

Universidad Politécnica de Madrid
Escuela Técnica Superior de Ingenieros Industriales
Departamento de Automática, Ingeniería Electrónica e Informática Industrial

Máster Universitario en Electrónica Industrial

**Compresión de imágenes
optimizada en consumo energético
para redes inalámbricas**

Autor: David Aledo Ortega

Tutor: Félix Moreno

Febrero de 2013



Trabajo Fin de Máster



Agradecimientos

Doy las gracias a mis amigos y compañeros, en especial a aquellos que aguantaron mis “optimizaciones” sin llegar a matarme...

Índice

Introducción y estado del arte:.....	9
I. Introducción	9
II. WWSN	9
II.1 Transmisión	11
II.2 Codificación.....	12
II.3 Cobertura en WWSN	16
III. Compresión de imágenes.....	18
III.1 JPEG	19
III.2 Transformación del espacio de color.....	20
III.3 DCT	21
III.4 DLMT	26
III.5 Cuantificación y lectura en zigzag	27
III.6 Compresión sin pérdidas	29
III.7 Redes Neuronales Artificiales	31
III.8 Compresión de imágenes mediante redes neuronales.....	33
Diseño del compresor tipo JPEG	39
I. Modificaciones a JPEG propuestas para reducir el consumo energético.....	39
II. Códigos Matlab para optimización de algoritmos	40
III. Conclusiones sobre la DLMT.....	42
IV. Descripción del sistema implementado:.....	47
IV.1 Sistema de pruebas en Virtex5.....	47
IV.2 Opciones parametrizables para optimizar el consumo energético.....	53
Propuesta de red neuronal distribuida en una WWSN.....	57
Resultados, conclusiones y líneas futuras.....	65
I. Resultados	65
II. Conclusiones.....	74
III. Líneas futuras	75
REFERENCIAS	79
ANEXO I: Simulaciones de las redes neuronales	83

Introducción y estado del arte

Introducción y estado del arte:

I. Introducción

Los compresores de imágenes y vídeo generalmente están diseñados para comprimir una sola vez y después descomprimir muchas veces (cada vez que se desea ver la imagen o vídeo). Por lo tanto, por regla general, los compresores son más complejos, lentos y consumen mucha más energía que los descompresores.

Sin embargo, en las redes de sensores inalámbricas de vídeo (WVSN: Wireless Visual Sensor Networks) la situación es la opuesta. Hay varios o muchos pequeños nodos con cámara alimentados por baterías. Cada nodo debe transmitir por radio la información generada por su cámara, pero enviar tanta información por radiofrecuencia consume demasiada energía. Las baterías durarían muy pocos días o incluso horas. Es necesario comprimir la información antes de transmitirla. Pero se ha de tener cuidado de no gastar más energía en comprimir que la que se utilizaría en enviar la diferencia entre la información sin comprimir y comprimida. Por lo que las WVSN requieren de compresores de bajo consumo. Mientras que la descompresión se realiza una única vez en la estación central de la red, que dispone de mayores recursos de energía y de computación.

En este proyecto se proponen, analizan y desarrollan (a distintos niveles) tres métodos diferentes con algunos puntos en común. Primero se desarrolla a fondo una serie de modificaciones de JPEG (que es uno de los métodos más ampliamente utilizados) basado en la transformada discreta del coseno (DCT: Discrete Cosine Transform). Después se analiza un compresor similar, pero basado en la transformada discreta López-Moreno (DLMT: Discrete Lopez-Moreno Transform). Y por último se propone el empleo de Redes Neuronales distribuidas en la red de sensores.

II. WVSN

Wireless Video Sensor Network (WVSN), también conocidas como Wireless Visual Sensor Network (WVSN), Wireless Multimedia Sensor Network (WMSN), Visual Sensor Network (VSN), Video Sensor Network, Camera Sensor Network, etc. son una variante de las redes de sensores inalámbricas (WSN: Wireless Sensor Network) donde los sensores captan imágenes o vídeo. Las WSN típicamente recogen datos escalares como temperatura o humedad. Pero para algunas aplicaciones este tipo de datos pueden ser insuficientes, demandándose cada vez más información visual (imágenes y vídeo). Este nuevo tipo de datos en las WSN requiere mayor capacidad

de computación y mayor ancho de banda en la transmisión, confiriendo unas características, problemas y desafíos especiales a las WVSN.

WVSN son (o serán) utilizadas en [1] [2]:

- Monitorización de tráfico: nº de vehículos, aparcamientos, control de infracciones...
- Seguimiento de personas mayores, discapacitadas o enfermas.
- Monitorización de fauna salvaje.
- Control de seguridad en entornos peligrosos: como minas, centrales de energía, etc.
- Vigilancia de seguridad: en aeropuertos, estaciones, estadios y otros lugares públicos. También se utilizan para prevención/previsión y monitorización de incendios o terremotos.
- Seguimiento de tropas enemigas en batallas, localización de objetivos, etc.
- Realidades virtuales: para visitar museos u otros lugares de interés a distancia.

Los principales desafíos y problemas que presentan las WVSN son:

- El principal problema y desafío en las WVSN es el consumo de energía. El resto son derivados de este o solucionados procurando minimizar el consumo energético.
- Encontrar un buen equilibrio entre la calidad de la imagen, transmisión, etc. y el consumo energético.
- La codificación y procesado de imágenes y vídeo es muchísimo más complejo que el realizado en el resto de WSN. Para reducir el volumen de datos a enviar por radiofrecuencia y por lo tanto reducir el consumo energético en la transmisión se requiere de compresión de datos. Pero la compresión consume energía también, pudiendo ser a veces incluso superior a la de transmisión (ya que la mayoría de protocolos de compresión fueron diseñados para sistemas con gran capacidad de computación sin restricciones energéticas).
- En la transmisión de datos hay que garantizar el suficiente ancho de banda para las aplicaciones en tiempo real, garantizar una cierta calidad minimizando los errores en la transmisión y a veces asegurar la privacidad de los datos. Las WSN convencionales, que transmiten datos escalares, se pueden permitir la redundancia de datos y los protocolos basados en *acknowledge*. Pero la gran densidad de datos a transmitir en WVSN y los altos requisitos temporales hacen inadecuados estos procedimientos en muchos casos [1].

- El problema de la cobertura: la captación de las cámaras es direccional y se ve afectada por obstáculos. El objetivo es cubrir toda el área deseada sin agujeros, minimizando el solapamiento (que se traduce en información redundante), siendo fiable aunque fallen sensores y cumpliendo los requisitos energéticos. El problema también se incrementa enormemente cuando se pasa de la cobertura en 2D a la cobertura en 3D.

A continuación se exponen las líneas de investigación que hay abiertas para resolver estos problemas y algunos ejemplos de sus soluciones.

II.1 Transmisión

ZigBee (IEEE 802.15.4), el protocolo utilizado por la mayoría de las WSNs, no es adecuado para transmitir vídeo [7]. Para lograr el ancho de banda necesario para transmitir imágenes o vídeo suele ser necesario utilizar Wi-Fi (IEEE 802.11), que para grandes volúmenes de datos incluso puede suponer un consumo menor que ZigBee según se muestra en el trabajo de la referencia [3]:

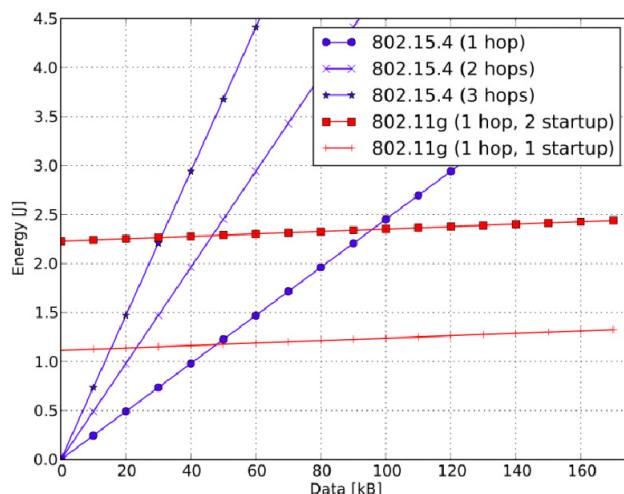


Figura 1. Comparación del consumo energético entre ZigBee y Wi-Fi.

El consumo de energía y el tiempo necesario para encender (o despertar de un modo de ahorro energético) en un módulo Wi-Fi son mucho mayores que en un módulo ZigBee. También son mayores las potencias de emisión, recepción y modo de ahorro en Wi-Fi que en ZigBee. Pero los tiempos necesarios para enviar cada bit son muchísimo menores en Wi-Fi, lo que implica una energía por bit también menor.

Una forma de reducir el consumo energético manteniendo el suficiente ancho de banda de transmisión para las imágenes o vídeos es utilizar dos módulos de comunicación en cada nodo. Uno de bajas prestaciones y consumo de energía como ZigBee

para baja tasa de datos y mensajes relacionados con el mantenimiento de la red y otro Wi-Fi para el envío de las grandes tasas de datos. La referencia antes mencionada [3] es un ejemplo de uno de los trabajos realizados en esta dirección.

Otra forma de ajustarse al ancho de banda disponible es transmitir sólo las imágenes que contengan información relevante. Estas redes se conocen como WVSN basadas en detección de eventos. La mayoría de los trabajos prácticos en WVSN son de este tipo, sobre todo los relacionados con vigilancia.

Otros problemas relacionados con la transmisión son los errores y pérdidas de datos. En el artículo [1] clasifican las técnicas de transmisión en WVSN en tres categorías:

- Las que solo consideran el envío de imagen o vídeo en un solo salto. Un ejemplo es la referencia [4] que utiliza la compresión basada en ondas (*wavelet transforms*) para crear una escalabilidad de paquetes de distinta resolución donde primero son enviados los paquetes de mayor prioridad y después si hay suficiente energía se van enviando los del resto de prioridades.
- Las que consideran múltiples saltos donde la estrategia se determina salto por salto. Un ejemplo es la referencia [5] donde en cada salto los datos son comprimidos y descomprimidos utilizando distintos rangos de compresión que son seleccionados en cada salto según las características de éste. Pero esta técnica no mejora la transmisión de extremo a extremo debido a los retardos e incrementos del consumo introducidos por cada compresión y descompresión.
- Las que realizan múltiples caminos de extremo a extremo. La mayoría combina códigos de corrección de errores y diversificación de los caminos para aumentar la fiabilidad. Dividen la información en pequeños paquetes que son enviados con una cierta redundancia por múltiples caminos. Sin embargo, estos algoritmos requieren que las probabilidades de éxito y los costes de transmisión sobre las rutas disponibles se conozcan a priori en el nodo de origen, lo que puede afectar a la aplicación práctica de estos mecanismos y a su rendimiento, especialmente en redes altamente dinámicas [1].

Y todavía queda por resolver los problemas de privacidad y autentificación de nodos para evitar ataques malintencionados a la red de sensores [6].

II.2 Codificación

La codificación del vídeo e imágenes es un aspecto importante a tener en cuenta en las redes visuales inalámbricas. Dicha codificación está sujeta a tres restricciones:

- Ancho de banda de la transmisión
- Consumo energético
- Capacidades de procesamiento del nodo

Es decir, se busca un algoritmo de compresión que permita comprimir el vídeo o las imágenes a un tamaño adecuado para la transmisión inalámbrica, pero que sea lo bastante sencillo computacionalmente como para poderse implementar en un sistema de recursos limitados con un consumo energético reducido, tratando de evitar una excesiva pérdida en la calidad de la imagen. Que es el objetivo de este proyecto.

También se puede reducir la cantidad de datos transmitida empleando otras técnicas como:

- Transmisión basada en eventos: Consiste en enviar datos únicamente cuando se detecta movimiento. Dicha detección se realiza en cada nodo, y hace que se active el sistema cuando los cambios entre fotogramas son lo bastante elevados. De esta forma se consigue reducir en gran medida el consumo energético, ya que la transmisión puede estar prácticamente desactivada mientras no hay movimiento, y se puede ahorrar ancho de banda en sistemas con multitud de nodos que transmitan por el mismo canal (a menos que todas las cámaras detecten movimiento simultáneamente, lo que haría que dicho canal se saturase). Este sistema es adecuado para sistemas de vigilancia.
- Agregación de imágenes: Cuando se tienen varias cámaras dirigidas hacia una misma zona, habrá partes de la imagen grabada que se solapen entre una cámara y otra. Agregando todas las imágenes en una sola, y comprimiendo y enviando dicha imagen, se evita la redundancia de enviar la parte solapada múltiples veces. También se puede emplear la correlación entre imágenes orientadas a una misma zona en lugar del simple solapamiento.

Consumo energético:

La selección de un codificador u otro está sujeta al contexto en que se va a emplear. Como ya se ha mencionado anteriormente, esto está limitado principalmente a dos factores: la tasa de bits que se puede transmitir y el consumo energético admisible.

Como puede verse en el siguiente gráfico, si se emplea el compresor MJPEG-2000 (cuyo consumo energético es relativamente bajo comparado con otros compresores), la mayor parte del consumo energético se produce en la codificación de vídeo [8].

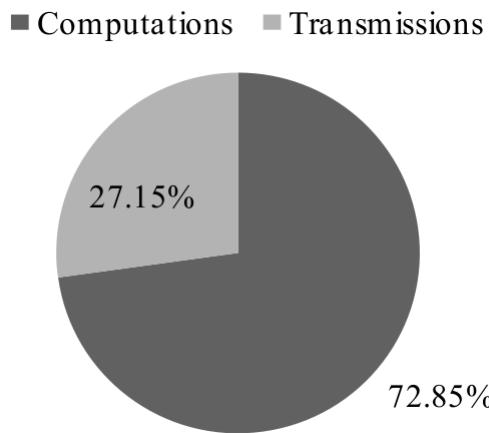


Figura 2. Comparación de los consumos de envío y compresión con un compresor MJPEG-2000.

En los siguientes gráficos, que comparan el compresor H.264 con MJPEG-2000, se puede comprobar que el consumo energético es aproximadamente proporcional al nivel de compresión, es decir, inversamente proporcional a la tasa de bits. Así, si se dispone de un ancho de banda prácticamente ilimitado y se quiere que el consumo energético sea mínimo, se utilizaría MJPEG-2000, mientras que si el interés principal es conservar el ancho de banda y no se está tan limitado en consumo energético sería mejor opción emplear H.264.

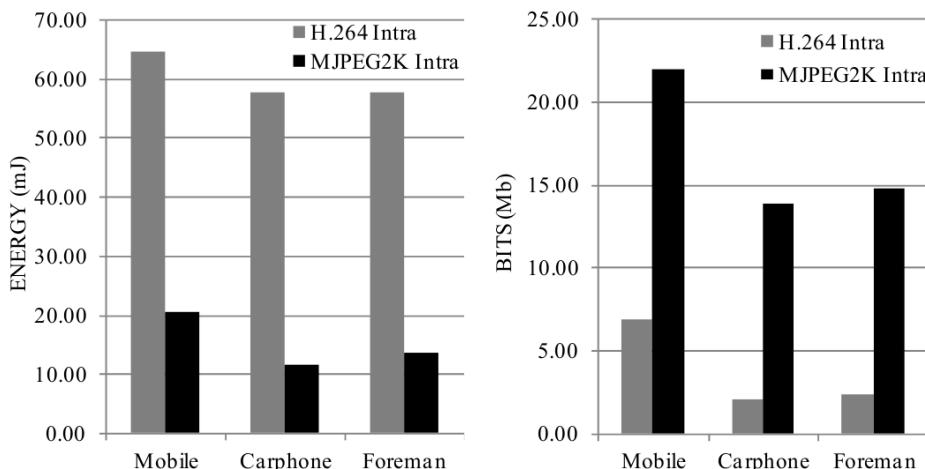


Figura 3. Gráficos comparativos entre H.264 y MJPEG

Estos datos han sido obtenidos en un sistema basado en un módulo TelosB, dotado de un módulo de comunicación 802.15.4 (ZigBee), y una plataforma Stargate, que posee un procesador Intel PXA255 de 400 MHz con un Linux embebido [8].

Ejemplo de compresión de vídeo en WVSN:

Como se ha dicho anteriormente, en redes inalámbricas, al existir ciertas limitaciones en el consumo energético y la capacidad de cómputo de los nodos, se requieren algoritmos de codificación distintos a los empleados habitualmente para la compresión de vídeo.

Una estrategia habitual es el empleo de codificación Wyner-Ziv [9]. Este modelo de codificación consiste en seleccionar sólo algunos de los fotogramas, denominados fotogramas clave, y comprimirlos con un algoritmo que dé buenas tasas de compresión, como puede ser JPEG-2000. El resto de los fotogramas (fotogramas Wyner-Ziv) se comprimirán con un algoritmo más sencillo aunque de peor calidad, como por ejemplo los basados en DCT. Esto supone una calidad inferior en los fotogramas Wyner-Ziv con respecto a los fotogramas clave.

Wyner-Ziv frame 'W'

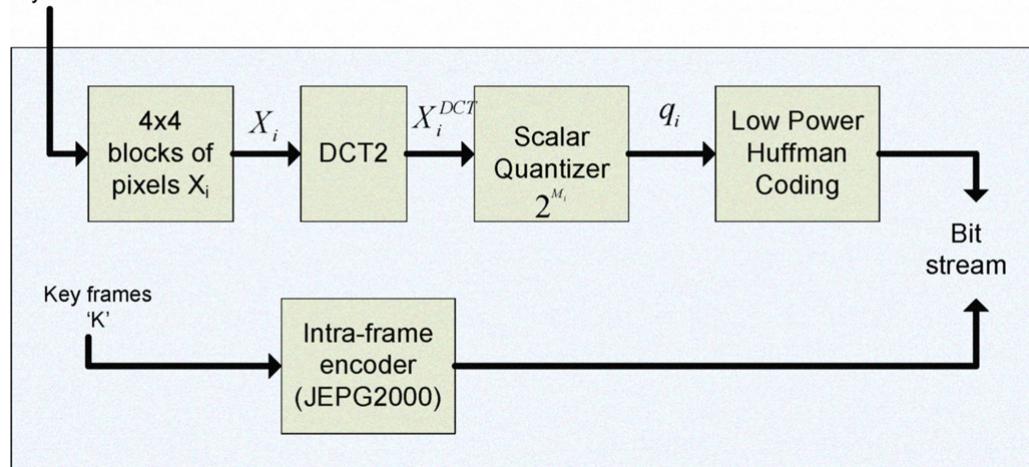


Figura 4. Diagrama de bloques de un codificador de vídeo Wyner-Ziv.

A continuación se realiza la decodificación de vídeo. Los fotogramas clave se decodifican empleando el algoritmo de decodificación correspondiente, mientras que los demás no sólo se calculan como la transformada inversa del algoritmo empleado, sino que además se emplean los fotogramas clave cercanos para inferir parte de la información perdida durante la compresión, empleando técnicas de interpolación.

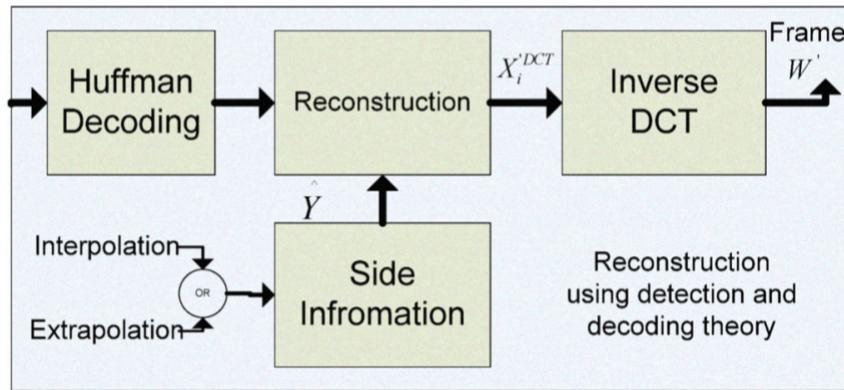


Figura 5. Diagrama de bloques de un decodificador de vídeo Wyner-Ziv.

Esta técnica permite reducir el consumo energético de los nodos de la red inalámbrica sin que la calidad del vídeo se vea afectada en gran medida.

II.3 Cobertura en WWSN

Otro de los problemas en las redes de vídeo inalámbricas es el de la cobertura: se desea que la red de cámaras abarque toda el área que se quiere monitorizar [12]. Esto presenta problemas en caso de haber obstáculos en la zona monitorizada, y tendrán que disponerse las cámaras de forma que los puntos ciegos de una sean visibles desde otra.

En la siguiente imagen se muestran en blanco las zonas cubiertas por una o más cámaras y en negro las que no están cubiertas por ninguna (puntos ciegos). Se observan las sombras que dejan unos obstáculos (en rosa).

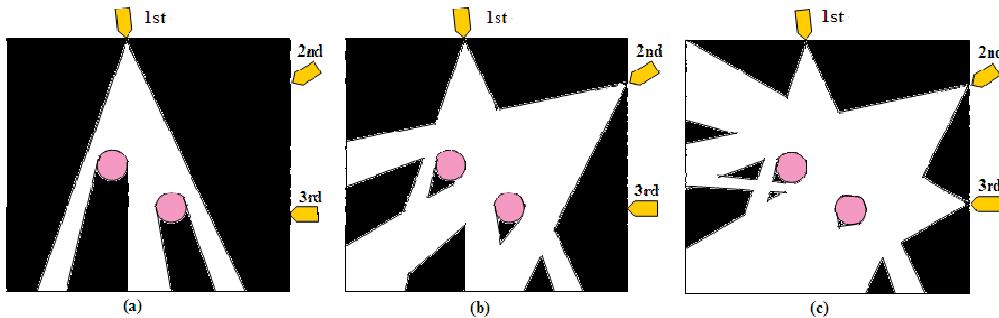


Figura 6. Cobertura y puntos ciegos con 1, 2 y 3 cámaras.

Por tanto, deberán colocarse las cámaras de forma que se minimicen los puntos ciegos.

Posicionamiento espacial:

Otra aplicación de las redes visuales es la de calcular la posición del espacio en la que se encuentran uno o varios objetos.

Una única cámara no es capaz de detectar la profundidad a la que se encuentra un objeto, siendo necesaria otra cámara perpendicular que proporcione una tercera coordenada. En caso de querer calcular posiciones de más de un objeto, es posible que no baste con dos cámaras y que sea necesario colocar más cámaras. Véase en la siguiente imagen cómo con sólo dos cámaras, dos objetos coplanaarios no se pueden ubicar con certeza, ya que podrían estar tanto en las posiciones A y B como en las posiciones C y D [10].

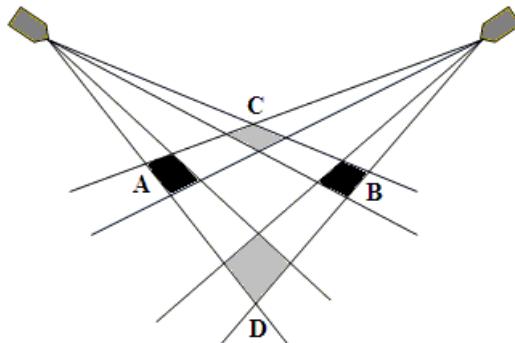


Figura 7. Problema de posicionamiento con dos cámaras.

Un modelo que se puede emplear para el posicionamiento de objetos es el de la construcción de un mapa de certeza, en el que se indican las posibles posiciones en las que puede estar un objeto en función de los datos recibidos por varias cámaras [10].

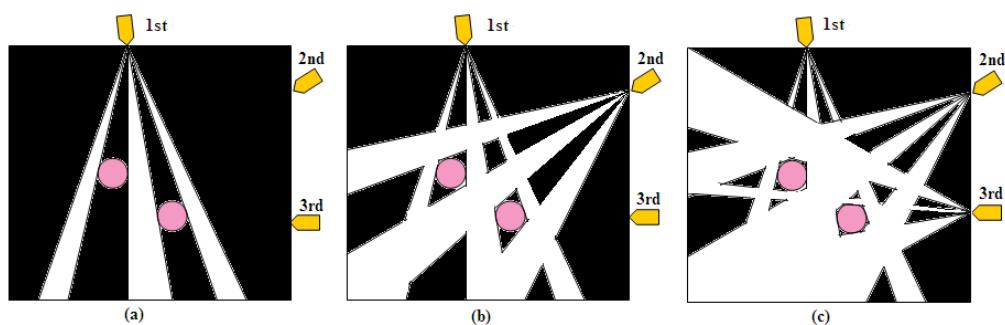


Figura 8. Posicionamiento con tres cámaras.

De esta forma, se puede saber cuándo existe ambigüedad en la posición de los objetos y cuántas cámaras hay que combinar para eliminarla.

Cobertura tridimensional:

Una aplicación interesante de las redes de vídeo inalámbricas es no limitarse a mostrar vídeo de una zona, sino ser capaces de reproducir un modelo tridimensional de dicha zona, con lo que se obtiene cobertura de vídeo desde cualquier punto de vista aunque no haya una cámara situada exactamente en dicho punto. Esto se consigue combinando las imágenes obtenidas mediante varias cámaras, las cuales graban un mismo objeto desde distintos ángulos, y procesando la información obtenida [11].

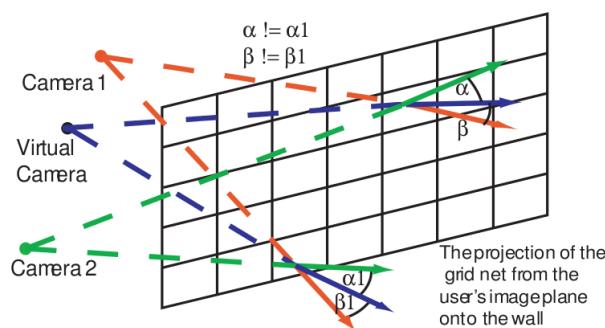


Figura 9. Emulación de una cámara virtual a partir de las imágenes de dos cámaras reales.

Esto conlleva cierta complejidad, ya que no sólo se trata de proyectar las imágenes grabadas sobre un modelo tridimensional de la zona a monitorizar generado previamente, sino que dicho modelo se genera en tiempo real, por lo que hay que aplicar las ya descritas técnicas de posicionamiento espacial a las imágenes recibidas.

III. Compresión de imágenes

La compresión de imágenes se basa en eliminar la información redundante de la imagen. Se diferencian dos tipos de redundancia:

- Redundancia estadística: debida a la alta correlación existente entre los píxeles vecinos en la gran mayoría de las imágenes. A esta redundancia también se la llama redundancia espacial para diferenciarla de la redundancia temporal (alta correlación entre imágenes sucesivas) existente en la compresión de video y que también forma parte de la redundancia estadística.
- Redundancia de percepción: es la parte de la información que el sistema de visión humano no percibe correctamente o con buena precisión. Como por ejemplo, las altas frecuencias espaciales.

Y dos tipos de compresión:

- Compresión sin pérdidas: utiliza técnicas de reducción de redundancia estadística de forma que sea posible recuperar una copia exacta de la imagen original tras

la descompresión. Ejemplos de este tipo de compresión son la codificación incremental, Run-Length Encode (RLE) y la codificación Huffman.

- Compresión con pérdidas: para lograr mayores ratios de compresión se elimina de forma irreversible parte de la información más irrelevante. Aprovechan tanto la redundancia estadística como la de percepción para intentar lograr los mayores ratios de compresión a costa de perder calidad en la imagen descomprimida (que nunca será exactamente igual a la original).

III.1 JPEG

JPEG son las siglas de Joint Photographic Experts Group. JPEG es un comité de expertos que se creó en los años 80, unión entre ISO/IEC e ITU-T, para compartir su experiencia y analizar el problema de la codificación de imágenes digitales. Este grupo ha creado varios estándares de codificación de imágenes, el más reciente el JPEG2000 que se basa en transformadas *wavelet*, pero este capítulo se va a centrar en el primero que desarrollaron y que comúnmente se denomina con las siglas del grupo. El nombre formal de este estándar es ITU-T Recommendation T.81 o ISO/IEC IS 10918-1.

Este estándar de compresión ha sido el más ampliamente utilizado hasta el momento. Utiliza la transformada discreta del coseno (DCT) para realizar una compresión con pérdidas aprovechando las deficiencias del ojo humano.

La secuencia de operaciones que realiza un compresor JPEG son las siguientes:

- Transformación del espacio de color.
- División de la imagen en bloques de 8x8 píxeles.
- Aplicación de la transformada discreta del coseno (DCT) a cada bloque.
- Cuantificación.
- Lectura de los bloques en zigzag.
- Compresión sin pérdidas:
 - Incremental.
 - RLE.
 - Huffman.

III.2 Transformación del espacio de color

Si la imagen es en color RGB (cada píxel está representado por tres números de 0 a 255 que definen la intensidad de los colores rojo, verde y azul) se realiza el cambio a YCbCr (también llamado YUV. Originalmente YUV refería a video analógico e YCbCr a video o imágenes digitales, pero actualmente suele utilizarse YUV para referirse a YCbCr). YCbCr describe cada píxel con tres números:

- Y: Luminancia. Es la intensidad del píxel. Esta componente describe la imagen en blanco y negro.
- Cb: Crominancia azul. Describe la diferencia de color azul.
- Cr: Crominancia roja. Describe la diferencia de color rojo.

YCbCr con referencia a RGB es tan solo un cambio en los ejes de referencia en el espacio del color.

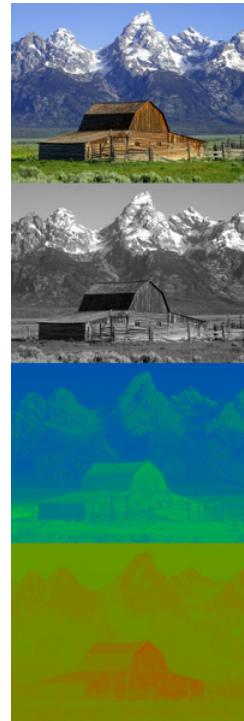


Figura 10. Descomposición YCbCr

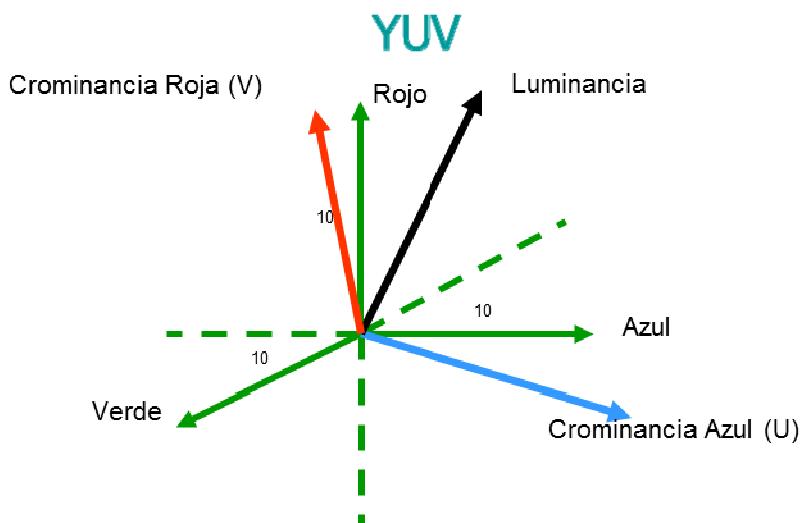


Figura 11. Cambio de ejes de coordenadas en el espacio de color entre RGB e YUV.

Las relaciones matemáticas entre RGB e YCbCr son:

$$\begin{bmatrix} Y \\ C_b \\ C_r \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.331 & 0.5 \\ 0.5 & -0.419 & -0.081 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix} \text{ y } \begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.403 \\ 1 & -0.344 & -0.714 \\ 1 & 1.773 & 0 \end{bmatrix} \cdot \begin{bmatrix} Y \\ C_b \\ C_r \end{bmatrix}$$

Dado que el ojo humano es más sensible a los cambios de intensidad lumínica que a los cambios de color se realiza un submuestreo de las componentes de crominancia. Cada dos o cuatro luminancias (Y) comparten los mismos valores de crominancias (Cr y Cb). Dando lugar a los distintos tipos de muestreo recogidos en la figura.

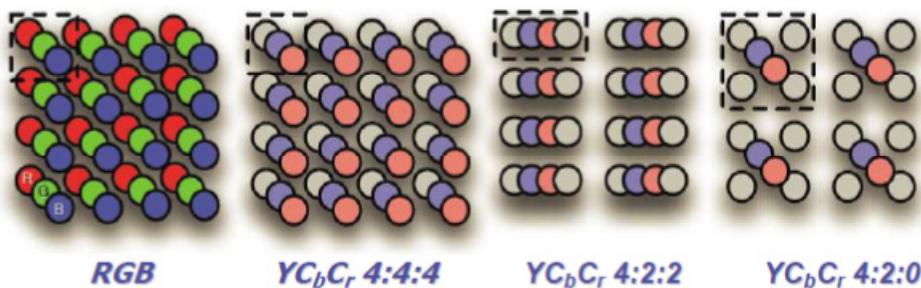


Figura 12. Tipos de muestreo en codificación de video e imagen digital.

III.3 DCT

La transformada discreta del coseno (DCT) se utiliza habitualmente para la compresión de imágenes y vídeo. Es una transformada basada en la transformada discreta de Fourier (DFT), pero utiliza únicamente números reales. Aunque la parte real de la DFT y la DCT están relacionadas, la DCT no es la parte real de la DFT.

Tanto la DFT como la DCT consisten en dividir una señal (discreta) en la suma de una serie de funciones (también discretas) ponderadas por unos coeficientes. Estas funciones, llamadas funciones base, son ortogonales y por tanto independientes (no existe la posibilidad de que una de estas funciones pueda representarse a través de una combinación de las demás, sin embargo, el conjunto completo pueden representar cualquier señal cuando se ponderan mediante coeficientes y se suman entre si). Las funciones base sólo dependen del número de muestras de la señal, y fijado éste las funciones base siempre son iguales. El conjunto de coeficientes que ponderan las funciones base son el resultado de la transformación directa. Al proceso de reconstruir la señal a partir de los coeficientes de la transformada directa se denomina transformación inversa (IDFT o IDCT).

La diferencia entre la DFT y la DCT es que la DCT utiliza únicamente funciones coseno, y por lo tanto sus coeficientes son números reales. Formalmente, la transformada de coseno discreta unidimensional es una función lineal invertible de R^N en R^N (equivalente una matriz cuadrada NxN). Existen ocho variantes diferentes de DCT

unidimensionales, siendo las utilizadas en compresión de imágenes la DCT-II cuya inversa es la DCT-III, también llamadas transformada directa del coseno (FDCT: Forward Discrete Cosine Transform) y transformada inversa del coseno (IDCT: Inverse Discrete Cosine Transform) respectivamente.

DCT-I:

$$y_j = \frac{1}{2} [x_0 + (-1)^j x_{N-1}] + \sum_{k=1}^{N-2} x_k \cos\left(\frac{\pi}{N-1} j\right)$$

DCT-II:

$$y_j = \sum_{k=0}^{N-1} x_k \cos\left[\frac{\pi}{N} j \left(k + \frac{1}{2}\right)\right]$$

DCT-III (IDCT):

$$x_k = \frac{1}{2} y_0 + \sum_{j=1}^{N-1} y_j \cos\left[\frac{\pi}{N} \left(k + \frac{1}{2}\right) j\right]$$

DCT-IV:

$$y_j = \sum_{k=0}^{N-1} x_k \cos\left[\frac{\pi}{N} \left(j + \frac{1}{2}\right) \left(k + \frac{1}{2}\right)\right]$$

Estas ecuaciones corresponden con DFTs reales de orden par de entradas de $M=2N$ puntos generados haciendo la extensión simétrica correspondiente de la señal x de N puntos según explica Martucci en su artículo "Symmetric Convolution and the Discrete Sine and Cosine Transforms" [12]. Las otras cuatro DCTs se corresponden con DFTs reales de orden impar.

La transformación se puede expresar matricialmente de la forma:

$$\vec{y} = C \cdot \vec{x}$$

Para que C sea una matriz ortonormal (y por tanto su inversa coincida con su transpuesta) se han de multiplicar las ecuaciones anteriores por unos coeficientes de escalamiento y normalización que pueden depender de j y k . Para la DCT-II:

$$C_{jk} = w(j) \cos\left(\frac{\pi}{N} j \left(k + \frac{1}{2}\right)\right)$$

$$w(j) = \begin{cases} 1/\sqrt{N} & \text{si } j = 0 \\ \sqrt{2/N} & \text{si } j \neq 0 \end{cases}$$

La primera función base de la DCT-II siempre es constante y a su coeficiente asociado (y_0) se le denomina componente de continua.

Se pueden componer dos (o más) grupos de funciones básicas para crear transformadas de dos (o más) dimensiones. La DCT bidimensional (DCT-2D) es una función lineal invertible de R^{NxN} en R^{NxN} que descompone el bloque de imagen en una suma de frecuencias espaciales. Los coeficientes y_{kl} de la DCT para bloques x_{ij} de 8x8 se expresan como:

$$y_{kl} = \frac{c(k)c(l)}{4} \sum_{i=0}^7 \sum_{j=0}^7 x_{ij} \cos\left(\frac{(2i+1)k\pi}{16}\right) \cos\left(\frac{(2j+1)l\pi}{16}\right)$$

Donde $k, l = 0, 1, \dots, 7$ y $c(x) = \begin{cases} 1/2 & \text{si } x = 0 \\ 1 & \text{si } x \neq 0 \end{cases}$

Mientras que la IDCT-2D se obtiene de:

$$x_{ij} = \sum_{k=0}^7 \sum_{l=0}^7 y_{kl} \frac{c(k)c(l)}{4} \cos\left(\frac{(2i+1)k\pi}{16}\right) \cos\left(\frac{(2j+1)l\pi}{16}\right)$$

El bloque de coeficientes de la DCT está ordenado de modo que la componente continua corresponde al elemento y_{00} y la frecuencia espacial crece con los índices k y l siendo y_{77} el coeficiente correspondiente a la mayor frecuencia.

La figura 8 representa un conjunto de 64 funciones base bidimensionales (imágenes base) que se generan multiplicando un conjunto de funciones base unidimensionales de ocho puntos ($N=8$) orientadas horizontalmente, por un conjunto verticalmente orientado de las mismas funciones. Las imágenes base orientadas horizontalmente representan las frecuencias horizontales y las orientadas verticalmente representan las frecuencias verticales. La fila superior y la columna de la izquierda tienen variaciones de intensidad en una sola dimensión. Para propósitos de ilustración, un gris neutro representa cero en estas figuras, el blanco representa amplitudes positivas, y el negro representa amplitudes negativas.

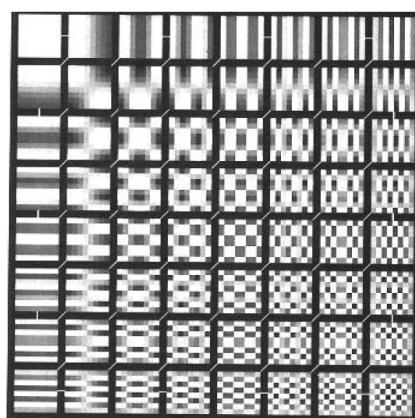


Figura 13. Imágenes base de una DCT-2D.

La ventaja que tiene la DCT frente a la DFT para la compresión de imágenes, a parte de solo utilizar números reales, es que produce una mejor compactación de la energía (consigue concentrar la mayor parte de la información en pocos coeficientes) y un menor efecto de bloque.

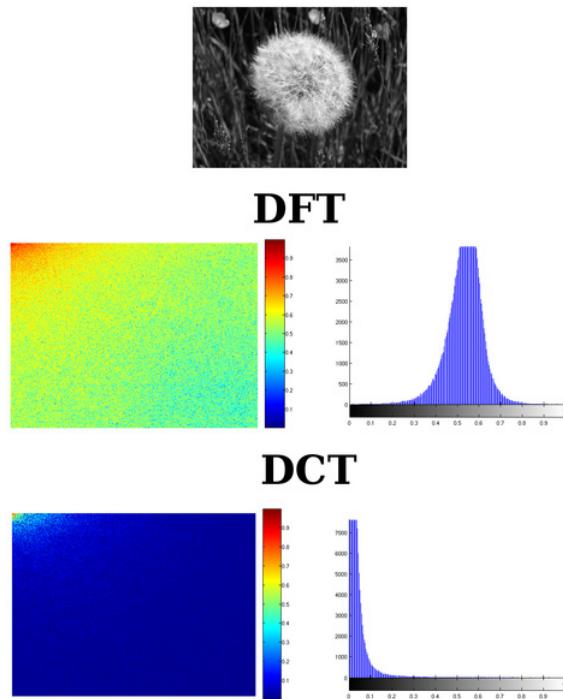


Figura 14. Compactación de energía de una DCT comparada con una DFT.

El efecto de bloque se produce cuando se divide la imagen en bloques de 8x8 píxeles o macrobloques de 16x16 píxeles para poder ejecutar los algoritmos de transformación. Cuando se lleva a cabo la DFT del bloque, se asume la periodicidad del mismo (que se repite a lo largo de todo el plano bidimensional que contiene la imagen). En la transformada de Fourier el píxel B del borde derecho será tratado por el algoritmo como si estuviera seguido por el píxel A. Si los niveles de gris en cada píxel difieren considerablemente cualquier reconstrucción del bloque a partir de únicamente un número limitado de coeficientes de Fourier dará lugar a valores erróneos en A y B. Este fenómeno es lo que se conoce como efecto de bloque, que tiende a hacer muy visibles los límites de los bloques en la compresión, especialmente cuando la proporción de compresión es elevada. Sin embargo, mientras que la teoría de Fourier implica la repetición de los bloques LxL, la teoría de la DCT impone esta repetición sobre bloques 2Lx2L, que están relacionados con los bloques originales LxL a través de simetrías especulares. La consecuencia de esta simetría specular es que después del

píxel B, le sigue otro píxel B, eliminando así la discontinuidad, esto provoca una reducción considerable del efecto de bloque.

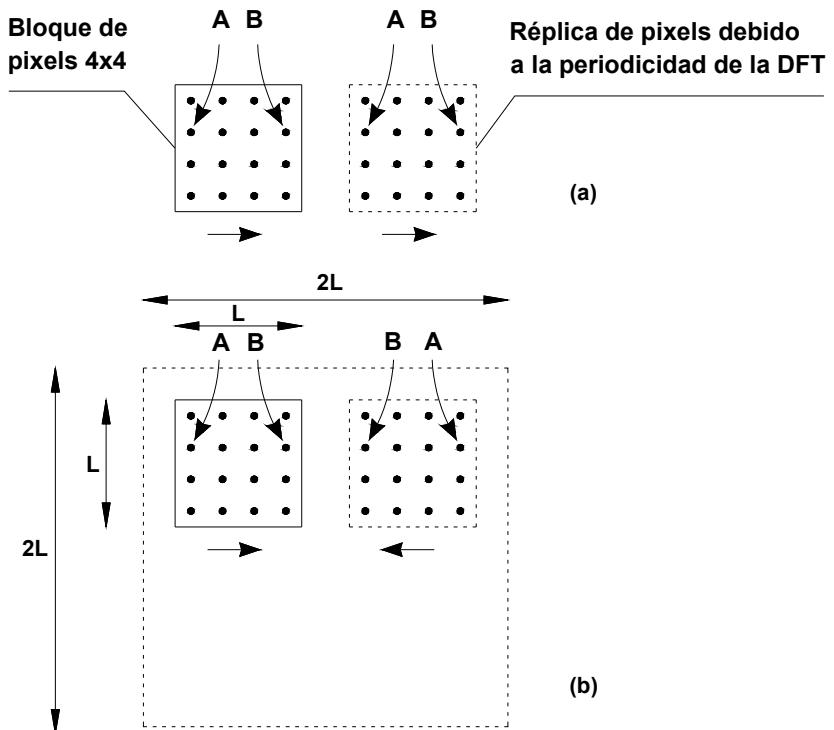


Figura 15. Periodicidad supuesta de un bloque 4x4 por (a) una DFT y (b) una DCT.

La transformación lineal óptima que minimiza el error cuadrático medio entre la imagen original y la imagen recuperada (después de transformar y comprimir la imagen) es la transformada de Karhunen-Loëve (KLT). La KLT realiza una descomposición de componentes principales (PCA) de los bloques de la imagen, por lo que las funciones bases (y la matriz de la transformación) dependen de las propiedades estadísticas de la imagen. Las funciones base (y su matriz de transformación) de la DCT dependen únicamente del orden N. Por lo que como se mencionó antes, fijado el número de puntos estas son siempre iguales sea cual sea la imagen. Una de las ventajas de la DCT es que siendo fija la transformación su eficiencia de compresión se aproxima a la de la KLT (la óptima) para imágenes con alto grado de correlación espacial.

III.4 DLMT

DLMT son las siglas de Discrete Lopez-Moreno Transform, que es una nueva transformada desarrollada en el Centro de Electrónica Industrial (CEI) de la Universidad Politécnica de Madrid [13]. Pretende ser una alternativa a la DCT antes explicada y a la Discrete Wavelet Transform (DWT), en la que no es necesario dividir la imagen en bloques o macrobloques, y que se utiliza tanto en el estándar de codificación JPEG2000 como en los próximos H264-SVC (Scalable Video Coding).

La DWT transforma una señal discreta en el dominio de tiempo en una señal discreta en el dominio de tiempo-frecuencia. Es decir, mediante la DWT se logra un análisis en los dominios de tiempo y frecuencia. Los efectos de bloque no se presentan incluso con relaciones altas de compresión. Si la transformada de Fourier descompone una señal en una suma de sinusoides de frecuencias diferentes, la DWT representa una señal como una superposición de varias *wavelets*. Estas *wavelets* son pequeñas "olas" que tienen su poder concentrado en unas ventanas de tiempo determinado. Aunque no existe una amplia literatura que explique el significado físico de la DWT (continua o discreta), se explica a través del concepto de escala de resolución que es muy común en la cartografía. Es decir, cuanto mayor es la escala de un mapa (dominio del tiempo en este caso), menor será la resolución (dominio de la frecuencia), es decir, que se puede cubrir un área geográfica grande pero con pocos detalles. Sin embargo, con la disminución de la escala del mapa es posible apreciar más detalles. En el caso de la señal de dominio tiempo a través de la DWT, disminuyendo el tiempo de análisis (escala) de la señal, la resolución aumenta en el dominio de la frecuencia. Por el contrario, el aumento del tiempo de análisis (en una escala en el dominio del tiempo) reduce la resolución en el dominio de la frecuencia.

El principal inconveniente de la DWT es su alto coste computacional en comparación con la DCT. Por otro lado, los sistemas basados en la compresión de DWT presentan sus resultados más eficaces para la adaptación de los algoritmos de cálculo para el tipo de señal (en el caso de la compresión de imágenes, el tipo de las imágenes y sus características espacio-temporales). Además, teniendo en cuenta los algoritmos de cálculo que se han propuesto, basado en el uso de múltiples etapas de filtrado de la señal, aparecen efectos de distorsión en los bordes de la imagen.

La DLMT propone un nuevo esquema intermedio entre la DCT y la DWT. La DLMT es computacionalmente muy similar a la DCT y utiliza funciones cuasi-sinusoidales: las funciones sinc. Por lo que se pretende que la aparición de efectos de bloque tenga una importancia relativamente baja. El uso de las funciones sinc permitirá lograr un control de varias resoluciones muy cercano al obtenido por una DWT, pero sin aumentar la complejidad computacional de la transformación. La DLMT también se puede aplicar sobre una imagen entera, en este caso, los cálculos necesarios se incrementan, pero gracias a su simplicidad, es fácil de implementar en hardware.

Las ecuaciones de la DLMT de una dimensión son:

$$X(k) = \sum_{n=0}^{N-1} x(n) \operatorname{sinc}\left(\frac{(2n+1)\pi k}{2N}\right)$$

Y las de la DLMT-2D, que es la realmente interesante para la compresión de imágenes:

$$X(k, l) = \frac{1}{(2N)^2} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} x(i, j) \operatorname{sinc}\left(\frac{(2i+1)\pi k}{2N}\right) \operatorname{sinc}\left(\frac{(2j+1)\pi l}{2N}\right)$$

La función sinc se define como:

$$\operatorname{sinc}(x) = \frac{\sin(x)}{x}$$

O normalizada:

$$\operatorname{sinc}_N(x) = \frac{\sin(\pi x)}{\pi x}$$

Esta función tiene una discontinuidad evitable en $x=0$, que normalmente se redefine como $\operatorname{sinc}(0)=1$ para que sea continua.

III.5 Cuantificación y lectura en zigzag

Tras el proceso de transformación (DCT) de los bloques se cuantifican los resultados. La cuantificación consiste en dividir los datos y redondear. Esta operación es la que realmente comprime la imagen. Es una compresión con pérdidas, por lo que es una operación irreversible. Para ello se utilizan tablas de cuantificación que describen las cantidades por las que se ha de dividir cada elemento de la matriz transformada. Cuanto mayor sea el valor por el que se divide un elemento menor será su precisión y también serán menores los bits necesarios para almacenarlo o transmitirlo.

Las tablas de cuantificación tienen en cuenta la sensibilidad del ojo humano a los distintos elementos de la matriz resultado de la DCT, dividiendo por valores mayores las componentes correspondientes a las mayores frecuencias espaciales (eliminación de redundancia de percepción).

Empleando distintas tablas de cuantificación se pueden conseguir imágenes de peor calidad pero altamente comprimidas o imágenes de mejor calidad pero menórricamente comprimidas. Una tabla de cuantificación muy utilizada para luminancias (Y) en compresores JPEG implementados en software es la mostrada en la figura 16. Para las crominancias (Cb y Cr) se utilizan tablas de cuantificación distintas a las empleadas para luminancias aprovechando las distintas características de la percepción del ojo. En sistemas puramente implementados en hardware suele usarse valores poten-

cia de 2 ya que dividir por ellos se puede realizar con una sencilla operación de desplazamiento de bits.

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Figura 16. Tabla de cuantificación de Losheller.

Tras efectuar la cuantificación normalmente aparecen numerosos ceros en la parte inferior derecha de la matriz. Esto sucede porque la mayoría de las imágenes no suelen tener elevadas componentes de altas frecuencias (una imagen típica contiene grandes áreas planas con cambios que tienden a ser graduales y con eventuales líneas o bordes) y además han sido divididos por valores elevados de la tabla de cuantificación.

Se realiza una lectura del bloque en zigzag para lograr la mayor cantidad de ceros en el final de la trama. Todos estos ceros finales serán sustituidos por un único elemento de final de bloque (EOB: *end of block*) eliminando gran cantidad de elementos a transmitir o almacenar.

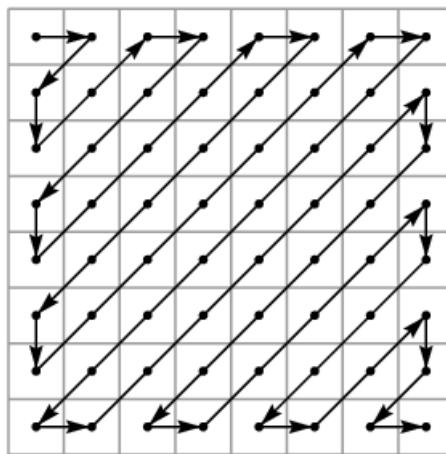


Figura 17. Recorrido en zigzag por la matriz cuantificada.

III.6 Compresión sin pérdidas

Una vez obtenida una trama unidimensional de datos después de la lectura en zig-zag de todos los bloques de imagen transformados se puede lograr una compresión adicional empleando las técnicas de compresión sin pérdidas. Debido a que tras el empleo de estas técnicas se obtienen elementos codificados con distinto número de bits, a todo este proceso también se la llama codificación de longitud variable (VLC: Variable Length Coding) o codificación entrópica.

Como el valor medio de bloques consecutivos suele ser similar, se utiliza codificación incremental en la componente de continua (1º elemento de cada trama leída en zigzag). Esto es transmitir o almacenar la diferencia con el anterior valor en lugar de su valor absoluto. La codificación incremental es eficiente para señales continuas o de baja variabilidad.

Run-Length Encoding (RLE):

En tramas de datos en las que se repiten numerosos elementos consecutivamente se puede comprimir la información indicando primero el número de veces que se repite el dato, posteriormente el dato repetido y así sucesivamente. Como ejemplo, la secuencia de 16 letras:

A-A-A-B-B-C-D-E-E-E-E-F-F-F-F

Aplicando RLE sería:

3-A-2-B-1-C-1-D-5-E-4-F

De tan solo 12 símbolos. Como se puede observar, si hay elementos que no se repiten, no solo no se comprimen, si no que se duplican. Pudiendo llegar a crear tramas con más elementos que la original por culpa de la aparición de "1"s o símbolos "se repite una sola vez". Y con los elementos que se repiten sólo 2 veces RLE no aporta mejora alguna. "BB" en principio ocupa lo mismo que "2B". Para evitar esto hay variantes de RLE donde se indica si se va a codificar el siguiente elemento como RLE o sin alterar. Una forma habitual de hacer esta indicación es utilizar el primer bit que codifica cada símbolo para diferenciar que el símbolo es un número de repeticiones o un símbolo de la trama. Considerando que si delante de un símbolo de la trama no hay un número de repeticiones es que debe aparecer sin repetir. Otras variantes utilizan símbolos especiales para indicar el comienzo y fin de las tramas sin alterar.

Codificación Huffman:

La codificación Huffman utiliza un menor número de bits para codificar los símbolos más probables que para codificar los símbolos más improbables. Para que no haya ambigüedad en la trama codificada se exige a la codificación de cada símbolo que no sea prefijo de la codificación de otro.

David A. Huffman desarrolló un método para generar estas codificaciones de forma óptima teniendo en cuenta las frecuencias de aparición de cada símbolo [14]. El método consiste en crear un árbol binario de abajo a arriba. En cada paso se escogen las dos frecuencias menores para que sean las dos hojas de un nodo que tendrá como frecuencia la suma de las frecuencias de sus dos hojas. Una vez que se tiene el árbol se dan valores “0” ó “1” a cada una de las ramas (como cada nodo tiene dos ramas a una de ellas se le asigna “0” y a la otra “1”).

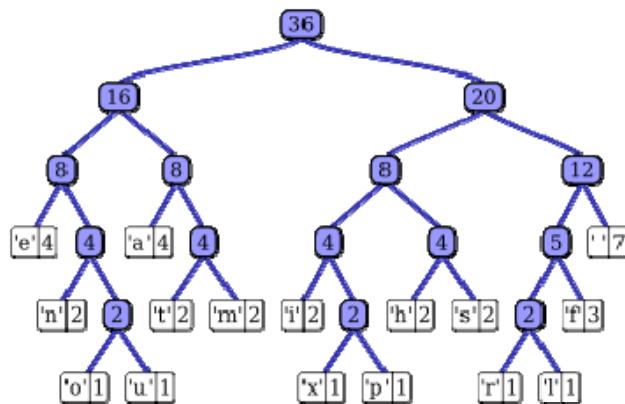


Figura 18. Árbol binario construido mediante el algoritmo de Huffman para “this is an example of a huffman tree”.

Símbolo	Frec.	Código
espacio	7	111
e	4	000
a	4	010
f	3	1101
t	2	0110
s	2	1011
n	2	0010
m	2	0111

Símbolo	Frec.	Código
i	2	1000
h	2	1010
x	1	10010
u	1	00111
r	1	11000
p	1	10011
o	1	00110
l	1	11001

Figura 19. Códigos generados asignando “1” a las ramas izquierdas y “0” a las de la derecha del árbol de la anterior.

Existen otras codificaciones (codificaciones aritméticas) que en lugar de codificar símbolos aislados codifican grupos de símbolos que se repiten con frecuencia. Estas codificaciones pueden llegar a resultados más óptimos pero son mucho más complejas y están protegidas por patentes por lo que la codificación Huffman es la más extendida y utilizada.

III.7 Redes Neuronales Artificiales

En inglés Artificial Neural Networks (ANN). Son una serie de algoritmos inspirados en el funcionamiento de nuestro cerebro. Estos algoritmos tienen las capacidades de aprender, generalizar y abstraer, por lo que se engloban dentro de la inteligencia artificial.

Los primeros modelos fueron publicados en 1943 por el neurobiólogo Warren McCulloch y el estadístico Walter Pitts. Unos pocos años después Donald Hebb desarrolló sus teorías sobre el aprendizaje. En los años 50 Frank Rosenblatt desarrolló el Perceptron, que fue la primera red neuronal artificial, y justo después Bernard Widrow y Marcial Hoff crearon el Adaline (Adaptive Linear Neuron) y su versión de dos capas Madaline. En los años 60 se redujo la investigación en este campo debido al estudio de Minsky y Papert sobre las limitaciones del Perceptron. Hasta que en los años 80 el desarrollo del algoritmo de aprendizaje Backpropagation y la red de Hopfield hicieron resurgir el interés en estos algoritmos.

Las redes neuronales artificiales se basan en un modelo funcional simplificado de una neurona. Este modelo multiplica cada entrada (x_i) por un peso (W_i), suma todas las entradas ponderadas y finalmente le aplica una función (f) a la suma para generar una salida. La función de salida suele ser una función lineal (que es el equivalente de no aplicar ninguna función) o una función de saturación como la función sigmoide para obtener resultados en el rango de 0 a 1 o la función tangente hiperbólica para obtener resultados en el rango de -1 a 1.

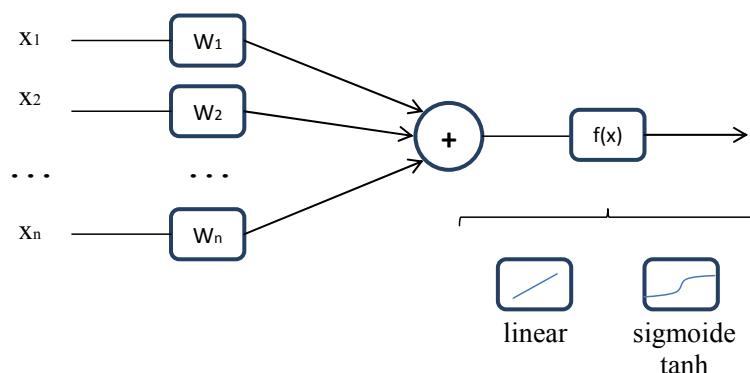


Figura 20. Modelo funcional simplificado de una neurona.

Matemáticamente: $y = f(\sum W_i x_i)$

La combinación de estas neuronas o nodos en redes estructuradas por capas permiten la implementación de funciones más complejas. La capacidad de aprendizaje y de adaptación de las redes neuronales proviene de los algoritmos iterativos que se utilizan para actualizar los valores de los pesos de las neuronas.

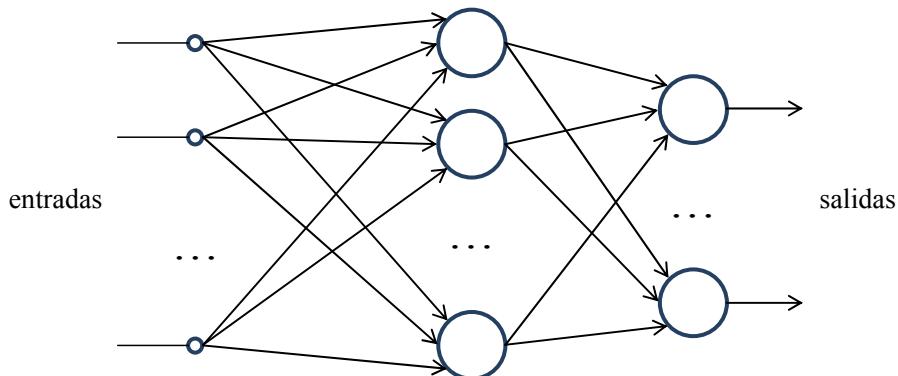


Figura 21. Esquema de una red neuronal de dos capas. Cada nodo representa un elemento de procesamiento como el de la figura X.

Existen principalmente dos métodos de aprendizaje:

- Supervisado: donde durante el entrenamiento se van especificando las salidas deseadas a las que debe tender la salida de la red.
- No supervisado: en este tipo de aprendizaje la propia red se auto-organiza para extraer características de los datos de entrada sin que ningún agente externo le indique si el resultado es correcto o no.

En las redes de aprendizaje supervisado primero existe una etapa de entrenamiento, donde se ha de proporcionar un conjunto de entradas junto al conjunto de salidas deseadas correspondiente a las entradas proporcionadas. Durante este entrenamiento se dan valores a los pesos de las neuronas para que aprendan las salidas deseadas. Despues se ha de comprobar el correcto funcionamiento con un conjunto de entradas y salidas distintas a los de entrenamiento llamado conjuntos de test. Una vez verificado el conjunto de test la red neuronal estará lista para su utilización en la aplicación entrenada. Este tipo de redes neuronales no calculan, si no que recuerdan lo aprendido. Además la información aprendida queda distribuida entre todos los pesos de las neuronas de la red. Por lo que si se daña alguno de estos datos la red neuronal continúa conservando su funcionalidad perdiendo tan sólo una pequeña parte de calidad o precisión.

Gracias a sus características las redes neuronales artificiales son empleadas en aplicaciones como:

- Reconocimiento de patrones (imágenes, voz, etc.).
- Análisis y procesamiento de señales.
- Filtrado.
- Clasificación.
- Etc.

III.8 Compresión de imágenes mediante redes neuronales

Las redes neuronales pueden ser empleadas para comprimir imágenes de varias formas. En el artículo “Neural Network Approaches to Image Compression” [15] se explican éstos métodos:

Como predictores no lineales en codificación predictiva:

La codificación predictiva aprovecha el alto grado de correlación que suele haber entre píxeles cercanos. El predictor utiliza los píxeles cercanos anteriormente procesados para hacer una predicción del siguiente píxel. La diferencia entre el valor del píxel y la predicción es almacenada o transmitida. Cuanto mejor sea la precisión del predictor menores serán estas diferencias y mayor será el ratio de compresión. Mediante modelos autorregresivos (modelos AR) se pueden conseguir buenos predictores lineales, pero cuando se es necesario un predictor no lineal las redes neuronales ofrecen muy buenos resultados. En la figura a se muestra un predictor realizado mediante una red neuronal multicapa. Otra variante consiste en utilizar una red de una sola capa para valorar entre los términos lineales y los términos de segundo orden aplicados a los píxeles anteriores como se muestra en la figura b.

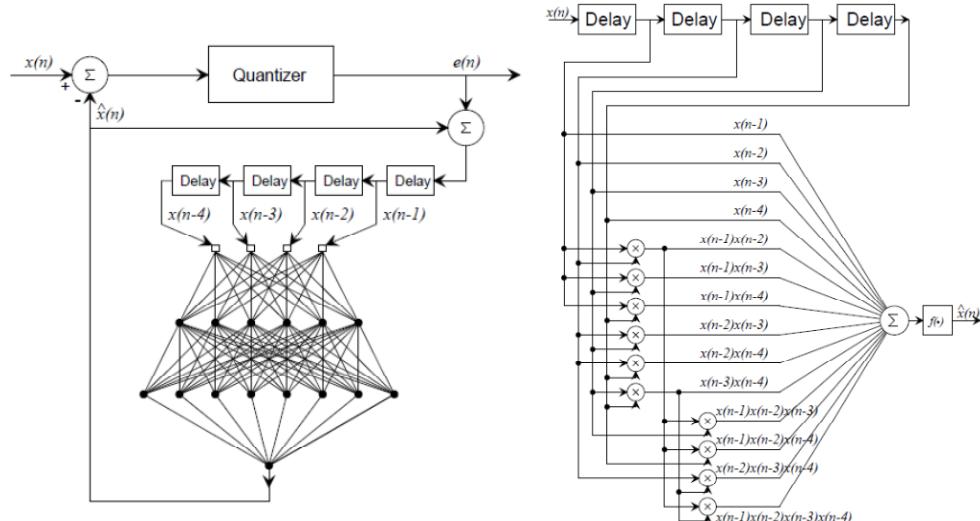


Figura 22. a) Codificación predictiva con red neuronal multicapa. b) Predictor de segundo orden.

Análisis de componentes principales (PCA):

La transformación lineal óptima para comprimir la mayor parte de la información en el menor número de componentes (minimizando el error cuadrático medio) es la transformación Karhunen-Loëve que consiste en quedarse con un número determinado de componentes principales. Ha sido demostrado que una red neuronal de funciones lineales con m salidas tiende a converger a las m primeras componentes

principales. Realizar esta conversión utilizando redes neuronales en lugar de calculando los autovectores puede suponer menor carga de almacenamiento (*storage overhead*) y ser más eficiente computacionalmente.

Adaptive Transform Coding:

Si aplicamos la misma transformación a todos los bloques de la imagen, la transformación Karhunen-Loève resulta óptima globalmente pero no resulta óptima localmente. Es decir, para un bloque concreto de la imagen, esta transformación puede dar resultados no muy buenos. Esto ocurre sobretodo en bordes y líneas donde las propiedades estadísticas de estas regiones suelen diferir bastante de las del resto de la imagen. Una solución propuesta consiste en utilizar un clasificador (implementado mediante redes neuronales) capaz de escoger para cada bloque la mejor transformación dentro de un grupo de transformaciones dado.

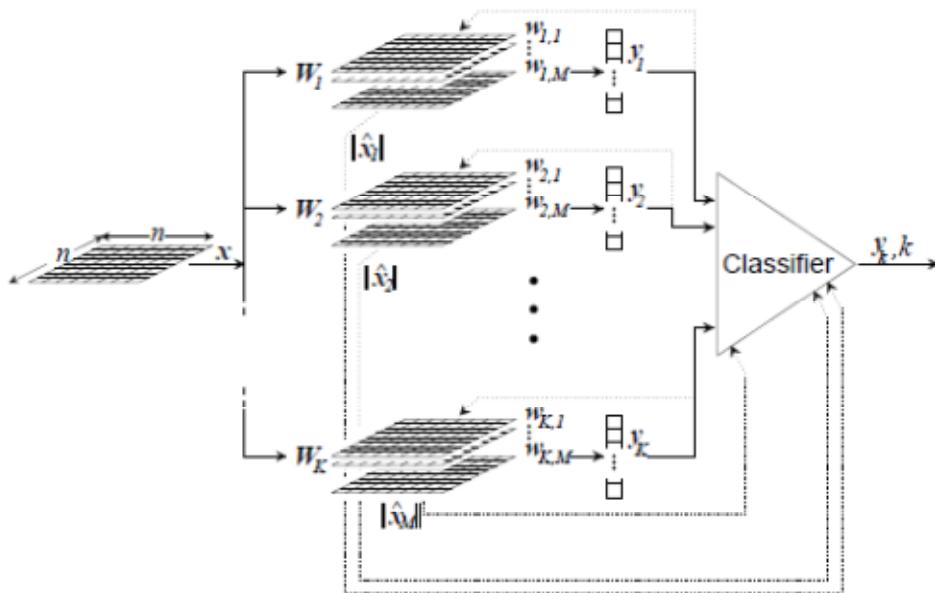


Figura 23. Compresor con selección de transformada.

Cuantificación vectorial utilizando redes neuronales no supervisadas (mapas auto-organizados: SOM):

La técnica de cuantificación vectorial consiste en aproximar cada bloque por el bloque más similar de una serie de bloques de imagen básicos. A estos bloques de imágenes básicos se les llama *codewords* y al conjunto de ellos *codebook*. Solamente es almacenado o transmitido el índice correspondiente al *codeword* seleccionado. Las redes neuronales SOM han resultado ser muy apropiadas para la generación del *codebook*.

Codificador y descodificador como una red neuronal multicapa:

El método más ampliamente utilizado para comprimir mediante redes neuronales consiste en entrenar una red neuronal donde la salida deseada objetivo es la propia información de entrada y donde se obliga a pasar la información por una capa oculta estrecha. Una vez entrenada la red, la primera mitad hará de compresor y la segunda de descomprimidor. El número de neuronas de la capa oculta determinará el factor de compresión. Si se utilizan en toda la red neuronal funciones lineales, como se ha mencionado antes, la red tenderá a quedarse con las primeras componentes principales (tantas como neuronas tenga la capa oculta). Sin embargo, las varianzas de cada una de estas componentes tienden a ser similares y la contribución al error de cada una de ellas aparece uniformemente distribuida, mientras que en un PCA puro la contribución al error de cada componente decrece con el valor de su autovalor correspondiente. Si permitimos funciones no lineales en algunas de las capas se pueden obtener resultados que van más allá de las transformaciones lineales.

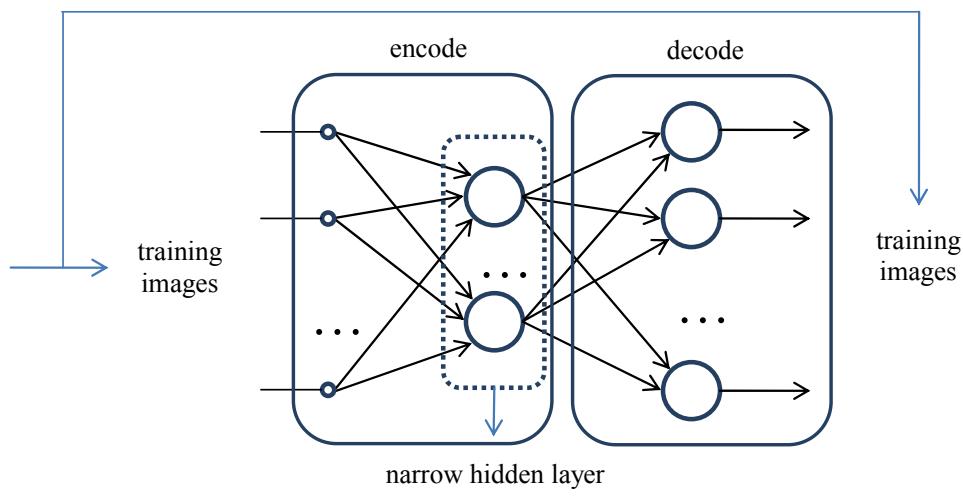


Figura 24. Auto-encoder.

Al ser algoritmos entrenados las redes neuronales alcanzan sus mejores resultados cuando se entrena para comprimir un tipo de imágenes concreto en lugar de como codificadores genéricos. Pudiendo aprender y explotar las características de las imágenes de la aplicación.

Diseño del compresor tipo JPEG

Diseño del compresor tipo JPEG

I. Modificaciones a JPEG propuestas para reducir el consumo energético

La primera modificación (y la más importante) consiste en no calcular las filas y columnas en las que se encuentran los coeficientes que representan las mayores frecuencias espaciales, ya que tras el proceso de cuantificación estos valores se anulan en la mayoría de los casos. Cierto es que obteniendo *bit-rates* de compresión similares se pierde algo de calidad respecto al método normalmente utilizado para JPEG. Pero la energía ahorrada no calculando todos estos elementos es muy grande. Además se ahorran multiplicadores que son circuitos grandes y costosos.

También se ha incluido la cuantificación en las matrices de coeficientes de la DCT (implementando matrices de coeficientes que tienen sus filas o columnas cuantificadas), ahorrando el proceso de dividir los resultados tras la transformación. De esta forma se obtienen tablas de cuantificación como la de la figura, que no son óptimas según la percepción humana (hay tablas de cuantificación que tienen en cuenta la sensibilidad del ojo humano a cada coeficiente) pero presentan resultados aceptables empleando el mínimo de recursos hardware.

	x2	x4	x8	x16	-	-	-	-
x2	4	8	16	32	256	256	256	256
x4	8	16	32	64	256	256	256	256
x8	16	32	64	128	256	256	256	256
x16	32	64	128	256	256	256	256	256
-	256	256	256	256	256	256	256	256
-	256	256	256	256	256	256	256	256
-	256	256	256	256	256	256	256	256
-	256	256	256	256	256	256	256	256

Figura 25. Tabla de cuantificación equivalente.

46341/2	46341/2	46341/2	46341/2	46341/2	46341/2	46341/2	46341/2
64277/4	54491/4	36410/4	12785/4	-12785/4	-36410/4	-54491/4	-64277/4
60547/8	25080/8	-25080/8	-60547/8	-60547/8	-25080/8	25080/8	60547/8
54491/16	-12785/16	-64277/16	-36410/16	36410/16	64277/16	12785/16	-54491/16
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

Figura 26. Matriz de coeficientes utilizada. Ha sido previamente multiplicada por 2^{17} y redondeada para poder ser utilizada por multiplicadores hardware de números enteros.

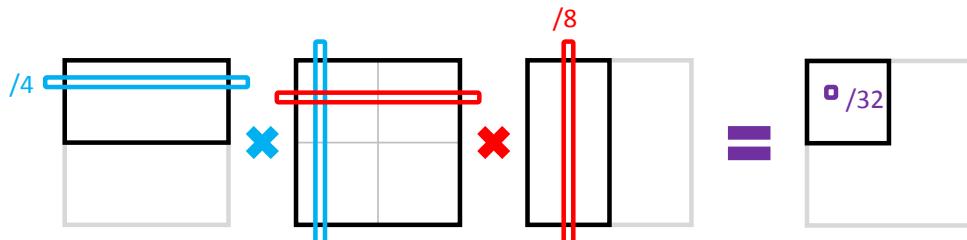


Figura 27. Esquema de la operación matricial implementada después de las modificaciones.

II. Códigos Matlab para optimización de algoritmos

Las nuevas metodologías de diseño para sistemas cada vez más complejos proponen nuevos niveles de abstracción. En el llamado nivel de algoritmo, se seleccionan los algoritmos a utilizar y los parámetros de estos. En este proyecto se han seleccionado tres algoritmos distintos: uno basado en la DCT, otro basado en la DLMT y otro basado en redes neuronales. Para probarlos, optimizarlos y ajustar sus parámetros se han desarrollado los algoritmos en Matlab y se han realizado varias pruebas y simulaciones hasta alcanzar unos resultados adecuados a este nivel de abstracción.

Se han creado varios algoritmos en Matlab para probar y ajustar un compresor tipo JPEG con las modificaciones propuestas. Se ha tenido en cuenta en los algoritmos Matlab las limitaciones de precisión que tendrá el algoritmo cuando se implemente en hardware. Estos algoritmos van creciendo en complejidad añadiendo cada uno un nuevo paso del algoritmo de forma que se puedan ir ajustando paso a paso que bits de datos intermedios van a ser almacenados o procesados, así como otros parámetros del algoritmo.

En las pruebas realizadas con los distintos niveles de corte (número de filas y columnas no calculadas) y tablas de cuantificación seleccionadas se ha determinado que si se emplea una codificación Huffman el empleo de RLE previo es contraproducente. Esto es debido a que para las condiciones simuladas ha resultado ser más ventajoso repetir varias veces un símbolo codificado con muy pocos bits (debido a que su frecuencia de aparición es muy alta) que insertar un símbolo indicador del número de repeticiones codificado con mayor cantidad de bits debido a su baja frecuencia de aparición.

También se han ensayado y valorado las siguientes opciones:

- Codificación incremental: solo ha supuesto beneficioso aplicarlo únicamente a la componente de continua (coeficiente (1,1) de la matriz resultado de aplicar la DCT).
- Tablas de cuantificación: se han probado distintas tablas teniendo en cuenta el balance entre la calidad subjetiva obtenida y el consumo energético. Sólo se han

probado tablas en las que todos sus elementos fueran potencias de 2, ya que dividir por potencias de 2 en hardware es un simple desplazamiento de bits. Se han probado tablas que equivalen a no calcular los elementos de varias filas y columnas. Y se han probado tablas que permiten incluir la cuantificación en la matriz de coeficientes de la transformada ahorrando el proceso de cuantificación posterior a realizar la transformación. La tabla seleccionada ha sido la mostrada en la figura 25.

- Tipo de transformación: a parte de la DCT-II (FDCT) también se ha probado la DLMT, la DCT-I, DCT-III (Es decir, aplicando primero la IDCT y después la FDCT) y la DCT-IV. Estos resultados se detallan en el capítulo “Conclusiones sobre la DLMT”.
- Mínima cuenta RLE: mínimo número de repeticiones que se codifican en RLE. Si tras RLE se realiza una codificación Huffman este parámetro es muy influyente, pero como se ha mencionado antes: los mejores resultados se obtienen sin RLE.
- *Clipping*: cuando se limita la precisión algunos resultados pueden presentar *overflow* (desbordamiento) o *underflow* (resultados menores que los representables). *Clipping* significa dejar como resultado el mayor representable posible para los *overflow* o el menor representable posible para los *underflow*. Pero para algunas pruebas o para ajustar otros parámetros a veces es conveniente eliminar la opción de *clipping*.

Para la tabla de cuantificación seleccionada y los parámetros ajustados, las simulaciones del codificador diseñado comprimen una secuencia de 47 imágenes de 256x256 píxeles a tan sólo un 2,3% del tamaño original con una calidad similar a la que se muestra en la siguiente figura (que es una de las 47 imágenes comprimidas).



Figura 28. Imagen de Lenna original de 256x256 e imagen comprimida y descomprimida con el comprisor simulado en Matlab.

En estas simulaciones se obtiene también el histograma (figura 29) utilizado para elaborar el diccionario para la codificación Huffman empleada.

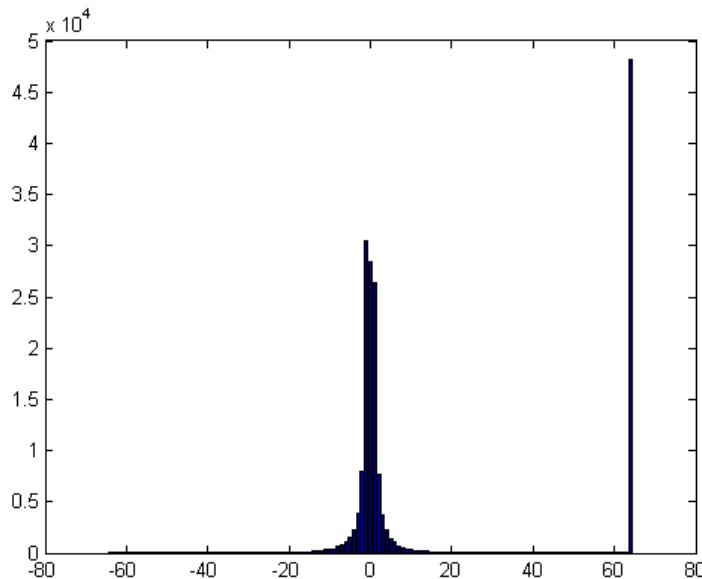


Figura 29. Histograma con las frecuencias de los símbolos obtenidos de la transformación de las 47 imágenes simuladas.

III. Conclusiones sobre la DLMT

Se probó la transformación DLMT con los mismos códigos Matlab que para la DCT, pero al no obtener resultados satisfactorios se han analizado las características matemáticas de la transformación.

La transformación es separable en cuanto a que es posible escribirla de la siguiente forma:

$$y_{kl} = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} x_{ij} \operatorname{sinc}\left(\frac{(2i+1)k\pi}{2N}\right) \operatorname{sinc}\left(\frac{(2j+1)l\pi}{2N}\right) = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} x_{ij} f_1(i, k) f_2(j, l)$$

Donde $f_1(i, k)$ es una función que solo depende de las variables i y k , y $f_2(j, l)$ otra función que solo depende de las variables j y k . Entonces es posible escribir la ecuación de la siguiente forma matricial:

$$\mathbf{Y} = \mathbf{C} \cdot \mathbf{X} \cdot \mathbf{C}^T$$

Donde \mathbf{C} es la matriz de la transformación unidimensional:

$$y_k = \sum_{i=0}^{N-1} x_i \operatorname{sinc}\left(\frac{(2i+1)k\pi}{2N}\right) \rightarrow \vec{y} = \mathbf{C} \cdot \vec{x}$$

$$c_{ki} = \text{sinc}\left(\frac{(2i+1)k\pi}{2N}\right)$$

En las transformaciones ortogonales como las de Fourier, seno y coseno la matriz inversa de C coincide con su transpuesta. Pero esta transformación no es ortogonal. Las funciones básicas tipo sinc de la transformación de una sola dimensión no son ortogonales entre sí, y por tanto:

$$C^{-1} \neq C^T$$

Y como consecuencia, la transformación inversa:

$$X = C^{-1} \cdot Y \cdot C^{-1,T}$$

No se corresponde con ninguna ecuación analítica conocida.

Para N=8, C=

1	1	1	1	1	1	1	1
0,9378	0,5194	0,0186	-0,2138	-0,1203	0,0709	0,123	0,0185
0,7649	-0,1434	-0,0186	0,0822	-0,0895	0,0622	-0,0202	-0,0182
0,5194	-0,1203	0,0185	0,0292	-0,0487	0,049	-0,0366	0,0178
0,253	0,1215	-0,0184	-0,0579	-0,0096	0,0334	0,0191	-0,0172
0,0186	0,0185	0,0183	0,0181	0,0178	0,0174	0,0169	0,0164
-0,1434	-0,0895	-0,0182	0,027	0,0285	0,0031	-0,0174	-0,0155
-0,2138	0,0292	0,0181	-0,0307	0,0236	-0,0077	-0,0072	0,0144

Figura 30. Matriz de coeficientes (C) de la DLMT para N=8.

Cuyo número de condición es 2.538,5. Lo que implica que los resultados van a ser sensibles a los redondeos debidos a las limitaciones de precisión.

Se han simulado en Matlab las transformaciones teniendo en cuenta las limitaciones de precisión que se encontrarían en el hardware. Para ello se han realizado códigos Matlab parametrizables para ajustar qué bits se deberían almacenar o transmitir en cada fase. Se han probado las siguientes operaciones:

- Realizar primero la transformación directa ($Y=C^*X^*C^T$) y luego la transformación inversa ($X=C^{-1}*Y*C^{-1,T}$)
- Realizar primero la transformación inversa ($Y= C^{-1}*Y*C^{-1,T}$) y luego la directa ($X=C^*Y^*C^T$). Consiguiendo así que el resultado de la primera transformación sea una descomposición en señales base tipo sinc de la señal original.
- Utilizar una modificación de la matriz C que posee un número de condición menor para ambas operaciones anteriores.

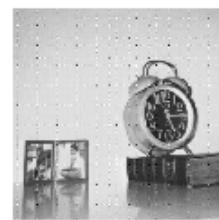
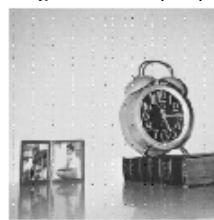
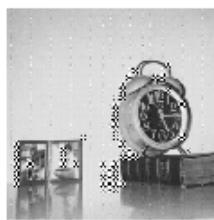
65536	65116	63865	61811	59003	55504	51394	46766
65536	61811	51394	36373	19668	4341	-7096	-13215
65536	55504	30837	4341	-11801	-13094	-4258	5298
65536	46766	9124	-13215	-8429	5298	7342	-1329
65536	36373	-7096	-10279	6556	4121	-5710	-1034
65536	25229	-14017	1973	5364	-5952	1935	2408
65536	14264	-11860	8394	-4539	1002	1638	-3050
65536	4341	-4258	4121	-3934	3700	-3426	3118

Figura 31. Matriz C^T modificada. Ha sido multiplicada por una potencia de 2 suficientemente grande para ajustar sus valores a 18 bits con signo y poder trabajar con números enteros.

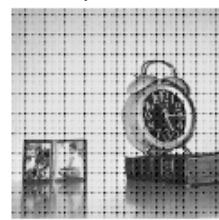
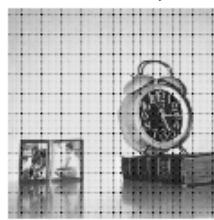
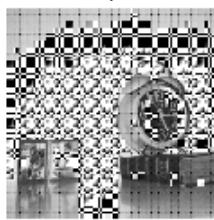
Las transformaciones se han realizado almacenando 16 bits por dato para los resultados intermedios. En las imágenes centrales de la figura se ve el resultado de guardar los bits fundamentales del resultado de la transformación y posteriormente realizar la transformación inversa correspondiente para recuperar la imagen. En las imágenes de la izquierda se ve el efecto de recortar el bit más significativo de los datos intermedios saturando el resultado si es necesario. Y en las imágenes de la derecha se ve el resultado de recortar el bit menos significativo saturando a cero si es necesario.



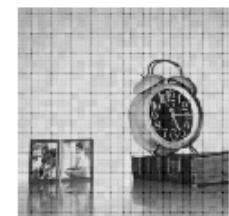
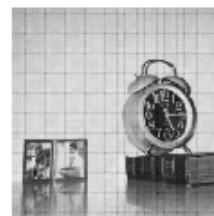
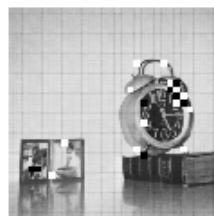
Imagen original (8 bits por píxel).



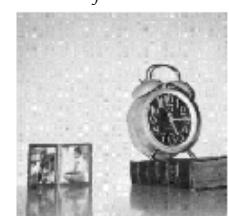
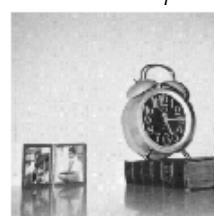
Transformación DLMT directa. 24 bits por dato en resultado final.



Transformación DLMT transpuesta e inversa. 24 bits por dato en resultado final.



Transformación DLMT inversa. 24 bits por dato en resultado final.



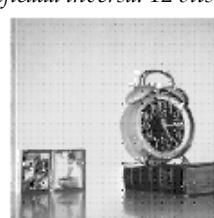
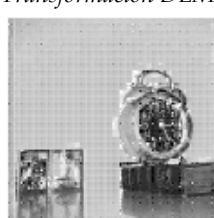
Transformación DLMT modificada directa. 16 bits por dato en resultado final.



Transformación DLMT modificada transpuesta directa. 16 bits por dato en resultado final.



Transformación DLMT modificada inversa. 12 bits por dato en resultado final.



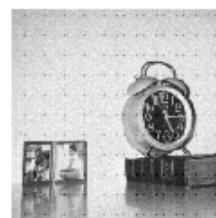
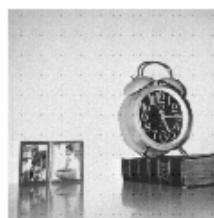
Transformación DLMT modificada transpuesta inversa. 9 bits por dato en resultado final.

Figura 32. Imágenes de las pruebas realizadas en Matlab para la DLMT.

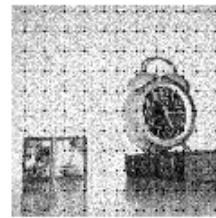
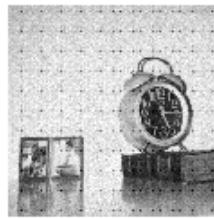
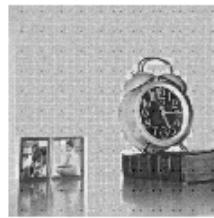
Como se puede ver en las figuras, es necesario almacenar el resultado final con más bits que la imagen original (en algunos casos hasta el triple) para conservar la calidad

de la imagen y que no se distorsione debido a la perdida de precisión de los datos en los redondeos.

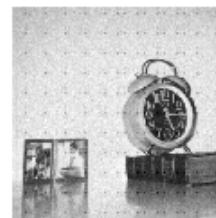
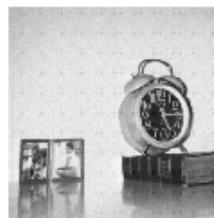
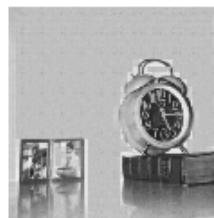
También se han ejecutado estas pruebas utilizando las transformaciones DCT-I, DCT-II (la utilizada en JPEG), DCT-III (aplicando primero IDCT y recuperando con FDCT) y DCT-IV, llegando a la conclusión de que la mejor transformación de las ensayadas para la compresión de imágenes en hardware es la DCT-II, ya que es la que muestra mejor calidad para el mismo número de bits en el resultado final e incluso conserva una calidad aceptable reduciendo el número de bits por dato.



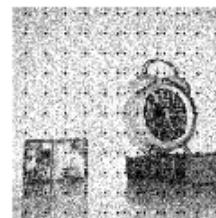
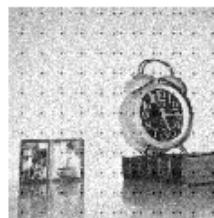
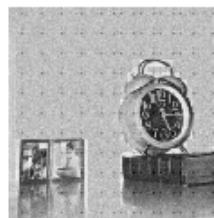
Transformación DCT-I. 8 bits por dato en resultado final.



Transformación DCT-I. 6 bits por dato en resultado final.



Transformación DCT-II. 8 bits por dato en resultado final.



Transformación DCT-II. 6 bits por dato en resultado final.

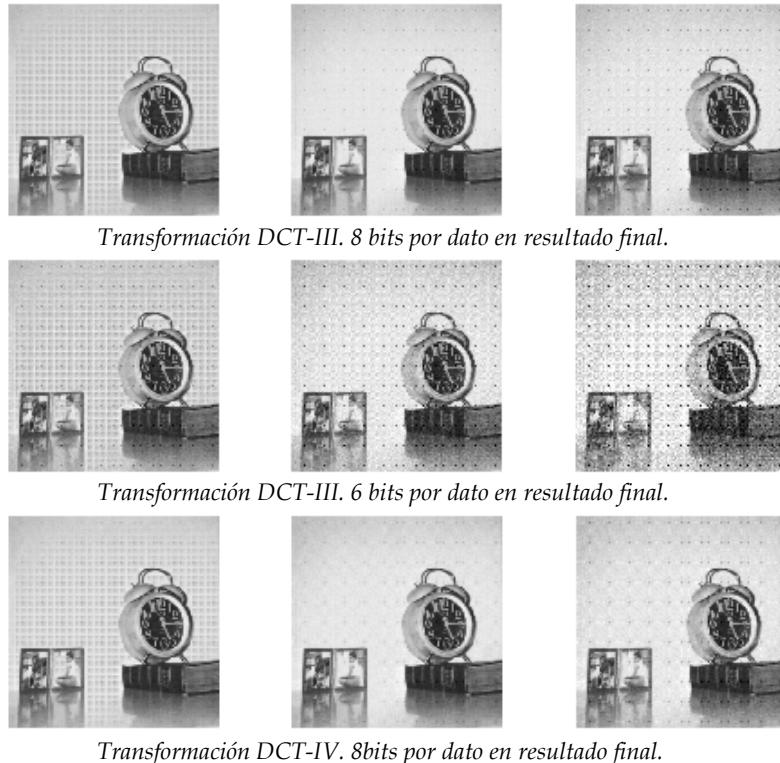


Figura 33. Imágenes de las pruebas realizadas en Matlab para las 4 DCTs.

IV. Descripción del sistema implementado:

IV.1 Sistema de pruebas en Virtex5

El compresor se ha implementado en la FPGA Spartan6 de la capa de procesamiento HiReCookie [16] de la plataforma Cookies [17]. Cookies es una plataforma para WSN cuya principal característica es la modularidad. Cada nodo se compone de varias capas intercambiables (de comunicaciones, alimentación, procesamiento y/o sensores). Y HiReCookie es una placa de procesamiento diseñada para permitir tareas de procesamiento complejas, como precisamente la compresión de imágenes, orientada a bajo consumo. Esta placa tiene como principal elemento de procesamiento una FPGA Spartan6 donde se ha implementado el compresor.

Pero antes de desarrollar el sistema definitivo en la HiReCookie, se ha depurado el sistema en una placa de desarrollo XUPV5-LX110T de Xilinx, que cuenta con una FPGA Virtex5. El sistema hardware-software se compone de:

- Cámara de vídeo TCM38230MD (A) de Toshiba (para la generación de datos) sobre la placa diseñada en el proyecto “Implementación hardware de las transformadas discretas DCT y DLMT para compresión de imágenes y su estudio comparativo” [18]. Se ha fabricado un cable conector especial para conectar esta placa con los pines de expansión de la placa de desarrollo.
- Procesador MicroBlaze instanciado en la FPGA. Que ejecuta las funciones Software del sistema.
- Bloque Hardware (periférico al MicroBlaze) “camera_capture” que incluye los procesos de configuración de la cámara, captación y sincronización de los datos de la cámara, transformación discreta seleccionable y parametrizable, y compresión sin pérdidas (RLE modificado y Huffman). Gran parte de este código se ha adaptado al fabricante Xilinx de los diseñados y desarrollados en el proyecto [18]. Se ha modificado el control para adaptarlo a las estrategias de bajo consumo necesarias en las WSN, se ha añadido el descompresor sin pérdidas y se ha integrado en el *wrapper* generado por EDK para comunicación con MircoBlaze mediante registros y FIFO de lectura (salida de datos del periférico) a través de bus PLB V46.
- Bloque Hardware (periférico al MicroBlaze) “compresor” que implementa la compresión sin pérdidas. Con comunicación con MircoBlaze mediante registros y FIFOs de escritura y lectura a través de bus PLB V46. Utilizado para las pruebas sin datos generados por la cámara.
- Bloque Hardware (periférico al MicroBlaze) “huff_decoder” que realiza la descompresión Huffman. Dado que el objetivo de este proyecto es desarrollar un compresor, no un descompresor, este bloque se ha implementado para poder observar los resultados obtenidos por el compresor desarrollado. Este bloque no está optimizado para bajo consumo de energía. Comunicación con MircoBlaze mediante registros y FIFOs de escritura y lectura a través de bus PLB V46.
- Función Software “sw_comprime_bloque” que realiza la compresión sin pérdidas. Utilizada para comparar los resultados del compresor Hardware y poder medir la aceleración que supone realizar la compresión mediante Hardware en lugar de mediante Software.
- Funciones Software para la comunicación y control de los bloques Hardware.
- Funciones Software para la descompresión total y visualización de los resultados.
- Bloque Hardware (periférico al MicroBlaze) TFT de la lista de IPs que ofrece Xilinx en su herramienta de desarrollo EDK. Gestiona el puerto de salida DVI para la visualización de las imágenes en un monitor VGA.
- Monitor VGA para la visualización de las imágenes.

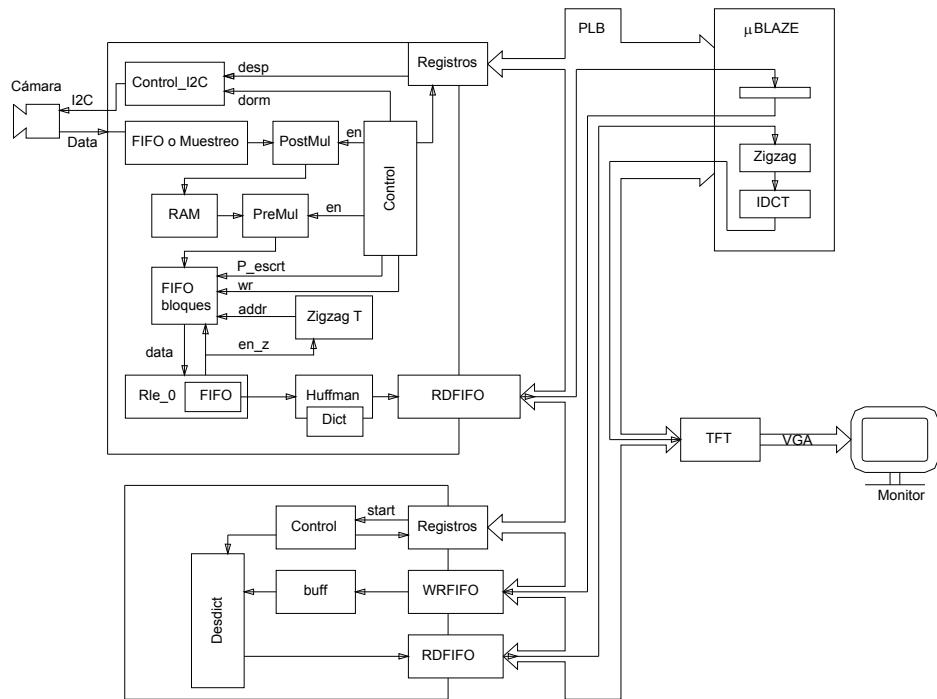


Figura 34. Esquema del sistema de depuración hardware-software implementado en la Virtex5.

Bloque camera_capture:

Se compone de los sub-bloques:

- “Control_i2c”: Genera las señales I2C para configurar la cámara, dormirla y despertarla. Creado en el proyecto [18].
- FIFO con relojes de escritura y lectura distintos para la sincronización de los datos generados por la cámara (que funciona con un reloj proveniente de un oscilador distinto al de la FPGA). Adaptado del proyecto [18]. Opcional. Esta forma de sincronización es válida para cualquier relación de frecuencias entre el reloj de la cámara y del comprresor. Pero para relaciones en las que la frecuencia del comprresor es al menos 4 veces mayor que la de la cámara se ha optado por un muestreo de las señales de la cámara como método de sincronización para ahorrar esta FIFO.
- “PostMul” y “PreMul”: Realizan las multiplicaciones matriciales de la transformación. Adaptados del proyecto [18].
- Memorias BRAM internas para el almacenamiento de datos intermedios.

- “FIFObloques”: FIFO especial en la que cada elemento en lugar de ser un único registro es un bloque de 8x8 elementos. Sirve de almacenamiento de datos intermedio entre la transformación y la compresión sin pérdidas.
- “zigzag” / “zigzagT”: Genera las direcciones de memoria necesarias para la lectura en zigzag de los resultados de la transformación. Las primeras versiones se realizaron con el bloque “zigzag” pero posteriormente se sustituyó por “zigzagT” que realiza una lectura de forma transpuesta ahorrando el control necesario para ordenar los resultados de “PreMul” (que genera los datos ordenados por columnas en lugar de por filas).
- “rle_0”: Cuenta el número de ceros consecutivos que aparecen en la trama de datos para eliminarlos si finaliza el bloque o volverlos a poner en la trama si aparece un siguiente elemento distinto de cero.
- “huffman_top”: Realiza la codificación Huffman de la trama de datos. Utiliza un diccionario fijo que se debe especificar en el package “huffman_pkg.vhd” cuando se define el genérico useROM como FALSE o en un fichero “dict.coe” de inicialización del bloque ROM que implementa el diccionario si se define el genérico useROM como TRUE. Se detalla su estructura y funcionamiento en el siguiente sub-apartado de este subcapítulo.

FSM COMPRESOR COMPLETO

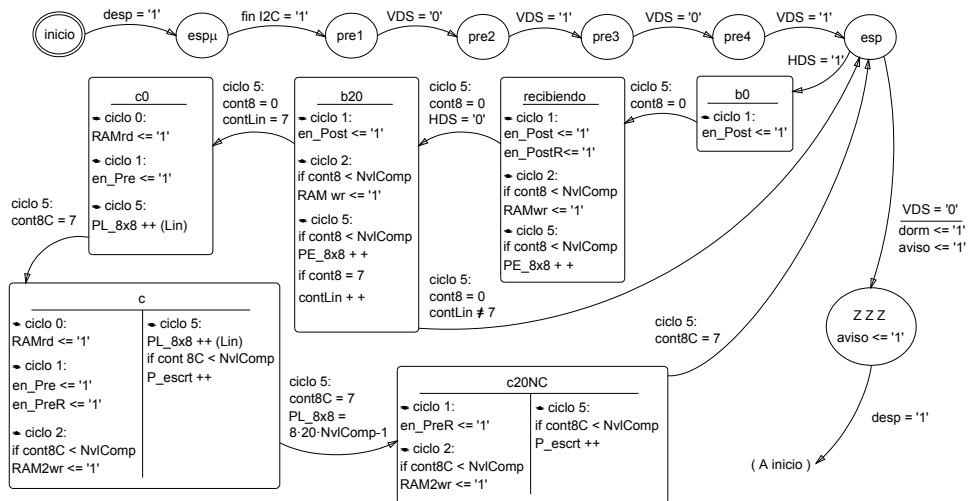


Figura 35. Diagrama de estados del control.

En la figura anterior se muestra un diagrama del nuevo control que se ha diseñado para adaptarlo a las estrategias de bajo consumo necesarias en las WWSN. Tras el inicio se espera a que se configure y encienda la cámara (cuando esto ha sucedido el bloque “Control_i2c” genera un pulso en la señal finI2C) y después se suceden varios estados que sirven para vigilar las señales provenientes de la cámara y saltar a los estados de procesamiento justo cuando la imagen esté lista. Tras haber procesado una imagen el “Control_i2c” genera la orden de dormir y se lleva el sistema a un estado de espera hasta que se reciba la orden de despertar la cámara. Cuando se despierta la cámara el control vuelve a los estados iniciales de vigilancia de las señales de la cámara para saltar al estado de procesamiento cuando la imagen esté lista.

Se ha configurado la cámara para que produzca imágenes en formato QQVGA a 30 fps, que produce píxeles con una frecuencia de 6,25 MHz. La frecuencia a la que funciona la FPGA del sistema es de 100 MHz. Se ha colocado un divisor de frecuencia al periférico de modo que funcione a 50 MHz, lo que supone que se recibe un píxel cada 8 ciclos de reloj.

Cuando comienza a recibirse una imagen, en el estado “b0” se realzan las multiplicaciones correspondientes a la primera fila del primer bloque hasta obtener los primeros resultados de los multiplicadores y acumuladores (MAC). Tras recibir los 8 primeros píxeles se pasa al estado “recibiendo” donde se va realizando la post-multiplicación matricial de la primera línea de los bloques y almacenando los resultados del anterior bloque en la RAM interna. Al acabar la línea se pasa al estado “b20” donde se guardan los resultados de la primera línea del último bloque (de la primera línea de bloques). Al finalizar el estado “b20” se incrementa “contLin” (contador del número de líneas) y se vuelve al estado de espera (“esp”). Línea por línea va repitiendo este proceso. Cada 8 líneas, cuando una línea de bloques completos está lista, en lugar de volver al estado de espera se pasa a los estados “c0”, “c” y “c20NC” donde se va leyendo la RAM interna por columnas y se realiza la pre-multiplicación matricial de forma análoga a la post-multiplicación matricial realizada por los estados “b0”, “recibiendo” y “b20”. Con la diferencia de que aquí se procesan las 8 columnas de los bloques (en lugar de una sola fila). Al acabar el estado “c20NC” se vuelve al estado de espera antes de que llegue la siguiente línea.

En la siguiente figura se muestra un esquema de un bloque de procesamiento (“PostMul” o “PreMul”) desarrollado en el proyecto [18]. Este bloque realiza una multiplicación matricial completa de matrices de 8x8. Para realizar la multiplicación propuesta y ahorrar el cálculo de los coeficientes correspondientes a las altas frecuencias espaciales el número de multiplicadores y acumuladores se reduce a la mitad. De forma que al realizar la post-multiplicación matricial solo se multiplican los 4 primeros coeficientes de cada fila de la matriz C^T y en la pre-multiplicación matricial solo los 4 primeros coeficientes de cada columna de C.

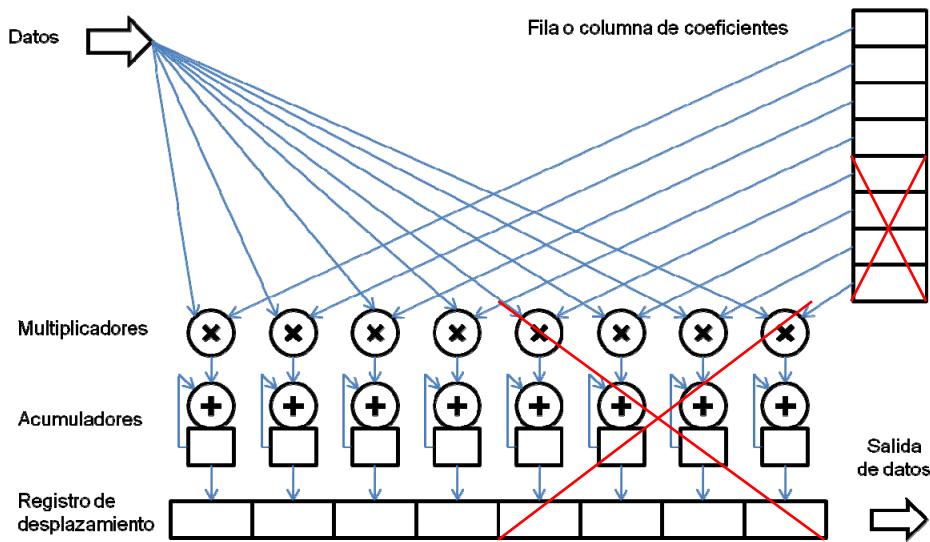


Figura 36. Esquema de un bloque de procesamiento [18].

Bloque compresor:

Se compone de los sub-bloques “FIFObloques”, “zigzag” / “zigzagT”, “rle_0” y “huffman_top”.

“FIFObloques” permite independizar el proceso de transformación de la compresión sin pérdidas.

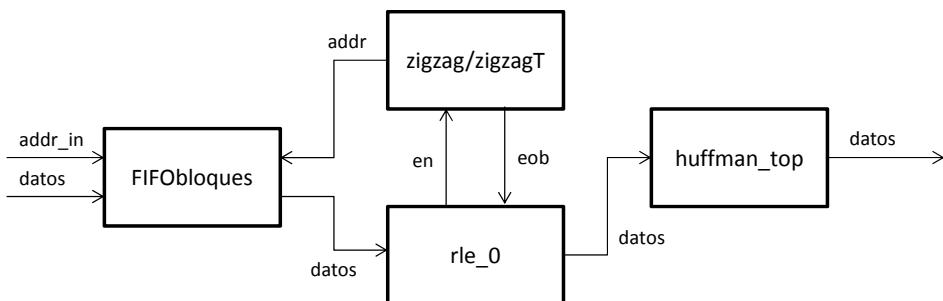


Figura 37. Diagrama de bloques del comprresor.

El bloque “rle_0” habilita el bloque “zigzag” (señal en) cada vez que necesita un nuevo dato, y “zigzag” genera la dirección de memoria necesaria (señal addr) para que “FIFObloques” proporcione el dato correcto. Cuando se ha terminado de leer un bloque de 8x8 datos, “zigzag” además activa la señal eob para indicar que se ha finalizado el bloque de datos.

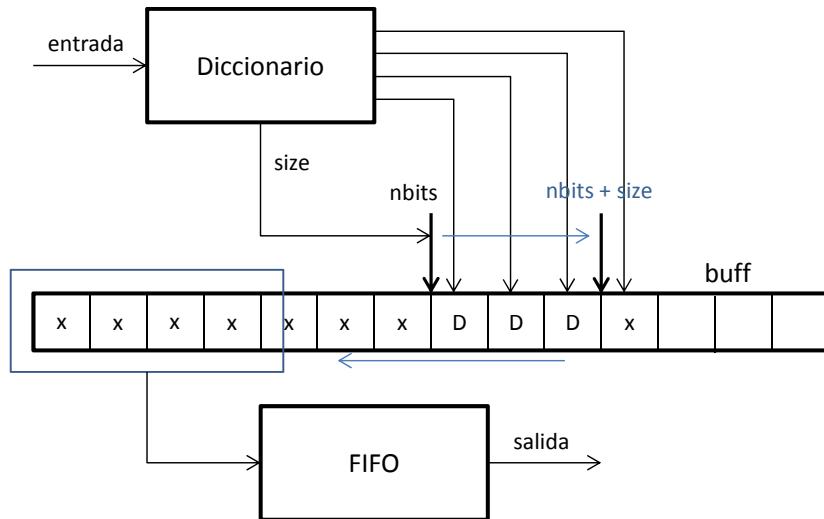


Figura 38. Esquema de funcionamiento de "huffman_top".

El diccionario es una *look up table* (LUT) en la que el dato de entrada direcciona una posición de memoria de la ROM donde se almacena su código Huffman (con bits de relleno hasta la longitud máxima) y la longitud del código (*size*). El código se introduce en un *buffer* (*buff*) a partir de la posición indicada por el registro *nbits*, al que se le suma el valor de *size*. Si la nueva longitud calculada es mayor o igual que la longitud de las palabras de salida (que es parametrizable) el *buffer* envía los bits de la izquierda y realiza un desplazamiento del resto de los bits a las posiciones liberadas.

IV.2 Opciones parametrizables para optimizar el consumo energético

Optimizar el hardware implementado en una FPGA para obtener bajo consumo no es algo trivial. Las primeras medidas que se han de tomar es procurar instanciar los elementos hardware embebidos en la FPGA tales como multiplicadores, DSPs y bloques de memoria que han sido ya optimizados por el fabricante. Para realizar un buen uso de estos elementos se ha de estudiar la tecnología del dispositivo y las recomendaciones del fabricante [19]. En este caso, Xilinx recomienda utilizar sus bloques de memoria embebida BRAM para memorias mayores de 4Kbits y bloques de RAM distribuida para implementar memorias menores de 4Kbits. Xilinx también recuerda que es importante deshabilitar los bloques de memoria cuando no se estén leyendo ni escribiendo.

Pero aparte de estas recomendaciones básicas, la arquitectura escogida y el control diseñado influirán notablemente en el consumo del sistema final. No es sencillo determinar a priori que arquitectura o control supondrán un menor consumo. En estos

diseños intervienen variables y opciones que pueden ser contradictorias como por ejemplo la elección de la frecuencia de funcionamiento. Reducir la frecuencia reduce la potencia consumida, pero a veces puede resultar más beneficioso aumentar la frecuencia para reducir el tiempo de procesamiento y poder dejar el sistema suspendido en un modo de bajo consumo. Existen simuladores que permiten obtener una cierta idea del resultado final. Pero la precisión de estos simuladores aún no es muy buena y la mejor forma de saber que diseño dará mejor resultado es medir el consumo real una vez implementados. Por este motivo se han dejado las siguientes opciones de diseño parametrizables en los códigos VHDL:

Almacenamiento de datos intermedios entre la transformación y el compresor. Debido a que el compresor tarda un número de ciclos variables según los datos a procesar se puede optar por dos filosofías opuestas:

- Almacenar los datos en una memoria lo suficientemente grande para que no llegue a vaciarse e ir procesando los datos mientras se sigue llenando la memoria. Esta opción permite parallelizar los bloques de procesamiento y simplifica el control, pero supone la implementación de grandes bloques de memoria. Al parallelizar la ejecución se reduce el tiempo de procesamiento lo que puede permitir o bien reducir la frecuencia o llevar el sistema o parte del sistema a un modo de bajo consumo hasta que se reciban los siguientes datos.
- Procesar los datos “*just-in-time*” según son generados. Esta opción complica bastante el control para permitir la perfecta sincronización entre bloques que tienen tiempos de procesamiento distintos y además variables según los datos. Pero reduce enormemente el tamaño de las memorias intermedias.

Para permitir mayor flexibilidad en esta opción de diseño se ha creado una memoria FIFO especial donde cada elemento en lugar de ser un único registro es un bloque de 8x8 elementos (resultados de la transformación que se van a comprimir). Esta FIFO parametrizable en profundidad permite generar soluciones intermedias a las dos descritas anteriormente.

Reutilización de multiplicadores en la transformación. Al no realizarse de forma simultánea la post-multiplicación y la pre-multiplicación (incluso si se realizaran ambos procesos simultáneamente pueden realizarse las multiplicaciones dato por coeficiente en distintos ciclos de reloj) es posible utilizar el mismo multiplicador para las dos. Esta opción aumentaría la lógica de control para conducir los datos correctamente al lugar que les corresponde, pero ahorraría multiplicadores. Pero en el caso de las FPGAs a utilizar de Xilinx (Virtex5 y Spartan6) disponen de elementos DSP en lugar de multiplicadores aislados. Estos elementos DSP incluyen también el sumador y el acumulador para realizar las operaciones MAC (multiplicación y acumulación). Como se requiere que cada acumulador sea independiente se han instanciado elementos DSP separados para las pre-multiplicaciones y las post-multiplicaciones.

Propuesta de red neuronal distribuida en una WVSN

Propuesta de red neuronal distribuida en una WVSN

Como se ha visto, las redes neuronales son algoritmos de computación principalmente en paralelo. Sin embargo los microprocesadores tienen limitadas sus capacidades de procesamiento en paralelo, siendo más eficaces ejecutando algoritmos principalmente secuenciales. Lo que lleva a que la mayoría de las implementaciones de las redes neuronales acaben siendo mucho más lentas de lo que podrían ser si se aprovechase enteramente el paralelismo del algoritmo.

Al intentar implementar estos algoritmos en hardware se encuentra otro problema: la operación que se ha de ejecutar masivamente en paralelo es la multiplicación. Los multiplicadores hardware son grandes, lentos y costosos si se comparan con otros circuitos electrónicos básicos como sumadores, multiplexores, etc. Lo que hace que incluso en implementaciones hardware el número de multiplicadores esté limitado y en consecuencia también esté limitada la capacidad de aprovechar el enorme paralelismo del algoritmo.

En aplicaciones de procesamiento de imágenes los píxeles de la imagen no se reciben todos de golpe, si no que se van recibiendo de uno en uno. Además en una WVSN no hay un único elemento de procesamiento. Hay distribuidos numerosos pequeños elementos de procesamiento. La propuesta que se hace en este proyecto se aprovecha de estos dos hechos. Primero descomponiendo la red neuronal en trozos que se implementarán cada uno en un nodo distinto de la red de sensores. Y segundo adjudicando sólo un multiplicador a cada neurona con un acumulador para ir procesando las entradas (píxeles) de una en una según van llegando.

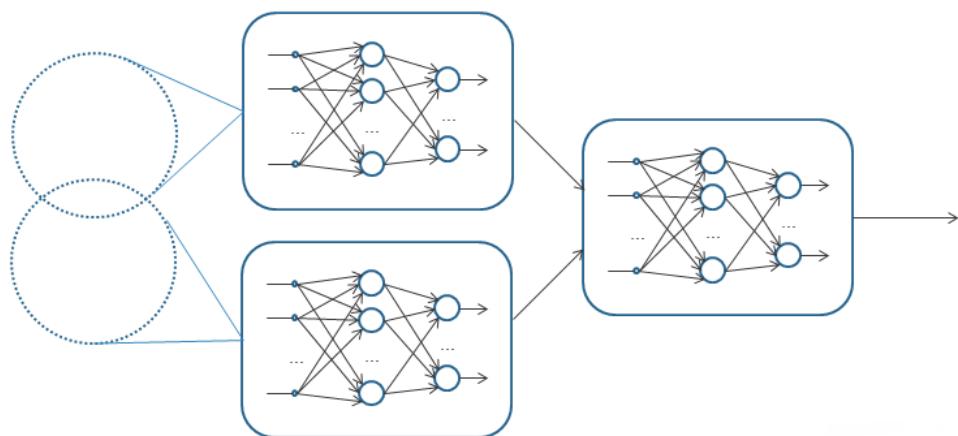


Figura 39. Esquema propuesto. La red neuronal se procesa de forma distribuida a lo largo de la red de sensores.

Cada rectángulo de la figura representa un nodo de la red de sensores inalámbrica. Cada nodo posee una unidad de procesamiento, en este caso una FPGA, donde se implementará una parte de la red neuronal. Los dos nodos de la izquierda representan dos nodos con cámara cuyas imágenes se solapan. Como se explicó anteriormente, si se desea que un grupo de cámaras captén un gran área sin huecos, es inevitable que se solapen las imágenes captadas por las cámaras. Una gran ventaja de este sistema de procesamiento/compresión distribuido es que el nodo de la derecha es capaz de detectar y procesar esa redundancia generada por el solapamiento de las imágenes evitando que se propague por el resto de la red.

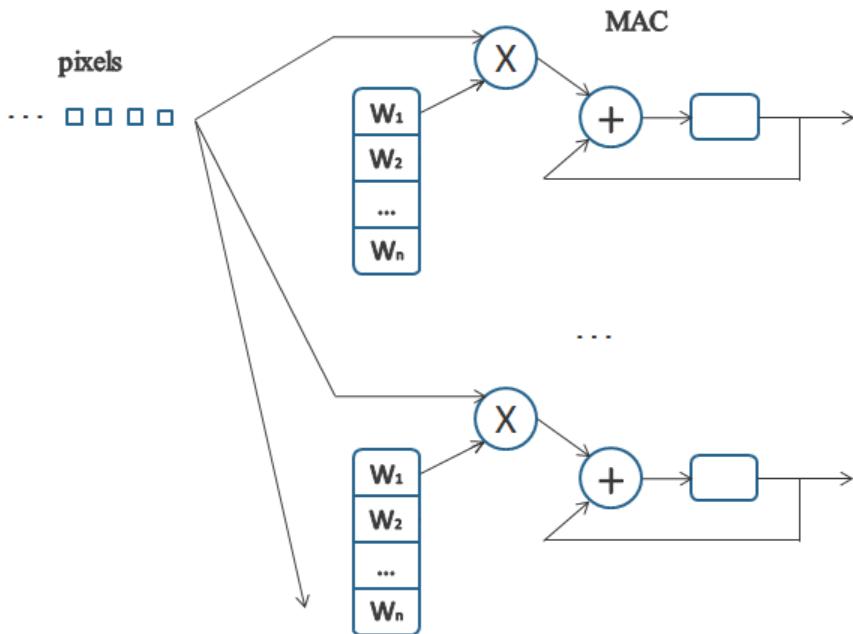


Figura 40. Esquema de la implementación de las neuronas.

Cada neurona se implementará con un multiplicador y acumulador (MAC). Al recibir el primer píxel cada neurona multiplicará éste por el primero de sus pesos y se almacenará el resultado en el acumulador. Al recibir el segundo píxel cada neurona lo multiplicará por el segundo de sus pesos, sumará este resultado al acumulado y almacenará la suma. Y así sucesivamente hasta que se reciba el último píxel del bloque y se pueda obtener la salida de las neuronas de la primera capa.

La red neuronal que se propone implementar está basada en la propuesta por Aran Namphol, Mohammed Arozullah y Steven Chin en el artículo “Higher order data compression with neural networks” [20]. Donde se describe una red neuronal jerárquica como se muestra en las figuras siguientes. Este sistema se compone de tres secciones: la sección de la capa de entrada, la sección de las capas ocultas, y la sección de la capa de salida.

ción de la capa de salida. La sección de la capa de entrada consiste en varias subredes que sirven como entrada a la red. La sección de las capas ocultas consta de tres partes: combinador, compresor y descombinador. El combinador actúa como multiplexor para las subredes de entrada. El número de nodos del combinador es menor que el número de entradas. El compresor comprime aún más la salida del combinador y produce la representación interna comprimida de las imágenes. Despues se descomprime y por último el descombinador actúa como demultiplexor. Además al ser la red simétrica y jerárquica pueden entrenar por separado primero las capas de fuera y posteriormente las de dentro. Para entrenar las capas de dentro utilizan como entrada y salida los valores que toman las neuronas de la capa oculta del entrenamiento de las capas de fuera. A esta forma de entrenar la han llamado Nested Training Algorithm (NTA).

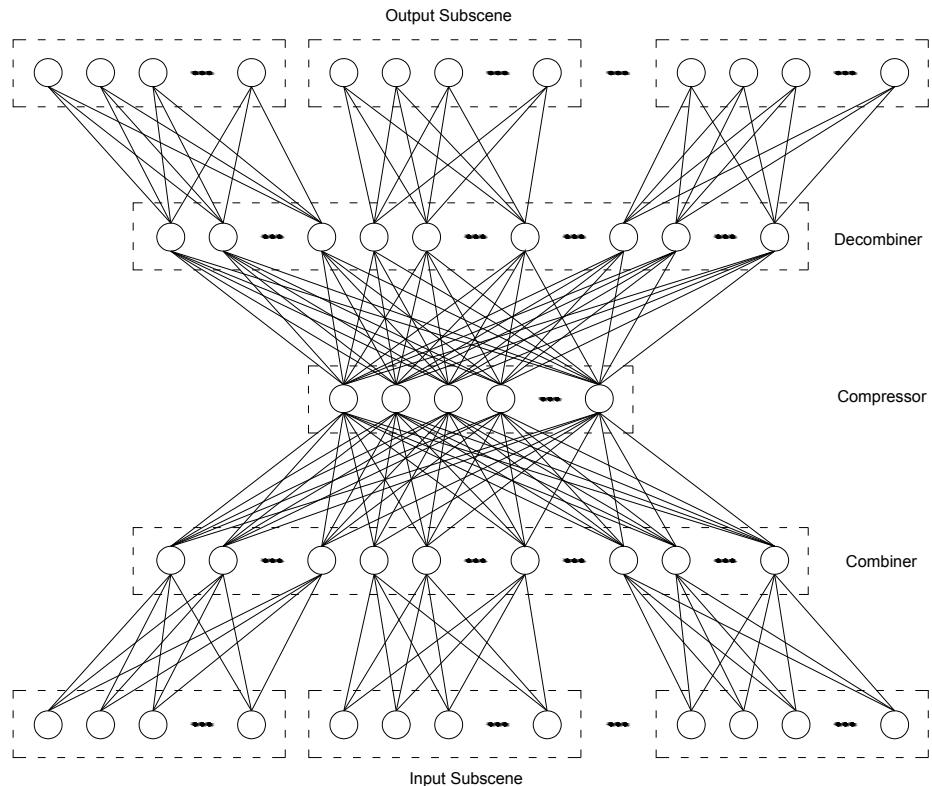


Figura 41. Red neuronal propuesta en [20].

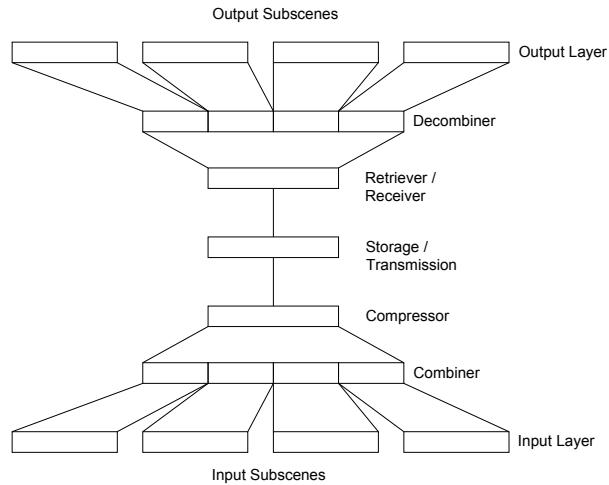


Figura 42. Esquema de compresión y descompresión utilizando la red neuronal propuesta en [20]

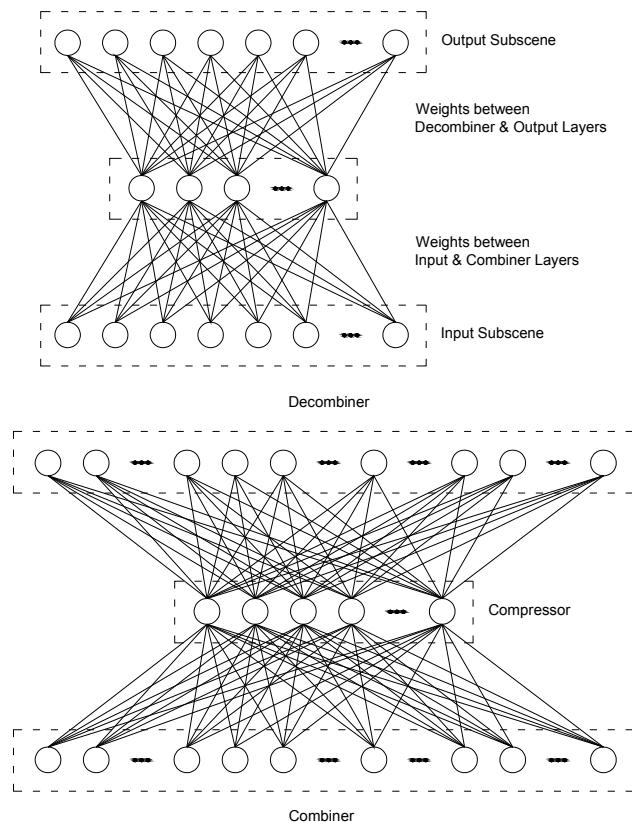


Figura 43. Capas de fuera (arriba) y capas de dentro (abajo) de la red neuronal.

Otra ventaja que tendrá la utilización de redes neuronales distribuidas en WVSN es su mayor seguridad frente a intrusos que pretenden obtener las imágenes transmitidas interceptando la comunicación entre dos nodos. No podrá reconstruir la imagen ya que la clave de reconstrucción está dispersa y distribuida por toda la red. Se necesitará del decodificador que se entrenó junto al codificador para poder recuperar las imágenes. Los nodos de la red inalámbrica se estarán enviando mensajes que solo tendrán sentido para ellas. Resolviendo parte del problema de seguridad que presentan las WVSN sin necesidad de cifrado.

Resultados, conclusiones y líneas futuras

Resultados, conclusiones y líneas futuras

I. Resultados

Uno de los principales resultados obtenidos es el compresor de imágenes optimizado para bajo consumo para el sistema HiReCookie. Ya que hasta entonces no se disponía de ninguno para esta plataforma.

Se ha obtenido también un sistema de compresión y descompresión, implementado en la placa de desarrollo XUPV5-LX110T de Xilinx (con la Virtex5) que permite la visualización de las imágenes descomprimidas en un monitor para probar, depurar y ajustar el compresor desarrollado.

Los códigos tanto de Matlab como VHDL desarrollados son altamente parametrizables, lo que permite utilizarlos para implementar compresores con otro tipo de transformaciones u otras estrategias distintas de compresión y optimización.

Se ha comparado el sistema de compresión sin pérdidas hardware con los mismos algoritmos ejecutados en MicroBlaze, comprobando y midiendo la aceleración que produce la solución propuesta frente al puramente software. Para las condiciones básicas (instrucciones y datos ubicados en RAM externa DDR2 con caché y 1 Kbyte de Heap y Stack) se ha medido el tiempo que tarda en comprimir un bloque de prueba: 14045 ciclos para el compresor software y 1516 ciclos para el hardware. Hay que tener en cuenta que gran parte de los ciclos se pierde en ejecutar las instrucciones de activación y lectura, el tiempo real empleado por el compresor hardware será menor. En la tabla de la figura 46 se muestran el resto de medidas realizadas para distintas condiciones.

```
C:\Xilinx\13.4\ISE_DS\ISE\bin\nt64\xtclsh.exe
Terminal requirements :
  (i) Processor's STDOUT is redirected to the MDM
  (ii) Processor's STDIN is redirected to the MDM
    Then, text input from this console will be sent to the MDM's UART port.
    NOTE: this is a line-buffered console and you have to press "Enter"
          to send a string of characters to the MDM.

out_words: 1
8A179C0D
100010100001011110011100000001101
out_words: 1
8A179C0D
100010100001011110011100000001101
Descomprimiendo...
out_words: 9
  3
  2
 -2
 -1
  1
 -1
  0
  0
 -1
```

Figura 44. Captura de pantalla del terminal de comunicación con el mdm del microBlaze.

En la figura 44 se muestran los resultados de comprimir y descomprimir el bloque de prueba de la figura 45. Primero se muestran los resultados del compresor software en hexadecimal y binario, después los del compresor hardware también en hexadecimal y binario y finalmente la secuencia descomprimida.

3	2	-1	0
-2	1	0	0
-1	-1	0	0
0	0	0	0

Figura 45. Bloque para pruebas de compresión y descompresión.

Config:	Heap y Stack	Code caché:	Data caché:	Code:	Data:	SW:	HW:
Basica						14045	1516
	40KB	0	0	IntBRAM	IntBRAM	3515	1578
		0	1	IntBRAM	IntBRAM	2852	1083
		1	1	DDR2	IntBRAM	11785	1545
		1	0	DDR2	IntBRAM	12202	2030
		0	1	DDR2	IntBRAM	32179	7655
		0	0	DDR2	IntBRAM	32170	7654
		0	1	IntBRAM	DDR2	3836	1067
		0	0	IntBRAM	DDR2	6731	1608
		1	1	DDR2	DDR2	14027	1514
		0	1	DDR2	DDR2	36492	7625
		1	0	DDR2	DDR2	15243	1986
		0	0	DDR2	DDR2	36479	7698
	1KB	0	0	IntBRAM	IntBRAM	2103	1031
		1	0	DDR2	IntBRAM	11533	1429
		0	0	DDR2	IntBRAM	32193	7660
		0	1	IntBRAM	DDR2	2439	1033
		0	0	IntBRAM	DDR2	5322	1050
		1	1	DDR2	DDR2	13353	1521
		0	1	DDR2	DDR2	36489	7617
		1	0	DDR2	DDR2	14371	1528
		0	0	DDR2	DDR2	36484	7695
	10KB	1	1	DDR2	DDR2	13472	1522
		0	0	DDR2	DDR2	36489	7694

Figura 46. Tabla de resultados de las medidas de tiempo de procesamiento de los compresores hardware y software para distintas condiciones.

Para poder comprender el resultado de la compresión se muestra una parte del diccionario utilizado en la siguiente figura.

Símbolo	Código
3	100010
2	10000
1	001
0	000
-1	11
-2	1011
EOB	01

Figura 47. Fragmento del diccionario utilizado para la codificación Huffman.

Descomponiendo los resultados en binario mostrados utilizando el diccionario se puede comprobar el correcto funcionamiento. Nota: recuérdese que los bloques se leen en zigzag.

100010 (3) 10000 (2) 1011 (-2) 11 (-1) 001 (1) 11 (-1) 000 (0) 000 (0) 11 (-1) 01 (EOB)

Una vez ajustado y depurado el sistema en la placa de desarrollo se han portado los códigos VHDL de los realizados para Virtex5 a los optimizados para Spartan6.

Se ha medido el consumo del compresor implementado en la Spartan6 de la capa de procesamiento HiReCookie. Para medir el consumo HiReCookie dispone de un amplificador diferencial conectado a una resistencia de *shunt* colocada en la línea de alimentación del núcleo de la FPGA según se muestra en la figura 48. En la figura 49 se muestra la medida a la salida del amplificador y en la tabla de la figura 50 se convierten los valores en milivoltios a consumo de corriente en miliamperios y potencia en miliwattios (la tensión de alimentación del núcleo de la FPGA es de 1,2V).

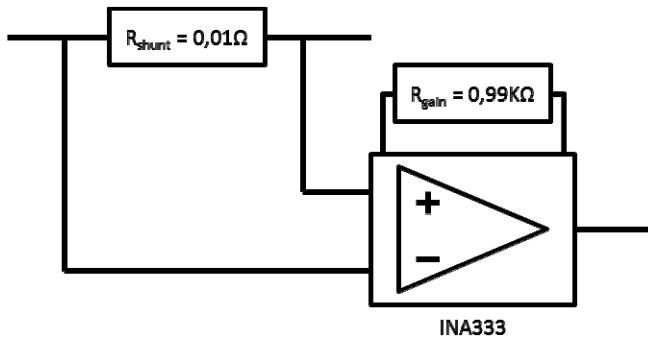
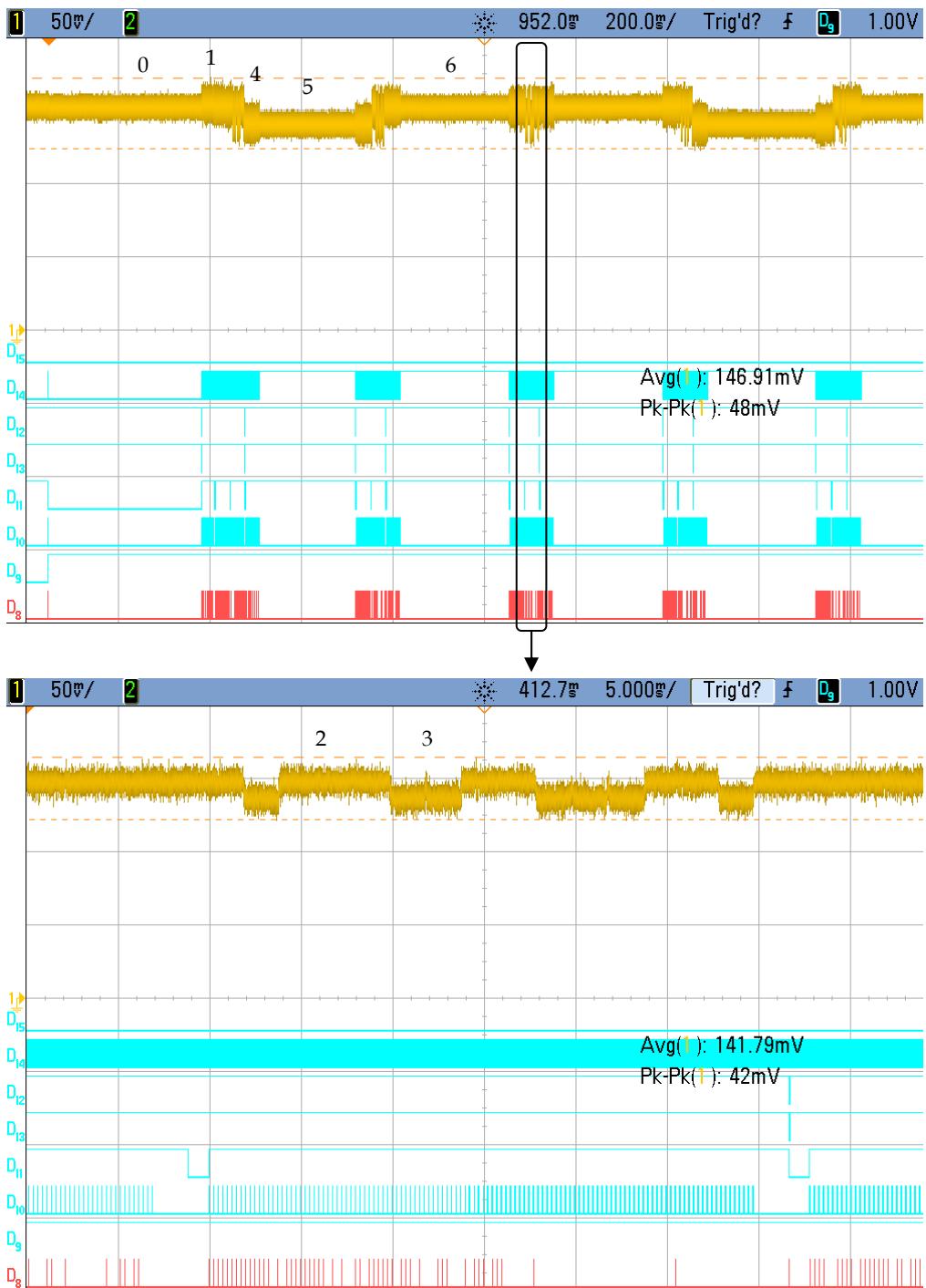


Figura 48. Esquema del circuito de medida del consumo.



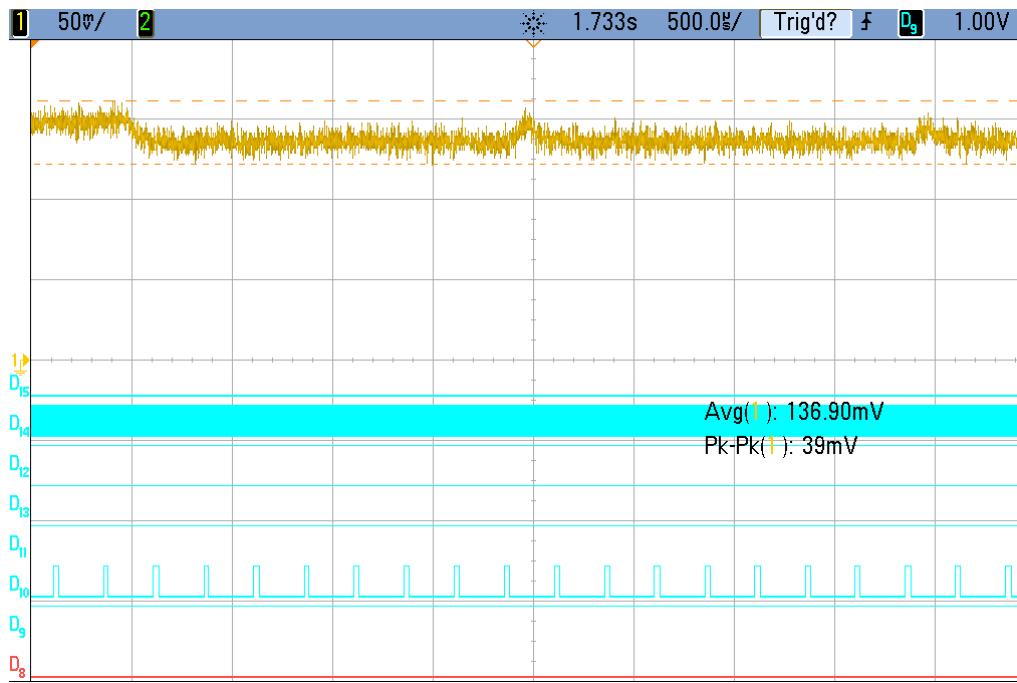


Figura 49. Capturas de osciloscopio a distintos zooms. La señal analógica amarilla representa la medida de la salida del amplificado, los pines D12 y D13 la comunicación I2C con la cámara, el pin D11 la sincronización vertical (HD) y el pin D10 la sincronización horizontal (HD).

Tramo	0	1	2	3	4	5	6
Medido (mV)	150	151	140,1	151,1	139,4	138,3	150
Corriente (mA)	147,1	148	137,4	148,1	136,7	135,6	147,1
Potencia (mW)	176,5	177,6	164,8	177,8	164	162,7	176,5

Figura 50. Tabla de valores promedios medidos.

En la figura 49 se observa primero como se va despertando y durmiendo la cámara. En la siguiente captura se muestra el procesamiento de una imagen (marcada por la señal de sincronismo vertical D11). La última captura es un zoom sobre varias líneas de imagen. Cada 8 líneas se realiza la pre-multiplicación matricial de una línea de bloques y su posterior compresión.

La tabla de la figura 51 muestra los recursos de la FPGA (Spartan6 XC6SLX150 de la HiReCookie) utilizados para implementar el compresor desarrollado. Con los parámetros y opciones de diseño actuales se obtiene una muy baja ocupación de la FPGA.

Slice Logic Utilization	Used	Available	Utilization
Number of Slice Registers	330	184,304	1%
Number used as Flip Flops	330		
Number used as Latches	0		
Number used as Latch-thrus	0		
Number used as AND/OR logics	0		
Number of Slice LUTs	534	92,152	1%
Number used as logic	477	92,152	1%
Number using O6 output only	382		
Number using O5 output only	0		
Number using O5 and O6	95		
Number used as ROM	0		
Number used as Memory	50	21,68	1%
Number used as Dual Port RAM	48		
Number using O6 output only	24		
Number using O5 output only	0		
Number using O5 and O6	24		
Number used as Single Port RAM	0		
Number used as Shift Register	2		
Number using O6 output only	2		
Number using O5 output only	0		
Number using O5 and O6	0		
Number used exclusively as route-thrus	7		
Number with same-slice register load	7		
Number with same-slice carry load	0		
Number with other load	0		
Number of occupied Slices	170	23,038	1%
Number of LUT Flip Flop pairs used	537		
Number with an unused Flip Flop	254	537	47%
Number with an unused LUT	3	537	1%
Number of fully used LUT-FF pairs	280	537	52%
Number of unique control sets	36		
Number of slice register sites lost to control set restrictions	164	184,304	1%
Number of bonded IOBs	52	338	15%
Number of RAMB16BWERS	2	268	1%
Number of BUFG/BUFGMUXs	1	16	6%
Number used as BUFGs	1		
Number used as BUFGMUX	0		
Number of DSP48A1s	8	180	4%

Figura 51. Tabla de utilización de recursos de la FPGA para el compresor.

Se ha medido también el consumo del mismo compresor pero multiplicando por las matrices de coeficientes de la DCT completas. La figura 52 muestra la captura de osciloscopio de las medidas tomadas para este compresor. La principal diferencia observada son los picos de consumo producidos durante la pre-multiplicación de cada línea de bloques (que se realiza cada 8 líneas de la imagen).

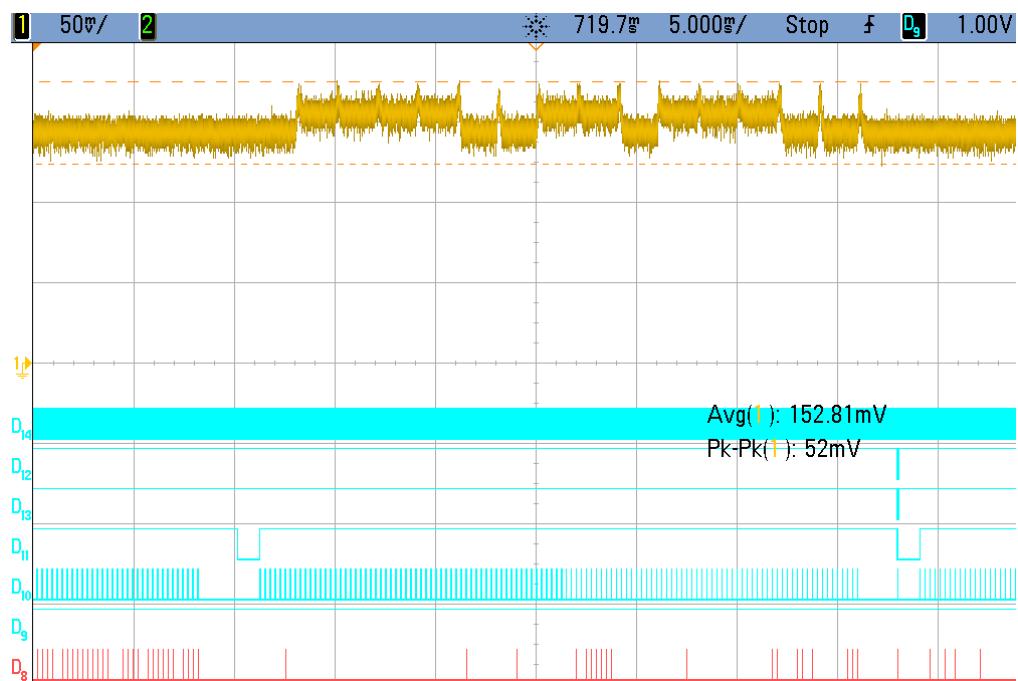


Figura 52. Captura de osciloscopio del compresor realizando las multiplicaciones matriciales completas.

Por otra parte se han simulado en Matlab las redes neuronales propuestas. Se han escogido para este propósito siete imágenes similares de motivo militar imitando una posible aplicación de vigilancia de un campo de batalla. Se ha escogido así para aprovechar la ventaja de las redes neuronales a la hora de aprender y explotar las características de un tipo de imágenes frente a codificadores genéricos. Se han tomado las tres primeras imágenes para el entrenamiento y todas (incluidas las tres de entrenamiento) para el conjunto de test. El conjunto de imágenes resultado para todas las simulaciones realizadas se recoge en el Anexo I. En las siguientes figuras se muestran algunos ejemplos de las imágenes de test no utilizadas para el entrenamiento.

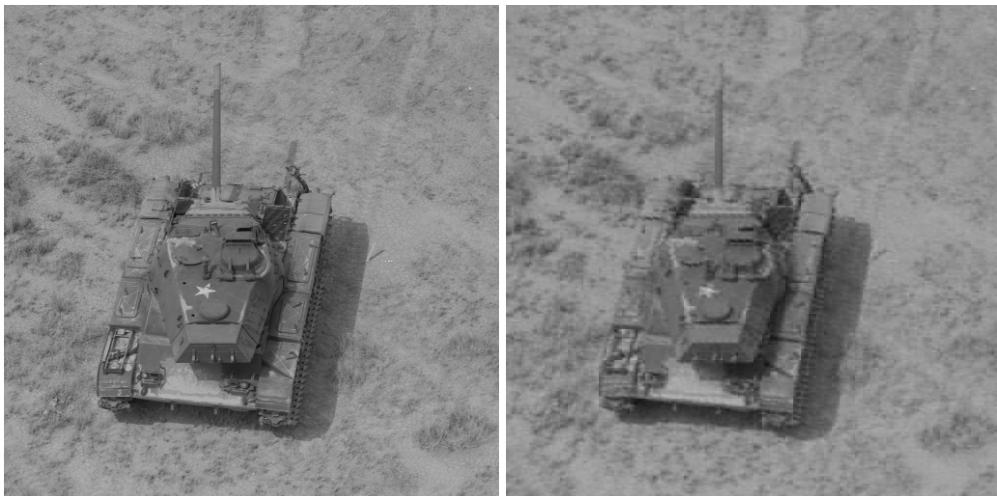


Figura 53. Imagen original (izquierda) y resultado (derecha) de la simulación de las capas de fuera.

En la figura anterior se muestra el resultado del entrenamiento de las capas de fuera. Se han utilizado imágenes de 512x512 píxeles y se ha simulado una red con 8 neuronas con función “tansig” en la capa oculta y 64 neuronas con función “purelin” en la capa de salida. Como entradas y salidas deseadas se han introducido los bloques de 8x8 píxeles de las imágenes y luego se han reconstruido las imágenes con los bloques obtenidos como resultado. Este paso supone una compresión de 1/8. Tras 1000 iteraciones del algoritmo de aprendizaje la red tiene un *performance* de 66.8 (medida basada en el error cuadrático medio que proporciona la herramienta nntraintool de Matlab para el entrenamiento de redes neuronales). Nota: tanto el *performance* como la calidad subjetiva de los resultados pueden variar de una simulación a otra. Los datos aportados son de una de las simulaciones realizadas considerada más representativa.

En la figura 54 se muestra la simulación del compresor aplicado a imágenes de 512x512 píxeles con una red interior de 32 neuronas en la capa oculta con función “tansig” y 8x16 neuronas en capa de salida con función “purelin”; y una red exterior como la anterior (8 neuronas en la capa oculta y 64 en la de salida). Se obtiene una compresión de $(1/8) * (1/4) = 1/32$.

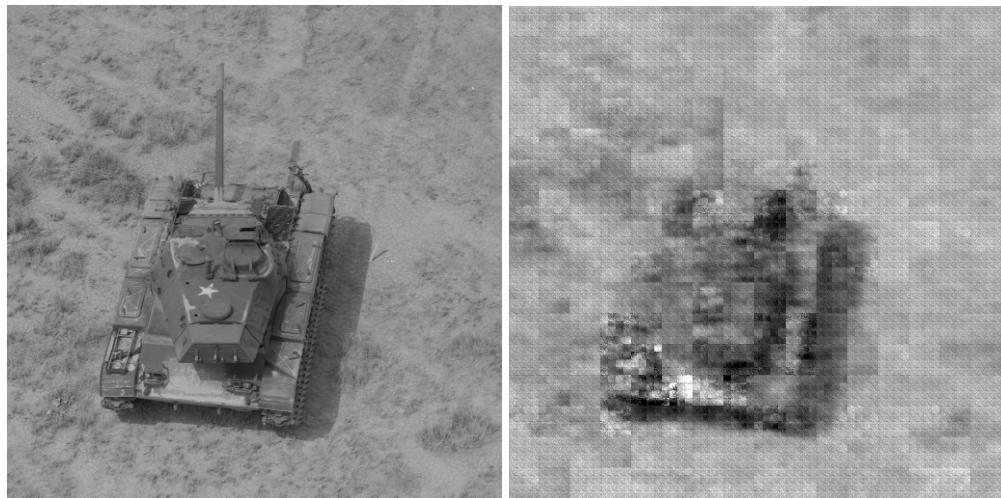


Figura 54. Imagen original (izquierda) y resultado (derecha) de la simulación del compresor completo con 32 neuronas en la capa oculta de la red interior.

En la siguiente figura se muestra la simulación del compresor aplicado a imágenes de 512x512 con una red interior de 16 neuronas en la capa oculta y una red exterior como las anteriores. Se obtiene una compresión de $(1/8) * (1/8) = 1/64$.

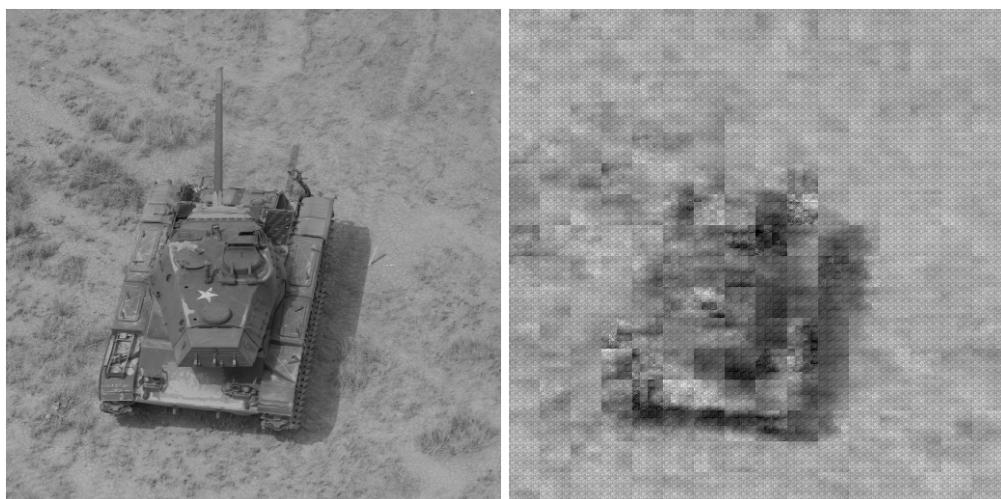


Figura 55. Imagen original (izquierda) y resultado (derecha) de la simulación del compresor completo con 16 neuronas en la capa oculta de la red interior.

También se han realizado simulaciones con imágenes de 256x256 píxeles obteniendo unos resultados de peor calidad con el compresor completo.



Figura 56. Simulación con imagen de 256x256. Original (izquierda), comprimida con la capa de fuera de 8 neuronas de capa oculta (centro) y comprimida con capa de fuera anterior y exterior de 32 neuronas de capa oculta (derecha).

II. Conclusiones

De los estudios matemáticos sobre la DLMT y las simulaciones en Matlab realizadas se ha concluido que la formulación actual de esta transformación no es adecuada para la compresión de imágenes en hardware. El elevado número de condición de las matrices obtenidas provoca problemas de calidad de imagen con las limitaciones de precisión propias del hardware. Acentuados al tratarse de un sistema de compresión de bajo consumo donde precisamente se busca la mejor calidad alcanzable para la menor precisión posible.

Sin embargo se ha logrado desarrollar un compresor eficaz y de bajo consumo empleando la DCT con las modificaciones propuestas. Este compresor diseñado para la plataforma Cookies con una capa de procesamiento HiReCookie permitirá el desarrollo de aplicaciones como las descritas en el apartado II.WVSN del capítulo de Introducción alargando la vida útil de los nodos con cámara. La implementación del compresor ocupa pocos recursos, lo que permite su instanciación en FPGAs de bajo coste y además incluir otros elementos de procesamiento y control en la misma FPGA.

De las medidas realizadas se deduce que los picos de consumo de procesamiento son pequeños en comparación con el consumo estático durante los tiempos en los que no se está procesando. Habría que intentar reducir este consumo estático revisando las máquinas de estado de control y las señales de habilitación (*enables*).

De las simulaciones sobre las redes neuronales propuestas obtenidas hasta el momento aún no se puede decidir si el proyecto puede ser viable o no. Aunque los resultados para la primera etapa de compresión (las redes exteriores) son excelentes los resultados actuales del compresor completo no son buenos. Se observa que para las arquitecturas simuladas la calidad del resultado empeora al reducir el tamaño de la imagen, por lo que habrá que simular arquitecturas con diferentes tamaños de bloques. Hace falta continuar con esta línea de investigación para mejorar la compresión con las redes interiores. Para ello hay que ajustar parámetros de las simulaciones

tales como algoritmos de aprendizaje, funciones de salida, arquitectura, etc... que permitan mejorar los resultados hasta ahora obtenidos. También hay que tener en cuenta que en la implementación en hardware habrá limitación de precisión en los datos, pesos, resultados y sobre todo en las funciones de salida. Habrá que adaptar estas funciones para que puedan ser implementadas en hardware sin comprometer el consumo del nodo para que esta propuesta sea viable.

III. Líneas futuras

Como línea futura inmediata queda terminar de ajustar y optimizar el compresor JPEG. Con las medidas realizadas se han de revisar los diseños para intentar reducir el consumo en tiempos de no procesamiento reduciendo el consumo del control y de memorias y añadiendo acciones de apagado de la FPGA que permite la HiReCookie y que aún no han sido incorporadas en esta primera versión del compresor.

Siguiendo la línea del compresor tipo JPEG surgen dos líneas futuras:

- Realizar un compresor de vídeo. Dar el salto de la compresión de imágenes aisladas a la compresión de una secuencia de imágenes eliminando la redundancia temporal. Existen varias formas de hacerlo:
 - Mediante técnicas de predicción y compensación de movimiento.
 - Codificación incremental aplicada y adaptada a las imágenes.
 - Aplicando transformaciones tridimensionales.
 - Codificadores Wyner-Ziv.
- Ajustar las ecuaciones de la DLMT para intentar lograr una transformación cuya matriz de coeficientes tenga un menor número de condición e incluso hacerla ortogonal si fuera posible. En caso de lograr una transformación cuyas características permitiesen su implementación en hardware el siguiente paso sería realizar un estudio sobre la sensibilidad del ojo humano a sus componentes, determinar qué tipos de tablas de cuantificación serían eficaces en la compresión mediante esta transformada y comparar los resultados con otros compresores (como los propuestos en este proyecto).

Siguiendo la línea de las redes neuronales distribuidas en la red de sensores inalámbricos, como se ha indicado en el anterior apartado de Conclusiones, se han de continuar los ajustes para mejorar los resultados de las redes interiores. Y posteriormente si se concluye que el proyecto es viable diseñar, optimizar y construir la red en hardware. Está planificado continuar con este trabajo como tesis doctoral.

REFERENCIAS

REFERENCIAS

- [1] Charfi, Y.; Wakamiya, N.; Murata, M.; , "Challenging issues in visual sensor networks," *Wireless Communications, IEEE* , vol.16, no.2, pp.44-49, April 2009
- [2] Xiaoxia Ren; Zhigang Yang; , "Research on the key issue in video sensor network," *Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on* , vol.7, no., pp.423-426, 9-11 July 2010
- [3] Winkler, T.; Rinner, B.; , "Power aware communication in wireless pervasive smart camera networks," *Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2009 5th International Conference on* , vol., no., pp.283-288, 7-10 Dec. 2009
- [4] C. D.-F. Vincent Lecuire and N. Krommenacker; , "Energy-Efficient Transmission of Wavelet-based Images in Wireless Sensor Networks", *EURASIP J. Image Video Process.*, 2007
- [5] H. Wu and; A. Abouzeid; , "Error Resilient Image Transport in Wireless Sensor Networks", *Comp. Net.*, vol. 50, Oct. 2006, pp. 2873-87
- [6] Harjito, B.; Song Han; , "Wireless Multimedia Sensor Networks Applications and Security Challenges," *Broadband, Wireless Computing, Communication and Applications (BWCCA), 2010 International Conference on* , vol., no., pp.842-846, 4-6 Nov. 2010
- [7] Ahmad, J.J.; Khan, H.A.; Khayam, S.A.; , "Energy efficient video compression for wireless sensor networks," *Information Sciences and Systems, 2009. CISS 2009. 43rd Annual Conference on* , vol., no., pp.629-634, 18-20 March 2009
- [8] Abid, H.; Qaisar, S.; , "Distributed video coding for wireless visual sensor networks using low power Huffman coding," *Information Sciences and Systems (CISS), 2010 44th Annual Conference on* , vol., no., pp.1-6, 17-19 March 2010
- [9] Soro, S.; Heinzelman, W.B.; , "On the coverage problem in video-based wireless sensor networks," *Broadband Networks, 2005. BroadNets 2005. 2nd International Conference on* , vol., no., pp.932-939 Vol. 2, 7-7 Oct. 2005
- [10] Karakaya, M.; Hairong Qi; , "Target detection and counting using a progressive certainty map in distributed visual sensor networks," *Distributed Smart Cameras, 2009. ICDS-C 2009. Third ACM/IEEE International Conference on* , vol., no., pp.1-8, Aug. 30 2009-Sept. 2 2009

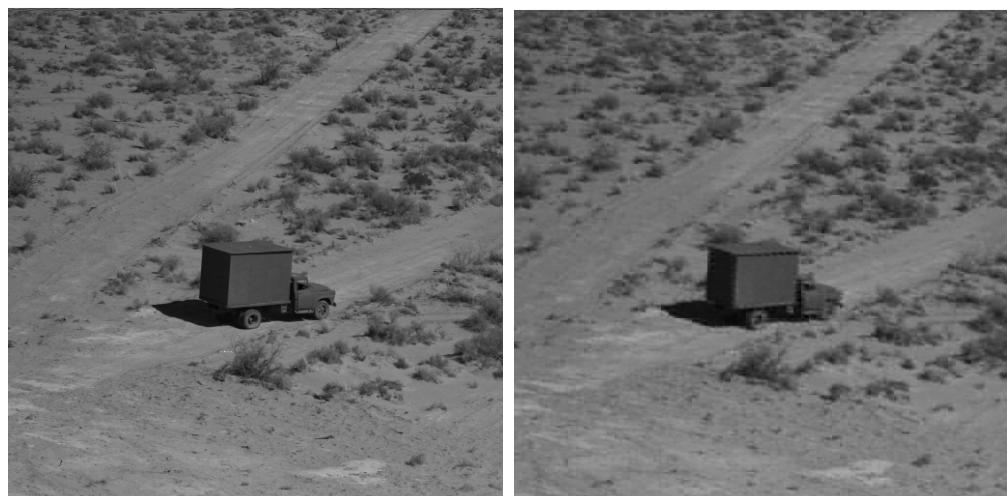
- [11] Soro, S.; Heinzelman, W.; , "Camera selection in visual sensor networks," *Advanced Video and Signal Based Surveillance, 2007. AVSS 2007. IEEE Conference on*, vol., no., pp.81-86, 5-7 Sept. 2007
- [12] Martucci, S.A.; , "Symmetric convolution and the discrete sine and cosine transforms ,"*Signal Processing, IEEE Transactions on* , vol.42, no.5, pp.1038-1051, May 1994
- [13] Moreno, F.; Aledo, D.; , "The DLMT. An alternative to the DCT," *IECON 2011 - 37th Annual Conference on IEEE Industrial Electronics Society*, vol., no., pp.2267-2272, 7-10 Nov. 2011
- [14] Huffman, D.A.; , "A Method for the Construction of Minimum-Redundancy Codes," *Proceedings of the IRE* , vol.40, no.9, pp.1098-1101, Sept. 1952
- [15] Dony, R.D.; Haykin, S.; , "Neural network approaches to image compression," *Proceedings of the IEEE* , vol.83, no.2, pp.288-303, Feb 1995
- [16] J. Valverde, A. Otero, M. Lopez, J. Portilla, T. Riesgo, "Using SRAM Based FPGAs for Power-Aware High Performance Wireless Sensor Networks", *in Sensors 2012*, 12, 2667-2692
- [17] J. Portilla, A. de Castro, E. de la Torre, T. Riesgo, "A Modular Architecture for Nodes in Wireless Sensor Networks", *Journal of Universal Computer Science (JUCS)*, vol. 12, nº 3, pp. 328 – 339, March 2006
- [18] David Aledo; "Implementación hardware de las transformadas discretas DCT y DLMT para compresión de imágenes y su estudio comparativo", *Proyecto fin de carrera*, Septiembre 2011
- [19] <http://www.xilinx.com/>
- [20] Namphol, A.; Arozullah, M.; Chin, S.; , "Higher order data compression with neural networks," *Neural Networks, 1991., IJCNN-91-Seattle International Joint Conference on* , vol.i, no., pp.55-59 vol.1, 8-14 Jul 1991

ANEXO I:

Simulaciones de las redes neuronales

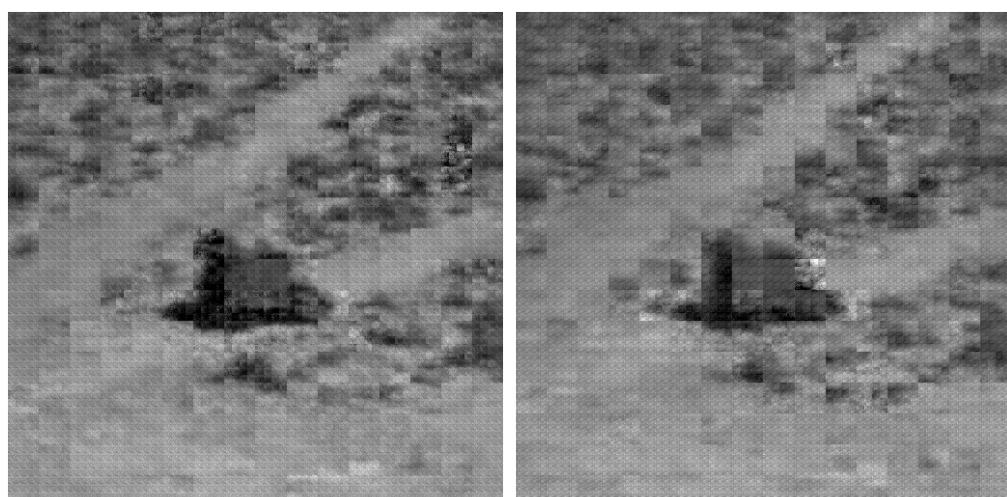
ANEXO I: Simulaciones de las redes neuronales

Imagen 1 de entrenamiento de 512x512 píxeles



Original

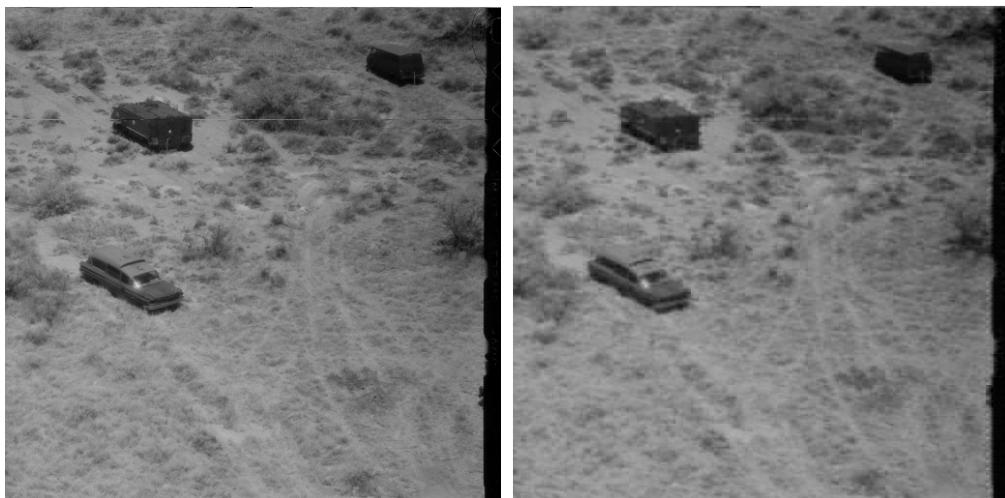
Comprimida con red exterior de 8 neuronas



Comprimida con compresor completo de 32 neuronas en capa oculta de red interior

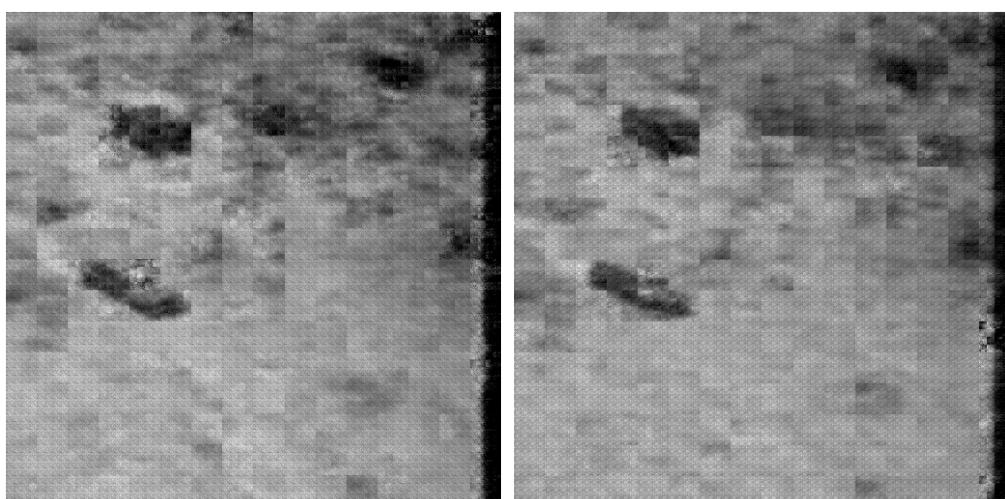
Comprimida con compresor completo de 16 neuronas en capa oculta de red interior

Imagen 2 de entrenamiento de 512x512 píxeles



Original

Comprimida con red exterior de 8 neuronas



Comprimida con compresor completo de 32 neuronas en capa oculta de red interior

Comprimida con compresor completo de 16 neuronas en capa oculta de red interior

Imagen 3 de entrenamiento de 512x512 píxeles



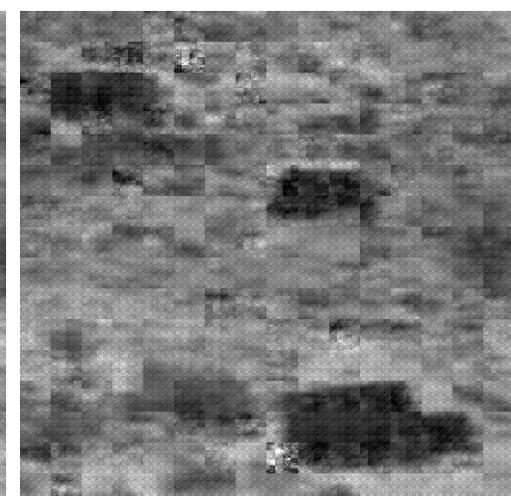
Original



Comprimida con red exterior de 8 neuronas



Comprimida con compresor completo de 32
neuronas en capa oculta de red interior



Comprimida con compresor completo de 16
neuronas en capa oculta de red interior

Imagen 4 de test de 512x512 píxeles



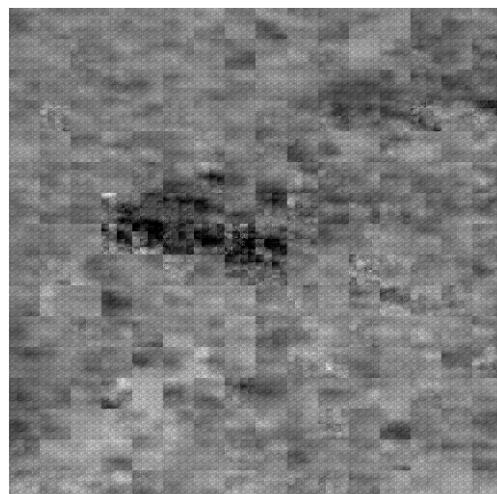
Original



Comprimida con red exterior de 8 neuronas

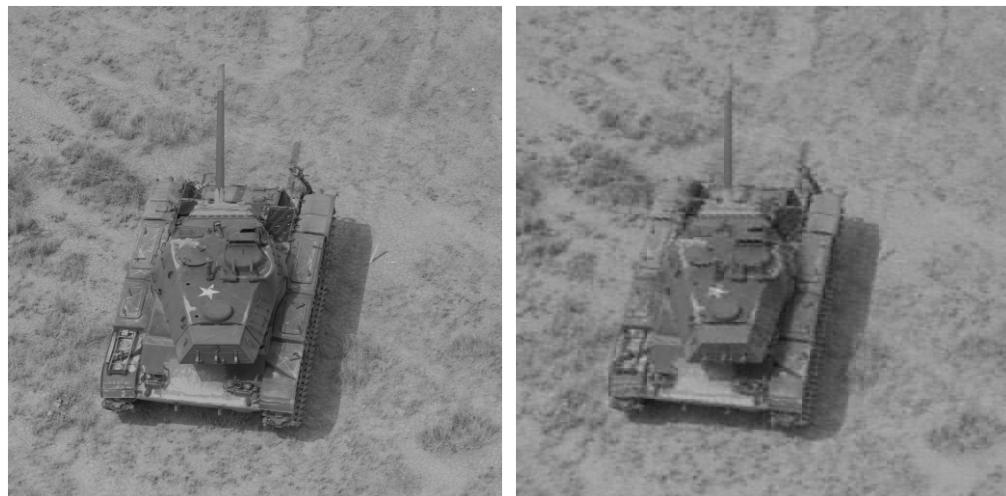


Comprimida con compresor completo de 32
neuronas en capa oculta de red interior



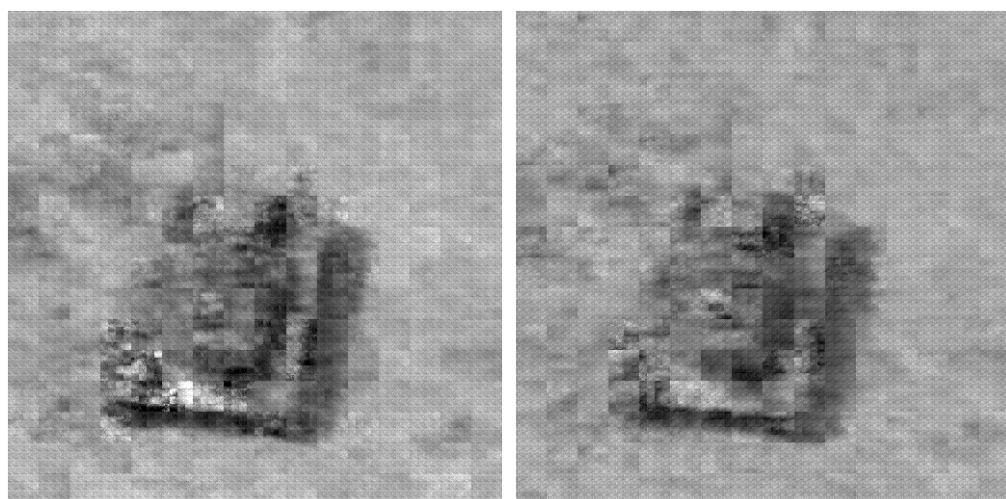
Comprimida con compresor completo de 16
neuronas en capa oculta de red interior

Imagen 5 de test de 512x512 píxeles



Original

Comprimida con red exterior de 8 neuronas



Comprimida con compresor completo de 32
neuronas en capa oculta de red interior

Comprimida con compresor completo de 16
neuronas en capa oculta de red interior

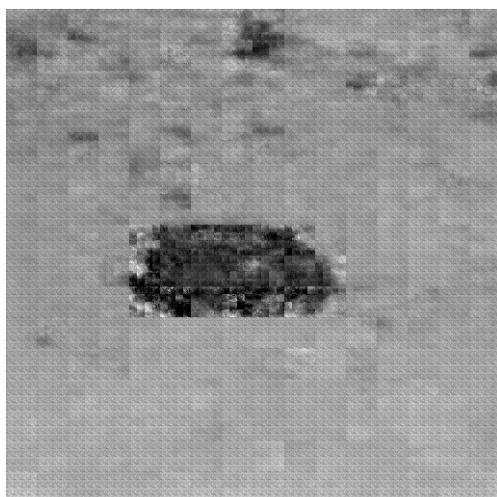
Imagen 6 de test de 512x512 píxeles



Original



Comprimida con red exterior de 8 neuronas



Comprimida con compresor completo de 32
neuronas en capa oculta de red interior



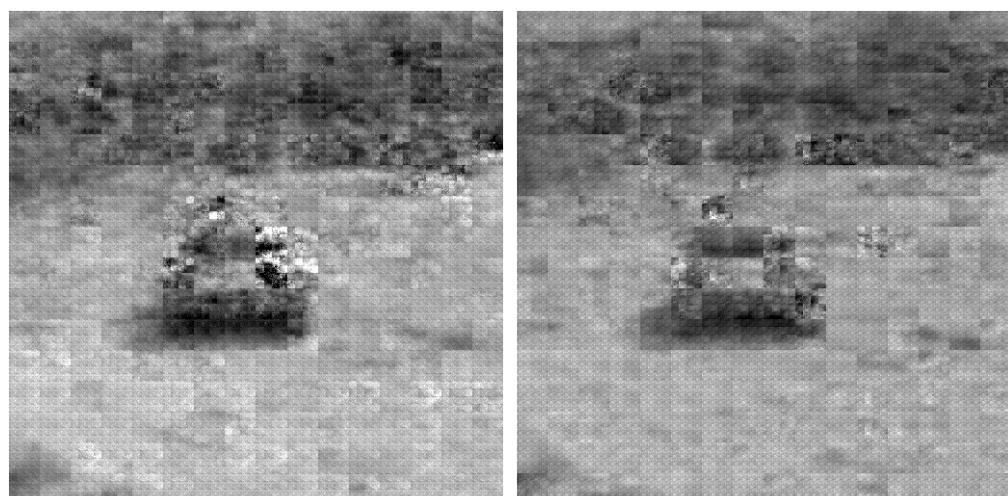
Comprimida con compresor completo de 16
neuronas en capa oculta de red interior

Imagen 7 de test de 512x512 píxeles



Original

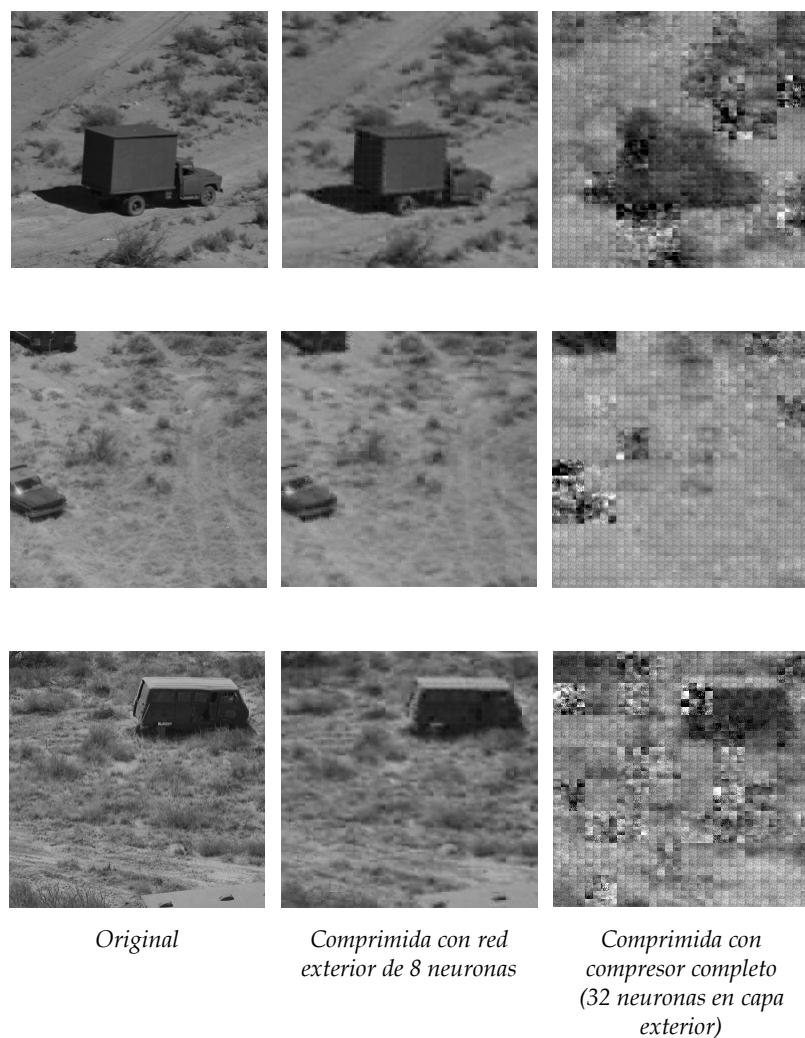
Comprimida con red exterior de 8 neuronas



Comprimida con compresor completo de 32
neuronas en capa oculta de red interior

Comprimida con compresor completo de 16
neuronas en capa oculta de red interior

Imágenes de entrenamiento de 256x256 píxeles



Imágenes de test de 256x256 píxeles

