sobre dichos vídeos, jerarquía de directorios en la que se van a ir almacenando los resultados obtenidos, librerías necesarias, módulos de código desarrollados (incluyendo descripción, entradas y salidas de cada una de las funciones implementadas), ficheros de configuración, etc., y, por último, la descripción del programa principal

Una vez expuesto el diseño, se completa con detalles sobre la implementación.

Se aporta información sobre el lenguaje de programación utilizado, los métodos escogidos para el almacenamiento de los datos obtenidos, cómo se realizan el tratamiento y visualización de los resultados o el modelado del comportamiento de cada uno de los algoritmos y la definición de umbrales de tolerancia a fallos.

Por último, se expone el grueso de los resultados y se finaliza con las conclusiones obtenidas a partir de los mismos.

2 ESTADO DEL ARTE

"Imagination is more important than knowledge. Knowledge is limited. Imagination encircles the world.".

Albert Einstein

En este apartado se pretende contextualizar el uso del fingerprinting de imágenes y vídeos, describiendo tanto su evolución en los últimos años como los avances más recientes.

2.1 Contexto

El hashing de imágenes es un proceso que consiste en examinar el contenido de una imagen y, a partir de éste, construir una huella, es decir, un valor que la identifique de manera única y exclusiva.

A diferencia de las funciones de hash criptográficas, que se basan en el efecto de avalancha de pequeños cambios en la entrada que conducen a cambios drásticos en la salida, los hashes perceptuales, al estar basados en los detalles de la propia imagen, están "cerca" entre sí si las características son similares.

Al utilizar hashes criptográficos los valores de hash son aleatorios, pues los datos utilizados para generar el hash actúan como una semilla aleatoria: datos iguales generarán el mismo resultado, y datos diferentes crearán resultados distintos, aunque la diferencia sea mínima. Cambiar un solo bit en un archivo dará como resultado un hash completamente diferente. Sabiendo esto, es inviable el uso de este tipo de hashes para comparar similitudes entre imágenes.

En contraste con los anteriores, los hashes perceptuales sí pueden compararse para comprobar similitud entre imágenes: datos parecidos darán como resultado hashes parecidos. Por este motivo, es interesante el uso de hashes de este tipo.

2.2 Antecedentes

El hashing de contenido audiovisual, tal como se ha mencionado al introducir este documento, es una técnica que se aplica en numerosos campos. Dejando a un lado el hashing de archivos de audio [4] y acotando el ámbito de aplicación al hashing de imágenes y vídeo sigue existiendo un abanico de posibilidades muy amplio.

En cuanto al hash de imágenes, se han desarrollado hasta la fecha multitud de algoritmos y aplicaciones para este cometido. Cabe destacar, por su aplicación en autenticación de contenido, búsquedas en bases de datos y reconocimiento de marcas de agua, los siguientes algoritmos:

- Un estudio publicado en 2006 sobre un algoritmo de hashing basado en la transformada de Fourier y la aleatorización controlada [4]. En dicho estudio se modela la robustez del hash de imágenes como un problema de prueba de hipótesis y se evalúa el rendimiento en varias operaciones de

procesamiento de imágenes, mostrando que la función hash propuesta es resistente a las modificaciones que preservan el contenido, como las distorsiones geométricas y de filtrado moderadas.

- Un artículo publicado a partir de una conferencia sobre hashing perceptual [5] que data del año 2000. Este artículo presenta una novedosa técnica de indexación de imágenes que utiliza estrategias de procesamiento de señales aleatorias para una compresión no reversible de imágenes en cadenas binarias aleatorias, demostrando que es robusto contra los cambios de imagen debidos a la compresión, distorsiones geométricas y otros ataques.
- Un artículo más reciente, publicado en 2015 [6] propone un esquema de hashing de imagen robusto que utiliza patrones binarios locales simétricos en el centro (CSLBP). En dicho método, las características CSLBP se extraen de cada bloque no superpuesto dentro de la imagen de escala de grises original. Para cada bloque, el código hash final se obtiene mediante el producto interno de su vector de características CSLBP y un vector de pesos pseudoaleatorio. Además, la descomposición del valor singular (SVD) se puede combinar con CSLBP para introducir un método de hashing más robusto llamado SVD-CSLBP.

Por otra parte, existen numerosas técnicas de hashing de vídeo. Sin embargo, resulta interesante aplicar el hashing de imágenes para la identificación de vídeos aplicando dichos métodos fotograma a fotograma. Concretamente, el enfoque más adecuado para lograr el objetivo de este proyecto resulta ser el hashing perceptual, es decir, el hashing basado en las características que muestra la imagen.

2.3 Situación actual

Actualmente existen numerosas implementaciones de métodos de hashing perceptual.

Encontramos, por ejemplo, una implementación realizada en lenguaje PHP [8] que incluye cuatro métodos de hashing. A raíz de esta implementación se realizan otras nuevas en distintos lenguajes, como por ejemplo la librería ImageHash, desarrollada en lenguaje Python. Dicha librería incorpora cuatro métodos para la obtención del hashing de imágenes, incluyendo además variantes para la implementación de cada método. Parece interesante, por tanto, evaluar las funcionalidades que ofrece esta librería y estudiar las prestaciones que cada uno de los métodos que incluye.

3 ENTORNO DE TRABAJO, LIBRERÍAS Y HERRAMIENTAS UTILIZADAS

"I was taught that the way of progress is neither swift nor easy.".

Marie Curie

El entorno de pruebas ha sido desarrollado en Python sobre la distribución libre Anaconda corriendo en una máquina con Windows 10. Se valoró el uso de Matlab como primera opción para desarrollar este estudio; sin embargo, el hallazgo de la librería ImageHash de Python y la posibilidad de integrarlo con la herramienta de edición de video FFmpeg fueron decisivos a la hora de escoger el enfoque de este estudio sobre las técnicas de fingerprinting o hashing detalladas anteriormente.

Las características físicas del equipo en el que se ha realizado la batería de pruebas son:

- Marca: Lenovo
- Modelo: Yoga 530 14-IKB
- Sistema operativo: Windows 10 Home de 64 bits
- Memoria RAM: 8GB
- Procesador: Intel Core i7-8550U (8th gen.)

Las versiones usadas para las herramientas mencionadas son:

- Anaconda 2019.03 para Windows
- Spyder 3.3.3
- IPython 7.4.0
- Python 3.7.3
- ImageHash v4.0
- FFmpeg 4.1.3

3.1 Anaconda

Anaconda [1] es una distribución de código abierto que integra los lenguajes Python y R. Permite procesar grandes volúmenes de datos y labores pesadas de computación a la par que permite simplificar el despliegue y la administración de los paquetes software.

Con más de 11 millones de usuarios en todo el mundo, es líder en la industria del desarrollo en Python y en R, pues permite:

- Acceder y descargar rápidamente más de 1500 paquetes de Python y R.
- Administrar librerías, dependencias y entornos gracias a Conda.
- Desarrollar y entrenar modelos tanto de machine learning como de deep learning mediante scikit-learn, TensorFlow y Theano.
- Realizar análisis profundos y complejos de grandes estructuras de datos con Dask, NumPy, pandas y Numba.
- Visualizar y presentar los resultados con Matplotlib, Bokeh, Datashader y Holoviews.

Figura 1: Herramientas de Anaconda

3.1.1 Anaconda Navigator

Anaconda Navigator [2] es una interfaz gráfica de usuario para escritorio incluida en Anaconda.

Permite iniciar las aplicaciones o herramientas de Anaconda y administrar fácilmente paquetes de Conda, entornos y canales sin la necesidad de usar comandos en un terminal.

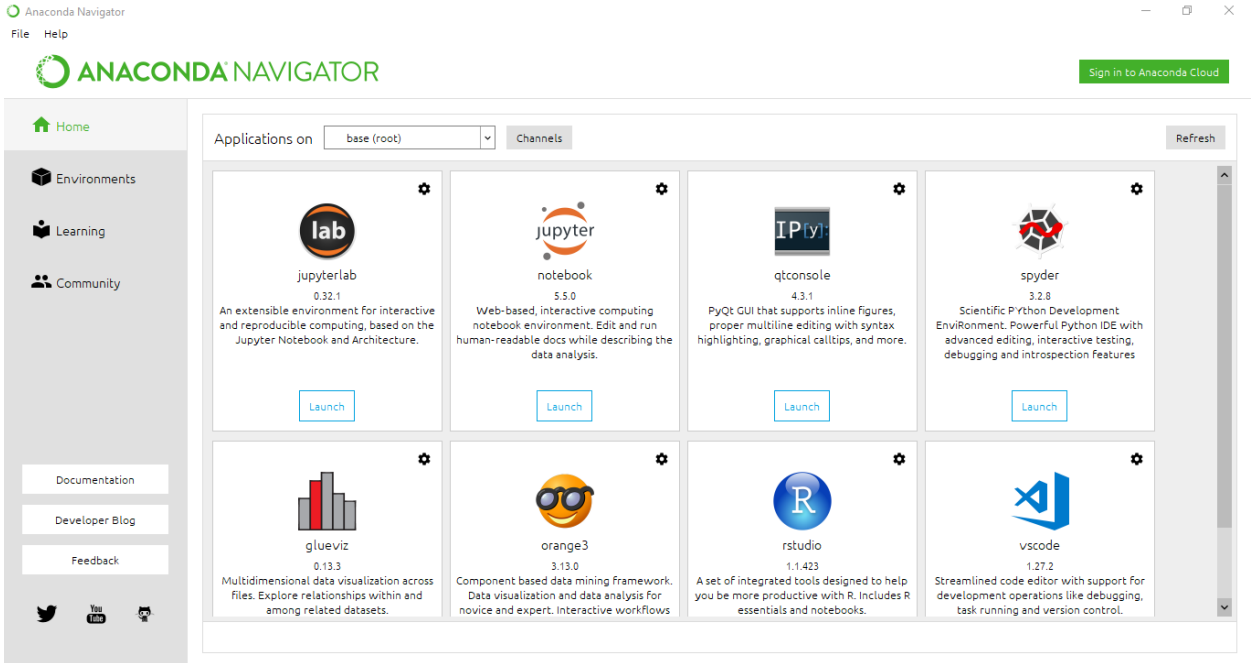


Figura 2: Anaconda - Home

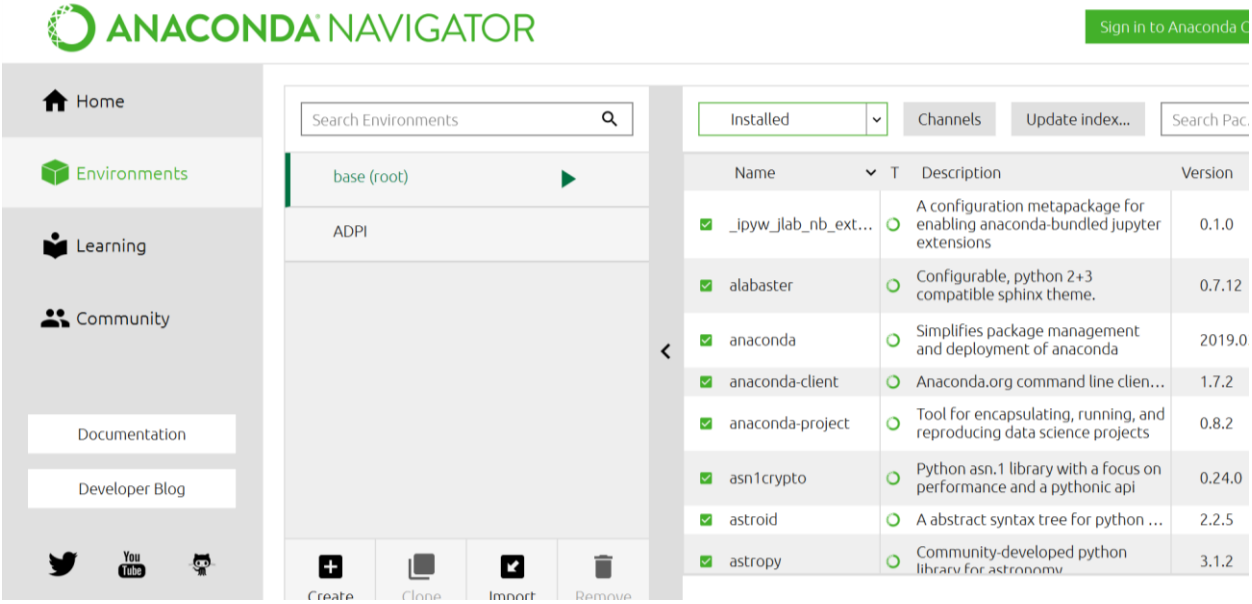


Figura 3: Anaconda Navigator - Environments

3.1.2 Spyder

Además de todas estas características, Anaconda integra Spyder [3], que es un poderoso entorno escrito en Python, para Python, y diseñado por y para científicos, ingenieros y analistas de datos.

Spyder cuenta con un amplio abanico de avanzadas funcionalidades de edición, análisis, depuración y creación de perfiles de una herramienta de desarrollo integral con la exploración de datos, la ejecución interactiva, la inspección profunda y grandes capacidades de visualización de un paquete científico.

Spyder ofrece integración con muchos paquetes científicos populares, incluidos NumPy, SciPy, Pandas, IPython, QtConsole, Matplotlib, SymPy y más. Y, además de sus muchas funciones integradas, Spyder se puede ampliar aún más a través de complementos de terceros; se puede usar como una biblioteca de extensión PyQt5, permitiendo aprovechar su funcionalidad e incrustar sus componentes (como la consola interactiva o el editor avanzado) en su propio software.

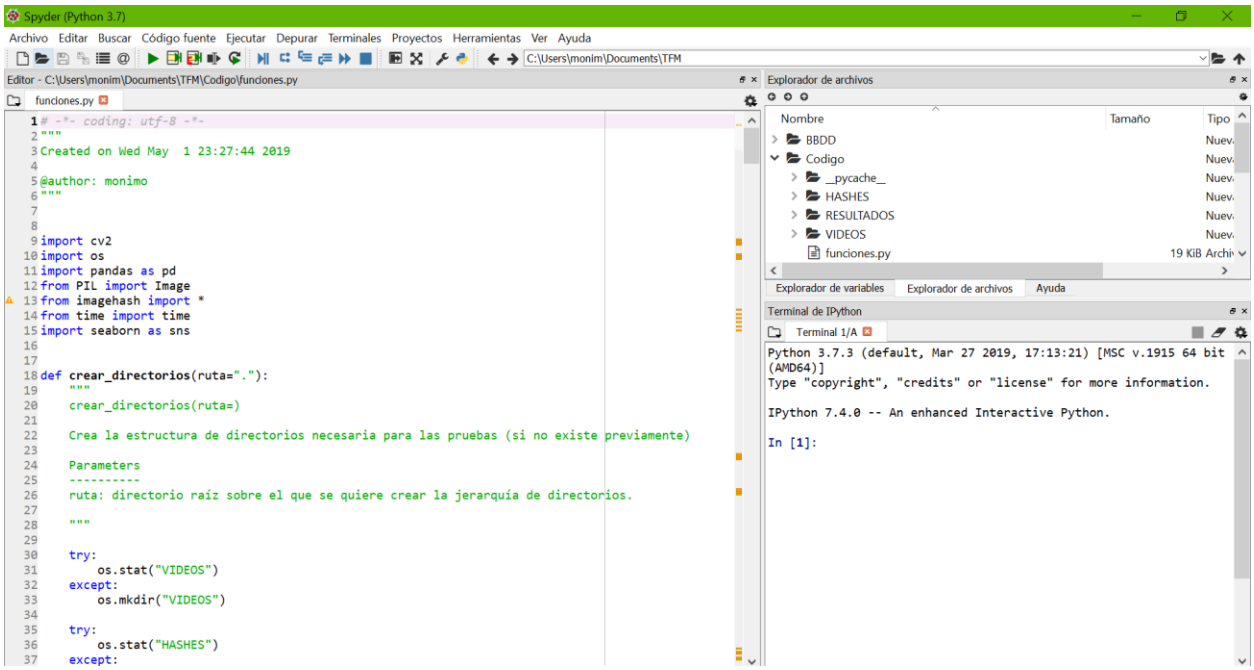


Figura 4: Spyder

3.2 Librería ImageHash

Tal como se ha comentado anteriormente, el objetivo de este trabajo es evaluar y comparar los distintos métodos de hashing que ofrece la librería 'imagehash' de Python, cuyas funcionalidades se detallarán más adelante.

Dichos métodos son los siguientes:

- Average hashing
- Perceptual hashing
- Difference hashing
- Wavelet hashing

En los apartados que siguen se van a explicar en profundidad estos cuatro métodos y sus variantes, implementadas también en la librería.

3.2.1 Average hashing

El método Average hashing, en adelante "ahash", consiste en realizar un promediado basado en las bajas frecuencias. Se fundamenta en la premisa de que, en las imágenes, las frecuencias altas aportan detalles, mientras que las frecuencias bajas muestran su estructura:

- Una imagen grande, por lo general, es detallada, con lo cual tiene muchas frecuencias altas.
- Una imagen muy pequeña carece de detalles, con lo cual la mayoría de sus frecuencias son bajas.

Los pasos que aplica el algoritmo progresivamente sobre la imagen en cuestión son los siguientes:

- **Reducir el tamaño.**

  El modo más rápido para eliminar las altas frecuencias y los detalles es reducir el tamaño de la imagen. Por defecto, el algoritmo reduce el tamaño a 8x8, es decir, a 64 píxeles en total, pero puede escogerse. No se mantiene la relación de aspecto, se fuerza el escalado. Esto permite comparar dos imágenes cualesquiera sin que afecte que tengan diferente tamaño.

- **Reducir el color**

  La imagen escalada se convierte a escala de grises. Suponiendo que se ha reducido el tamaño a la medida por defecto, 8x8, se pasa de una imagen de 64 píxeles con 64 valores de rojo, 64 valores de verde y 64 valores de azul, un valor de cada uno de los tres colores por cada píxel, a solamente un valor por píxel

- **Promediar el color**

  Se computa el valor medio de los 64 colores.

  - **Calcular el valor de cada bit**

    El valor de cada bit se establece en función de si está por encima o por debajo del valor medio calculado: si el valor de un píxel se encuentra por encima de la media dicho píxel se computará como un 1, y en caso contrario, como un 0.

  - **Construir el hash**

    Se unen los 64 valores obtenidos (unos y ceros) conformando un entero de 64 bits utilizando el sistema big-endian como criterio: se unen los bits siguiendo un orden de izquierda a derecha y de arriba hacia abajo.

El hash resultante no cambia si la imagen original se escala o cambia su relación de aspecto. Modificaciones como aumentar o reducir el brillo o el contraste, o incluso alterar los colores, no alterarán de forma reseñable el valor del hash.

Sin embargo, a pesar de que es rápido y sencillo, puede resultar demasiado rígido para comparar imágenes de forma fiable, ya que puede presentar errores a la hora de comparar imágenes a las que ese ha aplicado

corrección gamma o un histograma de color. Esto se debe a que los colores se mueven a lo largo de una escala no lineal, cambiando la ubicación del promedio y, por tanto, el resultado del algoritmo.

A continuación, se muestra paso por paso la transformación que sufriría una imagen al aplicársele este algoritmo tomando un escalado a 32x32 píxeles, que es el que se usará posteriormente para el estudio:

Figura 5: imagen original - ahash

Figura 7: imagen escalada – ahash

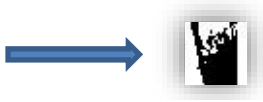


Figura 6: hash de la imagen - ahash

Se parte de una imagen real de 991x558 píxeles, que se comprime a 32x32 píxeles y se convierte a escala de grises. Posteriormente se calcula el valor medio (126.734375) y se computa el hash.

3.2.2 Perceptual hashing

El enfoque del perceptual hashing, phash en adelante, extiende el del método anterior, utilizando una transformada de coseno discreta (DCT) para reducir las frecuencias.

La Transformada Discreta del Coseno de dos dimensiones (2D-DCT) destaca por su capacidad para compactar la información, transformando los datos del dominio espacial al frecuencial. Esta propiedad se aprovecha en pro de minimizar la representación de los datos de origen.

Esta técnica consiste en:

1. **Reducir el tamaño**

   Al igual que se hace en método de ahash, se comienza reduciendo el tamaño de la imagen a 32x32 píxeles en lugar de a 8x8. El objetivo de este escalado es simplificar el cálculo de la DCT.

2. **Reducir el color**

   La imagen se convierte a escala de grises por el mismo motivo por el que se escala en el paso anterior: para reducir el coste computacional de la DCT.

3. **Calcular la DCT**

   Transforma los datos del dominio espacial al frecuencial

4. **Reducir la DCT**

   Aunque el algoritmo se ha aplicado sobre un conjunto de 32x32, se utilizan únicamente los 8x8 píxeles de la esquina superior izquierda para el cálculo del valor promedio

5. **Calcular umbral**

   Se calcula la mediana de entre los valores de la DCT (utilizando solo los valores de baja frecuencia de 8x8 DCT, excluyendo el primer término, ya que este coeficiente puede ser significativamente diferente a los otros valores y falsearía el resultado).

6. **Reducir de nuevo la DCT**

   Se establece el valor de cada uno de los 64 bits del hash tomando un 1 o un 0 dependiendo de si el valor de la DCT de cada píxel está por encima o por debajo del valor promedio calculado en el paso anterior, respectivamente.

7. **Construir el hash**

Se ordenan, siguiendo el criterio big-endian, los unos y ceros del hash conformando un entero de 64 bits.

El resultado no variará mientras la estructura general de la imagen permanezca inalterada; es decir, no se ve afectado por ajustes del gamograma y del histograma de color.

Al igual que con el valor de hash promedio, los valores de pHash pueden compararse utilizando el mismo algoritmo de distancia de Hamming: para cada posición, se compara el par de bits y se suma el número de diferencias.

Utilizando este algoritmo, la secuencia de transformaciones a la que se somete la imagen es la siguiente:

Figura 10: Imagen original - phash

Figura 9: imagen reescalada - phash

Figura 8: hash de la imagen - phash

En primer lugar, se reduce el tamaño de la imagen y se convierte a escala de grises. Una vez hecho esto, se calcula la DCT y se reducen los coeficientes. Por último, se calcula la media (-2.3848) y se construye el hash de la imagen.

3.2.2.1 Phash simple

Además de esta implementación, la librería incluye una variante llamada phash simple, que es exactamente igual a la descrita anteriormente excepto en que a la hora de calcular el umbral (paso 5) se toma el valor promedio de la DCT en lugar de la mediana.

Figura 13: imagen original - phash simple

Figura 12: imagen reescalada - phash simple

Figura 11: hash de la imagen - phash simple

Utilizando esta implementación alternativa, el resultado obtenido al aplicar el algoritmo sería el siguiente:

3.2.3 Difference hashing

Al igual que aHash y pHash, dHash es bastante sencillo de implementar y es muy preciso. Mientras que el algoritmo ahash se centra en los valores promedio y phash evalúa patrones de frecuencia, dhash se basa en el

cómputo de gradientes.

La implementación del algoritmo comprende las siguientes fases:

1. **Reducir el tamaño**

   Tal como se ha comentado anteriormente, la forma más rápida de eliminar las frecuencias altas y los detalles es reducir el tamaño de la imagen. En este caso, redúzcalo a 9x8 para que haya 72 píxeles en total. Al ignorar el tamaño y la relación de aspecto originales, este hash coincidirá con cualquier imagen similar, independientemente de cómo se reescale.

2. **Reducir el color**

   Se convierte la imagen de tamaño reducido a una imagen en escala de grises, permitiendo reducir el coste computacional al pasar de tener 3 valores de color por cada uno de los 72 píxeles a un total de 72 valores (uno por píxel).

3. **Calcular la diferencia**

   El algoritmo dHash se basa en medir la diferencia entre píxeles adyacentes, es decir, identifica la dirección relativa del gradiente. Al tener una imagen de 8 filas de 9 píxeles cada una, se tendrán 8 diferencias entre píxeles por fila, que se traducen en 64 valores de gradiente en el total de la imagen.

4. **Asignar los bits**

   El valor de cada bit del hash se establece simplemente en función de si el píxel izquierdo es más brillante que el píxel derecho (gradiente) tomando como valor un 1 en caso afirmativo, o un 0 en caso contrario.

Al igual que ocurría en el algoritmo ahash, la huella resultante de la ejecución de este algoritmo no cambiará si la imagen se escala o si cambia la relación de aspecto. Aumentar o disminuir el brillo o el contraste, o incluso alterar los colores tampoco cambiarán drásticamente el valor de hash. Incluso ajustes complejos como correcciones gamma o perfiles de color no afectarán el resultado.

Los valores de hash representan el cambio relativo en la intensidad del brillo. Para comparar dos hashes, basta con calcular la cantidad de bits que son diferentes entre dos hashes (distancia de Hamming).

Este algoritmo actúa sobre las imágenes de la siguiente manera:

Figura 16: imagen orignal - dhash

Figura 15: imagen reescalada - dhash

Figura 14: hash de la imagen - dhash

3.2.3.1 Dhash vertical

Además de esta implementación, la librería estudiada incluye una variación de este algoritmo llamada dhash vertical en la que el gradiente se computa de forma vertical en lugar de horizontal sobre una imagen de 9 filas de 8 píxeles cada una.

El resultado de esta implementación alternativa del algoritmo sería el siguiente:

Figura 19: imagen original - dhash vertical

Figura 18: imagen reescalada - dhash vertical

Figura 17: hash de la imagen - dhash vertical

3.2.4 Wavelet hashing

Este método trabaja en el dominio de la frecuencia al igual que el phash, pero utiliza la DWT en lugar de la DCT.

La transformada discreta de wavelet, DWT, es una transformada en frecuencia. Mientras que otras transformadas muy populares como la Transformada de Fourier o la DCT utilizan un conjunto de senos y cosenos como funciones base, la DWT utiliza solamente una, pero variando su forma desplazándola y escalándola. Esta única función básica puede escogerse de entre la Haar wavelet o la Daubechie-4 wavelet, entre otras. El escalado de la función nos da una gran "representación de tiempo-frecuencia" cuando la parte de baja frecuencia es similar a la señal original.

El procedimiento seguido por este algoritmo comprende los siguientes pasos:

1. **Reducir el tamaño**

   En primer lugar, al igual que ocurre en el resto de los algoritmos, se reduce el tamaño de la imagen a 8x8 píxeles por defecto (aunque este tamaño puede ser modificado).

2. **Calcular la DWT**

   Se calcula la DWT de la imagen escalada utilizando la función de Haar por defecto, aunque alternativamente puede escogerse la de Daubechie.

3. **Filtrar las frecuencias**

   Se elimina la frecuencia más baja de Haar (LL). Esto se hace porque la frecuencia de nivel más bajo consta de un único píxel y representa el contraste de la imagen, con lo cual no resulta útil a la hora de computar el hash.

4. **Tomar umbral**

   Se toma la mediana de los valores de la DWT como umbral para comparar los valores obtenidos y construir el hash

5. **Asignar los bits del hash**

   Por último, se construye el hash comparando el valor de la transformada de Wavelet en cada píxel con el umbral calculado en el apartado anterior, asignando un 1 en caso de que el valor del píxel sea superior a dicho umbral o un 0 en caso contrario. Una vez hecho esto se convierte en un entero de 64 bits siguiendo el criterio big-endian al igual que en los algoritmos anteriores.

El resultado de la aplicación de este último algoritmo sobre la imagen es el siguiente:

Figura 22: imagen original - whash

Figura 21: imagen reescalada - whash

Figura 20: hash de la imagen - whash

3.3 FFmpeg

FFmpeg [4] es un entorno de trabajo multimedia de software libre capaz de decodificar, codificar, multiplexar, demultiplexar, transmitir, filtrar y reproducir prácticamente cualquier tipo de archivo. Ofrece compatibilidad con prácticamente todos los formatos, desde los más antiguos y obsoletos hasta los más vanguardistas. Además, es capaz de compilar, y ejecutar las infraestructuras a través de Linux, Mac OS X, Microsoft Windows, BSD, Solaris, etc. bajo una amplia variedad de entornos de compilación, arquitecturas de máquinas y configuraciones.

Ofrece infinidad de soluciones, tanto para desarrolladores de aplicaciones como usuarios finales, combinando las mejores herramientas de software libre disponibles y dispone de una documentación muy extensa además de numerosos códigos de ejemplo.

La herramienta ffmpeg se usa desde la consola de comandos; sin embargo, Python puede explotar toda su funcionalidad gracias a la librería “os”, que permite ejecutar comandos del sistema como ffmpeg con los argumentos deseados.

4 DISEÑO DE LA BATERÍA DE PRUEBAS

"Projects we have completed demonstrate what we know, future projects decide what we will learn.".

Dr. Mohsin Tiwana

Una vez se han puesto en contexto las herramientas y entornos necesarios para desarrollar este estudio, se puede proceder a exponer la organización y el funcionamiento de la batería de pruebas diseñada.

Tal como se ha mencionado anteriormente, el objetivo de este trabajo es comparar una serie de métodos de hashing con el objetivo de discernir cuál de todos ellos es el más apropiado para comparar archivos de vídeo entre sí.

Para la realización de los análisis que se van a describir a continuación, se dispone de una base de datos muy completa que cuenta con 148 anuncios de televisión en formato MP4.

4.1 Análisis

Partiendo de los archivos de vídeo originales, anuncios de televisión concretamente, se han realizado mediante el uso de ffmpeg las siguientes modificaciones:

- Adición de ruido: el objetivo de este análisis es comprobar qué capacidad tiene cada algoritmo para detectar la similitud entre los hashes de los fotogramas del video original y los del video al que se ha añadido ruido.
- Reescalado: consiste en reducir el tamaño de los frames del vídeo y posteriormente devolverlos a su tamaño original para luego comparar los hashes de este vídeo modificado con los del original y evaluar la capacidad de detección de cada algoritmo de hashing.
- Reducción de la tasa de vídeo: se reduce el bitrate del vídeo original y se comparan los hashes de los fotogramas de este vídeo modificado con los del original.

Además de estas transformaciones de los vídeos originales, se realiza un cuarto análisis que consiste en analizar fotograma a fotograma el vídeo original, comparando el hash cada uno de ellos con el del fotograma inicial para comprobar cuál es el primero de ellos en que el algoritmo detecta que hay una diferencia sustancial (superior a un cierto umbral) entre ambos.

Por último, se mide el coste computacional que tiene cada algoritmo, es decir, el tiempo medio que necesita para obtener el hash de un fotograma de vídeo.

4.2 Estructura de directorios

Para la ejecución de las distintas pruebas y el posterior almacenamiento de los hashes y los resultados

obtenidos se ha ideado la siguiente estructura o jerarquía de directorios:

Figura 23: Jerarquía de directorios del entorno de pruebas

En el directorio Raíz se encuentran, además del código en Python de implementación de las pruebas (que se explicará en detalle en un punto posterior), los directorios:

- VIDEOS: contiene todos los vídeos necesarios para la batería de pruebas, tanto los originales como los modificados. Para ello, incluye los siguientes directorios:
  - ORIGINAL: en este directorio deberán situarse todos los vídeos sobre los que se quiera realizar el estudio.
  - LOWER_BR: incluirá una copia con bitrate reducido de cada vídeo situado en el directorio ORIGINAL
  - NOISE: contendrá los archivos de video de la carpeta ORIGINAL con ruido añadido.
  - RESIZED: este directorio albergará una copia de los vídeos originales tras reducir y posteriormente ampliar el tamaño de sus fotogramas.

- HASHES: En este directorio se almacenarán, en formato csv, los hashes de todos los archivos de vídeo (situados en cualquiera de los directorios de la carpeta VIDEO) a medida que se vayan procesando.
- RESULTADOS: Los archivos de este directorio son archivos csv que almacenan el resultado de comparar los hashes del vídeo original con los de sus respectivas modificaciones. También se almacenan los resultados de la detección de movimiento que, como se explicó anteriormente, consiste en comparar el hash del primer frame con el hash de los siguientes hasta detectar el primer frame "diferente" en base a un cierto umbral establecido. Además de estos archivos, contiene una serie de directorios:
    - COMPARACIONES: Contiene archivos de texto plano, de extensión .txt, que muestran a modo de resumen los resultados que contienen los archivos situados en el directorio superior: para cada par de frames (vídeo original y vídeo modificado) se comprueba qué método de hashing ha obtenido el mejor resultado y, una vez repasados todos los frames, se computa el éxito global de cada método.
    - DET_MOVIMIENTO: Este directorio contiene archivos de texto plano que muestran, para cada vídeo, en qué número de frame detectó cada algoritmo una diferencia significativa respecto al frame original.
    - PDFs: Contiene gráficas que representan las funciones de densidad de probabilidad de cada algoritmo de hashing, que resultan de comparar los hashes cada par de vídeos del directorio ORIGINAL.
    - TIEMPOS: Este último directorio contiene archivos de texto plano que muestran, por cada archivo de vídeo situado en el directorio ORIGINAL, el tiempo medio que se toma cada algoritmo para computar el hash de un frame.

4.3 Módulos de código

Los módulos de código se sitúan en el interior del directorio raíz, al mismo nivel que los directorios VIDEOS, HASHES y RESULTADOS.

Estos módulos de código son tres:

- Un primer módulo de funciones, "funciones.py", que incluye todas las funciones definidas para modificar los archivos de vídeo y extraer y almacenar resultados.
- Un fichero de configuración, "config.conf", en el que el usuario define el valor de los parámetros que precisa el programa principal.
- El programa principal, "pruebas.py", que hace uso del módulo de funciones para realizar la batería de pruebas definida por el usuario.

En los siguientes apartados se van a detallar estos módulos en profundidad.

4.3.1 Módulo de funciones

El módulo de funciones, tal como se describió en el punto inmediatamente anterior, incluye todas las funciones necesarias para realizar la batería de pruebas. A continuación, se van a describir dichas funciones en formato tabular, incluyendo su método de llamada, entradas, salidas y una breve descripción de su funcionalidad.

Método 1	crear_directorios
Descripción	Crea la estructura de directorios necesaria para la batería de pruebas (si no existe previamente)
Argumentos de entrada	• **Ruta:** directorio raíz sobre el que se quiere crear la jerarquía de directorios. Por defecto toma el valor "."
Salida	Esta función no tiene argumento de salida

Tabla 1: método crear_directorios

Método 2	extraer_hashes
Descripción	Almacena en un archivo de extensión csv, para cada fotograma no negro del vídeo en cuestión, el hash correspondiente extraído con cada uno de los métodos.
Argumentos de entrada	• **Ruta**: Directorio raíz donde se encuentran almacenado el vídeo del que se quieren extraer los hashes. • **Archivo**: Vídeo del que se van a extraer los hashes • **Índices**: en caso de que el vídeo sea un vídeo modificado (con ruido, escalado, etc.) este parámetro se corresponde con el índice de los fotogramas no negros del vídeo original. Por defecto toma el valor de lista vacía, "[ ]". • **Ruta_des**: ruta en la que se va a almacenar el resultado. Por defecto, el directorio "HASHES". • **Dim**: dimensión del fotograma escalado antes de calcular el hash. Por defecto toma el valor 32. • **Num_fotogramas**: número máximo de fotogramas no negros que se van a analizar del vídeo. Por defecto toma el valor 300.
Salida	Esta función no tiene argumento de salida

Tabla 2: método extraer_hashes

Método 3	medir_tiempos
Descripción	Almacena en un archivo de extensión csv el tiempo medio que necesita cada método para determinar el hash de un fotograma de un vídeo concreto.
Argumentos de entrada	• **Ruta**: Directorio raíz donde se encuentran almacenado el vídeo sobre el que se quiere medir el coste computacional. • **Archivo**: Vídeo del que se van a extraer los hashes • **Dim**: dimensión del fotograma escalado antes de calcular el hash. Por defecto toma el valor 32. • **Num_fotogramas**: número máximo de fotogramas no negros que se van a analizar del vídeo. Por defecto toma el valor 300.
Salida	Esta función no tiene argumento de salida

Tabla 3: método medir_tiempos

Método 4	add_noise
Descripción	Esta función se encarga de añadir ruido al archivo de vídeo pasado como parámetro
Argumentos de entrada	• **Ruta**: Directorio raíz donde se encuentran almacenado el vídeo al que se quiere añadir ruido. • **Archivo**: Vídeo al que se va a añadir ruido • **Ruta_des**: ruta en la que se va a almacenar el resultado. Por defecto, el directorio “VIDEOS/noise”.
Salida	Esta función no tiene argumento de salida

Tabla 4: método add_noise

Método 5	to_lower_br
Descripción	Esta función se encarga de codificar el archivo de vídeo pasado como argumento de entrada con un bitrate más bajo que el del original.
Argumentos de entrada	• **Ruta**: Directorio raíz donde se encuentran almacenado el vídeo que se quiere modificar. • **Archivo**: Vídeo que va a ser codificado con menor bitrate • **Ruta_des**: ruta en la que se va a almacenar el resultado. Por defecto, el directorio “VIDEOS/lower_br”.
Salida	Esta función no tiene argumento de salida

Tabla 5: método to_lower_br

Método 6	resize_vid
Descripción	Esta función realiza el escalado del vídeo pasado como parámetro y, posteriormente, lo devuelve a su escala original.
Argumentos de entrada	• **Ruta**: Directorio raíz donde se encuentran almacenado el vídeo que se quiere modificar. • **Archivo**: Vídeo sobre el que se va a realizar el reescalado • **Ruta_des**: ruta en la que se va a almacenar el resultado. Por defecto, el directorio "VIDEOS/ resized". • **Porcentaje**: porcentaje de escalado que se aplica sobre el tamaño original del vídeo en el primer paso.
Salida	Esta función no tiene argumento de salida

Tabla 6: método resize_vid

Método 7	comparar_hashes
Descripción	Almacena en un archivo de extensión csv el resultado de la comparación entre dos archivos de hash.
Argumentos de entrada	• **Original**: archivo que almacena los hashes del vídeo original. • **Video_mod**: archivo que almacena los hashes del vídeo modificado • **Dim_s**: tamaño del hash. Por defecto toma el valor 32.
Salida	Esta función no tiene argumento de salida

Tabla 7: método comparar_hashes

Método 8	detectar_movimiento
Descripción	Almacena en un archivo de extensión csv el resultado de comparar el hash del primer fotograma de un vídeo con el de os fotogramas restantes
Argumentos de entrada	• **Archivo**: Vídeo del que se van a extraer los hashes • **dim_s**: tamaño del hash. Por defecto toma el valor 32
Salida	Esta función no tiene argumento de salida

Tabla 8: método detectar_movimiento

Método 9	calcular_resultados
Descripción	Almacena en un archivo de extensión txt el resumen de comparar, para cada frame, los hashes obtenidos por los distintos métodos para computar finalmente el éxito global de cada método.

Argumentos de entrada	• **Ruta:** Directorio raíz.
	• **prefijo:** tipo de análisis sobre el que se quieren calcular los resultados. Puede tomar los valores:
	  ○ noise
	  ○ lower_br
	  ○ resized
	  ○ 2
	• **modo_max:** este parámetro indica si se quiere obtener el valor con porcentaje más alto para cada frame (tomando el valor True) o el menor (tomando el valor False). Por defecto toma el valor True.

Salida	Esta función no tiene argumento de salida

Tabla 9: método calcular_resultados

Método 10	representar_pdf
Descripción	Representa la pdf correspondiente a cada método de hashing en un mismo gráfico.
Argumentos de entrada	• **Ruta**: Directorio raíz donde se encuentran los archivos que se van a analizar. • **archivo**: archivo que contiene los resultados de comparar los hashes de dos vídeos • **ruta_des**: ruta en la que se van a almacenar los gráficos resultantes. Por defecto “RESULTADOS/PDFs”
Salida	Esta función no tiene argumento de salida

Tabla 10: método representar_pdf

Método 11	inferior_a_umbral
Descripción	Método aplicado a la detección de movimiento: Devuelve, para cada método de hashing, el índice del primer fotograma cuyo porcentaje de similitud (distancia de Hamming) con el fotograma inicial es inferior al umbral dado
Argumentos de entrada	• **Ruta**: Directorio raíz donde se encuentran los archivos que se van a analizar. • **archivo**: archivo csv que contiene los resultados de comparar para cada algoritmo el hash del primer fotograma con los hashes de los fotogramas restantes • **umbral**: Valor de referencia para la comparación. Por defecto toma el valor 0.85, es decir, se exige un parecido inferior al 85% (distancia de Hamming superior a 0.15).
Salida	Esta función no tiene argumento de salida

Tabla 11: método inferior_a_umbral

Método 12	obtener_pdfs
Descripción	Obtiene la función de densidad de probabilidad ajustando el histograma de los datos de entrada.
Argumentos de entrada	• **datos**: Vector columna que contiene los valores a ajustar. • **n_bins**: número de intervalos en que se va a dividir el histograma • **umbral**: valor de referencia para obtener el área deseada
Salida	• **xs**: valores para el eje de coordenadas de la pdf resultante • **ys**: valores para el eje de ordenadas de la pdf resultante • **maximo_x**: valor de x en el que se alcanza el área deseada • **maximo_y**: valor de y correspondiente a maximo_x • **yi**: índice en el que se alcanza el valor máximo correspondiente a los datos xs, ys

Tabla 12: método obtener_pdfs

Método 13	representar_umbral_pdfs
Descripción	Almacena en archivos "png" las gráficas resultantes de estimar las pdf de la distancia de Hamming correspondiente a los datos de entrada y el punto en el que cada una de ellas satisface un cierto umbral

Argumentos de entrada	• **Ruta**: Directorio raíz.
	• **n_bins**: número de barras del histograma. Por defecto toma el valor 25.
	• **análisis**: archivos sobre los que se quiere realizar el análisis. Por defecto "noise". Las opciones disponibles para este parámetro son:
	  ○ "noise"
	  ○ "lower_br"
	  ○ "resized"
	  ○ "2"
	• **umbral**: valor que se quiere que supere la integral de la pdf asociada a los datos de "análisis". Por defecto toma el valor 0.9999.
	• **analisis_vs**: archivos con los que se quieren comparar los anteriores. Por defecto "2". Las opciones disponibles para este parámetro son las mismas que para el parámetro "análisis".
	• **Umbral_vs**: valor que se quiere que supere la integral de la pdf asociada a los datos de "análisis_vs". Por defecto toma el valor 0.00009.

Salida	Esta función no tiene argumento de salida

Tabla 13: método representar_umbral_pdfs

Método 14	inferior_a_umbral_dif
Descripción	Detección de movimiento: Devuelve para cada método de hashing el índice del primer fotograma cuyo porcentaje de parecido con el fotograma inicial es inferior al umbral dado para cada algoritmo
Argumentos de entrada	• **Ruta**: Directorio raíz. • **umbrales**: valores de referencia para la comparación de porcentajes para cada algoritmo
Salida	Esta función no tiene argumento de salida

Tabla 14: método inferior_a_umbral_dif

Método 15	calcular_resultados_vs_umbral
Descripción	Almacena en un archivo de extensión txt el resumen de comparar, para cada frame, los hashes obtenidos por los distintos métodos para computar finalmente el éxito global de cada método en función del umbral personalizado de cada método.

Argumentos de entrada	• **Ruta:** Directorio raíz.
	• **prefijo:** tipo de análisis sobre el que se quieren calcular los resultados. Puede tomar los valores:
	  o noise
	  o lower_br
	  o resized
	  o 2
	• **umbrales:** valor de referencia de cada algoritmo para la detección de parecido entre dos hashes
	• **modo_max:** este parámetro indica si se quiere obtener el valor con porcentaje más alto para cada frame (tomando el valor True) o el menor (tomando el valor False). Por defecto toma el valor True.

Salida	Esta función no tiene argumento de salida

Tabla 15: método calcular_resultados_vs_umbral

4.3.2 Fichero de configuración

El sistema cuenta con un fichero de configuración que permite parametrizar tanto el análisis final de los datos, que consiste en representar las funciones de densidad de probabilidad de las distancias de Hamming que separan ciertos hashes, como los directorios en los que almacenar los datos de vídeo.

El objetivo de este fichero de configuración no es otro que evitar que se modifiquen los archivos de código, proporcionando una vía sencilla para la manipulación de los parámetros que condicionan el estudio.

Concretamente, el fichero de configuración diseñado se divide en cinco secciones, cada una con sus correspondientes parámetros:

- **Directorios:** contiene los parámetros asociados a los directorios en los que se almacenan los archivos de vídeo originales y en los que se almacenarán las modificaciones realizadas sobre estos. Los parámetros de esta sección son los siguientes:
  - ruta_v_original → directorio en el que se encuentran almacenados los archivos de vídeo que se quieren analizar
  - ruta_v_noise → directorio en el que se almacenarán los archivos mp4 una vez que se les haya añadido ruido
  - ruta_v_resized → en este directorio se situarán los archivos de vídeo a los que se les haya aplicado el reescalado
  - ruta_v_lbitrate → en este directorio se almacenarán los vídeos a los que se les haya reducido la tasa de bit.

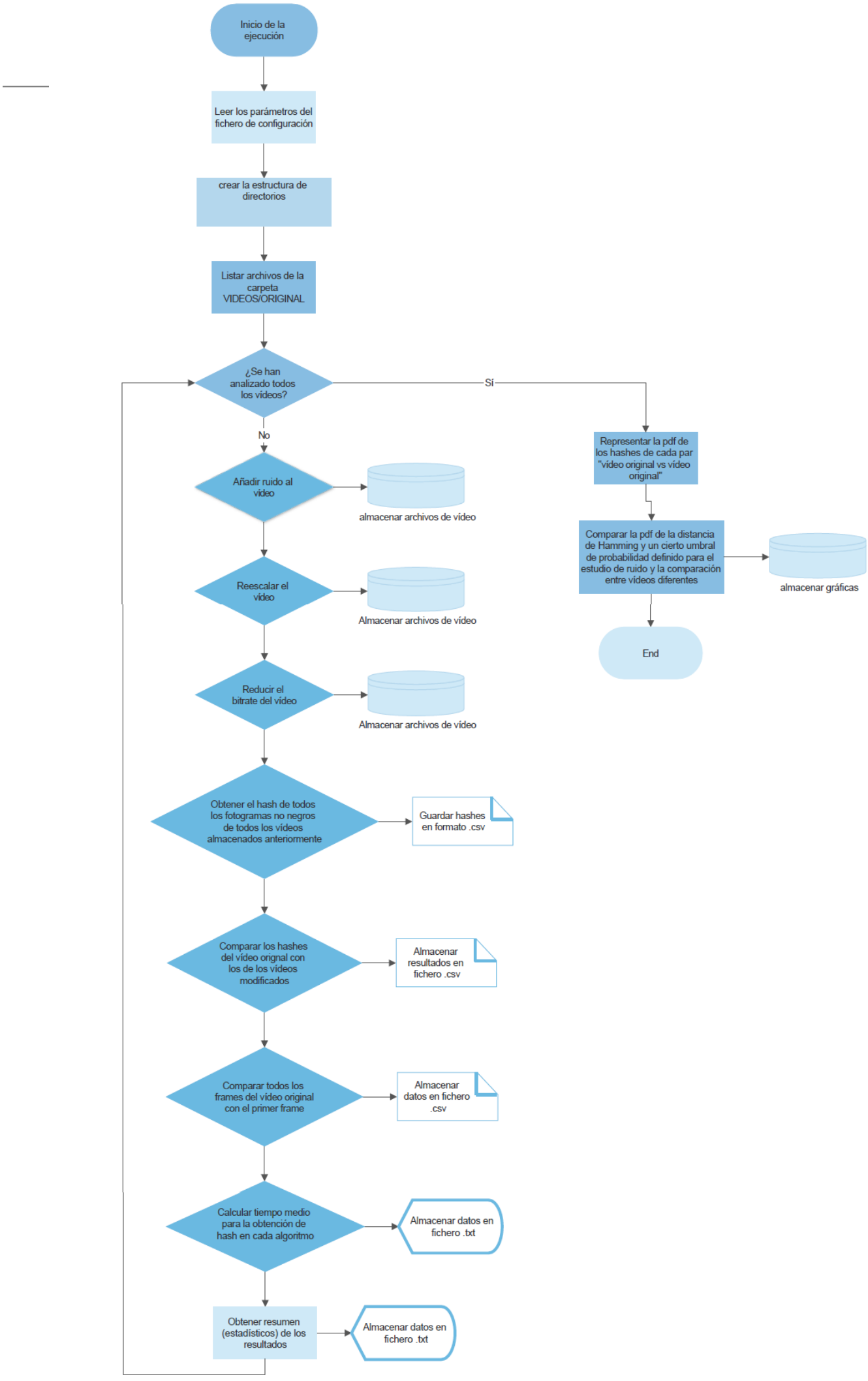
- **Parámetros detección movimiento:** contiene el umbral predefinido para la detección de movimiento:
  - Umbral → umbral que determina que dos hashes son lo suficientemente diferentes como para considerarse "movimiento" de la escena. Por defecto toma el valor 0.85.
- **Parámetros análisis pdf:** esta sección contiene los parámetros asociados al análisis de las funciones de densidad de probabilidad correspondientes a la distancia de Hamming entre los hashes de cada algoritmo de fingerprinting. Los parámetros que contiene esta sección son los asociados a la función "representar_umbral_pdfs" descrita en el método 13 del punto anterior de este documento:
  - n_bins → número de intervalos en los que se quiere dividir el histograma de forma previa a la estimación de la pdf de los datos
  - análisis → análisis cuyos datos quieren analizarse
  - umbral → umbral que se quiere que supere la integral de la pdf del análisis anterior
  - analisis_vs → análisis cuyos datos quieren compararse con los del análisis anterior
  - umbral_vs → umbral que se quiere que supere la integral de la pdf del segundo análisis definido.
- **Umbrales similitud:** contiene los umbrales calculados para la detección de similitud entre vídeos originales y vídeos modificados para cada uno de los algoritmos analizados:
  - Ahash → umbral para el algoritmo ahash
  - Phash → umbral para el algoritmo phash
  - Phash_s → umbral para el algoritmo phash simple
  - Dhash → umbral para el algoritmo dhash
  - Dhash → umbral para el algoritmo dhash vertical
  - Whash → umbral para el algoritmo whash
- **Umbrales diferencia:** contiene los umbrales calculados para cada uno de los algoritmos para la detección de vídeos diferentes:
  - Ahash → umbral para el algoritmo ahash
  - Phash → umbral para el algoritmo phash
  - Phash_s → umbral para el algoritmo phash simple
  - Dhash → umbral para el algoritmo dhash
  - Dhash → umbral para el algoritmo dhash vertical
  - Whash → umbral para el algoritmo whash

4.3.3 Programa principal

El programa principal, cuyo ejecutable es pruebas.py utiliza todas las funciones descritas anteriormente y el fichero de configuración para llevar a cabo el conjunto de pruebas diseñado.

Dicho conjunto o batería de pruebas está completamente automatizado: basta con que el usuario introduzca en el directorio "VIDEOS/ORIGINAL" los archivos de vídeo que desea analizar de forma previa a la ejecución del programa.

El diagrama de funcionamiento de este programa principal, a grandes rasgos, es el siguiente:


Inicio de la ejecución
Leer los parámetros del fichero de configuración
crear la estructura de directorios
Listar archivos de la carpeta VIDEOS/ORIGINAL
¿Se han analizado todos los vídeos?
Sí
No
Representar la pdf de los hashes de cada par "vídeo original vs vídeo original"
Comparar la pdf de la distancia de Hamming y un cierto umbral de probabilidad definido para el estudio de ruido y la comparación entre vídeos diferentes
almacenar gráficas
End
Añadir ruido al vídeo
almacenar archivos de vídeo
Reescalar el vídeo
Almacenar archivos de vídeo
Reducir el bitrate del vídeo
Almacenar archivos de vídeo
Obtener el hash de todos los fotogramas no negros de todos los vídeos almacenados anteriormente
Guardar hashes en formato .csv
Comparar los hashes del vídeo orignal con los de los vídeos modificados
Almacenar resultados en fichero .csv
Comparar todos los frames del vídeo original con el primer frame
Almacenar datos en fichero .csv
Calcular tiempo medio para la obtención de hash en cada algoritmo
Almacenar datos en fichero .txt
Obtener resumen (estadísticos) de los resultados
Almacenar datos en fichero .txt

Las librerías y módulos utilizados para apoyar este proyecto son las siguientes:

- ConfigParser: permite tanto la lectura como edición de archivos de configuración basados en secciones.
- Pandas: pandas es una librería construida como extensión de NumPy para manipulación y análisis de datos. En particular, ofrece estructuras de datos y operaciones para manipular tablas numéricas y series temporales.
- Matplotlib: es una librería para la generación de gráficos a partir de datos contenidos en listas o arrays en lenguaje Python y su extensión matemática NumPy. Proporciona una API, pylab, que cuenta con un diseño similar al de MATLAB.
- SciPy: es una librería libre y de código abierto. Contiene módulos para optimización, álgebra lineal, integración, interpolación, funciones especiales, FFT, procesamiento de señales y de imagen, resolución de ODEs y otras tareas para la ciencia e ingeniería.
- PIL: Python Imaging Library (PIL) es una librería gratuita que permite la edición de imágenes directamente desde Python. Soporta una variedad de formatos, incluidos los más utilizados como GIF, JPEG y PNG. Una gran parte del código está escrito en C, por cuestiones de rendimiento.
- Imagehash: librería que incorpora métodos de fingerprinting o hashing de imágenes
- Seaborn: Seaborn es una librería de visualización de datos de Python basada en matplotlib. Proporciona una interfaz de alto nivel para dibujar gráficos estadísticos atractivos e informativos.
- Glob: El módulo glob encuentra todos los nombres de la ruta especificada que coinciden con un patrón específico dado de acuerdo con las reglas utilizadas por el shell de Unix, aunque los resultados se devuelven en orden arbitrario.

5 IMPLEMENTACIÓN DE LA BATERÍA DE PRUEBAS

"Your first projects aren't the greatest things in the world, and they may have no money value, they may go nowhere, but that is how you learn - you put so much effort into making something right if it is for yourself".

Steve Wozniak

En este punto se va a detallar la implementación de la batería de pruebas describiendo, paso por paso, los resultados de la ejecución de cada uno de los módulos diseñados.

Para ello, se va a utilizar como ejemplo un vídeo publicitario del NUEVO MINI correspondiente al año 2014. Incluyendo este, los spots publicitarios utilizados para modelar los algoritmos son los siguientes:

- Actimel [13]
- Calzedonia [14]
- El corte inglés
- Nuevo MINI [15]
- Vitaldent

El escalado que se va a aplicar a los cuadros de vídeo previamente a la obtención del hash es de 32x32, con lo cual los hashes tendrán 1024 bits. Esto se traduce en 2^{1024} posibles combinaciones, que es un número enormemente grande (unos 309 dígitos), más que suficiente para las comparaciones que se van a realizar.

Los puntos que siguen a continuación pretenden describir, paso por paso, los resultados de la implementación del programa principal realizado.

5.1 Conceptos previos

Dado que el contenido que se va a analizar son anuncios de televisión, se ha tenido en cuenta que dicho tipo de vídeos tiene tanto al inicio como intercalados una serie de fotogramas negros. Estos fotogramas se han de eliminar de forma previa al análisis para realizar las pruebas únicamente sobre los fotogramas reales, que son los que resultan de interés.

Para determinar si un fotograma es negro o no se ha seguido el siguiente criterio:

- Un píxel se considera "negro" si, de entre los 256 niveles de gris posibles, tiene un valor igual o inferior a 18.
- Una vez contabilizados los píxeles "negros", si dicho número de píxeles supera el 92% del total, el frame se considera negro y se excluye del estudio.

5.2 Modificación de los vídeos de prueba

En primer lugar, se aplican las modificaciones descritas en puntos previos a los vídeos situados en el directorio correspondiente: adición de ruido, reducción de bitrate y reescalado.

5.2.1 Adición de ruido

La adición de ruido se realiza mediante la herramienta ffmpeg con el siguiente comando:

```
ffmpeg -y -i [video_origen] -vf noise=alls=25:allf=t -vcodec libx264 [ruta_destino]
```

El resultado de aplicar este comando es el siguiente:

Figura 26: imagen original

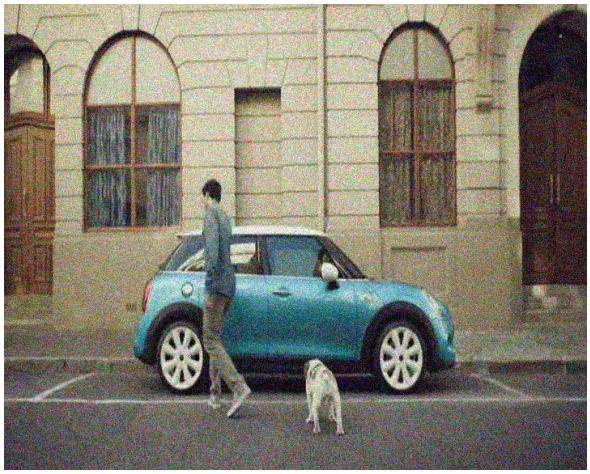


Figura 25: imagen con ruido

Tal como puede observarse, la imagen se ve distorsionada a causa de las interferencias provocadas por el ruido añadido.

Según el método ahash, por ejemplo, los hashes obtenidos para estos dos cuadros serían los siguientes:

- Vídeo original:
  ffffffffcfffffcf83dfffad83fffffd07ffffff0fffff870fff9f850307ff870107ff8701
  07ff870107ff870107ff870106ff870107ff870107ff87010fffe01fec3ff01fad1fe01f
  e5cfc01fe7ffe01fe7fb801f000180010004c0006006fc006006000040460020004010004b
  8e8000000000000000000000000000000000
- Vídeo ruidoso:

  ffffffffcfffffcf83dfffad83fffffd07ffffff0fffff870fff9f850307ff870107ff8701
  07ff870107ff870107ff870106ff870107ff870107ff87011fffe01fec3ff01fad1fe01f
  e7cfc01fe7ffc01fe7fbc01f000180014004c0006006fc006006000040460020004010002b
  ce0000000000000000000000000000000000

El porcentaje de similitud entre los bits de ambos hashes es del 99,12109375%.

5.2.2 Reescalado

Este análisis consiste en reducir el tamaño de los cuadros del vídeo para posteriormente volverlo a ampliar, recuperando la proporción y escala originales.

Esta transformación también se realiza mediante el uso de la herramienta ffmpeg, ejecutando los siguientes comandos de forma sucesiva:

```
ffmpeg -y -i [video_original] -filter:v scale={}:-1 -c:a copy [archivo temporal]
ffmpeg -y -i [archivo temporal] -filter:v scale={}:{} -c:a copy [video modificado]
```

El resultado de aplicar esta modificación es el siguiente:

Figura 28: imagen original

Figura 27: imagen reescalada

El resultado de este análisis es la pérdida de resolución por parte de la imagen.

Al comparar los hashes obtenidos por el método de average hashing, se obtiene una coincidencia del 99,4140625% en los bits de ambos.

5.2.3 Reducción de la tasa de bit

Este tercer análisis se realiza, de nuevo, gracias a la herramienta ffmpeg.

El comando utilizado para este propósito es el siguiente:

```
ffmpeg -y -i [video original] -c:v libx264 -b:v 700k -an [video modificado]
```

El resultado de aplicar dicho comando sobre los fotogramas del vídeo original tiene el siguiente efecto:

Figura 29: imagen original

Figura 30: imagen con bitrate reducido

El porcentaje de parecido entre los hashes de ambas imágenes, según el método ahash, es del 99,609375%.

5.2.4 Almacenamiento de resultados

Los hashes se almacenan en ficheros de extensión .csv con el siguiente formato:

Número de fotograma no negro	ahash	phash	phash_s	dhash	dhash_v	whash
0	ffde7…0a20	83b5…02a25	2acf0…8c2267	8b18e…0000	d1fe7…ff00	ffde7f…000
1	ffde7…00000	83b5…9b02b	2acf0e…c2267	9b18e…2000	d1fe7b…1ff00	ffde7…0000
…	….	….	….	….	…	….
248	afc2b…4456f	05ac2…dfe44	23ddc…78363a	0234…abbc42	9263cc3…abb	a142d…aac3
249	83b5…02a25	9b18e…2000	d1fe7b…1ff00	ffde7…0000	afc2b…4456f	83b5…9b02b

Tabla 16: Hashes de los fotogramas de un vídeo

El resultado de comparar los hashes del vídeo original con los correspondientes a cada una de las modificaciones anteriores se almacena en ficheros de extensión .csv con la siguiente estructura:

Número de fotograma no negro	ahash	phash	phash_s	dhash	dhash_v	whash
0	0.9912109375	0.984375	0.9873046875	0.955078125	0.9599609375	0.99609375
1	0.994140625	0.982421875	0.9775390625	0.9599609375	0.9599609375	0.9921875
2	0.994140625	0.986328125	0.9814453125	0.96875	0.9609375	0.9921875
…	….	….	….	….	…	….
248	0.990234375	0.97265625	0.9765625	0.9609375	0.9423828125	0.990234375
249	0.9951171875	0.978515625	0.982421875	0.958984375	0.9345703125	0.986328125

Tabla 17: Resultados comparativa modificaciones

Tal como puede observarse, los ficheros csv estarán compuestos por siete columnas: la primera de ellas identificará el fotograma al que se refieren los datos y las seis columnas restantes se corresponderán una-una con los métodos de hashing. Los valores de la tabla serán el porcentaje de parecido, en decimal, entre el hash de un mismo fotograma del vídeo original y el del vídeo modificado en cuestión.

5.3 Detección del primer fotograma "significativamente" diferente al primero

El objetivo de este estudio es obtener el índice del primer fotograma de un vídeo cuyo porcentaje de parecido con el hash del fotograma inicial es inferior a un cierto umbral dado. Dicho umbral se ha fijado en 0.85 para todos los métodos de hashing y uno de los objetivos de este trabajo será ajustarlo posteriormente en base a los resultados obtenidos.

La principal aplicación de este análisis es comprobar la sensibilidad de cada uno de los métodos ante pequeñas variaciones en la escena.

El resultado se almacena en un fichero de extensión .csv similar al de la Tabla 14, cuyos valores recogen el porcentaje de parecido (en decimal) entre cada fotograma y el primer fotograma no negro del vídeo en cuestión.

Número de fotograma comparado	ahash	phash	phash_s	dhash	dhash_v	whash
1	0.9912109375	0.962890625	0.9833984375	0.96875	0.9736328125	0.99609375
2	0.984375	0.92578125	0.9833984375	0.9521484375	0.958984375	0.986328125
…	….	….	….	….	…	….
248	0.64453125	0.51953125	0.5146484375	0.525390625	0.515625	0.650390625
249	0.6533203125	0.509765625	0.51171875	0.5400390625	0.5205078125	0.65234375

Tabla 18: Estructura fichero de comparación entre fotogramas

Una vez se tienen los resultados de la comparativa entre fotogramas, se evalúa en qué número de fotograma el porcentaje de parecido es inferior al umbral fijado. Dicho índice o número de fotograma se almacena en un fichero .txt a modo de resumen con la siguiente estructura:

RESULTADOS:

índice del primer fotograma con distancia de Hamming al fotograma incial superior al umbral dado:

ahash: 48

phash: 4

phash_s: 6

dhash: 6

dhash_v: 5

whash: 48

Figura 31: fichero ejemplo del resultado de la detección de movimiento

Los datos del fichero de ejemplo anterior se corresponden con los resultados de analizar nuevamente el anuncio publicitario de MINI.

5.4 Comparativa entre los vídeos originales

Con el objetivo de evaluar la robustez de cada uno de los algoritmos de hashing, se comparan entre sí, fotograma a fotograma, todos los pares posibles de vídeos diferentes que se encuentren en el directorio "VIDEOS/original".

El resultado de cada comparativa se almacenará en un fichero de extensión .csv similar al de la Tabla 14.

Número de fotograma no negro	ahash	phash	phash_s	dhash	dhash_v	whash
0	0.4130859375	0.509765625	0.5390625	0.48828125	0.5	0.412109375
1	0.4169921875	0.48828125	0.5322265625	0.490234375	0.49609375	0.4140625
2	0.419921875	0.490234375	0.5390625	0.478515625	0.4990234375	0.416015625
…	….	….	….	….	…	….
248	0.4248046875	0.484375	0.486328125	0.5244140625	0.490234375	0.416015625
249	0.435546875	0.4921875	0.4794921875	0.515625	0.4951171875	0.421875

Tabla 19: comparativa entre los vídeos originales

5.5 Medición del coste computacional de cada algoritmo

El objetivo de este apartado de la implementación es promediar el tiempo que se toma cada algoritmo para construir el hash de un fotograma de vídeo.

Los resultados se almacenan en ficheros de extensión .txt, uno por cada vídeo original, con el objetivo de computar posteriormente una media global.

La estructura de dicho fichero es la siguiente:

RESULTADO: TIEMPO MEDIO DE CADA MÉTODO

ahash: 0.003376389994765773 s

phash: 0.0044962412580496535 s

phash simple: 0.004097095643631136 s

dhash: 0.0032057882559419883 s

dhash vertical: 0.0032623737348049177 s

whash: 0.03433906991875132 s

Figura 32: Fichero ejemplo resultado tiempos medios

Dicho fichero refleja, en segundos, el tiempo medio que necesita cada algoritmo para construir el hash de un fotograma. Los datos del fichero de ejemplo anterior se corresponden con los tiempos asociados a los fotogramas del anuncio publicitario de MINI.

5.6 Representar las funciones de densidad de probabilidad

Por último, se representan una serie de funciones de densidad de probabilidad. El objetivo de estas gráficas es modelar las distribuciones aleatorias por las que se rigen, aproximadamente, cada uno de los algoritmos en los diversos análisis realizados.

En el estudio que nos ocupa, se han realizado dos tipos de representaciones gráficas.

5.6.1 Funciones de densidad de probabilidad de la comparación entre hashes de vídeos diferentes

Las primeras representaciones gráficas que se llevan a cabo se corresponden con los análisis comparativos entre los hashes de cada par de vídeos originales, es decir, entre vídeos completamente diferentes. Se almacena, en formato png, una gráfica por cada análisis comparativo realizado. En cada una de estas gráficas se encuentran superpuestas las pdfs de los distintos algoritmos.

Un ejemplo de representación de este tipo es el siguiente, que corresponde al análisis comparativo entre los hashes del vídeo publicitario de MINI vs el vídeo publicitario de CALZEDONIA.

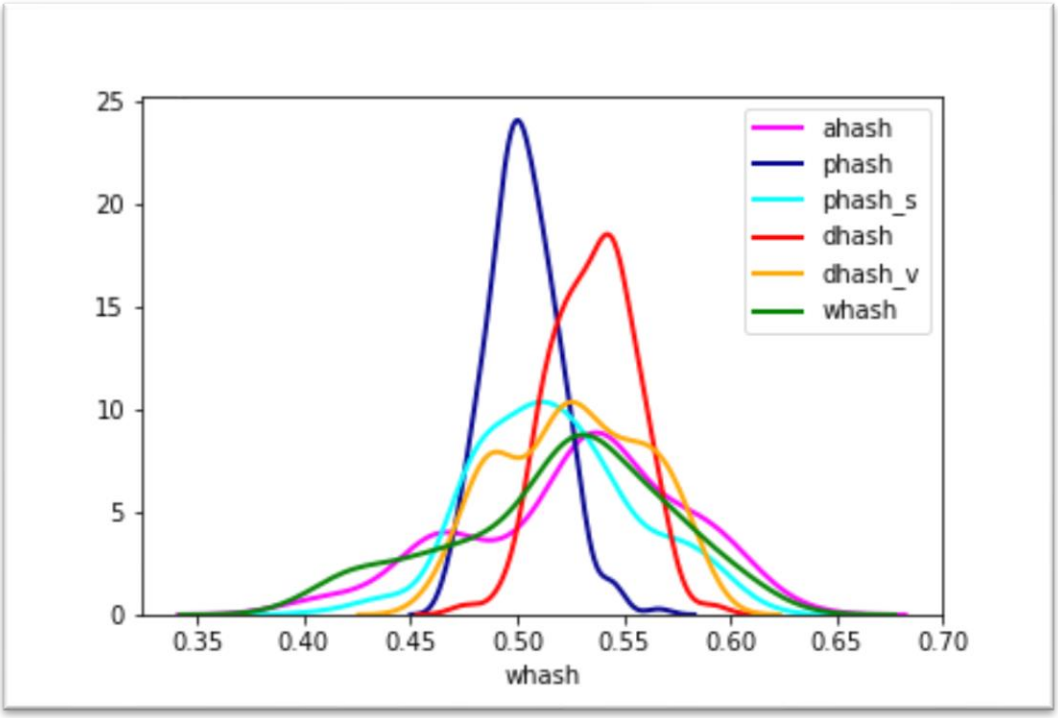


Figura 33: Gráfica de ejemplo de comparación entre vídeos diferentes

Estas representaciones no se corresponden con la distancia de Hamming, sino con su valor complementario; es decir, se representa el porcentaje de parecido que existe entre los fotogramas de ambos vídeos.

5.6.2 Funciones de densidad de probabilidad de la comparación de vídeos diferentes vs análisis de vídeos modificados y cálculo de umbrales

Este último análisis consiste en representar una gráfica por cada algoritmo aunando por un lado todos los datos recogidos en la comparativa de vídeos originales vs vídeos modificados, y por otro los datos de la comparativa entre vídeos originales diferentes. A diferencia del resto de análisis, no se tiene un resultado por cada comparación realizada, sino que se unen todos los datos recogidos para tratar de modelar el comportamiento global de cada algoritmo.

Una vez se tienen las dos funciones de probabilidad mencionadas se integran hasta un cierto valor con el propósito de definir umbrales para detectar tanto falsos positivos como falsos negativos en la comparación de hashes. Concretamente se busca:

- Establecer un umbral lo suficientemente alto en la comparación entre vídeos que han sido

modificados, (concretamente entre vídeos originales y el mismo vídeo al que se han aplicado las distintas modificaciones) para evitar que se produzcan falsos positivos en la comparación.

- Determinar un umbral lo suficientemente bajo en la distancia de Hamming para detectar vídeos diferentes, evitando que se produzcan falsos negativos en las comparativas.

Para determinar estos umbrales se han establecido dos límites de integración:

- El primero consiste en establecer el primer umbral en el lugar en que la integral de la primera función de probabilidad alcance el valor 0.9999
- El segundo establece el umbral en el punto en que la integral de la segunda función de probabilidad toma el valor 0.00009

A continuación, se muestra una gráfica de ejemplo, que se corresponde con el resultado obtenido para el algoritmo dhash:

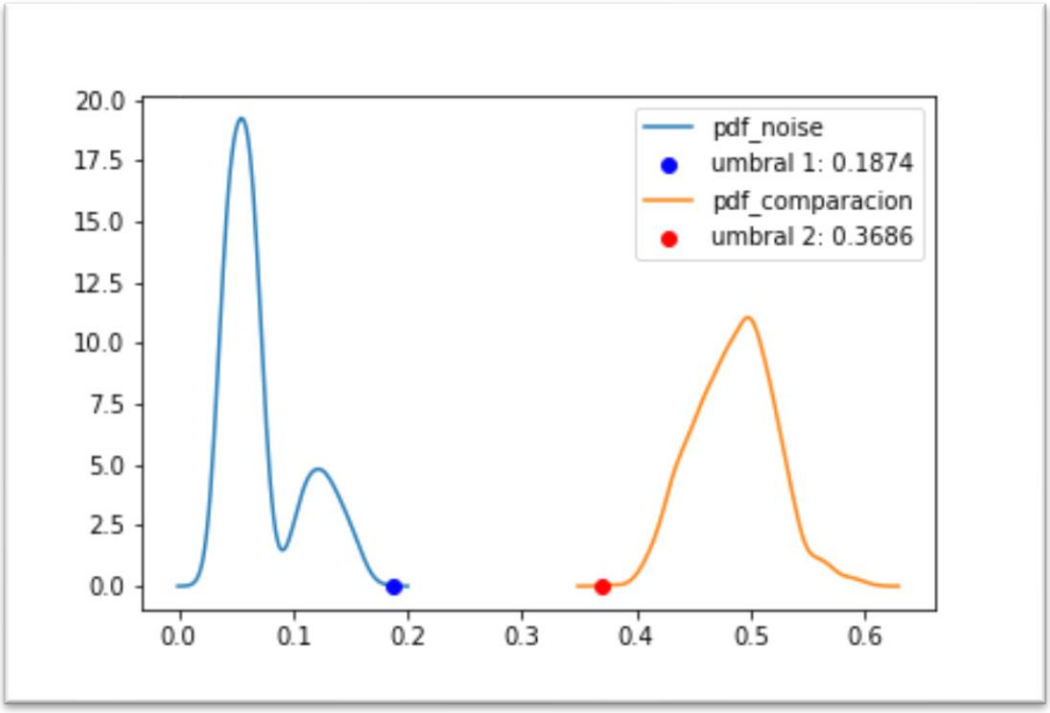


Figura 34: Ejemplo de modelado del algoritmo dhash y evaluación de umbrales

6 RESULTADOS

"Sometimes when you innovate, you make mistakes.
It is best to admit them quickly, and get on with
improving your other innovations".

Steve Jobs

Una vez realizadas todas las pruebas detalladas anteriormente, hay que analizar los resultados. Dichos resultados se encuentran adjuntos al completo en los anexos. Por lo que las líneas que siguen se limitarán a describir dichos resultados con el objetivo de comparar los métodos entre sí.

Para la realización de esta batería de pruebas se han empleado archivos de vídeo diferentes a los utilizados para parametrizar las funciones, correspondientes a spots publicitarios:

- Aegon
- Densia
- Giorgi
- Pediasure

A continuación, se muestran los resultados obtenidos para cada uno de los estudios realizados sobre estos vídeos.

6.1 Umbrales para cada algoritmo

Para comprobar el éxito de cada algoritmo en la detección de similitudes tras aplicar modificaciones sobre los vídeos originales se ha tenido en cuenta un umbral particular para la distancia de Hamming en cada uno de los algoritmos. Dichos umbrales, obtenidos a partir de las pdf de los algoritmos según el método descrito anteriormente, son los siguientes:

- Umbral para ahash: 8,50%
- Umbral para phash: 9,36%
- Umbral para phash simple: 64,04%
- Umbral para dhash: 18,59%
- Umbral para dhash vertical: 19,33%
- Umbral para whash: 7,86%

Estos umbrales representan el porcentaje máximo de diferencia entre dos hashes para considerar que corresponden al mismo fotograma modificado; es decir, si el porcentaje de diferencias entre dos hashes es inferior a dicho umbral, dichos hashes corresponden a dos fotogramas similares.

La representación gráfica de las funciones densidad de probabilidad de las que proceden estos umbrales se encuentran en el **Anexo A**.

6.2 Análisis de adición de ruido

Inicialmente se empleaba el mismo umbral para comparar todos los métodos, sin embargo, tal como se ha comprobado al representar la pdf de cada uno de los algoritmos cada uno de ellos tiene una sensibilidad distinta. Los resultados completos correspondientes a este análisis se encuentran en el **Anexo B**.

Los resultados de comprobar la similitud entre los archivos de vídeo tras la adición de ruido, empleando los umbrales obtenidos anteriormente, son los siguientes:

- ahash: 100%
- phash: 99,91%
- phash simple: 100%
- dhash: 94,27%
- dhash vertical: 94,63%
- whash: 100%

Si se comparan estos porcentajes entre sí, se obtiene el porcentaje de éxito de cada uno de los algoritmos relativo al éxito total:

- ahash: 16,98%
- phash: 16,97%
- phash simple: 16,98%
- dhash: 16,02%
- dhash vertical: 16,07%
- whash: 16,98%

6.3 Análisis de reescalado

El resultado de analizar la similitud entre los vídeos originales y los vídeos reescalados en base a los umbrales determinados por las funciones de densidad de probabilidad son los siguientes:

- ahash: 100%
- phash: 100%
- phash simple: 100%
- dhash: 100%
- dhash vertical: 100%
- whash: 100%

El éxito relativo al total de estos algoritmos tomaría los siguientes valores:

- ahash: 16,66%
- phash: 16,66%
- phash simple: 16,66%
- dhash: 16,66%
- dhash vertical: 16,66%
- whash: 16,66%

Salta a la vista que todos los algoritmos detectan exitosamente el reescalado de los archivos de vídeo.

6.4 Análisis de reducción de la tasa de bit

Siguiendo el mismo procedimiento que los análisis anteriores, se comparan entre sí los archivos originales y los modificados con la tasa de bit reducida, obteniendo los siguientes resultados:

- ahash: 100%
- phash: 100%
- phash simple: 100%
- dhash: 100%
- dhash vertical: 100%
- whash: 100%

Al comparar los algoritmos en base a sus propios umbrales, los resultados son los siguientes:

- ahash: 16,66%
- phash: 16,66%
- phash simple: 16,66%
- dhash: 16,66%
- dhash vertical: 16,66%
- whash: 16,66%

Al igual que en el caso anterior, los algoritmos detectan exitosamente la modificación.

6.5 Análisis de detección de vídeos diferentes

Los resultados de detectar vídeos diferentes en base a los umbrales calculados son los siguientes:

- ahash: 100%
- phash: 100%
- phash simple: 1,36%
- dhash: 100%
- dhash vertical: 100%
- whash: 100%

El éxito relativo de cada uno de los algoritmos se muestra a continuación:

- ahash: 19,95%
- phash: 19,95%
- phash simple: 0,25%
- dhash: 19,95%
- dhash vertical: 19,95%
- whash: 19,95%

6.6 Análisis de detección de movimiento

Los resultados medios de la detección del primer fotograma diferente al primero en las secuencias de vídeo, utilizando para todos los algoritmos el mismo umbral predefinido, con valor 0.85, son:

- ahash: 45.4
- phash: 8.2
- phash simple: 8.4
- dhash: 17.2
- dhash vertical: 21.4
- whash: 40.8

El éxito de cada algoritmo empleando los umbrales personalizados calculados es:

- ahash: 16.8
- phash: 4.8
- phash simple: no es capaz de detectarlo
- dhash: 26.6
- dhash vertical: 33.4
- whash: 13.4

6.7 Coste computacional

El coste computacional asociado a cada algoritmo, es decir, el tiempo que necesita cada uno de ellos para obtener una huella de 32x32 bits es el siguiente:

- ahash: 0.24 milisegundos
- phash: 0.31 milisegundos
- phash simple: 0.28 milisegundos
- dhash: 0.22 milisegundos
- dhash vertical: 0.22 milisegundos
- whash: 2.34 milisegundos

6.8 Resumen de los resultados

Una vez vistos los resultados obtenidos, parece interesante resumirlos en una única tabla comparativa de modo que pueda apreciarse de forma global el funcionamiento de cada uno de los algoritmos y así distinguir cuáles de ellos ofrecen las prestaciones más atractivas.

A continuación, se muestra la tabla de propiedades de los algoritmos, donde los subapartados total y relativo se refieren a los porcentajes de éxito de cada método por separado y a los porcentajes de éxito de cada algoritmo sobre el éxito total conseguido por los seis algoritmos, respectivamente:

		Ahash	Phash	Phash simple	Dhash	Dhash vertical	whash
Umbral para la distancia de Hamming en la detección de similitudes (%)		8,50	9,36	64,04	18,59	19,33	7,86
Coste computacional (ms)		0.24	0.31	0.28	0.22	0.22	2.34
Ruido (%)	Total	100	99.91	100	94.27	94.63	100
	Relativo	16.98	16.97	16.98	16.02	16.07	16.98
Tasa de bit (%)	Total	100	100	100	100	100	100
	Relativo	16.66	16.66	16.66	16.66	16.66	16.66
Reescalado (%)	Total	100	100	100	100	100	100
	Relativo	16.66	16.66	16.66	16.66	16.66	16.66
Vídeos diferentes (%)	Total	100	100	1.36	100	100	100
	Relativo	19.95	19.95	0.25	19.95	19.95	19.95
Índice medio de detección de movimiento		16.8	4.8	inf	26.6	33.4	13.4

Tabla 20: Resumen de los resultados obtenidos

Tras analizar los resultados obtenidos, reflejados en forma resumida en la tabla final del punto anterior, es el turno de evaluar cuál o cuáles de los algoritmos ofrecen las prestaciones más interesantes.

En primer lugar, el valor del umbral para la detección de fotogramas similares es un buen indicador de la sensibilidad de cada algoritmo. Dicho umbral toma un valor relativamente bajo, en torno al 10%, para los algoritmos ahash, phash y dhash, un valor en torno al 20% para las variantes del algoritmo dhash, y un valor superior al 60% para el algoritmo phash simple. Esto excluye inmediatamente a este último, el algoritmo phash simple, de entre los posibles candidatos pues al tener un umbral tan alto es propenso a ofrecer falsos negativos en las comparaciones entre vídeos similares, además de que es incapaz de reconocer el movimiento en la mayoría de los casos.

Una vez descartado el algoritmo phash simple, quedan los algoritmos ahash, phash, dhash, dhash vertical y whash.

El coste computacional es similar en todos ellos, en torno a 0.2-3 milisegundos, excepto en el algoritmo ahash, que tiene un coste computacional de unos 2.3 milisegundos, es decir, casi 10 veces superior al de los demás. Este hecho hace del algoritmo whash el segundo en ser descartado, pese a sus buenos resultados en las comparaciones.

Los cuatro algoritmos restantes (ahash, phash, dhash y dhash vertical) ofrecen prestaciones muy altas a la hora de descartar vídeos diferentes y de detectar vídeos modificados. Sin embargo, en base a los análisis de detección de ruido y de detección de movimiento, destacan por encima de los demás los algoritmos ahash y phash al obtener casi un 100% de precisión en las pruebas de ruido realizadas y por tener un índice de detección de movimiento bastante inferior al de los demás algoritmos.

7 Conclusiones y Posibles Líneas de Futuro

"The science of today is the technology of tomorrow".

Edward Teller

Una vez finalizado el estudio y habiendo evaluado y comparado entre sí los algoritmos, solo resta comentar los resultados obtenidos y concluir el documento aportando ideas o propuestas de mejora así como posibles líneas futuras de investigación.

7.1 Conclusiones

En base a los valores obtenidos en los diferentes análisis, los resultados son concluyentes. Los algoritmos que ofrecen las mejores prestaciones en cuanto al coste computacional y la baja probabilidad de arrojar falsos negativos en la detección de vídeos similares son los algoritmos average hash y perceptual hash, seguidos muy de cerca por las variantes del algoritmo difference hash, tal como recoge la siguiente tabla:

	Ahash	Phash	Dhash	Dhash vertical
Umbral para la distancia de Hamming en la detección de similitudes (%)	8,50	9,36	18,59	19,33
Distancia entre el umbral de detección de similitudes y el de detección de diferencias (%)	17.68	33.8	18.27	20.90
Coste computacional (ms)	0.24	0.31	0.22	0.22
Ruido (%)	100	99.91	94.27	94.63
Tasa de bit (%)	100	100	100	100
Reescalado (%)	100	100	100	100
Vídeos diferentes (%)	100	100	100	100
Índice medio de detección de movimiento	16.8	4.8	26.6	33.4

Tabla 21: Prestaciones algoritmos escogidos

Observando esta tabla de resultados se concluye que, si el objetivo perseguido es la precisión del algoritmo, los candidatos más adecuados serían average hash y perceptual hash, ya que son los que poseen el umbral de detección más bajo.

Por otro lado, aunque las diferencias entre el coste computacional entre estos cuatro algoritmos son casi imperceptibles, el asociado al algoritmo phash es casi una décima de milisegundo superior al de los demás.

En todos los algoritmos, la distancia entre los umbrales calculados para la detección de vídeos similares (para evitar falsos positivos) y la detección de vídeos diferentes (para evitar falsos negativos) es suficientemente grande como para que no exista solapamiento entre las curvas de las funciones de densidad, es decir, que si se toma como umbral para todas las comparaciones el primero de ellos, los resultados obtenidos serán exitosos,

Por tanto, en base a estas premisas, el algoritmo **ahash** resulta ser el más preciso sin verse penalizado por el coste computacional.

7.2 Posibles líneas de futuro

Pese a que se han realizado análisis relativos a las alteraciones más frecuentes sufridas por los archivos de vídeo, podría ampliarse este estudio realizando análisis más exhaustivos y profundos, añadiendo pruebas como la susceptibilidad a pequeños giros en los fotogramas del vídeo (por ejemplo, si se tratara de un vídeo grabado en 'screener'), o modificaciones del encuadre o de zoom, entre otras.

Por otro lado, otras líneas de continuación serían: incluir algún otro algoritmo de hashing al estudio, aplicar el estudio a otra librería de fingerprinting o a otro lenguaje de programación, repetir el estudio con diferentes tamaños de hashing, etc.

Por último, sería interesante también aplicar los algoritmos escogidos a un sistema de detección real, como por ejemplo en la detección de anuncios de televisión en flujos de vídeo [1]. Esto permitiría evaluar su comportamiento en un entorno real y comprobar si realmente cumplen con las especificaciones y prestaciones que presumimos que tienen.

REFERENCIAS

[1] J. J. A. Armenteros, Sistema Inteligente de Reconocimientos de Anuncios de Televisión, 2016.

[2] M. ,. Z. K. ,. R. H. Li, «A VQ-Based Joint Fingerprinting and Decryption Scheme for Secure and Efficient Image Distribution,» 2018.

[3] «Haztutele: Telemadrid convierte al espectador en reportero, protagonista y productor,» [En línea]. Available: https://www.panoramaaudiovisual.com/2014/12/16/haztutele-telemadrid-convierte-al-espectador-en-reportero-protagonista-y-productor/.

[4] A. A. D. D. Allimes., Optimización de un sistema de Audio Fingerprinting para la detección de anuncios en tiempo real.

[5] A. Swaminathan, Y. Mao y M. Wu, «Robust and secure image hashing,» 05 June 2006. [En línea]. Available: https://ieeexplore.ieee.org/abstract/document/1634363.

[6] R. Venkatesan, S.-M. Koon, M. Jakubowski y P. Moulin, «Robust image hashing,» 2000. [En línea]. Available: https://ieeexplore.ieee.org/abstract/document/899541.

[7] S. M. K. Y. Reza Davarzani, «Perceptual image hashing using center-symmetric local binary patterns,» 2015. [En línea]. Available: https://link.springer.com/article/10.1007/s11042-015-2496-6.

[8] «Perceptual Hashing PHP,» [En línea]. Available: https://github.com/jenssegers/imagehash.

[9] «Anaconda,» [En línea]. Available: https://www.anaconda.com/distribution/.

[10] «Anaconda Navigator,» [En línea]. Available: https://anaconda.org/anaconda/anaconda-navigator.

[11] «Spyder,» [En línea]. Available: https://anaconda.org/anaconda/spyder.

[12] «FFmpeg,» [En línea]. Available: https://ffmpeg.org/about.html.

[13] «Spot publicitario Actimel,» [En línea]. Available: https://www.youtube.com/watch?v=mrag19kSnWs.

[14] «Spot publicitario Calzedonia,» [En línea]. Available: https://www.youtube.com/watch?v=DJ5K7hTK92A.

[15] «Spot Publicitario Nuevo MINI,» [En línea]. Available: https://www.youtube.com/watch?v=kTxoSAgjri4.

ANEXO A: RESULTADOS FUNCIONES DE DENSIDAD DE PROBABILIDAD

A continuación, se muestran las gráficas que modelan el comportamiento de cada uno de los algoritmos en la detección de similitudes entre vídeos modificados versus la detección de diferencias entre vídeos completamente distintos.

A.1) Average hash

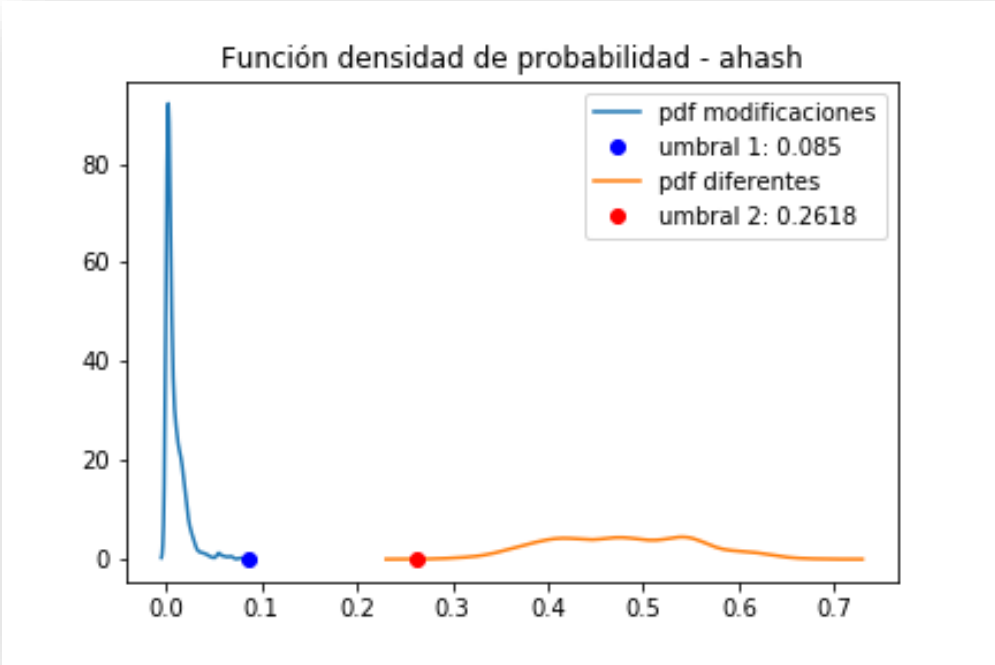


Figura 35: FDP Average Hash

A.2) Perceptual hash

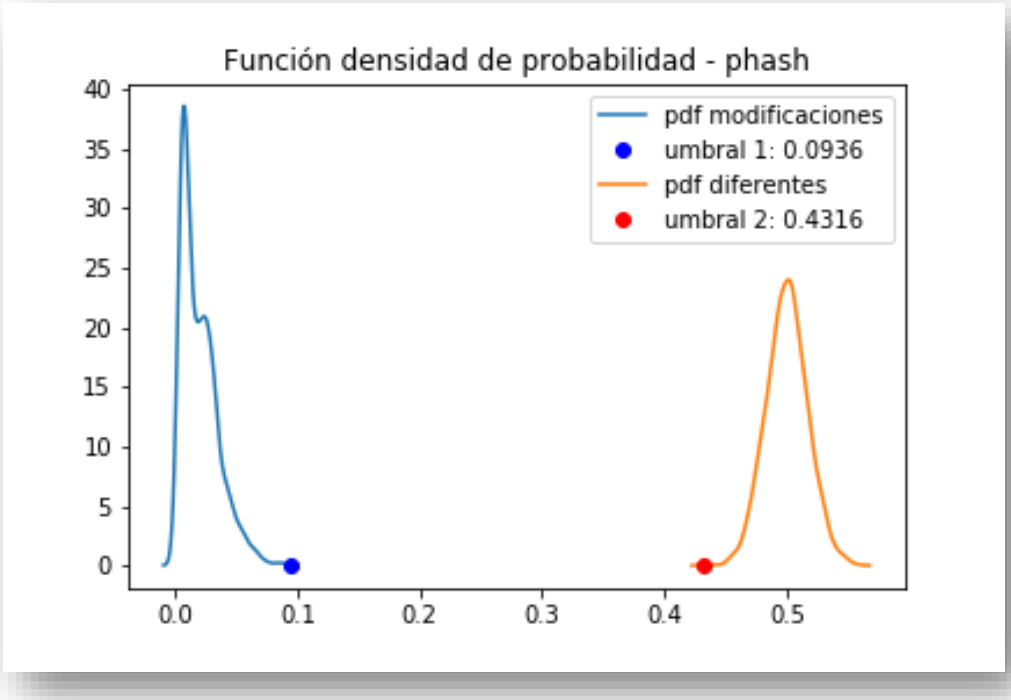


Figura 36: FDP Perceptual Hash

A.3) Perceptual hash simple

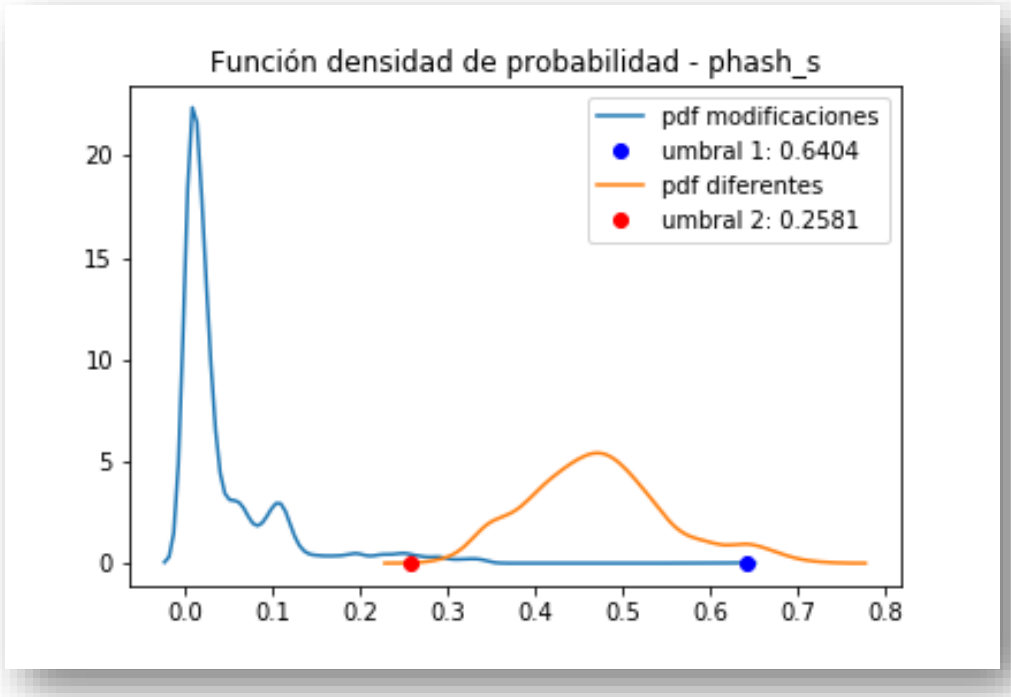


Figura 37: FDP Perceptual Hash Simple

A.4) Difference hash

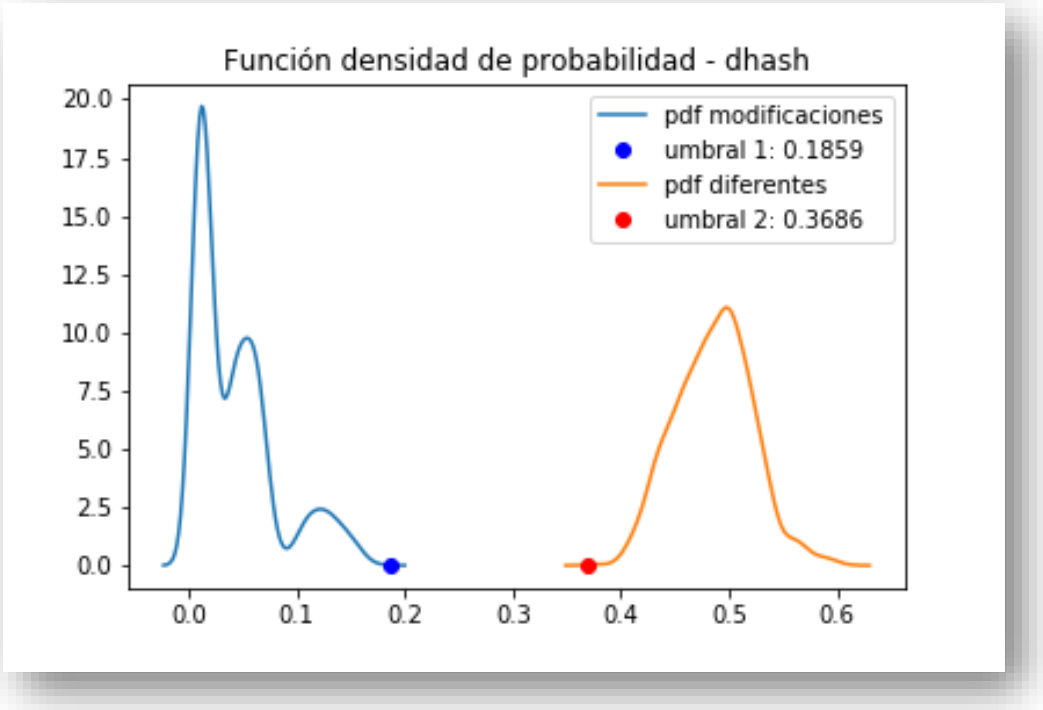


Figura 38: FDP Difference Hash

A.5) Difference hash vertical

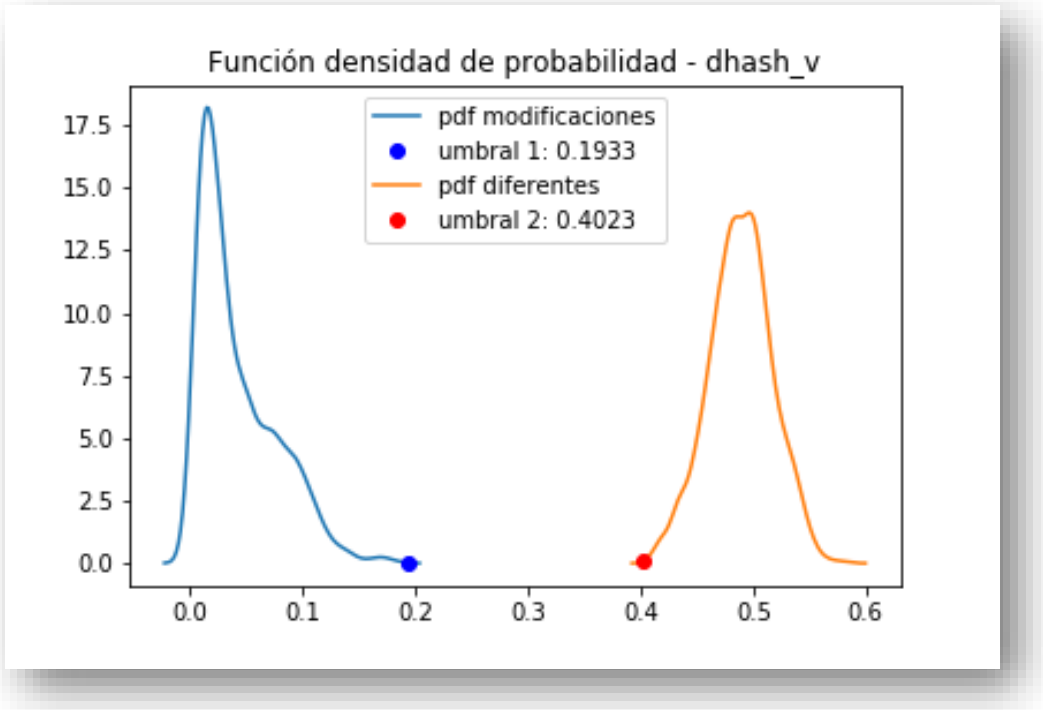


Figura 39: FDP Difference Hash Vertical

A.6) Wavelet hash

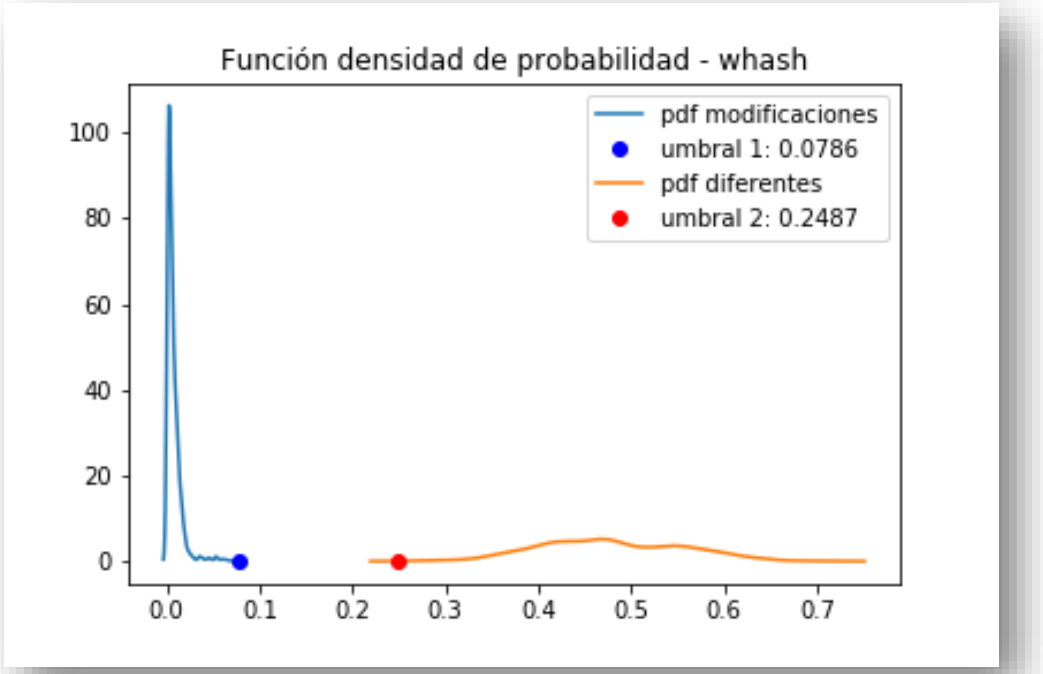


Figura 40: FDP Wavelet Hash

ANEXO B: RESULTADOS ANÁLISIS DE RUIDO

En cuanto al análisis de vídeos con ruido añadido, a continuación, se muestran los resultados en función de los distintos umbrales utilizados.

B.1) Mismo umbral para todos los algoritmos

En primer lugar, se analizaron los algoritmos bajo la misma premisa, utilizando un umbral de 0.85 como referencia para determinar si dos hashes correspondían al mismo fotograma modificado o no.

Los resultados obtenidos bajo estas condiciones fueron los siguientes:

RESUMEN DE LOS RESULTADOS: PROPORCIÓN DE ÉXITO DE CADA MÉTODO SEGÚN UMBRAL GENÉRICO: 0.15

- ahash: 1.0
- phash: 1.0
- phash simple: 0.8641344046749452
- dhash: 0.9744338933528123
- dhash vertical: 0.9861212563915267
- whash: 1.0

Tabla 22: Resultados ruido - umbral predefinido

B.2) Umbrales adaptados a cada algoritmo

Si se emplea un umbral adaptado a cada algoritmo en lugar del umbral genérico, los resultados cambian considerablemente:

RESUMEN DE LOS RESULTADOS: PROPORCIÓN DE ÉXITO DE CADA MÉTODO SEGÚN UMBRALES (mínimo valor)

- ahash: 1.0
- phash: 0.999104744852283
- phash simple: 1.0
- dhash: 0.9427036705461056
- dhash vertical: 0.946284691136974
- whash: 1.0

Tabla 23: Resultados ruido - umbral personalizado

ANEXO C: RESULTADOS ANÁLISIS DE REESCALADO

Si se reduce la escala de los fotogramas de los archivos de vídeo para luego devolverlas a su tamaño original y compararlos con los del vídeo original, se obtienen los siguientes resultados en función del umbral escogido.

C.1) Mismo umbral para todos los algoritmos

Al igual que en el caso anterior, en primer lugar, se comparan los algoritmos en base al mismo umbral, 0.15, obteniendo los siguientes resultados:

RESUMEN DE LOS RESULTADOS: PROPORCIÓN DE ÉXITO DE CADA MÉTODO SEGÚN UMBRAL GENÉRICO: 0.15

- ahash: 1.0
- phash: 1.0
- phash simple: 0.9948867786705624
- dhash: 1.0
- dhash vertical: 1.0
- whash: 1.0

Tabla 24: Resultados reescalado - umbral predefinido

C.2) Umbrales adaptados a cada algoritmo

Al cambiar este umbral general por un umbral adaptado a cada algoritmo, los resultados son los siguientes

RESUMEN DE LOS RESULTADOS: PROPORCIÓN DE ÉXITO DE CADA MÉTODO SEGÚN UMBRALES RELATIVO A LOS DEMÁS

- ahash: 1.0
- phash: 1.0
- phash simple: 1.0
- dhash: 1.0
- dhash vertical: 1.0
- whash: 1.0

Tabla 25: Resultados reescalado - umbral personalizado

ANEXO D: RESULTADOS ANÁLISIS DE REDUCCIÓN DE TASA DE BIT

A continuación, se muestran los resultados correspondientes a la reducción de la tasa de bit de los archivos de vídeo.

D.1) Mismo umbral para todos los algoritmos

Utilizando el mismo umbral, distancia de Hamming inferior a 0.15 para todos los algoritmos, se obtienen los resultados que siguen:

RESUMEN DE LOS RESULTADOS: PROPORCIÓN DE ÉXITO DE CADA MÉTODO SEGÚN UMBRAL GENÉRICO: 0.15

- ahash: 1.0
- phash: 1.0
- phash simple: 0.9992695398100804
- dhash: 1.0
- dhash vertical: 1.0
- whash: 1.0

Tabla 26: Resultados reducción tasa de bit - umbral predefinido

D.2) Umbrales adaptados a cada algoritmo

Al emplear un umbral personalizado para cada algoritmo, se obtienen resultados más satisfactorios:

RESUMEN DE LOS RESULTADOS: PROPORCIÓN DE ÉXITO DE CADA MÉTODO SEGÚN UMBRALES (mínimo valor)

- ahash: 1.0
- phash: 1.0
- phash simple: 1.0
- dhash: 1.0
- dhash vertical: 1.0
- whash: 1.0

Tabla 27: Resultados reducción tasa de bit - umbral personalizado

ANEXO E: RESULTADOS ANÁLISIS DE DETECCIÓN DE MOVIMIENTO

Por último, se estudia la capacidad de cada algoritmo para detectar el movimiento. Los resultados derivados de usar tanto umbrales predefinidos como personalizados son los siguientes.

E.1) Mismo umbral para todos los algoritmos

Si se utiliza un umbral de 0.85 como referencia para determinar si dos hashes son lo suficientemente diferentes entre sí, los resultados obtenidos son:

RESUMEN DE LOS RESULTADOS: PROPORCIÓN DE ÉXITO DE CADA MÉTODO SEGÚN UMBRAL GENÉRICO: 0.15

- ahash: 1.0
- phash: 1.0
- phash simple: 0.03964925657643919
- dhash: 1.0
- dhash vertical: 1.0
- whash: 1.0

Tabla 28: Resultados detección movimiento - umbral predefinido

E.2) Umbrales adaptados a cada algoritmo

Si se emplea un umbral adaptado a cada algoritmo en lugar del umbral genérico, los resultados cambian levemente:

RESUMEN DE LOS RESULTADOS: PROPORCIÓN DE ÉXITO DE CADA MÉTODO SEGÚN UMBRALES PERSONALIZADOS

- ahash: 1.0
- phash: 1.0
- phash simple: 0.013605442176870748
- dhash: 1.0
- dhash vertical: 1.0
- whash: 1.0

Tabla 29: Resultados detección movimiento - umbral personalizado