Diseño y experimentación de un cuantizador vectorial hardware basado en redes neuronales para un sistema de codificación de video.

1 author:

Agustín Ramírez-Agundis
Instituto Tecnológico de Celaya
12 PUBLICATIONS **118** CITATIONS

Departamento de Ingeniería Electrónica

Diseño y experimentación de un cuantizador vectorial hardware basado en redes neuronales para un sistema de codificación de video

Tesis Doctoral

Presentada por: Agustín Ramírez Agundis

Dirigida por: Dr. D. Rafael Gadea Gironés

Dr. D. Ricardo Colom Palero

Valencia, Octubre de 2008

Dedicatoria

El camino resultó más complejo y prolongado de lo esperado, pero fue también el motivo de nuevas vivencias y conocimientos. En el trayecto se incorporaron a mi vida otros seres que se han convertido en fuente inagotable de alegrías. Al final del viaje mi Madre ya no está físicamente con nosotros.

A la memoria de Doña Lupita y a mis nuevos retoños están dedicados los frutos de esta travesía.

Con mucho cariño y de manera especial se los dedico a mis hijos como motivación para continuar avanzando.

Con el entrañable afecto de siempre a mis seis hermanos y sus familias.

A mis amigos.

Agradecimientos

Le expreso mi gratitud a Rafa Gadea por brindarme su guía, su acompañamiento y sus palabras de aliento en los momentos críticos. A Ricardo Colom le agradezco su manifiesta y permanente disposición para apoyarme sin reserva alguna. En la actitud de mis directores de Tesis y en la de otros profesores de su Departamento de Ingeniería Electrónica he visto reflejado el espíritu de fraternidad con otros pueblos de la Universidad Politécnica de Valencia.

Muchas gracias también a todos mis compañeros del Departamento de Ingeniería Electrónica del Instituto Tecnológico de Celaya, por su respaldo, por su confianza y por permitirme compartir la esperanza y el ánimo de ser mejores. En lo particular le agradezco a Justo por su preocupación para superar los obstáculos administrativos y por su impulso. A Ramiro Rico por rescatar una causa que se encontraba casi perdida.

Un agradecimiento especial va para todos los mexicanos. Ellos aportaron los recursos que permitieron financiar mi participación en el programa de doctorado cuya culminación se materializa en esta Tesis.

Resumen - Resum - Abstract

Resumen

El objetivo general de esta Tesis es el estudio de las redes neuronales artificiales (ANN) con implementación hardware enfocadas hacia la compresión de imágenes y video en tiempo real. Como objetivos específicos, la Tesis se propone: explorar la factibilidad de utilizar las redes neuronales en las diferentes etapas de un sistema de compresión de imágenes; evaluar las redes *Self Organizing Feature Map* (SOM) en su implementación hardware utilizadas para la cuantización vectorial de imágenes; analizar la capacidad para procesar video en tiempo real de un sistema de compresión de imágenes que combine la cuantización vectorial basada en redes neuronales con otras técnicas; y estructurar un sistema para realizar el entrenamiento de redes neuronales utilizando esquemas de co-diseño hardware-software.

La Tesis expone primeramente los conceptos fundamentales relacionados con la compresión de imágenes considerando tanto los principios teóricos subyacentes como las técnicas que se usan para llevar a cabo las tareas involucradas con las diferentes etapas que integran un compresor. A continuación efectúa una revisión de los trabajos de investigación en los que las ANN se utilizan para la compresión de imágenes, tarea que es precedida por un breve repaso del desarrollo que se ha observado en el campo de las ANN.

En la parte práctica la Tesis tiene dos apartados. En el primero se desarrolla una red neuronal tipo SOM que se utiliza como cuantizador vectorial para la aplicación de que se ocupa. Partiendo del análisis de seis arquitecturas susceptibles de ser usadas, la red SOM se diseña utilizando una arquitectura masivamente paralela tipo SIMD y se implementa en hardware sobre una FPGA. Finalmente se experimenta con la red y se presentan los resultados.

En el segundo apartado se estructura el banco de entrenamiento para la red SOM utilizando una metodología de codiseño hardware-software en la cual la red neuronal SOM se integra al banco como núcleo de un neurocoprocesador en una placa de aplicación de FPGAs. Este apartado concluye con dos breves aplicaciones de investigación mediante las cuales se ilustra la utilidad del banco y una aplicación mayor en la que se diseña un sistema de compresión que integra la transformada wavelet y la cuantización vectorial basada en la red SOM.

Resum

L'objectiu general d'esta Tesi és l'estudi de les xarxes neuronals artificials (ANN) amb implementació *hardware* enfocades cap a la compressió d'imatges i vídeo en temps real. Com a objectius específics la Tesi es planteja: explorar la factibilitat d'utilitzar les xarxes neuronals en les diferents etapes d'un sistema de compressió d'imatges; avaluar les xarxes *Self Organizing Feature Map* (SOM) en la seua implementació *hardware* utilitzades per a la cuantització vectorial d'imatges; analitzar la capacitat per a processar vídeo en temps real d'un sistema de compressió d'imatges que combine la cuantització vectorial basada en xarxes neuronals amb altres tècniques; i estructurar un sistema per a realitzar l'entrenament de xarxes neuronals utilitzant tècniques de co-disseny *hardware-software*.

La Tesi exposa primerament els conceptes fonamentals relacionats amb la compressió d'imatges considerant tant els principis teòrics subjacents com les tècniques que s'usen per a dur a terme les tasques involucrades amb les diferents etapes que integren un compressor. A continuació efectua una revisió dels treballs d'investigació en què les ANN s'utilitzen per a la compressió d'imatges, tasca que és precedida per un breu repàs del desenrotllament que s'ha observat en el camp de les ANN.

En la part pràctica la Tesi té dos apartats. En el primer es desenrotlla una xarxa neuronal tipus SOM que s'utilitza com *cuantizador* vectorial per a l'aplicació que s'ocupa. Partint de l'anàlisi de sis arquitectures susceptibles de ser usades, la xarxa SOM es dissenya utilitzant una arquitectura massivament paral·lela tipus SIMD i s'implementa en *hardware* sobre una FPGA. Finalment s'experimenta amb la xarxa i es presenten els resultats.

En el segon apartat s'estructura el banc d'entrenament per a la xarxa SOM utilitzant una metodologia de co-disseny *hardware-software* en la qual la xarxa neuronal SOM s'integra al banc com a nucli d'un *neurocoprocesador* en una placa d'aplicació de FPGAs. Este apartat conclou amb dos breus aplicacions d'investigació per mitjà de les quals s'il·lustra la utilitat del banc i una aplicació major en què es dissenya un sistema de compressió que integra la transformada *wavelet* i la cuantització vectorial basada en la xarxa SOM.

Abstract

The general objective of this Thesis is the study of artificial neural networks (ANN) with hardware implementation focused towards real-time image and video compression. As specific objectives the Thesis pursuits: to explore the feasibility of using ANNs at the different stages of an image compression system; to evaluate the Self Organizing Feature Map (SOM) neural networks with hardware implementation when they are used for image vector quantization; to analyze the aptitude to real time video processing of an image compression system that mixes ANN based vector quantization with other techniques; and to construct a system to realize the ANN training using hardware-software co-design approach.

The Thesis firstly exposes the image compression fundamentals taking into account both, the theoretical principles and the techniques used to carry out the tasks involved in the compressor stages. After that, there is a survey for the research works where the artificial neural networks are used to image compression. This task is preceded by a briefly review of the development in the ANN's field.

Regarding its practical section, the Thesis has two parts. At the first one, a SOM neural network is developed to be used as a vector quantizer for image compression. As a departure point, there is an analysis of six architectures that could be used, then the SOM net is designed using the parallel massively SIMD architecture and it is implemented over an FPGA device. Finally the experimentation work is carried out and the results are presented.

In the second one, the SOM net training bench is structured using a hardware-software codesign approach. The SOM neural network is integrated in the bench as a core of a neuro-coprocessor in an FPGA application board. This part concludes with two small research applications to illustrate the bench usability and a major application where a complete image compression system is designed with a mixed wavelet transform-vector quantization scheme.

Indice

Indice de tablas

Indice de figuras

Lista de acrónimos y símbolos

2D-DWT	*Bi-Dimensional Discrete Wavelet Transformer*
ADPCM	*Adaptable Differential Pulse Code Modulation*
ANN	*Artificial Neural Network*
ART	*Adaptive-Resonance Theory*
ASIC	*Application Specific Integrated Circuit*
BLRNN	*Bi-Linear Recurrent Neural Network*
BNN	*Biological Neural Network*
BTSVQ	*Binary Tree Search Vector Quantization*
CMZ	*Cottrell-Munro-Zipser*
CPLD	*Complex Programmable Logic Device*
CVQ	*Classified Vector Quantization*
CWT	*Continuous Wavelet Transform*
DCT	*Discrete Cosine Transform*
DFT	*Discrete Fourier Transform*
DPCM	*Differential Pulse Code Modulation*
DSP	*Digital Signal Processing*
DWT	*Discrete Wavelet Transformer*
EP	*Elemento de Procesamiento*
FPGA	*Field Programmable Gate Array*
FT	*Fourier Transform*
GLA	*Generalized Lloyd Algorithm*
HDL	*Hardware Description Language*
HT	*Haar Transform*
HVQ	*Hierarchical Vector Quantization*
HVS	*Human Visual System*
HW	*Hardware*
HWT	*Hadamard-Walsh Transform*
IDFT	*Inverse Discrete Fourier Transform*

JPEG	*Joint Photographic Experts Group*
JTAG	*Joint Test Action Group (estándar IEEE 1149.1)*
KLT	*Karhunen-Loeve Transform*
LBF	*Lineal Base Function*
LBG	*Lindel-Buzo-Gray*
LUT	*Look-Up Table*
LVQ	*Learning Vector Quantization*
LZW	*Lempel ZivWelch*
MCPS	*Millions-of Connections Per Second*
MCUPS	*Millions-of Connections Updated Per Second*
MLP	*Multi-Layer Perceptron*
MSE	*Mean Square Error*
MSVQ	*Multi-Stage Vector Quantization*
NMSE	*Normalized Mean Square Error*
NN	*Neural Network*
PCA	*Principal Component Analysis*
PE	*Processing Element*
PET	*Positron Emission Tomography*
PSNR	*Peak Signal-to-Noise Ratio*
PVQ	*Predictive Vector Quantization*
RBF	*Radial Base Function*
RBG	*Red Green Blue*
RTL	*Register Transfer Level*
SIMD	*Single-Instruction Multiple-Data*
SoC	*System on Chip*
SOFM	*Self-Organizing Feature Map*
SOM	*Self-Organizing Feature Map*
SW	*Software*
TDNN	*Time Delayed Neural Network*
TSVQ	*Tree Search Vector Quantization*
ULSI	*Ultra Large Scale Integration*
UP	*Unidad de Procesamiento*
VHDL	*Very High-speed Hardware Description Language*
VLSI	*Very Large Scale Integration*
VQ	*Vector Quantization*
WPBS	*Word-Parallel Bit-Serial*
WTA	*Winner Take All*

Capítulo 1

Introducción

1.1 Antecedentes

Subyacentes a esta tesis se encuentran tres circunstancias como sus principales motivaciones, siendo éstas: (i) la importancia que ha adquirido la compresión de imágenes y de video en las últimas dos décadas hasta llegar a la situación actual en la que su aplicación está presente en casi todos los ámbitos de la vida cotidiana, (ii) el gran desarrollo tecnológico que se ha observado en los dispositivos lógicos programables, particularmente las FPGAs, y (iii) la potencialidad que ofrecen las redes neuronales artificiales como resultado de una mayor comprensión de la manera como funcionan las biológicas y la generación de modelos, arquitecturas y algoritmos con el propósito de emularlas.

La implementación hardware es un aspecto determinante en la aplicación objeto de esta tesis. Esto por dos razones, la primera reside en el hecho de que las redes neuronales de tamaño considerable en general, y de forma particular las competitivas, necesitan tiempos prolongados para su entrenamiento. La segunda consiste en que las redes neuronales de manera inherente tienen un procesamiento masivamente paralelo, característica que no es posible explotar en una implementación software, que es secuencial por naturaleza. Así que acelerar el proceso de aprendizaje y aprovechar el paralelismo inherente de las redes neuronales es una doble motivación para sus implementaciones hardware.

En cuanto al tema de la compresión de imágenes y video, se observa la conveniencia de estudiar la aplicación de técnicas que en el papel presentan características favorables para esta aplicación, específicamente la cuantización vectorial y la transformada wavelet, para reemplazar a otras de desarrollo anterior como son la cuantización escalar y la transformada discreta del coseno. Para esto se hace necesario contar con herramientas que faciliten su integración y una experimentación fluida.

Por otra parte, el hecho de contar con dispositivos lógicos programables de gran tamaño y la incorporación en ellos de poderosos bloques funcionales para el procesamiento digital, como es el caso de las nuevas familias de FPGAs, han dado lugar a nuevos paradigmas para el diseño lógico, destacando el procesamiento hardware reconfigurable. Las FPGAs han dejado de ser únicamente dispositivos para el prototipado de diseños que finalmente se producen en circuitos integrados de aplicación específica (ASICs) y han pasado a ser verdaderas plataformas de procesamiento digital de señales.

1.2 Objetivo

El objetivo general de esta Tesis es el estudio de las redes neuronales con implementación hardware enfocadas hacia la compresión de imágenes y video en tiempo real, con los siguientes objetivos específicos:

1. Explorar la factibilidad de utilizar las redes neuronales en las diferentes etapas de un sistema de compresión de imágenes con posibilidades de competir con otras técnicas en cuanto a velocidad, tasa de compresión y grado de distorsión.
2. Evaluar las redes SOM en su implementación hardware utilizadas para la cuantización vectorial de imágenes.
3. Analizar la capacidad para procesar video en tiempo real de un sistema de compresión de imágenes que combine el cuantizador vectorial basado en redes neuronales con otras técnicas complementarias.
4. Estructurar un sistema para realizar el entrenamiento de redes neuronales utilizando técnicas de co-diseño hardware-software.

1.3 Metodología

Para alcanzar estos objetivos el trabajo de investigación se enfoca hacia las siguientes tareas:

- Realizar una exhaustiva revisión de las aplicaciones de las redes neuronales para la compresión de imágenes, considerando aquéllas que se orientan hacia algún tipo de transformación sobre la imagen, las que realizan la cuantización vectorial y las relacionadas con métodos predictivos. La revisión se centrará en tres aspectos: las tasas de compresión y niveles de distorsión alcanzados, su simplicidad conceptual y la factibilidad para su implementación hardware.
- Diseñar una red neuronal hardware con características apropiadas para la experimentación y su evaluación como son la posibilidad de operación tanto en la fase de entrenamiento como en la de recall, así como flexibilidad para elegir el tamaño de la red en cuanto a número de neuronas, el tipo y tamaño de los datos de entrada y la precisión utilizada para los pesos sinápticos y establecer los parámetros de aprendizaje.
- Evaluar el comportamiento de la red como cuantizador vectorial. La evaluación, por lo que corresponde a la distorsión que introduce el cuantizador para una tasa de compresión dada, tomará como referencia el bien conocido y probado algoritmo para generación de diccionarios denominado LBG. Por lo que respecta al rendimiento en velocidad, se tomarán como referencia los resultados obtenidos por diseños afines.
- Integrar la red en un sistema de compresión de imágenes que involucre las etapas típicas de un compresor y comprobar su capacidad para procesar video en tiempo real, es decir, verificar que sea capaz de procesar más de 30 imágenes por segundo.
- Diseñar un banco de entrenamiento en el que se incorpore la red implementada sobre una FPGA. El banco tendrá como objetivo proporcionar la funcionalidad para una experimentación ágil y flexible.

1.4 Organización

Esta tesis consta de tres partes. La primera sirve como fundamentación teórica de los conceptos y técnicas relacionados con la compresión de imágenes y está plasmada en el Capítulo 2 y la primera sección del Capítulo 3. Por su importancia de acuerdo al enfoque de la tesis, se dedica una atención especial a la cuantización vectorial, a la transformada wavelet y a las redes neuronales.

La segunda parte es una revisión de trabajos de investigación dedicados a la aplicación de las redes neuronales para la compresión de imágenes con la finalidad de tener un panorama del estado del arte en este tema. A este apartado se dedica el Capítulo 3, que inicia con una breve exposición de los conceptos fundamentales de las redes neuronales.

El tercer apartado es básicamente el resultado de la parte práctica de la investigación. Está a su vez dividido en dos partes. La primera de ellas se expone en el Capítulo 4 y está dedicada al diseño de la red neuronal desarrollada. Por principio de cuentas, en este capítulo se revisan y comparan algunas arquitecturas factibles para la implementación hardware de la red SOM. Igualmente, se analizan algunas alternativas para el diseño de los bloques funcionales que conforman este tipo de red. Posteriormente se expone el diseño de la red y los resultados de la experimentación.

Continuando con la parte práctica de la investigación, el Capítulo 5 se enfoca, en primer término, al diseño físico de un banco de entrenamiento para la red desarrollada. El banco está basado en un codiseño hardware-software y se incluyen dos breves aplicaciones de investigación que muestran las bondades de contar con una herramienta como ésta. El Capítulo 5 concluye con el diseño de un sistema completo de compresión de imágenes que integra la transformada wavelet, la cuantización vectorial y una técnica de codificación estadística.

Finalmente, en el Capítulo 6 se hace una breve discusión acerca de las conclusiones generales del trabajo de investigación que dio origen a esta tesis. También se plantean algunas tareas que se desprenden de la tesis a manera de propuesta para trabajos futuros en esta línea de investigación.

Capítulo 2

Compresión de imágenes

En este capítulo se exponen los fundamentos teóricos para la compresión de imágenes[1], poniendo énfasis en cada una de las etapas de procesamiento que conforman un sistema de compresión. Este capítulo junto con el siguiente constituyen la base teórica general para la comprensión del resto de la tesis.

2.1 Representación digital de imágenes

Una imagen se define como la distribución espacial de la luz irradiada en un plano. Matemáticamente, una imagen es una función continua de dos variables, *L=l(x,y)*, tal que x e y son las coordenadas de un punto en el espacio bidimensional y la magnitud de L es la intensidad de la luz irradiada en ese punto.

Las imágenes son continuas por lo que se refiere a los valores de las coordenadas x e y, así como también por lo que respecta a la intensidad. Para representar una imagen de manera digital es necesario discretizar las coordenadas y también la intensidad. La discretización de las coordenadas se lleva a cabo mediante un procedimiento denominado *muestreo* y la de la intensidad por medio de otro al que se le llama *cuantización*. De este modo, una imagen digital es aquélla para la cual las coordenadas x e y, así como los valores de la intensidad, son cantidades finitas y discretas. A cada punto en el plano bidimensional con coordenadas discretizadas se le conoce como *pixel*.

El muestreo consiste en tomar la intensidad de la imagen considerándola con un valor constante en cada una de las regiones donde se intersectan las franjas de la retícula rectangular con espaciamiento uniforme definida así:

$$r_{i,j} = \{(i\Delta x_1, j\Delta x_2) \,|i, j \in \mathbf{Z};\ 0 \leq i \leq M-1,\ 0 \leq j \leq N-1\}\,, \tag{2.1}$$

donde M y N son, respectivamente, el número de franjas horizontales (filas) y el número de franjas verticales (columnas) de la retícula de muestreo. Para abreviar, en adelante se hablará de una imagen de M x N pixeles.

Por su parte, la cuantización Q tiene como propósito establecer una correspondencia entre los valores continuos de la intensidad de la imagen y un conjunto que contiene únicamente n valores discretos, matemáticamente se describe como sigue:

$$Q = [0, \infty] \xrightarrow{n} \{L_1, L_2, \cdots, L_n\}\,, \tag{2.2}$$

a los n valores discretos $L_1, L_2, \cdots L_n$ se les denomina *niveles de cuantización*. Comúnmente se utilizan niveles de cuantización homogéneamente espaciados, quedando entonces determinados así:

$$L_k = L_{k-1} + \Delta L, k \in \{2, 3, \cdots, n\}. \tag{2.3}$$

A menudo se utiliza el término *gray levels* (tonos de gris) para hacer referencia a imágenes monocromáticas. Las imágenes de color se representan como la combinación de varias imágenes individuales monocromáticas. Por ejemplo, en el sistema para imágenes de color denominado *RGB* una imagen consiste de tres imágenes individuales denominadas la componente R (rojo), la componente B (azul) y la componente G (verde). De este modo, las técnicas de procesamiento desarrolladas para imágenes monocromáticas pueden extender su aplicación a imágenes de color procesando cada una de las componentes de la imagen separadamente.

Los métodos y sistemas que se analizan y desarrollan en esta tesis están dedicados a imágenes monocromáticas digitales muestreadas en M x N pixeles con 256 niveles de cuantización homogéneamente espaciados, representados por los números enteros comprendidos entre 0 y 255. De esta manera, una imagen de M x N pixeles quedará representada como una matriz de M filas y N columnas cuyos elementos son valores enteros de 8 bits. A la representación digital de imágenes mediante una matriz de pixeles con valores cuantizados se le conoce como *representación espacial*. La Fig. 2.1 muestra una imagen digital de 6 x 8 pixeles con 8 niveles de cuantización.[1]

Además de su simplicidad, la representación espacial de una imagen permite aplicar directamente las propiedades y operaciones del álgebra matricial para su análisis y procesamiento, así como hacer uso de las herramientas computacionales que se han desarrollado extensamente en ese campo.

La representación espacial de una imagen de M x N pixeles también se puede expresar en el ámbito de espacios vectoriales en función de un conjunto de imágenes base cuya dimensión es M x N de la siguiente manera:

$$L = \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} L_{i,j} \; {}^{i,j}\mathbf{P}, \tag{2.4}$$

donde $L_{i,j}$ es la intensidad de imagen en el pixel i,j e ${}^{i,j}\mathbf{P}$ es el elemento i,j del conjunto de imágenes base $\mathbf{P}$. ${}^{i,j}\mathbf{P}$ tiene valor uno para el pixel situado en la fila i, columna j y tiene valor cero para todos los demás, expresándose entonces matemáticamente como lo establece la Ec. 2.5. No es difícil demostrar que el conjunto de imágenes base así definido, $\mathbf{P}$, es ortonormal.

$${}^{i,j}\mathbf{P} : \; p_{i',j'} = \begin{cases} 1 & \text{if } i = i' \wedge j = j' \\ 0 & \text{en caso contrario} \end{cases} \tag{2.5}$$

El empleo de conjuntos de imágenes base para la representación espacial de imágenes parece a primera vista un asunto trivial; sin embargo, es importante en tanto que es el fundamento para la aplicación de transformaciones que derivan en otros tipos de representaciones

[1] El nivel en cada pixel no representa a la imagen justamente en un punto, sino en una región rectangular. Ese nivel debe corresponder al valor promedio de la luz irradiada en esa región

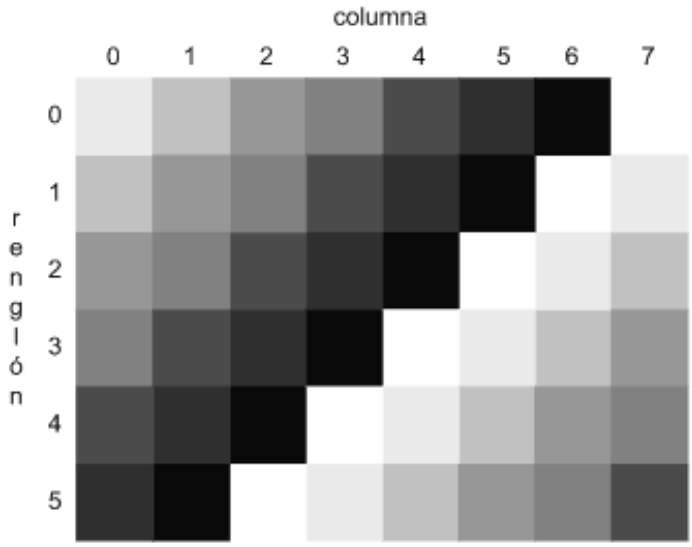


Fig. 2.1: Representación espacial de una imagen

más complejas pero que permiten realizar ciertos tratamientos de imagen de manera más eficiente. Más adelante, en este mismo capítulo, se dedicarán dos secciones a algunas de las transformaciones más empleadas para la compresión de imágenes.

2.2 Fundamentos de la compresión de imágenes

2.2.1 Teoría de la información

En este apartado se incluyen una serie de términos y métodos de medida empleados ampliamente a lo largo de la tesis. La mayor parte de ellos se derivan del trascendente trabajo de C. E. Shannon [2] que a casi 60 años de su publicación continúa apareciendo como el fundamento de campos de investigación hoy emergentes.

Sistemas de la información

Un sistema de comunicación, según Shannon, es un sistema del tipo representado esquemáticamente por la Fig. 2.2[2] y está formado por cinco partes: (a) una *fuente de información* que produce un mensaje o una secuencia de mensajes que deben ser comunicados; (b) un *transmisor* que opera de alguna manera sobre el mensaje para producir una señal apropiada para su transmisión sobre el canal; (c) un *canal*, es decir el medio usado para transmitir la señal desde el transmisor hasta el receptor; (d) un *receptor*, el cual realiza la operación inversa a la del transmisor, reconstruyendo el mensaje a partir de la señal; y (e) un *destino*, que es la persona o cosa a la cual va dirigido el mensaje.

En la teoría de la información, una fuente se define como el par ordenado $F = (S, P)$, donde $S = \{s_1, s_2, ..., s_n\}$ es un conjunto finito de mensajes o símbolos denominado alfabeto

[2]La compresión de la señal se origina por la estructura estadística del mensaje y por la naturaleza propia del destino final de la información[2]

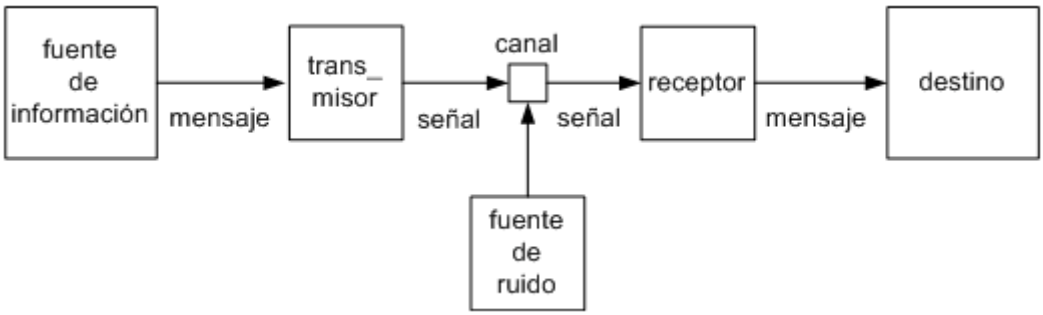


Fig. 2.2: Sistema de comunicación

y $P : S[0,1]$ es la distribución de probabilidades de S. La probabilidad de que ocurra un mensaje s_i se expresa como $P(s_i)$ o simplemente como p_i.

Entropía

De manera intituiva se observa que cuanto más alta sea la probabilidad de que ocurra un mensaje, más bajo será su contenido de información. En el caso extremo, si de antemano se conoce cual es el mensaje que se enviará, es decir que tiene probabilidad 1, entonces dicho mensaje no tiene información alguna. En otras palabras, lo que ya conocemos no nos proporciona ningún nuevo conocimiento, sólo aquello que ignoramos nos aporta información al conocerlo. Extendiendo la idea de información de un simple mensaje a una fuente, una fuente constituida por un gran número de mensajes con baja probabilidad tendrá más información que una fuente con pocos mensajes que presenten alta probabilidad.

Shannon definió una función para medir la cantidad de información que produce una fuente $F = (S, P)$; a esta función le denominó *entropía* de la fuente y está expresada así:

$$H(p_1, p_2, ..., p_n) = -\sum_{k=1}^{n} p_k \, log_b \, p_k, \tag{2.6}$$

la unidad de medida depende de la base logarítmica b empleada. Si se utiliza la base dos, la entropía se mide en *bits* (*binary units*); si se usa e como base, entonces la entropía se mide en *nats* (*natural units*); y si se emplea como base el diez, la entropía se mide en *hartleys*.

La entropía de una fuente significa que aunque no se pueda conocer que mensaje se va a producir, sí se sabe que en promedio se espera recibir H bits de información por mensaje. La entropía tiene varias propiedades importantes, una de ellas es que su valor es exactamente igual a la cantidad mínima de bits requeridos para codificar una fuente de modo que los mensajes se puedan reconstruir sin sufrir deterioro alguno. Es decir, si la entropía de una fuente es de H bits, entonces los mensajes no se pueden codificar con códigos que en promedio tengan menos de H bits.

Se ha demostrado que el valor de la entropía es máximo cuando todos los símbolos de la fuente tienen la misma probabilidad de ocurrir, es decir $p_1 = p_2 = ... = p_n = 1/n$. En tal caso $H = log_2 \, n$. Entonces la entropía está acotada entre 0 y $log_2 \, n$.

Una imagen o una secuencia de imágenes se puede considerar como una fuente de información. En el caso simple de una sola imagen de M x N pixeles en tonos de gris, el alfabeto

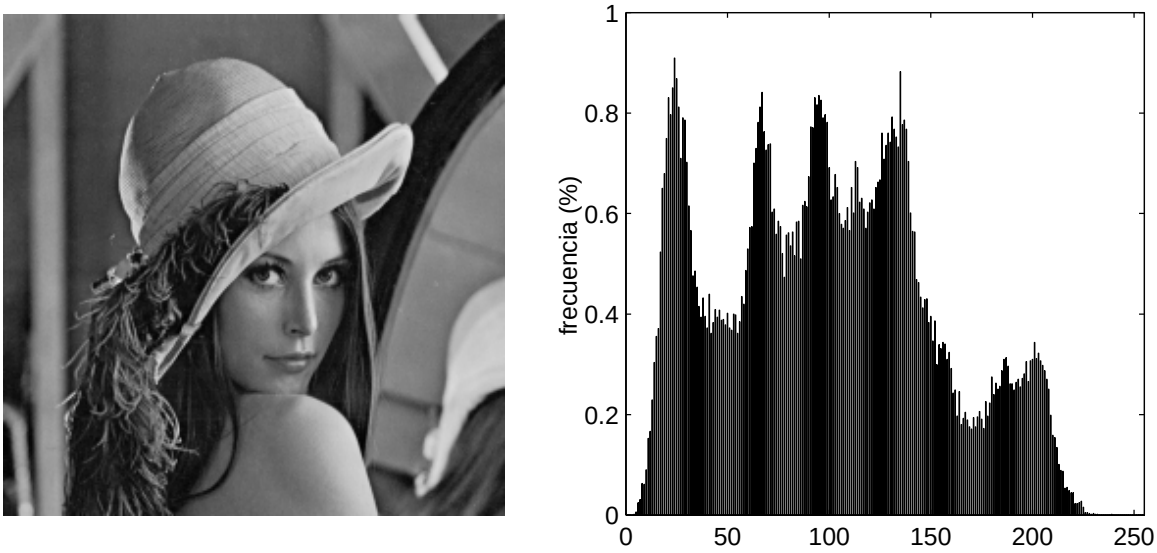


Fig. 2.3: Imagen Lena de 256 x 256 pixeles en tonos de gris y su histograma

S estará constituido por los 256 valores enteros comprendidos entre 0 y 255 y la distribución de probabilidades P será el conjunto de valores obtenidos al calcular la frecuencia con la que ocurre cada símbolo en la imagen dividida entre el producto de M x N. Por otra parte, la gráfica de la función de probabilidades así obtenida es el histograma de la imagen.

En la Fig. 2.3 se muestra la imagen Lena de 256 x 256 pixeles en tonos de gris, así como su histograma. La entropía de esta imagen considerándola como una fuente de información es de 7,5683 bits. Esto significa que su codificación no puede realizarse con menos de 7,5683 bits por código como promedio, cabe hacer notar que la entropía para cualquier imagen monocromática tiene un valor máximo igual a 8 (log_2 256).

Antes de concluir este apartado, conviene recordar que existen dos tipos de fuentes de información: *fuentes de memoria nula* y *fuentes de Markov*. En las fuentes de memoria nula cada mensaje o símbolo se trata estadísticamente de manera independiente, es decir, sin tomar en cuenta cualquier nexo que pudiera tener con los mensajes que le antecedieron o con los que le habrán de suceder. En las fuentes de Markov ocurre lo contrario, de modo que la probabilidad de aparición de un símbolo está condicionada por los últimos m símbolos que han aparecido, siendo m el orden de la fuente de Markov. Por ejemplo, tratando cualquier texto escrito en el idioma español como fuente de memoria nula, el símbolo k tiene una probabilidad muy baja de ocurrir, mientras que el símbolo r tiene una probabilidad alta; si el mismo texto se trata como una fuente de Markov de primer orden, el símbolo k continúa teniendo en todo caso una probabilidad muy baja, pero ahora la probabilidad del símbolo r dependerá del símbolo previo, siendo cero para cualquiera de los casos en que j, h, m, n, q, v, x, y o z sea el símbolo previo.

2.2.2 Origen de la compresión de imágenes

La compresión de imágenes digitales es posible porque éstas contienen elementos redundantes e irrelevantes. En otras palabras, el proceso de formación de la imagen que da lugar a su representación espacial inserta elementos que no son indispensables para su adecuada inter-

pretación por la máquina o ser humano a la que finalmente está destinada. La redundancia se ha dividido en tres tipos, que son: a) redundancia de código, b) redundancia entre pixeles, y c) irrelevancia psicovisual. La compresión de imágenes consiste en eliminar en la mayor medida posible las redundancias e irrelevancias de todo tipo.

Codificadores

La redundancia de código resulta al utilizar la misma cantidad de bits para representar cualquiera de los niveles de cuantización con los que se trabaja. Por ejemplo, para imágenes en tonos de gris se asignan uniformemente códigos de ocho bits a los 256 niveles, dejando de lado la estructura estadística de la fuente. Por lo tanto, la reducción de la redundancia de código se realiza partiendo del análisis estadístico de la fuente y empleando, por ejemplo, una codificación no uniforme.

Así, se han desarrollado diversos sistemas de codificación con códigos de longitud variable, en los cuales a los tonos de gris más frecuentes se les asigna un código corto y a los menos frecuentes un código largo, de manera que para una imagen completa el valor promedio de la cantidad de bits empleados para codificar cada pixel es menor que ocho. Huffman [3] desarrolló hace 55 años el método óptimo para reducir la redundancia de código. La eficiencia de un compresor para reducir la redundancia de código se mide tomando como referencia la entropía de la imagen. Cuanto más se acerque el valor promedio de bits por pixel del código a la entropía, más eficiente será el codificador. El valor promedio de bits por pixel para una imagen con dimensión M x N se calcula así:

$$\hat{b} = \frac{1}{MN} \sum_{k=1}^{256} b_k \, f_k, \tag{2.7}$$

siendo $\hat{b}$ el promedio de bits por pixel, b_k el número de bits usados para el código del símbolo s_k y f_k la frecuencia con la que ocurre ese símbolo en la imagen.

Otro método que ha mostrado ser muy eficiente es el que da lugar a la *codificación aritmética* [4] [5]. Al igual que la de Huffman, la codificación aritmética opera en base a la distribución de probabilidades de los símbolos del alfabeto. La diferencia consiste en que en lugar de asignar un código a cada símbolo, la codificación aritmética va asignando un número fraccionario a cada secuencia de símbolos de cierta longitud producida por la fuente. A partir de la distribución de probabilidades de la fuente, la codificación aritmética funciona así:

1. Dividir el intervalo comprendido entre 0 y 1 en subintervalos, uno para cada símbolo y de tamaño proporcional a su probabilidad.

2. Tomar un símbolo del mensaje para iniciar la secuencia (en el ejemplo que se muestra en la Fig. 2.4, sería la a).

3. Acotar el número fraccionario asignado a la secuencia con los límites del subintervalo de ese símbolo (en el ejemplo sería de $0,1$ a $0,5$).

4. Escalar y desplazar los subintervalos originales (los que cubren el intervalo 0 a 1) de modo que ahora cubran el subintervalo que actualmente acota al número fraccionario.

5. Si la precisión usada lo permite, tomar el siguiente símbolo del mensaje, agregarlo a la secuencia y repetir los dos pasos anteriores; si no, la secuencia quedará representada por cualquier número comprendido dentro de la acotación actual.

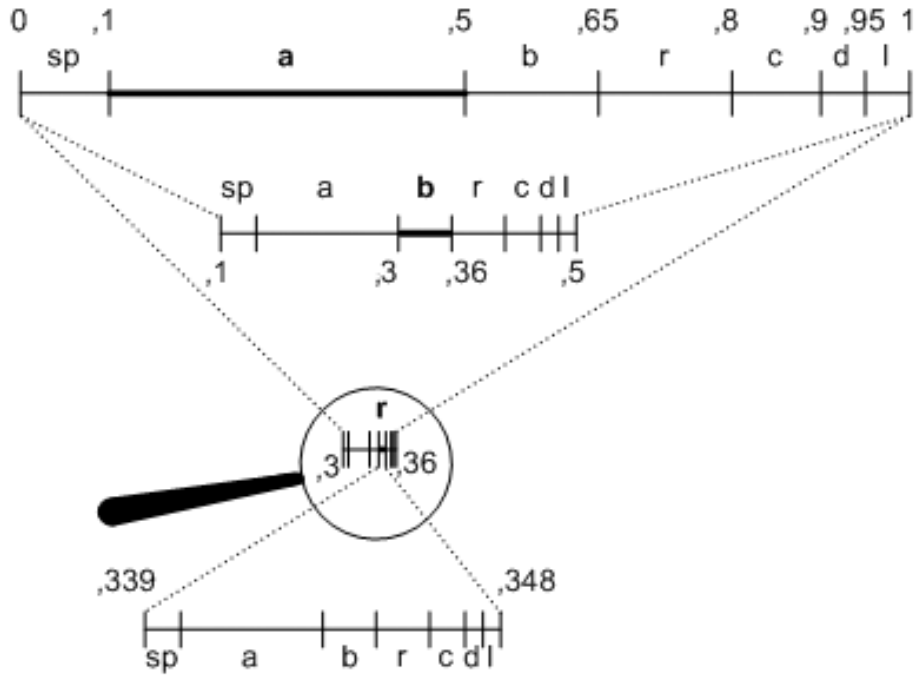


Fig. 2.4: Codificación aritmética

6. Repetir los pasos a partir de número 2 para otra secuencia y así continuar hasta terminar el mensaje.

La Fig. 2.4 ilustra la codificación aritmética cuando se utiliza *abracadabra la cabra* como mensaje. Se generan 7 subintervalos, uno para la a de tamaño 2/5, uno para la b y otro para la r de tamaño 3/20, uno para la c y otro para el *espacio* de tamaño 1/10 y uno para la d y otro para la l de tamaño 1/20. Los subintervalos se acomodaron siguiendo el orden *espacio*, a, b, r, c, d y l, siendo ésta una selección arbitraria. Así, el primer símbolo del mensaje, a, fija 0,1 y 0,5 como límites para el número fraccionario; al tomar el segundo símbolo, b, la acotación queda en 0,3 y 0,36; al tomar el tercer símbolo, r, los límites serán 0,339 y 0,348; si se decide finalizar en ese punto la secuencia, entonces el código de *abr* será el número binario que con la menor cantidad de bits exprese un número comprendido entre esos límites. En este caso el código será $,01011$ que representa al decimal $0,34375$, requiriéndose, por lo tanto, sólo 5 bits en lugar de los 24 que se usarían empleando el código ASCII. Claro que el mensaje *abracadabra la cabra* está pensado para obtener una buena compresión.

En el decodificador, como $0,34375$ está ubicado en el rango de la a ($0,1$ a $0,5$), ése es el primer símbolo; entonces, al número fraccionario se le resta el límite inferior ($0,1$) y se divide entre su probabilidad ($2/5$), lo cual da $0,609375$; repitiendo el procedimiento, este número está en el intervalo de la b ($0,5$ a $0,65$), que será el segundo símbolo de la secuencia, se resta el límite inferior, $0,5$, y se divide entre su probabilidad, $3/20$, con lo cual se obtiene $0,72916667$; este último valor se encuentra dentro del subintervalo de la r ($0,65$ a $0,8$) y así concluye la decodificación.

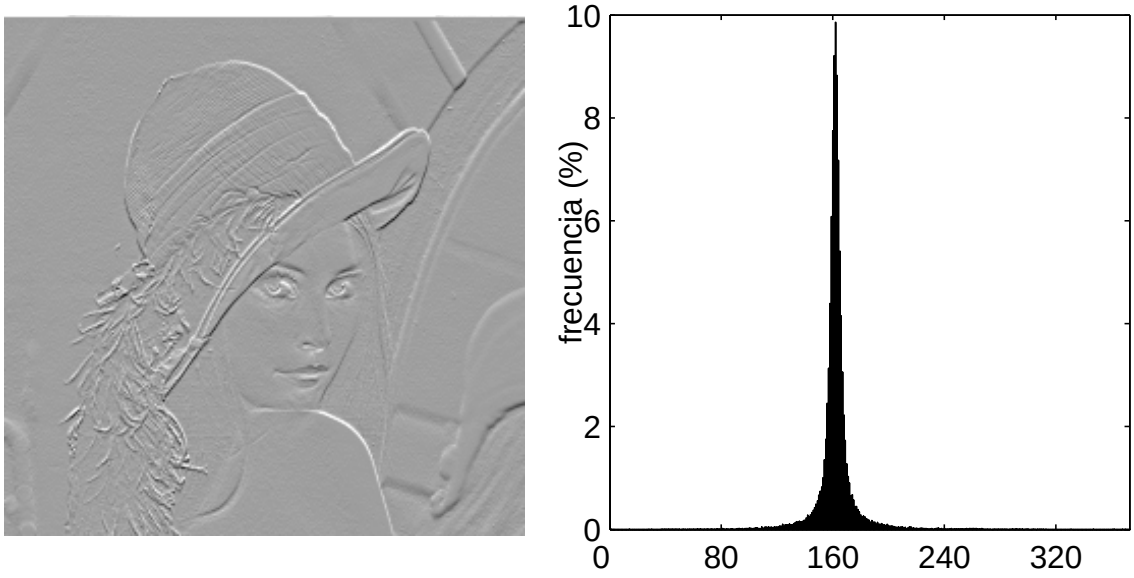


Fig. 2.5: Imagen transformada y su histograma

Redundancia entre pixeles

Al recorrer la imagen a lo largo y ancho, el valor de la intensidad cambia suavemente de un pixel a los adyacentes. Los cambios ocurren de manera más o menos abrupta sólo en los pixeles que forman parte de la frontera entre dos objetos de la imagen, a estos pixeles se les asigna el atributo de pixeles del *borde*. Por tanto, a excepción de los pixeles del borde, existe una alta interdependencia entre cualquier par de pixeles vecinos, de modo que la información que proporciona cada pixel por separado es muy pequeña y, por consiguiente, si cada pixel se codifica de manera independiente esa codificación será muy ineficiente. La interdependencia entre pixeles vecinos se mide por medio de la función de autocorrelación de la imagen, es por eso que se dice que los pixeles en la representación espacial de una imagen están correlados.

Para ilustrar el efecto de la correlación entre pixeles de una imagen, en la Fig. 2.5 se muestra la imagen Lena y su histograma después de aplicarle la transformación expresada mediante la Ec. 2.8. Cada elemento de la matriz resultante es igual a la diferencia entre el valor del pixel en la imagen original y el del pixel vecino superior. Los elementos de la primera fila de la matriz son iguales a los pixeles de la primera fila de la imagen original. Para poder mostrar la imagen, la matriz transformada se desplazó y se escaló de modo que las diferencias quedaran comprendidas en el rango 0 a 255. El histograma y la entropía se obtuvieron a partir de la matriz transformada con desplazamiento, pero sin escalamiento.

$$L'_{i,j} = \begin{cases} L_{i,j}, & i = 1, j = 1, 2, ..., 256 \\ L_{i,j} - L_{i-1,j}, & i = 2, 3, ..., 256; j = 1, 2, ..., 256 \end{cases} \tag{2.8}$$

El histograma muestra una distribución normal que tiene su media alrededor del nivel 160 y una varianza muy baja. La media es 160 porque ese valor es el del desplazamiento que se aplicó a la matriz. Por lo tanto, se tiene que un gran número de las diferencias entre pares de pixeles verticalmente adyacentes son cero o cercanas a cero como consecuencia de la correlación entre pixeles en la imagen original.

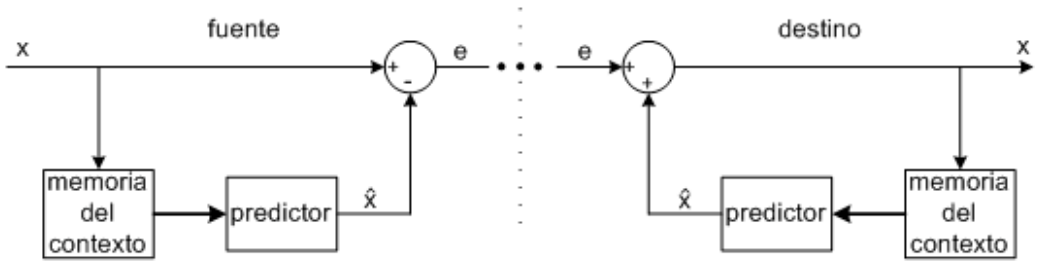


Fig. 2.6: Codificación predictiva

La entropía para la imagen transformada es igual a 5,0438 bits, siendo este valor aproximadamente 2,5 bits menor que la entropía de la imagen original. En otras palabras, cada pixel de la imagen original aporta en promedio un poco más de 7,5 bits de información, en tanto que los de la imagen transformada únicamente 5 bits. Esto se debe a que cada pixel en la representación original de la imagen no incorpora su dependencia respecto de los pixeles vecinos, mientras que cada elemento de la matriz que representa a la imagen transformada sí la incluye, aunque sea de manera burda en este ejemplo. El menor valor de la entropía para la imagen transformada también significa que se puede obtener una codificación con menor cantidad de bits por código en promedio manteniendo la capacidad para reconstruir completamente la imagen original en el destino. En otras palabras, reduciendo la correlación entre pixeles y después aplicando una codificación eficiente se puede lograr una mayor compresión de la imagen en relación a la que se obtendría aplicando únicamente la codificación.

Para comprimir una imagen, por lo que se refiere a la reducción de la redundancia entre pixeles, se han desarrollado principalmente dos técnicas. Una de ellas se centra en generar funciones óptimas para la predicción de la intensidad de cada pixel a partir de las intensidades de los pixeles vecinos y se conoce como *codificación diferencial* (differential coding) o *codificación predictiva* (predictive coding). La mayoría de las técnicas de este tipo se derivan de tres trabajos presentados a mediados del siglo pasado que han sido pilares fundamentales en el campo de la compresión de imágenes [6] [7], [8]. El valor proporcionado por la función de predicción para cada pixel se resta del valor del pixel y la diferencia resultante es lo que se transmite, tal como se ilustra en la Fig. 2.6. A esta diferencia e se le denomina *error de predicción.*

Se llama contexto al grupo de pixeles que son vecinos del que está siendo codificado y a partir de los cuales se predice su valor. El contexto en el lado del destino es igual al contexto en el lado de la fuente, lo mismo ocurre con la función de predicción. El contexto sólo puede incluir pixeles que ya han sido codificados, de otra manera la codificación no podría ser revertida en el destino. Es así que cuando el valor de un pixel se ha reconstruido en el destino, se almacena para incorporarse en el contexto en el momento apropiado. El predictor es lineal si la función de predicción es lineal, es decir, si su salida es una combinación lineal de los elementos del contexto.

La técnica de codificación diferencial o predictiva forma parte de otra que tiene un campo de aplicación más extenso y a la que se conoce como *modulación de códigos por pulsos diferenciales* (DPCM); asimismo, frecuentemente se le llama ADPCM en el caso en que los coeficientes de la función de predicción se puedan ir modificando para adaptarse a la distribución probabilística de la fuente cuando ésta sea variante con el tiempo.

Como ejemplo de un predictor lineal típico, se encuentra el empleado en la sección dedicada a la compresión sin pérdidas del estándar JPEG (Joint Photographic Experts Group) propuesto por la CCITT (International Telephone and Telegraph Consultative Commitee) en 1992 [9]. El estándar establece un esquema basado en la codificación entrópica del error de predicción. Para la predicción se utiliza el contexto que se muestra en la Fig. 2.7 y para la función de predicción se puede elegir entre los 8 casos que se describen en la Tabla 2.1. En el contexto, los pixeles obscuros (a, b y c) son los que ya han sido codificados y el claro es el que está siendo codificado.

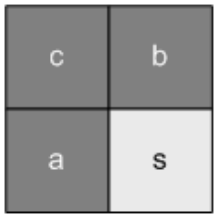


Fig. 2.7: Contexto para el estándar JPEG sin pérdidas

Caso	Función
1	$\hat{s}$=f(a,b,c) = 0
2	$\hat{s}$=f(a,b,c) = a
3	$\hat{s}$=f(a,b,c) = b
4	$\hat{s}$=f(a,b,c) = c
5	$\hat{s}$=f(a,b,c) = a+b-c
6	$\hat{s}$=f(a,b,c) = a+(b-c)/2
7	$\hat{s}$=f(a,b,c) = b+(a-c)/2
8	$\hat{s}$=f(a,b,c) = (a+b)/2

Tabla 2.1: Predictores utilizados por JPEG

En el otro tipo de técnicas empleadas para reducir la correlación entre pixeles, éstas se basan en aplicar cierta transformación sobre la imagen. En las siguientes secciones se tratarán las transformaciones comúnmente empleadas para la compresión de imágenes.

Irrelevancia psicovisual

El ojo humano no responde con la misma sensibilidad a toda la información visual. Cierta información tiene menor importancia con relación a otra en el proceso visual normal. Se dice que esta información es psicovisualmente irrelevante y, por lo tanto, se puede eliminar sin que se altere significativamente la calidad de la percepción de la imagen.

En general, un observador busca características de identificación, como bordes o regiones de diferentes texturas, y luego las combina mentalmente en grupos reconocibles. A continuación, el cerebro relaciona estos grupos con el conocimiento previamente obtenido con el fin de completar el proceso de interpretación de la imagen.

A diferencia de la redundancia de código y la redundancia entre pixeles, la irrelevancia psicovisual está determinada tanto por aspectos objetivos del sistema de percepción óptico, como por aspectos subjetivos y, por tanto, la evaluación de las técnicas de compresión de imágenes que reducen la irrelevancia psicovisual contiene también aspectos subjetivos.

2.2.3 Clasificación de compresores de imágenes

Los sistemas de compresión de imágenes se clasifican en dos grandes grupos: a) compresión sin pérdidas y b) compresión con pérdidas. En los dos grupos, las etapas que constituyen el sistema deben ser reversibles, de manera que la imagen se pueda reconstruir en el destino. La imagen reconstruida puede ser idéntica a la imagen original o puede ser ligeramente diferente.

Compresión sin pérdidas

En el caso de los compresores sin pérdidas la imagen reconstruida debe ser idéntica a la original. Para lograr esto se requiere que cada una de las etapas entregue a la salida la información íntegra que se le proporciona a la entrada. Es decir, en cada etapa la información puede cambiar su forma pero debe ser totalmente preservada de modo que, con referencia a la Fig. 2.8, se cumpla la igualdad $Irec{=}Iorig$.

Para un compresor sin pérdidas la única medida de rendimiento es su *tasa de compresión* (compression rate), es adimensional y se expresa normalmente como *cr:1*. La tasa de compresión se calcula dividiendo el número total de bits necesarios para la representación original de la imagen entre la cantidad total de bits requeridos para su representación a la salida del compresor, es decir:

$$cr = \frac{8\,M\,N}{\hat{b}\,M\,N} = \frac{8}{\hat{b}}, \tag{2.9}$$

siendo $\hat{b}$ el promedio de bits por pixel expresado por la Ec. 2.7. Al valor de $\hat{b}$ se le denomina *tasa de bit* (bit rate), se expresa en *bits por pixel* (bpp) y también se emplea comúnmente como medida de la compresión.

La tasa de compresión para un compresor sin pérdidas es baja, difícilmente se alcanza un valor de 2:1 (4 bits por pixel). Sin embargo, los sistemas de compresión sin pérdida son obligatorios en aplicaciones tales que por razones legales o científicas las pérdidas de información son inaceptables.

En un sistema de compresión de imágenes sin pérdidas se pueden distinguir dos etapas que operan secuencialmente. La primera es la etapa de decorrelación y la segunda es la de codificación. En la Fig. 2.8 se ilustra el sistema completo, incluyendo el compresor en el lado de la fuente y el descompresor en el lado del destino. Para la compresión, primero se reduce la correlación entre pixeles y posteriormente se aplica alguna técnica de codificación estadística. En el lado del destino la secuencia sigue el orden inverso, al igual que el propósito de sus etapas.

Para la etapa de decorrelación se utilizan tanto la codificación predictiva anteriormente descrita (Fig. 2.6), como las técnicas de transformación, sólo que en ambos casos se tiene el condicionante de que el proceso debe ser totalmente reversible, es decir, proporcionar la posibilidad de recuperar la información íntegramente. Si se emplea la codificación predictiva para compresión sin pérdidas, lo apropiado es el uso de predictores lineales.

Las técnicas de transformación apropiadas para la compresión sin pérdidas son las lineales con funciones reales. Por tanto, transformadas como la *DCT*, la *wavelet* o la de *Fourier* son

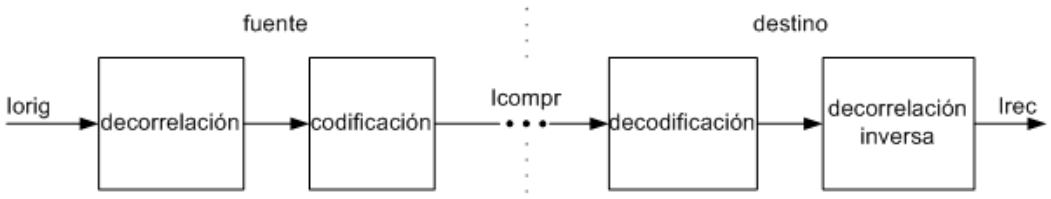


Fig. 2.8: Esquema del sistema de compresión sin pérdidas

inapropiadas. La transformación más comúnmente usada en este caso es la denominada *Walsh-Hadamard* [10] [11] cuya matriz de transformación reúne una serie de propiedades que simplifican su aplicación.

Para la etapa de codificación, además de las métodos estadísticos o entrópicos ya señalados (codificación Huffman y codificación aritmética), es necesario mencionar la técnica conocida como *LZW* (Lempel Ziv Welch) [12] [13] que tiene la particularidad de proporcionar al mismo tiempo reducción de la correlación entre pixeles. La técnica *LZW* cae dentro de la familia de *compresores basados en diccionarios* y fue propuesta originalmente para compresión de textos. Su funcionamiento está basado en una búsqueda en diccionarios dinámicos. Se toma una secuencia de símbolos de la imagen a comprimir y se intenta encontrar una coincidencia dentro del diccionario; si la coincidencia se logra, la secuencia de entrada se reemplaza por el índice de la posición donde se encontró en el diccionario; si no, se agrega al diccionario y se toma otra secuencia para su búsqueda.

A partir de los métodos y técnicas mencionados, se han desarrollado diversos esquemas de compresión sin pérdidas, incluyéndose varios de ellos como componentes en los estándares de compresión de imágenes y video más recientes. En [14] se presenta una revisión completa y actualizada de esos esquemas. Cabe destacar que las técnicas desarrolladas para reducir la redundancia de código han alcanzado niveles de rendimiento que están muy próximos al valor óptimo. En consecuencia, los esfuerzos de investigación se están dirigiendo principalmente a abordar el aspecto de la correlación.

Compresión con pérdidas

Tomando en consideración los aspectos relativos a la redundancia psicovisual, se pueden obtener tasas de compresión más altas que las que se consiguen con la compresión sin pérdidas. Por ejemplo, si se suavizan las regiones de textura de la imagen dejando intactos los bordes, la interpretación de la imagen no cambia en absoluto y los cambios en la apreciación visual pueden ser poco significativos. El suavizado de una imagen se obtiene reduciendo la diferencia entre pixeles vecinos y, por tanto, las técnicas de codificación diferencial operando sobre imágenes suavizadas obtienen mayores tasas de compresión.

El análisis de la redundancia o irrelevancia psicovisual de las imágenes ha dado lugar al surgimiento de una nueva rama de investigación y estudio que involucra a ciertas áreas de la ingeniería, de la medicina y de la psicología. Esta rama forma parte del campo de conocimiento conocido como *factores humanos* (human factors) y se le ha denominado *compresión perceptual de imágenes* (perceptual image compression). Su propósito es diseñar modelos simples y eficientes de mecanismos de procesamiento espacial, temporal y de color en los humanos para incorporarlos a las nuevas tecnologías de compresión de imágenes y video.

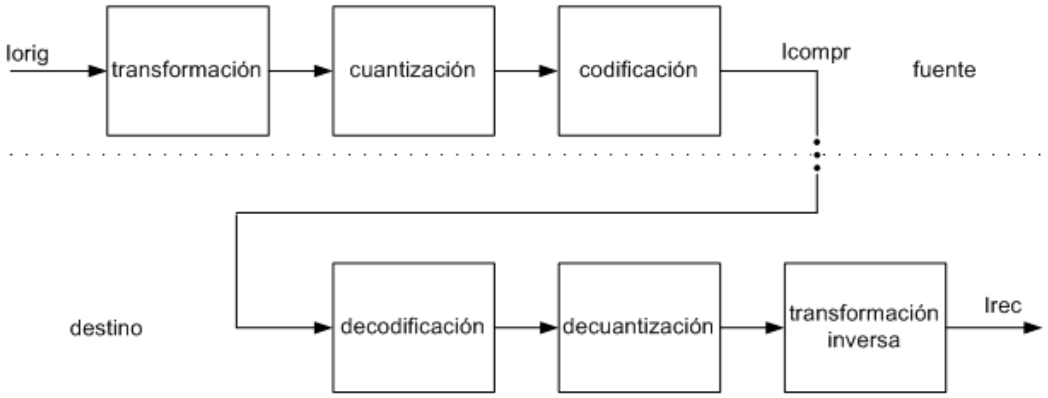


Fig. 2.9: Esquema del sistema de compresión con pérdidas

El área de aplicaciones médicas es una de las más prolíficas en cuanto a la contribución de la percepción visual de imágenes comprimidas; sólo como ejemplo, se puede citar una publicación reciente enfocada en la percepción visual de imágenes producidas por resonancia magnética [15]. También cabe mencionar que los estudios de percepción visual se han introducido en una de las etapas del estándar JPEG.

Así, un compresor de imágenes con pérdidas es aquél que además de la redundancia de código y de la correlación entre pixeles, también explota la irrelevancia psicovisual con la finalidad de obtener altas tasas de compresión aun a costa de que la imagen reconstruida en el destino no sea idéntica a la original, pero sí lo más similar posible, sobre todo desde el punto de vista perceptual, aunque son también importantes las medidas cuantitativas.

El esquema general de un sistema de compresión con pérdidas se muestra en la Fig. 2.9. Una diferencia que surge a primera vista al compararlo con el compresor sin pérdidas es el bloque de cuantización en el lado de la fuente y su recíproco decuantizador en el lado del destino.

Al igual que en el caso de la compresión sin pérdidas, la etapa de transformación tiene el propósito de reducir la correlación entre pixeles, sólo que ahora se tiene una mayor libertad al momento de seleccionar el tipo de transformada, esto debido a que ya no existe el imperativo de preservar la información en su totalidad, es decir, son aceptables en cierto grado las pérdidas que ocurren al aplicar la transformada directa en la fuente y las que se presentan después al reconstruir la imagen en el destino mediante la transformación inversa.

Cabe decir que el objetivo de la etapa de transformación se puede lograr también mediante la técnica DPCM ya mencionada, sólo que utilizando esquemas de predicción no lineal capaces de reducir la correlación entre pixeles en mayor medida al eliminar también dependencias de mayor orden.

De la cuantización ya se ha hablado al inicio de este capítulo, en referencia a la discretización de los valores de intensidad de una imagen. En general, la cuantización es una regla de correspondencia entre dos conjuntos de valores que tienen diferente tamaño, siendo el conjunto imagen el de menor tamaño como condición para que la cuantización produzca una compresión. Al representar la intensidad de un pixel por medio de 256 niveles de gris, el tamaño del conjunto dominio es infinito, mientras que el del conjunto imagen es 256.

De acuerdo a la Fig. 2.9, los elementos del conjunto imagen del cuantizador son en sí los símbolos del alfabeto a codificar; por tanto, el código será más compacto en la medida en que

ese conjunto imagen tenga menor tamaño. Para ilustrar el proceso de cuantización, se puede tomar como ejemplo la aplicación de la cuantización escalar uniforme, que es la forma más simple y está establecida por:

$$\hat{x} = \left\lfloor \frac{x}{2^k} \right\rfloor, \tag{2.10}$$

donde 2^k es el paso de cuantización, x es el valor original recibido por el cuantizador y $\hat{x}$ es el valor cuantizado.

En este caso, si el alfabeto original consta de n símbolos (tamaño n para el conjunto dominio), entonces el alfabeto a codificar tendrá sólo $n/2^k$ símbolos (tamaño $n/2^k$ para el conjunto imagen). Entonces, la razón de compresión introducida por el cuantizador será $\log_2 n/(\log_2 n - k)$. Por ejemplo, si $n = 256$ y $k = 3$, la razón de compresión será 1,6:1.

La cuantización se divide en dos grandes tipos, la escalar y la vectorial. En la cuantización escalar los símbolos se cuantizan uno por uno, mientras que en la vectorial se cuantizan por grupos. El empleo de un tipo u otro depende de la aplicación específica. Por ejemplo, para la cuantización de los coeficientes que resultan de la transformada wavelet se ha demostrado que es más eficiente la cuantización vectorial [16]. La sección 2.5 estará dedicada a la cuantización vectorial, siendo uno de los temas centrales de esta tesis.

Al evaluar el rendimiento de un compresor con pérdidas se deben tomar en cuenta dos medidas que se pueden considerar puramente de tipo cuantitativo y una que tiene que ver más con la percepción visual de la imagen reconstruida. De una de las medidas cuantitativas ya se ha hablado y tiene dos formas de expresarse, la razón de compresión y la tasa de bit. La segunda medida está encaminada a estimar la distorsión numérica que sufre la imagen en el proceso de compresión-reconstrucción. Este segundo criterio tiene también dos formas de expresarse, que son el *error medio cuadrático* (MSE) y el *valor pico de la relación señal a ruido* (PSNR):

$$\begin{aligned} MSE &= \frac{1}{NM} \sum_{i=1}^{M} \sum_{j=1}^{N} (x_{ij} - \hat{x}_{ij})^2, \\ PSNR &= 10\, log_{10}\, \frac{2^n - 1}{MSE}, \end{aligned} \tag{2.11}$$

siendo M y N la cantidad de filas y columnas de la imagen, respectivamente; x_{ij} la intensidad del pixel ubicado en la fila i, columna j de la imagen original; y $\hat{x}_{ij}$ la intensidad del pixel en la misma posición de la imagen reconstruida. n es la cantidad de bits utilizados para cuantizar los niveles de gris, por lo que generalmente tiene un valor de 8, de modo que el numerador de la segunda expresión es 255.

El criterio MSE proporciona una medida adimensional y la calidad de la compresión será mejor en cuanto menor sea su valor. El PSNR se expresa en decibelios y entre mayor sea su valor, mejor será el rendimiento del compresor evaluado.

Es obvio que existe un compromiso entre los dos criterios de medida. Es decir, a mayor tasa de compresión, menor relación señal a ruido y viceversa.

La valoración perceptual de la calidad de los compresores ha surgido recientemente como un campo de investigación que tiene como base el *sistema visual humano* (HVS). Se están desarrollando procedimientos de medición muy completos, tanto para evaluar compresores como para proporcionar elementos para su ajuste. En [17] se expone un método multifactorial para la evaluación de compresores que toma en cuenta los siguientes factores: efecto de bloque,

errores de borde y disparidad visual. Estos tres factores se estiman numéricamente y al final se engloban en un índice de rendimiento que se calcula como su suma ponderada.

2.3 Métodos de transformación

Como ya se ha señalado, los métodos basados en la aplicación de ciertas transformadas sobre la imagen a comprimir dan lugar a una reducción de la correlación que existe entre los pixeles. Pero no sólo es eso, al transformar una imagen la información que está contenida en ella cambia de forma, es decir se lleva de un dominio a otro, lo cual muchas veces facilita y agiliza su tratamiento. El propósito de esta sección es exponer algunos aspectos relevantes de las transformadas comúnmente empleadas para la compresión de imágenes. Para esto, a partir de la transformada discreta de Fourier, se plantean los aspectos fundamentales de las transformaciones, posteriormente el estudio se extiende al caso de dos dimensiones y por último se abordan las transformadas de interés para la compresión de imágenes. La transformada wavelet, por su importancia en sí y la que tiene en esta tesis se trata en la siguiente sección.

2.3.1 Generalidades

Una transformación consiste en traducir la información contenida en una señal de un dominio a otro. Generalmente el dominio original es el tiempo, en el caso de las imágenes es el espacial; el dominio final es comúnmente la frecuencia. El propósito de llevar a cabo ese cambio de dominio es el de realzar ciertas propiedades de la señal que se quiere procesar. Por ejemplo, para eliminar el ruido de una señal su tratamiento en el dominio de la frecuencia es más simple, basta con remover las componentes de frecuencia en las cuales se observa que el ruido predomina sobre la señal.

La transformada discreta de Fourier

La *transformada de Fourier* (FT) ha sido durante mucho tiempo la más ampliamente estudiada y aplicada para el procesamiento de señales, entre otros campos. La *transformada discreta de Fourier* (DFT) resulta al considerar el tiempo y la frecuencia como variables discretas en la FT. La función de transformación es:

$$\begin{aligned} DFT : \{y_k\} \rightarrow \{Y_u\} \quad 0 \leq k, u \leq N-1, \\ Y_u = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{-j2\pi \frac{u}{N} k} y_k, \end{aligned} \tag{2.12}$$

siendo $\{y_k\}$ el conjunto de N valores obtenidos al muestrear la señal y y $\{Y_u\}$ los N correspondientes valores en el dominio de la frecuencia que se generan como resultado de la transformación.

Si el muestreo de la señal se realiza con una frecuencia f_m, es decir tomando valores de y cada $\Delta t = 1/f_m$ segundos, entonces los valores de Y_u corresponderán a las frecuencias

determinadas por:

$$S_k = \begin{cases} \dfrac{k}{N} f_m, & 0 \leq k \leq \dfrac{N}{2} \\ \dfrac{N-k}{N} f_m, & \dfrac{N}{2} + 1 \leq k \leq N-1 \end{cases} . \tag{2.13}$$

Los valores de frecuencia van de 0 a $f_m/2$ que es la *frecuencia de Nyquist* o *frecuencia de doblez* (folding frequency).

Bases vectoriales

La Ec. 2.12 se puede reescribir para describir la transformada matricialmente así:

$$\begin{bmatrix} Y_0 \\ Y_1 \\ \vdots \\ Y_{N-1} \end{bmatrix} = \begin{bmatrix} W_{0,0} & W_{0,1} & \dots & W_{0,N-1} \\ W_{1,0} & W_{1,1} & \dots & W_{1,N-1} \\ \vdots & \vdots & \ddots & \vdots \\ W_{N-1,0} & W_{N-1,1} & \dots & W_{N-1,N-1} \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{N-1} \end{bmatrix}, \tag{2.14}$$

donde los elementos de la matriz $\mathbf{W}$ se obtienen a partir de la función de transformación, es decir:

$$W_{u,k} = \frac{1}{\sqrt{N}} e^{-j2\pi \frac{u}{N} k}. \tag{2.15}$$

La Ec. 2.14 se puede escribir de manera compacta de este modo:

$$\mathbf{Y} = \mathbf{W}\mathbf{y}. \tag{2.16}$$

Se puede probar que la matriz $\mathbf{W}$ es unitaria, es decir que su inversa es igual a su transpuesta conjugada:[3]

$$\mathbf{W}^{-1} = \mathbf{W}^{*T}. \tag{2.17}$$

Eso significa que las filas de $\mathbf{W}$ forman una base ortonormal del espacio vectorial de dimensión N.

A la operación que devuelve la información a su dominio original se le conoce como *transformada inversa* y así en el caso de la DFT se habla de la *transformada inversa de Fourier* (IDFT), descrita por:

$$\mathbf{y} = \mathbf{W}^{*T}\mathbf{Y}. \tag{2.18}$$

Por lo tanto, $Y_0, Y_1, ..., Y_{N-1}$ son los coeficientes de la combinación lineal que genera el vector $\mathbf{y} = [y_0, y_1, ..., y_{N-1}]^T$ a partir de la base vectorial $\mathbf{W}$, entonces:

$$\mathbf{y} = Y_0\mathbf{W}_0^T + Y_1\mathbf{W}_1^T + \ldots + Y_{N-1}\mathbf{W}_{N-1}^T. \tag{2.19}$$

En otras palabras, cada uno de los coeficientes producidos por la transformación indica la porción del correspondiente vector de la base que está contenida en el vector transformado y es por eso que se le denomina componente. Desde el punto de vista de las bases vectoriales,

[3] El asterisco indica el complejo conjugado

al proceso consistente en la realización de la transformada directa se le denomina *análisis* y conduce a la descomposición del vector de entrada en porciones de los vectores de la base. Como una operación recíproca, a la aplicación de la transformada inversa se le llama *síntesis* y tiene como propósito reconstruir el vector original a partir de sus componentes y en función de la base vectorial.

A la matriz de transformación se le llama *kérnel* o *núcleo*. Debido a que simplifican el cálculo de su inversa, las matrices unitarias (como es el caso del kérnel de la DFT) y las ortogonales son de particular importancia. Una matriz es ortogonal si es unitaria y sólo contiene elementos reales, de manera que, de acuerdo a la Ec. 2.17, su inversa es igual a su transpuesta.

2.3.2 Extensión a dos dimensiones

Hasta aquí, lo expuesto se ha centrado en la transformada aplicada sobre un array unidimensional, ejemplificándose mediante la DFT operando sobre una serie de N valores obtenidos de una señal mediante su muestreo. Para llevar a cabo una transformación sobre un array bidimensional de valores, como es el caso de la representación espacial de una imagen, la manera que se observa a primera vista consiste en formar un array unidimensional alineando todas las columnas de la matriz en una sola. Pero ésa no es la única manera, ni la más adecuada.

De manera general, a la transformación que toma como entrada el array de valores $y_{i,j}$, $i, j = 0,\ 1,\ \ldots\ N-1$, para producir en su salida el array $Y_{u,v}$, $u, v = 0,\ 1,\ \ldots\ N-1$, se le denomina *transformada bidimensional discreta* y, llamándosele T_{bd}, se le puede representar como la combinación lineal:

$$\begin{gathered} T_{bd} : \{y_{i,j}\} \ \rightarrow\ \{Y_{u,v}\} \quad 0 \leq i, j, u, v \leq N-1, \\ Y_{u,v} = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} y_{i,j}\ \mathtt{f}(i, j, u, v). \end{gathered} \tag{2.20}$$

Si la función $\mathtt{f}(i, j, u, v)$ se puede descomponer como el producto de dos funciones con variables separadas, tal como se muestra en la Ec. 2.21, se dice que la transformación bidimensional es *separable*. La función $\mathtt{f}_r$ es independiente de j y v y por tanto sólo depende de la fila en la que se aplica, mientras que $\mathtt{f}_c$ únicamente depende de la columna. De este modo, la transformación bidimensional se ha separado en dos transformaciones unidimensionales. Entonces, la transformada bidimensional se obtiene aplicando una de las transformaciones unidimensionales sobre las filas (o las columnas) para, a continuación, aplicar la otra sobre las columnas (o filas) de la matriz resultante.

$$\mathtt{f}(i, j, u, v) = \mathtt{f}_r(i, u)\mathtt{f}_c(j, v) \tag{2.21}$$

Si la transformación es separable, la Ec. 2.20 se puede reescribir así:

$$\begin{gathered} T_{bd} : \{y_{i,j}\} \ \rightarrow\ \{Y_{u,v}\} \quad 0 \leq i, j, u, v \leq N-1, \\ Y_{u,v} = \sum_{i=0}^{N-1} \left[\sum_{j=0}^{N-1} y_{i,j} \mathtt{f}_c(j, v) \right] \mathtt{f}_r(i, u) = \sum_{i=0}^{N-1} \mathtt{f}_r(i, u) \sum_{j=0}^{N-1} y_{i,j}\ \mathtt{f}_c(j, v). \end{gathered} \tag{2.22}$$

Si las funciones son idénticas, es decir $\mathtt{f}_r = \mathtt{f}_c = \mathtt{f}$, se tiene el caso de una *transformada simétrica*. Entonces, siendo $\mathbf{F}$ la matriz formada con los valores de $\mathtt{f}$ evaluada en i, $u = 0, 1 \ldots, N-1$, la transformada se expresa matricialmente así:

$$\mathbf{Y} = \mathbf{F}\boldsymbol{y}\mathbf{F}. \tag{2.23}$$

En los casos en que la matriz $\mathbf{F}$ sea unitaria u ortogonal, la transformada bidimensional inversa es:

$$\boldsymbol{y} = \begin{cases} \mathbf{F}^{*T}\mathbf{Y}\mathbf{F}^{*T} & \text{para } \mathbf{F} \text{ unitaria} \\ \mathbf{F}^{T}\mathbf{Y}\mathbf{F}^{T} & \text{para } \mathbf{F} \text{ ortogonal} \end{cases}. \tag{2.24}$$

Análogamente a lo que sucede con la transformada unidimensional, cada uno de los coeficientes que se obtienen al aplicar la transformación directa indica la porción del elemento respectivo de la base contenido en la matriz que está siendo transformada y permiten reconstruirla mediante la transformada inversa. Hablando en el ámbito de las imágenes, la base está constituida por N^2 *imágenes base* y cada uno de los elementos de la matriz de coeficientes que resulta al aplicar la transformada sobre una imagen, representa la porción que ésta contiene de la imagen base correspondiente a la posición donde se ubica dicho elemento. El conjunto de imágenes base se obtiene efectuando el producto externo de todos los pares posibles de vectores fila del kérnel.

Para finalizar este apartado, conviene enfatizar los requerimientos que debe satisfacer una transformación para quedar representada mediante la Eq. 2.23 (análisis) y la Ec. 2.24 (síntesis): a) ser una transformación separable, b) ser una transformación simétrica y c) su kérnel ser una matriz unitaria u ortogonal.

2.3.3 Transformadas empleadas para la compresión de imágenes

Existen algunas transformadas bidimensionales con ciertas características que las hacen apropiadas para ciertas aplicaciones, ya sea porque facilitan o agilizan el procesamiento o porque son más eficientes. En este apartado se revisarán algunas de ellas, incluyendo su función de transformación, su kérnel y ciertas propiedades que las vuelven interesantes para la compresión de imágenes.

La transformada de Karhunen-Loeve (KLT)

La KLT [18] es la transformada que da lugar a la decorrelación óptima. Sin embargo, su kérnel se forma a partir de los autovectores de la matriz de covarianza de la imagen que se quiere transformar. La aplicación de la KLT para la compresión de imágenes ha quedado restringida debido a que es computacionalmente compleja y a que los coeficientes de la matriz de transformación son dependientes de la imagen a transformar, por lo cual se requiere obtenerlos cada vez. Sin embargo, se han desarrollado compresores de imágenes basados en la KLT implementada sobre redes neuronales, algunos de ellos se tratarán en el capítulo 3.

El procedimiento para obtener el kérnel de la KLT unidimensional se puede condensar en los siguientes pasos: a) descomponer la secuencia de valores a transformar en L segmentos no solapados de tamaño N (esto supone que la secuencia consta de L x N valores); b) para cada posición en el segmento obtener la media de sus L valores; c) restar en todos los segmentos el

valor de la media para cada posición;[4] d) calcular y formar la matriz de covarianza C a partir de los L segmentos (la matriz de covarianza será simétrica de tamaño N por N y proporcionará una medida de la correlación entre las posiciones de los segmentos);[5] e) obtener los autovalores y autovectores de la matriz de covarianza; f) formar la matriz de transformación Φ, la cual estará compuesta por los autovectores obtenidos, colocando cada uno como una fila de Φ, comenzando por el correspondiente al mayor autovalor hasta terminar con el correspondiente al menor autovalor.

La matriz Φ es el kérnel de la KLT, por lo tanto, la transformación queda representada por la Ec. 2.25. Φ es una matriz ortogonal, de tal modo que la transformada inversa se obtiene mediante la Ec. 2.26.

$$\begin{aligned} KLT : \boldsymbol{y} &\rightarrow \mathbf{Y}, \\ \mathbf{Y} &= \Phi(\boldsymbol{y} - \boldsymbol{y}_m), \end{aligned} \tag{2.25}$$

siendo $\boldsymbol{y}_m$ el vector formado por los valores medios que se obtienen en el paso (b).

$$\boldsymbol{y} = \Phi^T \mathbf{Y} + \boldsymbol{y}_m \tag{2.26}$$

A la KLT también se le conoce como análisis de componentes principales (PCA), esto debido a que los elementos del vector $\mathbf{Y}$ son en verdad los coeficientes para reconstruir $\boldsymbol{y}$ como una combinación lineal de los autovectores de la matriz de covarianza.

Para comprimir una imagen, ésta se descompone en bloques de N pixeles y cada bloque se toma como un segmento para aplicar el procedimiento descrito, obteniendo así el kérnel de la transformación. El siguiente paso consiste en truncar la matriz del kérnel descartando las últimas P filas. Posteriormente se aplica la transformación sobre cada uno de los bloques de la matriz original. De tal modo, los vectores transformados serán de tamaño $N - P$, con lo cual se habrá obtenido la compresión, además de que la pérdida de información es mínima debido a que la parte de la matriz descartada corresponde a los componentes de menor varianza.

La transformada discreta del coseno (DCT)

La DCT fue definida en [19] junto con el desarrollo de un algoritmo para su cálculo basado en la FFT. Comparada con las transformadas KLT, DFT, HWT[6] y Haar, la DCT presenta el rendimiento más cercano al de la KLT, que es bien conocido como el óptimo. Originalmente se desarrolló para aplicarse en el procesamiento digital de señales y para reconocimiento de patrones. Por su eficiencia para la decorrelación de señales y la simplicidad de su cómputo, la DCT es la que se utiliza más ampliamente en los estándares recientes de compresión de imágenes, destacadamente el JPEG, con excepción de la versión 2000.

La función de transformación de la DCT consiste en un desarrollo de cosenos, de allí su

[4] De este modo los L segmentos estarán representando una secuencia de valores para N variables aleatorias con valor medio igual a cero

[5] Cada elemento en la diagonal principal de C es la varianza propia de la posición correspondiente en los segmentos, mientras que cada elemento fuera de la diagonal principal es la covarianza entre las posiciones respectivas. Si la señal está totalmente decorrelacionada, C será una matriz diagonal

[6] La transformada Hadamard-Walsh, que se tratará adelante, también la de Haar

nombre, y se describe así:

$$DCT : \{y_{i,j}\} \rightarrow \{Y_{u,v}\} \quad 0 \leq i,j,u,v \leq N-1,$$

$$Y_{u,v} = a_u\, a_v \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} y_{i,j}\, cos\frac{\pi(2i+1)u}{2N}\, cos\frac{\pi(2j+1)v}{2N}, \tag{2.27}$$

$$a_u = \begin{cases} 1/\sqrt{N}, & u = 0 \\ \sqrt{2/N}, & u \neq 0 \end{cases}.$$

La función de transformación para la transformada inversa (IDCT) es también un desarrollo de cosenos:

$$IDCT : \{Y_{u,v}\} \rightarrow \{y_{i,j}\} \quad 0 \leq i,j,u,v \leq N-1,$$

$$y_{i,j} = \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} a_u\, a_v\, Y_{u,v}\, cos\frac{\pi(2u+1)i}{2N}\, cos\frac{\pi(2v+1)j}{2N}. \tag{2.28}$$

Al inspeccionar la función de transformación, se observa que ésta es separable y simétrica, y los elementos de la matriz son reales, por lo cual el par de transformadas se puede escribir así:

$$\mathbf{Y} = \mathbf{C}\boldsymbol{y}\mathbf{C},$$

$$\boldsymbol{y} = \mathbf{C}^T\mathbf{Y}\mathbf{C}^T, \tag{2.29}$$

donde $\mathbf{C}$ es el kérnel de la transformación y sus elementos se calculan mediante la Ec. 2.30, siendo a_u las constantes definidas en la Ec. 2.27.

$$C_{i,u} = a_u\, cos\frac{\pi(2i+1)u}{2N} \tag{2.30}$$

En la Fig. 2.10 se muestran las imágenes base para $N = 8$. Cada imagen contiene 64 pixeles formando una matriz cuadrada. La imagen de la esquina superior izquierda presenta una tonalidad constante. A medida que se avanza hacia abajo o hacia la derecha se observa que los cambios de tonalidad pasan de ser suaves (frecuencia baja) en las primeras filas y columnas, hasta ser muy rápidos (frecuencia alta) en las últimas. Sabiendo que las imágenes comunes están caracterizadas primordialmente por las bajas frecuencias, es posible llevar a cabo una compresión con pérdidas aplicando una cuantización escalar no uniforme sobre la matriz de coeficientes que resulta de la transformación. Es decir, la cuantización deberá ser tal que los coeficientes de las imágenes base de alta frecuencia sean modificados en mayor medida que los correspondientes a las imágenes base de baja frecuencia. El estándar JPEG [9] para compresión con pérdidas tiene como base la DCT sobre bloques de 8 x 8 pixeles y establece el uso de tablas de cuantización escalar no uniforme que se eligen de acuerdo a la clase de imágenes para las que el compresor está destinado.

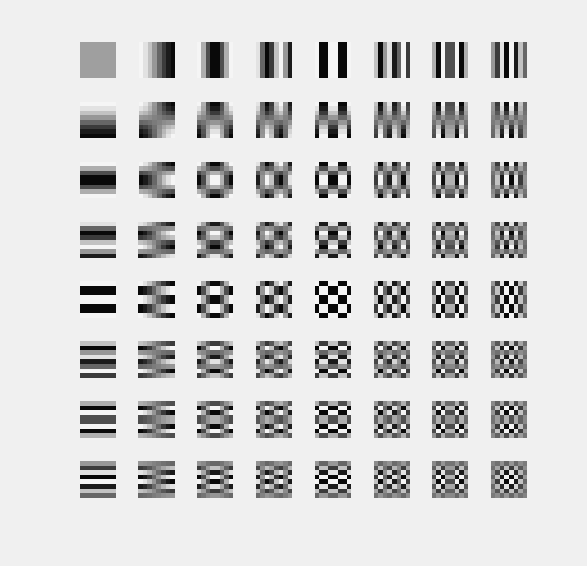


Fig. 2.10: Imágenes base para la DCT

La transformada de Hadamard-Walsh (HWT)

La HWT [10] [11] se ubica dentro del grupo de transformadas con base rectangular y no sinusoidal como es el caso de la DFT y la DCT. Su función de transformación está establecida como se muestra en la Ec. 2.31. Es una transformada separable, ortogonal y simétrica, por lo que se puede escribir como el par que se incluye en las dos últimas líneas de la Ec. 2.31. N es la dimensión del espacio de transformación y debe ser una potencia de 2, de manera que $n = log_2(N)$ es el entero que aparece en el sumatorio.

$$h(i,u) = \frac{1}{\sqrt{N}}\Big(-1\Big)^{\sum_{k=0}^{n-1} b_k(i)b_k(u)},$$

$$b_0(i) = \mathrm{mod}(i,2), \quad b_k(i) = \mathrm{mod}\Big(\frac{i - b_{k-1}(i)}{2}, 2\Big), \tag{2.31}$$

$$\mathbf{Y} = \mathbf{H}\boldsymbol{y}\mathbf{H},$$

$$\boldsymbol{y} = \mathbf{H}\mathbf{Y}\mathbf{H},$$

H es el kérnel de la HWT y se puede obtener evaluando la función $h(i,u)$ pero es más sencillo

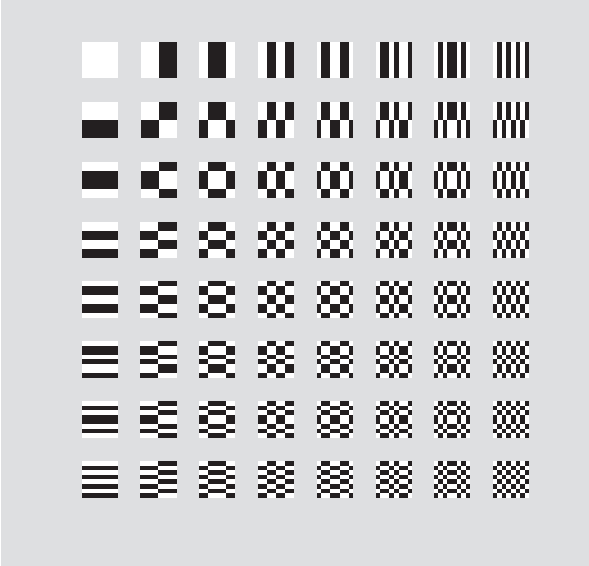


Fig. 2.11: Imágenes base para la HWT

hacerlo recursivamente mediante las siguientes ecuaciones:

$$\mathbf{H}_N = \begin{cases} [+1], & N = 1, \\ \frac{1}{\sqrt{2}} \begin{bmatrix} \mathbf{H}_{N/2} & \mathbf{H}_{N/2} \\ \mathbf{H}_{N/2} & -\mathbf{H}_{N/2} \end{bmatrix}, & N = 2, 4, ... \end{cases} . \tag{2.32}$$

La HWT destaca por la simplicidad a que da lugar el hecho de que el kérnel contenga solamente unos, ya sea con signo positivo o negativo. Como consecuencia, al aplicar la transformación sobre las columnas de la matriz a transformar basta con sumar los valores que correspondan con posiciones de la matriz que tengan +1 y restar los que correspondan a posiciones con -1. La Fig. 2.11 muestra las imágenes base para $N = 8$.

La transformada de Haar(HT)

En el caso unidimensional, la función de transformación para la HT [20] se puede describir como se hace a continuación. Siendo $\{f_i, i = 0, 1, \ldots N-1\}$, el vector de muestras que se quiere transformar, con N igual a una potencia de dos, se crea el *vector promedio bf*, de tamaño $N/2$, que se obtiene al barrer el vector de entrada de izquierda a derecha, tomando sus elementos de dos en dos y obteniendo el promedio de cada pareja; es decir, cada elemento de *bf* es el promedio de dos muestras consecutivas de f. A continuación, se genera el *vector aproximación* $\hat{f}$ de tal manera que $\hat{f}_i$ es igual al valor del promedio al cual f_i dio lugar. Finalmente se obtiene el *vector residuo* r cuyos elementos son las diferencias entre los elementos correspondientes del vector de entrada y del vector aproximación. El proceso queda descrito matemáticamente

por:

$$\begin{aligned} bf_i &= \frac{f_{2i} + f_{2i+1}}{2} \quad i = 0, 1, \ldots N/2 - 1, \\ \hat{f}_i &= f_i - bf_{\lfloor i/2 \rfloor} \quad i = 0, 1, \ldots N - 1, \\ r_i &= f_i - \hat{f}_i \quad i = 0, 1, \ldots N - 1. \end{aligned} \tag{2.33}$$

Se puede comprobar fácilmente que para el vector residuo se cumple que $r_{2i+1} = -r_{2i}$ para $i = 0, 1, \ldots N/2 - 1$. Es decir, examinando el vector residuo de izquierda a derecha se observa que está formado por parejas de valores adyacentes con igual magnitud y signo contrario. Por lo tanto, para poder recuperar los valores del vector de entrada sólo se requiere la mitad de los valores de r, por ejemplo, los que están en posiciones pares. La HT queda descrita por el siguiente par de expresiones:

$$\begin{aligned} bf_i &= \frac{f_{2i+1} + f_{2i}}{2} \\ af_i &= \frac{f_{2i+1} - f_{2i}}{2} \end{aligned} \qquad i = 0, 1, \ldots N/2 - 1. \tag{2.34}$$

De este modo, el vector de entrada con N elementos se ha descompuesto en un par de vectores con sólo la mitad de elementos cada uno. Considerando esta transformación en el ámbito del procesamiento de señales, el proceso descompone el vector de entrada en uno para la banda de frecuencias bajas, bf, y otro para la banda de frecuencias altas, af.

Para el caso bidimensional, el procedimiento es similar. Si la matriz que se quiere transformar es $\{f_{i,j}\ i, j = 0, 1, \ldots N - 1\}$, entonces ésta se barre de izquierda a derecha y de arriba hacia abajo, tomando cada vez bloques no solapados de 2 x 2 elementos para generar las cuatro submatrices bbf, baf, abf, aaf, cada una con $N/2$ x $N/2$ elementos cuyo valores se obtienen así:

$$\begin{aligned} bbf_{i,j} &= \tfrac{1}{2}\big((f_{2i,2j} + f_{2i,2j+1}) + (f_{2i+1,2j} + f_{2i+1,2j+1})\big), \\ baf_{i,j} &= \tfrac{1}{2}\big((f_{2i,2j} - f_{2i,2j+1}) + (f_{2i+1,2j} - f_{2i+1,2j+1})\big), \\ abf_{i,j} &= \tfrac{1}{2}\big((f_{2i,2j} + f_{2i,2j+1}) - (f_{2i+1,2j} + f_{2i+1,2j+1})\big), \\ aaf_{i,j} &= \tfrac{1}{2}\big((f_{2i,2j} - f_{2i,2j+1}) - (f_{2i+1,2j} - f_{2i+1,2j+1})\big), \\ & \qquad i,j = 0,1,\ldots N/2-1. \end{aligned} \tag{2.35}$$

Los paréntesis interiores se han colocado con el propósito de enfatizar el sentido de las bandas de frecuencia. El signo entre los paréntesis indica la banda de frecuencia en el sentido vertical, mientras que el signo que separa a los términos dentro del paréntesis indica la banda de frecuencia en el sentido horizontal. El signo + corresponde a la banda de frecuencias bajas y el − a la de frecuencias altas. Esto mismo está implícito en la denominación de cada matriz, la primera letra indica la banda de frecuencia en el sentido vertical y la segunda en el sentido horizontal.

En la Fig. 2.12 se ilustra la descomposición de Lena en las cuatro sub-bandas. Las dos imágenes en la parte superior corresponden a la banda de frecuencias bajas en el sentido vertical y las dos de la izquierda a la banda de frecuencias bajas en el sentido horizontal. Es evidente que la mayor parte de la información queda concentrada en la sub-banda correspondiente a las frecuencias bajas en ambos sentidos y la menor en la sub-banda correspondiente a las frecuencias altas en los dos sentidos.

La aplicación de la transformada de Haar que da lugar a la descomposición de la imagen en cuatro sub-bandas de la manera descrita, representa sólo un primer nivel de resolución.

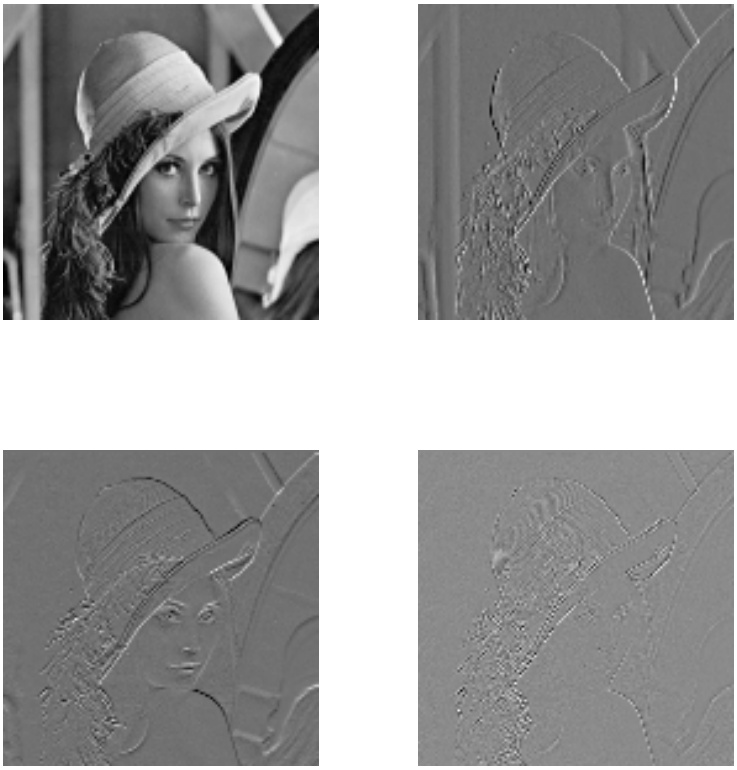


Fig. 2.12: Descomposición de Lena en cuatro sub-bandas mediante la HT

Es decir, el mismo procedimiento se puede aplicar sobre la matriz correspondiente a la sub-banda de frecuencias bajas en los dos sentidos (cuadrante superior-izquierdo) para obtener la descomposición para el segundo nivel de resolución y así sucesivamente. Cabe decir que la transformada de Haar, cuando se le considera desde el punto de vista de una familia de funciones, es un caso de la transformada wavelet.

2.4 La transformada wavelet

La transformada wavelet se puede abordar desde distintos puntos de vista [21][22]. Se presenta en este apartado una rápida revisión de sus fundamentos, destacando aquellos aspectos aplicables al análisis multiresolución.

2.4.1 La transformada wavelet continua

De manera muy general, la transformada wavelet continua (CWT) de una función $f(t)$ es la descomposición de $f(t)$ en un conjunto de funciones $\psi_{s,\tau}(t)$ que forman una base y son

llamadas *funciones wavelet*. La CWT se define así:

$$W_f(s,\tau) = \langle f, \psi_{s,\tau} \rangle = \int_{-\infty}^{\infty} f(t)\psi^*_{s,\tau}(t)dt, \tag{2.36}$$

donde $\langle f, \psi_{s,\tau} \rangle$ es el producto interno de f, la función a transformar, y $\psi_{s,\tau}$, la base funcional.

La transformada wavelet continua inversa está dada por:

$$f(t) = \frac{1}{C_\psi} \int_0^{\infty} \int_{-\infty}^{\infty} W_f(s,\tau)\psi_{s,\tau} ds \frac{d\tau}{\tau^2}. \tag{2.37}$$

Las funciones wavelet se generan efectuando una traslación y un cambio de escala sobre una misma función ψ a la que se le llama la *función madre* o *wavelet madre*, como se muestra a continuación:

$$\psi_{s,\tau}(t) = \frac{1}{\sqrt{s}} \psi\Big(\frac{t-\tau}{s}\Big), \tag{2.38}$$

siendo s el factor de escala y τ el factor de traslación.

Las funciones wavelet $\psi_{s,\tau}(t)$ generadas a partir de la misma función wavelet madre $\psi(t)$ tienen diferente escala s y ubicación τ, pero todas tienen la misma forma. El factor de escala siempre debe ser mayor que cero. Las ondas wavelet se dilatan cuando $s > 1$ y se contraen cuando $s < 1$. De este modo, modificando el valor de s se cubren rangos diferentes de frecuencias. Los valores grandes del parámetro s corresponden a frecuencias bajas, es decir una escala grande para $\psi(t)$, por lo contrario, valores pequeños de s corresponden a frecuencias altas o una escala pequeña para $\psi(t)$.

2.4.2 Descomposición wavelet

La CWT con factores de dilatación y traslación continuos se comporta como una transformada ortonormal en el sentido de que la transformada wavelet inversa permite reconstruir la señal mediante la integración de todas las proyecciones de la señal sobre la base de funciones wavelet. Sin embargo, la base de funciones wavelet no es ortonormal y por lo tanto la CWT es redundante. Una base de funciones wavelet es ortonormal sólo si s y τ se discretizan.

Cuando la función $f(t)$ es continua y las funciones wavelet son continuas con factor de escala y traslación discretas, la WT se convierte en una secuencia de coeficientes wavelet y al proceso se le conoce como la descomposición wavelet. A estas funciones wavelet continuas con factores de escala y traslación discretos se les denomina funciones wavelet discretas. Los factores de escala y traslación discretos se pueden expresar así:

$$\begin{aligned} s &= s_o^i, \\ \tau &= k\tau_o s_o^i, \end{aligned} \tag{2.39}$$

donde i y k son enteros y s_o es el paso de dilatación fijo, que debe ser mayor que uno. En cuanto a τ, su valor depende de s y τ_o es el paso de traslación que debe ser mayor que cero.

Entonces, las funciones wavelet discretas quedan descritas por:

$$\psi_{i,k}(t) = s_o^{-i/2} \psi\Big(s_o^{-i}t - k\tau_o\Big). \tag{2.40}$$

Las funciones wavelet discretas definidas por la Ec. 2.40 serán ortogonales con respecto a sus factores de traslación y dilatación si se selecciona una base wavelet adecuada. La condición de ortogonalidad está dada por:

$$\int \psi_{i,k}(t)\psi^*_{m,n}(t)dt = \begin{cases} 1 & \text{si } i = m \text{ y } k = n \\ 0 & \text{en caso contrario} \end{cases} \qquad \forall\, i,k \in Z\ . \tag{2.41}$$

Para las funciones wavelet que cumplan con esta condición, para la misma escala i, las funciones $\psi_{i,k}(t)$ desplazadas con pasos discretos k son ortogonales; igualmente, para una misma traslación k las funciones wavelet con diferentes escalas i son también ortogonales.

Cualquier señal arbitraria $f(t)$ se puede reconstruir a partir de la base funcional conformada por las funciones wavelet ortogonales como su suma ponderada por los coeficientes wavelet $W_f(s,\tau)$ de la manera siguiente:

$$f(t) = \sum_s \sum_\tau W_f(s,\tau)\psi_{s,\tau}(t). \tag{2.42}$$

Si para la discretización del factor de dilatación se toma $s_o = 2$, el muestreo en la frecuencia será diádico. Al intervalo de muestreo para la frecuencia 2^i se le llama en música una octava, es decir una octava es el intervalo entre dos valores de frecuencia cuya relación es igual a 2. Si además se selecciona $\tau_o = 1$, se tiene el caso de una *función wavelet diádica*. Las funciones wavelet diádicas quedan definidas por:

$$\psi_{i,k}(t) = 2^{-i/2}\psi(2^{-i}t - k). \tag{2.43}$$

2.4.3 Función de escala y bases ortonormales

En el análisis por multiresolución, la transformada wavelet ortogonal se basa en la *función de escala* $\phi(t)$. La función de escala es una función valuada real en el campo de los números reales. Es diferente de la función wavelet, sobre todo porque su valor medio no es igual a cero, pero se normaliza mediante $\int \phi(t)dt = 1$. La función básica de escala se desplaza por el factor de traslación discreto k y se dilata mediante el factor de escala diádico 2^{-i}, para generar la *base de funciones de escala* representada por:[7]

$$\phi_{i,k}(t) = 2^{-i/2}\phi(2^{-i}t - k). \tag{2.44}$$

Siendo $\phi(t)$ la función básica de escala cuyas traslaciones generan el subespacio Vo, en el siguiente nivel de resolución más fino, V_{-1}, $\phi(t)$ se puede expresar como una combinación lineal del conjunto $\{\phi(2t-k)\}$ generado por $\phi(2t)$. De modo que las funciones de escala para dos niveles consecutivos de resolución satisfacen la *relación dos-escalas* expresada por:

$$\phi(t) = \sum p(k)\phi_k(2t - k). \tag{2.45}$$

Los elementos de la secuencia discreta $p(k)$ son los coeficientes de ponderación para formar la combinación lineal, se les llama *coeficientes de inter-escala* y corresponden a los coeficientes de un filtro paso-bajo.

[7] El paso de dilatación y el de traslación no necesariamente deben ser 2 y 1, respectivamente, aunque son estos los valores más apropiados por razones de simplicidad

Las funciones básicas de escala empleadas satisfacen la condición de ortogonalidad, de modo que las funciones obtenidas para diferentes traslaciones $\phi(t-k)$ con $k \in Z$, forman un conjunto ortonormal.

Con su factor de traslación discreto, en cada nivel de resolución i, las funciones de escala y las wavelet forman dos bases ortonormales. Las funciones de escala y las wavelet en múltiples niveles de resolución son la versión dilatada de la función de escala básica y de la wavelet madre, respectivamente.

Si $\psi(t)$ es una wavelet madre en el espacio Vo, entonces se puede representar como una combinación lineal de las funciones de la base ortonormal de V_{-1} constituida por las funciones de escala $\{\phi(2t-k)\}$ como se expresa a continuación:

$$\psi(t) = \sum q(k)\phi_k(2t-k). \tag{2.46}$$

Los elementos de la secuencia $q(k)$ corresponden a los coeficientes de un filtro discreto paso-alto. La Ec. 2.46 es también una relación dos-escalas que permite generar las funciones wavelet a partir de las funciones de escala. Las relaciones dos-escalas expresan la interdependencia entre las funciones continuas $\phi(t)$ (función de escala) y $\psi(t)$ (función wavelet), por una parte, y las secuencias discretas de coeficientes inter-escala $p(k)$ y $q(k)$, por la otra.

2.4.4 Descomposición wavelet multiresolución

Cualquier función $f(t) \in V_0$ se puede representar como la combinación lineal de las funciones de escala trasladadas $\phi(t-k)$ en V_0 de la manera siguiente:

$$f(t) = \sum_k c_o(k)\phi(t-k), \tag{2.47}$$

con los coeficientes dados por:

$$c_o(k) = \langle f, \phi_{o,k} \rangle = \int f(t)\phi(t-k)dt. \tag{2.48}$$

La función que se va a analizar pertenece al subespacio vectorial V_0, el cual corresponde al nivel de discretización al comenzar la descomposición. En el siguiente nivel de resolución más burda, $i = 1$, existen dos subespacios mutuamente ortogonales V_1 y W_1, generados por las bases ortonormales $\{\phi_{1,k}(t)\}$ y $\{\psi_{1,k}(t)\}$, respectivamente. Debido a que el espacio V_0 es la suma directa de V_1 y W_1, existe una forma única de expresar una función $f(t) \in V_0$, como combinación lineal de funciones v_1 y w_1, donde $v_1 \in V_1$ y $w_1 \in W_1$. En particular, la función $f(t) \in V_0$ se puede descomponer en V_1 y W_1 de acuerdo con:

$$f(t) = (P_1 + Q_1)f(t), \tag{2.49}$$

donde las dos componentes P_1 y Q_1 son las proyecciones ortonormales de $f(t)$ sobre V_1 y W_1, respectivamente, y por tanto:

$$\begin{aligned} P_1 f(t) &= \sum_n c_1(n)\phi_{1,n}(t), \\ Q_1 f(t) &= \sum_n d_1(n)\psi_{1,n}(t). \end{aligned} \tag{2.50}$$

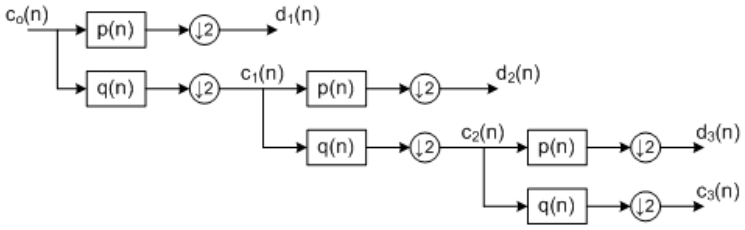


Fig. 2.13: Arbol de filtros para la descomposición wavelet en tres niveles

Los coeficientes se calculan mediante:

$$\begin{aligned} c_1(k) &= 2^{-1/2} \sum_n p(n-2k) c_0(n), \\ d_1(k) &= 2^{-1/2} \sum_n q(n-2k) c_0(n). \end{aligned} \tag{2.51}$$

Generalizando para $i = 2, 3, \ldots M$, se tiene que:

$$\begin{aligned} P_i f(t) &= \sum_n c_i(n) \phi_{i,n}(t), \\ Q_i f(t) &= \sum_n d_i(n) \psi_{i,n}(t), \\ c_i(k) &= 2^{-1/2} \sum_n p(n-2k) c_{i-1}(n), \\ d_i(k) &= 2^{-1/2} \sum_n q(n-2k) c_{i-1}(n). \end{aligned} \tag{2.52}$$

Es de hacer notar que $p(n-2k)$ y $q(n-2k)$ son independientes del nivel de resolución i. Las secuencias sucesivas de aproximaciones discretas $c_i(n)$ son versiones de resolución cada vez más baja de la versión original $c_0(n)$, cada una con un muestreo doblemente esparcido en relación con el de la versión previa. Las secuencias sucesivas de coeficientes wavelet $d_i(n)$ contienen la diferencia de información entre las aproximaciones en los niveles de resolución i e $i-1$. Las secuencias $c_i(n)$ y $d_i(n)$ se pueden calcular a partir de $c_{i-1}(n)$ de manera iterativa por medio del filtrado.

Extendiendo la descomposición hasta el nivel M, la función original $f(t)$ se puede representar como una serie de funciones de detalle más una función de aproximación suavizada. A esta representación se le conoce como la *descomposición en serie wavelet* y está dada por:

$$f(t) = \sum_{k \in Z} 2^{-M/2} c_M(k) \phi(2^{-M} t - k) + \sum_{i=1}^{M} \sum_{k \in Z} 2^{-i/2} d_i(k) \psi(2^{-i} t - k). \tag{2.53}$$

Los coeficientes de aproximación $c_i(n)$ y los coeficientes discretos wavelet $d_i(n)$ se obtienen mediante el algoritmo implementado como un banco de filtros discretos paso-bajo (p_n) y paso-alto (q_n) con operaciones intercaladas de diezmado. El algoritmo en árbol de tres niveles se muestra en la Fig. 2.13.

La reconstrucción se realiza siguiendo un procedimiento inverso al de la descomposición. La secuencia original de la señal $c_0(n)$ se reconstruye con las secuencias de coeficientes de aproximación $c_i(n)$ y de coeficientes wavelet $d_i(n)$ para $0 < i \leq M$, donde $i = M$ es el nivel

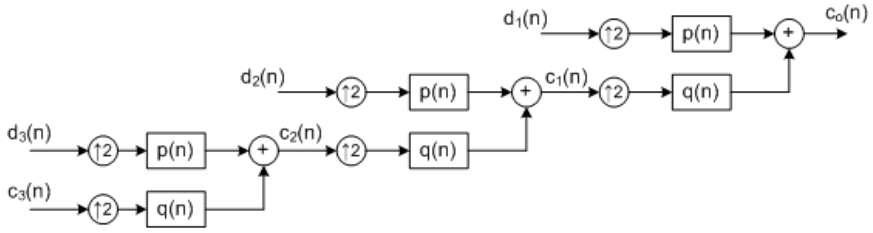


Fig. 2.14: Arbol de filtros para la reconstrucción wavelet en tres niveles

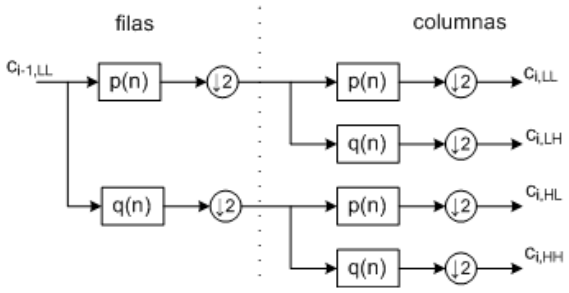


Fig. 2.15: Arbol de filtros para la descomposición wavelet bidimensional

más bajo de resolución. El cómputo iterativo se efectúa mediante la Ec. 2.54 y en la Fig. 2.14 se muestra el árbol con el banco de reconstrucción.

$$c_{i-1}(n) = 2^{-1/2} \sum_k c_i(k) p(n-2k) + 2^{-1/2} \sum_k d_i(k) q(n-2k) \tag{2.54}$$

2.4.5 Transformada wavelet bidimensional

La transformada wavelet bidimensional 2D-WT se aplica a señales 2-D, como es el caso de las imágenes. Mientras que para la DWT unidimensional se usan filtros unidimensionales, para la 2D-DWT se emplean filtros bidimensionales. Los filtros 2-D pueden ser separables o no separables. Un filtro 2-D es separable si su función de transferencia se puede expresar de manera que $f(n1, n2) = f(n1)f(n2)$.

El cálculo se realiza filtrando la imagen primero por filas y después por columnas. En la Fig. 2.15 se muestra la descomposición bidimensional para pasar del nivel de resolución $i-1$ al nivel i. En éste, el bloque de coeficientes $\{c_{i,LH}\}$ contiene las frecuencias altas verticales ya que se obtienen con un filtro paso-bajo en la dirección horizontal y un paso-alto en la dirección vertical. Por su parte, $\{c_{i,HL}\}$ contiene las frecuencias altas horizontales y $\{c_{i,HH}\}$ contiene las frecuencias altas horizontales y verticales. Los $\{c_{i,LL}\}$ tienen la información de frecuencia baja y sobre ese bloque se aplica nuevamente la descomposición para obtener el siguiente nivel de resolución.

En la Fig. 2.16 se muestra la distribución por sub-bandas de la descomposición wavelet bidimensional cuando se aplica a una imagen de N x N pixeles y en la Fig. 2.17 se muestra el resultado obtenido al aplicar la 2D-DWT sobre la imagen Lena de 512 x 512 pixeles para descomponerla en sus cuatro sub-bandas del primer nivel de resolución.

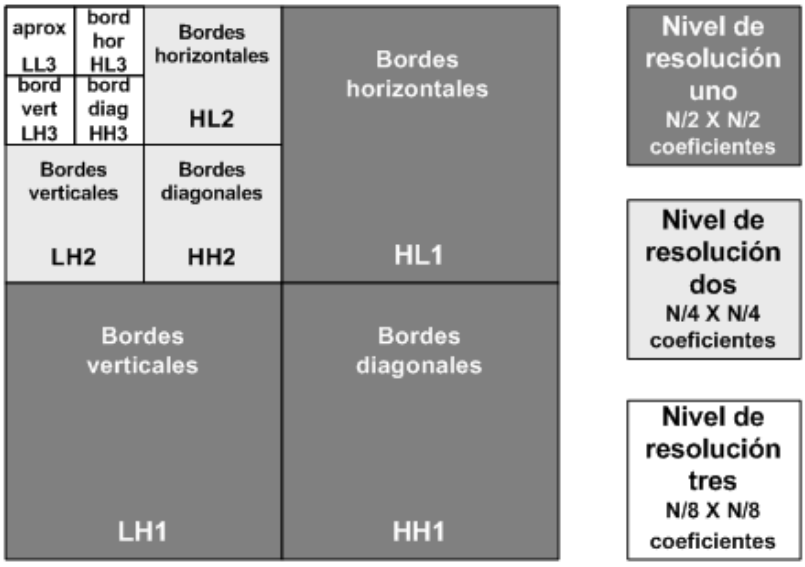


Fig. 2.16: Descomposición wavelet bidimensional en tres niveles

2.5 Cuantización vectorial

Como se mencionó anteriormente, la etapa de cuantización[8] se puede aplicar tomando sólo un elemento de la secuencia de datos de entrada a la vez y se le conoce entonces como cuantización escalar, o bien tomando un grupo de elementos a la vez y es entonces el caso de la cuantización vectorial, se denomina VQ por su sigla del inglés (Vector Quantization). En esta sección se expondrán los fundamentos de la VQ [23].

2.5.1 Definición y generalidades

En su significado más simple, un cuantizador escalar recibe en su entrada un valor numérico x, lo compara con los valores de un conjunto finito predeterminado $x_0, x_1, \ldots, x_{N-1}$ y entrega como salida el x_k tal que $|x - x_k|$ es mínimo, o sea el valor del conjunto más cercano al valor de entrada. Una alternativa para la salida consiste en que ésta sea un índice con la posición del valor más cercano en el conjunto y no el valor más cercano en sí.

La VQ es una generalización de la cuantización escalar considerándola desde el punto de vista expuesto en el párrafo anterior. La diferencia reside en el hecho de que la entrada del cuantizador no es un sólo valor, sino un vector de tamaño L; asimismo, el conjunto finito estará conformado por N vectores del mismo tamaño. De este modo, el cuantizador recibirá como entrada un vector $\mathbf{X}$, lo comparará con el conjunto de vectores $\mathbf{X}_0, \mathbf{X}_1, \ldots, \mathbf{X}_{N-1}$ y entregará como salida el valor de i tal que $d(\mathbf{X}, \mathbf{X}_i)$ sea mínima, siendo d la función que mide la distancia entre dos vectores. En otras palabras, la salida será el índice del vector del conjunto más cercano al de entrada.

[8] Se ha decidido utilizar el término cuantización y no cuantificación. En el diccionario de la Real Academia Española están presentes los verbos *cuantificar* y *cuantizar*. El primer vocablo proviene del latín *quantum* y su significado es "expresar numéricamente una magnitud"; el segundo proviene del inglés *quantize* y significa "aplicar los conceptos y métodos de la mecánica cuántica al estudio de un fenómeno físico"

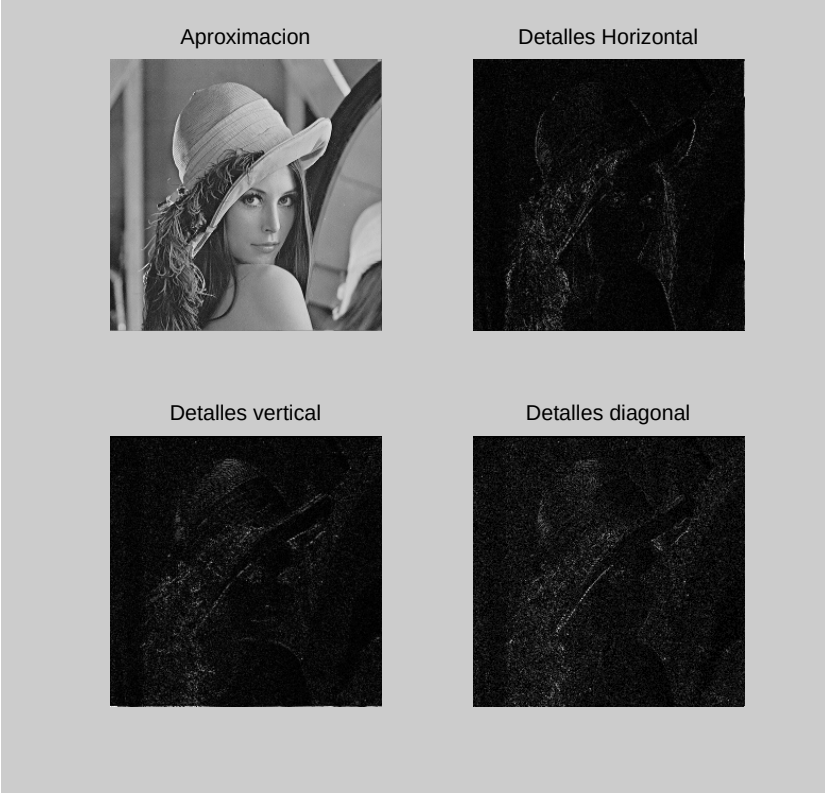


Fig. 2.17: Descomposición wavelet sobre Lena en un nivel de resolución

Al conjunto predeterminado de vectores se le llama *diccionario*, a los vectores de este conjunto se les denomina *vectores código* o *palabras código*. A la cantidad de vectores que conforman el diccionario se le nombra *tamaño* y al número de elementos de que consta cada vector se le conoce como *dimensión*. A la posición que ocupa un vector en el diccionario se le llama *índice*. La distancia $d(\mathbf{X}, \mathbf{X}_i)$ se calcula por medio de una función a la cual se le denomina *medida de distorsión*.

Formalmente, un cuantizador vectorial Q de dimensión k y tamaño N es un mapeo del espacio euclidiano k-dimensional cuyo dominio es el conjunto de todos los posibles valores de $\mathbf{x} \in \Re^k$ y cuyo rango es el conjunto C de N vectores $\mathbf{y}_1, \mathbf{y}_2, \ldots \mathbf{y}_N$, $\mathbf{y}_i \in \Re^k$ para todo $i \in 1, 2, \ldots, N$. Tal cuantizador vectorial lleva a cabo una partición de $\Re^k$ en N regiones o clases, R_i. La i-ésima clase está definida por:

$$R_i = \left\{ \mathbf{x} \in \Re^k : Q(\mathbf{x}) = \mathbf{y}_i \right\}. \tag{2.55}$$

Todos los vectores de la clase R_i quedarán representados por uno sólo al que se le denomina *vector representativo* y no es otro que el vector código o palabra código que ocupa la posición i en el diccionario.

Un cuantizador vectorial se puede descomponer en dos partes denominadas *codificador* y *decodificador*[9]. La Fig. 2.18 ilustra esta descomposición. El codificador entrega como salida

[9]Estos términos pueden dar lugar a confusión ya que se pueden confundir con las etapas que llevan el mismo nombre en un sistema de compresión aunque en realidad nada tienen que ver. Tal vez sea más claro, aunque menos apropiado, denominarlos como *cuantizador directo* y *cuantizador inverso*

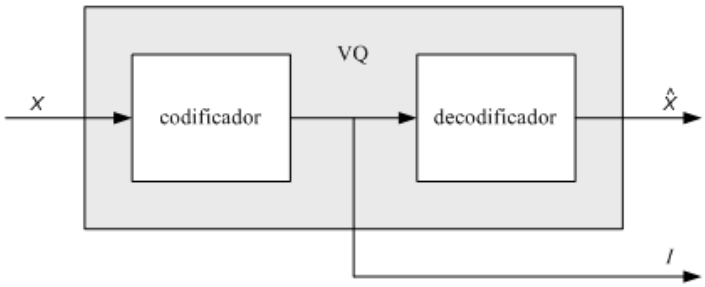


Fig. 2.18: Secciones de un VQ, el codificador y el decodificador

un índice, el cual señala la región de $\Re^k$ en la que se encuentra el vector de entrada de acuerdo a la partición a la que da lugar el VQ. En un sentido más particular, el índice muestra la posición en el diccionario del vector código más cercano al de entrada. En todo caso, el decodificador recibe como entrada el índice y entrega como salida el vector código respectivo.

Si a un VQ se le considera de esta manera, el codificador es la sección que da lugar a la partición de $\Re^k$, mientras que el decodificador consiste básicamente en el diccionario. Por tanto, las condiciones para que un VQ sea óptimo se dividen en dos cuestiones: a) dado un diccionario, ¿cuál es la partición que origina la menor distorsión global?; y b) dada una partición, ¿cual es el diccionario que minimiza esa distorsión?.

Como respuesta a la primera pregunta se tiene que, dado su diccionario, la partición para un rendimiento óptimo del VQ es aquélla en la cual cada región está formada por todos los vectores que son más cercanos al vector código al que está asociada esa región que a cualquiera de los demás, es decir, las regiones de esta partición están determinadas por:

$$R_i = \left\{ \mathbf{x} \in \Re^k : d(\mathbf{x}, \mathbf{y}_i) \leq d(\mathbf{x}, \mathbf{y}_j), j \in \{1, 2, \ldots N\} \right\}. \tag{2.56}$$

En cuanto a la segunda interrogante, dada la partición del VQ, el diccionario para un rendimiento óptimo es el que se forma con los centroides de cada una de las regiones.

En realidad los cuantizadores vectoriales que operan de acuerdo con lo descrito en los primeros dos párrafos de esta sección, son sólo un tipo de los cuantizadores vectoriales considerados en su sentido más amplio. A este tipo se le denomina Voronoi[10] y se caracteriza porque la partición de $\Re^k$ queda completamente determinada por el diccionario y una medida de distorsión. Comúnmente cuando se habla de cuantización vectorial se está refiriendo a este tipo sin mencionarlo. Un VQ es de la clase Voronoi si su i-ésima clase se define de acuerdo con la Ec. 2.56

La Fig. 2.19 muestra el esquema para el codificador de un VQ de la clase Voronoi. El bloque *localizador NNV* localiza en el diccionario el vector código más cercano al de entrada y entrega el índice correspondiente.

2.5.2 Medidas de distorsión

La función empleada como medida de la distorsión es un aspecto importante de los VQ. Existe un gran número de funciones apropiadas para medir la distorsión. La mayoría de las que son

[10]También se le conoce como NNV por las siglas del inglés Nearest Neighbor Vector (vector vecino más cercano)

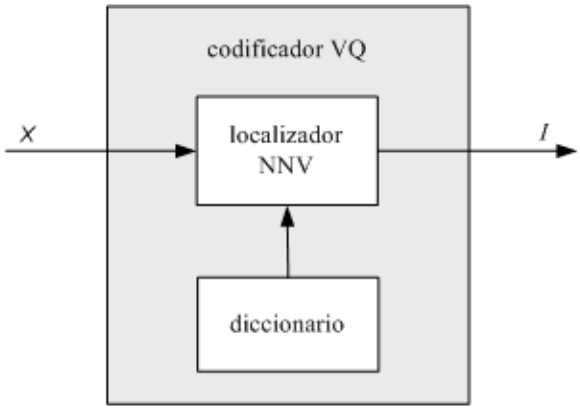


Fig. 2.19: Esquema del codificador para un VQ de la clase Voronoi

de interés para la VQ son aquéllas en las que la distorsión vectorial se obtiene como la suma de las distorsiones escalares, tal como se expresa a continuación:

$$d_v(X, \hat{X}) = \sum_{i=1}^{k} d_s(x_i - \hat{x}_i), \tag{2.57}$$

donde d_s es la distorsión escalar.

Dentro de este grupo, son de interés particular aquéllas en las que la distorsión escalar se calcula como el valor absoluto de la diferencia escalar elevado a una potencia entera positiva, es decir:

$$d_s(x_i - \hat{x}_i) = |x_i - \hat{x}_i|^n \,. \tag{2.58}$$

Cuando $n = 2$ se tiene el caso del *cuadrado del error* o *distancia euclidiana cuadrada*, siendo ésta la función más ampliamente usada y estudiada. Matemáticamente la distancia euclidiana cuadrada se expresa así:

$$\begin{aligned} d(X, \hat{X}) &= \left\| X - \hat{X} \right\|^2 \\ &= (X - \hat{X})^T (X - \hat{X}) \\ &= \sum_{i=1}^{k} (x_i - \hat{x}_i)^2 \end{aligned} \,. \tag{2.59}$$

La distancia euclidiana cuadrada es importante por dos razones. La primera de ellas reside en el hecho de que los análisis matemáticos que dan lugar a las condiciones que debe satisfacer un VQ para que sea óptimo están desarrollados en base a tal medida de distorsión. La segunda es su simplicidad de cálculo.

Se tiene una variante en la que se introduce una matriz de ponderación, con lo cual se realza la influencia de algunos componentes de los vectores o bien se les da preponderancia a algunos vectores código respecto a los restantes. La distancia euclidiana cuadrada ponderada queda expresada por la Ec. 2.60, en la cual M es la matriz de ponderación.

$$d(X, \hat{X}) = (X - \hat{X})^T M (X - \hat{X}) \tag{2.60}$$

La distancia euclidiana cuadrada es computacionalmente simple, sin embargo para efectos de su implementación hardware es costosa debido a los multiplicadores requeridos para obtener los cuadrados de las diferencias. Esto es de particular importancia cuando el localizador del NNV opera en paralelo, es decir, cuando obtiene simultáneamente la distancia entre un vector de entrada y todos los vectores del diccionario, ya que en tal caso son necesarios N multiplicadores. Como alternativa se tiene una medida de distorsión que no requiere multiplicadores, ésta es la *distancia Manhattan* o *distancia taxicab* definida en la geometría no euclidiana desarrollada por Minkowsky hacia fines del siglo XIX. La distancia Manhattan es el caso especial de la familia descrita por la Ec. 2.58 cuando $n = 1$ y, por tanto, queda expresada por:

$$\begin{aligned} d(X, \hat{X}) &= \left\| X - \hat{X} \right\|^1 \\ &= \sum_{i=1}^{k} |x_i - \hat{x}_i| \end{aligned} . \tag{2.61}$$

La distancia Manhattan tiene ciertas propiedades que se pueden explotar para reducir los pasos requeridos para la búsqueda exhaustiva en el diccionario del vector más cercano. Una de estas propiedades establece que $\sum |x_i - y_{ij}| \geq |a - b_j|$, siendo $a = \sum x_i$ y $b_j = \sum y_{ij}$. Entonces los valores de b_j para $j = 1, 2, \ldots N$ son las N constantes que se obtienen sumando los k elementos de cada vector código, de modo que se pueden calcular fuera de línea y almacenar como una extensión del diccionario. Cuando se procese algún vector de entrada X para buscar su vector código más cercano, primero se calcula a y posteriormente cuando se compara con algún vector de código Y_j se calcula $|a - b_j|$ y si el valor obtenido es mayor que la distancia mínima obtenida hasta el momento, entonces Y_j se descarta como el posible vector más cercano. En [24] se hace uso de esta propiedad para reducir en más de un 40 por ciento el costo computacional, al método los autores le denominan *eliminación de cálculos innecesarios*.

2.5.3 Generación del diccionario

El diseño de un VQ del tipo Voronoi se puede dividir en dos partes. Una tiene como propósito generar su diccionario y la otra delinear un esquema que permita cuantizar lo más rápidamente posible los vectores de entrada. Para la generación del diccionario el punto de partida es un conjunto de vectores representativo de la clase sobre la cual está destinado a trabajar el VQ, al que se denomina *conjunto de entrenamiento*.

Generalidades

Para un conjunto de entrenamiento dado, el diseño de un diccionario óptimo es un problema de optimización cuya función objetivo es la minimización de la distorsión entre los vectores del conjunto de entrenamiento X y el diccionario C dada por:

$$D(C, X) = \frac{1}{kp} \sum_{i=1}^{p} \sum_{j=1}^{k} (X_j^i - Y_j^{C(X^i)})^2, \tag{2.62}$$

donde p es el número de vectores en el conjunto de entrenamiento, k es el tamaño de los vectores, X^i, $i = 1, 2, \ldots p$, son los vectores de entrenamiento y $Y^{C(X^i)}$ es el vector del diccionario al que queda asociado X^i.

Los métodos para la generación del diccionario se pueden agrupar en dos grandes clases: los determinísticos y los estocásticos.

Dentro de los determinísticos, está el algoritmo generalizado de Lloyd y sus variantes, que han sido los más ampliamente utilizados, principalmente por la simplicidad para su implementación software y por su rápida convergencia, esto a pesar de su deficiencia que consiste en que para un diccionario inicial dado, la solución converge al valor mínimo local más cercano de la función objetivo. En otras palabras, los algoritmos determinísticos no tienen la capacidad para localizar la solución óptima global.

Los métodos estocásticos para generación de diccionarios han tenido un desarrollo más reciente con el propósito de obtener una aproximación de la solución óptima global en un número finito de pasos. Entre ellos están los algoritmos basados en *annealing simulada*, los evolutivos o genéticos [25] y los basados en la búsqueda *tabu*, destacando este último tipo por la atención que ha observado recientemente [26] [27].

Mención aparte, por su interés para esta tesis, merecen los métodos para generación de diccionario que se basan en redes neuronales. Dentro de éstos destacan las redes competitivas que serán tratadas en el siguiente capítulo, de manera particular las redes SOM.

A la fecha el algoritmo generalizado de Lloyd, particularmente una de sus variantes, el LBG, continúa siendo la referencia de comparación más comúnmente empleada. Además, se utiliza como parte de los algoritmos estocásticos, principalmente para la sintonización final del diccionario una vez que la solución al problema de optimización se ubica en la vecindad del óptimo global. Por todo esto, a continuación se describe brevemente.

El algoritmo generalizado de Lloyd (GLA) y el algoritmo LBG

El GLA debe su nombre a que el algoritmo es la generalización al caso vectorial del procedimiento para cuantización escalar propuesto por Lloyd en [28].[11] Linde, Buzo y Gray lo formalizaron y desarrollaron una variante a la que se le denomina, por las iniciales de los autores, *algoritmo LBG* [29].

Todo el proceso del GLA se lleva a cabo utilizando un conjunto de vectores que sea representativo de la clase de vectores sobre los cuales el VQ esté destinado a operar, al cual se le conoce como *conjunto de entrenamiento*. El GLA es iterativo y cada paso se divide en dos fases. En la primera, partiendo del diccionario obtenido en la iteración anterior, se busca la partición óptima del espacio k-dimensional, de modo que cada vector del conjunto de entrenamiento pertenezca a la región asociada al vector código más cercano. En la segunda, habiéndose obtenido dicha partición, se calculará el centroide para cada región, generando así los nuevos vectores código. El GLA se basa en la distancia euclidiana cuadrada como medida de distorsión. El proceso se ejecuta iterativamente hasta que se satisfaga cierto criterio de terminación.

En la Tabla 2.2 se muestra la fase del algoritmo que genera la partición en N clases del conjunto de entrenamiento constituido por M vectores del espacio k-dimensional. En el lazo interno se obtiene el índice ($indx$) del vector código más cercano al vector de entrenamiento que se pretende ubicar en alguna clase; el valor finalmente obtenido para el índice se almacena en la posición correspondiente a ese vector de entrenamiento en el array p. El número de multiplicaciones requeridas para llevar a cabo la partición es igual a M x N x k y del mismo orden son la cantidad de sumas y restas necesarias.

[11] El algoritmo se dio a conocer en un congreso organizado por el Institute of Mathematical Statistics en septiembre de 1957. Posteriormente fue publicado en el número especial sobre cuantización que se cita

Paso	Operación
1	Para cada X_i, $\quad i = 1, 2, \ldots, M$
2	$indx \leftarrow 0, \quad d_{min} \leftarrow 10^8$
3	Para cada Y_j, $\quad j = 1, 2, \ldots, N$
4	$d = d(X_i, Y_j)$
5	Si $d < d_{min}, \quad indx \leftarrow j,\ d_{min} \leftarrow d$
6	siguiente j
7	$p_i \leftarrow indx$ (ubicar X_i en la clase $indx$)
8	siguiente i

Tabla 2.2: Fase del GLA para efectuar la partición

La Tabla 2.3 corresponde a la fase del algoritmo dedicada a obtener los centroides para cada clase. Esta fase requiere realizar M x k sumas y N x k divisiones.

Paso	Operación
1	Para cada clase, $\quad j = 1, 2, \ldots, N$
2	$Y_j \leftarrow [0]; \quad qty_j \leftarrow 0$
3	siguiente j
4	Para cada X_i, $\quad i = 1, 2, \ldots, M$
5	$indx \leftarrow p_i$
6	$qty_{indx} \leftarrow qty_{indx} + 1; \quad Y_{indx} \leftarrow Y_{indx} + X_i$
7	siguiente i
8	Para cada clase, $\quad j = 1, 2, \ldots, N$
9	$Y_j \leftarrow Y_j / qty_j$
10	siguiente j

Tabla 2.3: Fase del GLA para el cálculo de centroides

Generalmente el criterio de finalización del proceso iterativo es la variación relativa de la distorsión media entre dos pasos consecutivos. A medida que el proceso de entrenamiento avanza, la distorsión converge hacia un valor asintótico. El proceso termina cuando la variación relativa se considera suficientemente pequeña. La distorsión media obtenida después de la iteración p se calcula mediante la Ec. 2.63, siendo Q^p el VQ que se ha generado hasta esa iteración y $\hat{X}_i^p$ el vector obtenido al aplicar tal VQ sobre X_i.

$$\begin{aligned} d_m^p &= \frac{1}{M} \sum_{i=1}^{M} d(X_i, Q^p(X_i)) \\ &= \frac{1}{M} \sum_{i=1}^{M} d(X_i, \hat{X}_i^p) \end{aligned} \tag{2.63}$$

La variación relativa de la distorsión media después del paso p queda entonces expresada por:

$$d_{rel}^{p} = \frac{\left| d_m^p \; - \; d_m^{p-1} \right|}{d_m^p}. \tag{2.64}$$

Un tema importante para la generación del diccionario mediante un algoritmo iterativo como lo es el GLA, es el del diccionario inicial. El algoritmo original propone partir de un diccionario inicial generado aleatoriamente. Sin embargo, de ese modo existe la probabilidad de que algún vector de ese diccionario inicial no sea el más cercano para todos los vectores del diccionario y, como consecuencia, la región correspondiente quedará vacía. Inclusive podría darse el caso de que eso sucediera para varios vectores y entonces se tendrían varias regiones vacías. Se han desarrollado ciertas técnicas con el objetivo de generar un diccionario inicial apropiado.

El algoritmo LBG resuelve el problema del diccionario inicial de otra manera. El LBG parte de un diccionario inicial con sólo dos vectores código. Estos dos vectores se obtienen por bipartición de uno solo que es igual al centroide de todos los vectores del conjunto de entrenamiento. La bipartición se realiza aplicando una ligera perturbación aleatoria a todos los componentes del centroide en dos ocasiones, una para cada vector código producido por la bipartición. Con ese pequeño diccionario inicial de dos vectores se aplica el GLA iterativamente como se describió anteriormente. Cuando se satisface el criterio de terminación, se efectúa nuevamente la bipartición para obtener así un nuevo diccionario inicial, ahora con 4 vectores. El ciclo bipartición-entrenamiento se repite hasta que el diccionario con el número predeterminado de vectores haya sido sometido al entrenamiento.

2.5.4 Compresión de imágenes mediante VQ

La aplicación de la cuantización vectorial para la compresión de imágenes se puede realizar de varias maneras, cualquiera que sea el esquema utilizado, el proceso que se sigue para comprimir una imagen por medio de un VQ necesariamente requiere como primer paso descomponer la imagen en bloques no solapados de k pixeles. Generalmente se usan bloques cuadrados, de manera que $k = l^2$, siendo l el número de pixeles por lado.

Así, cada bloque de pixeles da lugar a un vector de dimensión k y entonces la representación espacial de la imagen pasa a ser una secuencia de vectores de dimensión k. Cada vector se codifica mediante el VQ y como resultado se obtiene un índice con $log_2 N$ bits, donde N es el tamaño del diccionario.

De este modo, la razón de compresión se calcula mediante la Ec. 2.65, es decir dividiendo la cantidad de bits necesarios para representar un bloque entre el número de bits requeridos para expresar el valor del índice.

$$CR = \frac{8k}{log_2 N} \tag{2.65}$$

2.5.5 Técnicas especiales para VQ

VQ por búsqueda en árbol (TSVQ)

Se trata de un algoritmo que define el diccionario mediante una estructura en árbol. En la Fig. 2.20 se ilustra la manera como opera un TVSQ para el caso de un árbol binario,

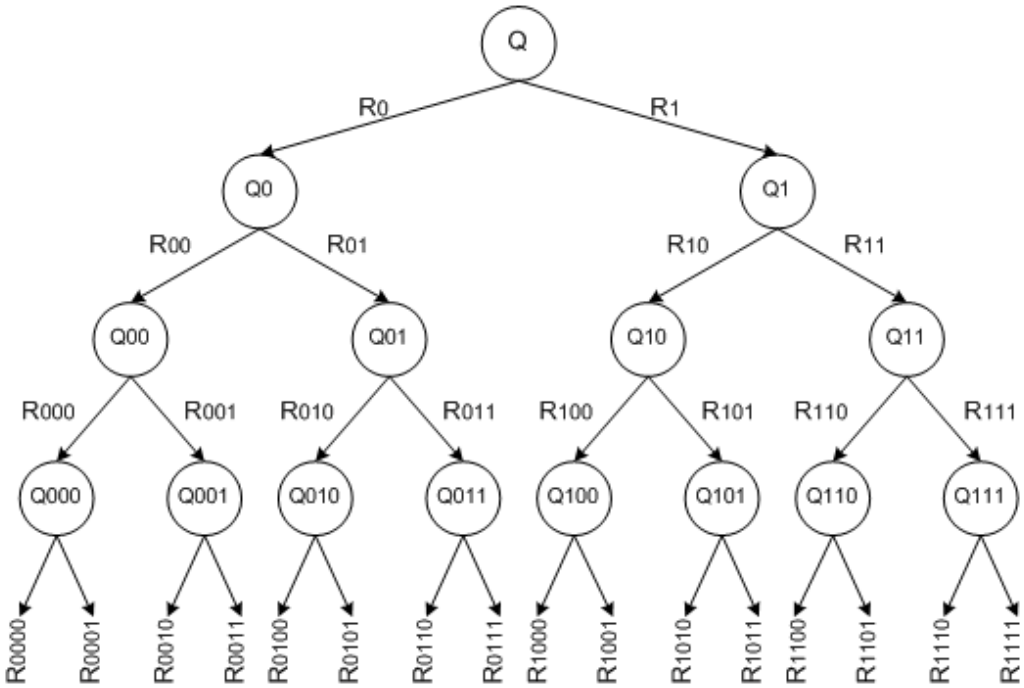


Fig. 2.20: Arbol de particiones y cuantizadores para un TVQ

que es el más sencillo. El cuantizador, globalmente visto, está formado por un conjunto de cuantizadores, cada uno con tamaño igual a dos. Cada cuantizador individual corresponde a un nodo del árbol. Sobre cada nodo incide una rama y emergen dos; la rama incidente representa una región del espacio k-dimensional y las que emergen representan las subregiones que resultan de la partición originada por ese cuantizador individual. El nodo raíz corresponde a la cuantización más alta, indicada como Q y opera sobre el espacio vectorial en su totalidad, particionándolo en dos regiones, R_0 y R_1. En el segundo nivel, Q_0 opera sobre la región R_0 y la particiona en R_{00} y R_{01}, mientras que Q_1 hace lo propio para particionar R_1 en R_{10} y R_{11} y así sucesivamente. El árbol de la Fig. 2.20 tiene cuatro niveles, de manera que los ocho cuantizadores representados por los nodos hoja dan lugar a las 16 subregiones representadas por las 16 ramas que emergen de esos nodos.

En este caso, la codificación de un vector de entrada consiste en aplicar sobre ese vector un cuantizador de cada nivel, seleccionándolo de acuerdo al índice obtenido en el nivel inmediato superior. El índice para cada cuantizador individual sólo puede ser cero o uno. Si en el nivel l se aplica el cuantizador $Q_{ii...i}$ y se produce como índice un cero, entonces en el nivel $l+1$ se aplicará el cuantizador $Q_{ii...i0}$, de otro modo será el $Q_{ii...i1}$, es decir, el cuantizador de la izquierda si el índice fue cero o el de la derecha en caso contrario.

De este modo, la cuantización de un vector mediante su búsqueda a través del árbol binario de cuatro niveles requerirá aplicar cuatro cuantizadores de tamaño 2, en lugar de aplicar uno sólo de tamaño 16. Así, al cuantizar un vector, el número de multiplicaciones requeridas para calcular la distorsión será igual a $8k$ en lugar de $16k$.

Para generar los diccionarios se utiliza, por ejemplo, el algoritmo LBG. Primero se aplica para generar el diccionario del cuantizador Q, para esto se emplea el conjunto de entrenamiento en su totalidad. Como resultado, además del diccionario, el conjunto de entre-

namiento queda dividido en dos subconjuntos. A continuación, para generar los diccionarios Q_0 y Q_1 se aplica nuevamente el algoritmo en dos ocasiones, en una se utiliza como conjunto de entrenamiento uno de los subconjuntos generados en el primer nivel y en la otra ocasión el subconjunto restante. En referencia a la Fig. 2.20, para generar el diccionario de Q_0 se utiliza R_0 como conjunto de entrenamiento y para el de Q_1 se emplea R_1. Este proceso se repite avanzando hacia abajo hasta generar los diccionarios de los nodos hoja.

El árbol no necesariamente tiene que ser binario como el de la Fig. 2.20. Tampoco tiene que ser homogéneo e inclusive puede estar desbalanceado. Obviamente todo es más simple si se elige una estructura homogénea. Por ejemplo, un árbol homogéneo de n niveles con diccionarios de tamaño m hará las veces de un solo diccionario de tamaño m^n. La implementación hardware de un VQ se simplifica si el árbol es binario y en tal caso se le conoce como BTSVQ.

VQ clasificada

La cuantización vectorial clasificada (CVQ) surge como repuesta a la importancia de los bordes contenidos en las imágenes. Como bordes se entienden las zonas de una imagen en las que se produce un cambio brusco en el nivel de luminancia, marcando el contorno de algún objeto. Los bordes, aunque ocupan espacios reducidos dentro de la imagen, requieren una buena reproducción con la finalidad de satisfacer los criterios de evaluación basados en la percepción visual. En los algoritmos básicos de cuantización vectorial es muy pequeño el número de regiones producidas por los vectores de borde, debido a su escasa frecuencia de aparición. Como consecuencia, la cuantización de un vector de borde puede dar como resultado un vector código que no se ajusta bien a él. Para solucionar este problema surge la clasificación.

Para generar el diccionario, los vectores de entrenamiento se dividen en dos tipos, borde y textura, y se generan diccionarios distintos para cada tipo. Los vectores de entrada se clasifican también para codificarlos con el diccionario correspondiente a su tipo. Este concepto se puede ampliar aumentando las posibilidades de la clasificación a más de dos tipos; por ejemplo, algunas técnicas de clasificación se basan en la orientación geométrica de los bordes. Para la clasificación se utiliza algún algoritmo de detección de bordes.

VQ multietapas

La MSVQ[12] divide la tarea de codificación en etapas. Cada etapa realiza una cuantización burda, es decir empleando un diccionario pequeño. La cuantización de la primera etapa se aplica sobre el vector de entrada. A continuación, una segunda etapa opera sobre el vector de error, es decir sobre la diferencia entre el vector de entrada y el producido por la decodificación en la primera etapa. Posteriormente se emplea una tercera etapa de cuantización para cuantizar el error de la segunda etapa y así sucesivamente. El índice para la cuantización global se formará concatenando los índices producidos en cada etapa.

El diccionario de la primera etapa se genera utilizando el conjunto de entrenamiento que consta de M vectores k-dimensionales. Para generar el diccionario de las etapas posteriores se sigue el procedimiento recursivo siguiente. Si en la etapa p se utiliza T^p como conjunto de entrenamiento para producir el cuantizador Q^p, entonces el primer paso para construir el cuantizador de la etapa $p+1$ consiste en obtener los vectores del conjunto de entrenamiento

[12]MSVQ es la sigla de multi-stage vector quantization

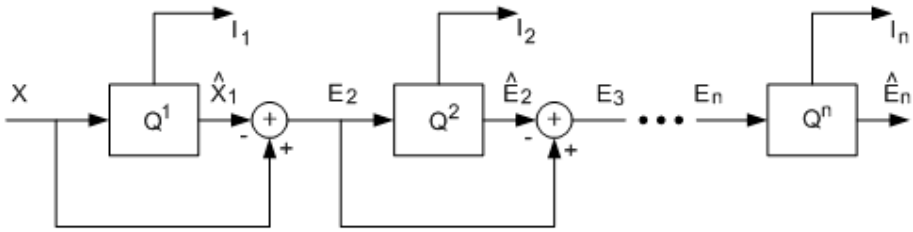


Fig. 2.21: Codificador para un MSVQ

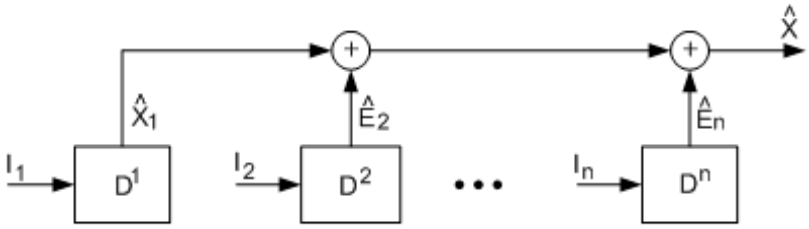


Fig. 2.22: Decodificador para un MSVQ

mediante $T_i^{p+1} = T_i^p - Q^p(T_i^p)$, para $i = 1, 2, \ldots, M$. El segundo paso es aplicar el algoritmo de generación seleccionado (por ejemplo el LBG) para producir Q^{p+1}.

La Fig. 2.21 ilustra la manera como se codifica un vector mediante la MSVQ. La Fig. 2.22 muestra como se lleva a cabo la decodificación.

VQ jerárquica (HVQ)

La cuantización vectorial jerárquica es una técnica apropiada para aplicaciones en las que se tengan señales con pocos cambios durante intervalos prolongados. Por ejemplo, en el caso de las imágenes esto significa la presencia de zonas extensas con cambios muy suaves, o sea un gran número de pixeles con muy alta correlación. Surge entonces la tentación de utilizar un cuantizador con dimensión variable, de modo que para los intervalos prolongados sin cambios bruscos se aplique una cuantizador de dimensión grande y para los intervalos con cambios bruscos uno de dimensión pequeña. Sin embargo, eso no es simple y, por lo tanto, tampoco lo más apropiado. Una alternativa es la técnica HVQ.

El punto de partida de la HVQ consiste en extraer del vector a cuantizar, al que se le denomina en el ámbito de esta técnica como *supervector*, un *vector de rasgos* con menor dimensión, que describe ciertos atributos del supervector que toman en cuenta la correlación entre sus componentes.

De manera general, si X de dimensión $k = p \cdot r$ es el supervector, el primer paso del procedimiento para aplicar esta técnica consiste en descomponerlo en r subvectores S_i de tamaño p. A continuación se aplica a cada subvector S_i el extractor de rasgos para obtener el valor escalar respectivo f_i. Con los valores de f_i se forma el vector de rasgos F. Se aplica entonces sobre F una cuantización vectorial para obtener el vector de rasgos cuantizado $\hat{F}$. El vector de rasgos cuantizado se combina con el supervector original, en una operación a la que se le llama reducción, para generar el *supervector reducido* Y. Finalmente el vector reducido se parte en sus subvectores Y_i. Con esto concluye la primera etapa de la técnica HVQ. Si el

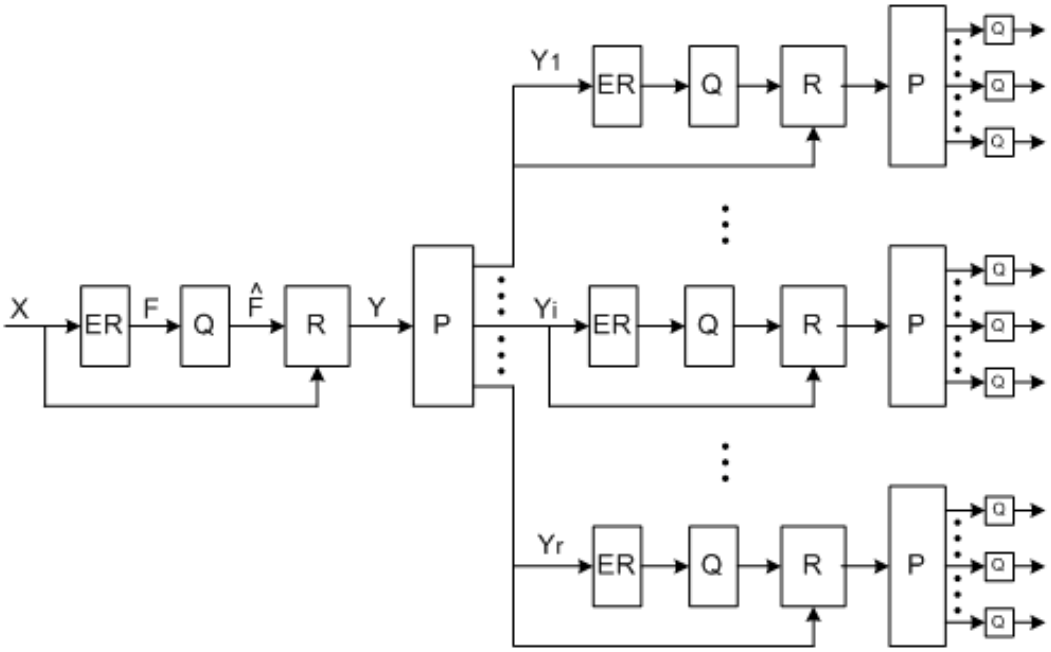


Fig. 2.23: Estructura en árbol para un HVQ de dos etapas

HVQ es de una sola etapa, cada uno de los Y_i se cuantiza vectorialmente por separado. La alternativa es aplicar a cada Y_i el mismo procedimiento en una segunda etapa, generando así una estructura en árbol. En la Fig. 2.23 se muestra un HVQ de dos etapas.

VQ predictiva (PVQ)

Al aplicar la VQ individualmente a cada vector de una secuencia de vectores formados a partir de la señal a codificar, se pierde la consideración de la dependencia que existe entre ellos. Se dice que el VQ carece de memoria. Una forma de tomar en cuenta la dependencia entre vectores es extender al caso vectorial la técnica de predicción escalar. La Fig. 2.24 muestra el esquema para un cuantizador vectorial predictivo, incluyendo el codificador en el extremo de la fuente y el decodificador en el del destino.

Si el predictor es de orden m, cuando se codifica X_n estarán almacenados en la memoria los vectores aproximados $\hat{X}_{n-1}, \hat{X}_{n-2}, \ldots \hat{X}_{n-m}$. En función de esos vectores se obtiene $\tilde{X}_n$ que es la predicción de X_n. La diferencia entre X_n y $\tilde{X}_n$ será el error de predicción vectorial, E. A este error se le aplica la VQ para obtener el índice y el error de predicción cuantizado $\hat{E}$. La suma de la predicción y el error de predicción produce el vector aproximado, el cual se almacena en la memoria para emplearse al momento de codificar los próximos vectores de la secuencia de entrada.

En el destino, a partir del índice se obtiene el error de predicción y éste se suma con el vector de predicción obtenido por el predictor en función de los m vectores aproximados previos.

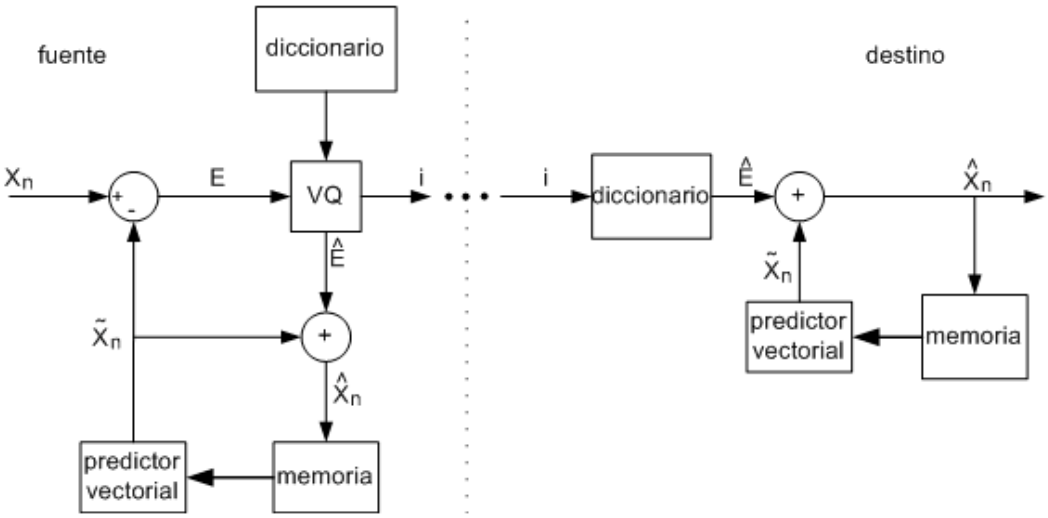


Fig. 2.24: Cuantizador vectorial predictivo

2.5.6 Implementación hardware de VQ

La cuantización vectorial VQ ha sido un campo intensivo de investigación a lo largo del último cuarto de siglo. Durante ese tiempo el interés principal se ha centrado en los VQ de tipo Voronoi. Para esta clase, los trabajos de investigación se enfocaron inicialmente en el desarrollo de soluciones software para los principales problemas encontrados al generar diccionarios que satisfacieran las condiciones de optimalidad por medio de algoritmos con complejidad computacional suficientemente baja para que fueran apropiados para los procesadores de propósito general existentes. Posteriormente los esfuerzos se han dedicado a mejorar los métodos con el fin de reducir la complejidad computacional sin deterioro del rendimiento en lo que se refiere a la medida de la distorsión introducida por la cuantización.

Por lo que respecta a la implementación hardware, los esfuerzos de la investigación se han orientado al diseño de arquitecturas simples para emplear los algoritmos más eficientes. Por ejemplo, como ya se mencionó antes, para evitar la complejidad computacional de la distancia euclidiana cuadrada, ésta se ha reemplazado en la mayoría de las implementaciones hardware de VQ con la distancia Manhattan. El efecto de esta simplificación sobre el rendimiento de la VQ ha sido estudiada con profundidad y se ha llegado a la conclusión de que para VQ de dimensión grande el efecto es despreciable [30]. Así, por ejemplo en [31] se presenta la implementación de una memoria asociativa basada en un VQ que utiliza distancia Manhattan.

Una solución alterna al problema de las multiplicaciones involucradas en el cálculo de la distancia euclidiana consiste en utilizar LUT's[13] para almacenar todos los posibles valores de $(x_i - y_{ij})^2$, siendo x_i el i-ésimo elemento del vector a cuantizar y y_{ij} el correspondiente elemento del j-ésimo vector código en el diccionario. Así, si los elementos se representan con n bits, la tabla con todos los posibles valores de $(x_i - y_{ij})^2$ tendrá 2^n elementos con $2n$ bits cada uno y se accede a ellos de manera directa utilizando $|x_i - y_{ij}|$ como índice [32].

La implementación hardware de VQ facilita esquemas de búsqueda en paralelo.[14] Cuando

[13] De look-up tables

[14] En la búsqueda secuencial el vector a cuantizar se va comparando con los del diccionario, uno a la vez; en la búsqueda en paralelo el vector de entrada se compara con varios de los vectores del diccionario simultáneamente.

la búsqueda es masivamente paralela, si el VQ es de tamaño N, una vez que se han obtenido las N distancias es necesario compararlas para determinar cuál es el vector código más cercano al de entrada. A esta etapa se le ha dado en llamar Winner Take All (WTA) y se ha convertido en un tópico de investigación. En [33] se expone un WTA basado en un filtro de paso mínimo con procesamiento de palabras en paralelo y bits en serie que en una sola etapa compara 64 distancias de 12 bits con una tasa de 1 bit por ciclo. Posteriormente, en [34] se presenta una versión mejorada con una tasa de 6 bits por ciclo.

Otros esfuerzos se han dedicado al diseño de procesadores basados en VQ para su empleo en sistemas de codificación de video y en aplicaciones de concordancia de patrones (pattern matching) en varios campos específicos. Son de particular importancia las soluciones basadas en redes neuronales que hacen uso de su paralelismo intrínseco para generar diccionarios en tiempos reducidos, destacando en este campo el papel que juegan las redes competitivas. La cuantización vectorial mediante redes neuronales se abordará en el siguiente capítulo.

A la búsqueda se le denomina masivamente paralela cuando la comparación se realiza con todos los vectores código a la vez

Capítulo 3

Redes neuronales para compresión de imágenes

El surgimiento de las redes neuronales como un campo muy vasto de investigación en las matemáticas aplicadas y en varias ramas de la ingeniería data de hace ya más de 60 años. Para el advenimiento de la era moderna en las redes neuronales se considera clave el trabajo conjunto del neurofisiólogo Warren McCulloch y el matemático Walter Pitts [35], en el cual abordaron el estudio de la manera como el cerebro puede producir patrones de gran complejidad a partir de multitud de células básicas interconectadas, es decir las neuronas. En el artículo propusieron el modelo neuronal conocido como MCPN, finalizando en el diseño e implementación de una red de neuronas usando circuitos eléctricos muy simples.

A lo largo de estos años, las redes neuronales artificiales (ANN) se han desarrollado con objetivos que se pueden ubicar en dos grandes grupos: los que tienen que ver con la investigación sobre la manera como funciona el cerebro humano y los que intentan diseñar sistemas que resuelvan problemas complejos con base en técnicas que emulan el comportamiento neuronal humano. La evolución de las ANN a lo largo del tiempo ha sufrido altibajos, marcados por la aparición de nuevas tecnologías computacionales y electrónicas.

En este capítulo se revisan las ANN bajo el enfoque de su aplicación para la compresión de imágenes. En la primera sección se exponen los aspectos fundamentales de las ANN; en la segunda se tratan las técnicas basadas en ANN para compresión de imágenes; en la tercera se estudia con mayor detenimiento el tipo de red neuronal más apropiado para la VQ y, por último, se examinan ciertos aspectos relevantes para la implementación hardware de las ANN.

3.1 Introducción

Una ANN es un intento de emular, de manera simplificada, el funcionamiento de las redes neuronales biológicas del cerebro humano. Se dice que es un intento, entre otras razones, porque aún está lejana la comprensión completa de la funcionalidad del cerebro. Conviene entonces tener una idea general de la caracterización de una red neuronal biológica.

3.1.1 Redes neuronales biológicas (BNN)

El sistema nervioso central está compuesto completamente por dos tipos de células: las *neuronas* y las *células gliales*. Las neuronas son las estructuras básicas de procesamiento y las

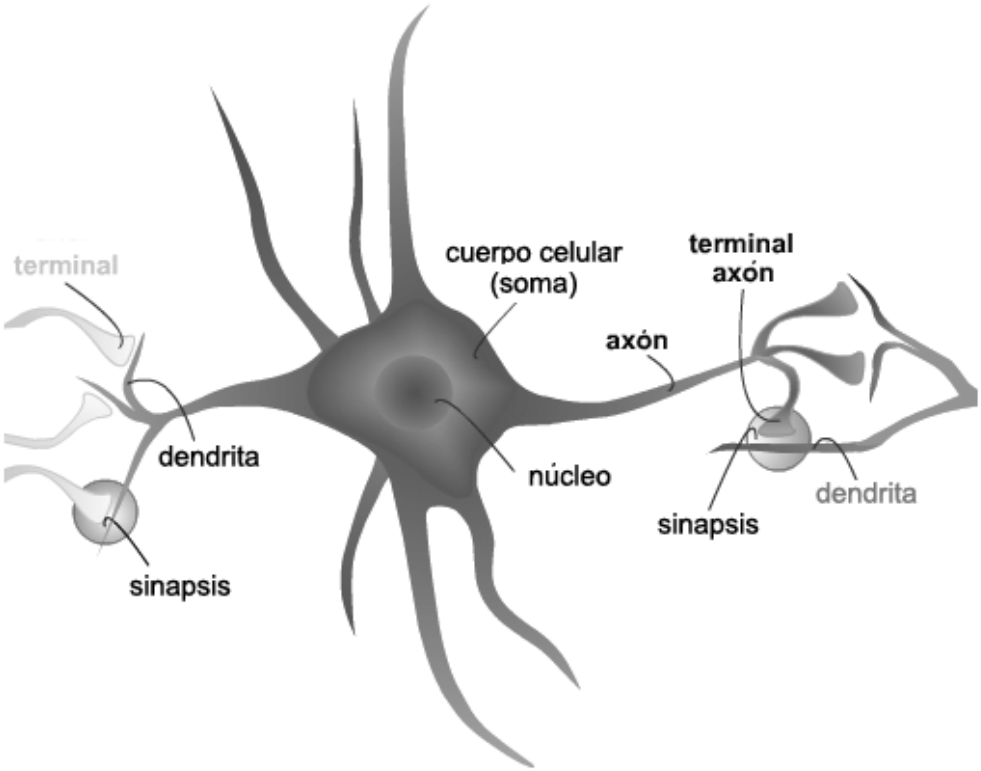


Fig. 3.1: La neurona biológica

células gliales proporcionan a las neuronas soporte y nutrición, además de que participan en la transmisión de las señales a través del sistema nervioso. La complejidad del procesamiento realizado por las neuronas en el cerebro humano es tal que requiere una diversidad de neuronas, se estima que existen 10 mil tipos. En cuanto a la cantidad de neuronas, se calcula que sólo en el cerebro su número es del orden de 200 billones y que cada neurona está conectada con otras en un número que va de 5 mil a 200 mil. El número de células gliales es entre diez y 50 veces el número de neuronas.

La función de las neuronas es recibir información de entrada proveniente de otras neuronas, procesarla y finalmente enviar esa información como salida hacia otras neuronas. Por lo tanto, las neuronas procesan toda la información que fluye hacia, desde o en el interior del sistema nervioso central.

En la Fig. 3.1 se puede apreciar que la neurona está compuesta por su *núcleo celular* al que también se le denomina *soma*, varias ramificaciones de poca longitud a las que se les llama *dendritas* y funcionan como conexiones de entrada, y *los axones* que son ramificaciones de mayor longitud[1] y sirven como conexiones de salida.

La información recorre las neuronas en forma de impulsos eléctricos generados por fenómenos iónicos. Hasta tal punto es importante el proceso, que una gran parte del diseño de la membrana celular parece destinado a ello, en exclusiva. La célula mantiene su fluido intracelular a un potencial de unos 80 mV por debajo del fluido extracelular mediante elaborados mecanismos de flujo de iones (sodio, potasio y cloro) a través de la membrana. Ante un

[1]Un axón alcanza a medir más de un metro

estímulo eléctrico en un área cercana, los mecanismos reguladores dan lugar a una reducción de la diferencia de potencial entre ambos lados de la membrana (*despolarización*), que se realimenta positivamente cuando se supera cierto umbral.

Una vez que se alcanza la situación en la que el estímulo excitador (*potencial de acción*) ha sido regenerado y puede provocarse la reacción similar en zonas vecinas, los mecanismos reguladores vuelven a conseguir el estado de reposo. Así se propaga la información de una región a otra. También existen *estímulos de inhibición*, que actúan para mantener el estado de reposo.

Debido a la base iónica del proceso, existen lapsos durante los cuales los estímulos no tienen respuesta (*período refractario*), lo que limita la frecuencia de transmisión a unos 200 potenciales de acción por segundo. Este es el mecanismo que conduce por el axón los estímulos recibidos en las dendritas después de haber sido procesados en el núcleo. La necesidad de regenerar los impulsos cada pocos milímetros en los Nodos de Ranvier de los axones, limita la velocidad de transmisión a unos 100 metros por segundo.

Lo más complicado es lograr que los impulsos pasen de una célula a otra a través del medio extracelular que las separa. Se necesitan estructuras especializadas para salvar las ínfimas distancias entre neuronas. Estas son las *sinapsis*, que utilizan moléculas viajeras o *neurotransmisores* que se liberan al llegar los potenciales de acción, y circulan desde la *célula presináptica*, o emisora, hasta la *postsináptica*, o receptora.

La membrana postsináptica se puede volver más permeable a los iones positivos que a los negativos y, entonces, se puede inducir un estímulo excitador o inhibidor. El efecto de cada sinapsis se integra con el de las restantes en el cuerpo de la célula receptora, obedeciendo a reglas tan sutiles como las de clasificación y si se alcanza el umbral necesario se producirá un potencial de acción. Una mayor intensidad del conjunto de estímulos recibidos produce mayor cantidad de impulsos de salida, hasta un máximo (*saturación*), teniéndose una relación no lineal entre ambos.

El comportamiento ante diversos estímulos puede variar de forma muy diversa a lo largo del tiempo, esto se logra mediante cambios en el modo de combinar los estímulos recibidos en cada neurona. Estos cambios (*aprendizaje*) son objeto de intensos estudios, pues dictan el comportamiento de extensos conjuntos de neuronas. Esto tiene especial interés en la investigación de los sistemas neuronales artificiales.

Desde luego, la neurofisiología es mucho más compleja, y serán necesarios muchos avances en ella para entender mejor el funcionamiento del sistema nervioso y desarrollar modelos más sofisticados, incluyendo, por ejemplo, la realimentación en las propias sinapsis, para construir sistemas neuronales más avanzados. La investigación sobre redes neuronales artificiales posibilita descartar o desarrollar hipótesis sobre el sistema nervioso y de esta forma ambos campos de estudio se complementan.

3.1.2 Redes neuronales artificiales (ANN)

A partir del conocimiento acerca de la funcionalidad de las redes neuronales biológicas sintetizado en el punto anterior, se pueden destacar los siguientes aspectos que son clave para los modelos de ANN:

- Una neurona recibe estímulos provenientes de muchas otras.
- La permeabilidad de la membrana posináptica atenúa o fortalece los estímulos.

- Los estímulos se combinan de cierta manera en el núcleo neuronal.
- Una intensidad suficiente de la combinación de estímulos provoca que la neurona emita una salida.
- La salida de una neurona particular puede ir a varias neuronas.

Adicionalmente se generan las siguientes conclusiones:

- El procesamiento de información es local.
- La memoria es distribuida:
  - A largo plazo reside en la permeabilidad de las sinapsis.
  - A corto plazo corresponde a las señales enviadas por las neuronas.
- La permeabilidad de la conexión sináptica se puede modificar por la experiencia.
- Los neurotransmisores pueden ser excitadores o inhibidores.
- Los sistemas neuronales biológicos son tolerantes a fallos en dos aspectos:
  - Son capaces de reconocer estímulos diferentes a los procesados anteriormente.
  - El número de neuronas decrece a lo largo de la vida, pero las facultades cerebrales no se ven por ello mermadas; incluso, otras partes del cerebro logran asumir, en buena parte, las funciones que realizaban áreas dañadas por lesiones cerebrales.

De todo ello se desprende que las ANN deben ser capaces de: a) llevar a cabo un procesamiento masivamente paralelo;[2] b) trabajar con memoria distribuida; c) aprender en base a su experiencia, y d) soportar una muy alta conectividad.

El modelo de McCulloch-Pitts.

La neurona artificial más simple es la propuesta en 1943 por McCulloch y Pitts. En su modelo, Fig. 3.2, cada neurona recibe como entrada señales binarias a través de ramas sin peso. Estas ramas son de dos tipos, de excitación y de inhibición. La neurona de McCullloch-Pitts ajusta su salida de la manera siguiente: si recibe un *uno* por al menos una rama de inhibición, la salida de la neurona es *cero* (*estado inhibido*); de otro modo, si la suma total de las señales a través de las ramas de excitación es mayor o igual que cierto umbral θ, la neurona se dispara y entrega como salida un *uno* (*estado excitado*), en el caso contrario ($\sum x_i < \theta$), la salida es *cero*. Con este esquema particular, la funcionalidad del modelo neuronal de McCulloch-Pitts se obtiene definiendo un número apropiado de ramas de excitación y de inhibición y asignando un valor pertinente para el umbral.

Aunque inicialmente se destacó el hecho de que la red de McCulloch-Pitts puede calcular las funciones lógicas, el modelo tiene muchas limitaciones. Primero, debido al número restringido de parámetros libres en la red, es poco flexible para aplicarse a problemas diferentes. Segundo, el aprendizaje se obtiene cambiando las conexiones entre las unidades y, obviamente, el desconectar y conectar físicamente las ramas es más problemático que el cambio numérico de los pesos de las ramas.

[2]Una red está conformada por elementos de procesamiento muy simples (PE), cada uno de los cuales desempeña el trabajo específico de una neurona, que procesa de manera independiente la información que recibe

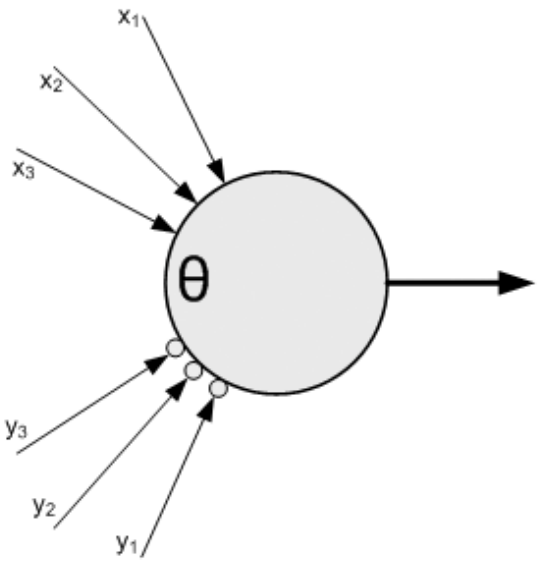


Fig. 3.2: Modelo de McCulloch-Pitts

El perceptrón

En 1949 D. Hebb [36] postuló un principio fundamental para el proceso de aprendizaje a nivel celular:

> Si la *neurona A* es estimulada repetidamente por la *neurona B* en momentos en que la *neurona A* está activa, entonces la *neurona A* se volverá más sensible a los estímulos provenientes de la *neurona B*.

Este enunciado tan simple trajo profundas consecuencias, pues implicó modificar la parte del modelo relacionada con la sinapsis, lo cual condujo a la incorporación de lo que se denominó *pesos sinápticos ajustables* con la finalidad de intensificar o atenuar los efectos de las señales de entrada. Cuando los modelos neuronales acogen este principio se dice que poseen *aprendizaje hebbiano*.

Pretendiendo probar prácticamente el aprendizaje hebbiano, en la década de los cincuentas se desarrollaron varios aparatos con elementos mecánicos, eléctricos, electromecánicos y luminosos. Es así que en 1958 [37] se presenta el *perceptrón*, cuyo modelo se muestra en la Fig. 3.3.

En este modelo, las señales de entrada se someten a una ponderación antes de ser combinadas con las restantes. A los coeficientes de ponderación se les llama *pesos sinápticos*. De este modo, la combinación de las señales de entrada ponderadas, s, se obtiene mediante la Ec. 3.1, en la cual $X = (x_1, x_2, \ldots, x_N)^T$ y $W = (w_1, w_2, \ldots, w_N)^T$ son los vectores columna formados por las señales de entrada y los pesos sinápticos, respectivamente. El aprendizaje hebbiano se manifiesta al modificar el valor de los pesos como fruto de la experiencia de la neurona integrada en una red.

$$s = W^T X = \sum_{i=1}^{N} w_i x_i \tag{3.1}$$

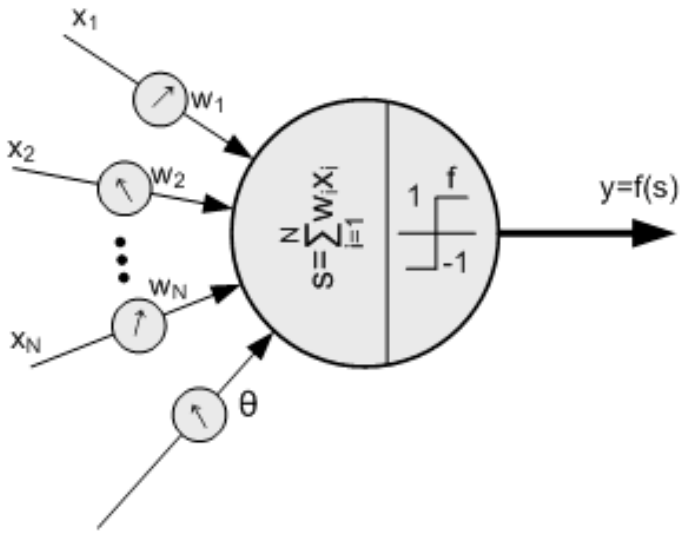


Fig. 3.3: Modelo del perceptrón

En cuanto a la salida, ésta está definida por la *función de activación* $y = f(s)$. Originalmente se estableció como una función de dos valores, presumiblemente correspondiendo al principio *todo-nada* de las neuronas biológicas. Así, si la combinación s es mayor que el umbral establecido θ, entonces la salida será 1, de otro modo será 0, o, de manera alternativa, la salida bipolar con valores 1 y -1, para los dos casos, respectivamente.

Posteriormente se han venido proponiendo nuevas funciones de activación, sobre todo funciones continuas diferenciables que permiten utilizar algoritmos de aprendizaje basados en la variación de la salida, como es el caso de los que se ubican en el tipo conocido como *del gradiente*. En la Fig. 3.4 se muestran las gráficas de la función de activación para la función de *umbral unipolar*, la función de *umbral bipolar*, la función *sigmoidal unipolar*[3] y la función *sigmoidal bipolar*. Para las cuatro funciones $\theta = 1$ y para las sigmoidales $\alpha = 4$.

En resumen, el modelo del perceptrón es el que se ha venido empleando para las ANN hasta la fecha, presentando como única diferencia respecto al original cierta diversidad para la función de activación. Se puede decir que las ANN han sido desarrolladas bajo los siguientes supuestos:

- El procesamiento de información ocurre en muchos elementos de procesamiento (PE) sencillos denominados neuronas.
- Las señales pasan entre las neuronas a través de unas conexiones que tienen asociado un peso, el cual, en una red neuronal típica, multiplica la señal transmitida.
- Los pesos se ajustan durante el proceso denominado aprendizaje.

[3]La función unipolar sigmoidal está definida como:

$$y = \frac{1}{1 + e^{-\alpha(s - w_0)}} \tag{3.2}$$

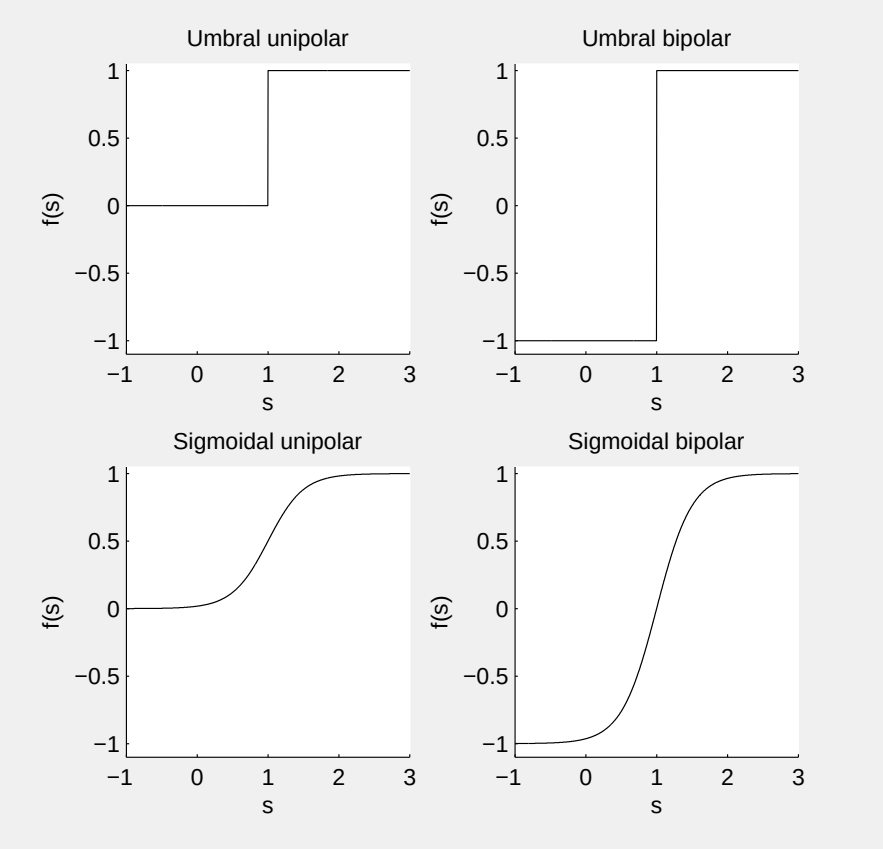


Fig. 3.4: Funciones de activación

- Cada neurona aplica una función de activación (usualmente no lineal) a la suma de las entradas ponderadas para determinar la salida.

3.1.3 Clasificación de las ANN

Un tema importante dentro de estos conceptos básicos sobre las ANN es la caracterización de una red neuronal, siendo tres los aspectos bajo los cuales puede realizarse tal caracterización:

- El patrón de interconexiones entre neuronas, es decir su arquitectura.
- El método para determinar los pesos en las conexiones, o sea el tipo de algoritmo para su entrenamiento o aprendizaje.
- Sus funciones de base y de activación.

Arquitectura de las ANN

Es práctica común visualizar la organización de las neuronas en capas. Los factores fundamentales para determinar el comportamiento de una neurona son su función de activación y el patrón de conexiones a través de las cuales recibe y envía señales. Las neuronas pertenecientes a una misma capa suelen tener el mismo patrón de interconexión y la misma función de activación; en otras palabras, normalmente las neuronas pertenecientes a una misma capa se

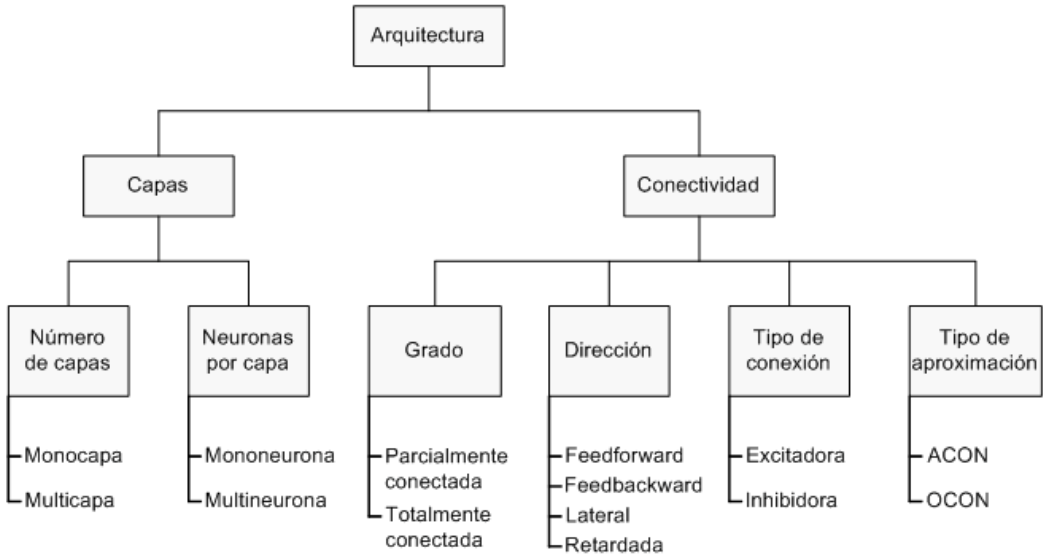


Fig. 3.5: Clasificación de las ANN de acuerdo a su arquitectura

comportan de igual manera. A la organización de las neuronas en capas y los patrones de interconexión entre capas se denomina *arquitectura de la red*.

De acuerdo a su arquitectura, se ha adoptado la clasificación de las ANN que se observa en la Fig. 3.5.

Las ANN, desde el punto de vista de su organización por capas, se clasifican en redes *monocapa* y redes *multicapa*. Para determinar el número de capas, las unidades de entrada no se consideran como una capa, ya que no realizan computación. Entonces, para contabilizar el número de capas, deben observarse los grupos de interconexiones entre grupos de neuronas. Este punto de vista es bastante interesante en tanto que realmente los pesos son los que contienen la información importante.

En la Fig. 3.6 se muestran dos redes, la de la izquierda es monocapa y la de la derecha multicapa, en este caso con dos capas. A x_1, x_2 y x_3 se les llama las *unidades de entrada*. En la red de la derecha, y_1 y y_2 conforman una capa a la que se le denomina *capa oculta*, mientras que z_1, z_2 y z_3 conforman la *capa de salida*.

Como se puede observar, una red multicapa es una red con una o más capas ocultas situadas entre las unidades de entrada y las neuronas de salida. Las redes multicapa pueden resolver problemas más complicados que las redes monocapa, a cambio, obviamente, de un entrenamiento mucho más costoso y complejo.

El tamaño de las redes depende del número de capas y del número de neuronas por capa. El número de unidades en las capas ocultas está directamente relacionado con la capacidad de la red para adecuarse a la aplicación específica para la cual fue diseñada. Para que el comportamiento de la red sea mejor, se debe determinar apropiadamente el número de neuronas de la capa oculta, esto es particularmente importante por lo que se refiere a su capacidad de *generalización*.[4]

[4]Conviene recordar la razón última del aprendizaje de una red neuronal, que no es otra que conseguir una red que ante entradas que no han participado en el aprendizaje sea capaz de clasificarlas de forma correcta (si la aplicación es la clasificación, por ejemplo). Dicho de otro modo, el aprendizaje de la red tiene que ser un método que permita la extracción de ciertas normas generales que le permita enfrentarse a casos nuevos

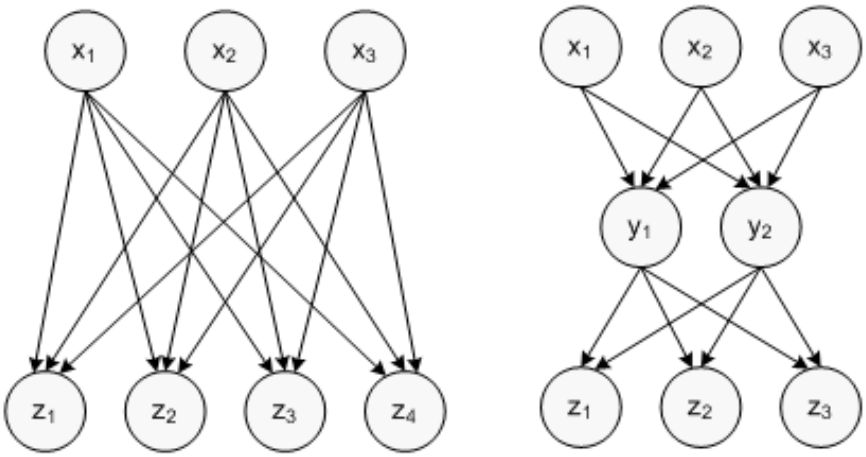


Fig. 3.6: Una red monocapa y una red multicapa

En cuanto a su conectividad, las dos redes que se muestran en la Fig. 3.6 son de dirección *feedforward*, es decir las conexiones tienen un sólo sentido que va del extremo de entrada al de salida, en este caso de arriba a abajo. Algunas ANN, como el perceptrón multicapa (MLP), aunque son típicamente feedforward, en la fase de aprendizaje tienen una conexión típicamente feedbackward, de allí su denominación *backpropagation*.

Las capas competitivas forman parte de un gran número de redes neuronales. Entre las neuronas de una capa competitiva existen implícitas *conexiones laterales* de inhibición pues, aunque no estén conectadas, cada una de las neuronas tiene cierta influencia sobre sus vecinas. El valor que se asigne a los pesos durante el proceso de aprendizaje de la red para las conexiones feedforward que llegan a esta capa dependerá precisamente de esta interacción lateral. Normalmente no se muestran las conexiones laterales entre neuronas de una capa competitiva, pero su efecto está presente de esa manera.

Las conexiones con retardo se utilizan en las *redes dinámicas*. En este tipo de redes su funcionamiento se describe mediante ecuaciones en diferencia o diferenciales dependiendo de la naturaleza discreta o continua de las variables. En particular, una red neuronal con retardo en el tiempo (TDNN) es una red dinámica simple que se puede considerar como una red estática en la que se introducen valores de la entrada en instantes anteriores [38].

Por lo que se refiere al grado de conectividad, una red se considera *totalmente conectada* cuando las salidas de las neuronas de una misma capa están conectadas a todas las neuronas de la siguiente capa; en caso contrario se tiene una red *parcialmente conectada*.

Entrenamiento

En toda aplicación de las redes neuronales existen dos fases, éstas son la *fase de entrenamiento* y la *fase de operación*.[5] En la fase de entrenamiento, se usa un conjunto de datos con la finalidad de que la red ajuste sus pesos sinápticos de modo que su comportamiento se corresponda con el de la aplicación que habrá de desempeñar durante la fase de operación. Es decir, durante la fase de entrenamiento se lleva a cabo el aprendizaje de la red.

(mejor dicho, no utilizados en el entrenamiento)

[5]También se le llama *fase de recall* por el término en inglés

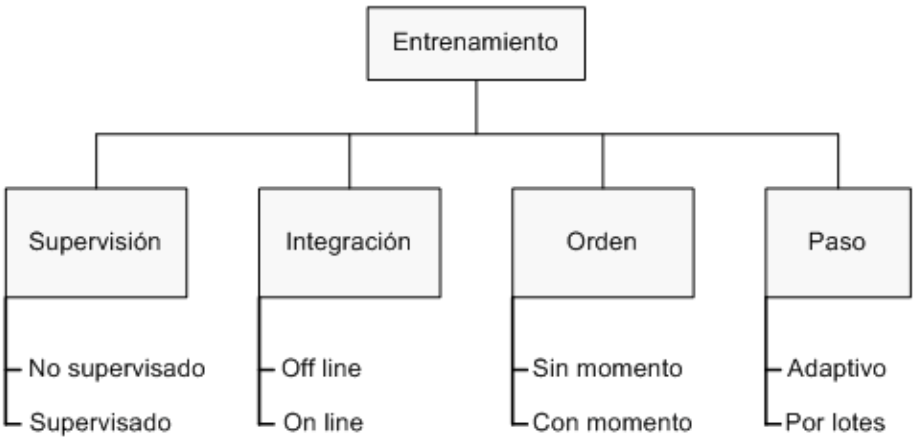


Fig. 3.7: Clasificación de las ANN por los factores del proceso de entrenamiento

Por lo que se refiere a los factores que determinan el entrenamiento, las ANN se pueden clasificar como se muestra en la Fig. 3.7.

Para el entrenamiento supervisado, el conjunto de datos de entrenamiento se puede separar en dos partes: los vectores con los datos que se aplicarán sobre la entrada y los vectores con los datos respectivos que la red debería entregar como respuesta a la salida. Si la red tiene N unidades de entrada y M neuronas de salida, entonces los vectores de entrada serán N-dimensionales y los de salida serán M-dimensionales. El conjunto tendrá tantos vectores de entrada como los de salida. Si X^p es el vector de entrada aplicado a la red en un momento dado, Z^p el correspondiente vector de salida deseado[6] y $\hat{Z}^p$ el vector de salida obtenido, entonces los pesos sinápticos de la red se ajustarán, de acuerdo con algún algoritmo preestablecido, de modo que el error se reduzca. El error es la distancia entre Z^p y $\hat{Z}^p$ y comúnmente se mide usando el cuadrado de la distancia euclidiana:

$$E(W, X^p) = \frac{1}{M} \sum_{j=1}^{M} (Z_j^p - \hat{Z}_j^p)^2, \tag{3.3}$$

siendo W una matriz cuyos elementos son los pesos sinápticos del conjunto de neuronas que componen la ANN.

El error acumulado al aplicar todos los patrones de entrenamiento contenidos en el conjunto será:

$$J(W) = \frac{1}{MP} \sum_{p=1}^{P} \sum_{j=1}^{M} (Z_j^p - \hat{Z}_j^p)^2, \tag{3.4}$$

donde P es el tamaño del conjunto de entrenamiento.

Así, regularmente el entrenamiento es un proceso iterativo que tiene como objetivo final conseguir la matriz de pesos W tal que el error $J(W)$ es mínimo.

En el entrenamiento no supervisado cada patrón tiene únicamente el vector de entrada. Es decir, la salida que genera la red cuando se le aplica un vector de entrada no se compara

[6] A cada pareja formada por X^p y Z^p se le llama un patrón de entrenamiento

contra algún vector de salida deseado. De este modo, el ajuste de los pesos da lugar a una auto-organización de la red que la hace capaz de identificar similitudes entre los patrones que se le aplican durante el entrenamiento, formando clases. Es ésta la base de la funcionalidad de las redes neuronales biológicas, por lo que se refiere al aprendizaje. Para el entrenamiento no supervisado, los pesos se ajustan con la finalidad de minimizar la función de error descrita así:

$$J(W) = \frac{1}{MP}\sum_{p=1}^{P}\sum_{j=1}^{M}(X_j^p - C_j^{k(X^p)}(W))^2, \tag{3.5}$$

donde P es el tamaño del conjunto de entrenamiento y $C^{k(X^p)}$ es el centroide de la clase donde queda ubicado el vector X^p como resultado de la clusterización[7]. Este centroide es dependiente de la matriz de pesos W.

La aplicación fundamental de las redes con aprendizaje no supervisado es la clasificación. Ejemplo de dichas redes son los *mapas auto-organizativos de Kohonen*, que serán tratados con mayor detalle en la sección 3.3, por ser uno de los temas centrales de esta tesis. También tienen este tipo de aprendizaje redes tales como las *ART1* y *ART2* (Adaptative Resonance-Theory) y el *Neocognitrón*.

Las aplicaciones de las ANN pueden ser muy diversas y así son también los requerimientos que aquéllas establecen. Un tipo de aplicaciones son las que requieren que la fase de operación se realice en tiempo real, sin que esta exigencia se extienda a la fase de entrenamiento. En otras palabras, para la implementación del algoritmo de entrenamiento la velocidad de operación del sistema no es un factor determinante, mientras que para la implementación de la fase de operación sí lo es. Entonces se tiene el *entrenamiento off-line*.

Por otro lado, hay aplicaciones que demandan la operación en tiempo real para ambas fases y entonces se presenta el *entrenamiento on-line*. Las aplicaciones que requieren entrenamiento on-line son las que dan lugar a sistemas no lineales cuyos parámetros varían con el tiempo y con las condiciones de operación. Tal es el caso, como ejemplo, de la detección de fallos en sistemas eléctricos [39].

El proceso de entrenamiento es iterativo. Los patrones de entrenamiento se van aplicando uno a uno hasta barrer el conjunto en su totalidad. Una vez que se han aplicado todos los patrones, se dice que se completó un epoch.[8] Al completarse un epoch, se evalúa la función de error y si resulta menor que cierto valor preestablecido, entonces el proceso termina; si no es así, se lleva a cabo otra iteración. Algunos algoritmos permiten elegir entre realizar el ajuste de los pesos hasta completar un epoch o bien hacerlo en cuanto se aplique un patrón de entrenamiento.

Cuando el ajuste de los pesos se realiza hasta completar un epoch,[9] se tiene el *entrenamiento por lotes* o *entrenamiento batch*. En el entrenamiento batch el valor de cada peso se ajusta con el valor promedio del conjunto de ajustes generados para ese mismo peso al aplicar todos los patrones de entrenamiento, uno por uno. Si el valor de cada peso se actualiza de inmediato, usando el valor de ajuste generado por el patrón de entrenamiento aplicado, se tiene el *entrenamiento adaptativo* o *entrenamiento incremental*. El entrenamiento batch produce una convergencia más suave pero impide aprovechar el relajamiento de los pesos a que

[7] La clusterización es un procedimiento a través del cual un conjunto de objetos se separan en grupos con base en varias características particulares

[8] Se acostumbra utilizar el término en inglés, de modo que es poco frecuente encontrar su traducción *época*

[9] Rigurosamente hablando, un lote no necesariamente debe incluir a todo el conjunto de entrenamiento, sino a cierta cantidad de sus elementos

da lugar el entrenamiento incremental.

Ciertos algoritmos de entrenamiento son mejores[10] cuando para el ajuste de los pesos se considera no sólo su cambio, sino también su rapidez de cambio. A los algoritmos que toman en cuenta la rapidez de cambio se les denomina *con momento*. En los algoritmos con momento se requiere tener presente el valor actual de los pesos, así como el valor que tenían en la iteración anterior.

Función de base y función de activación

Para su estudio analítico, la interconexión de las redes se representa matemáticamente por su función de base $\eta(W, X)$, donde W es la matriz de pesos y X el vector de entrada. La función de base tiene dos formas típicas, la *función de base lineal* (LBF) y la *función de base radial* (RBF). La LBF es una función de tipo hiperplano, es decir, una función de primer orden. El valor de entrada a la neurona j de la red es una combinación lineal de las entradas:

$$\eta_j(X, W) = \sum_{i=1}^{N} w_{i,j} x_i. \tag{3.6}$$

Por su parte, la RBF es una función de tipo hiperesférico. En general, una función de base radial es una función cuyo dominio son los números reales y su valor depende únicamente de la distancia al origen o, alternativamente, de la distancia a algún otro punto C llamado el *centro*, de modo que $\phi(X, C) = \phi(\|X - C\|)$, donde $\|V\|$ es la norma de V, por lo común se utiliza la norma euclidiana, de modo que la RBF queda definida así:

$$\eta_j(X, W) = \sqrt{\sum_{i=1}^{N} (x_i - w_{i,j})^2}. \tag{3.7}$$

La función de segundo orden se puede extender a otra más general llamada *función de base elíptica*.

Para la función de activación existe un número considerable de variantes. Líneas atrás, en la Fig. 3.4, se mostró la gráfica para cuatro de ellas. Otras funciones de activación comunes son la gaussiana, $f(s) = \alpha e^{\beta s^2}$, y la tangente hiperbólica, $f(s) = (e^{\beta s} - e^{-\beta s})/(e^{\beta s} + e^{-\beta s})$.

3.1.4 Enfoques de la investigación en el campo de las ANN

Las redes neuronales están encontrando aplicación en numerosas áreas relacionadas con el procesamiento de señales y con tareas de clasificación y reconocimiento de patrones complejos, tales como reconocimiento de caracteres y voz, filtrado de ruido, compresión de datos, predicción de series temporales y optimización. Otra área de investigación importante es el control automático, donde las redes neuronales se están estudiando para el control adaptativo de robots y vehículos autónomos, y la identificación y control de procesos complejos o desconocidos.

La teoría que subyace a las redes neuronales se deriva de muchas disciplinas tales como la neurociencia, las matemáticas, la psicología, la física, la ingeniería, la biología y las ciencias computacionales. El tema de las ANN comprende principalmente tres áreas. La primera

[10] Por mejores se entiende que tienen una convergencia más rápida y que son capaces de evadir los mínimos locales y así arribar al mínimo global

se refiere a la teoría y arquitectura, esto es la definición matemática y el análisis de la red neuronal independientemente del soporte hardware o software que tenga. La segunda área es la implementación de la red neuronal en un hardware físico o en un lenguaje software. La tercera área es la de las aplicaciones.

El interés principal para el desarrollo de esta tesis reside en esa segunda área, particularmente en el ámbito de las implementaciones hardware, pero no por eso se pueden dejar de lado las otras dos. En [40] se hace un recuento del desarrollo de las redes neuronales y sus logros en los últimos tiempos, planteando conclusiones como las siguientes:

- El conocimiento en las redes neuronales se ha enriquecido por el desarrollo de estructuras matemáticas más sólidas, por la elaboración de algoritmos más potentes y más eficientes, por la revelación de misterios acerca de las redes biológicas tanto a nivel de las neuronas individuales como en niveles jerárquicos y de funciones cerebrales.

- En el campo de las implementaciones hardware, el progreso también se ha mantenido. El reto ha consistido y aún continúa siendo el de emular lo que hace la biología y cómo lo hace, para lograr la capacidad de construir sistemas más grandes y más complejos, para implementar en hardware técnicas eficientes de aprendizaje, para mejorar la velocidad o el consumo de potencia, para desarrollar nuevas técnicas de representación de estados y señales, para explotar eficientemente las propiedades estadísticas y de ruido, y para emplear todas estas ideas en aplicaciones del mundo real donde las soluciones hardware eficientes son necesarias para lograr alta miniaturización, alta velocidad y bajo consumo de potencia.

- Hasta el momento el estado del arte en las redes neuronales ha alcanzado un importante nivel de madurez, en donde los investigadores e ingenieros son capaces de construir sistemas muy complejos que se pueden aplicar a problemas del mundo real.

Para el número especial del que [40] sirve como editorial, se seleccionaron inicialmente 70 artículos para finalmente publicar 35 en la edición especial. Los artículos se clasificaron primero en dos grandes grupos: desarrollos principalmente analógicos y trabajos principalmente digitales. Dentro de los digitales, los artículos se distribuyeron en 4 clases: implementaciones basadas en FPGA (5), implementaciones basadas en DSP (2), implementaciones basadas en ASIC (7) e implementaciones mixtas (1). Entre los artículos de implementación en ASIC, uno está dedicado a la cuantización vectorial jerárquica combinando un ASIC con FPGA's. Dos más están dedicados a implementar redes del tipo Self-Organizing Feature Map (SOM).

3.2 Implementación hardware de las redes neuronales

Hasta hace pocos años, los científicos y los ingenieros en el campo de las redes neuronales han dependido de las simulaciones por ordenador, a veces realizadas sobre computadoras especializadas, para realizar su labor de investigación y desarrollo. Principalmente se han empleado microprocesadores gracias a su pequeño tamaño, bajo precio, alto nivel de prestaciones y bajo consumo. El inconveniente fundamental de tales simuladores es que el paralelismo espacio temporal en el procesamiento de la información, inherente a las redes neuronales, se pierde parcial o completamente, de modo que el tiempo de computación de la red simulada, especialmente para una gran asociación de neuronas, crece hasta alcanzar órdenes de magnitud que hacen imposible o al menos retrasan en gran medida la obtención de resultados.

El entrenamiento de una red es de lejos la función más costosa computacionalmente. La velocidad de entrenamiento se calcula en número de conexiones actualizadas por segundo (Connection Updates Per Second) y depende en gran medida de las características arquitectónicas de la red, tales como el número de capas y el número de nodos por capa. Así, el entrenamiento precisa de tiempos prolongados cuando se usan plataformas convencionales. Investigadores de Sharp, por ejemplo, necesitaron 2 meses para entrenar su OCR Kanji sobre una estación de trabajo Sun 4/330. De aquí la necesidad de hardware que explote el paralelismo inherente de las redes neuronales.

Por lo tanto, se ha iniciado el camino hacia la integración hardware de las redes neuronales, prueba de ello es el número de chips bajo desarrollo, así como los que ya están en el mercado. Mediante el diseño especializado de hardware neuronal se puede conseguir una apreciable reducción del tiempo de computación, logrando incluso aplicaciones en tiempo real. Además, ofrece un volumen estructural muy inferior al logrado por simuladores. Este aspecto es muy importante, pues ahora se precisan redes neuronales integradas para sistemas móviles o descentralizados, en robótica por ejemplo, de modo que debieran estar disponibles como componentes microelectrónicos.

3.2.1 Clasificación del hardware neuronal

Si la clasificación de las redes neuronales constituye una tarea ardua por el gran crecimiento de diferentes alternativas acogidas bajo el mismo apelativo, no lo es menos la de las alternativas hardware disponibles en el mundo de las redes neuronales. A partir de este momento y en lo sucesivo se utilizará en esta tesis el apelativo de neurocomputadoras para hacer referencia a todas las vertientes de implementaciones hardware de redes neuronales. La construcción de neurocomputadoras es cara en términos de tiempo de desarrollo y recursos, y además se desconoce si tendrán perspectivas reales de uso comercial. Por otra parte, el número de nuevas variedades de redes neuronales y nuevos paradigmas sigue creciendo incesantemente y tampoco parece haber un claro consenso sobre cómo explotar las capacidades de las tecnologías VLSI y ULSI para la implementación hardware de las redes neuronales. Todas estas consideraciones no hacen sino destacar las dificultades que supone el mundo de las neurocomputadoras; sin embargo, como contrapartida, muchos de los nuevos paradigmas no se conocen en todas sus posibilidades porque éstas sólo se pueden estudiar en total profundidad mediante implementaciones hardware.

La mayoría de los productos disponibles comercialmente en el terreno de las neurocomputadoras son implementaciones dedicadas a los tipos de redes neuronales más conocidos y exitosos, como es el caso del perceptrón multicapa, la red de Hopfield o los modelos de Kohonen. Estas implementaciones dedicadas, por lo general, no ofrecen mucha flexibilidad para la simulación de paradigmas alternativos.

La sexta generación de computadoras se puede considerar como una integración de propuestas provenientes de las ciencias cognoscitivas y la neurociencia en la programación y diseño de computadoras. La máquina del futuro debe realizar el control de un sistema de computación que integre diferentes subsistemas, cada uno de los cuales es bastante especializado en su estructura o arquitectura y alguno de ellos dotado de sensores y motores. Algunos de estos subsistemas se podrían implementar mediante redes neuronales.

Tras estas primeras consideraciones, se puede hacer la siguiente clasificación de las neurocomputadoras que coincide prácticamente con la realizada por Rückert [41], sin dejar de lado otras aportaciones que han sido realizadas por otros autores a lo largo de varias revisiones

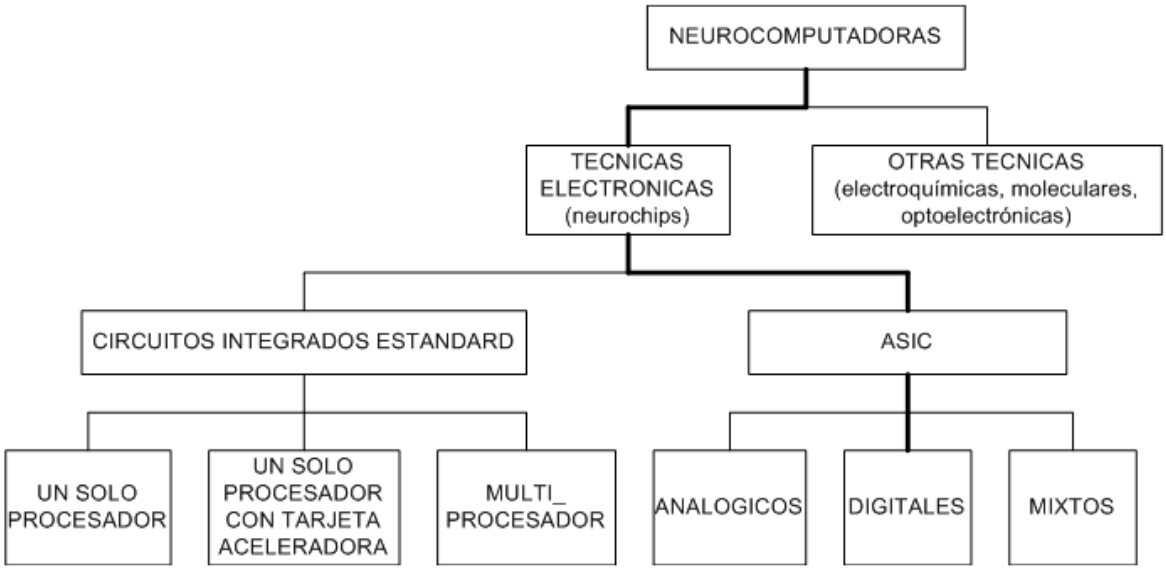


Fig. 3.8: Clasificación de las neurocomputadoras

realizadas sobre el tema de implementaciones hardware de redes neuronales: como las de Treleaven [42], Ramacher y Rückert [43] Nordström et al. [44], Vellasco [45], Ienne [46], Glesner y Pöchmüller [47] y Heemskerk [48]. La clasificación puede observarse en la Fig. 3.8.

3.2.2 Aportaciones deseables

Velocidad

Este es el aspecto fundamental que ha motivado el desarrollo de implementaciones hardware de redes neuronales. Las aplicaciones prácticas suelen demandar soluciones en tiempo real para la emulación de redes neuronales grandes que manejan conjuntos de entrenamiento de considerables dimensiones. En la mayoría de los casos sólo se puede dar respuesta a estas aplicaciones mediante implementaciones hardware específicamente diseñadas para el problema en consideración. En este campo de aportación habría dos subconjuntos:

- Aceleración de la fase de recall. Es decir, se tiene la red neuronal ya entrenada y se quiere que haga un determinado procesamiento lo más rápido posible, justificado normalmente por aplicaciones en tiempo real.

- Aceleración del entrenamiento. El entrenamiento suele ser off-line, es decir que se realiza fuera del sistema y no necesita realizarse una vez que la red ya está operando en un sistema. El que se haga off-line no quita que sea importante acelerarlo lo máximo posible, porque algunos de estos entrenamientos, como el que se comentó anteriormente de reconocimiento de caracteres, son excesivamente lentos. Esta lentitud se hace insoportable si se está explorando cuál es la topología, patrones de entrenamiento y variables de entrada adecuados para obtener una solución optima, porque esta exploración puede suponer la realización de cientos o miles de entrenamientos. Es difícil encontrar aplicaciones en los que el entrenamiento se realiza “on the fly”, es decir mientras el sistema

funciona. Los hay, pero son muy especiales. En estos casos el entrenamiento en tiempo real fija restricciones de velocidad muy exigentes.

Portabilidad

En algunos casos la velocidad del sistema no es lo más importante, más bien lo es la capacidad de autonomía para trabajar con datos remotos. Para estos casos, el que la implementación sea pequeña, de bajo consumo y portable es preferible a implementaciones de gran tamaño aunque éstas sean más flexibles y más rápidas.

Conectividad

Una de las características inherentes de los modelos de redes neuronales es la fuerte colaboración en paralelo de las unidades individuales que las componen. Esto supone grandes esfuerzos de conectividad que si no son bien resueltos en las implementaciones hardware dan lugar a unos costes de área de silicio enormes para garantizar esa conectividad. Para evitar esos problemas, aquellas arquitecturas que se rigen por su conectividad local entre unidades, como son las arquitecturas sistólicas, podrían ser las preferibles.

Memoria

El almacenamiento de los pesos de las conexiones de la red en forma compacta y segura es fundamental porque en definitiva establece la funcionalidad de la red. Además, y como dificultad añadida, los algoritmos de aprendizaje de la mayoría de los modelos de redes neuronales trabajan adaptando estos pesos, como es el caso de los algoritmos de Kohonen, con lo que además de almacenar los pesos, éstos deberán ser fácilmente modificados, y además tendrán que almacenarse las modificaciones que tengan que realizarse sobre los mismos (por ejemplo en la versión Batch SOM) o incluso las modificaciones ya realizadas (por ejemplo en la versión con momentum del algoritmo Backpropagation). Esto presenta una gran dificultad y es sin duda la principal razón que limita la inclusión del aprendizaje en las versiones analógicas de los Neurochips.

Precisión

Este es el factor más limitante en las implementaciones digitales, puesto que el conseguir una buena precisión (necesaria para la mayoría de los algoritmos de aprendizaje para que éstos converjan) se logra a través de soluciones con un gran tamaño de silicio y con poca velocidad.

Tecnología

Esto, más que una aportación deseable, es una necesidad y una elección que vendrá determinada por los requerimientos y limitaciones impuestas por las especificaciones del problema o aplicación a solucionar. Como es sabido, las implementaciones digitales son muy flexibles, con capacidad de almacenamiento fácil y compacta, inmunes al ruido, aunque con las contrapartidas de la enorme área necesaria y velocidad no excesivamente alta. Por otra parte, las implementaciones analógicas son compactas y rápidas; sin embargo son seriamente penalizadas por el problema del almacenamiento de los pesos. En este contexto, las soluciones mixtas analógico digitales, basadas en aritmética “pulse-coded” y en técnicas de capacitancias

conmutadas, están emergiendo como alternativas que consiguen integrar flexibilidad, aprendizaje, velocidad y tamaño pequeño.

Proceso de diseño

Esta es otra necesidad que se debe satisfacer para las implementaciones hardware de redes neuronales. Es sin duda alguna el aspecto más negativo que tienen las implementaciones hardware frente a las implementaciones software. Dentro de las implementaciones hardware, las implementaciones digitales puras son las que tienen mayores posibilidades y, por tanto, es necesaria una metodología de trabajo fiable, rápida y eficaz para diseñar implementaciones hardware de redes neuronales basadas en el uso extensivo de lenguajes de descripción de hardware (HDL), y sus herramientas asociadas, y en el fácil prototipado en FPGA's. La extensión de los HDL al mundo analógico podrá también facilitar las implementaciones mixtas anteriormente descritas, pero cabe recalcar que ahora no es más que una perspectiva de futuro.

3.2.3 Neurochips

Si se revisan los valores de rendimiento alcanzados por neurocomputadoras realizadas a través de C.I. estándar, se puede apreciar que no son en modo alguno desdeñables [48][49]. Sin embargo en dichos proyectos las funciones neuronales se programan en procesadores de propósito general, mientras que en las neurocomputadoras diseñadas con base en neurochips, éstos se encargan de realizar las funciones neuronales, lo cual puede acelerar el tiempo de iteración en cerca de dos órdenes de magnitud.

Se han barajado y se barajan muchas alternativas tecnológicas a la hora de la realización de los neurochips. Lo que debe quedar claro es que la implementación directa en circuitos puede alterar en muchos casos el funcionamiento exacto de los elementos computacionales. Esto se debe principalmente a la precisión limitada, cuya influencia es básica para el correcto funcionamiento de los paradigmas originales de los modelos neuronales. Una metodología de diseño de los neurochips debe poder evaluar en fases tempranas del proceso de diseño los efectos de la precisión. También es de vital importancia tener en cuenta el otro gran aspecto de los neurochips, que es la integración en arquitecturas de gran escala, es decir, se debe tomar en cuenta que muchos de estos neurochips tienen que ser interconectados, por lo tanto el problema de comunicación entre ellos se debe resolver con eficacia.

Al igual que otros sistemas de procesamiento, una red neuronal consiste en un conjunto de elementos implicados en tareas tanto de computación como de comunicación. La complejidad de los sistemas neuronales no deriva de la complejidad de sus dispositivos sino de la multitud de formas en las cuales un gran número de estos dispositivos pueden interactuar. Así, la computación neuronal es una propiedad que surge del comportamiento colectivo de muchas unidades de procesamiento primitivas, altamente interconectadas. Los aspectos clave para lograr la integración con éxito de tales sistemas son los siguientes:

- Los modelos de redes neuronales se deben implementar con base en arquitecturas modulares, regulares, replicables y altamente paralelas que las hagan atractivas a la integración VLSI.

- Las unidades de procesamiento (neuronas) deben estar organizadas de forma compacta de modo que puedan integrarse en el chip un número suficientemente elevado de ellas,

dado que la mayor parte de las aplicaciones requieren redes con un gran número de neuronas.

- Las celdas de memoria (sinapsis) deben ser igualmente compactas, eficientes y deben almacenar los pesos como valores modificables.
- El paralelismo, la flexibilidad, las prestaciones y su relación con el área de silicio. Estos aspectos conducen a sistemas radicalmente diferentes.
- El planteamiento de soluciones al problema de la compleja conectividad entre neuronas (geometría de la red). Un esquema de interconexión de altas prestaciones impone una severa limitación sobre el número posible de elementos de procesamiento.
- La minimización del número de terminales de E/S del chip. Este aspecto influye considerablemente en el coste del mismo, dado que el encapsulado puede ser incluso más caro que el propio chip. Además, la alimentación de los terminales de salida representa un alto porcentaje de la potencia total requerida por el chip.
- El diseño debe ser escalable de modo que pudiera hacer uso de nuevos procesos tecnológicos caracterizados por parámetros geométricos más pequeños.

3.2.4 Neurochips digitales

La implementación digital de redes neuronales representa una tecnología madura y bien comprendida, que ofrece mayor flexibilidad, escalabilidad y precisión que la implementación analógica. A continuación se describen los requisitos que debe verificar una implementación digital:

- Altas prestaciones. Los sistemas requieren un número muy alto de unidades de procesamiento. Dada la necesidad de múltiples pasadas de los patrones de entrenamiento, se precisa un ratio de procesamiento muy elevado para conservar el tiempo de entrenamiento en un rango razonable. Por otro lado, durante la etapa de recall, si bien la operación es mucho más simple, el requerimiento de tiempo real de nuevo dicta un ratio de procesamiento muy elevado. Debido a la tecnología hardware y a los algoritmos actuales, la ejecución paralela es la única forma de lograr los ratios de procesamiento requeridos.
- Reconfigurabilidad. La investigación en redes neuronales produce una gran variedad de nuevos algoritmos, la programabilidad es por tanto necesaria. Además, a veces se precisa combinar más de un tipo de red en la realización de sistemas. Los sistemas programables son menos eficientes que los sistemas de propósito especial optimizados, aunque la mayor parte de las operaciones son repetitivas, lo cual puede utilizarse para incrementar la eficiencia.
- Escalamiento. Cualquier sistema paralelo idealmente debería ser escalable en el sentido de que su factor de mejora debería permanecer aproximadamente constante con el incremento del número de procesadores. En otras palabras, la complejidad del sistema no debería introducir un costo adicional de comunicación y sincronización. Los arrays sistólicos con sus modelos de interconexión local y regular y su concurrencia en la computación y la comunicación son ideales en este aspecto.

El nivel de prestaciones dicta claramente una arquitectura paralela. Los algoritmos de redes neuronales se corresponden con una serie de operaciones del tipo producto-suma. En tales operaciones, el factor limitante es la velocidad con que se puede realizar una serie de sumas, dado que todas las multiplicaciones pueden hacerse en paralelo. Se precisa pues, escoger una arquitectura que, verificando los requisitos expuestos anteriormente, esté bien adaptada para tales operaciones. Igualmente, una decisión a tomar está en el número de elementos de procesamiento (granularidad del paralelismo) y su forma de sincronización. El uso de una señal de reloj global simplifica grandemente el control de los elementos de procesamiento, pero crea problemas de sincronización en grandes arrays. Debido a la limitada velocidad de propagación de las señales en el chip o en la tarjeta, los procesadores más alejados de la fuente de reloj recibirán los pulsos de reloj con un retardo relativo de unos pocos nanosegundos respecto al procesador más próximo a la fuente (clock skew). La alternativa consiste en un esquema asíncrono donde cada procesador tiene su propio reloj y sincroniza con los otros mediante señales de protocolo.

Los principales inconvenientes de la implementación digital VLSI son:

- Un área de silicio ocupada mucho mayor.
- Una velocidad relativamente más baja.
- Las dimensiones de las celdas de memoria dedicadas al almacenamiento de los pesos sinápticos.
- Un mayor coste de interconexión de las unidades de procesamiento.
- La implementación de unidades de procesamiento con capacidad para evaluar alguna función de activación de tipo sigmoidal, lo cual implica el diseño de un procesador complejo o el uso de tablas en memoria.

Unidades de procesamiento de baja complejidad

En una primera línea de aproximación, cada elemento de procesamiento (EP) se corresponde con una neurona que tiene asociado un conjunto de pesos sinápticos implementados mediante un registro de desplazamiento circular; en cada paso de tiempo, la señal de salida de una neurona y el peso sináptico asociado a la conexión entre la misma y la de aquella cuya entrada se está calculando, se presentan y procesan por el EP, de modo que sólo se requiere un multiplicador dentro del mismo. Esta arquitectura obviamente conduce a una reducción de la complejidad espacial de orden N, de este modo se produce una serialización de la operación suma. Los principales puntos de interés de esta arquitectura basada en registros de desplazamiento, son el uso de bloques sencillos y bien establecidos y la inherente simplicidad de la red de interconexión.

Unidades de procesamiento de alta complejidad

Una segunda línea de aproximación se basa en el diseño de procesadores neuronales potentes y relativamente complejos funcionando con un grado de paralelismo relativamente bajo. Aun cuando tal solución hace posible implementar el aprendizaje en la red, gracias al uso de una arquitectura en cierto modo similar a la de un procesador de propósito general, esta solución precisa de una gran área de silicio para cada procesador. En este caso, cada procesador realiza la función de varias neuronas en subsiguientes pasos de tiempo.

Unidades de procesamiento de complejidad media

Otra propuesta interesante, que representa un punto medio entre la aproximación paralela y la serie, consiste en la identificación de un conjunto básico de "cadenas de operaciones" comunes a los algoritmos de redes neuronales más importantes y crear el conjunto correspondiente de bloques implementados sobre silicio. La arquitectura del chip estaría basada en el concepto de array sistólico. Esta, como ejemplo, ha sido la opción tomada por Siemens con su chip MAI16. Cada chip tiene 4 cadenas de computación sistólica y cada cadena computa 4 conexiones por ciclo de reloj sobre una submatriz de 4x4 leída y almacenada internamente. El chip tiene un reloj de 50 MHz y puede ejecutar 800 millones de operaciones multiplicar-acumular por segundo. La arquitectura sistólica implementada a nivel de chip se puede extender además a nivel de placa.

3.2.5 Implementación hardware basada en ASICs

Valoración y comentarios.

En el caso de tener un ASIC como solución a una red neuronal, está claro que se tiene una solución específica para la dualidad red neuronal-aplicación, lo cual conlleva como ventaja evidente que la solución es óptima en términos de rendimiento, eficacia y velocidad, por cuanto ha sido diseñada específicamente para resolverlo de esa manera.

La desventaja evidente es que se está hablando de una solución de elevado coste, tanto en tiempo como en dinero para el desarrollo de la misma. Por lo cual el ingeniero debe evaluar la viabilidad técnica y económica de la elección de esta alternativa. La aplicación debe tener unas condiciones de contorno muy severas para justificar este tipo de soluciones.

Por otro lado se tiene la poca flexibilidad de este tipo de soluciones. Por ejemplo, suponiendo que se tiene una determinada clasificación realizada mediante un perceptrón multicapa y se desea disponer de una variable más en el campo de entradas. Este cambio, por leve que parezca, puede invalidar completamente la adecuación del chip específicamente diseñado y replantee un rediseño casi total.

Estas consideraciones de partida son perfectamente válidas considerando los ASIC programables por máscara. En el caso de los ASIC programables eléctricamente (CPLD, FPGA), las ventajas no son tan considerables, pero también las desventajas son mucho más ligeras. La solución ya no será tan veloz y óptima que con un ASIC programable por máscara, sin embargo el tiempo de diseño y los costes de desarrollo se ven reducidos y la flexibilidad fruto de la reprogramación o la reconfiguración permite adecuar el diseño a cambios de las condiciones de contorno de la aplicación.

Rendimientos de implementaciones con FPGA.

Es difícil comparar rendimientos de varios métodos basados en FPGA, en primer lugar porque suelen implementar diferentes arquitecturas de redes neuronales, con diferentes opciones de implementación (precisión, entrenamiento incluido o no, tamaño de las topologías). Además, los tipos de las FPGA y, lo que es más importante, las familias varían y gran parte de las mejoras que se atribuyen los diseñadores son en realidad mejoras de la tecnología que subyace. Se puede observar este efecto en los resultados obtenidos por implementaciones realizadas con tres tecnologías del mismo fabricante de FPGA para la misma topología y arquitectura de un MLP que se condensan en la Tabla 3.1.

Familia	Virtex 2P	Virtex 4	Virtex 5
Tecnología	0.13 μm, nueve capas	0.09 μm, diez capas	0.065 μm, doce capas
Dispositivo	XC2VP2	XC4VLX15	XC5VLX30
F_{max} obtenida	83 MHz	89 MHz	105 MHz
Cadencia recall	$0,084\mu s$	$0,078\mu s$	$0,067\mu s$
Cadencia entren	$0,35\mu s$	$0,32\mu s$	$0,275\mu s$
Rendimiento MCPS	452	484	572
Rendimiento MCUPS	109	117	138

Tabla 3.1: Resumen de resultados para un MLP(2-6-2-2)

3.3 Técnicas para compresión de imágenes basadas en el MLP

Con el propósito de efectuar una revisión de las técnicas de compresión de imágenes basadas en redes neuronales, en esta sección se exponen dos de las más empleadas, siendo éstas la transformación auto-asociativa implementada por medio de un *perceptrón multicapa* (MLP)[11] y los predictores no-lineales basados en el MLP. Además, la siguiente sección estará dedicada a otra técnica, la cuantización vectorial mediante redes competitivas. Por el interés que tiene en general dentro del campo de las redes neuronales y, en particular, para la compresión de imágenes, primero se exponen los fundamentos del MLP.

3.3.1 Fundamentos del perceptrón multicapa

El MLP es sin duda la red neuronal más ampliamente utilizada. A continuación se presentan los aspectos básicos de su arquitectura y su algoritmo de entrenamiento.

Estructura del MLP

Por su estructura, el MLP es una red multicapa, multineurona, con conexión hacia adelante, totalmente conectada, no recurrente, y entradas únicamente excitadoras. Por lo que se refiere al entrenamiento, emplea entrenamiento supervisado con base en un algoritmo denominado backpropagation, con función de base lineal y, comúnmente, función de activación sigmoidal. La Fig. 3.9 muestra un MLP de tres capas, con 4 unidades de entrada, 5 neuronas en la primera capa oculta, 7 en la segunda y 3 en la capa de salida, por lo cual se le denomina MLP 4-5-7-3. Esta red contiene 76 conexiones.

La estructura del MLP permitió:

- Tener una hipersuperficie de separación en lugar de un hiperplano (que es lo que se puede obtener con una red monocapa) con lo cual se pueden resolver problemas que no son linealmente separables (que son los únicos que pueden resolver las redes monocapa).
- Con esta hipersuperficie de separación, dividir el espacio en regiones no conectadas y cada una de estas regiones asociarla a una clase en particular.

[11] Por su sigla del inglés Multi Layer Perceptron

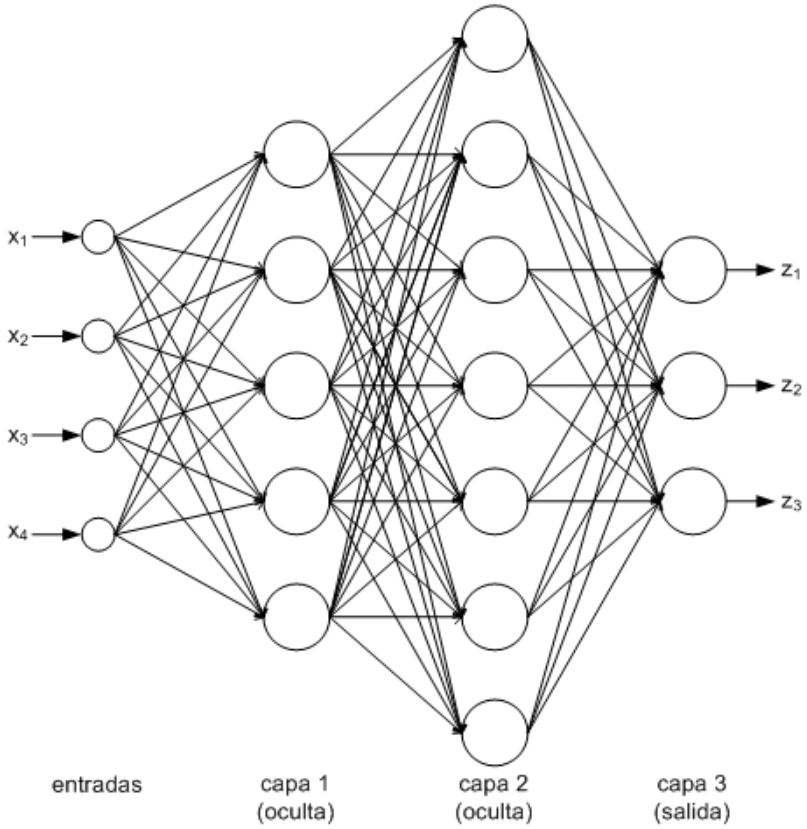


Fig. 3.9: Un MLP de tres capas 4-5-7-3

- Tener regiones no conectadas cerradas, para lo cual se recurre a dos capas ocultas o a una capa oculta con un número muy elevado de unidades.
- Resolver el problema de la OR exclusiva, que fue el talón de Aquiles para el perceptrón.
- Convertir cualquier conjunto de datos no linealmente separable en un conjunto de datos linealmente separable mediante una transformación entre espacios.

El algoritmo backpropagation

Rumelhart, Hinton y Williams [50] propusieron el algoritmo backpropagation. En ese trabajo los autores expusieron el algoritmo para el aprendizaje de redes multicapa dando una serie de ejemplos para mostrar la potencialidad del método desarrollado. La publicación dio lugar a un fuerte impulso a la investigación de las ANN. A continuación se describe brevemente el algoritmo.

Como se mencionó atrás, el entrenamiento tiene como propósito determinar los pesos de la red para minimizar el error total dado por la Ec. 3.3. Ya que no existe una solución analítica se requiere un algoritmo iterativo y el backpropagation es uno de ellos. En cada iteración se efectúan los siguientes pasos: a) se toma un patrón y se aplica a la red el vector de entrada, b) se calcula el error en la salida de la red usando la Ec. 3.3, c) se ajustan los pesos en la capa de salida de modo que se reduzca el error, y d) se propaga el error hacia la capa de

entrada para ajustar los pesos de las capas ocultas. A continuación se detalla brevemente el algoritmo.

En la parte fundamental del aprendizaje, es decir la que se refiere al ajuste de los pesos, el algoritmo se basa en el *gradiente del error*. Por lo tanto, será necesario obtener las derivadas parciales del error con respecto a cada uno de los pesos para así determinar la dirección en la cual el gradiente del error tiene su valor más negativo. Aplicando la regla de la cadena para calcular las derivadas parciales y debido a que la función de base es lineal, el proceso se simplifica y la derivada parcial se convierte en la derivada de la función de activación. Esta es la razón por la cual la función de activación debe ser diferenciable y, preferentemente, muy fácil de derivar. Por ejemplo, si la función de activación $f(s)$ es sigmoidal, entonces $f'(s) = f(s)(1-f(s))$ y sólo es necesario evaluar una función. A partir de estas consideraciones se derivan las ecuaciones que se utilizan para la actualización de los pesos, quedando como siguen:

$$z_j^l = \begin{cases} x_j, & \text{para } l = 0 \\ f(s_j^l), & \text{para } l = 1, 2, \dots L \end{cases}, \tag{3.8}$$

$$s_j^l = \sum_{k=1}^{N_{l-1}} z_k^{l-1} w_{jk}^l \quad l = 1, 2, \dots L;\ j = 1, 2, \dots N_l,$$

$$\delta_j^l = \begin{cases} f'(s_j^l)(t_j - z_j^l), & \text{para } l = L \\ f'(s_j^l) \sum_{k=1}^{N_{l+1}} w_{kj}^{l+1} \delta_k^{l+1}, & \text{para } l < L \end{cases}, \tag{3.9}$$

$$\Delta w_{ji}^l = \eta \delta_j^l z_i^{l-1}. \tag{3.10}$$

Donde w_{ji}^l es el peso de la conexión que va de la neurona i de la capa $l-1$ a la neurona j de la capa l;[12] Δw_{ji}^l es el valor de ajuste que se suma al valor actual de ese peso para obtener el valor nuevo; s_j^l es la entrada neta a la neurona j de la capa l; N_l es el número de neuronas en la capa l; z_j^l es la salida de la neurona j de la capa l; f es la función de activación y f' es su derivada con respecto a la entrada neta; t_j es la salida deseada en la neurona j de la capa de salida; δ_j^l es la derivada parcial del error con respecto a la entrada neta de la neurona j de la capa l; y L es el número de capas, sin considerar como tal a la formada por las unidades de entrada.

El procedimiento consiste en, primero, recorrer la red desde el extremo de la entrada (capa 1) hasta el extremo de la salida (capa L) y, segundo, volverla a recorrer, pero ahora en sentido contrario, es decir, desde la capa L hasta la capa 1. En el primer recorrido, mediante las fórmulas dadas por la Ec. 3.8, para cada neurona se evalúa su función de base y con el valor resultante se evalúa su función de activación, con lo cual se obtiene su salida, que se usará como una de las entradas para todas las neuronas de la capa siguiente. Al llegar a la última capa, la salida producida se utilizará para obtener el valor de δ para cada una de sus neuronas, usando el primer caso de la Ec. 3.9. Entonces comenzará el recorrido hacia atrás, en el cual los valores de δ de las neuronas de la capa L se usarán para obtener los de la capa

[12]Si $l = 1$, se hace referencia a la primera capa oculta, de modo que en ese caso $l - 1 = 0$ se referiría a las unidades de entrada

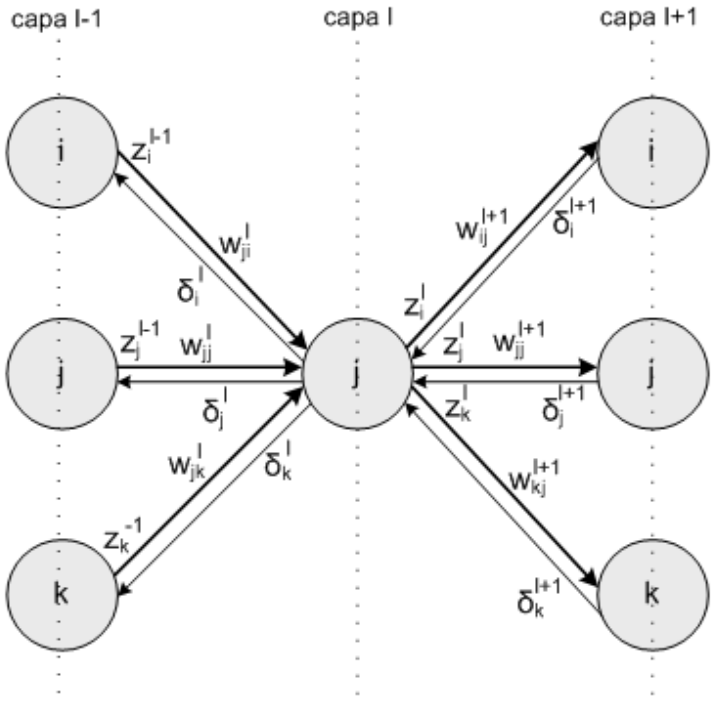


Fig. 3.10: El doble recorrido de la red en el algoritmo backpropagation

$L-1$, mediante el segundo caso de la Ec. 3.9, y así se continuará hasta llegar a la capa 1. Finalmente se calculan los valores de ajuste con la Ec. 3.10 y se actualizan todos los pesos.

En la Fig. 3.10 se ilustra gráficamente el algoritmo backpropagation. Las rectas gruesas corresponden a las conexiones en el sentido *forward*, estas conexiones asocian las salidas de la capa anterior con los pesos sinápticos de la capa actual; mientras que las rectas delgadas se refieren al recorrido en el sentido backward y asocian los valores de δ y sus pesos sinápticos para las neuronas de la siguiente capa.

η es un parámetro del entrenamiento al que se denomina *rapidez de aprendizaje*. El valor de este parámetro influye en la rapidez con la que el algoritmo converge para minimizar la función de error. Un valor muy grande de η provoca que en la búsqueda del mínimo el algoritmo oscile a su alrededor antes de alcanzarlo, mientras que un valor muy pequeño da lugar a un avance muy lento en esa búsqueda.

3.3.2 Compresión de imágenes por transformación autoasociativa mediante el MLP

La aplicación del MLP para la compresión de imágenes arranca con el ya clásico trabajo de Cottrell, Munro y Zipser en 1985 [51], en el cual mediante el esquema denominado cuello de botella (bottle-neck) se lleva a cabo una transformación de los vectores de entrada del espacio N-dimensional al M-dimensional, siendo $M < N$. Al esquema de red que se muestra en la Fig. 3.11 se le ha denominado CMZ. La red tiene N entradas e igual número de salidas. La capa oculta consta de M neuronas.

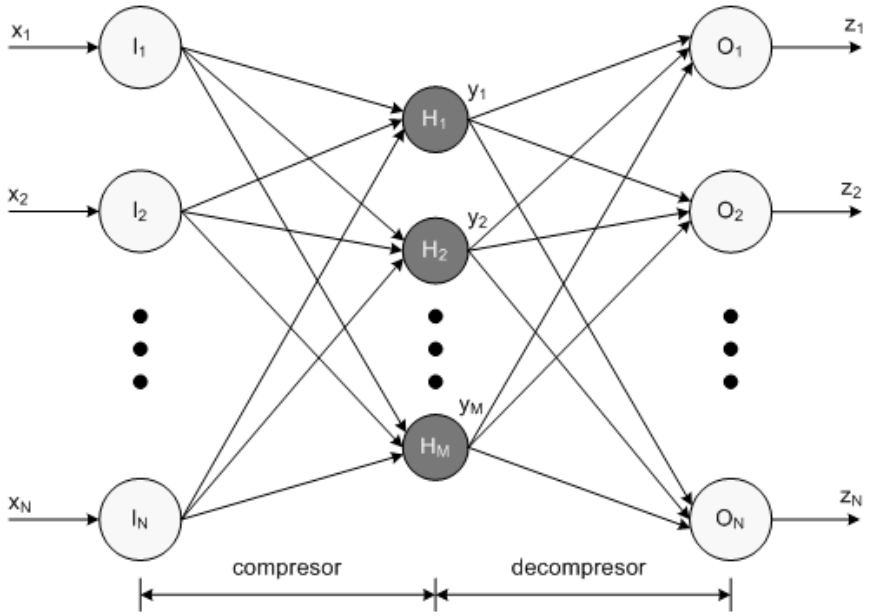


Fig. 3.11: Esquema cuello de botella del perceptrón multicapa para compresión de imágenes

Operación y entrenamiento

Cottrell, Munro y Zipser vislumbraron el compresor de imágenes como el diseño de un codificador en el cual la red debe realizar una transformación de identidad a través del cuello de botella. Es decir, los datos aplicados en la entrada de la red deben aparecer idénticos en su salida después de haber pasado por el canal angosto formado por las neuronas de la capa oculta. El que los datos de salida sean idénticos a los de la entrada es sólo el requerimiento de diseño, ya que en realidad la diferencia entre ellos es el error a minimizar como función de los pesos sinápticos.

Para comprimir una imagen, ésta se descompone en bloques de N pixeles no solapados. La codificación de un bloque se lleva a cabo colocando en la entrada de la red los valores de los N pixeles que lo componen. El código resultante estará formado por los M valores que se obtienen en la salida de la capa oculta. Es decir, el codificador estará constituido por la sección de la red comprendida por las neuronas de la capa oculta y las conexiones que provienen de las unidades de entrada. En el momento de reconstruir la imagen, para decodificar un bloque se proporciona a las neuronas de la capa de salida los M valores del código y como resultado se obtienen los valores de los N pixeles que componen el bloque reconstruido. En otras palabras, el decodificador estará formado por la parte de la red comprendida por las neuronas de la capa de salida y las interconexiones provenientes de la capa oculta.

La razón de compresión es igual al cociente N/M. Por ejemplo, si la imagen se descompone en bloques de 8 x 8 pixeles ($N = 64$) y la capa oculta de la red tiene 8 neuronas ($M = 8$), entonces la razón de compresión será igual a 8:1.

Para el entrenamiento de la red, se construye el conjunto de entrenamiento formado por bloques de imágenes correspondientes a la clase para la cual el compresor está destinado. A continuación, cada bloque se presenta en la entrada de la red y al mismo tiempo se emplea como la salida deseada. Los pesos sinápticos se ajustan de acuerdo al algoritmo backpropa-

gation.

Durante la fase de entrenamiento, la red opera como un todo, es decir como un MLP de dos capas. Para la fase de recall, la red se distribuye en dos partes. En el lado del codificador se tendrá únicamente la capa oculta y las conexiones de sus neuronas con las entradas. En el lado del decodificador estará la capa de salida y las conexiones que en la red original tenía con la capa oculta.

En [52] se analiza la operación de la red, obteniéndose interesantes conclusiones. Los autores observan que la red produce un conjunto de imágenes base, una para cada neurona de la capa oculta. Posteriormente demuestran que estas imágenes base tienden a generar las primeras M componentes principales de la imagen, siendo M el número de neuronas ocultas. Cuando se usa como función de activación una función lineal, las imágenes base generan exactamente las componentes principales. Sin embargo, a diferencia de lo que ocurre con las varianzas de los coeficientes de las componentes principales, las varianzas de los valores producidos por las neuronas ocultas tienden a ser iguales. Además, la contribución de las neuronas ocultas parecen distribuirse uniformemente, a diferencia de lo que sucede en el análisis de las componentes principales, en el cual la contribución del error decrece según el valor del autovalor asociado.

Variaciones a partir del esquema CMZ

Como derivaciones del esquema CMZ, se desarrollan otros que incorporan variaciones en la estructura de la red o en los algoritmos de aprendizaje. Asimismo, se desarrollan sistemas basados en esta técnica con miras a campos de aplicación específicos.

En cuanto a las variaciones en la estructura de la red, una de ellas consiste en incluir más de una capa oculta. En esta línea, Abdel y Fahmy [53] experimentan con redes de dos (64-6-64), cuatro (64-6-6-6-64) y seis capas (64-6-6-6-6-6-64). Así, la razón de compresión es igual a 64:6 y usando funciones de activación no lineales obtienen valores para el MSE de 147,3 para el MLP de dos capas, 102,8 para el de cuatro capas y 84,15 para el de seis capas. Estos resultados se obtienen al aplicar los compresores sobre imágenes no incluidas en el conjunto de entrenamiento. Ya que el mayor tamaño de la red trae consigo un aumento considerable en el tiempo de entrenamiento, en el trabajo utilizan una modificación del algoritmo backpropagation que consiste en propagar el error hacia atrás sólo si su valor está por encima de un umbral que va descendiendo a medida que avanza el entrenamiento. También aplican un escalamiento a los valores de los pixeles con la finalidad de enfatizar el error en las zonas oscuras.

Otra línea consiste en el empleo de redes CMZ en paralelo, cada red con distinta relación N/M, con lo cual bloques comprimidos por diferentes redes tendrán diferente relación de compresión. El empleo de redes en paralelo tiene como propósito el que los bloques correspondientes a los bordes de la imagen se compriman mediante redes con menor relación N/M y los bloques de las regiones planas lo hagan con redes con mayor N/M. En [54] se propone una estructura de redes en paralelo, con complejidad computacional reducida. La estructura consta de cuatro redes CMZ, todas para el mismo tamaño de bloque ($N = 64$) y valores para M de 4, 8, 12 y 16; el trabajo expone el algoritmo para la selección óptima de la red para la compresión de cada bloque a fin de obtener una tasa de compresión fija. Para una tasa de compresión de 8:1, se obtuvo un valor de 30,5 dB para el PSNR al comprimir la imagen Lena, superando en más de 12 decibelios el resultado de Cottrell et al [51]. Para otras imágenes los resultados son en todos los casos desfavorables al compararlos con los obtenidos por medio

del algoritmo JPEG.

En la misma línea consistente en redes en paralelo con distinta relación N/M, sólo que ahora teniendo como objetivo segmentar la imagen en bloques de varios tamaños, Watanabe y Mori [55] introducen una estructura de múltiples redes neuronales, cada una con número diferente de unidades de entrada, pero todas con la misma cantidad de neuronas en la capa oculta. De este modo, las redes con menor tamaño de bloque se aplican para comprimir las regiones de borde y de textura y las de mayor tamaño de bloque se utilizan para comprimir las regiones planas. Usando redes para bloques de 1, 4, 16 y 64 pixeles, con razón de compresión igual a 8:1, el valor del PSNR obtenido al emplear la imagen Lena tanto para el entrenamiento como para la prueba es de 35,05 decibelios. Con la misma estructura, para imágenes diferentes a Lena, que es la empleada para el entrenamiento, se obtienen valores para el PSNR que van de 28,77 a 33,64 decibelios.

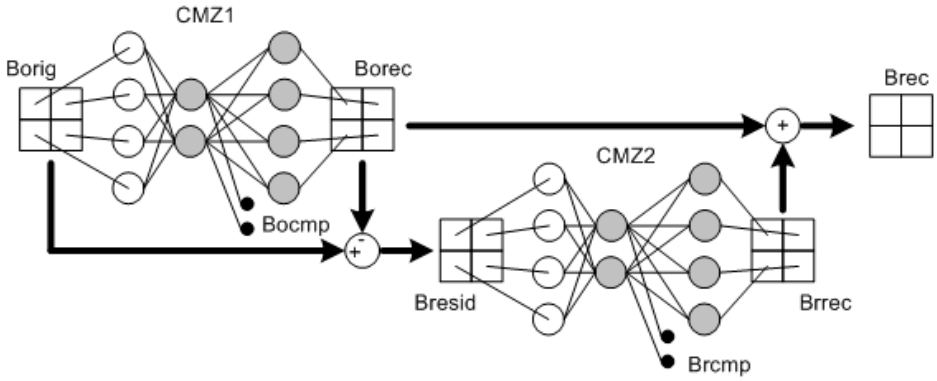


Fig. 3.12: Esquema cuello de botella en dos etapas.

En otra vertiente, se tiene el trabajo de Charif y Salam [56], que utiliza dos redes CMZ, una para comprimir la imagen original (CMZ1) y la otra (CMZ2) para comprimir la imagen residual que resulta al reconstruir la imagen generada por el primer compresor, Fig. 3.12. Una vez que concluye el entrenamiento, el sistema compresor estará constituido por el codificador y el decodificador de CMZ1 y por el codificador de CMZ2; así, cuando se presenta al sistema una imagen, ésta se codifica mediante el codificador de CMZ1 y después se recupera mediante el decodificador de la misma, después la diferencia entre la imagen original y la imagen recuperada se comprime por el codificador de CMZ2. El sistema de descompresión está constituido por los dos decodificadores, el de la CMZ1 y el de la CMZ2. La imagen comprimida generada por el codificador de la CMZ1 se decodifica mediante el decodificador de esa misma red. Igualmente, el error comprimido se descomprime usando el decodificador de CMZ2. Finalmente, los dos resultados se suman. Trabajando con bloques de 16 x 16 pixeles ($N = 256$ para las dos redes), $M = 4$ para CMZ1 y $M = 12$ para CMZ2, para una razón de compresión igual a 16:1, la calidad del compresor se evalúa empleando como medida el SNR[13], obteniendo un valor de 7,4 cuando el sistema se aplicó sobre la imagen usada también para el entrenamiento.

La Tabla 3.2 concentra los resultados y algunas observaciones para las aplicaciones des-

[13]Por la sigla de Signal to Noise Ratio, medida que se calcula mediante la expresión $SNR = \frac{\|imagen\ original\|}{\|imagen\ original - imagen\ reconstruida\|}$, donde $\|\bullet\|$ es la norma euclidiana. El uso en este caso de la métrica SNR no permite comparar sus resultados con los de la mayoría, ya que éstos usan el PSNR

Referencia	Capas	Etapas	CR	Calidad	Observaciones
Abedl[53]	5 y 7	Ninguna	64:6	PSNR=28,01 dB con 5 capas, 28,88 dB con 7	Una sola red con capas múltiples
Benbenisti[54]	3	Ninguna	8:1	Valores de PSNR que van de 12,8 a 30,5 dB	Varias redes en paralelo, cada bloque se comprime con la red más adecuada para minimizar el error con una tasa de compresión fija
Watanabe[55]	3	Ninguna	8:1	Con 3 redes PSNR entre 28,95 y 33,17 dB; con 4, entre 28,77 y 34,48	La red con mayor razón de compresión comprime las regiones planas, los bordes con la de menor compresión
Charif[56]	3	Ninguna	16:1	SNR = 7,5 con dos redes, con una sola 6,3	2 redes, una codifica la imagen y otra el error
Steudel[57]	3	Cuantización escalar	21,3:1 8:1	NMSE=0,789%; NMSE=0,18%	Enfocado a imágenes médicas

Tabla 3.2: MLP para transformación autoasociativa

critas del MLP como compresor de imágenes mediante la transformación autoasociativa.

3.3.3 Codificación predictiva mediante redes neuronales

En el Capítulo dos se expuso la codificación predictiva como una técnica encaminada a reducir la correlación entre pixeles. Los predictores lineales no aprovechan cabalmente la correlación de mayor orden que existe en las imágenes. Esto ocasiona un pobre rendimiento en las regiones que contienen bordes y texturas, por lo cual se justifica el empleo de predictores no lineales. Sin embargo, los métodos convencionales para el diseño de predictores no lineales son generalmente complicados. Las redes neuronales proporcionan una buena solución a tales problemas difícilmente tratables matemáticamente, esto debido a que las redes están basadas en el aprendizaje y, por consiguiente, no requieren establecer modelos estadísticos para las fuentes de datos.

Un predictor tiene como propósito estimar el valor de un píxel como una función que se evalúa con los valores de un grupo de píxeles vecinos previamente estimados. A ese grupo se le denomina ventana de contexto o simplemente contexto. Así, la predicción usa la siguiente ecuación:

$$\hat{x}(r,s) = P[x(r-i, s-j), (i,j) \in C], \tag{3.11}$$

siendo $x(r,s)$ el valor real del pixel o señal, $\hat{x}(r,s)$ el valor estimado por el predictor, P la función de predicción y C el contexto.

El MLP como predictor no lineal

El perceptrón multicapa se puede utilizar como un predictor no lineal. La entrada a la red son los pixeles almacenados en la memoria del contexto. La salida es el valor de la predicción. De este modo, el número de unidades de entrada es igual al tamaño del contexto y hay una sola neurona en la capa de salida. El número de capas ocultas y la cantidad de neuronas en cada una de ellas es variable. En la Fig. 3.13 se muestra el esquema de un predictor no lineal basado en un MLP 4-7-1.

Revisión de trabajos de predicción no lineal basada en NNs

El empleo de las redes neuronales en predictores no lineales se remonta a los trabajos de Nasrabadi, Dianat y Venkataraman [58] y Manikopoulus [59], mismos que tienen como antecedente los esquemas no neuronales DPCM. Las investigaciones en este campo exploran alternativas en cuanto a la aplicación de los predictores directamente sobre los píxeles de la imagen o bien sobre coeficientes obtenidos mediante alguna transformación de la misma, el diseño del contexto en cuanto a forma y tamaño, predicción vectorial y empleo de redes recurrentes.

En [60], los autores incorporan un predictor no lineal en un sistema de compresión de imágenes basado en la descomposición wavelet. Utilizan un contexto de vecindad de píxeles dentro del mismo nivel (intra-scale). Experimentan con dos estructuras de red, una totalmente conectada y una convencional y prueban contextos rectangulares y piramidales. La red se entrena usando un conjunto fijo de imágenes y los pesos de la red así obtenidos se incluyen como parte del codificador/decodificador. Experimentan con diferentes tamaños de la capa

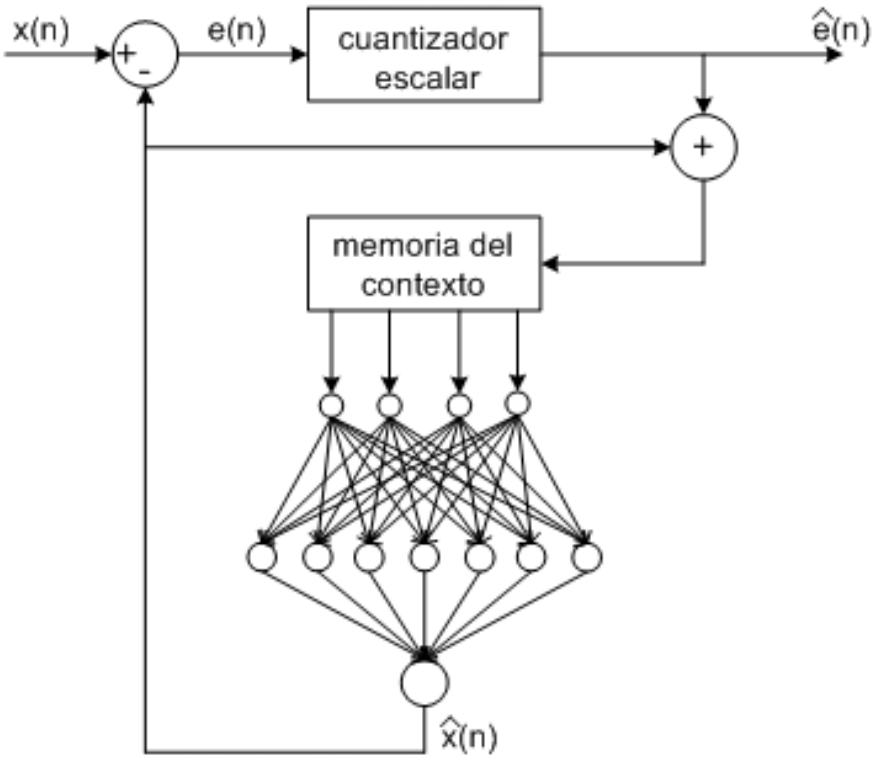


Fig. 3.13: Codificación basada en un MLP como predictor no lineal

oculta. Utilizan los contextos formados con los coeficientes wavelet, compensando así la cuantización en el aprendizaje.

En [61], se expone un codificador DPCM con un predictor que utiliza una red neuronal bilineal recurrente (BLRNN), con 3 unidades de entrada feed-forward, 3 unidades de entrada para la sección recurrente y una unidad de entrada para la sección bilineal, además de una neurona en la capa intermedia y una neurona en la capa de salida. Como función de activación usan la tangente inversa. Utilizan los 3 píxeles contiguos para el contexto y un cuantizador de 4 bits. La red BLRNN se entrena con los datos obtenidos cada vez que la ventana de contexto se desplaza hacia la derecha. Después del entrenamiento la imagen completa produce el conjunto de pesos para la red BLRNN y entonces la red se integra al codificador.

En [62][63] se presenta un sistema adaptativo no lineal para la codificación predictiva de imágenes usando un perceptrón multicapa, para un compresor sin pérdidas. El entrenamiento se realiza usando un grupo de 12 contextos tomados de un área de entrenamiento localizada en la proximidad del píxel que está siendo codificado, en lugar de realizar el entrenamiento con datos separados. Cada contexto está formado, a su vez, por los 12 píxeles contiguos. En [64] se realiza la implementación hardware basada en una FPGA de un predictor con esta estructura.

En [65] los autores emplean el esquema PVQ (Predictive Vector Quantization), mediante el cual un predictor estima el bloque actual a partir de los bloques previos y posteriormente el vector residual se cuantiza vectorialmente, usando un diccionario relativamente pequeño. En el trabajo se utiliza el contexto de bloques que se muestra en la Fig. 3.14.

En la Tabla 3.3 se recopilan algunas características de los predictores no lineales basados

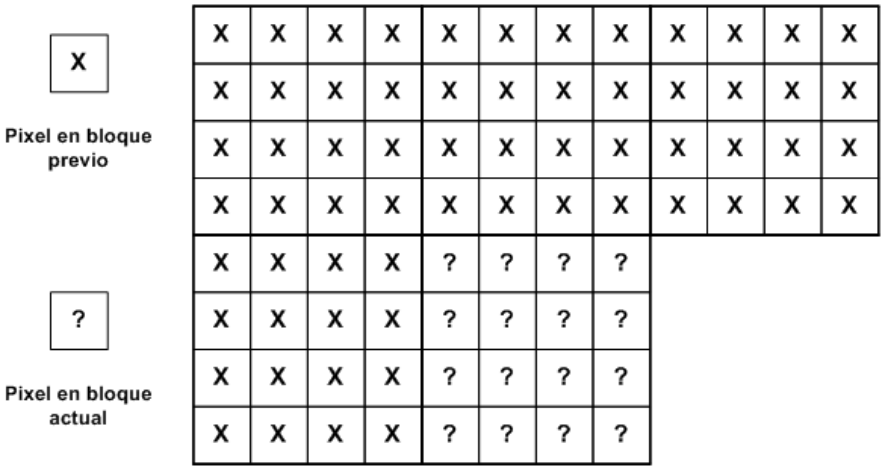


Fig. 3.14: Contexto para la predicción vectorial

en redes MLP.

3.4 Redes SOM para cuantización vectorial

3.4.1 Redes competitivas

Para las redes con aprendizaje competitivo se considera que las neuronas compiten unas con otras con el fin de llevar a cabo una tarea determinada. Con este tipo de aprendizaje se pretende que cuando se presente a la red cierta información de entrada, sólo se active una de las neuronas de salida de la red, o una por cierto grupo de neuronas. Por lo tanto, las neuronas compiten para activarse quedando finalmente una, o una por grupo, como neurona vencedora, y el resto quedan inactivas. En una red competitiva multicapa la competición entre neuronas se realiza en todas las capas de la red, existiendo en estas redes neuronas con conexiones de autoexcitación (signo positivo) y conexiones de inhibición (signo negativo) respecto a las neuronas vecinas.

El objetivo del aprendizaje competitivo es formar categorías[14] con base en los datos que se introducen en la red para su entrenamiento. De esta forma, datos que son similares se clasifican de modo que forman parte de la misma categoría y por tanto deben activar la misma neurona de salida. Debido a que se trata de un aprendizaje no supervisado que opera con base en la correlación entre los datos de entrada, las clases o categorías se deben crear mediante la propia red.

A principios de 1959, Frank Rosenblatt [37] creó su *clasificador simple espontáneo*, una red con aprendizaje no supervisado basado en el perceptrón, el cual aprendía a clasificar vectores de entrada en dos clases con igual número de términos.

Christoph Von Der Malsburg [66] introdujo el principio de mapas auto-organizativos en 1973, que permitió a las redes clasificar entradas de modo que las neuronas situadas en un vecindario cercano a la neurona ganadora, respondieran a entradas similares. La topología de esta red imitaba de alguna forma las estructuras encontradas en la corteza visual de los

[14] Comúnmente se emplea clusters, por el término en inglés

Referencia	Red	Transformación	Cuantización	Codificación	CR	Calidad
Burges[60]	MLP varias	Wavelet	Ninguna	Ninguna	Diversas comparativas con resultados relativos	
Park[61]	Bilineal recurrente	Ninguna	Escalar, 4 bits	Ninguna	2:1	PSNR = 39,2 dB
Marusic[62]	MLP 12-10-1	Ninguna	Ninguna	Aritmética	1,84:1	Sin pérdidas, predictor adaptativo
Marusic[63]	MLP 12-10-1 Recursiva	Ninguna	Ninguna	Aritmética	2,10:1 2,09:1	Sin pérdidas, predictor adaptativo
Gadea[64]	MLP 12-10-1	Ninguna	Ninguna	Ninguna	Compara entropia de imagen original e imagen residual, implementación hardware	
Rizvi[65]	MLP 64-30-16 Pred. vectorial	Ninguna	Vectorial	Ninguna	33:1	PSNR = 30,57 dB

Tabla 3.3: MLP para predicción no lineal

gatos, estudiada por David Hubel y Torten Wiesel [67]. La regla de aprendizaje generada en base a tal principio produjo gran interés.

Por otra parte, en los años ochenta, principalmente como fruto de los trabajos de Grossberg y Carpenter, ciertos principios que se derivaron a partir del análisis experimental del sistema de visión humano, del sistema de lenguaje, del desarrollo cortical y del refuerzo del aprendizaje, condujeron a proponer la *resonancia adaptiva* como una teoría cognoscitiva para el procesamiento de información. La teoría de resonancia adaptiva evolucionó hacia una serie de modelos de redes neuronales, con aprendizaje supervisado o no supervisado, para reconocimiento de patrones y predicción. Los modelos con aprendizaje no supervisado incluyen las redes ART1 [68] para patrones de entrada binarios y ART2 [69] para patrones de entrada analógicos. El modelo ARTMAP [70] combina dos módulos con aprendizaje no supervisado para llevar a cabo aprendizaje supervisado.

Uno de los aspectos centrales de todos los sistemas ART es un proceso de concordancia de patrones que compara una entrada externa con los códigos activos en la memoria interna. La búsqueda de concordancia conduce ya sea hacia un estado resonante, que persiste el tiempo suficiente para su aprendizaje, o a la creación de un nuevo código. Si la búsqueda concluye en un código establecido, la representación en memoria puede permanecer sin cambio o bien incorporar información nueva a partir de las partes coincidentes. Si la búsqueda termina en un código nuevo, la representación en memoria aprende la entrada actual. Este proceso de aprendizaje basado en concordancia es el fundamento de la *estabilidad*[15] de los modelos ART. El aprendizaje basado en concordancia permite que la memoria cambie sólo cuando la entrada externa es suficientemente próxima a los códigos internos, o cuando ocurre algo completamente nuevo.

Las redes competitivas que sin duda han encontrado mayor aceptación en cuanto a aplicación se refiere, son las que se han derivado a partir de los esquemas y algoritmos desarrollados por Teuvo Kohonen a lo largo de los últimos 20 años. Son frecuentes las referencias a las *redes de Kohonen*, aunque en realidad son tres los esquemas fundamentales que él ha desarrollado, siendo éstos:

1. VQ: Red competitiva para cuantización vectorial. Esta es una red autoasociativa estrechamente relacionada con los clasificadores k-media. Es una red monocapa con aprendizaje no supervisado. Cada neurona corresponde a una partición del espacio k-dimensional cuyo centroide es el vector código constituido por los pesos de las conexiones de esa neurona. La Fig. 3.15 muestra su arquitectura; las conexiones laterales en el modelo conceptual de la neurona competitiva son inhibitorias, provienen de y van hacia las restantes neuronas de la misma capa.

   El algoritmo de aprendizaje consiste en encontrar el vector código más cercano al vector de entrada, para así determinar cuál es la neurona ganadora. Como segundo paso, el vector código de la neurona ganadora se modifica para acercarlo al vector de entrada. El vector código se mueve cierta porción de la distancia que lo separa del vector de entrada. Esa porción está determinada por la razón de aprendizaje, como lo establece la siguiene ecuación:

$$W_{ng}(t+1) = W_{ng}(t) + \alpha(W_{ng}(t) - X(t)), \tag{3.12}$$

[15]Una característica fundamental de la memoria humana consiste en su habilidad para aprender nuevos conceptos (plasticidad) sin necesidad de olvidar por ello otros aprendidos en el pasado (estabilidad)

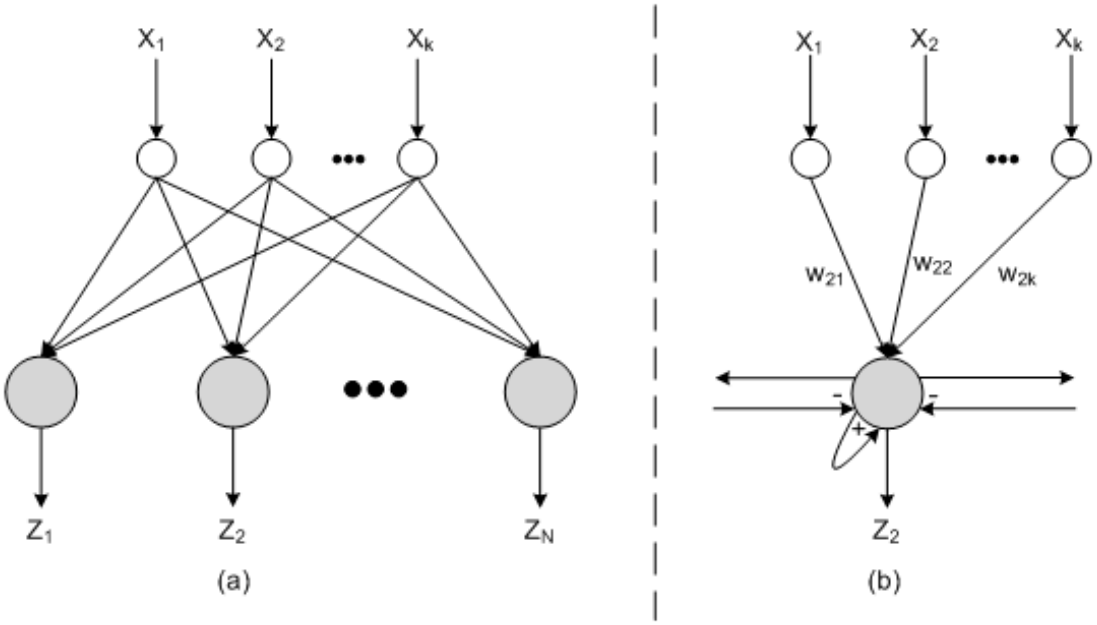


Fig. 3.15: Estructura de una red de Kohonen tipo VQ, (a) Arquitectura para la operación en la fase de recall, (b) Modelo conceptual de una neurona durante el entrenamiento

siendo $W_{ng}(t)$ y $W_{ng}(t+1)$, respectivamente, el valor actual y el valor nuevo del vector formado por los pesos de las conexiones incidentes sobre la neurona ganadora (es decir el vector código del cluster que corresponde a la neurona ganadora), $X(t)$ el vector de entrada aplicado y α la razón de aprendizaje.

La determinación del vector código más cercano al de entrada se realiza mediante el cálculo de la distancia euclidiana entre ambos.

2. LVQ[16]: Es una red competitiva para clasificación supervisada [71]. Es una red de dos capas para aprendizaje supervisado. La capa oculta es de tipo competitivo, por lo cual lleva a cabo una autoasociación de los vectores de entrada de la misma manera como lo hace una red VQ. La capa de salida es de tipo lineal. Por cada neurona en la capa oculta se tiene un vector código para el cuantizador vectorial. A la vez, por cada neurona de la capa de salida se tiene una clase. La cantidad de neuronas en la capa de salida es menor que la de la capa oculta. De este modo, cada vector código queda asociado a alguna de las clases, mientras que a cada clase le corresponde uno o más vectores código. Un vector de entrada se clasifica encontrando el vector código al que es más cercano y asignándole la clase en la que éste está ubicado. La Fig. 3.16 muestra la estructura de este tipo de red.

3. SOM[17]: Las redes de este tipo aprenden a clasificar vectores de entrada de acuerdo a la manera como éstos se encuentran localizados en el espacio de entrada. Las redes

[16]Por la sigla de Learning Vector Quantization
[17]En la literatura es común encontrar indistintamente SOFM y SOM

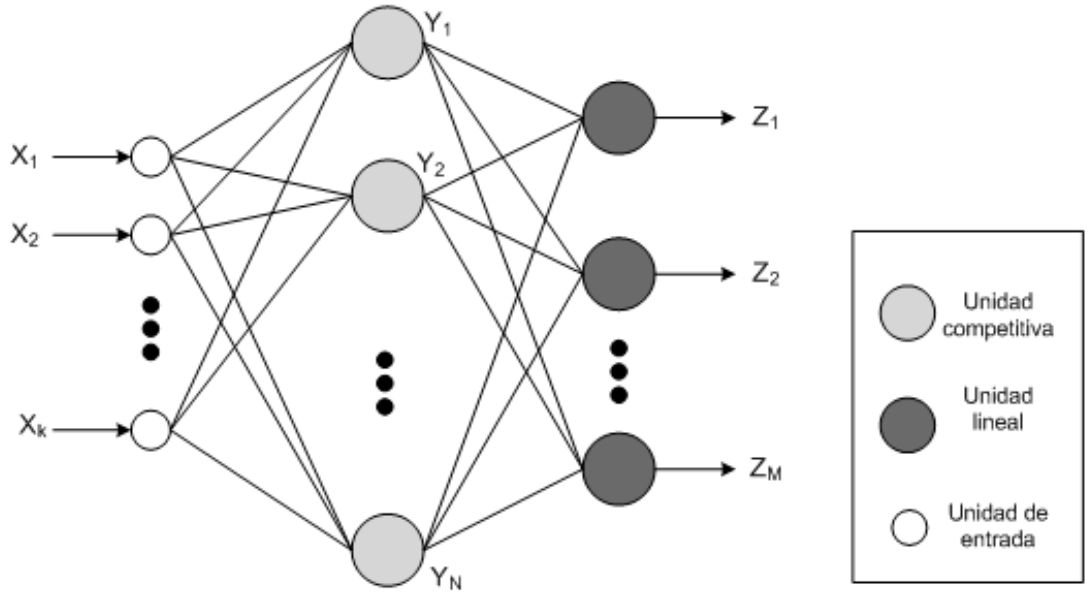


Fig. 3.16: Estructura de una red de Kohonen LVQ

SOM difieren de las redes VQ en cuanto a que neuronas que son vecinas en el mapa autoorganizado aprenden a reconocer regiones que son vecinas en el espacio de entrada. De este modo, las redes SOM aprenden durante el entrenamiento tanto la distribución (como lo hacen las redes VQ) como la topología de los vectores de entrada. Por tanto, el aspecto geométrico de la disposición de neuronas de una red es la base de las redes SOM. El siguiente apartado estará dedicado a este tipo de redes.

La Fig. 3.17 ilustra la arquitectura de las redes SOM. Las neuronas de salida forman una estructura geométrica de modo que se puede establecer una función de vecindad entre ellas. Vectores de entrada que son cercanos unos a otros activan neuronas vecinas.

3.4.2 El algoritmo SOM

Fundamento fisiológico de las redes SOM

Se ha observado que en el córtex de los animales superiores aparecen zonas donde las neuronas detectoras de rasgos se encuentran topológicamente ordenadas [72], de forma que la información captada desde el entorno a través de los órganos sensoriales se representa internamente en forma de mapas bidimensionales. Por ejemplo, en el área somatosensorial, las neuronas que reciben señales de sensores que se encuentran próximos en la piel se sitúan también próximas en el córtex, de manera que reproducen, de forma aproximada, el mapa de la superficie de la piel en una zona de la corteza cerebral. En el sistema visual se han detectado mapas del espacio visual en zonas del cerebro. Por lo que respecta al sentido del oído, existen en el cerebro áreas que representan mapas tonotópicos, donde los detectores de determinados rasgos relacionados con el tono de un sonido se encuentran ordenados en dos dimensiones.

Aunque en gran medida esta organización neuronal está predeterminada genéticamente, es probable que parte de ella se origine mediante el aprendizaje. Esto sugiere, por tanto, que

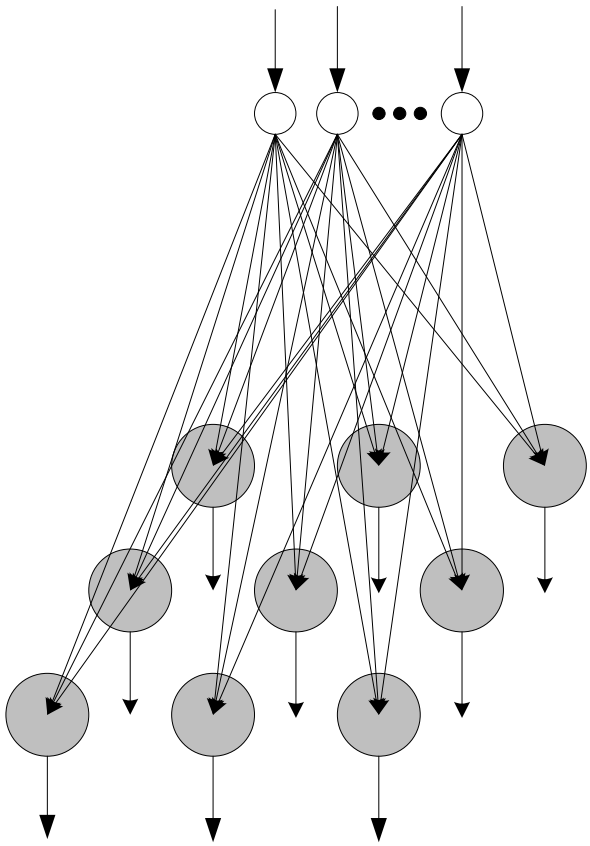


Fig. 3.17: Estructura de una red de Kohonen tipo SOM

el cerebro podría poseer la capacidad inherente de formar mapas topológicos de la información recibida del exterior [73].

Por otra parte, también se ha observado que la influencia que una neurona ejerce sobre las demás es función de la distancia entre ellas, siendo muy pequeña cuando están muy alejadas. Así, se ha comprobado que en determinados primates se producen interacciones laterales de tipo excitatorio entre neuronas próximas en un radio de 50 a 100 micras, de tipo inhibitorio en una corona circular de 150 a 400 micras de anchura alrededor del círculo anterior, y de tipo excitatorio muy débil, prácticamente nulo, desde ese punto hasta una distancia de varios centímetros. Este tipo de interacción tiene la forma típica de un sombrero mexicano, tal como se muestra en la Fig. 3.18.

Teniendo como punto de partida tales comportamientos observados, el modelo de red autoorganizado presentado por Kohonen pretende emular de forma simplificada la capacidad del cerebro para elaborar mapas topológicos a partir de las señales recibidas del exterior.

Derivación del algoritmo

Una red SOM se puede describir formalmente como una asociación suave, ordenada y no lineal entre datos de entrada de dimensión grande y los elementos de un arreglo regular con menor dimensión. Esta asociación, la cual se asemeja a la cuantización vectorial clásica, se implementa de la manera que se describe a continuación.

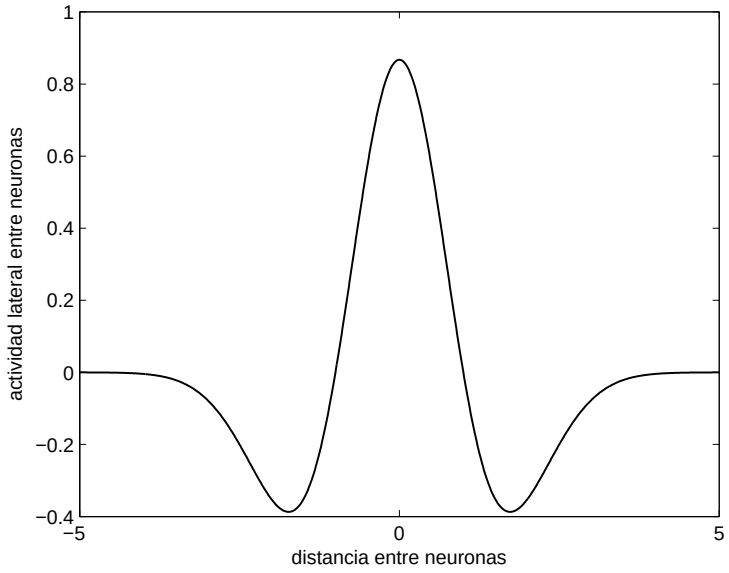


Fig. 3.18: Interacción lateral entre neuronas

Primero, se parte de que el conjunto de variables de entrada $\{\xi_j\}$ se puede definir como un vector real $x = [\xi_1, \xi_2, \ldots, \xi_n]^T \in \Re^n$ y considerando que a cada elemento en el arreglo SOM se puede asociar un vector real paramétrico $m_i = [\mu_{i1}, \mu_{i2}, \ldots, \mu_{in}]^T \in \Re^n$ al que se le puede denominar *modelo*. Se supone, además, que existe una medida para la distancia entre x y m_i representada por $d(x, m_i)$. Entonces, la imagen de un vector de entrada x en el arreglo SOM se define como el elemento m_c del arreglo que mejor concuerda con x, es decir, aquél que tiene como índice:

$$c = arg \min_i d(x, m_i). \tag{3.13}$$

A diferencia de la cuantización vectorial clásica, el algoritmo SOM tiene como tarea definir m_i de manera tal que el mapeo sea ordenado y descriptivo de la distribución de x. Se puede tratar de determinar m_i mediante un proceso de optimización siguiendo la idea de la cuantización vectorial clásica. En ese caso, se coloca un conjunto finito de vectores código $\{m_i\}$ en el mismo espacio de las señales x con el propósito de aproximarlas. Si $p(x)$ es la función de densidad probabilística de x y si m_c es el vector más cercano a x en el espacio de la señal (o sea que $d(x, m_c)$ es la distancia más pequeña), entonces el cuantizador vectorial minimizará la función para el valor medio esperado de la función de error dada por:

$$E = \int f\left[d(x, m_c)\right] p(x) dx, \tag{3.14}$$

donde f es alguna función monótonamente creciente de la distancia d.

El índice c también es una función de x y de todos los m_i, por lo que el integrando de E no es continuamente diferenciable (c cambia abruptamente cuando se cruza una frontera en el espacio de la señal en la cual dos vectores de código dan lugar a un mismo valor para la función distancia). Por tanto la minimización de E puede conducir a un tratamiento complicado.

Para simplificar el proceso de minimización, la función de error de cuantización se expresa de manera discreta, quedando así:

$$E_d = \sum_i h_{ci} f\left[d(x(t), m_i(t))\right], \tag{3.15}$$

siendo h_{ci} una función para suavizar la distancia entre las unidades c e i del arreglo y $m_i(t)$ es la aproximación de m_i en el tiempo t.

De este modo, el algoritmo para la optimización queda expresado de la siguiente manera:

$$m_i(t+1) = m_i(t) - \frac{1}{2}\lambda(t)\frac{\partial E_d}{\partial m_i(t)}, \tag{3.16}$$

donde $\lambda(t)$ es un factor escalar positivo pequeño que determina el paso del gradiente en el tiempo t. Si la función $\lambda(t)$ se selecciona apropiadamente, la secuencia de los valores de $m_i(t)$ converge a la solución para $\{m_i\}$.

A partir de la Ec. 3.16 se pueden derivar diferentes algoritmos SOM. Por ejemplo, si $d(x, m_i) = \|x - m_i\|$ y $f(d) = d^2$, donde $\|\bullet\|$ es la norma Euclidiana, se obtiene el algoritmo tradicional SOM, que queda expresado así:

$$m_i(t+1) = m_i(t) + h_{ci}(t)\left[x(t) - m_i(t)\right], \tag{3.17}$$

en donde $\lambda(t)$ se ha combinado con $h_{ci}(t)$. La definición más simple, pero aun más efectiva, para $h_{ci}(t)$ es: $h_{ci}(t) = \alpha(t)$, donde $\alpha(t)$ es otro parámetro escalar pequeño, si la distancia (radio de vecindad) entre las unidades i y c es menor o igual que cierto límite especificado, o $h_{ci}(t) = 0$ en caso contrario.

En el transcurso del proceso de ordenamiento, tanto $\alpha(t)$ como el radio decrecen monótonamente con el tiempo, con $0 < \alpha(t) < 1$. Si $N_c(t)$ es la región donde se encuentran las neuronas vecinas a la ganadora cuyo valor se debe actualizar, entonces el algoritmo SOM más simple queda definido por la siguiente ecuación:

$$m_i(t+1) = \begin{cases} m_i(t) + \alpha(t)\left[x(t) - m_i(t)\right] & i \in N_c(t) \\ m_i(t) & i \notin N_c(t) \end{cases}. \tag{3.18}$$

El algoritmo SOM en su forma operacional

En resumen, las redes SOM son de una sola capa, del tipo competitivo, con entrenamiento no supervisado y se caracterizan porque sus neuronas conforman un arreglo generalmente bidimensional, aunque también puede ser unidimensional o incluso tridimensional. La topología del arreglo se establece al definir cierta función para calcular la distancia que separa geométricamente cada par de neuronas. En la Fig. 3.19 se muestran dos arreglos bidimensionales SOM con diferente topología, rectangular para el de la izquierda y hexagonal para el de la derecha. Con línea gruesa punteada se señalan las neuronas situadas dentro de la región de vecindad de la neurona 1 con radio igual a uno y con línea simple punteada las que se encuentran en su vecindad de radio dos. En el arreglo con topología rectangular, la región de vecindad con radio uno contiene 4 neuronas y la de radio dos contiene 12; mientras que para la topología hexagonal, esos mismos valores son de 6 y 18 neuronas respectivamente.

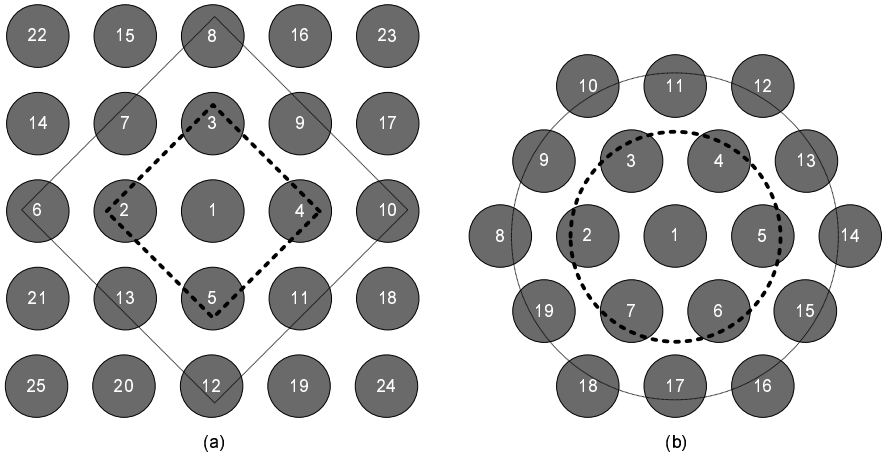


Fig. 3.19: Topologías para los mapas SOM

El algoritmo para el entrenamiento SOM es iterativo. El algoritmo consiste de los siguientes pasos[18]:

1. Iniciar los pesos de todas las conexiones de la red con valores generados aleatoriamente.
2. Iniciar la razón de aprendizaje (α) y el factor de vecindad (η) con valores comprendidos en el rango entre cero y uno.
3. Iniciar el radio de la región de vecindad con un valor convenientemente grande.
4. Iniciar $error = 0$
5. Para cada vector del conjunto de entrenamiento $x_j,\ j \in \{1, 2, \ldots, T\}$:
   - 5.1. Iniciar d_{min} igual a un valor suficientemente grande
   - 5.2. Para cada vector código $W_i,\ i \in \{1, 2, \ldots, N\}$
     - 5.2.1. Calcular la distancia $d_{ij} = d(m_i, x_j)$
     - 5.2.2. Si $d_{ij} < d_{min}$
       - $d_{min} = d_{ij}$
       - $g = i$
   - 5.3. $m_g = m_g + \alpha(x_j - m_g)$
   - 5.4. $m_k = m_k + \alpha\eta(x_j - m_k)$ para $k \neq g$ y k dentro de la región de vecindad.
   - 5.5. $error = error + (x_j - m_g)^2$
6. Reducir α, η y el radio de la región de vecindad.
7. Si $error \leq er_{req}$
   - repetir desde 4.
8. Finaliza

[18]El conjunto de entrenamiento consta de T vectores y la red tiene N neuronas

Los pasos 1 a 3 se realizan una sola vez y tienen como propósito establecer los valores iniciales tanto para los pesos de la red como para los parámetros del entrenamiento, siendo éstos la tasa de aprendizaje y el factor de vecindad. Los pasos 4 a 7 se ejecutan una vez por cada epoch que requiera el entrenamiento. En cada epoch se inicia en cero el valor del error (paso 4) y luego para cada vector de entrenamiento se calcula su distancia respecto a cada vector código (5.2.1); cada distancia calculada se compara con el valor mínimo (d_{min}) que se tiene hasta el momento y si es menor se actualiza tanto d_{min} como el índice de la neurona ganadora (5.2.2); posteriormente se ajusta el peso de la neurona ganadora (5.3) y el de las neuronas ubicadas dentro de su región de vecindad (5.4); finalmente se acumula al error el valor cuadrático de la diferencia entre el vector de entrada y el vector código asociado a la neurona ganadora. Cuando se han aplicado todos los vectores del conjunto de entrenamiento, es decir cada vez que finaliza un epoch, se reducen los parámetros y la región de vecindad. Finalmente se revisa el criterio de terminación del algoritmo, estando éste determinado por la comparación del error acumulado con un valor requerido (er_{req}); si el error calculado es mayor que el requerido, el proceso se repite a partir del paso 4 para llevar a cabo un nuevo epoch.

3.4.3 Variantes

Distancias distintas a la euclidiana

Para obtener una representación ordenada y organizada del conjunto de datos en una red neuronal, el principio esencial que utiliza la SOM es permitir que el aprendizaje de cada neurona dependa de la activación de las neuronas vecinas. Esta activación depende de la determinación de la neurona que mejor representa a un dato de entrada, que a su vez está en función de la métrica de distancia que se esté utilizando.

De tal manera, se pueden obtener variantes del algoritmo si en lugar de utilizar la distancia Euclidiana se utilizan otros tipos de funciones de distancia. El algoritmo SOM para una métrica de distancia generalizada queda expresado de la siguiente manera:

$$m_i(t+1) = m_i(t) - \lambda \nabla_{m_i(t)} e(t), \tag{3.19}$$

donde $e(t)$ es la medida de distorsión definida así:

$$e(t) = \sum_{i \in L} h_{ci}(t) d(x(t), m_i(t)), \tag{3.20}$$

siendo L el conjunto de índices para todas las neuronas del arreglo.

En el caso particular de la distancia Manhattan, la actualización de los pesos para cada componente de los vectores queda expresada así:

$$\mu_{ij}(t+1) = \mu_{ij}(t) + \alpha(t) h_{ci}(t) sgn\left[\xi_j(t) - \mu_{ij}(t)\right], \tag{3.21}$$

siendo μ_{ij} la componente j del vector de pesos m_i y $sgn\left[\bullet\right]$ el signo de su argumento.

Versión Batch

A la versión original del algoritmo SOM que se expuso en la sección 3.3.2 se le puede considerar como SOM adaptiva o incremental, ya que a cada patrón de entrenamiento corresponde una actualización de los pesos para la neurona ganadora y las que se encuentren dentro de su región de vecindad. En la versión batch del algoritmo SOM la actualización se lleva a cabo después de que se han determinado las unidades ganadoras para todos los patrones.

Para el algoritmo Batch SOM, partiendo de cierta asignación inicial para los valores de los vectores de pesos, en cada epoch, como primer paso, se forman las regiones de Voronoi.[19] A continuación se calcula el centroide de cada región, es decir el promedio de los patrones de entrada incluidos en esa región. Finalmente se ajustan los vectores de pesos por medio de:

$$W_i = \frac{\sum_j n_j h_{ji} \bar{X}_j}{\sum_j n_j h_{ji}}, \tag{3.22}$$

donde n_j es el número de patrones de entrada que mapean en la neurona j, los valores del índice j son los que corresponden a todas las neuronas situadas en la región de vecindad de la neurona i y $\bar{X}_j$ es el centroide de la región de Voronoi j.

SOM con aprendizaje supervisado

El algoritmo SOM se puede aplicar directamente para la clasificación de un conjunto de datos. Sin embargo, en muchas aplicaciones se requiere que el algoritmo sea capaz de reconocer patrones que se encuentran dentro de un conjunto de clases previamente determinado. Para estos casos es necesario emplear una red neuronal con aprendizaje supervisado. Por tal motivo, aunque las SOM fueron destinadas originalmente para tareas de clasificación no supervisada, se observó que la separabilidad de clases se puede mejorar considerablemente si se toma en cuenta durante el entrenamiento la información relativa a la identidad de clase de los patrones que se utilicen durante ese proceso.

Esta idea se implementó en [74]. Para esto, los vectores de entrada se extienden de modo que queden compuestos por dos partes, X_s y X_u, donde X_s es el patrón original de entrada y X_u es un vector con N elementos, siendo N la cantidad de clases. Si un patrón X_s pertenece a la clase j, entonces el elemento j de X_u será igual a 1 y los elementos restantes serán 0. Así, los vectores de entrada estarán dados por la siguiente expresión:

$$X = \left[X_u^T, \epsilon X_s^T\right]^T, \tag{3.23}$$

donde ϵ es una constante con valor pequeño, típicamente igual a 0,1.

Debido a que los componentes X_u son iguales para vectores que están en la misma clase y diferentes si están en clases distintas, se incorpora indirectamente el aprendizaje supervisado. Los vectores de pesos también deben consistir de dos partes, siendo éstas W_s y W_u de modo que $W = \left[W_u^T, W_s^T\right]^T$. En la fase de recall, para clasificar un vector de entrada X, éste se compara con los componentes W_s de los vectores de pesos.

Como se mencionó anteriormente, existe además el algoritmo LVQ propuesto por Kohonen que está destinado expresamente a clasificación supervisada. A su vez, el algoritmo LVQ tiene tres variantes conocidas como LVQ1, LVQ2 y LVQ3.

[19] La región de Voronoi correspondiente a la neurona k estará constituida por todos los patrones de entrada que se mapean en esa neurona, es decir $V_k = \{X | X \Rightarrow \eta_k\}$

3.4.4 Compresión de imágenes basada en VQ mediante redes SOM

El proceso de compresión de imágenes basado en VQ mediante redes SOM

Utilizando la VQ por medio de las redes SOM, al igual que con la transformación auto-asociativa mediante el MLP, el proceso de compresión comienza por descomponer la imagen en bloques y representar cada uno de éstos por medio de un vector. Para codificar un bloque, su vector correspondiente se suministra como entrada a una red cuyo número de unidades de entrada es igual al tamaño de los vectores y cuyo número de unidades de salida es igual al tamaño elegido para el diccionario; entonces, al bloque se le codifica mediante el índice de la neurona ganadora.

Para el entrenamiento de la red se genera una base de vectores de entrenamiento a partir de imágenes diversas. Los vectores de la base se aplican aleatoriamente a la red para que mediante un aprendizaje auto-organizativo, después de un tiempo de estabilización o convergencia, todos los vectores hayan quedado distribuidos en tantos clusters como neuronas de salida tenga la red. El vector de pesos para determinada neurona de salida[20] corresponderá al centroide del cluster respectivo. El diccionario estará integrado por la matriz de pesos. Durante la codificación, la red determinará cuál de los clusters aloja de mejor manera al vector del bloque a codificar, en otras palabras, a cual de los centroides está más cercano el vector del bloque. La decodificación es directa, ya que un bloque queda codificado como un índice del diccionario; basta con tomar ese índice y con él acceder al diccionario.

Revisión de algunas redes SOM desarrolladas

Los orígenes de la cuantización vectorial mediante redes con aprendizaje auto-organizativo se remontan a los trabajos de Kohonen [75] y Nasrabadi [76]. A partir de entonces se desarrollan investigaciones orientadas a experimentar con distintos tamaños de bloque y de diccionario, aplicar la cuantización directamente sobre los píxeles de imagen o bien después de haber aplicado sobre ésta una determinada transformación, el tipo de red competitiva utilizada para la cuantización, variaciones en los algoritmos de aprendizaje para reducir el tiempo de convergencia y esquemas diversos de cuantización vectorial para reducir el tamaño de las redes.

En la línea de aplicar alguna técnica de transformación previa a la VQ, en [77] se aplica el algoritmo SSC (synaptic space competitive) para desarrollar un sistema de codificación de imágenes que combina la descomposición multiresolución mediante la 2D-DWT y la VQ basada en la red neuronal competitiva. En el algoritmo SSC sólo se actualizan los pesos de la neurona ganadora, por lo tanto no se establece ningún tipo de vecindad entre las neuronas. La descomposición wavelet se realiza en 3 niveles utilizando la función Daubechies de 6 coeficientes. Después de la descomposición wavelet, la sub-banda LL3 se codifica usando cuantización escalar de 8 bits; las sub-bandas HL3, LH3 y HH3 se codifican usando diccionarios multiresolución de tamaño 256 generados con el algoritmo SSC para bloques de 2 x 2 coeficientes; las sub-bandas HL2, LH2 y HH2 se codifican también con diccionarios de tamaño 256 pero con bloques de 4 x 4; el nivel de resolución 1 se descarta. Se obtiene así una razón de compresión de 0,3125 bpp, con un PSNR de 29,02 dB cuando se codifica la imagen Lena de 256 x 256 pixeles en tonos de gris.

[20] El vector de pesos para una neurona de salida es el formado por los pesos de las conexiones que van de cada unidad de entrada a esa neurona

En [78] se presenta un método para la compresión de imágenes médicas que combina la transformada wavelet y un algoritmo SOM mejorado. El algoritmo incorpora para la actualización de pesos la frecuencia ganadora y el parámetro de momento (considerar tanto la variación actual del peso como la previa). El entrenamiento se realiza en sólo dos epochs; en el primero se genera el diccionario de manera que refleje la distribución de los patrones de entrenamiento, mientras que en el segundo el diccionario se regenera para desplazar sus vectores hacia el centroide de las regiones. El método se implementa en C++. Se usan imágenes médicas de color con 128 x 128 pixeles. Las imágenes se descomponen en los tres planos correspondiendo a los canales RGB y cada plano se transforma y cuantiza para generar los datos de compresión. Cada plano se divide en bloques de 4 x 4. Se comparan los resultados usando los algoritmos LBG, el SOM convencional, el SOM mejorado que se propone y el SOM mejorado combinado con la transformada wavelet. Con la combinación wavelet-SOM mejorado se obtienen valores para el PSNR alrededor de 38 dB, superando en más de un dB a los valores obtenidos con el algoritmo LBG.

En [79] los autores exponen un método de compresión de imágenes que combina la transformada discreta del coseno y la cuantización vectorial. Para la VQ utilizan una versión del algoritmo de Kohonen desarrollado por ellos mismos en [80] denominado FKA (Fast Kohonen Algorithm). El FKA está basado en un proceso de creación/modificación de pesos. Se establece un valor de umbral para la distancia mínima entre los patrones de entrenamiento y los vectores de pesos. La red se inicia con una sola neurona, cuyos pesos son iguales a los elementos del primer patrón. En cada iteración se obtiene la distancia mínima entre el patrón que se presenta y todos los vectores de pesos; si ésta excede el valor del umbral, se crea una neurona con sus pesos iguales a los del patrón; en caso contrario, el vector de pesos de la neurona ganadora se modifica acercándolo al centroide de los patrones de entrada que han quedado asociados a esa neurona[21]. El proceso se detiene cuando en un epoch no existe creación alguna de neuronas. En este algoritmo el número de neuronas es adaptativo y, por lo tanto, no existen las que se conocen como neuronas muertas. Los resultados de la experimentación combinando la DCT y la VQ con FKA son sobresalientes en todos los aspectos: PSNR de 30,4 dB (contra 28,2 dB sin la DCT) para una razón de compresión igual a 90,7:1 (bloques de 8 x 8 y red con 50 neuronas) y tiempo de entrenamiento de 9,1 segundos contra 24 usando el algoritmo original de Kohonen.

Usando también la DCT, en [81] se presenta un sistema completo de compresión de imágenes, integrando la etapa de transformación, la de cuantización vectorial por medio de una red SOM y una etapa de codificación que combina la codificación diferencial basada en el principio del gradiente suave como un predictor de primer orden muy sencillo y la codificación entrópica RLC de longitud variable. Las imágenes se descomponen en bloques de 4 x 4 pixeles sobre los cuales se aplica la DCT dando por resultado vectores de 16 coeficientes, de los cuales sólo se consideran los 6 u 8 primeros (primordialmente asociados a las frecuencias bajas). Los vectores así obtenidos se aplican a la red que opera como cuantizador vectorial. Para aprovechar la propiedad que caracteriza a las redes SOM de preservar en el mapa los rasgos del espacio de entrada, se incluye el codificador diferencial mencionado y finalmente se realiza la codificación entrópica. Como resultado de la experimentación, se concluye que la cuantización es mejor cuando se mantienen sólo 6 coeficientes de la DCT, con valores para el PSNR de 24,7 dB con una razón de compresión de 25,22:1 (diccionario con tamaño 128) para la imagen Lena.

[21] Para esto, se lleva un conteo de la cantidad de patrones asociados a cada neurona

Por el lado de la aceleración del aprendizaje, en [82] se experimenta con una variante del algoritmo SOM denominada SFSN (Sample Frequency Sensitive Network) en la que se detecta el momento en el que la red se estabiliza, es decir, las clases formadas por los patrones de entrenamiento no cambian de un epoch al siguiente. El número de epochs requerido para la estabilización se estima experimentalmente alrededor de 32. De esta manera se reduce la complejidad del tiempo de entrenamiento de $O(N^2)$ a $O(N)$, donde N es el tamaño del conjunto de patrones de entrenamiento. Como una mejora adicional, para incrementar la calidad de la compresión se codifica también la imagen residual. Para la imagen Lena se alcanza un valor del PSNR de 27,58 dB con una razón de compresión de 24:1 sin codificar la imagen residual y estos valores cambian a 43,11 dB y 15:1 cuando sí se codifica.

Otra línea es la que se enfoca hacia el diseño de redes jerárquicas para cuantización vectorial. En [83] se desarrolla y experimenta una red SOM jerárquica (HSOM) que reduce la complejidad temporal de O(N) a O(logN) para disminuir el tiempo de entrenamiento. Los autores emplean la técnica de cuantización vectorial estructurada en árbol para generar la red HSOM, quedando ésta conformada como un conjunto de redes SOM organizadas en un árbol jerárquico. Para el aprendizaje de las redes utilizan entrenamiento por lotes. Desarrollan una HSOM de dos niveles consistente en 11 redes SOM con 10 neuronas cada una, una SOM para el nodo raíz y 10 para los nodos hoja, todas con 9 unidades de entrada. Para la imagen Lena de 512 x 512 en tonos de gris, con bloques de 3 x 3 pixeles, se obtuvo una fidelidad de 70,72 como valor del SNR y una razón de compresión de 12:1. La implementación es totalmente software en el ambiente de Matlab.

En cuanto a las implementaciones hardware, uno de los primeros trabajos se reporta en [84]. Los autores desarrollan un diseñó VLSI para un compresor de imágenes basado en una red SOM incluyendo en el algoritmo la técnica de sensibilidad a la frecuencia ganadora (FSW). Con esta técnica se modifica el aprendizaje competitivo mediante la inclusión de la frecuencia ganadora y un umbral superior para ésta (Fth). Se experimenta aplicando la técnica FSW únicamente en el primer epoch o en los primeros dos. Si la frecuencia ganadora para alguna neurona alcanza el valor de Fth, esa neurona dejará de competir para los patrones de entrenamiento restantes. Para imágenes de 256 x 256 pixeles y bloques de 5 x 5, los autores reportan un valor de distorsión de 115,04 para el MSE (PSNR=27,52 dB) cuando la FSW se aplica sólo durante el primer epoch y de 112,08 (PSNR=27,63 dB) si se aplica durante dos epochs, con una razón de compresión igual a 25:1. Se implementa un neurochip prototipo híbrido, análogo/digital, para una SOM con 64 neuronas y 25 unidades de entrada, ocupando un área de 31 mm^2 con tecnología CMOS de 2 μm, presentando una cadencia de 500 ns por vector codificado y 3200 MCPS.

En [86] se lleva a cabo la implementación hardware de una SOM con señales de pulsos modulados por fase y circuitos digitales de amarre de fase (DPLLs). El sistema utiliza el DPLL como un elemento de cálculo ya que su operación es muy similar al procesamiento requerido por la SOM. La implementación de la SOM consiste en la DPLL-SOM, dos moduladores de fase y un generador de señal de reloj. Con los moduladores de fase los valores de los vectores de entrada se suministran en forma binaria; el generador de reloj proporciona varias señales de tiempo, como son la señal de referencia y la señal de modo, que indica si se encuentra operando en el modo de búsqueda del ganador o en el modo de actualización. La red experimental tiene 50 unidades de entrada y 25 neuronas de salida. Utiliza distancia Manhattan y cuando fue implementada sobre una FPGA Virtex 2 alcanzó un rendimiento de 4,89 MCUPS y 9,78 MCPS.

En [87] los autores reportan la implementación de una red SOM a partir de un diseño

parametrizado en el cual al momento de la síntesis es posible escoger la cantidad de neuronas de salida, la dimensión de los vectores de entrada y el ancho de los mismos. Primero los autores diseñaron una red SOM en un solo chip que consiste en un arreglo SIMD de 256 procesadores, representando cada uno a una neurona, empleando la distancia Manhattan para medir la diferencia entre vectores. Después presentan un core consistente en un arreglo SIMD y un procesador RISC con una arquitectura Harvard de 16 bits como controlador. Con tecnología de standard-cell de 0,18 μm el core alcanza una frecuencia de operación de 200 Mhz, logrando así 11000 MCPS para una red de 256 neuronas, ocupando un área de 21,16 mm ^{2}.

En [88] se reporta la implementación hardware de una SOM usando un algoritmo rápido basado en el truncamiento de los datos en las iteraciones iniciales, de modo que la comparación de distancias es burda al principio y se va a afinando a medida que el proceso de entrenamiento avanza. La arquitectura hardware es también tipo SIMD con procesamiento por componente y se emplea la distancia Manhattan. Para la comparación de distancias se utiliza el esquema WPBS descrito en 4.2.2. Los autores aseveran que la actualización de pesos se realiza en un solo pulso de reloj; esto implica que únicamente se modifique el peso de la neurona ganadora y no los de las vecinas; por lo tanto, más que una red SOM se trata de una red competitiva simple. El circuito se implementa sobre una FPGA Virtex 2 pudiendo alojar una red de 256 neuronas. Operando a 70 MHz, la red alcanza 15700 MCUPS en la fase de entrenamiento al procesar vectores de dimensión igual a 128 con datos de 16 bits e igual cantidad de MCPS en la de recall.

En la Tabla 3.4 se concentran algunas características y resultados de las redes SOM para cuantización vectorial revisadas.

Referencia	Transformación	Codificación	CR	Calidad	Observaciones
Madeiro[77]	Wavelet	Ninguna	25,6:1	PSNR=29,02 dB	Diccionario multiresolución
Li[79]	DCT	Ninguna	90,7:1	PSNR=30,4 dB	Algoritmo SOM rápido
Amerijckx[81]	DCT	Diferencial + entrópica	18,5:1	PSNR=25,4 dB	Implementación software de las tres etapas
Soliman[82]	Ninguna	Ninguna	24:1	PSNR = 27,58 dB	Detiene el entrenamiento cuando las neuronas ganan el mismo número de veces en dos iteraciones
Barbalho[83]	Ninguna	Ninguna	12,12:1	SNR=70,72	Arbol jerárquico de redes (HSOM)
Fang[84]	Ninguna	Ninguna	20:1	PSNR=28,97 dB	Implementación VLSI analógica incorpora sensibilidad a la frecuencia ganadora

Tabla 3.4: Redes SOM para cuantización vectorial

Capítulo 4

Diseño de la red SOM

En este capítulo se describe el diseño de la red SOM. En la primera sección, se analizan las arquitecturas que han sido desarrolladas para la implementación hardware de este tipo de red. A continuación se analizan algunos esquemas que han sido propuestos para la implementación de los principales bloques que componen la red. Posteriormente se efectúa la descripción del diseño y por último se presentan algunos resultados de la implementación y de la experimentación.

4.1 Arquitecturas SOM

El diseño hardware de una red SOM requiere tomar en cuenta sus dos fases de operación, la de entrenamiento y la de recall. En la de recall el principal aspecto a considerar es la velocidad de operación, cuestión que se vuelve crítica cuando la red se utiliza para aplicaciones en tiempo real, situación más exigente en la medida en que crezca el tamaño de la red (dimensión de los vectores de entrada y cantidad de vectores en el diccionario, hablando en términos de cuantización vectorial). Para la operación de la red en la fase de entrenamiento, aunque con menor relevancia, la velocidad continúa siendo una cuestión importante; no obstante, también es necesario tomar en cuenta todo lo relacionado con el almacenamiento de los pesos sinápticos para tener la posibilidad de acceder a ellos de manera fácil y rápida, tanto para su lectura como, sobre todo, su actualización. Sin embargo, el aspecto crítico en el diseño de la red respecto a la fase de entrenamiento es la eficiencia en el uso de los recursos de hardware, con el propósito de reducir el área ocupada, ya que ésta crece proporcionalmente con el tamaño de la red.

En las dos fases de operación de una red SOM, dado un vector de entrada x es necesario determinar cuál es la neurona ganadora. En otras palabras, se necesita encontrar cuál es la neurona cuyo vector de pesos es más cercano a ese vector de entrada que se aplica a la red. La expresión a evaluar para obtener la neurona ganadora es $c = arg\ \ min_i\, d(x, \mathbf{w}_i)$, siendo c el índice de la neurona ganadora y $\mathbf{w}$ el diccionario o conjunto de vectores de pesos. Esto implica calcular la distancia entre el vector x y los N vectores de que consta el diccionario. A su vez, el cálculo de la distancia entre vectores requiere tomar en cuenta la contribución de cada uno de sus k elementos. Así, para la distancia Euclidiana y para la Manhattan se tienen las expresiones siguientes:

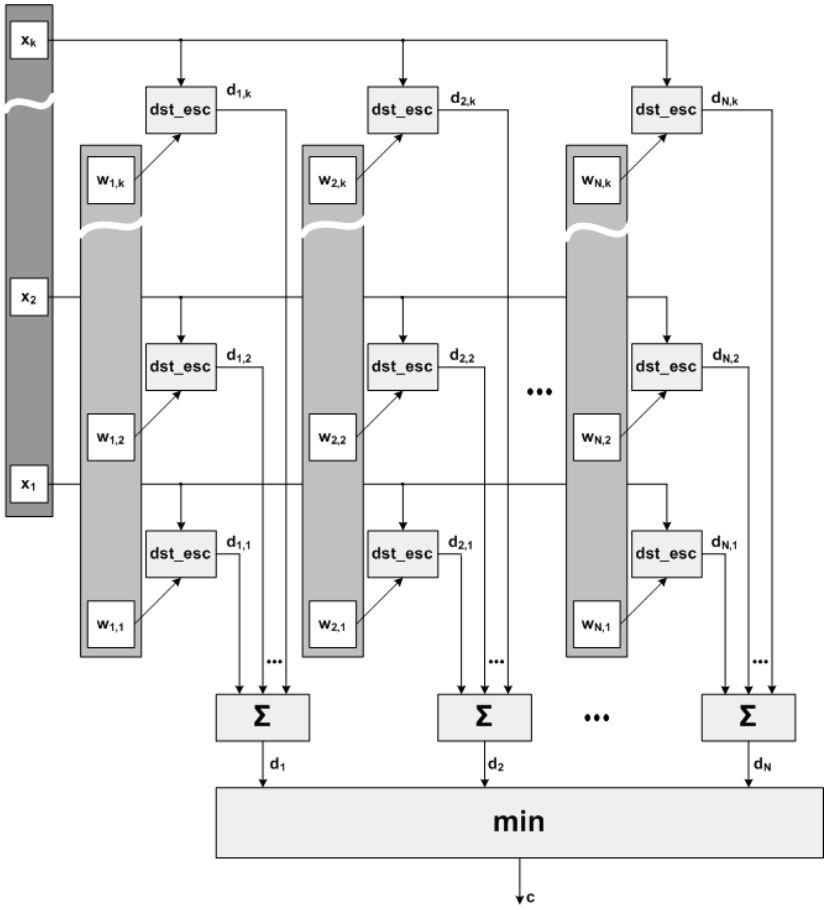


Fig. 4.1: SIMD para recall con procesamiento en paralelo de un vector completo

$$\begin{aligned} c &= arg \min_{i=1}^{N} \sum_{j=1}^{k} (x_j - \mathbf{w}_{ij})^2 \quad Distancia\ Euclidiana, \\ c &= arg \min_{i=1}^{N} \sum_{j=1}^{k} |x_j - \mathbf{w}_{ij}| \quad Distancia\ Manhattan. \end{aligned} \tag{4.1}$$

Con el propósito de su análisis, a continuación se exponen cinco arquitecturas para la implementación hardware de la red SOM. Cada uno de estos esquemas corresponde a cierto nivel de paralelismo.

4.1.1 Arquitecturas tipo SIMD

SIMD con paralelismo a nivel vector

Con el mayor nivel de paralelismo, en el esquema para la red SOM que se muestra en la Fig. 4.1, cada bloque dst_esc calcula la contribución de la componente j del vector de entrada (x_j) y de pesos ($\mathbf{w}_{i,j}$) a la distancia d_i. Es decir, $d_{i,j}$ es la componente j de la distancia entre el vector de entrada x y el vector de pesos $\mathbf{w}_i$. Si k es el tamaño de los vectores y N es el número de neuronas en la red (o, lo que es lo mismo, la cantidad de vectores en

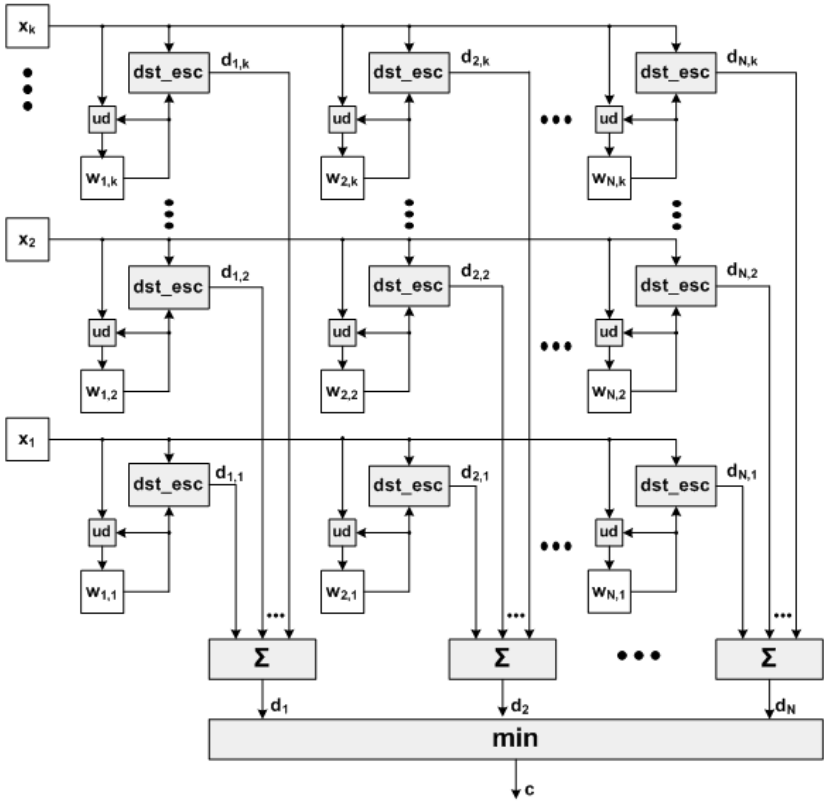


Fig. 4.2: SIMD con procesamiento en paralelo de un vector completo incluyendo entrenamiento

el diccionario), entonces habrá N x k bloques dst_esc. Las distancias escalares $d_{i,j}$, para i constante y $j = 1, 2, \ldots k$, se suman en su respectivo bloque $\mathbf{\Sigma}$ para obtener la distancia entre vectores d_i. Finalmente, las distancias entre vectores $(d_i, i = 1, 2, \ldots N)$ se comparan en el bloque min para obtener el índice c de la neurona ganadora.

Por lo tanto, este esquema utiliza N x k bloques para el cálculo de las distancias escalares[1], N sumadores (cada uno con k entradas) y un comparador de valor mínimo con N entradas. En este esquema un vector de entrada se procesa en un sólo paso. Por lo tanto, todos sus elementos deben estar disponibles simultáneamente; asimismo, se requiere que los pesos estén almacenados en registros para así tener también la posibilidad de acceder a ellos de manera simultánea.

El mismo esquema se muestra en la Fig. 4.2, sólo que en ésta se incluye, para cada unidad y para cada componente, el bloque de actualización de pesos, ud. El nuevo valor para un peso se calcula en función de su valor previo y el de la respectiva componente del vector de entrada.

[1]De acuerdo a la Ec. 4.1, la distancia escalar es el valor absoluto de la diferencia para el caso de la distancia Manhattan y el cuadrado de la diferencia para el caso de la distancia Euclidiana.

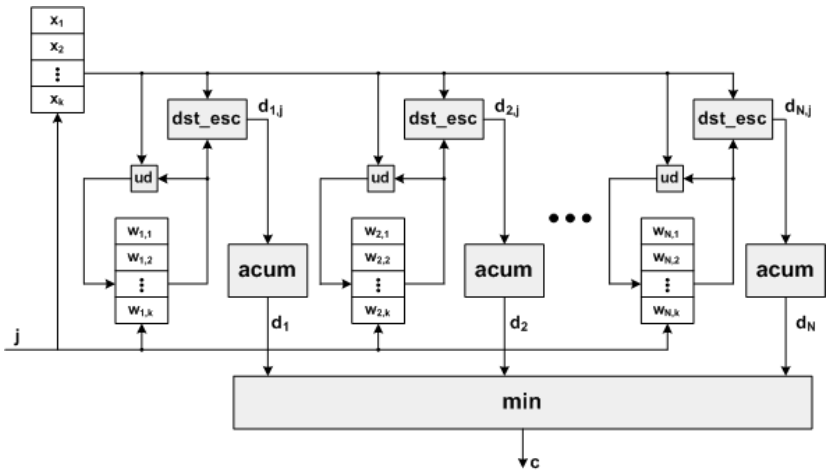


Fig. 4.3: SIMD para procesamiento en paralelo de una componente

SIMD con paralelismo a nivel componente

Con un menor nivel de paralelismo, en la Fig. 4.3 se muestra el esquema que procesa una componente del vector de entrada a la vez, para todos los vectores del diccionario. La señal j elige la componente que se está procesando y varía desde 1 hasta k. Las distancias obtenidas para cada valor de j se van acumulando. Una vez que se han aplicado las k componentes, en la salida de los acumuladores se tiene la distancia entre el vector de entrada y el respectivo vector del diccionario. Finalmente las N distancias así obtenidas se comparan para obtener el índice de la neurona ganadora.

Para este esquema se requiere que los elementos del vector de entrada se presenten secuencialmente, lo mismo debe ocurrir con los correspondientes a los vectores de pesos. Por lo tanto, cada vector de pesos se encuentra almacenado en una memoria de igual tamaño. Los lugares de estas memorias se van recorriendo uno a uno para ir accediendo secuencialmente a los elementos de los vectores. La señal j se emplea para proporcionar la dirección a todas las memorias.

Para este esquema son necesarios N bloques dst_esc, N acumuladores y un comparador de valor mínimo con N entradas. El procesamiento de un vector de entrada requiere de k pasos.

SIMD con paralelismo a nivel neurona

Transponiendo la arquitectura anterior, se obtiene otra que procesa, neurona por neurona, todos los componentes de un vector en paralelo. En la Fig. 4.4 se presenta el esquema para esta arquitectura. La señal i selecciona la neurona que se está procesando y varía desde 1 hasta N. Las distancias escalares obtenidas para un valor fijo de i se suman para obtener la distancia entre el vector de entrada y el vector de pesos de la neurona i[2]. Esta distancia d_i se compara con la mínima que se tiene hasta el momento. Si es menor, la distancia mínima se actualiza y también el valor del índice correspondiente. Cuando se han cubierto todos los valores de i, el bloque indx contendrá el índice de la neurona ganadora.

[2]El vector de pesos de cada neurona está almacenado en diferentes memorias sinápticas con la finalidad de acceder a todas sus componentes en paralelo

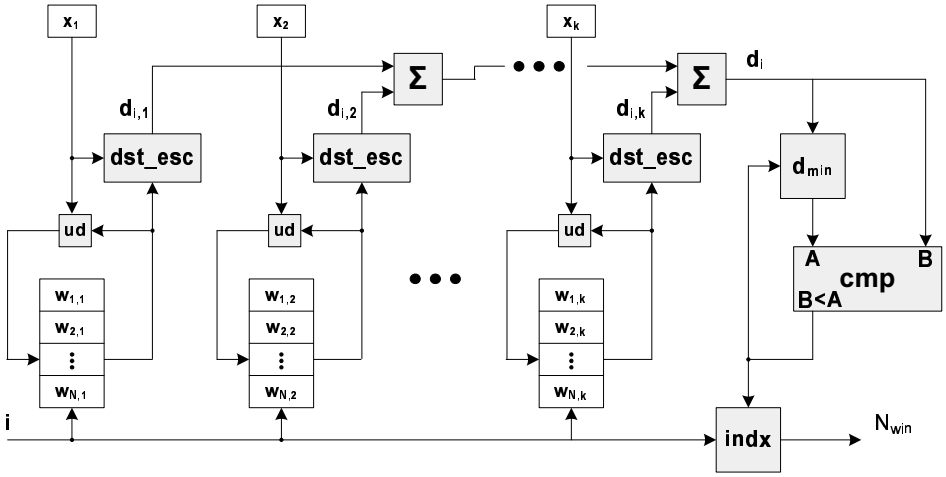


Fig. 4.4: SIMD para procesamiento en paralelo de una neurona

Para esta arquitectura se requiere que los elementos del vector de entrada se presenten simultáneamente, lo mismo debe ocurrir con los correspondientes al vector de pesos de la neurona que se está procesando.

En cuanto a los recursos necesarios, esta arquitectura emplea k bloques dst_esc, $k-1$ sumadores y un comparador de 2 entradas. El procesamiento de un vector de entrada requiere N pasos.

4.1.2 Procesamiento secuencial

Sin ningún paralelismo, el esquema que se muestra en la Fig. 4.5 se basa en la reutilización de un solo bloque dst_esc. Los pesos, que se obtienen al acceder a las memorias de la misma manera como en el esquema anterior, se someten a una multiplexación en el tiempo, de modo que primero se obtiene la distancia entre el vector de entrada y el vector de pesos correspondiente a la primera neurona, después se hace lo mismo, pero ahora con el vector de pesos correspondiente a la segunda neurona y así se continúa hasta llegar al vector de pesos de la última neurona. Cada vez que se obtiene una distancia, ésta se compara con la mínima obtenida hasta el momento; si la nueva distancia obtenida es menor que la mínima, ésta se actualiza y lo mismo se hace con el índice correspondiente.

La señal j se emplea para acceder secuencialmente a los elementos de uno de los vectores de pesos, simultáneamente a los del vector de entrada, mientras que la señal i se utiliza para ir seleccionando los vectores de pesos, uno a uno. El procesamiento de un vector de entrada por medio de este esquema requiere N x K pasos. Para este esquema sólo se requiere un bloque dst_esc, un acumulador y el comparador de valor mínimo es más sencillo, esto debido a que sólo se compara la distancia recién obtenida con la mínima obtenida hasta ese momento.

4.1.3 Arquitecturas sistólicas

Arquitectura sistólica matricial

El término "sistólico" proviene de la actividad neuro-muscular que se propaga como si fuera una onda. En los arreglos sistólicos electrónicos, los datos se transfieren en una dirección, por

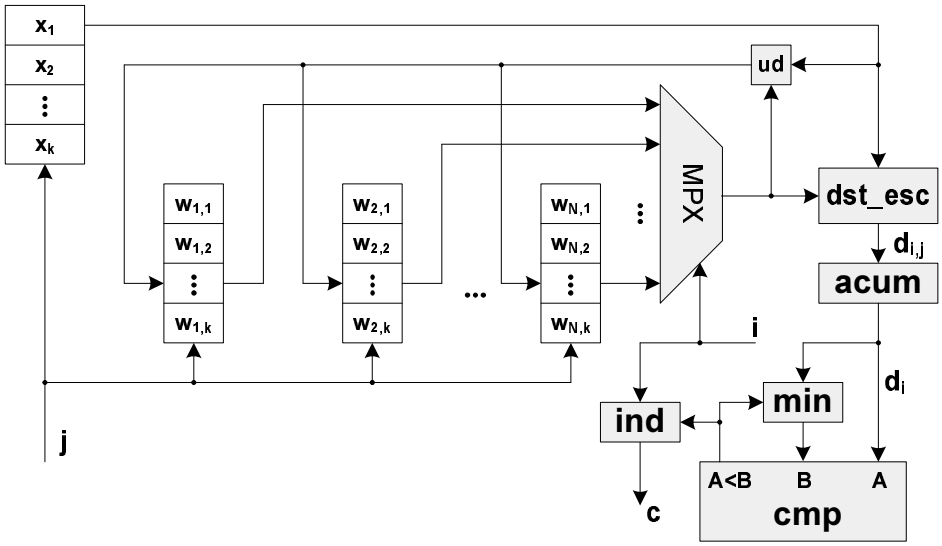


Fig. 4.5: Procesamiento secuencial

ejemplo, hacia abajo, mientras que los resultados parciales de los cálculos se mueven en la dirección ortogonal, es decir, hacia la derecha. Se dice, entonces, que el cómputo es espacialmente iterativo debido a que cada nodo de procesamiento recibe los datos y las instrucciones de los nodos adyacentes. Por lo tanto, no se requiere incluir buses.

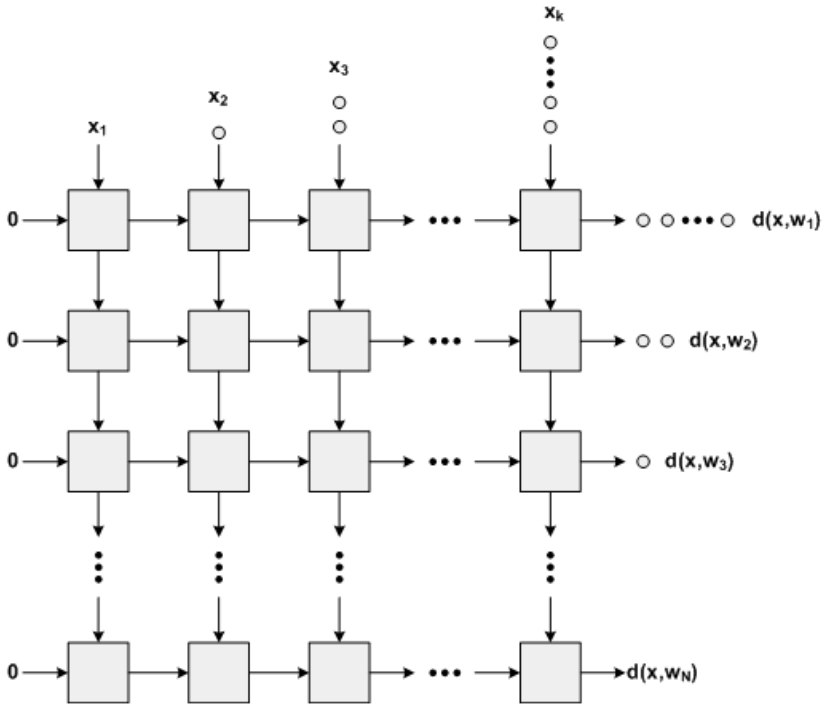


Fig. 4.6: Arquitectura sistólica matricial para una red SOM

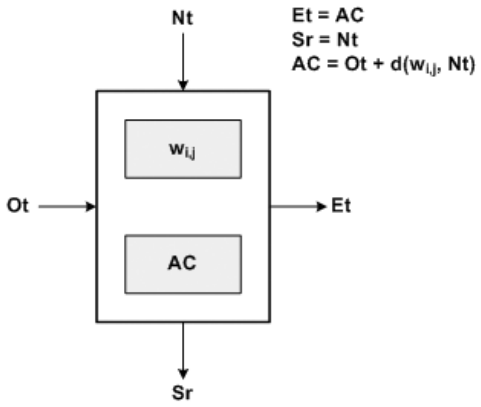


Fig. 4.7: Flujo de datos y cómputo para la arquitectura sistólica matricial

En la Fig. 4.6 se muestra un arreglo sistólico para obtener las distancias entre un vector de entrada x y los N vectores de pesos de una red SOM. Cada renglón de celdas corresponde a un vector de pesos, es decir, a un vector del diccionario; por lo tanto, se tienen k columnas, siendo k la dimensión de los vectores, y N renglones. Los elementos del vector de entrada circulan verticalmente de arriba a abajo, uno en cada columna y con un retardo de $(j-1)$ periodos, donde j es el número de columna $(j = 1, 2, \ldots k)$. Las distancias parciales entre el vector de entrada y los vectores del diccionario circulan horizontalmente de izquierda a derecha. Las distancias finales se obtienen en el lado derecho, también con un retardo que crece de arriba a abajo, es decir de un vector de diccionario a otro.

En la Fig. 4.7 se muestra el flujo de datos y el cómputo para una unidad de procesamiento (UP). A través de la entrada *Norte* (*Nt*) la UP recibe la componente del vector de entrada correspondiente al peso allí almacenado. Se calcula entonces la distancia entre esos dos datos (ya sea la Euclidiana o la Manhattan). El valor obtenido se suma al valor recibido por la entrada *Oeste* (*Ot*) y el resultado de la suma se envía por la salida *Este* (*Et*).

Este arreglo sistólico es un pipe-line bidimensional. Para utilizar cabalmente su capacidad de cómputo, se deben aplicar varios vectores de entrada de manera consecutiva, por lo cual las distancias se deben registrar en el extremo derecho el número de veces adecuado para su sincronización, y posteriormente ser comparadas para determinar así la neurona ganadora. Esto implica un procesamiento por lotes. En la Fig. 4.8 se muestra la temporización para la arquitectura sistólica matricial en el caso de vectores de dimensión 4 y tamaño de diccionario igual a 6.

Arquitectura sistólica lineal

Con únicamente N unidades de procesamiento, en la Fig. 4.9 se muestra un arreglo sistólico lineal. En la figura (a) se observa el arreglo de procesadores y en la figura (b) se aprecia la manera como fluyen los datos y como se lleva a cabo su procesamiento. Los elementos del vector de entrada fluyen a través del canal horizontal número 1. Cada UP procesa los elementos de uno de los vectores de pesos, uno a uno, junto con el correspondiente elemento del vector de entrada. Las distancias escalares se van acumulando en la misma UP. Después de k ciclos de iniciado el procesamiento de un vector de entrada x, la primera UP contendrá la distancia entre éste y el primer vector del diccionario $\mathbf{w}_1$. Un ciclo adicional después, la

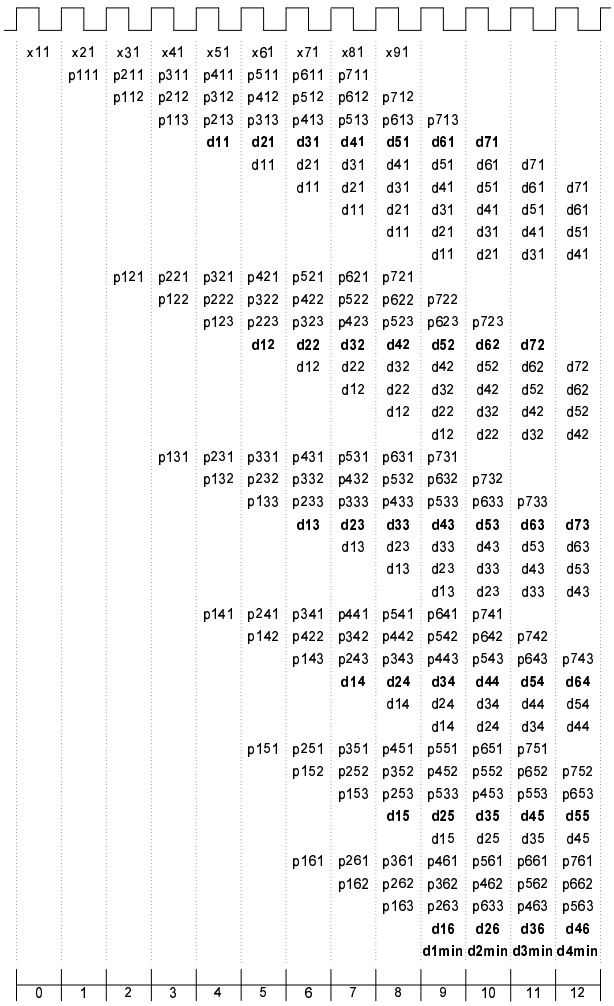


Fig. 4.8: Temporización del flujo de datos para un arreglo sistólico matricial

segunda UP contendrá la distancia entre x y $\mathbf{w}_2$ y, así sucesivamente, hasta que, $N + k - 1$ ciclos contados a partir del comienzo del proceso, en la última UP se tendrá la distancia entre x y $\mathbf{w}_N$. El proceso se repite cada k ciclos.

Para llevar a cabo la comparación de las distancias y así determinar cual es la mínima y el índice correspondiente a la UP en la que ésta se generó, se tienen los canales horizontales señalados con los números 2 y 3. En el ciclo k aparecerá en la salida Et_2 de la primera UP el valor calculado para $d(x, \mathbf{w}_1)$ y en la salida Et_3 el valor constante 1; en el ciclo $k + j - 1$, en la salida Et_2 de la j-ésima UP se tendrá el valor mínimo obtenido al comparar las distancias $d(x, \mathbf{w}_1)$, $d(x, \mathbf{w}_2)$, ... $d(x, \mathbf{w}_j)$ y en la salida Et_3 estará el índice correspondiente a la UP que produjo ese valor mínimo; finalmente, en el ciclo $k + N - 1$, a la salida de la última UP se tendrá la distancia mínima (en Et_2) y el índice correspondiente (en Et_3).

Al igual que para el caso del arreglo sistólico matricial, en el lineal el procesamiento de los vectores de entrada debe ser por lotes y de la misma manera debe realizarse el entrenamiento. En la Fig. 4.10 se muestra la temporización para la arquitectura sistólica lineal para el mismo caso de vectores de dimensión 4 y tamaño de diccionario igual a 6. x_{ij} es la componente j del vector de entrada i. Después de 9 periodos de reloj, contados a partir de la primera

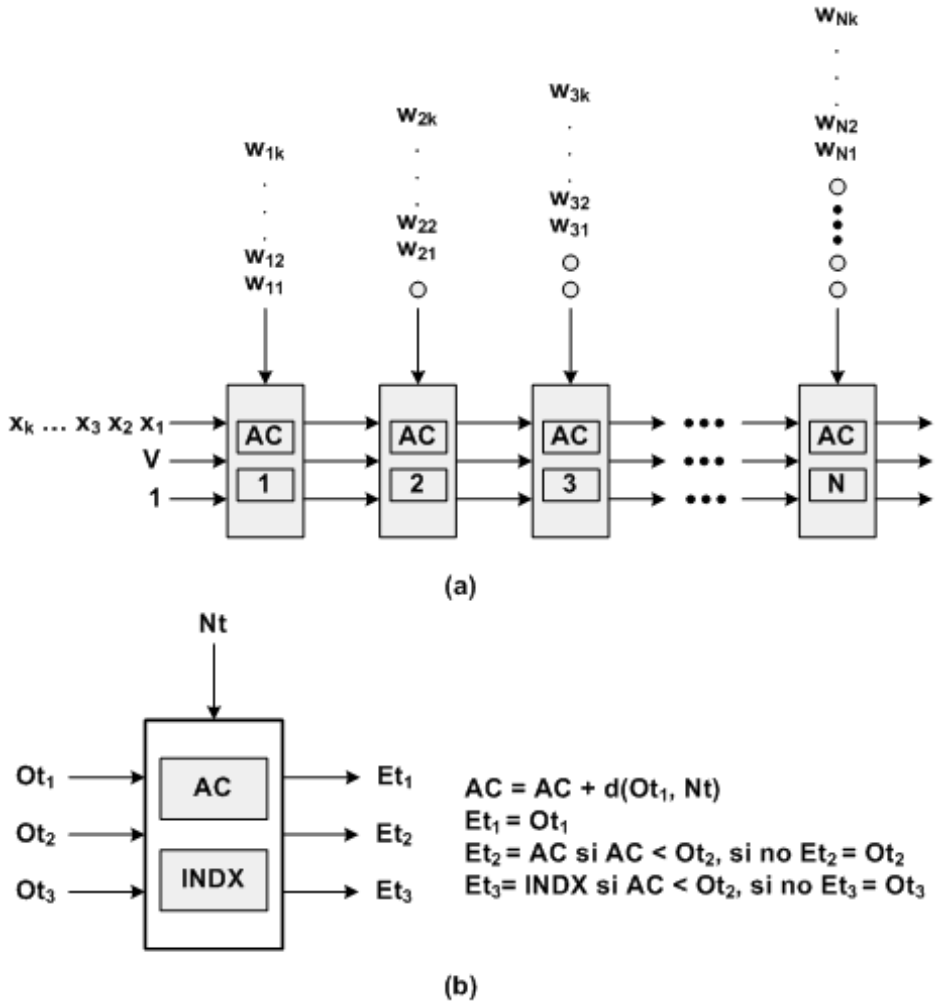


Fig. 4.9: Arquitectura sistólica lineal

componente del primer vector de entrada, se obtendrá, en la salida Et_2 de la última UP, la más pequeña de las distancias que existen entre ese vector de entrada y los seis vectores de pesos.[3] Posteriormente, cada cuatro periodos de reloj se obtiene la distancia mínima para el más reciente vector de entrada. p_{rst} es la distancia parcial entre el vector de entrada r y el vector de pesos s cuando se han considerado las primeras t componentes de los vectores. m_{rs} es la distancia mínima para el vector de entrada r hasta el momento en que se han comparado las distancias correspondientes a los primeros s vectores de pesos.

Se debe señalar que existe la arquitectura sistólica lineal que se obtiene al transponer la que se acaba de analizar. Para abreviar, se omite su presentación.

[3]En Et_3 de la UP del extremo derecho se obtendrá el índice correspondiente al vector de pesos para el cual se generó la distancia mínima.

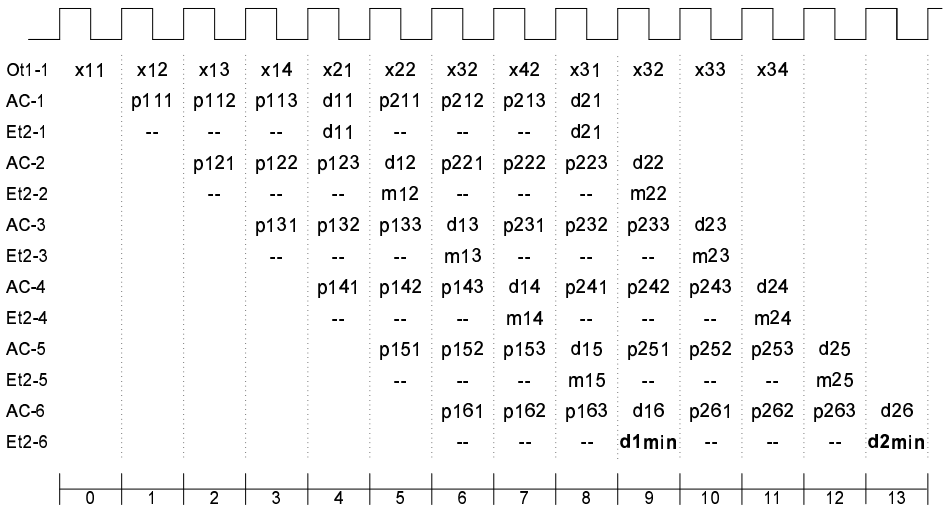


Fig. 4.10: Temporización de datos para una arquitectura sistólica lineal

4.1.4 Comparación de arquitecturas

En la Tabla 4.1 se comparan las seis arquitecturas descritas. Para la comparación se toman en cuenta tres aspectos: la cantidad y complejidad de los bloques requeridos, el rendimiento en cuanto a velocidad (latencia y cadencia) y el tipo de entrenamiento más apropiado para la arquitectura (por lotes o adaptativo). Para los sumadores y los comparadores se incluye la cantidad de ellos (1) y el número de sus entradas (2).

De la Tabla 4.1 se desprende que el mejor esquema desde el punto de vista de la velocidad es el del arrreglo SIMD con procesamiento por vectores, ya que tanto su latencia como su cadencia son de un ciclo de reloj. Sin embargo, dicho esquema, al igual que el sistólico matricial, requiere una cantidad considerable de recursos, lo cual los hace inviables para ser utilizados para redes de mediano o gran tamaño.

En el otro extremo se encuentra la arquitectura con procesamiento secuencial. Es la más simple y, por lo tanto, requiere una cantidad mínima de recursos. Sólo que es la peor por lo que respecta a la velocidad, pues su latencia y cadencia son iguales al producto de la dimensión de los vectores y el número de neuronas de la red.

El mejor equilibrio entre velocidad y ocupación de recursos lo presentan las arquitecturas SIMD con procesamiento en una sola dimensión y la sistólica lineal. Sin embargo, la sistólica lineal adolece del problema consistente en que el procesamiento debe ser por lotes, lo cual además hace más complejo el diseño debido a los problemas de sincronización, principalmente por lo que se refiere a la etapa dedicada a la actualización de pesos.

Al comparar las dos arquitecturas SIMD con procesamiento en paralelo en una sola dimensión, por componente en un caso y por neurona en el otro, se debe destacar que con respecto a la cadencia y a la cantidad de unidades de procesamiento la mejor alternativa depende de la relación entre el número de unidades de entrada (dimensionalidad del diccionario) y la cantidad de neuronas (tamaño del diccionario). Usualmente es más grande la cantidad de neuronas (N) que el número de entradas (k) y en tal caso es preferible el paralelismo por componente.

Por otra parte, una desventaja del SIMD con paralelismo de neurona reside en la cadena de $k-1$ sumadores que implica una trayectoria combinacional muy larga que reduce la

Esquema		SIMD vector	SIMD componente	SIMD neurona	Secuencial	Sistólico matricial	Sistólico lineal
Comparadores	(1)	1	1	1	1	1	N
	(2)	N	N	2	2	N	2
Cálculo distancia		k*N	N	k	1	k*N	N
Sumadores	(1)	N	0	k-1	0	0	0
	(2)	k	0	2	0	0	0
Acumuladores		0	N	0	1	k*N	N
Registros		0	0	0	0	N*(N-1)/2	0
Latencia		1	k	N	k*N	k+N-1	k+N-1
Cadencia		1	k	N	k*N	1	k
Procesamiento		Adaptativo	Adaptativo	Adaptativo	Adaptativo	Lotes	Lotes

Tabla 4.1: Comparativa de arquitecturas para la implementación SOM

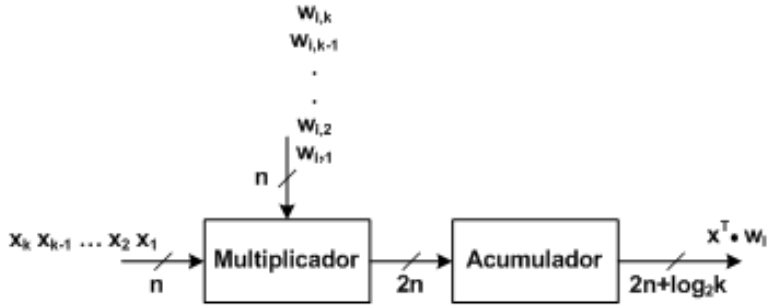


Fig. 4.11: Cálculo del producto interno para la distancia Euclidiana

frecuencia máxima de operación. Sin embargo, una ventaja de esta arquitectura consiste en que la comparación es secuencial, lo cual es conveniente por el ahorro tanto en tiempo como en ocupación de recursos.

Por todo lo anterior, el análisis se decanta hacia la arquitectura SIMD con procesamiento en paralelo de una componente, tomando en cuenta velocidad, recursos y simplicidad.

4.2 Bloques de procesamiento

4.2.1 Cálculo de distancias

Cálculo para la distancia Euclidiana

La distancia Euclidiana entre el vector de entrada x y el vector de pesos $\mathbf{w}_i$ se calcula mediante la siguiente expresión:

$$d_i = \sqrt{\sum_{j=1}^{k}(x_j - w_{ij})^2}. \tag{4.2}$$

Ya que es posible realizar las comparaciones usando el cuadrado de las distancias, se puede omitir la extracción de la raíz cuadrada. Además, el cuadrado de la distancia se puede reescribir de la siguiente manera:

$$\begin{aligned} d_i^2 &= \sum_{j=1}^{k}(x_j^2 - 2x_j w_{ij} + w_{ij}^2) \\ [h] \quad &= \sum_{j=1}^{k} x_j^2 - 2\sum_{j=1}^{k} x_j w_{ij} + \sum_{j=1}^{k} w_{ij}^2 \ . \\ &= \sum_{j=1}^{k} x_j^2 - 2x^T \cdot w_i + \sum_{j=1}^{k} w_{ij}^2 \end{aligned} \tag{4.3}$$

Considerando los tres términos de esta última expresión, el primero es común para todas las comparaciones; por lo tanto, no se requiere calcularlo. El tercero es una constante para cada vector de pesos[4] y se puede almacenar como una componente adicional de cada uno

[4]En realidad, durante el entrenamiento es necesario actualizarlo, mientras que para la fase de recall es una constante

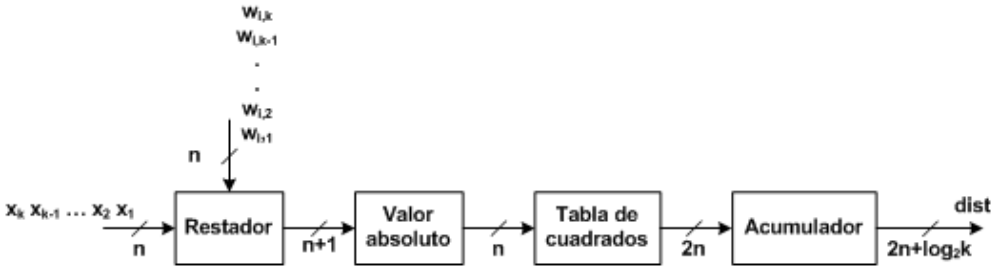


Fig. 4.12: Distancia Euclidiana basada en tabla de cuadrados

de ellos. Entonces, sólo es necesario calcular el segundo, que es igual al doble del producto interno del vector de entrada transpuesto y el vector de pesos. El cálculo del producto interno es una tarea para la cual los procesadores de señales tienen recursos especialmente diseñados (multiplicadores/acumuladores). En la Fig. 4.11 se observa el esquema para el cálculo del producto interno de los vectores x y $\mathbf{w}_i$.

Otra manera eficiente para el cálculo de la distancia Euclidiana consiste en almacenar en una tabla los cuadrados para todos los posibles valores de las diferencias escalares, $(x_j - w_{ij})^2$. Si los valores de x_j y w_{ij} se representan mediante n bits, entonces esta tabla estará formada por 2^n valores de $2n$ bits. De esta forma, para calcular la distancia Euclidiana entre x y $\mathbf{w}_i$ se sumarán los valores de la tabla a los que se accede empleando como índices los valores de $x_j - w_{ij}$ para $j = 1, 2, \ldots k$. La Fig. 4.12 muestra el esquema para el cálculo de la distancia Euclidiana con base en una tabla para los cuadrados de cantidades de n bits.

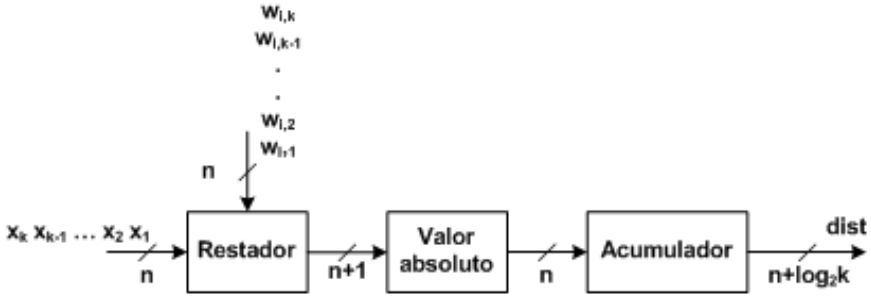


Fig. 4.13: Distancia Manhattan

Cálculo para la distancia Manhattan

La distancia Manhattan entre el vector de entrada x y el vector de pesos $\mathbf{w}_i$ se calcula por medio de la siguiente expresión:

$$d_i = \sum_{j=1}^{k} |x_j - w_{ij}| \,. \tag{4.4}$$

El empleo de la distancia Manhattan simplifica computacionalmente el procesamiento. La Fig. 4.13 ilustra el esquema para el cálculo de la distancia Manhattan.

4.2.2 Comparador de distancias

Para determinar la distancia mínima de las obtenidas entre el vector de entrada y los N vectores de pesos, existen varios métodos. A continuación se presentan dos de ellos.

Comparador word parallel-bit serial

El comparador word parallel-bit serial (WPBS) compara todas las distancias simultáneamente, aunque bit por bit. En la Fig. 4.14 se muestra la estructura de este comparador.

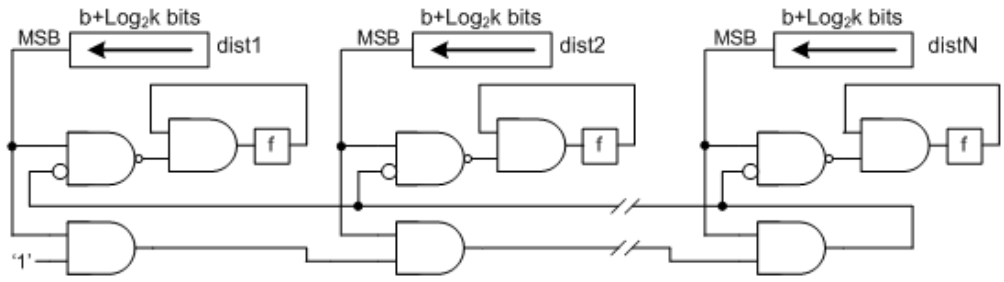


Fig. 4.14: Comparador word parallel-bit serial

La comparación se lleva a cabo de la siguiente manera. Cada uno de los flip-flops f se establece en '1' inicialmente. El proceso de comparación se inicia a partir del bit más significativo (MSB) de cada uno de los registros de desplazamiento. Todos esos bits están conectados a la cadena de compuertas AND que se observa en la parte inferior de la Fig. 4.14. La salida en la compuerta AND al final de la cadena estará en '0' sólo si al menos uno de los MSB está en '0'. En tal caso, todos los flip-flops cuyo MSB es '1' se limpian, indicando así que dejan de ser considerados como candidatos para la distancia mínima. A continuación, las distancias en los registros de corrimiento se desplazan un bit hacia la izquierda para efectuar la misma prueba con el siguiente bit. Después de que se hayan probado todos los bits, únicamente el flip-flop correspondiente a la distancia mínima estará en '1'. Pudiera darse el caso de que varias distancias tuvieran igual valor y fuera éste el mínimo, entonces varios flip-flops terminarían en '1' y sería necesario emplear algún criterio para elegir alguna de las unidades de procesamiento correspondientes a esos flip-flops como la ganadora.

El tamaño de los registros de corrimiento es igual al de las distancias. De este modo, de manera estricta, el número de bits de los registros de corrimiento y, por tanto, de ciclos de reloj requeridos para llevar a cabo la comparación, es igual a $b + Log_2\ k$, siendo b el número de bits de los elementos de los vectores de entrada y k la dimensión de los vectores. En la práctica, es posible truncar las distancias para considerar únicamente cierta cantidad de bits. Así, si sólo se consideran los L bits más significativos, se requerirán L ciclos de reloj.

Arbol de comparadores

En este caso, la determinación de la distancia mínima y, por tanto, del índice de la unidad de procesamiento que la genera se realiza por medio de bloques comparadores/multiplexores que reciben en su entrada un par de distancias y devuelven en su salida la menor de aquéllas y un bit adicional que señala en cual de las dos entradas se encuentra la distancia menor. La comparación, entonces, se realiza por niveles. En el primer nivel se tienen $N/2$ bloques

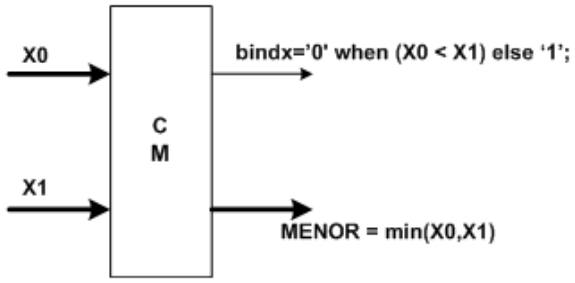


Fig. 4.15: Bloque para comparación y multiplexado de distancias

en los que se comparan las N distancias que corresponden a los N vectores de pesos. En el segundo hay $N/4$ bloques que comparan las $N/2$ distancias resultantes del primer nivel. Así se continúa, de modo que la cantidad de bloques en un nivel es igual a la mitad de la del nivel previo, formando así un árbol binario. Si N es una potencia de dos, lo cual es lo usual, la comparación completa se llevará a cabo en $Log_2\ N$ niveles.

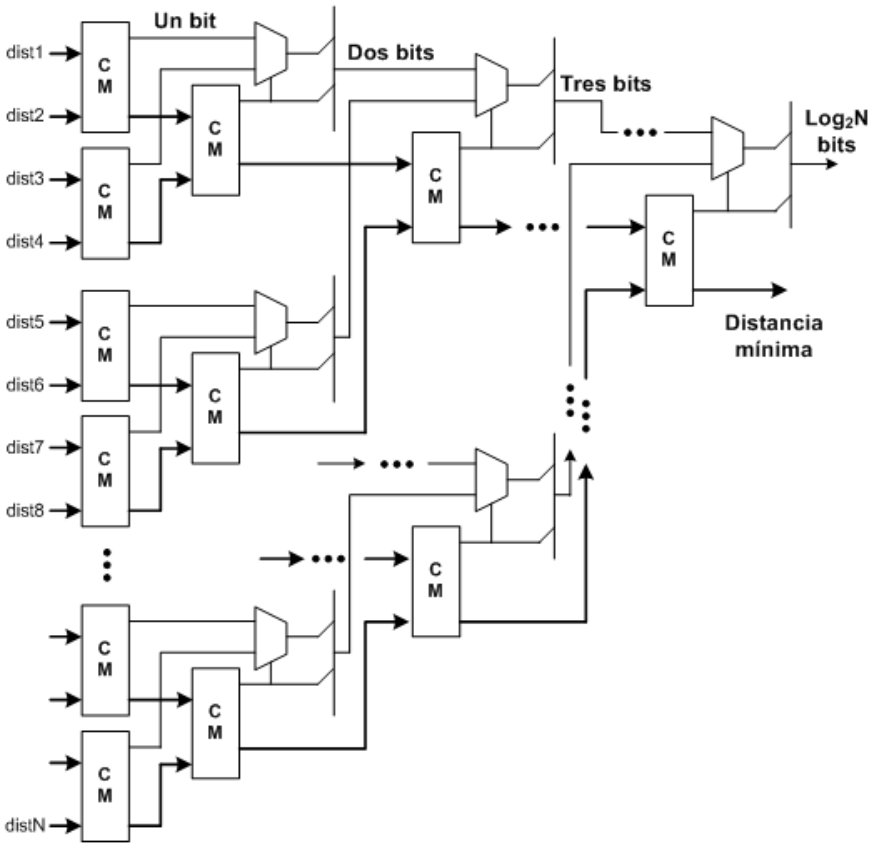


Fig. 4.16: Comparación en árbol

En la Fig. 4.15 se ilustra la descripción del bloque de comparación y multiplexado (CM) y en la Fig. 4.16 se muestra la estructura del comparador como un árbol formado a partir de esos bloques. En el árbol se observan dos ramas paralelas, a través de una de ellas (línea gruesa)

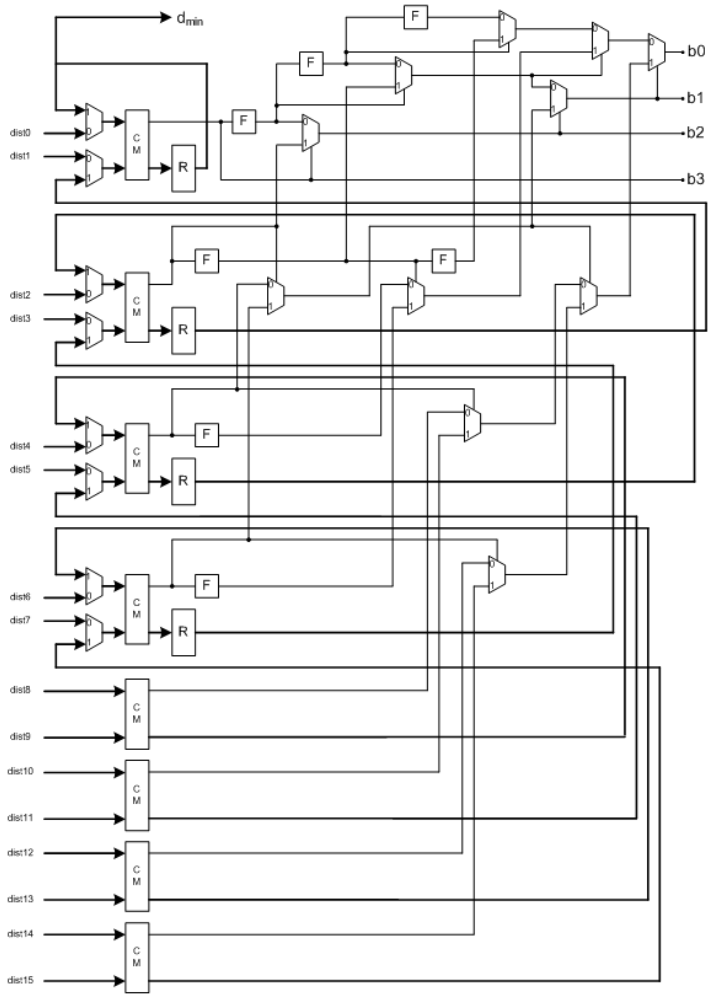


Fig. 4.17: Reutilización de bloques comparadores/multiplexores

se propagan las distancias mínimas obtenidas en cada bloque. Por la otra rama se tienen los índices obtenidos en cada bloque de modo que se van concatenando para ir conformando el índice correspondiente a la distancia mínima. En el primer nivel, este índice es solamente de un bit, pues sólo se han comparado dos distancias; en el segundo nivel, el índice es de dos bits ya que se han comparado cuatro distancias; finalmente el índice es de $Log_2\ N$ bits debido a que se han comparado N distancias.

Arbol de comparadores con reutilización

Como alternativa al árbol expandido de bloques comparadores/multiplexores se tiene un esquema basado en la reutilización de dichos bloques de manera que se requieren únicamente los del primer nivel. De este modo, se reducen los recursos demandados por el diseño, a costa de requerir $Log_2\ N$ ciclos de reloj para llevar a cabo la comparación. La Fig. 4.17 muestra el diagrama para este esquema correspondiente a un comparador de 16 distancias.

Cada uno de los bloques marcados con una F es un flip-flop en el que se memoriza el bit que señala cual de las distancias comparadas en el bloque CM correspondiente fue menor.

Esquema	Word parallel bit serial	Arbol de comparadores	Reutilización de comparadores
Flip flops	L*(N+1)	0	L*N/4+N/2-1
LUTs de 4 entradas	4*N+L*(N+1)	2*(N-1)*L+N/4	3*L*N/2
Cadencia (ciclos/comparación)	L	1	Log_2N

Tabla 4.2: Evaluación de comparadores

Estos flip-flops forman una cadena cuya longitud depende del número de veces que el bloque CM asociado se reutilice. Los bloques señalados con una R son registros que memorizan la distancia mínima obtenida en los bloques CM. Para los cuatro bloques CM inferiores no se requieren flip-flops ni registros, esto debido a que no son reutilizados. Los siguientes dos, de abajo hacia arriba, requieren sólo un flip-flop y el registro contendrá finalmente a su salida la distancia mínima obtenida en la primera comparación y en su entrada la de la segunda. El siguiente bloque se emplea en tres ocasiones y, por lo tanto, utiliza dos flips-flops; a la salida del registro se tendrá la distancia mínima de la segunda comparación y a la entrada la de la segunda. El bloque CM superior se usa cuatro veces, como consecuencia requiere tres flips-flops. Los multiplexores tienen como propósito conformar el índice correspondiente a la distancia mínima, de manera que este índice estará constituido finalmente por $b3$, $b2$, $b1$ y $b0$.

Evaluación de comparadores

En la Tabla 4.2 se evalúan los tres esquemas de comparadores descritos. Para la evaluación se toman en cuenta dos aspectos: la ocupación de recursos requeridos para su implementación mediante FPGAs (flip-flops y LUTs) y la cadencia de operación. La evaluación se realiza como función del número de vectores en el diccionario (N) y de la cantidad de bits considerados para comparar las distancias (L). La Tabla 4.3 concreta la evaluación para el caso de N=16 y L=12, incluyendo ocupación, frecuencia máxima de operación y throughput.

Analizando los datos de la Tabla 4.2, considerando únicamente el aspecto velocidad, se deduce que el esquema WPBS es el más adecuado cuando el tamaño de los datos a comparar es pequeño, esto debido a que su cadencia es independiente del número de número de neuronas. Por lo contrario, si el tamaño de los datos es grande, es más adecuado el esquema en árbol ya que la cadencia es independiente del tamaño de los datos. Comparando los dos esquemas en árbol, cuando se reutilizan los bloques comparadores se tienen dos ventajas: primero, la reducción en el uso de recursos y, quizá más importante, la reducción de la longitud de la trayectoria combinacional, con lo cual se incrementa considerablemente la frecuencia máxima de operación.

Por lo que se refiere a la ocupación de recursos, el esquema WPBS presenta un mayor equilibrio entre los requeridos para la lógica combinacional y los de la secuencial. Sin embargo, no se observa una diferencia notable entre la ocupación demandada por el esquema de árbol con reutilización y el WPBS.

Esquema	Word parallel bit serial	Arbol de comparadores	Reutilización de comparadores
Flip flops	208	0	55
LUTs de 4 entradas	262	364	288
Slices	150	196	168
Frecuencia máxima (megahertzios)	313	54	260
Throughput (comparaciones/μs)	26	54	65

Tabla 4.3: Ocupación, frecuencia máxima y throughput para N=16 y L=12

4.2.3 Memoria sináptica

Un aspecto importante en el diseño hardware de una red neuronal es el de la memoria en la que se almacenan los pesos sinápticos. Para el diseño basado en una FPGA, es interesante analizar las dos alternativas para implementar esta memoria, siendo éstas la basada en memoria distribuida y la que utiliza bloques de memoria. La elección entre estas dos alternativas debe tomar en consideración los siguientes aspectos: tamaño, latencia y ubicación.

La memoria distribuida se conforma mediante las LUTs de los bloques lógicos de la FPGA y, por lo tanto, es la adecuada cuando se requieren memorias de tamaño pequeño. La memoria en bloques, como su nombre lo indica, está estructurada con cierto tamaño, en cuanto a cantidad de bits, pero es posible configurar su profundidad y su ancho. Por ejemplo, para las familias Virtex 2 y Virtex 4 de Xilinx, cada bloque de memoria tiene 18 Kbits, con posibilidad de configurarlo como un array de 16K x 1, 8K x 2, 4K x 4, 2K x 9, 1K x 18 o 512 x 36. En particular, la Virtex 4 XC4VLX80 tiene 200 bloques.

Para la red SOM se requieren N memorias sinápticas de k lugares y P bits por lugar, siendo N el número de neuronas, k el número de unidades de entrada y P la precisión empleada para los pesos. Así, una SOM de 256 neuronas, 16 unidades de entrada y pesos con 16 bits utilizará 256 memorias de 16 x 16.

En tal caso, si se emplea memoria distribuida, se requerirán 4096 LUTs[5] para las 256 memorias, lo cual significa menos del 6% de las 71680 LUTs disponibles en la Virtex 4 XC4VLX80. Si se utiliza memoria en bloques directamente, los bloques disponibles serían insuficientes. Sin embargo, tomando en cuenta que los bloques tienen memoria de dos puertos totalmente independientes, es posible implementar dos memorias sinápticas en un sólo bloque. Siendo así, 128 bloques bastarían, sólo que su uso sería muy ineficiente, ya que únicamente se aprovecharían 512 bits de los 18 Kbits disponibles en cada bloque.

Otro inconveniente de los bloques de memoria resulta de que la lectura es necesariamente síncrona, lo que implica una latencia de un ciclo. Durante la fase de actualización de los pesos es necesario leer el valor actual del peso, emplear ese valor para calcular el nuevo e inmediatamente escribirlo para así realizar la actualización. Si la lectura es asíncrona, como ocurre con la memoria distribuida, esto se podrá realizar en un solo ciclo. Si la lectura es síncrona, se requerirán dos ciclos o bien hacer uso del doble puerto y llevar un doble direccionamiento.

[5]Las LUTs son de cuatro entradas y por tanto con una LUT se implementa una memoria de 16 x 1

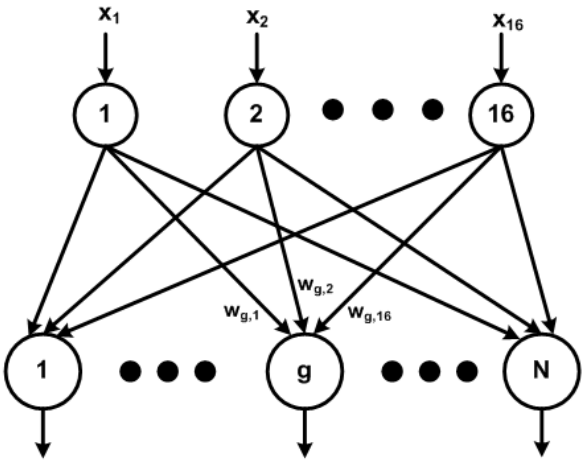


Fig. 4.18: Estructura de la red SOM diseñada

Un problema adicional de los bloques de memoria para esta aplicación tiene que ver con las tareas de emplazamiento y ruteado. Si se utiliza memoria distribuida, la memoria sináptica quedará alojada próxima a las unidades de procesamiento, lo cual facilitará esas tareas. No sucede así con los bloques de memoria, ya que éstos no se ubican de la manera tan dispersa como se encuentran las LUTs y, por lo tanto, la memoria sináptica estará alejada de las unidades de procesamiento y se complicará principalmente la tarea de ruteado.

4.3 Desarrollo de la red

Se diseñó una red neuronal SOM persiguiendo dos objetivos. Primero, la red debería ser capaz de codificar video en tiempo real mediante cuantización vectorial. Segundo, la red debería proporcionar entrenamiento on-line con el propósito de abarcar aplicaciones adaptivas. El diseño se orientó hacia una implementación completamente hardware en un dispositivo FPGA.

La Fig. 4.18 muestra la estructura de la red SOM diseñada. Consta de 16 unidades de entrada (la red recibe vectores con dimensión igual a 16) y N neuronas en la capa de salida (diccionario de tamaño N).

La Fig. 4.19 ilustra la arquitectura del hardware. El diseño se divide en tres secciones: el arreglo de unidades de procesamiento, el generador de direcciones para acceder tanto a los datos provenientes de la memoria externa donde se almacenan los vectores de entrada, como a los del almacenamiento de los pesos sinápticos, y la unidad de control. El arreglo de unidades de procesamiento se distribuye en módulos, cada uno con 16 unidades, y se tiene un máximo de 16 módulos (hasta 256 neuronas en la capa de salida). La cantidad de módulos requerida se selecciona al momento de la síntesis por medio del parámetro $indc\_lng$, cuyo valor es igual a $Log_2\ N$.

El bloque adicional cmp_glb compara las distancias mínimas obtenidas por los módulos. Su salida es el índice de la neurona ganadora (indice en la Fig. 4.19) con $indc\_lng$ bits. Este índice se divide en dos partes: (1) los $indc\_lng - 4$ bits más significativos o *índice global*, y (2) los cuatro bits menos significativos o *índice local*. El índice global (g_indc en la Fig. 4.19) determina en cual módulo está localizada la neurona ganadora y el índice local específica

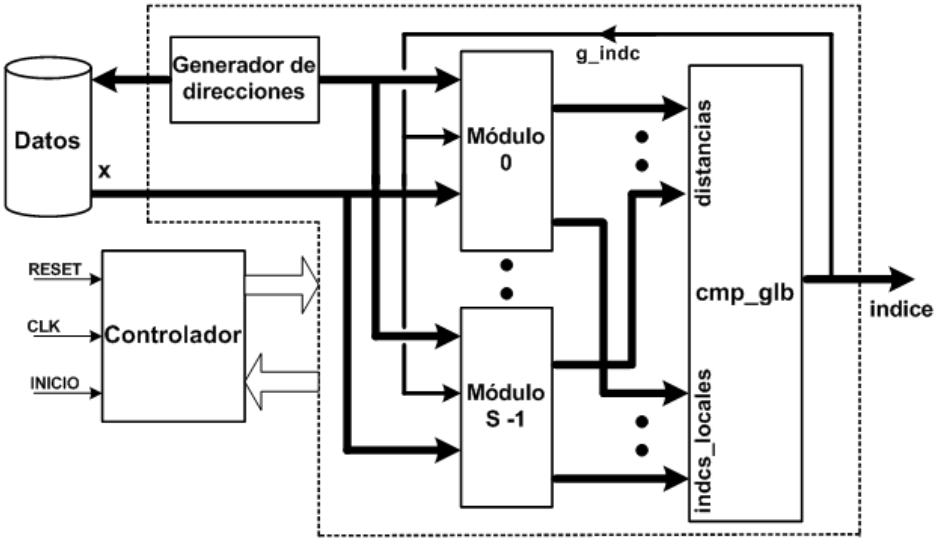


Fig. 4.19: Arquitectura del diseño

cuál es la neurona ganadora dentro de ese módulo. El índice global se realimenta a todas las unidades con la finalidad de utilizarse en la etapa de actualización de pesos.

A continuación se describirá la arquitectura del diseño considerando cada una de las secciones.

4.3.1 Generador de direcciones

Los vectores de entrada se encuentran almacenados en una memoria externa. Para procesar un vector de entrada, sus elementos se deben leer de la memoria y aplicarse en la entrada de la red, elemento por elemento, hasta que se haya barrido el vector completamente. Cada elemento del vector de entrada se procesa conjuntamente con el elemento respectivo del vector de pesos en todas las neuronas de la red. Durante la etapa de entrenamiento el barrido se realiza en dos ocasiones, la primera para determinar la neurona ganadora y la segunda para actualizar los pesos de la neurona ganadora y los de las neuronas que se encuentren dentro de su región de vecindad. El bloque generador de direcciones se construye mediante contadores con la finalidad de efectuar el barrido de todos los vectores del conjunto de entrenamiento y el barrido de los elementos del vector que se esté procesando, tanto para el vector de entrada como para los vectores de pesos.

4.3.2 Arreglo de unidades de procesamiento

Para esta etapa el diseño emplea la arquitectura SIMD con paralelismo a nivel componente. Es decir, existe una unidad de procesamiento en el circuito por cada neurona de la red y se procesan los vectores componente por componente. La Fig. 4.20 muestra el diagrama a bloques de un módulo. Cada uno de los módulos tiene tres entradas: x es el elemento del vector de entrada que se está procesando; addr es la dirección del lugar de la memoria de pesos sinápticos donde se encuentra almacenado el elemento correspondiente en todos los vectores de pesos y g_indc indica cuál es el módulo que contiene la neurona ganadora. Las salidas de

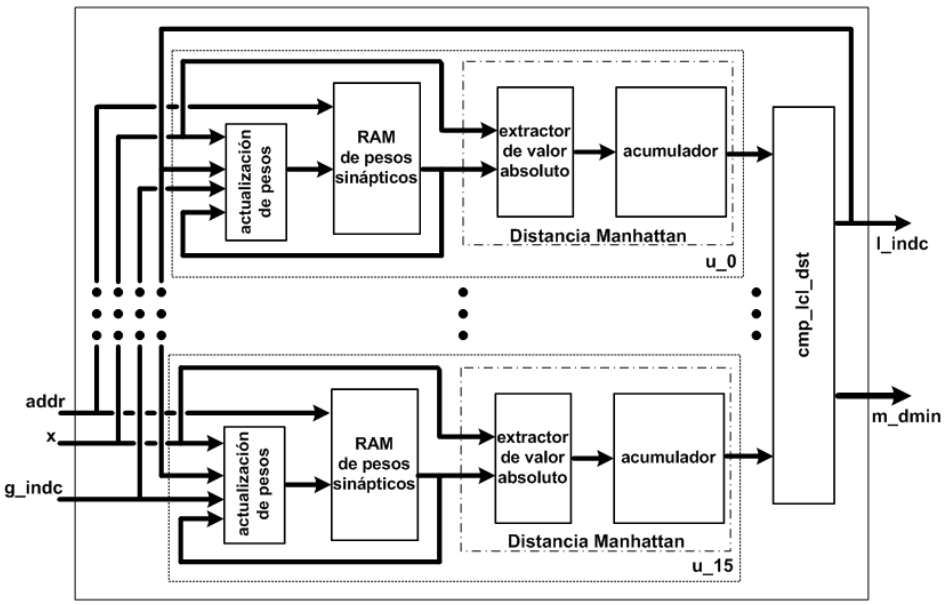


Fig. 4.20: Estructura de un módulo con 16 unidades de procesamiento

los módulos son el índice local, l_indc y la distancia que existe entre el vector de entrada y el vector de pesos de la neurona ganadora, m_dmin.

Cada módulo consta de 16 unidades de procesamiento (u_0 a u_15) y el comparador de distancias cmp_dst_lcl que determina cuál es la neurona ganadora dentro de ese módulo y genera el correspondiente índice local.

Cada unidad de procesamiento está formada por tres bloques: la memoria RAM para los pesos sinápticos, el actualizador de pesos y el evaluador de distancia Manhattan. A continuación se describirá cada uno de estos bloques.

El diseño de las unidades de procesamiento es genérico, tanto en lo correspondiente al tamaño de los datos de entrada como a la precisión empleada para los pesos sinápticos. Los pesos constan de una parte entera del mismo tamaño que los datos de entrada y una parte fraccionaria cuyo tamaño se establece, al igual que el de los datos de entrada, en tiempo de implementación. En este capítulo, para obtener resultados de la implementación y para la experimentación se emplean 8 bits para los datos (cantidad de bits con la que se representa la intensidad de los pixeles de una imagen) y 20 bits para los pesos sinápticos, de los cuales son 8 para la parte entera y 12 para la parte fraccionaria. La precisión utilizada para los pesos sinápticos es suficiente de acuerdo a un análisis previo del algoritmo de entrenamiento.

RAM para pesos sinápticos

Como se mencionó anteriormente, el diseño genérico se implementó para pesos sinápticos de punto fijo con 8 bits para la parte entera y 12 bits para la parte fraccionaria. Dado que la red tiene 16 unidades de entrada, para cada neurona se requiere un bloque de memoria con 16 lugares de 20 bits.

El aprendizaje de una red neuronal se puede realizar utilizando entrenamiento adaptativo o entrenamiento por lotes. El algoritmo original para la red SOM incluía únicamente el entrenamiento adaptativo, en el cual la actualización de los pesos se realiza inmediatamente

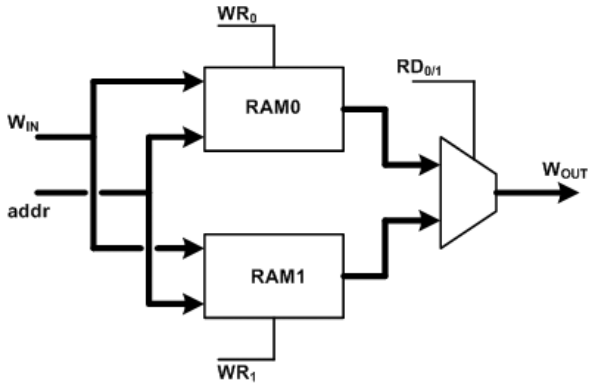


Fig. 4.21: Memoria de pesos sinápticos para entrenamiento por lotes

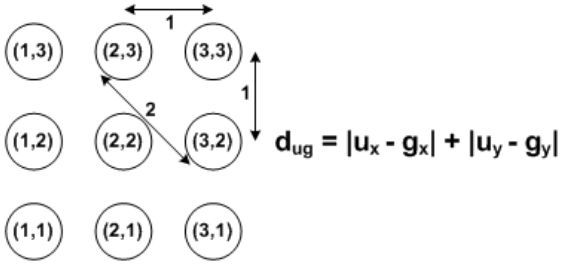


Fig. 4.22: Topología empleada para el mapa

después de que se procese un vector de entrada. Posteriormente Kohonen propuso el denominado algoritmo para mapeo por lotes [89], en el cual los pesos de la red se actualizan hasta que todo el conjunto de entrenamiento haya sido aplicado. La red SOM que se diseñó puede trabajar con las dos clases de entrenamiento, adaptativo o por lotes.

Para el entrenamiento por lotes los pesos sinápticos emplean dos bloques de RAM, de 16 lugares con 20 bits cada uno, como se muestra en la Fig. 4.21. Durante los epochs pares el bloque RAM0 se utiliza para determinar la neurona ganadora y la actualización de pesos se realiza sobre el bloque RAM1. Para los epochs nones se invierte el uso que se da a los bloques de memoria. Para el entrenamiento adaptativo sólo se utiliza un bloque de RAM.

Actualizador de pesos

De acuerdo al algoritmo de Kohonen para el entrenamiento de la red SOM , si g es la neurona ganadora cuando se aplica el vector de entrada x, entonces el peso de cualquier neurona u se actualiza usando la siguiente ecuación:

$$\mathbf{w}_u(t+1) = \mathbf{w}_u(t) + \alpha(t)\eta(t, d_{u,g})(x(t) - \mathbf{w}_u(t)). \tag{4.5}$$

En la Ec. 4.5, α es la tasa de aprendizaje y η es la función de vecindad. Los valores de α y η decrecen con el tiempo. Adicionalmente, la función de vecindad toma un valor más pequeño a medida que la neurona u esté más alejada de la neurona g. El término $d_{u,g}$ es la distancia entre las neuronas g y u.

La topología del mapa para la red SOM diseñada emplea un patrón rectangular y distancia Manhattan. El patrón rectangular, por su simplicidad, es el más apropiado para la implementación hardware. La topología se muestra en la Fig. 4.22.

Con la finalidad de simplificar el diseño, la razón de aprendizaje y la función de vecindad toman como valores únicamente potencias negativas de dos, de manera que las multiplicaciones se obtienen mediante operaciones de desplazamiento. De este modo, las funciones quedan descritas por las siguientes ecuaciones:

$$\alpha(t) = \begin{cases} 2^{-(k+\beta)} & \text{para} \quad 1 \leq k+\beta \leq 7 \\ 2^{-7} & \text{para} \quad k+\beta > 7 \end{cases}, \tag{4.6}$$

$$\eta(t) = \begin{cases} 1 & \text{si} \quad d_{u,g} = 0 \\ 0 & \text{si} \quad d_{u,g} > 1 \\ 2^{-1} & \text{si} \quad 1 \leq k+\beta \leq 6,\ d_{u,g} = 1 \\ 0 & \text{si} \quad k+\beta > 6,\ d_{u,g} = 1 \end{cases}. \tag{4.7}$$

En la Ec. 4.6 y Ec. 4.7, $k = \lfloor (t-1)/\delta \rfloor$, siendo t el número de epoch y δ una constante que se establece al momento de la síntesis, al igual que β. δ determina la rapidez con la cual desciende el valor de α, mientras que β define su valor inicial.

De este modo, cuando $\beta = 1$ y $\delta = 4$, la razón de aprendizaje es igual a 50% para los epochs 1 a 4; después decrece a 25% y permanece con ese valor para los epochs 5 a 8; posteriormente continúa siendo dividido entre dos cada 4 epochs hasta que alcanza un valor constante en el epoch 29. Por su parte, la función de vecindad es igual a 1 cuando $d_{ug} = 0$, es decir la neurona cuyo peso se va a actualizar es la ganadora, y es igual a cero para todas las neuronas que están alejadas de la ganadora una distancia mayor que 1. Para las neuronas inmediatamente adyacentes a la ganadora ($d_{ug} = 1$), la función de vecindad es igual a 0,5 para los epochs 1 a 24 y posteriormente se hace igual a 0. Adicionalmente, la función de vecindad es igual a cero para todas las neuronas que no están en el mismo módulo que la ganadora; de esta manera, la red no genera un solo mapa, sino $N/16$ mapas, uno por cada módulo.

La implementación hardware de la Ec. 4.5, utilizando los parámetros establecidos por la Ec. 4.6 y la Ec. 4.7, es muy simple. Los valores de $x(t) - \mathbf{w}_u(t)$ se desplazan hacia la izquierda una, dos y hasta siete veces. Los valores desplazados se conectan a las respectivas entradas de un multiplexor. La salida del multiplexor se selecciona con base en el número de epochs que han transcurrido y en el valor de la función de vecindad. Finalmente, la salida del multiplexor se suma con $\mathbf{w}_u(t)$ para obtener el nuevo valor de este vector de pesos. Tanto el restador como el sumador tienen una longitud de 20 bits.

Evaluador de la distancia Manhattan

Para medir la diferencia entre dos vectores se utiliza la distancia Manhattan. Como se mencionó anteriormente, ésta también se utiliza para obtener la distancia entre las neuronas requerida para la función de vecindad. Por lo tanto, la diferencia entre los dos vectores x y y de dimensión k se calcula con base en la ecuación siguiente:

$$d_{x,y} = \sum_{i=1}^{k} |x_i - y_i| . \tag{4.8}$$

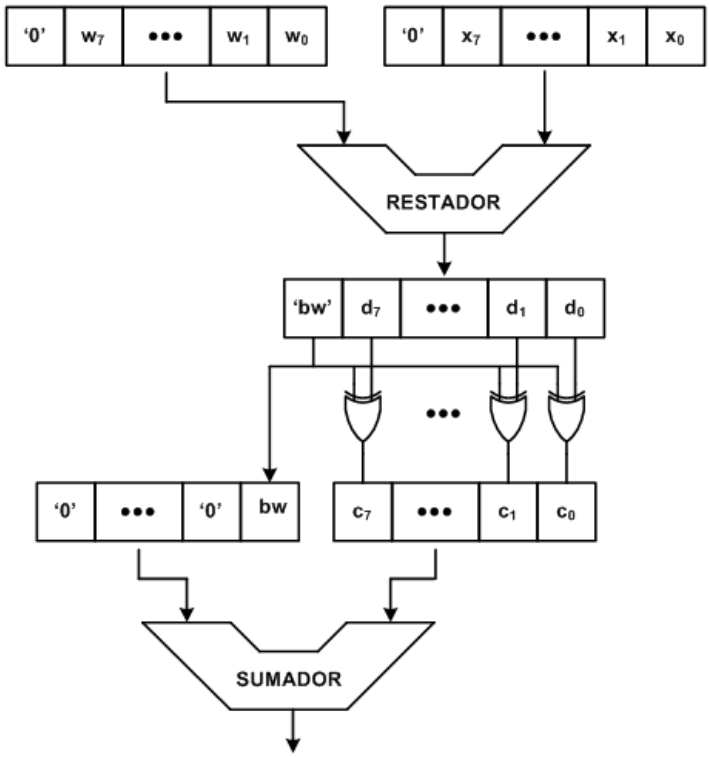


Fig. 4.23: Extractor de valor absoluto

La implementación hardware del evaluador de distancia Manhattan se obtiene de manera directa y está formado por el extractor del valor absoluto de las diferencias y el acumulador de valores absolutos. La Fig. 4.23 muestra la estructura del extractor de valores absolutos de las diferencias. La componente a procesar del vector de entrada y la parte entera de la componente respectiva del vector de pesos se extienden un bit (de 8 a 9 bits) y después se restan. El bit más significativo de la diferencia así obtenida es el bit de borrow. Si éste es '0', el valor absoluto será igual a la diferencia; si el bit de borrow es '1', entonces el valor absoluto es igual al complemento a dos de la diferencia.

Debido a que para el diseño se consideraron vectores de dimensión igual a 16 y ya que los valores absolutos son de 8 bits, el acumulador que produce la distancia Manhattan es de 12 bits.

Cabe hacer notar que en el diseño realizado se tiene un solo restador cuyo resultado se utiliza tanto para obtener la distancia Manhattan como para calcular el ajuste que se aplica para la actualización de pesos. El restador recibe como sustraendo los 20 bits de la componente del peso que se está procesando y como minuendo la componente respectiva del vector de entrada desplazada hacia la izquierda 12 bits. La salida completa del restador (21 bits) se emplea en el bloque de actualización de pesos y solamente se utilizan los 9 bits más significativos para obtener el valor absoluto de la diferencia. En otras palabras, un solo restador está compartido por estas dos secciones para reducir la ocupación de recursos. Esto es relevante debido a que el ahorro se multiplica por la cantidad de neuronas de la red.

Comparador de distancias

Para los comparadores de distancias (cmp_lcl_dist en la Fig. 4.20) se empleó el esquema de árbol de comparadores/multiplexores con reutilización que se describió en la sección 4.2.2. Para cada uno de los módulos se tiene un árbol de cuatro niveles. De este modo, en el primer nivel habrá 8 comparadores/multiplexores, cada uno recibe un par de distancias, una asociada a un índice par y una asociada a un índice non. En su salida se tendrá la distancia menor y un bit que será '0' si ésta fue la correspondiente al índice par y '1' en caso contrario. Las distancias se van propagando a través del árbol al mismo tiempo que los bits se van concatenando, hasta que finalmente, a la salida del cuarto nivel, se tendrá la distancia mínima y el índice de 4 bits correspondiente.

Para el comparador global (cmp_glb en la Fig. 4.18) se utiliza el mismo esquema, sólo que en este caso la cantidad de niveles depende del número de neuronas de la red, de acuerdo a la siguiente relación:

$$C_{niv} = Log_2\, N \; - \; 4. \tag{4.9}$$

Aprendizaje sensible a la frecuencia ganadora

Con la finalidad de reducir la dependencia del algoritmo de aprendizaje de la red SOM respecto a la inicialización de los vectores de pesos, se incluyó en la implementación la técnica de sensibilidad a la frecuencia ganadora de las neuronas. Para esto, se integró para cada unidad de procesamiento un contador que se activa cada vez que la respectiva neurona resulta ganadora. La salida de cada contador se suma con la correspondiente distancia Manhattan calculada y el resultado de la suma es el que se emplea para la comparación.

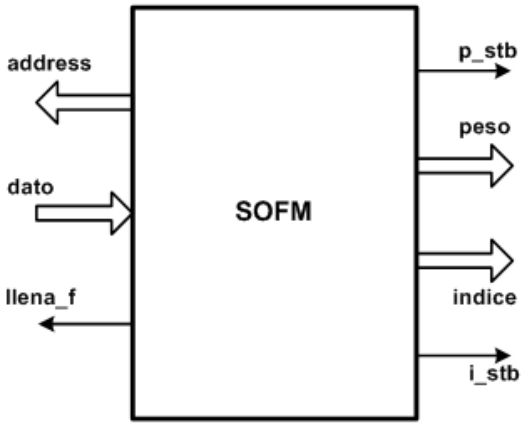


Fig. 4.24: Interface de la red

4.3.3 Interface de la red

La red requiere una memoria externa de la cual tomará los valores iniciales de los vectores de pesos y los vectores de entrada a procesar, pudiendo ser éstos pertenecientes al conjunto de entrenamiento o bien formados por algún bloque de una imagen a comprimir. Asimismo, la memoria se podría utilizar para almacenar los pesos resultantes del entrenamiento y para guardar los índices obtenidos al procesar una imagen.

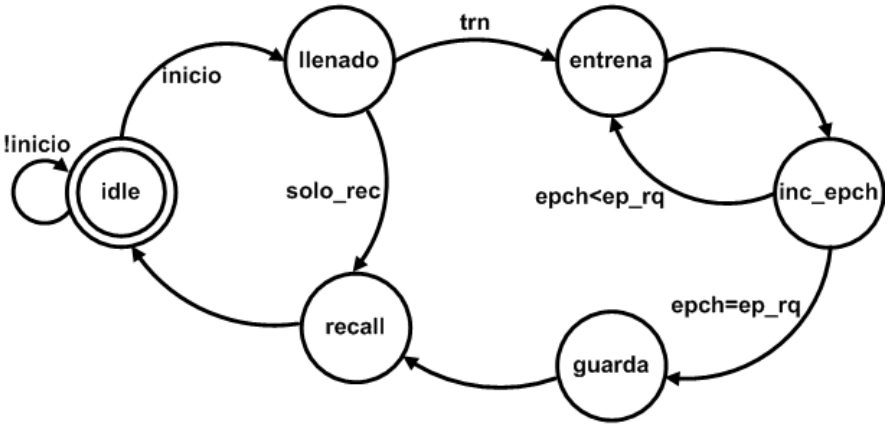


Fig. 4.25: Máquina de estados para el controlador de la SOM

Claro está que en la operación de la red para procesar imágenes en tiempo real tal memoria no será necesaria, pues los vectores de pesos se habrán llenado previamente y estarán almacenados en una memoria ROM, mientras que los vectores a procesar llegarán uno a uno, elemento por elemento, y el índice de la neurona ganadora para cada vector de entrada se transmitirá inmediatamente después de que éste haya sido procesado por la red.

En la Fig. 4.24 se muestra la interface de la red con el exterior considerando la existencia de una memoria externa en la que se tienen almacenados los pesos iniciales de la red y los vectores de entrada a procesar. La señal llena_f, cuando está activa, indica que la operación de la red se encuentra en la fase de llenado de los vectores de pesos y, por lo tanto, que por la entrada de datos deben llegar valores de peso. La señal i_stb se activa para indicar que en la salida indice está disponible el índice que resultó al procesar el vector de entrada más reciente, mientras que la señal p_stb indica que en la salida peso se encuentra un valor de peso resultante del entrenamiento, listo para ser almacenado.

4.3.4 Controlador

El controlador de la red es una máquina de estados finitos. La Fig. 4.25 muestra el diagrama de transición de estados con un alto nivel de abstracción. Cada estado del diagrama corresponde a una etapa de operación de la red, la cual está regida por su correspondiente sub-máquina de estados.

Durante la etapa *llenado* se llenan los vectores de pesos de la red. Los valores de los pesos serán aleatorios en el caso en que la red vaya a operar en la fase de entrenamiento o bien serán los valores obtenidos de un entrenamiento previo si es que va a trabajar únicamente en la fase de recall. Estos valores estarán almacenados en la memoria externa, por lo cual sus localidades serán barridas, una a una, para ir leyéndolos y almacenándolos en las memorias RAM de pesos sinápticos, vector por vector y peso por peso. Es en esta etapa cuando se activa la señal llena_f antes mencionada.

Una vez que los vectores de pesos están llenos, la operación de la red puede pasar a la etapa de entrenamiento o a la de recall. En la etapa de entrenamiento se completa un epoch, a continuación se incrementa el contador de epochs y si aun no se han completado los requeridos se ejecuta uno más, continuando así hasta alcanzar la cantidad requerida. Concluido el entrenamiento la operación pasa a la etapa *guarda* en la cual se transmiten los

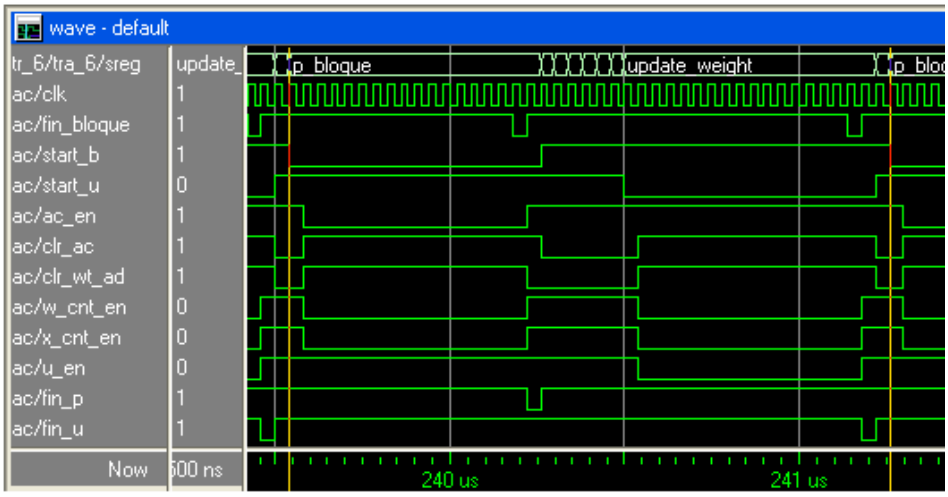


Fig. 4.26: Diagrama de tiempo para el procesamiento de un vector en la fase de entrenamiento

vectores de pesos generados por el entrenamiento para que sean almacenados externamente. Finalmente se realiza la etapa *recall* con todos los vectores de entrenamiento para generar así un índice por cada uno de aquéllos o bien con los vectores formados con todos los bloques de la imagen que se vaya a procesar. Durante el recall la red va proporcionando el indíce de la neurona ganadora cada vez que procese un vector de entrada, repitiéndose la operación hasta procesar toda una imagen.

El procesamiento de un vector se realiza componente por componente. Para el caso del entrenamiento se requieren dos pasadas, la primera para determinar la neurona ganadora y la segunda para actualizar los pesos de la ganadora y de las que se encuentren ubicadas dentro de su vecindad. Durante la primera pasada los valores absolutos de las diferencias entre las componentes del vector a procesar y las de los vectores de pesos se van acumulando secuencialmente, por lo cual se requiere un ciclo de reloj por cada componente. Cuando los valores absolutos de las k componentes han sido acumulados, entonces se tendrán disponibles las N distancias y el proceso continuará con su comparación para determinar la neurona ganadora. La comparación, ya que emplea el esquema de árbol de comparadores/multiplexores con reutilización, tomará $indc\_lng$ ciclos adicionales.

La Fig. 4.26 corresponde al diagrama de tiempo para el procesamiento de un vector en la fase de entrenamiento para una red con 64 neuronas y vectores de dimensión igual a 16. La primera gráfica indica el estado en el que se encuentra la máquina. Durante 16 ciclos de reloj la máquina permanece en el estado *p_bloque*, un ciclo por cada componente del vector. Posteriormente la máquina transcurre por los 6 estados requeridos para la comparación, con un ciclo de máquina por cada uno de ellos. Finalmente se tiene el estado *update_weight* en el cual permanece durante otros 16 ciclos de reloj. Así, en total el procesamiento de un vector requiere 38 ciclos de reloj. En general, para una red con N neuronas y vectores con dimensión igual a k, la cantidad de ciclos de reloj necesarios para entrenarla con un vector será $Log_2\ N + 2k$.

Para la operación de la red en la fase de recall no se requiere la etapa de actualización de pesos. Por lo tanto, sólo serán necesarios $Log_2\ N + k$ ciclos de reloj para el procesamiento de un vector de entrada. Además, para esta fase es apropiado efectuar el proceso de comparación de las distancias obtenidas para un vector de entrada simultáneamente con el cálculo de las

distancias correspondientes al siguiente vector, para lo cual únicamente es necesario registrar las distancias al momento que finalice su cálculo. Considerando que $Log_2\ N < k$, la cadencia en la fase de recall es k ciclos por vector y la latencia es igual a $Log_2\ N + k$ ciclos de reloj.

4.4 Implementación, pruebas y resultados

En esta sección se presentan algunos aspectos acerca de la implementación de la red.

4.4.1 Ocupación y frecuencia de operación

El hardware para la red SOM desarrollada fue diseñado utilizando VHDL. Para la implementación se utilizó la FPGA XC4VLX80-11FF1148 de la familia Virtex 4 de Xilinx. La síntesis se realizó utilizando la herramienta Synplify de Synplicity y la implementación usando el ISE de Xilinx.

Con este dispositivo fue posible implementar una SOM con hasta 256 neuronas, para vectores de dimensión 16, con datos de entrada de 8 bits y precisión de 20 bits para los vectores de pesos.

La Tabla 4.4 muestra los datos de ocupación de los recursos del dispositivo según el reporte después del emplazado y ruteado.

Recurso	Utilizado	Disponible	Utilización
Flip-flops	6952	71680	9%
LUTs	67252	71680	93%
Slices	34388	35840	95%
Compuertas equivalentes	1202026		

Tabla 4.4: Ocupación de recursos para la red completa

La máxima frecuencia de operación reportada después del emplazamiento y ruteado sobre la FPGA fue de 77 MHz. El diseño se implementó también con tecnología standard-cell[85] de 0,35 μm, con lo cual la frecuencia máxima de operación obtenida fue de 100 MHz.

Neuronas	Bits por índice	Entrenamiento			Recall		
		Q_{tc}	MCUPS		Q_{rc}	MCPS	
			75 MHz	100 MHz		75 MHz	100 MHz
16	4	36	533	711	20	960	1280
32	5	37	1038	1384	21	1829	2438
64	6	38	2021	2695	22	3491	4655
128	7	39	3938	5251	23	6678	8904
256	8	40	7680	10240	24	12800	17067

Tabla 4.5: Rendimiento de la red en MCUPS y MCPS

La Tabla 4.5 recopila el rendimiento en velocidad de la red SOM desarrollada, tomando en cuenta 1, 2, 4, 8 y 16 módulos. Para la red con 16 módulos (256 neuronas), el circuito basado en tecnología de standard-cell alcanza 10240 MCUPS (millones de conexiones actualizadas por segundo) en la fase de entrenamiento y 17067 MCPS (millones de conexiones procesadas por segundo) en la fase de recall, cuando trabaja con una frecuencia de 100 MHz. Trabajando a 75 MHz, los valores correspondientes son 7680 MCUPS y 12800 MCPS.

El cálculo de las MCUPS y MCPS se efectúa por medio de las ecuaciones siguientes:

$$\begin{aligned} MCUPS = \quad & kN/Q_{tc}T = \quad kNf/Q_{tc}, \\ MCPS = \quad & \qquad\qquad\quad kNf/Q_{rc}. \end{aligned} \tag{4.10}$$

Siendo k el número de entradas de la red, N el número de neuronas, T el periodo de reloj (en microsegundos), $f = 1/T$ la frecuencia de operación (en megahertzios), Q_{tc} la cantidad de ciclos de reloj requeridos para entrenar la red con un vector y Q_{rc} los requeridos para que la red procese un vector en la fase de recall.

En este caso, $k = 16$, N es la primera columna de la Tabla 4.5, Q_{tc} es igual a 32 más la cantidad de bits necesarios para expresar los índices (segunda columna) y Q_{rc} es igual a esa misma cantidad de bits más 16.

Recurso	Utilizado	Disponible	Utilización
Flip-flops	6825	71680	9%
LUTs	16972	71680	23%
Slices	9093	35840	25%
Compuertas equivalentes	438060		

Tabla 4.6: Ocupación de recursos para la red reducida

Al realizar la implementación de la red después de eliminar los bloques empleados sólo para la actualización de los pesos de la red y, por lo tanto, con posibilidad de operar únicamente en la fase de recall, la ocupación utilizando el mismo dispositivo es la que se muestra en la Tabla 4.6. La frecuencia máxima de operación alcanzada para esta red simplificada es de 125 MHz. De este modo, el rendimiento de la red con 256 neuronas operando con esta frecuencia es de 32000 MCPS considerando que el cálculo de las distancias y su comparación para vectores de entradas consecutivos se realiza de manera simultánea.

4.4.2 Comparación del rendimiento en velocidad con otros diseños

El rendimiento de la red, en cuanto a velocidad se refiere, fue evaluado al compararlo con los resultados reportados para tres redes SOM[86][87][88] implementadas en hardware, las cuales fueron comentadas en el Capítulo 3.

La Tabla 4.7 muestra la comparación de los resultados. Con la finalidad de que los datos sean comparables, todos corresponden al caso de una red con 256 neuronas y vectores con dimensión igual a 16 y valores de 8 bits. De acuerdo con algunos autores [90] en la tabla se ha incluido como métrica del rendimiento los MCUPS*, que se calculan utilizando la siguiente ecuación:

Diseño	Hikawa[86]	Hendry[87]		Tamukoh[88]	Desarrollada	
Tecnología	FPGA Virtex 2	Std-cell 0,65 μm	Std-cell 0,18 μm	FPGA Virtex 2	FPGA Virtex 2[7]	Std-cell 0,35 μm
MCPS	800	1321	11070	10240	11947	17067
MCUPS	400	661	nda	10240	7168	10240
N_U	9	nda	nda	1	5	5
MCUPS*	14	–	–	40	140	200
Frec. (MHz)	200,2	50	200	70	70	100

Tabla 4.7: Comparación de la red con otros diseños

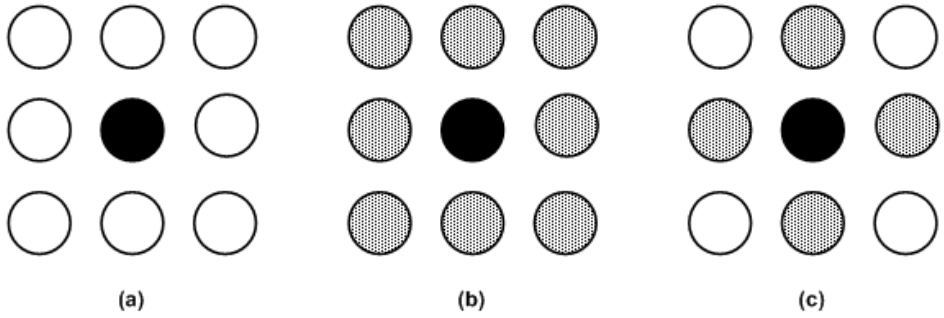


Fig. 4.27: Regiones de vecindad para tres de los diseños comparados (a) Tamukoh, (b) Hikawa, (c) Desarrollada

$$MCUPS* = kN_U/Q_{tc}T = \quad kN_U f/Q_{tc} \ . \tag{4.11}$$

Siendo N_U el número de neuronas realmente actualizadas al procesar un vector durante el entrenamiento. Cabe resaltar que esta métrica es más significativa, ya que la empleada tradicionalmente (los MCUPS) proporciona una idea equivocada del rendimiento pues hace referencia a todas las conexiones de la red cuando en realidad sólo son unas cuantas las que el circuito es capaz de actualizar. En este caso, para los diseños que se comparan las neuronas cuyos pesos se actualizan son las que se muestran en la Fig. 4.27. En negro se muestra la neurona ganadora y sombreadas las que se consideran dentro de la región de vecindad. Para el diseño de Hendry no se tiene disponible esa información. De la Fig. 4.27 resultan los valores de N_U que se encuentran en la tabla.

4.4.3 Experimentación de la red como codificador

Calidad de la red como compresor de imágenes

La experimentación se realizó empleando imágenes en tonos de gris de 256 por 256 pixeles. En particular se emplearon tres imágenes que han sido ampliamente consideradas en los trabajos

[7]La red desarrollada se implementó sobre una Virtex 2 XC2V6000 sólo con el propósito de realizar esta comparación. La frecuencia máxima de operación sobre este dispositivo es de 70 MHz y a ésta corresponden los cálculos

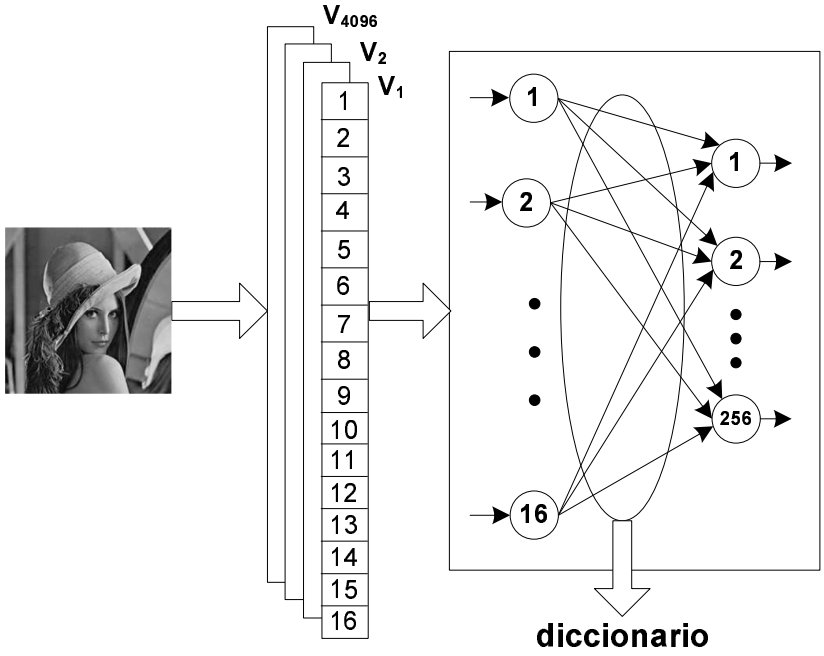


Fig. 4.28: Entrenamiento con los vectores generados con Lena

de investigación en el campo de la compresión de imágenes. Estas son Lena, Bird y Goldhill.

Cada imagen se descompuso en bloques de 4 por 4 pixeles, cuyos valores de intensidad se arreglaron en vectores de dimensión 16 como datos de 8 bits. De esa manera, cada imagen quedó representada por 4096 vectores con las características señaladas.

La red con 256 neuronas se entrenó utilizando como conjunto de entrenamiento los 4096 vectores generados con la imagen Lena, tal como se ilustra en la Fig. 4.28. El entrenamiento se realizó durante 100 epochs, teniendo la razón de aprendizaje un valor inicial de 0,5 ($\beta = 1$) y reduciéndose a la mitad cada cuatro epochs ($\delta = 4$), de acuerdo con el comportamiento que se describió en la sección 4.3.2. El entrenamiento se realizó en el modo adaptativo.

Para el entrenamiento, los vectores de pesos se inicializaron con valores aleatorios en el rango comprendido entre 120 y 135. El entrenamiento durante los 100 epochs, operando la red con una frecuencia de reloj de 75 MHz, se completó en 218,5 milisegundos.

La gráfica de la Fig. 4.29 muestra la manera como evolucionó el aprendizaje de la red. El eje vertical corresponde a la distorsión representada por el PSNR en decibelios y el eje horizontal a la cantidad de epochs.

Como resultado del entrenamiento se generó un diccionario para el cuantizador vectorial que se usará para la compresión de imágenes. El diccionario está compuesto por los 256 vectores de pesos correspondientes a las 256 neuronas de la red.

Posteriormente, la red así entrenada se utilizó en la fase de recall operando de esta manera como cuantizador vectorial. La cuantización se aplicó sobre los 4096 vectores generados con la imagen Lena, con lo cual se obtuvieron los 4096 valores de índice correspondientes. Este conjunto de valores del índice constituye la imagen codificada con una razón de compresión de 16:1 o 0,5 bpp, que se calcula con la Ec. 4.12. Finalmente se empleó el diccionario de 256 palabras para efectuar la de-cuantización vectorial, obteniendo de esa manera la imagen reconstruida. La Fig. 4.30 ilustra este proceso.

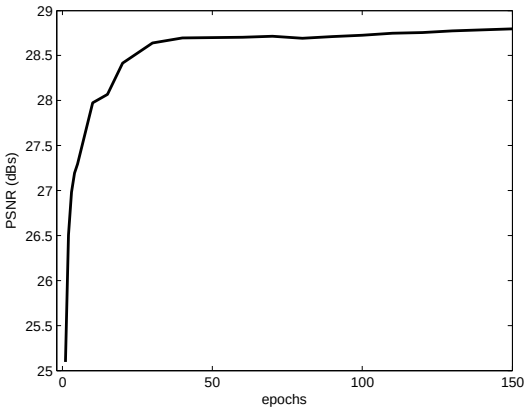


Fig. 4.29: Evolución del aprendizaje

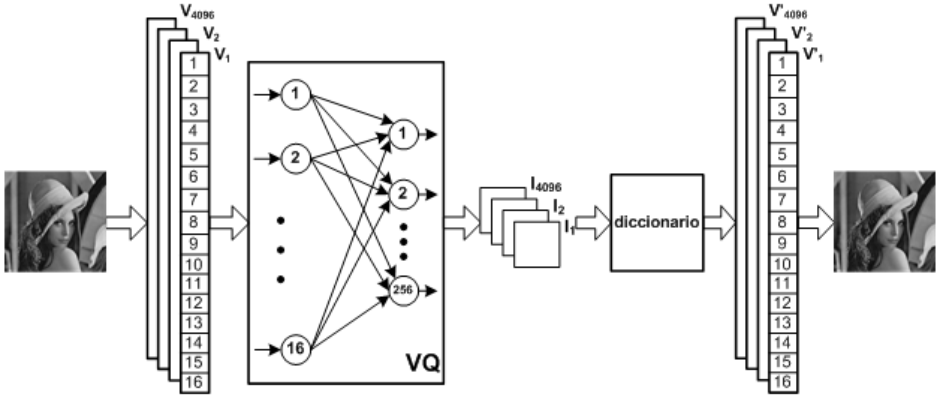


Fig. 4.30: Proceso de codificación - reconstrucción

$$CR = \frac{8k}{log_2\ N} = \frac{8 * 16}{log_2\ 256} = 16 \tag{4.12}$$

Con el propósito de evaluar la capacidad de generalización de la red, se aplicó el proceso de codificación-reconstrucción también sobre las imágenes Bird y Goldhill. Después, utilizando como métrica de evaluación el *valor pico de la relación señal a ruido* (PSNR), se calculó la distorsión que introdujo el proceso de codificación-reconstrucción para las tres imágenes.

Con la finalidad de contar con una referencia para evaluar el diccionario obtenido mediante el entrenamiento de la red, se utilizaron otros dos diccionarios de igual tamaño generados por medio del algoritmo LBG, uno empleando distancia Manhattan y otro con distancia Euclidiana. Estos diccionarios se generaron con entrenamientos de 100 iteraciones para cada diccionario parcial[8]. El proceso de generación tomó 459 segundos cuando se ejecutó una implementación en Matlab del algoritmo LBG con distancia Euclidiana y 309 segundos con distancia Manhattan, en ambos casos utilizando un procesador Xeon 3040 a 1.86 GHz.

En la Tabla 4.8 se presentan para su comparación los valores del PSNR en decibelios

[8]Al diccionario final se llega a través de diccionarios parciales generados siguiendo el ciclo de bipartición-entrenamiento descrito en 2.5.3

Diccionario	Lena	Bird	Goldhill
Red generada con la SOM	29,089	31,391	25,451
LBG, Manhattan	29,009	31,448	25,338
LBG, Euclidiana	29,218	31,490	25,503

Tabla 4.8: Comparativa de diccionarios

obtenidos a partir de la codificación-reconstrucción de las imágenes de Lena, Bird y Goldhill usando los tres diccionarios, el generado por el entrenamiento de la red y los dos producidos por el algoritmo LBG, uno con distancia Manhattan y otro con distancia Euclidiana. En los nueve casos la codificación se realizó con la red SOM operando en la fase de recall, llenando previamente los vectores de pesos con el diccionario correspondiente.

De la Tabla 4.8 se desprende que en la calidad del compresor influye la función empleada para evaluar la distancia entre vectores. Para las tres imágenes consideradas, el valor del PSNR se reduce cuando se usa la distancia Manhattan respecto al obtenido con la Euclidiana; en el peor caso la reducción es de 209 milésimas de decibelio.

Comparando las dos primeras filas se observa que la diferencia entre el compresor que utiliza el diccionario generado por la red y el que emplea el diccionario generado por el algoritmo LBG con distancia Manhattan es mínima. Para dos imágenes es superior el correspondiente a la red (alrededor de 100 milésimas de decibelio) y en el otro es inferior (menos de 50 milésimas). Cabe señalar que el algoritmo LBG trabaja con una implementación software utilizando variables con precisión de 64 bits, mientras que para la red se tiene una precisión de 20 bits por lo que se refiere al procesamiento de los pesos.

La calidad del compresor con el criterio de la percepción visual se puede juzgar comparando la imagen original con la imagen reconstruida. En la Fig. 4.31 se muestran las imágenes Lena y Bird originales en la parte superior, en la parte central las obtenidas a través del proceso de compresión-reconstrucción usando el diccionario generado por el entrenamiento de la red y en la parte inferior las producidas utilizando el diccionario generado por el algoritmo LBG. En todos los casos la compresión se realiza mediante la red trabajando como cuantizador vectorial en la fase de recall, lo que cambia es el diccionario utilizado.

Cabe hacer notar que la experimentación fue realizada con la red operando físicamente sobre la Virtex 4, utilizando para ello el banco de entrenamiento desarrollado que se describirá en el siguiente capítulo de esta tesis.

Codificación de video en tiempo real

Con la finalidad de valorar la capacidad de la red para codificar video en tiempo real, en la Tabla 4.9 se muestra el tiempo que toma la codificación de una imagen con diferentes formatos. Se incluyen dos casos, el de la red completa con capacidad de entrenamiento (f=75 MHz) y el de la red reducida utilizable sólo para el recall (f=125 MHz), incluyendo en este último caso el procesamiento simultáneo de la comparaciòn de distancias para un vector de entrada con el cálculo de las distancias para el siguiente vector. Todo esto para una red con 256 neuronas.

A partir de los resultados mostrados en la Tabla 4.9, se desprende que la red reducida operando como cuantizador vectorial con una frecuencia de 125 MHz puede codificar más de

Fig. 4.31: Percepción visual del proceso de compresión-reconstrucción

		Escala de grises 8 bits por pixel	Color 24 bits por pixel
f=75 MHz	256 x 256	1,31	3,93
	512 x 512	5,24	15,73
	640 x 480	6,14	18,43
	800 x 600	9,60	28,80
	1024 x 768	15,73	47,19
f=125 MHz	256 x 256	0,52	1,57
	512 x 512	2,10	6,29
	640 x 480	2,46	7,37
	800 x 600	3,84	11,52
	1024 x 768	6,29	18,87

Tabla 4.9: Tiempo para la codificación de una imagen (ms)

30 imágenes por segundo para todos los formatos considerados, mientras que la red completa lo puede lograr con excepción de las imágenes en color de 1024 x 768 pixeles. La codificación de imágenes con este último formato requeriría el empleo de dos redes, una para dos de los colores y la otra para el restante.

4.5 Conclusiones

Se diseñó una red SOM alcanzando el objetivo propuesto de procesar video en tiempo real y tener la capacidad de actualización de los pesos para poder encarar posibles aplicaciones que demanden adaptabilidad de la red.

Como antesala al desarrollo de la red, se analizaron varias arquitecturas y se revisaron diversos esquemas para sus bloques componentes principales.

La red desarrollada presenta las siguientes características y funcionalidades que la hacen susceptible para diversas aplicaciones.

1. Está estructurada en módulos con 16 neuronas.

2. La tasa de aprendizaje es configurable tanto en valor inicial como en rapidez de decaimiento.

3. El entrenamiento se puede realizar por lotes o en el modo adaptativo.

4. El entrenamiento incluye la técnica de sensibilidad a la frecuencia ganadora.

5. El diseño es genérico por lo que respecta al tamaño de los datos, la precisión de los pesos y el número de bits para el truncamiento de las distancias[9].

[9]Esta característica se empleó en un sistema de compresión que se expondrá en el siguiente capítulo

6. El diseño hardware ha sido incorporado como un neurocoprocesador en un ordenador anfitrión para integrar un banco de entrenamiento híbrido hardware/software.[10].

En cuanto a los resultados de la implementación y la experimentación cabe destacar los siguientes:

1. Se pudo implementar una red con 256 neuronas incluyendo su funcionalidad de aprendizaje en el dispositivo FPGA utilizado.

2. La red del mismo tamaño capaz de trabajar únicamente en la fase de recall demandó únicamente el 25% de los recursos de ese mismo dispositivo y una frecuencia máxima de operación 67% mayor.

3. La red desarrollada superó en velocidad a otros diseños similares reportados recientemente.

4. La calidad del compresor medida por los valores del PSNR después del proceso codificación-reconstrucción es aceptable para la razón de compresión alcanzada.

5. Como codificador de video, la red trabajando en la fase de recall satisface los requerimientos de velocidad para imágenes de hasta 1024 x 768 pixeles.

6. La experimentación se desarrolló físicamente sobre la FPGA utilizada.

Algunos resultados iniciales de la investigación en la que se sustenta este capítulo están reportados en el artículo con la siguiente referencia:
A. Ramirez, R. Gadea, R. Colom; "A hardware design of a massive-parallel, modular NN-based vector quantizer for real-time video coding"; *Elsevier Microprocessors and Microsystems*; **32**(1), 33-44, 2008.

[10]Este será uno de los temas que se desarrollará en el siguiente capítulo

Capítulo 5

Banco de entrenamiento y aplicaciones de investigación

Este capítulo tiene dos objetivos. Primero, exponer el desarrollo del banco de entrenamiento para la red SOM empleando la metodología de codiseño Simulink/System Generator para integrar un sistema en el que la red implementada sobre una FPGA opere como neurocoprocesador en el ordenador anfitrión. Este primer objetivo se planteó como uno fundamental desde la concepción del trabajo de investigación para esta tesis doctoral. Por lo tanto, esta sección está enfocada en este propósito e incluye el diseño del banco de entrenamiento y dos aplicaciones de investigación desarrolladas para comprobar su funcionalidad y las ventajas que aporta, principalmente en lo que se refiere al tiempo de entrenamiento.

El segundo objetivo de este capítulo es el diseño e implementación hardware de un sistema de compresión de imágenes que integra la transformada wavelet y la cuantización vectorial. Para la cuantización vectorial se utiliza la red SOM expuesta en el Capítulo 4.

5.1 Diseño del banco de entrenamiento para redes SOM

El desarrollo de la tecnología digital involucra sus dos campos fundamentales, el software y el hardware. Por el lado del software surgen lenguajes que responden a nuevos paradigmas de programación; a la vez aparecen entornos para el desarrollo de aplicaciones de propósito general u orientadas hacia áreas muy específicas; también se generan nuevos estándares o se actualizan los existentes para permitir el desarrollo de los sistemas cada vez más grandes, extensos y complejos a los que da lugar la incorporación de las comunicaciones.

Por la parte del hardware destaca el incesante avance en los procesos de fabricación de semiconductores, más rápidos, más pequeños, más baratos y con menor disipación de potencia, constatando aquello que previó Gordón Moore cuando dijo en 1975 que la cantidad de transistores por unidad de área se duplicaría cada 2 años [91]. Esto redunda en procesadores de propósito general más poderosos y en circuitos de aplicación específica asequibles ahora para un rango mayor de desarrolladores de equipos electrónicos, tanto en el área de la investigación como en la comercial. Los dispositivos programables, como las FPGAs, que permiten crear lógica reconfigurable y acortar los tiempos para que los diseños lleguen al mercado, aspecto de gran relevancia actualmente por la competencia existente, crecen en tamaño y en potencialidad al incorporárseles diversos bloques de procesamiento como se puede observar en la Tabla. 5.1 [92][93].

Característica	Virtex 4	Virtex 5
Tecnología de proceso	90 nm, triple capa	65 nm, triple capa
Voltaje de operación	1,2	1,0
LUTs	De 4 entradas	De 6 entradas
RAM distribuida	64 bits por CLB	256 bits por CLB
Reg. de desplazamiento	64 bits por CLB	128 bits por CLB
Gestión del reloj	500 MHz, DCM	550 MHz, DCM y PLL
Bloques RAM/FIFO	18 Kbits/bloque	72 Kbits/bloque
Bloques DSP, slices	Hasta 512 DSP48	Hasta 640 DSP48E
Bloques DSP, MAC	25 x 18 bits	18 x 18 bits
Bloques DSP, disipación	2,3 mW/100 MHz	1,38 mW/100 MHz
Conectividad serie	622 MBPS a 6.5 GBPS	500 MBPS a 6.5 GBPS
MAC Ethernet	10/100/1000 MBPS	10/100/1000 MBPS
	Hasta 2 PowerPC 405	Hasta 2 PowerPC 440
Procesadores	32-bits RISC	32-bits RISC
	680 DMIPS a 500 MHz	1100 DMIPS a 550 MHz
PCI Express	Implementación IP soft	Bloque interconstruido

Tabla 5.1: Comparación de características de Virtex 4 vs Virtex 5.

Un aspecto trascendente en el desarrollo del hardware es el relativo a los lenguajes para su descripción, conocidos genéricamente como *HDL* (*Hardware Description Language*). El método original de descripción a través de diagramas de bloques o de conexiones dio paso a un primer nivel de abstracción y de generalización mediante las *listas de conexiones* (*netlist*). A partir de allí surgieron los primeros lenguajes propiamente dichos y, básicamente a nivel RTL (Register Transfer Level), trajeron consigo la posibilidad de escribir diseños hardware de manera estructurada, dando lugar a esquemas *de arriba a abajo* (*top-down*), a través de descripciones funcionales más que por medio de componentes lógicos, aunque no se llegaron a estandarizar.

Posteriormente, aparecieron los primeros lenguajes que pronto alcanzaron un uso muy generalizado y, por lo tanto, su estandarización, VHDL y Verilog. Inicialmente estuvieron orientados hacia la simulación y verificación funcional de circuitos, aunque pronto fueron desarrolladas las herramientas para utilizarlos también para la síntesis e implementación. Posteriormente llegaron los lenguajes para hardware basados en *C*, hasta no hace mucho destinado exclusivamente al software, destacando SystemC [94] y Handel-C [95]. Estos lenguajes aportan un mayor nivel de abstracción y facilitan la descripción de las soluciones a nivel sistema [96].

Los avances en estos dos campos en realidad han estado fuertemente interrelacionados, pero actualmente lo están en mayor medida debido a la disponibilidad de herramientas cada vez más poderosas para integrar software y hardware. Las soluciones software han alcanzado un nivel muy alto en cuanto a funcionalidad, flexibilidad, amigabilidad para el usuario y costo,

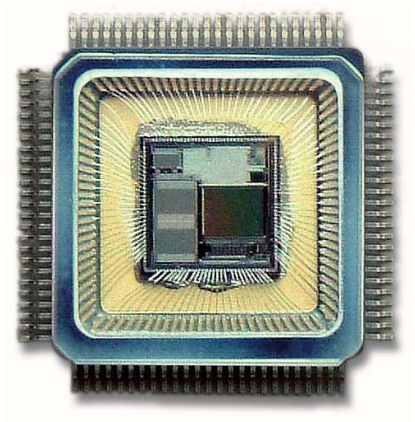


Fig. 5.1: Integración HW-SW a nivel IC

de modo que no es sencillo decidir reemplazarlas por soluciones hardware. Sin embargo, el empleo de las nuevas herramientas ha dado lugar al desarrollo de metodologías para soluciones híbridas.

Prácticamente, en estos días la primera pregunta que se plantea un diseñador es: ¿cuáles tareas debe desarrollar en hardware y cuáles en software?, a esto es lo que se le conoce como la *partición hardware/software*. En [97], el autor establece los cinco beneficios que se deben tomar en cuenta al momento de realizar la elección de una implementación hardware sobre una software: (1) relación rendimiento/costo; (2) tiempo para alcanzar el mercado; (3) confiabilidad y tolerancia a fallos; (4) flexibilidad y reusabilidad; (5) satisfacción del cliente; y (6) protección del diseño.

La integración hardware/software (HW-SW) para diseñar un sistema ocurre en dos categorías, a nivel circuito integrado o a nivel ordenador. En el primer caso tenemos lo que se ha denominado *SoC*[1] (System on Chip), en el que en un mismo dispositivo (ver Fig. 5.1[2]) conviven uno o varios procesadores que realizan ciertas tareas mediante la ejecución de un conjunto de programas desarrollados expresamente para cierta aplicación específica (software) y circuitos diseñados para llevar a cabo otras tareas complementarias (hardware).

En la segunda categoría de integración HW-SW, a nivel de ordenador, al procesador principal del sistema, a través de su estructura de buses, se conectan dispositivos hardware que hacen las veces de coprocesadores. La operación del procesador central está regida por el software de aplicación y el software del sistema. Por su parte, los coprocesadores llevan a cabo el procesamiento de datos de acuerdo con la estructura e interconexión de los bloques lógicos que los componen (Fig. 5.2[3]).

El codiseño HW-SW, en las dos categorías mencionadas, se ha extendido gracias a herramientas que permiten superar los problemas que surgen debido a la complejidad a que da lugar la mezcla de los dos campos del diseño digital. Uno de estos problemas consiste en que los diseños son más propensos a errores, pues éstos aparecen en el desarrollo del hardware, en el del software y, adicionalmente, al momento de su integración. Por lo tanto, los flujos tradicionales de diseño de HW y de SW operando de manera independiente no son suficientes

[1]Un SoC se puede definir como un circuito integrado de muy alta densidad que incluye al menos un procesador, estructura de buses del sistema, memoria caché y periféricos hardware propios.

[2]Imagen tomada de NC1503-VISoc de Neuricam, S. p. A.

[3]Imagen tomada de OctaPlex Inc.

Fig. 5.2: Integración HW-SW a nivel ordenador

y ha sido necesario generar tecnologías para establecer un puente entre ellos sin introducir nuevas fuentes de error. La Fig. 5.3 muestra el tradicional flujo de diseño RTL y la Fig. 5.4 corresponde al flujo de codiseño SoC.

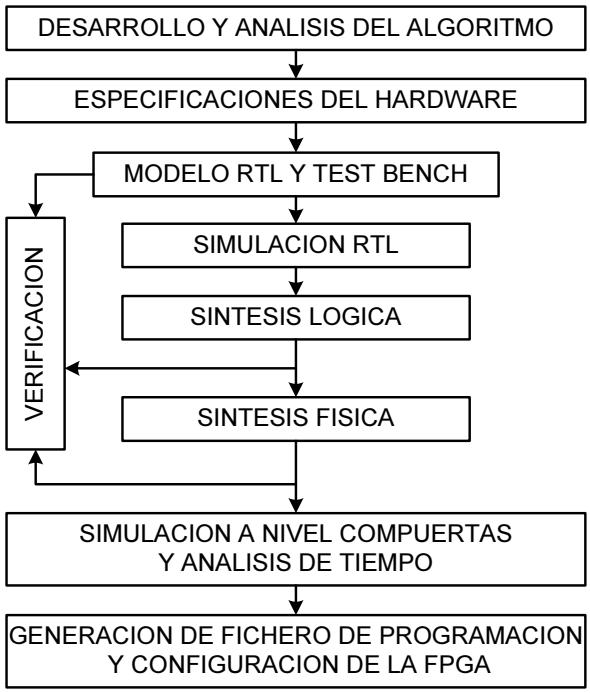


Fig. 5.3: Flujo tradicional de diseño RTL

Fundamentalmente el propósito de las herramientas de codiseño HW-SW es el proporcionar la administración unificada de las descripciones de hardware y software, así como la automatización de algunas tareas de diseño como son la simulación del sistema completo (*co-simulación*), la verificación integral (*co-verificación*) y la generación de la interface entre el hardware y el software. Tanto en el plano comercial como en el académico se han desarrollado

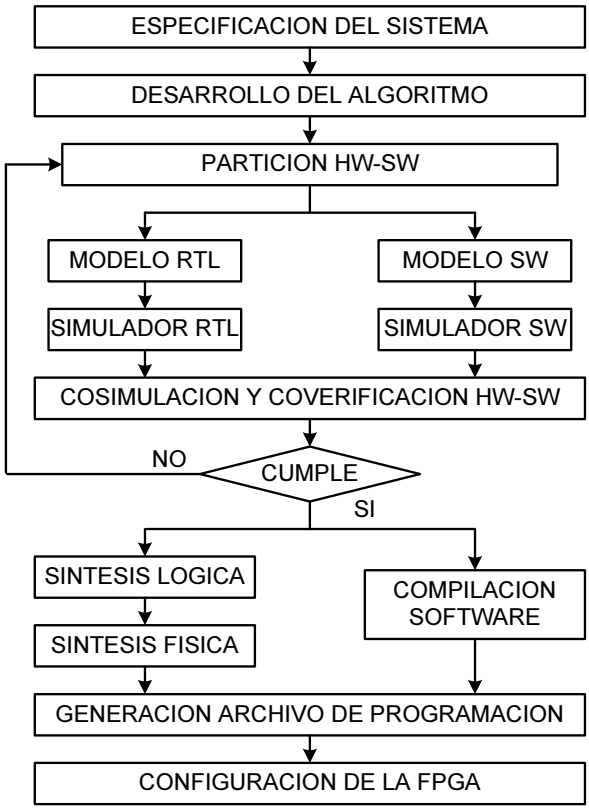


Fig. 5.4: Flujo de codiseño para SoCs

herramientas y metodologías para el diseño HW-SW [97][98][99][100].

El banco de entrenamiento de redes SOM desarrollado cae en la segunda categoría de integración HW-SW. En otras palabras, el sistema está integrado, por una parte, por el software ejecutándose en el procesador del ordenador anfitrión; este software consiste en una serie de programas para la lectura y almacenamiento de datos en disco duro, para el preprocesamiento necesario con la finalidad de proporcionar adecuadamente estos datos al coprocesador y para la recopilación de los resultados del entrenamiento, su análisis y presentación. Por la otra, estará la red neuronal implementada sobre una FPGA alojada en una placa que incluye también bancos de memoria y circuitos de interface con el bus PCI del sistema, conformando todo esto la parte hardware que actuará como neurocoprocesador.

5.1.1 Esquema de codiseño Simulink/System Generator

El codiseño HW-SW a nivel ordenador requiere integrar herramientas para (1) la descripción a nivel sistema; (2) la generación del código software; (3) la generación del código hardware; (4) la co-simulación HW-SW; (5) la implementación hardware, (6) la puesta en operación. Como antecedente de este esquema se tiene el trabajo reportado en [101]. Así, Xilinx desarrolló la herramienta denominada System Generator (SG) [102] con la finalidad de incorporarla con el Simulink de Matlab [103] y de esta manera integrar un esquema de codiseño HW-SW a nivel ordenador.

Como es bien sabido, Simulink es un entorno gráfico que permite el diseño y simulación

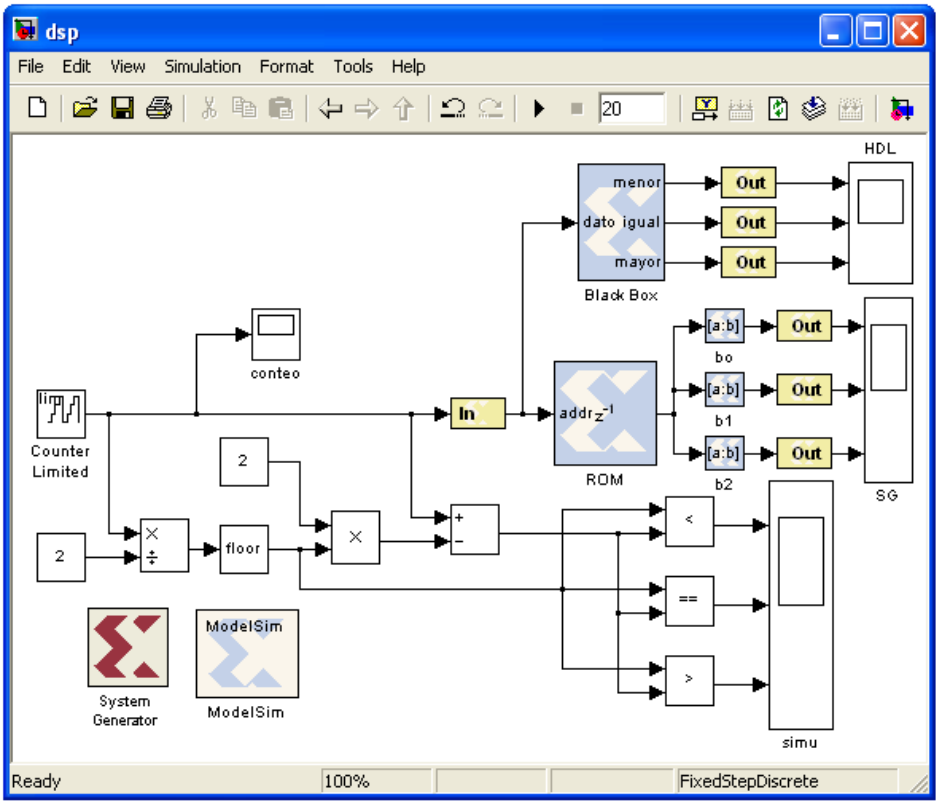


Fig. 5.5: Diagrama a bloques para codiseño Simulink-SG

de modelos usando una metodología basada en diagramas de bloques. Proporciona además bibliotecas de bloques para las tareas comunes de procesamiento en muy diversas áreas y, lo que es importante para el codiseño, crear nuevas bibliotecas e incorporar otras proporcionadas por terceras partes, en este caso las que suministra el fabricante de los dispositivos FPGA (por lo que se refiere a Xilinx, a través de System Generator) y las del fabricante de la placa donde está alojada la FPGA. La Fig. 5.5 muestra un ejemplo que incluye bloques de los tres tipos.

Para un codiseño basado en este esquema, en el diagrama del modelo existirán dos tipos de bloques: los propios de Simulink o *bloques SW* y los que corresponden al System Generator o *bloques HW*[4]. Entre los bloques HW se distinguen también los denominados como *black box*, siendo éstos los que quedan asociados con archivos HDL al momento de su configuración.

En la Fig. 5.6 se muestra el esquema que integra las herramientas de diseño involucradas. En el ciclo de la izquierda, el SG sirve como eslabón entre Simulink y ModelSim para llevar a cabo la co-simulación. En otras palabras, en el ordenador anfitrión, mediante la combinación Matlab/Simulink se realiza la simulación de los bloques SW del modelo y simultáneamente, de manera sincronizada, ModelSim lleva a cabo la simulación de los bloques HW.

En la parte derecha se observa el ciclo de integración HW-SW para la implementación y la operación en tiempo de ejecución (*real time* o *run time*). En este ciclo, queda enlazada, por una parte, la herramienta de implementación ISE de Xilinx con la finalidad de generar el archivo de configuración de la FPGA a partir de los archivos HDL y realizar la configuración.

[4]Estos se distinguen por llevar en el icono el logotipo de Xilinx.

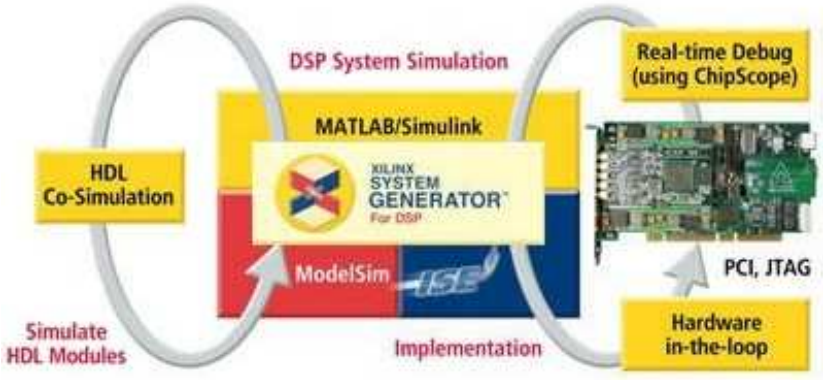


Fig. 5.6: Flujo de diseño del esquema Simulink-SG

Por la otra, además del Simulink, está la placa basada en la FPGA con sus dispositivos y controladores asociados. Como se observa en la figura, es posible realizar la operación paso por paso con propósitos de depuración. La comunicación con la placa en el ciclo de la derecha se realiza ya sea a través del bus PCI o del JTAG. Este es el medio por el cual se lleva a cabo la configuración de la FPGA y la transferencia se datos hacia y desde la placa.

Básicamente hay dos maneras para sincronizar la operación del hardware en la FPGA con la del software en Simulink. La primera de ellas está basada en la técnica *paso a paso* (single step) con la cual la operación del hardware está atada a la del software. Esto se logra aplicando un pulso de reloj al hardware por cada pulso de simulación en Simulink[5]. Con esta técnica es posible realizar las funciones de depuración y verificación. La desventaja de la operación paso a paso es la ralentización que lleva implícita para la operación del sistema.

La otra técnica de sincronización consiste en permitir que el hardware opere libremente bajo una frecuencia especificada y utilizar algún mecanismo explícito de sincronización tal como banderas implementadas por medio de registros con la finalidad de coordinar las transferencias de datos entre Simulink y el hardware. Con esta técnica las entradas y salidas del diseño se muestrean y escriben asíncronamente.

5.1.2 Estructura del sistema

El diseño del sistema tiene como objetivo contar con un banco de entrenamiento para la red SOM implementada en hardware con la flexibilidad que permiten los parámetros de entrenamiento y el aprovechamiento del pre y post procesamiento de los datos de entrada que aporta Matlab-Simulink, utilizado como entorno de operación. El sistema utiliza como coprocesador una placa basada en un dispositivo FPGA conectada a la PC anfitriona a través del bus PCI.

Aspectos generales de la placa

La placa empleada es la ADM-XRC-4 LX80 de Alpha-Data [104]. En la Fig. 5.7 se muestra su estructura.

[5]Para esto, la metodología Simulink/SG obliga a que todos los circuitos secuenciales que se describan con algún lenguaje HDL tengan un puerto `ce` (clock enable).

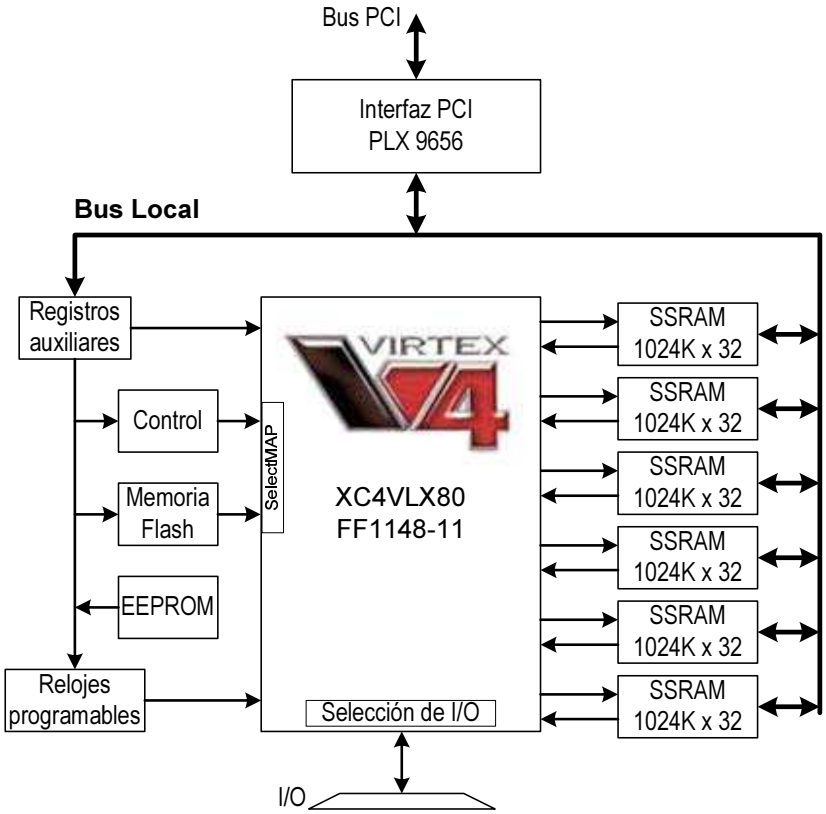


Fig. 5.7: Estructura de la placa ADM-XRC-4

El bus local es la interconexión entre el puente PCI y la FPGA, proporcionando una interface con buses de datos y de direcciones separados. La transferencia de datos entre el procesador y la FPGA utiliza un protocolo simple basado en transferencias por rachas con señales de control de inicio y fin. El bus de datos es de 32 bits con frecuencia de reloj de hasta 66 MHz. Las transferencias de datos las puede iniciar el puente PCI o la FPGA.

La lógica de control gestiona la configuración de la FPGA, da soporte al JTAG, supervisa la temperatura de operación y coteja la identificación de la placa. Está implementada en un dispositivo Spartan 3 y también se utiliza para acceder al estado en que se encuentra funcionado el sistema mediante la herramienta Chipscope.

Cada uno de los seis bancos de memoria ZBT (Non Bus Turnaround memory) tiene su propio dominio de reloj, de manera que los diseños se pueden optimizar fácilmente cuando existen fuentes múltiples de reloj.

La memoria EEPROM contiene datos de inicialización útiles para el puente PCI y también información referente a los recursos de la placa, tales como número de serie, capacidad de memoria y tipo de FPGA.

La placa tiene varias fuentes de reloj. El LCLK está dedicado al bus local y tiene una frecuencia de hasta 66 MHz. Se tiene también un oscilador con frecuencia programable para generar el reloj MCLK con frecuencia entre 20 y 500 MHz, éste es el reloj de la FPGA. Para establecer retardos precisos en las líneas de entrada/salida se tiene el reloj REFCLK a 200 MHz. Por último hay una señal de reloj para cada uno de los bancos de memoria con trayectorias de retroalimentación para compensación de retardos (de-skew).

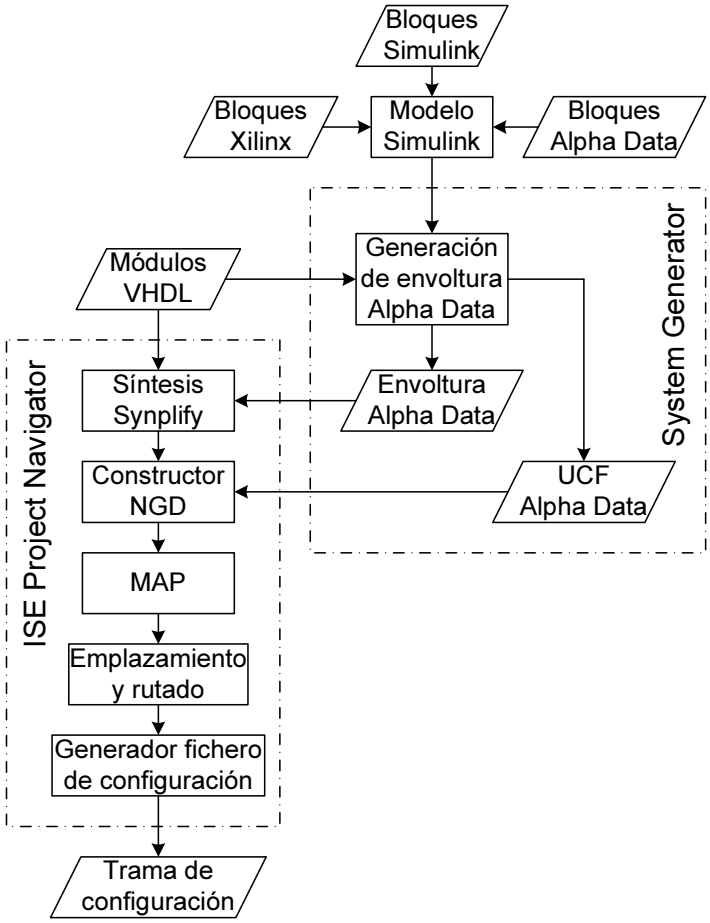


Fig. 5.8: Flujo de diseño Simulink-SG con Alpha Data

La configuración de la FPGA se puede realizar utilizando SelectMAP. También se puede configurar desde la memoria flash o por medio del JTAG. La manera más rápida es activando el modo SelectMAP de 8 bits, con lo cual la velocidad de configuración es de 66 MB/s.

El fabricante de la placa proporciona una biblioteca [105] de bloques para Simulink que permiten gestionar las tareas de comunicación y de transferencia entre el procesador y la tarjeta. Entre los bloques destaca el que hace las veces de interface con el puente PCI. Este bloque (PLX 32 bit interface) proporciona los buses de datos y de dirección, permite acceder al registro de estado y al registro de control, y gestiona las señales de protocolo para el manejo del bus local de la placa.

La Fig. 5.8 muestra el flujo de codiseño utilizando el esquema de Simulink-SG basado en la placa de Alpha Data.

El neurocoprocesador

El banco está integrado básicamente por dos componentes, siendo éstos el neurocoprocesador implementado en la placa y el modelo Simulink como entorno de entrenamiento. La Fig. 5.9 ilustra el esquema funcional del neurocoprocesador. Además de la FPGA donde se aloja la red SOM, se utilizan dos bancos de memoria RAM, el banco SRAM0 que será de sólo lectura

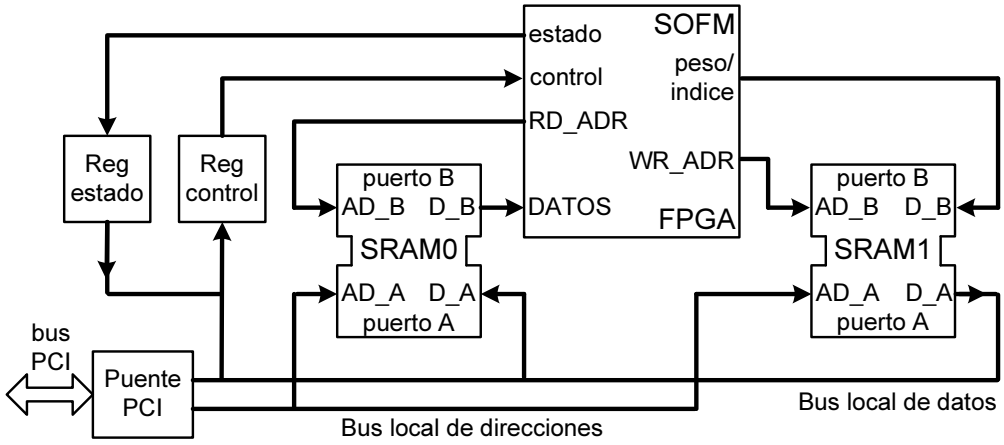
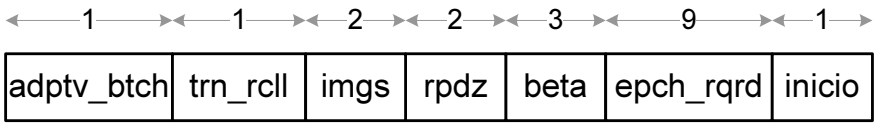


Fig. 5.9: Esquema funcional del neurocoprocesador

para la red y el SRAM1 de sólo escritura. Cada banco de memoria proporciona 1M x 32 bits de memoria RAM síncrona estática de doble puerto con una latencia de 3 ciclos de reloj. Las transferencias de datos entre el procesador anfitrión y los bancos de memoria se realizan a través de los buses locales de datos y direcciones que están conectadas a los puertos A de los bancos. El dispositivo FPGA está conectado directamente a los puertos B. Cuando Simulink inicia la operación del entrenamiento, escribe en el banco SRAM0 los valores iniciales de los pesos sinápticos y los vectores del conjunto de entrenamiento. Al finalizar la operación de la red, Simulink leerá del banco SRAM1 los pesos finales y los índices generados por el recall.

También se utilizan los registros de control y de estado. El registro de estado se emplea simplemente para que la red señalice que concluyó su operación. Por medio del registro de control Simulink establece los parámetros del entrenamiento, siendo éstos los que se muestran en la Fig. 5.10, donde se puede observar también su ancho, su ubicación en el registro y su descripción.



inicio => señal de inicio activa en bajo
epoch_rqrd => cantidad de epochs requeridos (1 a 512)
beta => valor inicial de la tasa de aprendizaje (1/2 a 1/128)
rpdz => caída de la tasa de aprendizaje (cada 4, 8, 16 o 32 epochs)
imgs => número de imágenes en el conjunto de entrenamiento (1 a 4)
trn_rcll => entrenamiento (0) o sólo recall (1)
adptv_btch => entrenamiento adaptivo (0) o batch (1)

Fig. 5.10: Formato de la palabra de control

En realidad el bit 0 del registro de control no es de configuración, sino el que se usa como señal de inicio. Cuando Simulink lo coloca momentáneamente en bajo se arranca la operación de sistema. Los parámetros *beta* y *rpdz* funcionan de acuerdo a lo establecido en §4.3.2[6].

[6]El parámetro *rpdz* está relacionado con el valor de δ de modo que para rpdz="00" se tiene $\delta = 4$, para

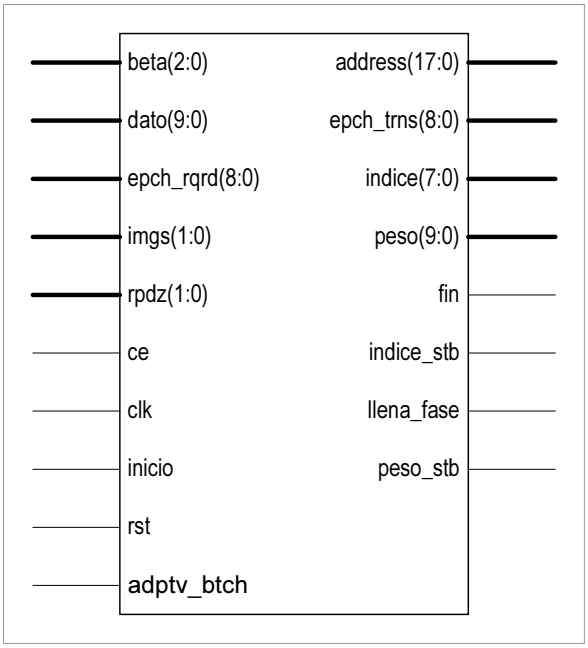


Fig. 5.11: Puertos de la red

La Fig. 5.11 muestra los puertos de la red implementada para datos de 10 bits y pesos con precisión de 18 bits[7]. En el lado izquierdo están las entradas, siendo éstas las correspondientes a los datos y a los parámetros, así como las señales de control y reloj. En el lado derecho como salidas está la de direccionamiento[8] para la memoria SRAM0, la de pesos y su señal de sincronía asociada (`peso_stb`), la de índices con su sincronía respectiva (`indice_stb`), la que indica la cantidad de epochs transcurridos y la que señala el fin del entrenamiento.

La señal de salida `llena_fase` se activa cuando la red se encuentra en el estado de operación en el que lee los valores de los pesos iniciales antes de comenzar el entrenamiento. Como consecuencia, esta señal se utiliza para controlar un multiplexor externo que selecciona datos o pesos iniciales.

La operación del neurocoprocesador está regida por dos relojes, el del bus local (`LCLK`) y el de la FPGA (`MCLK`). La frecuencia de operación para estos dos relojes se establece desde Simulink, teniéndose la única limitación de 66 MHz como máxima para `LCLK` y que la frecuencia de `MCLK` no sea menor que la de `LCLK`.

El modelo en Simulink como entorno de entrenamiento

En Simulink se deben generar dos modelos. Uno es el modelo de desarrollo y simulación y otro es el de ejecución (run-time model), la Fig. 5.12 muestra el modelo de simulación. Tomados de la biblioteca de Alpha Data se tienen tres bloques. El principal es el que se observa en naranja y proporciona un registro de control y un registro de estado para tareas

"01" $\delta = 8$ y así sucesivamente

[7]Cabe recordar que el diseño de la red es genérico en cuanto al tamaño de los datos de entrada y la precisión de los pesos. En este caso se implementó la red para datos de 10 bits y pesos de 18 bits previendo la aplicación que se presenta en la sección 5.2 de esta tesis.

[8]Con capacidad para gestionar conjuntos de entrenamiento con hasta cuatro imágenes de 256 x 256 pixeles

de control simples, una interface para transferencia asíncrona de datos mapeada en memoria y una interface para transferencia rápida de bloques de datos bajo un esquema DMA/FIFO; también suministra la interface ADLOCB que permite que el procesador del sistema acceda a los bancos de RAM. Con este esquema, se conectan en anillo sólo los bancos a los que sea necesario acceder desde el procesador. Los otros dos bloques (en azul) corresponden a los bancos de memoria empleados, enlazados por el ADLOCB por la parte del puente PCI y conectados de manera independiente por la parte de la FPGA.

El bloque en verde es la interface de cosimulación. Las entradas en el formato de Simulink se convierten al formato de SG y las salidas SG se convierten al formato Simulink. De esta manera es posible estimular las entradas y procesar las salidas, todo, empleando la gran variedad de bloques existentes en Simulink. En este caso se tiene como única entrada la señal de reset y como salidas la cantidad de epochs ejecutados y la señalización de final del entrenamiento.

La mayoría de los pequeños bloques de Xilinx tienen la finalidad de separar, extender y concatenar los datos. Se tienen dos bloques *Black Box* (BB), uno de ellos es el de la red SOM y el otro tiene el propósito de generar las direcciones para la memoria SRAM1 en la que se van guardando los pesos de salida de la red al final del entrenamiento y los índices producidos por el recall que se efectúa como paso final del proceso.

La Fig. 5.13 muestra el modelo de Simulink para la ejecución. Aparentemente es muy simple, pero en realidad contiene varias tareas que se realizan mediante funciones m incrustadas en los dos bloques coloreados en gris. Estos bloques son subsistemas que se procesan cuando ocurre una señal de disparo (trigger) en la terminal dedicada para tal fin. El subsistema de la izquierda (`llena_dispara`) recibe la palabra de control con los parámetros del entrenamiento y su función m asociada, cuando recibe el disparo producido por el bloque `step`, realiza las siguientes acciones: (i) inicializa la placa[9]; (ii) escribe la palabra de control en el registro respectivo; (iii) escribe en el banco SRAM0 los patrones de entrenamiento y los pesos sinápticos iniciales; y, (iv) activa el bit de inicio para que comience el procesamiento por parte de la red.

El subsistema de la derecha (`postprocesa`) al ser disparado por la señal `fin`, producida por la red al terminar el entrenamiento, efectúa la lectura del banco SRAM1 y coloca en el Workspace de Matlab los pesos generados por el entrenamiento y los índices producidos por el recall. Con los pesos y los índices como variables de Matlab, es posible efectuar las tareas de postprocesamiento que se requieran con el auxilio de toda la potencialidad que brinda Matlab.

Las funciones m incrustadas en `llena_dispara` y `postprocesa` se escribieron específicamente para el desarrollo del banco de entrenamiento. Como parte de estas funciones, para las tareas que tienen que ver con la placa (inicialización, lectura y escritura en bancos de memoria y escritura en el registro de control) se realizan llamadas a funciones proporcionadas por su fabricante [106].

El bloque verde es la interface de ejecución asociado al de simulación que se encuentra en la Fig. 5.12. Dentro de la ventana de configuración de este bloque se establece el archivo de configuración de la FPGA y el modo de operación, ya sea paso a paso o con frecuencia libre de acuerdo a lo mencionado en §5.1.1, también se fija la frecuencia para la señal de reloj `MCLK`. En el extremo izquierdo del diagrama, como constantes, se establecen los valores para los parámetros de entrenamiento.

[9]Dentro de la inicialización se fija la frecuencia de la señal de reloj LCLK

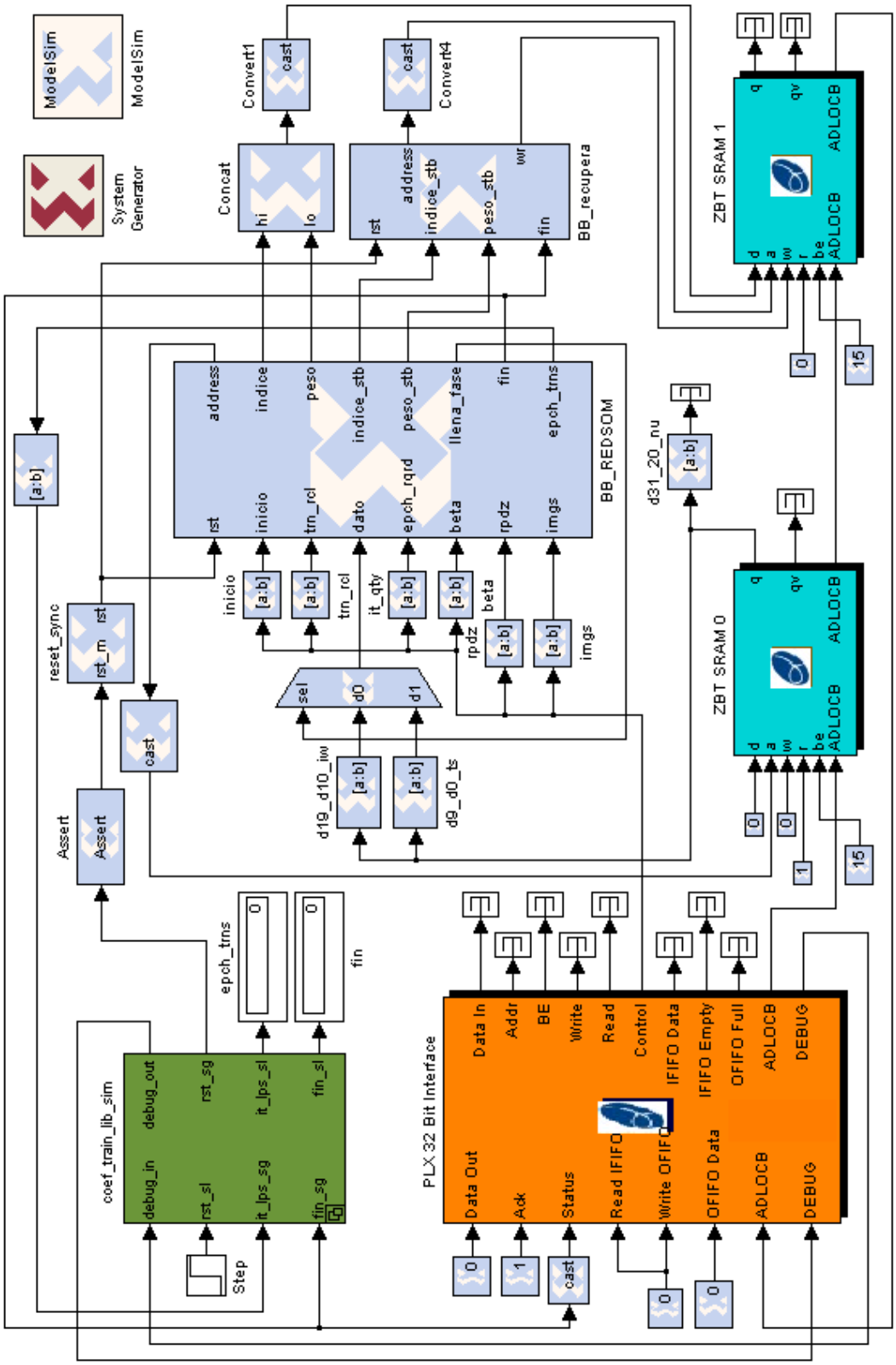


Fig. 5.12: Modelo de desarrollo y cosimulación

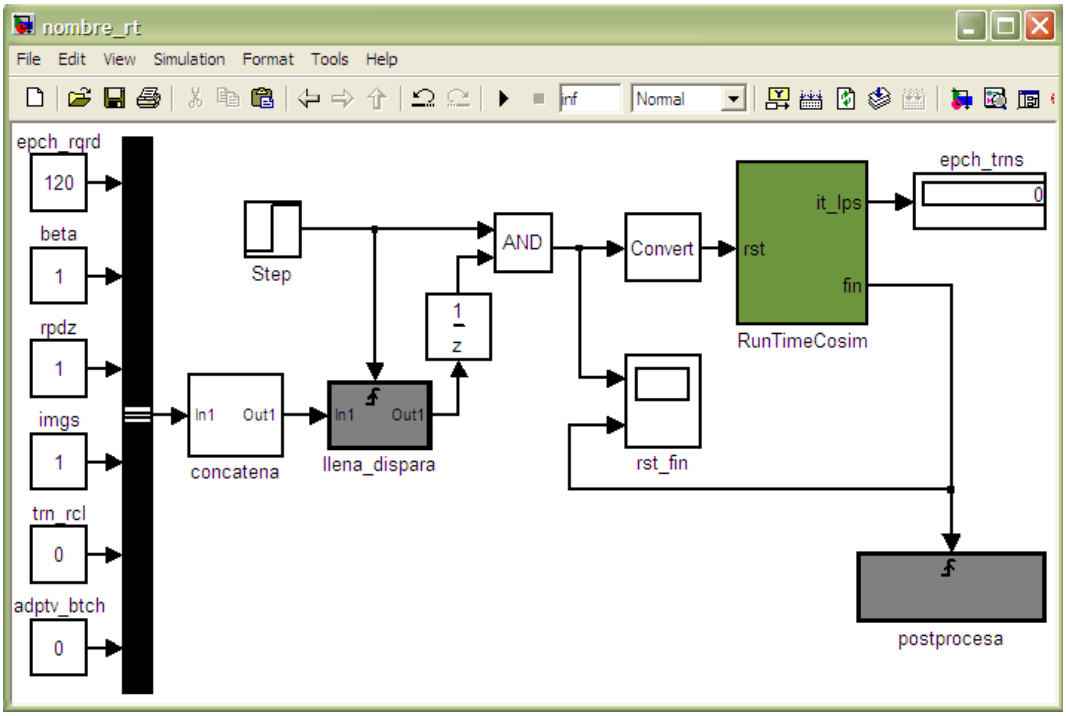


Fig. 5.13: Modelo Simulink para la ejecución del entrenamiento

5.1.3 Aplicaciones de investigación para prueba del sistema

En este apartado se presentan dos aplicaciones de investigación para el banco de entrenamiento desarrollado, con algunas variantes para adecuarlo a los requerimientos específicos de cada una de ellas.

Acelerador del algoritmo LBG con una implementación HW/SW

En esta aplicación se utiliza la red neuronal en su forma reducida para trabajar únicamente en la fase de recall. El objetivo es llevar a cabo el procesamiento del algoritmo LBG de manera acelerada al implementarlo de manera hibrida, hardware-software.

El algoritmo LBG se describió en el Capítulo 2 de esta tesis. Parte de un diccionario formado solamente por un vector obtenido como el promedio de todos los vectores que forman el conjunto de entrenamiento. Después se sigue un proceso que consta de dos fases. En la primera se lleva a cabo la bipartición del diccionario para duplicar así el número de sus vectores. La segunda fase se realiza de manera iterativa y tiene las siguientes dos etapas: (i) la clusterización de los vectores del conjunto de entrenamiento en tantos clusters como vectores tiene el diccionario hasta el momento[10]; (ii) la obtención del nuevo diccionario, o sea el cálculo del centroide para cada cluster que será su nuevo vector representativo. Este proceso de dos fases se repite hasta alcanzar la cantidad requerida de vectores en el diccionario.

El algoritmo LBG es conceptualmente muy simple y fácil de implementar. Sin embargo, su eficiencia computacional es baja debido a que requiere comparar de manera exhaustiva

[10]Cada vector de entrada se asigna al cluster cuyo vector representativo le sea más cercano; el vector representativo de un cluster es el vector del diccionario al que se encuentra asociado de manera biunívoca

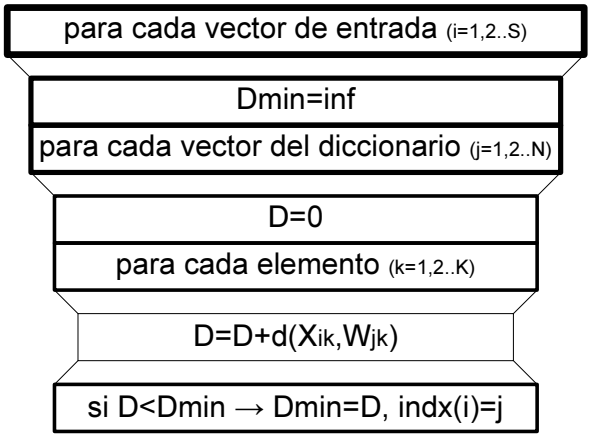


Fig. 5.14: Clusterización software

cada vector del conjunto de entrenamiento con cada vector del diccionario. Por ello, se han desarrollado varios trabajos con el propósito de reducir el tiempo requerido para su procesamiento [107][108][109].

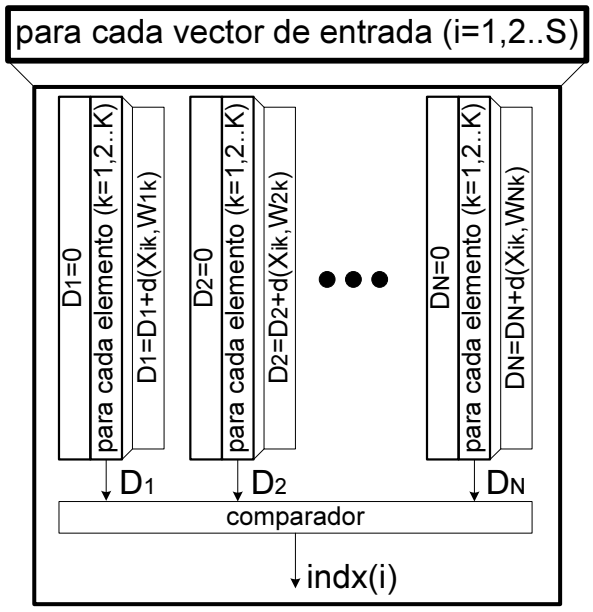


Fig. 5.15: Clusterización hardware

La clusterización de la manera descrita es en realidad la cuantización vectorial de los vectores del conjunto de entrenamiento usando como diccionario el que se tenga hasta el momento en que se realice. Por lo tanto, su procesamiento se puede efectuar utilizando la red SOM operando en la fase de recall. La Fig. 5.14 muestra el procesamiento software de la etapa de clusterización y en la Fig. 5.15 se observa el procesamiento hardware mediante la red SOM.

Comparando estas dos alternativas de procesamiento, la complejidad para el caso software es de $S * N * K$, mientras que para el hardware es únicamente $S * K$, esto debido a que el

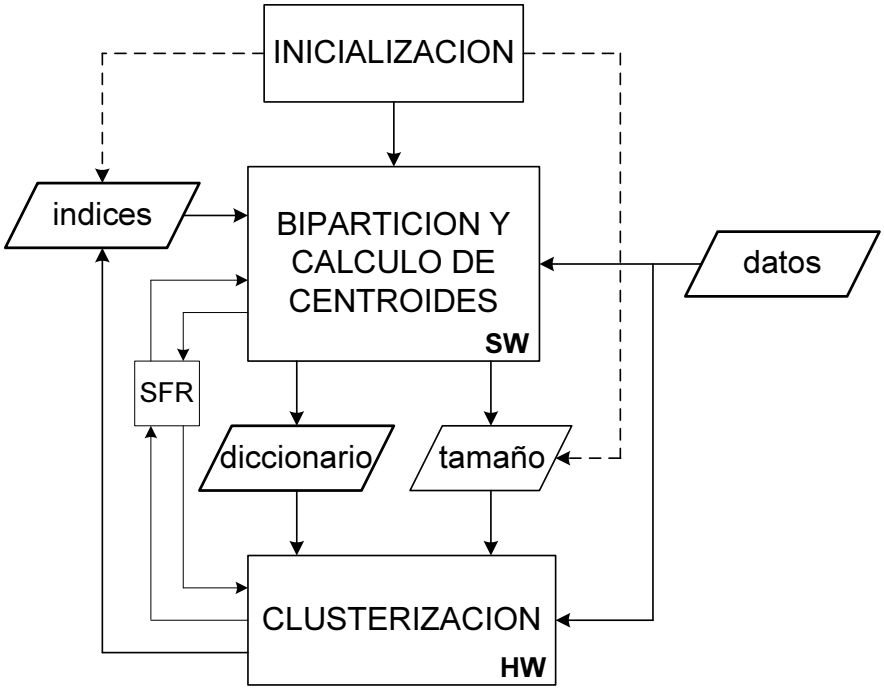


Fig. 5.16: Diagrama a bloques de la implementación híbrida HW-SW

procesamiento software es puramente secuencial, mientras que en el hardware los N vectores del diccionario se procesan en paralelo. Por lo tanto, suponiendo que los procesadores operaran a la misma frecuencia, el ahorro en tiempo si se utiliza la alternativa hardware es de $1/N$ respecto a la alternativa software.

La Fig. 5.16 muestra el esquema utilizado en el cual se tiene una implementación hibrida HW-SW. En el bloque SW se realizan las tareas de bipartición y cálculo de los centroides, mientras que en el HW se efectúa la clusterización. La integración de los dos bloques se lleva a cabo con base en dos memorias. En una de ellas el bloque SW escribe el diccionario que genera mediante el cálculo de los centroides y, en su caso[11], por la bipartición. En la otra el HW escribe los índices que resultan de la cuantización vectorial o clusterización. El pequeño bloque `SFR` es un semáforo a través del cual se coordina la operación del HW y el SW. El bloque SW también lleva el control de iteraciones y tamaño del diccionario. Los datos del conjunto de entrenamiento se encuentran en otra memoria que es leída tanto por el SW como por el HW.

La Fig. 5.17 es el modelo del codiseño Simulink-SG para esta aplicación. En este modelo se utilizan tres bancos de memoria SRAM, cada uno con la finalidad mencionada en el párrafo anterior.

Para la experimentación se utilizaron cuatro imágenes de 256 x 256 pixeles para producir el conjunto de vectores de entrenamiento, formando cada vector con un bloque de 4 x 4 pixeles. De esta manera el conjunto se compone de 16384 vectores de tamaño 16. Se utilizó este conjunto para generar un diccionario con 256 vectores. Se realizaron 100 iteraciones para cada tamaño parcial del diccionario. Con la finalidad de evaluar la aceleración del algoritmo

[11]La bipartición sólo ocurre hasta que se completan las iteraciones requeridas, mientras que el cálculo de los centroides se produce en cada iteración

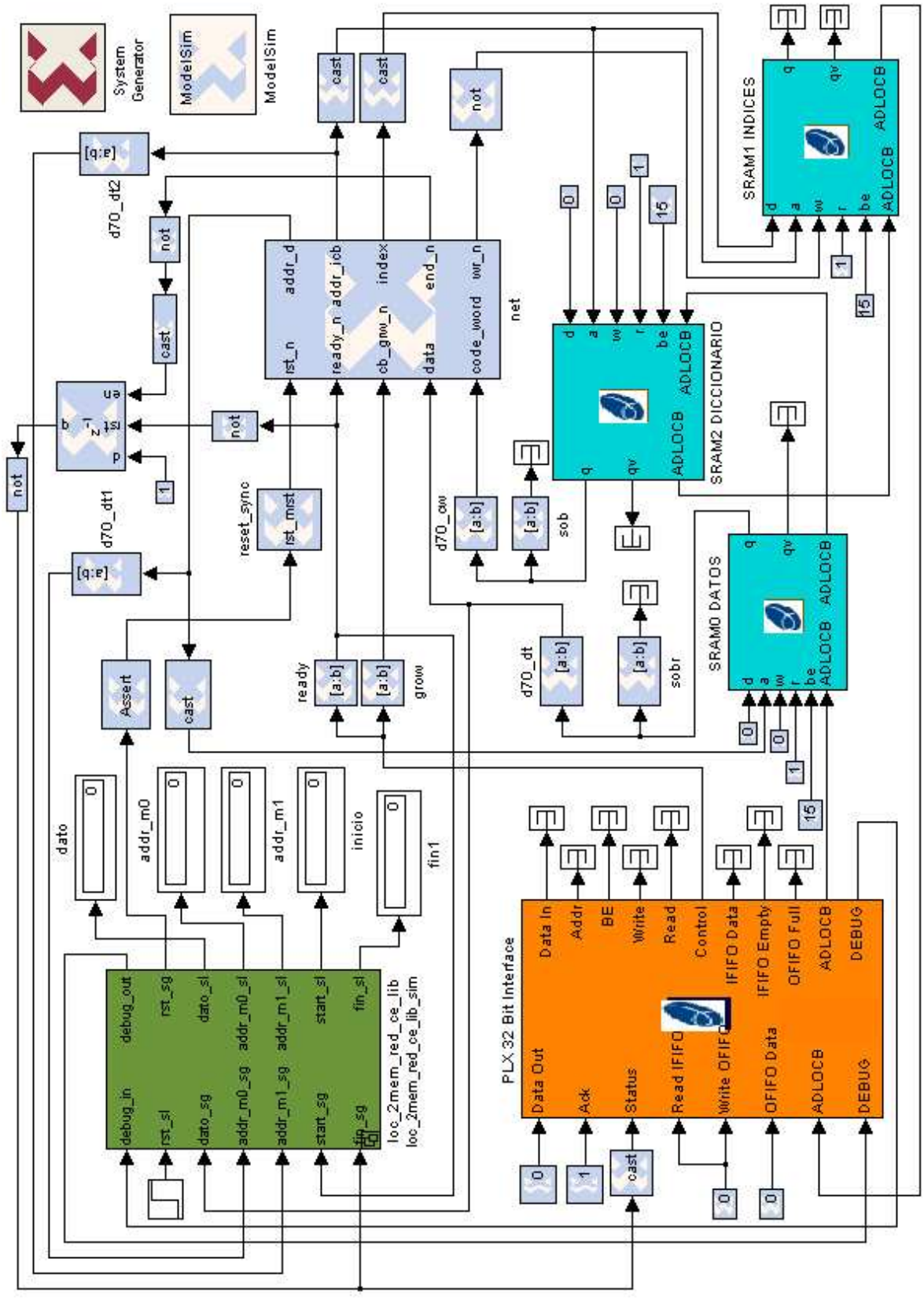


Fig. 5.17: Codiseño Simulink-SG para el acelerador del LBG

con la implementación HW-SW, también se generó el diccionario con una implementación puramente software en Matlab. El procesador utilizado fue un Pentium IV operando a 3 GHz. En la Tabla 5.2 se muestran los resultados.

Implementación	SW (Euclidiana)	SW (Manhattan)	Híbrida HW-SW
Tiempo (sgs)	2506	1925	249
PSNR Lena	27,91	27,49	27,49
PSNR Goldhill	27,01	27,20	27,22
PSNR Bridge	25,00	24,75	24,87
PSNR Camera	26,97	26,41	26,37

Tabla 5.2: Evaluación del acelerador

Efectos en el diccionario del esparcimiento de los pesos iniciales en una red SOM

Cuando se entrena una red SOM, el rendimiento final depende significativamente de la inicialización de los vectores de pesos. Por lo tanto, es aconsejable generar cierta cantidad de diccionarios SOM, cada uno obtenido a partir de un conjunto de pesos generados aleatoriamente. Posteriormente estos diccionarios serían analizados y se seleccionaría el que diera lugar a una mejor distribución de los vectores de entrenamiento. Esta es una tarea lenta y pesada, hasta ahora no automatizada. Para conjuntos de entrenamiento grandes no es práctica y, por consiguiente, lo que se hace comúnmente es generar un sólo diccionario y aceptarlo como el resultado final.

En esta aplicación del banco de entrenamiento, aprovechando la rapidez con la que se generan los diccionarios, se analiza experimentalmente cual es el efecto que tiene en el mapa la dispersión de los valores iniciales de los pesos generados aleatoriamente. El banco de entrenamiento se configuró para ejecutar una serie de 10 sesiones de entrenamiento para la misma red. En cada sesión se utilizó un conjunto de vectores iniciales diferente. Los pesos se generaron de manera aleatoria con valores dentro de una ventana cuyo centro es el centroide de todos los vectores del conjunto de entrenamiento y con un ancho creciente de una sesión a la siguiente. Para la primera sesión de entrenamiento el ancho de la ventana es igual a 9, para la segunda es igual a 17, y así continúa creciendo hasta llegar a 81 para la décima sesión.

Se utilizó una SOM con 256 neuronas y 16 unidades de entrada. Las cuatro imágenes Lena, Goldhill, Bridge y Camera se descompusieron en bloques de 4 x 4 pixeles para así formar el conjunto de entrenamiento con 16384 vectores. Se empleó el entrenamiento batch y para cada sesión se realizaron 100 epochs.

Al final de cada sesión se utilizaron los índices generados para obtener la cantidad de veces que las neuronas resultaron ganadoras en el último epoch y finalmente se obtuvo la desviación estándar de esos valores. También se obtuvo para cada sesión la suma de las distorsiones producidas por la cuantización vectorial de las cuatro imágenes, empleándose el PSNR como métrica.

La Fig 5.18 muestra las gráficas de estos resultados. Arriba está la gráfica de desviación estándar y abajo la de distorsión, teniendo como variable en el eje horizontal el ancho de la ventana donde se ubican los pesos iniciales para cada sesión. En otras palabras, las gráficas

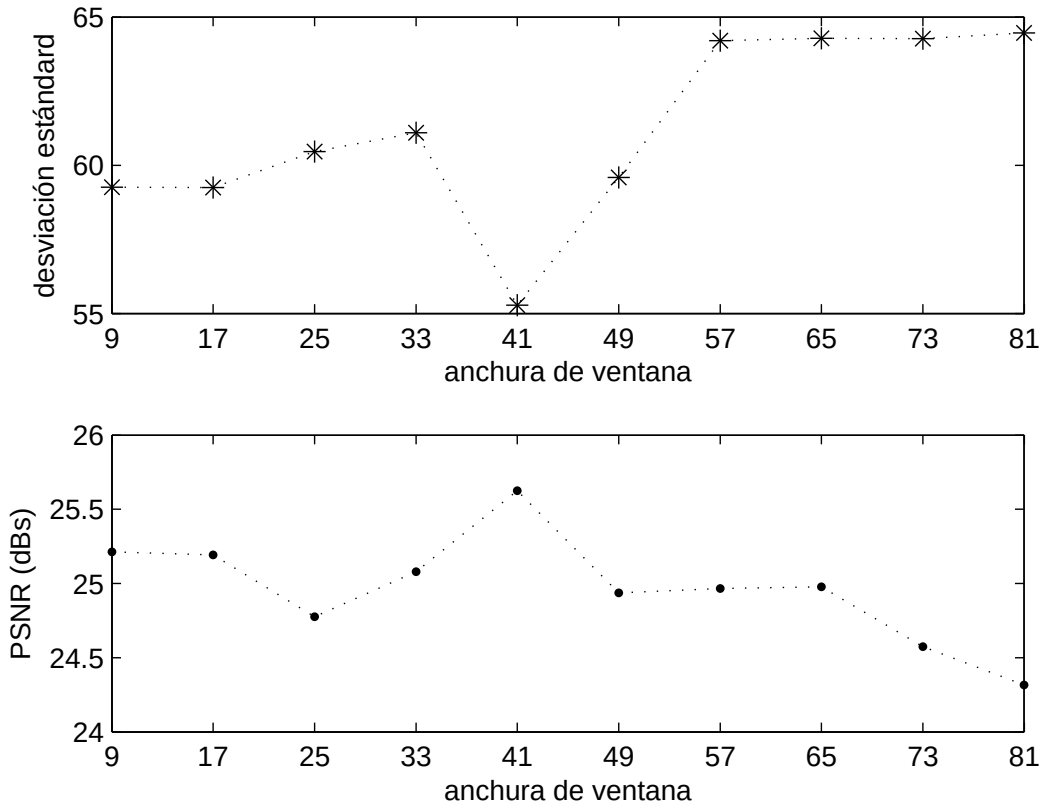


Fig. 5.18: Distribución de vectores y distorsión en función de la dispersión de pesos iniciales

muestran la desviación estándar y la distorsión como funciones de la dispersión de los pesos iniciales.

Los resultados en ambas gráficas son consistentes. El mejor diccionario se generó cuando el ancho de la ventana tuvo un valor de 41. Con esta dispersión de los pesos iniciales la red SOM mapea los vectores de entrenamiento con una distribución más uniforme entre los vectores del diccionario y, a la vez, minimiza el valor de la distorsión.

Usando el banco de entrenamiento operando con una frecuencia de 50 MHz, el tiempo requerido para completar las diez sesiones de entrenamiento fue de 62 segundos. Cuando se ejecutó la misma serie usando el SOM Toolbox [110], ejecutándose en un procesador Pentium 4 a 3 GHz el proceso tomó 572 segundos. En otras palabras, el banco de entrenamiento HW-SW reduce el tiempo de entrenamiento en 89% con respecto a la implementación puramente software en Matlab.

5.2 Sistema 2D-DWT/VQ para compresión de imágenes

Como se expuso en el Capítulo 2, la compresión de imágenes consiste en una secuencia de operaciones, comúnmente: transformación, cuantización y codificación. La 2D-DWT se considera

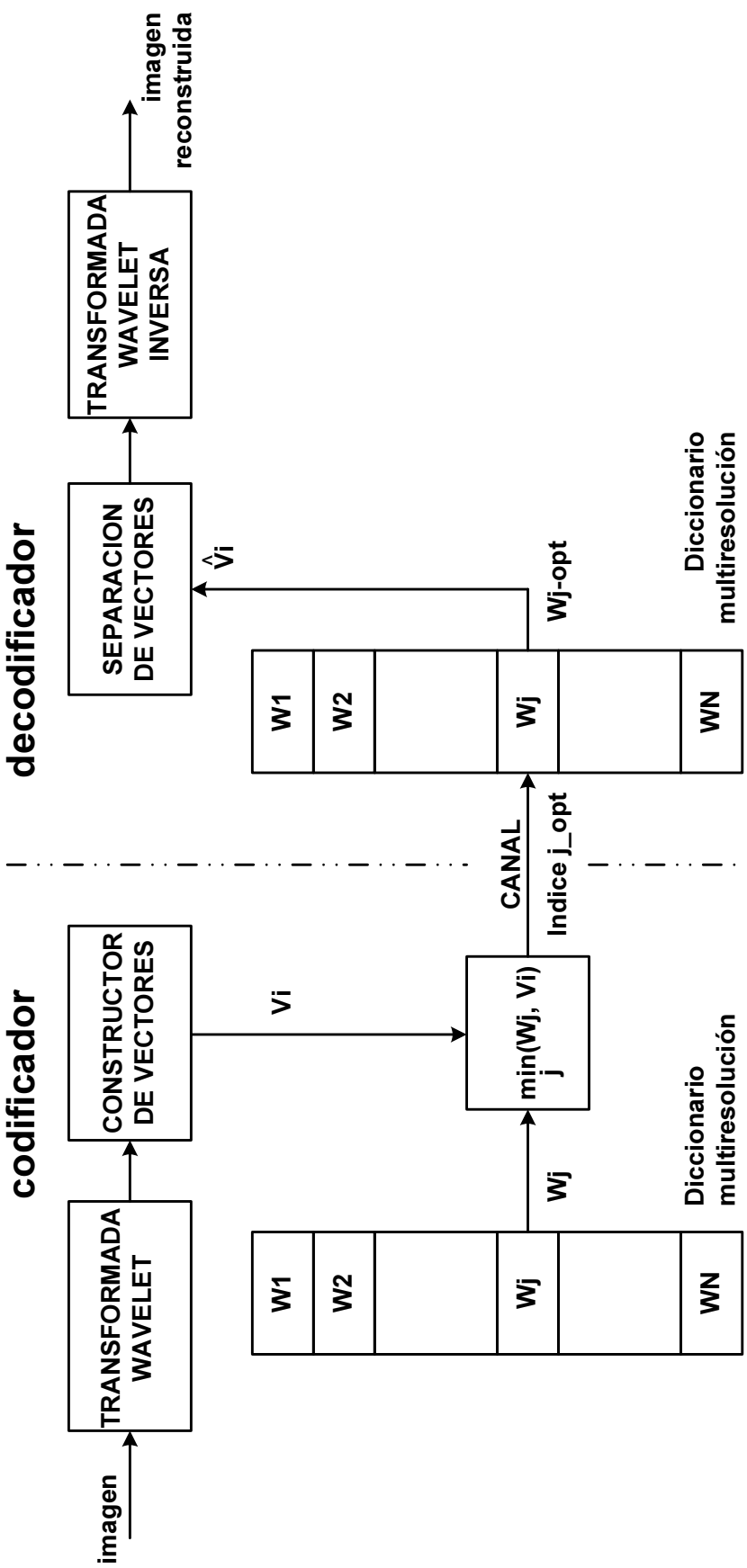


Fig. 5.19: VQ multiresolución

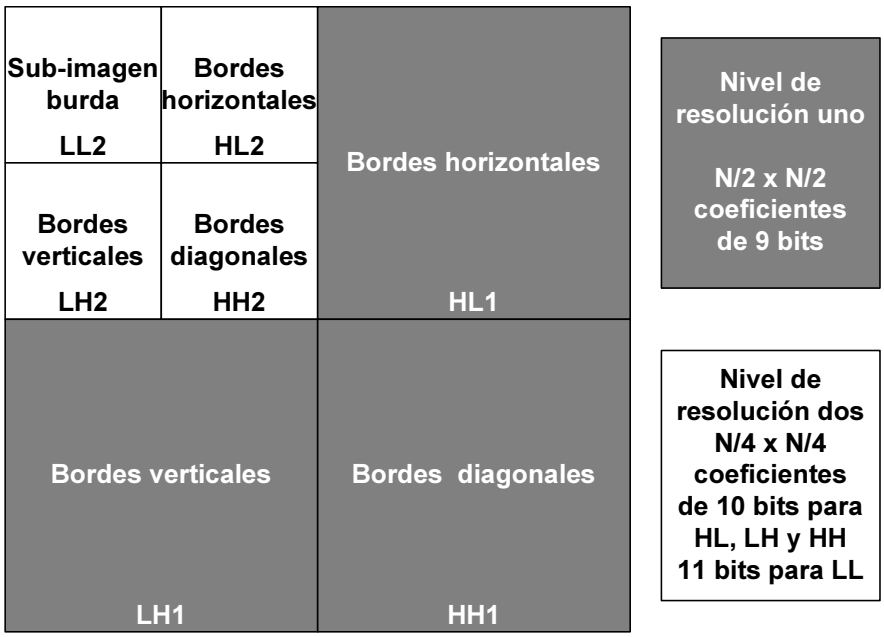


Fig. 5.20: Descomposición wavelet aplicada

una transformación muy adecuada para esta aplicación ya que una compresión eficiente requiere que los datos transformados contengan información espacial y frecuencial de la imagen original. El empleo de la 2D-DWT como técnica de preprocesamiento para la cuantización vectorial mejora la eficiencia de la compresión en dos sentidos, primero, extrae rasgos de resolución y orientación de los datos de la imagen y, segundo, reduce los efectos de bloque que introduce la VQ.

La combinación 2D-DWT/VQ para compresión de imágenes fue estudiada inicialmente por Antonini et al en [16] generando como uno de sus resultados el esquema de codificación/decodificación que se muestra en la Fig. 5.19. Este esquema se basa en un conjunto de diccionarios a los que ellos denominaron *diccionarios multiresolución*. Con diccionarios multiresolución, la VQ se puede considerar como una clase especial de *cuantización vectorial clasificada* (CVQ). En [16] también se probó que la VQ se comporta mejor que la cuantización vectorial cuando se aplica sobre los coeficientes wavelet.

Como parte de la investigación en la que se sustenta esta tesis, se integró un sistema de compresión de imágenes que incluye las etapas de transformación y cuantización basado en el esquema de la Fig. 5.19 empleando la red neuronal expuesta en el Capítulo 4 para la cuantización vectorial multiresolución.

5.2.1 Generalidades del esquema desarrollado

La cuantización vectorial multiresolución comienza por aplicar a la imagen la transformada wavelet para obtener separadamente, en varios niveles de resolución, los coeficientes de las distintas sub-bandas, cada una de las cuales proporciona información direccional de la imagen. En este desarrollo se utilizaron dos niveles para la descomposición wavelet, con lo cual se obtienen las 7 sub-bandas que se ilustran por medio de la Fig.5.20.

Después de la etapa 2D-DWT, las matrices de coeficientes obtenidas tienen conjuntamente

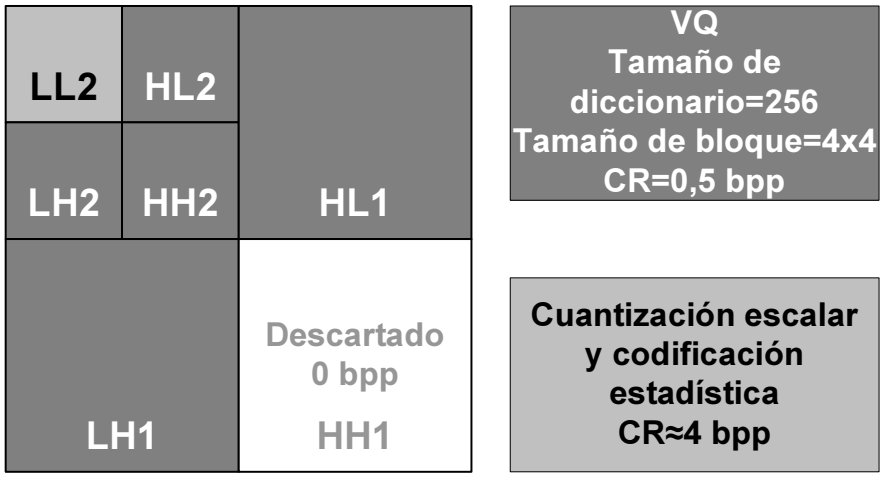


Fig. 5.21: Cuantización de las sub-bandas

la misma cantidad de elementos que la matriz original de pixeles. Sin embargo, la cantidad de bits requeridos para los coeficientes wavelet creció cuando la 2D-DWT se aplicó para la transformación de la imagen original al primer nivel de resolución, y nuevamente cuando se realizó la transformación del primer nivel al segundo. Por lo tanto, los datos han sufrido una expansión. La compresión en realidad se logra por medio de la cuantización y codificación de los coeficientes wavelet en las etapas posteriores.

En la Fig. 5.21 se muestra el proceso aplicado a cada una de las sub-bandas para obtener la compresión de la imagen. Debido a que es muy poca la información que contiene, la sub-imagen diagonal del nivel de resolución uno se descarta. Las restantes sub-bandas de alta frecuencia en los dos niveles de resolución se cuantizan formando bloques de 4 x 4 coeficientes y usando diccionarios de tamaño 256 con vectores de dimensión 16, de manera que se obtiene una razón de compresión de 0,5 bpp. La sub-imagen burda del nivel de resolución dos contiene la mayor parte de la información y, por consiguiente, se debe codificar con la menor cantidad de pérdidas posible. Así, para esta sub-banda se utilizó un esquema que combina cuantización escalar y codificación estadística para producir una razón de compresión alrededor de 4 bpp.

La Fig. 5.22 bosqueja el sistema desarrollado. La sincronización de las operaciones y el manejo del flujo de datos entre las etapas hacen necesario el uso de bloques de memoria para el almacenamiento temporal de los datos de entrada y de los coeficientes generados en cada nivel de resolución. Los cuadros en gris muestran los bloques de memoria requeridos para este propósito y su respectivo tamaño. Los cuadros en negro corresponden a los bloques para el almacenamiento de los cinco diccionarios propios del esquema multiresolución, con las consideraciones expuestas en el párrafo anterior.

Los pixeles de la imagen a codificar se almacenan en una memoria y de allí pasan al bloque 2D-DWT que lleva a cabo la transformación wavelet, primero la del primer nivel de resolución y sólo con un pequeño retraso la del segundo, entregando en ese mismo orden los coeficientes respectivos. Por cada coeficiente generado este bloque envía también una señal de sincronización.

Se tienen dos bloques VQ, uno se utiliza para cuantizar las tres sub-bandas del nivel dos (`VQ2`)y el otro se usa para las sub-imágenes horizontal y vertical del nivel uno (`VQ1`). Los multiplexores se utilizan con la finalidad de reutilizar los bloques cuantizadores. La

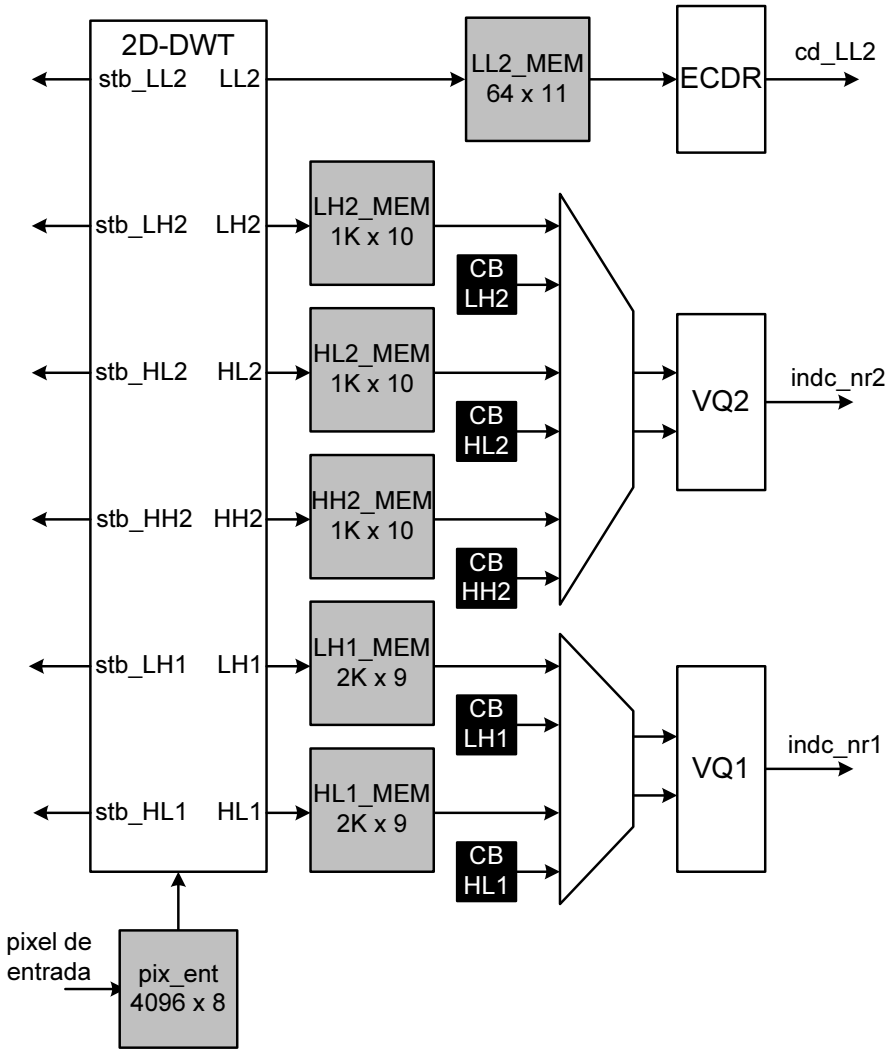


Fig. 5.22: Diagrama a bloques del sistema

reutilización es posible debido a que, siendo éste el caso crítico, el tiempo necesario para cuantizar vectorialmente una franja compuesta por cuatro renglones de coeficientes del nivel de resolución dos es menor que un tercio del tiempo que requiere el bloque 2D-DWT para generarla.

Para los coeficientes de la sub-banda LL2, el bloque `ECDR` realiza su cuantización escalar y clasifica las diferencias entre coeficientes consecutivos en cuatro grupos de acuerdo con su magnitud. Los coeficientes, una vez cuantizados, clasificados y codificados, se transmiten en serie. Para lograr la sincronía entre la generación de coeficientes y su codificación, se tiene una memoria que almacena una fila.

A continuación se expondrán las tres etapas que integran el compresor, se presentan su estructura y resultados de su implementación.

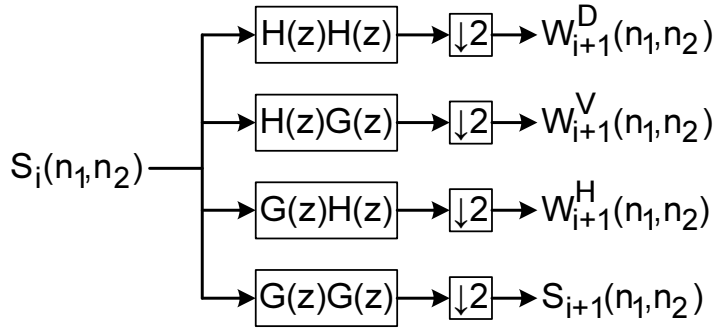


Fig. 5.23: Diagrama a bloques del banco de filtros no separables para la 2D-DWT

5.2.2 La etapa 2D-DWT.

La transformada 2D-DWT opera sobre señales bidimensionales, como es el caso de las imágenes. Mientras que los filtros unidimensionales (1D) se utilizan para el cálculo de la transformada 1D-DWT, los filtros bidimensionales (2D) se usan para la 2D-DWT. Los filtros 2D pueden ser separables o no separables. Un filtro 2D, $f(x, y)$, es separable si se puede escribir como $f(n_1, n_2) = f_1(n_1)\, f_2(n_2)$, donde f_1 y f_2 son filtros 1D.

Con filtros 2D no separables es posible descomponer una imagen en sus cuatro subimágenes sin necesidad de aplicar la DWT de manera independiente primero sobre renglones y después sobre columnas o viceversa. La Fig. 5.23 ilustra la arquitectura de la 2D-DWT mediante filtros 2D no separables para un nivel de resolución. Cada rama de esta arquitectura consta de un filtro 2D y un diezmador por 2. La entrada $S_i(n_1, n_2)$ es para los coeficientes de aproximación del nivel de resolución i y como salidas se obtienen los coeficientes de aproximación y los coeficientes de detalle en cada dirección del nivel $i + 1$.

Estructura de los filtros

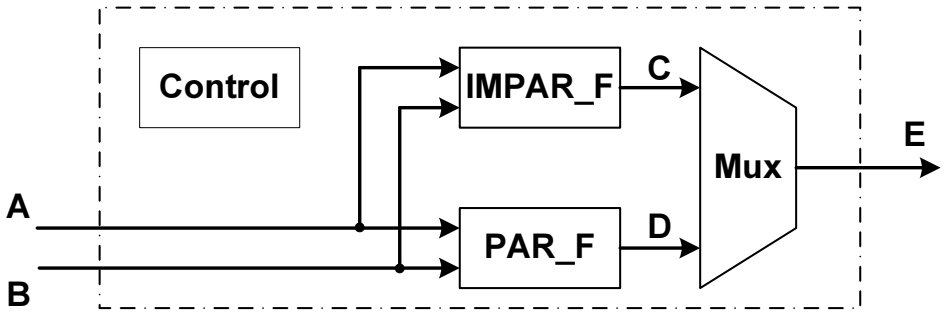


Fig. 5.24: Estructura par-impar de un filtro no separable

La 2D-DWT utilizada emplea los módulos de filtros 2D y el método desarrollados en [111]. La Fig. 5.24 es el diagrama a bloques de un filtro 2D no separable con una estructura par-impar. Está compuesto principalmente de dos unidades de filtrado, un multiplexor y un controlador simple. El filtro IMPAR_F calcula los coeficientes wavelet impares (salida C) y el filtro PAR_F hace lo mismo para los coeficientes pares (salida D). Por las entradas A y B llegan, respectivamente, los pixeles impares y pares de la imagen de entrada. Con esta

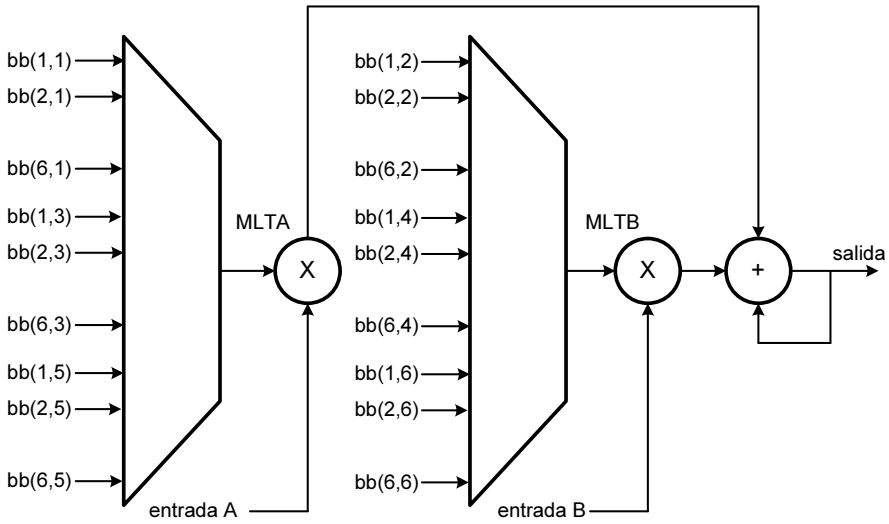


Fig. 5.25: Organización del filtro para procesar dos pixeles en paralelo

estructura se procesan dos pixeles en paralelo.

Esta estructura divide entre cuatro la cantidad de ciclos de reloj requeridos para procesar una imagen; se divide entre dos al procesar a la vez dos pixeles y nuevamente entre dos porque se generan simultáneamente dos coeficientes, uno par y uno impar. Para el caso de una matriz de coeficientes del filtro con tamaño 6 x 6, en la Fig. 5.25 se observa la organización del filtro para procesar dos pixeles en paralelo, fundamentada en [112]. Se tienen los dos multiplicadores MLTA y MLTB. MLTA multiplica los coeficientes impares del filtro por los pixeles impares y MLTB multiplica los pares. Las entradas $bb(i,j); i,j \in [1,6]$ son los elementos de la matriz de coeficientes del filtro, considerándose pares los correspondientes a valores pares de j.

Al iniciar el cálculo de una fila de coeficientes, el filtro par permanece ocioso mientras el filtro impar procesa el primero. Posteriormente los dos filtros operan a la vez, es decir, mientras el filtro par calcula el coeficiente 2, el filtro impar calcula el 3, y así continúan progresivamente. Al llegar el momento de calcular el último coeficiente de la fila ocurrirá lo contrario, el filtro impar estará ocioso mientras el par calcula el último coeficiente. Por lo tanto, la cantidad de ciclos de reloj necesarios para calcular una fila de coeficientes será[12]:

$$F(M,L) = \frac{L^2}{4}\left(\frac{M}{2}+1\right) = \frac{L^2M}{8} + \frac{L^2}{4}. \tag{5.1}$$

De este modo, la cantidad de ciclos necesarios para calcular el total de coeficientes del primer nivel de resolución será:

$$T(M,L) = \frac{M}{2}F(M,L) = \frac{L^2M^2}{16} + \frac{L^2M}{8}. \tag{5.2}$$

El factor $L^2/4$ en la Ec. 5.1 es la cantidad de ciclos necesarios para calcular un coeficiente, siendo L x L el tamaño de la matriz de coeficientes del filtro. La cantidad de multiplicaciones para calcular un coeficiente es L^2, sólo que por la estructura del filtro par-impar se procesan en paralelo 2 pixeles y se generan dos coeficientes a la vez, por eso se tiene la división entre 4.

[12]Si la imagen tiene M x M pixeles, el primer nivel de resolución constará de M/2 x M/2 coeficientes.

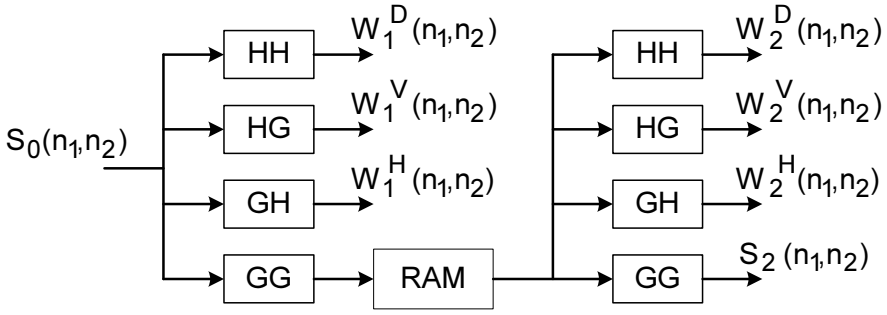


Fig. 5.26: 2D-DWT de dos niveles con filtros no separables

Usando una implementación directa, la 2D-DWT con dos niveles de resolución se diseñó como se muestra en la Fig. 5.26. Cada uno de los bloques de esta arquitectura (HH, HG, GH y GG) está construido utilizando para los filtros 2D la estructura par-impar explicada en los párrafos anteriores. Esos bloques incluyen, además del filtro, el diezmador por dos. Se utilizó una memoria RAM de doble puerto para interconectar los dos niveles.

Tipo de filtro empleado

Se utilizaron los filtros biortogonales CDF 2/2 cuyos coeficientes quedan determinados por las ecuaciones siguientes:

$$\begin{aligned} G_S(z) &= \tfrac{\sqrt{2}}{4}(z^{-1}+2+z), \\ G(z) &= \tfrac{\sqrt{2}}{8}(-z^{-2}+2z^{-1}+6+2z-z^2). \end{aligned} \tag{5.3}$$

Para la descomposición wavelet un filtro CDF 2/2 está formado por un conjunto de cinco coeficientes para el filtro pasa-baja y tres para el pasa-alta. Los coeficientes se arreglan como una matriz de 6 x 6 (L=6); de este modo, el procesamiento de una imagen de 512 x 512 pixeles en el primer nivel de resolución tomará 592128 ciclos.

Resultados de la implementación de la 2D-DWT.

Para la implementación se utilizó la Virtex 4 LX80-11. La Tabla 5.3 muestra los resultados en cuanto a ocupación de recursos. Para el almacenamiento de coeficientes necesario para enlazar el primer nivel de resolución con el segundo se utilizaron bloques de memoria, siendo éstos los que se reportan en la tabla.

Recurso	Utilizado	Disponible	Utilización
Flip-flops	1080	71680	1%
LUTs	1498	71680	2%
Slices	1018	35840	2%
Bloques de RAM	6	200	3%

Tabla 5.3: Ocupación de recursos por la 2D-DWT en la FPGA

La máxima frecuencia obtenida después del emplazamiento y ruteado fue de 62 MHz. Operando a 50 MHz el diseño toma 11,84 ms para procesar una imagen de 512 x 512 pixeles.

5.2.3 Etapa de cuantización vectorial

Para la cuantización vectorial se utilizó la red SOM desarrollada en el Capítulo 4, tanto para la generación de los cinco diccionarios como para la codificación, esto último en el modo de recall. La SOM tiene 16 unidades de entrada y 256 neuronas de salida con la finalidad de generar diccionarios de tamaño 256 para vectores de dimensión 16. En realidad se tienen dos redes que utilizan varias matrices de pesos. Una red realiza la cuantización vectorial de los coeficientes de las tres sub-bandas de frecuencias altas del nivel de resolución dos, por lo cual se tienen tres diccionarios que se van aplicando de manera alternada de acuerdo a la sub-banda específica que se deba procesar. La otra red es para la cuantización de las sub-bandas de alta frecuencia horizontal y vertical del nivel de resolución uno, por tanto, con dos diccionarios.

La red desarrollada en el Capítulo 4 se adaptó para el tipo de datos producidos por la etapa previa para los valores de los coeficientes wavelet. Los datos tienen un tamaño que varía de acuerdo a la sub-banda y al nivel de resolución de que se trate. El mayor tamaño es de 10 bits, como se observa en la Fig. 5.22. Por otra parte, estos datos son números con signo.

Como consecuencia, se modificó el diseño de la red con la finalidad de parametrizarla en cuanto al tamaño y el tipo de datos y la precisión de los vectores de pesos. El principal cambio se realizó en el bloque que obtiene la diferencia entre un componente del vector de entrada y uno del vector de pesos, cuyo resultado se emplea tanto para calcular el valor absoluto como para actualizar el peso cuando la red opera en el modo de entrenamiento. En la Fig. 5.27 se ilustra el procesamiento que se realiza en ese bloque con la modificación realizada.

Los pesos (`W`) son números con signo de punto fijo con `e` bits en la parte entera y `f` bits en la parte fraccionaria. Los datos de entrada (`X`) son enteros con signo de `e` bits. Antes de efectuar la resta, el bit de signo de los dos operandos se extiende hacia la izquierda, además la parte fraccionaria del dato de entrada se llena con ceros. El resultado de la resta con `e+f+1` bits va directamente al actualizador de pesos. Los `e+1` bits de la parte entera se utilizan para obtener el valor absoluto con `e` bits.

Ya que los coeficientes de las sub-bandas de alta frecuencia del nivel de resolución 2 tienen 10 bits, la implementación de la red se realizó para `e=10` y `f=8`. Por lo tanto, los acumuladores de valor absoluto tendrán 14 bits y el tamaño de cada bloque de memoria para el almacenamiento de los pesos será de 16 x 18.

Resultados de la implementación del VQ

El diseño se implementó sobre la red de la placa del banco de entrenamiento. Primero se realizó la implementación de la red completa con la finalidad de efectuar el entrenamiento off-line y así generar los diccionarios. Después se implementó la red omitiendo el bloque de actualización de pesos y simplificando otros involucrados también con el entrenamiento para obtener una red reducida que sea capaz de operar únicamente en el modo de recall y ser utilizada como cuantizador vectorial.

En la Tabla 5.4 están los resultados de ocupación de recursos para las dos redes, la completa y la reducida.

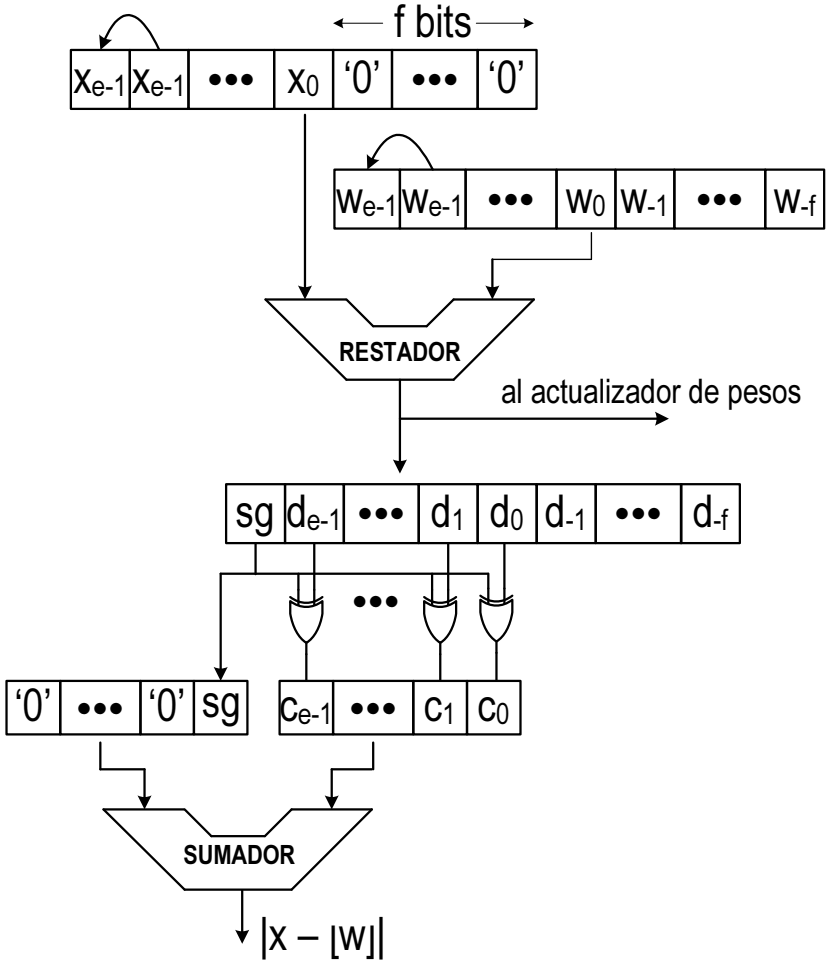


Fig. 5.27: Diferencia y valor absoluto para datos con signo de punto fijo

La máxima frecuencia alcanzada después del emplazamiento y ruteado es de 51 Mhz para la red completa y de 105 MHz para la red reducida. Para el recall, la red procesa un vector en 24 ciclos. Por lo tanto, operando a 100 MHz como VQ, la red reducida cuantiza un vector en 240 ns (con un rendimiento de 17067 MCPS), de modo que una matriz de 256 x 256 coeficientes (correspondientes a los del nivel de resolución 1 para una imagen de 512 x 512 pixeles) se cuantiza en 0,983 ms.

5.2.4 Codificador estadístico

Los coeficientes wavelet de la sub-banda LL2 se cuantizan escalarmente y después se codifican usando una técnica estadística. La cuantización escalar se realiza directamente desplazando hacia la derecha los coeficientes para así descartar los cuatro bits menos significativos. Entonces, los coeficientes cuantizados son números enteros de siete bits.

La codificación estadística empleada hace uso de la codificación diferencial con códigos prefijo de longitud variable. Esencialmente, la codificación diferencial consiste en tratar las diferencias entre coeficientes consecutivos en lugar de los valores de los coeficientes por sí mismos. Obviamente, el primer coeficiente se debe codificar directamente. La base de este

Recurso	Utilizado	Disponible	Utilización
	Red completa		
Flip-flops	11678	71680	16%
LUTs	65891	71680	91%
Slices	35838	35840	99%
	Red reducida		
Flip-flops	7915	71680	11%
LUTs	21152	71680	29%
Slices	11674	35840	32%

Tabla 5.4: Ocupación de recursos por la red en la FPGA

artificio reside en que los cambios ocurren de manera suave, por lo que la diferencia entre dos coeficientes consecutivos o adyacentes se pueden representar con menos bits. Los datos a codificar se obtienen con la ecuación siguiente:

$$d_i = \begin{cases} coef_i & i = 1 \\ coef_i - coef_{i-1} & i > 1 \end{cases} . \tag{5.4}$$

El código con prefijo variable utilizado se describe por medio de la Tabla 5.5. Los símbolos del código consisten en un prefijo seguido de un grupo de bits, también con longitud variable, que representan el entero con signo que se está codificando. Para que los prefijos sean distinguibles y, por tanto, sea posible llevar a cabo la decodificación, es necesario que ninguno sea la parte inicial de cualquier otro.

Grupo	Formato del símbolo	Cantidad de símbolos	Rango
1	**0**bbb	8	$[-4, +3]$
2	**10**bbb	8	$[-8, -5] \bigcup [+4, +7]$
3	**110**bbbb	16	$[-16, -9] \bigcup [+8, +15]$
4	**111**bbbbbbbb	128	$[-80, -17] \bigcup [+16, +79]$

Tabla 5.5: Formato para los símbolos del diccionario

La codificación basada en códigos de longitud variable será eficiente en la medida en que los valores más frecuentes se codifiquen con los símbolos de menor longitud y con muy poca frecuencia se requieran los símbolos más largos. En este caso se tienen símbolos de 4, 5, 7 y 11 bits, de manera que la compresión será mayor entre más valores a codificar estén en el rango del grupo 1 (-4 a +3). Para estimar la eficiencia de este codificador cuando procese coeficientes producidos por la transformación wavelet de imágenes típicas, se realizó un análisis estadístico utilizando seis imágenes. Los resultados se muestran en la Tabla 5.6 como porcentajes de distribución de las diferencias entre valores consecutivos de los coeficientes wavelet de la sub-banda LL2 en los rangos correspondientes a los cuatro grupos de símbolos.

Imagen	Ocurrencias %			
	Rango 1	Rango 2	Rango 3	Rango 4
Lena	82,23	8,98	5,66	3,13
Peppers	81,35	8,76	5,78	4,11
Boat	73,94	12,00	8,34	5,72
Barbara	75,52	11,79	8,17	4,52
Goldhill	71,76	16,30	9,07	2,87

Tabla 5.6: Distribución de coeficientes de LL2 en los grupos de símbolos

En la Fig. 5.28 se muestra el diagrama a bloques del codificador. El bloque `sq` realiza la cuantización escalar para reducir a 7 el número de bits utilizados para representar cada coeficiente. En el bloque `sbstr` se resta el coeficiente cuantizado actual con el previo que fue capturado en su momento en el registro `reg`. La diferencia resultante de la resta se compara en el bloque `cmprdr` para ubicarla en algún grupo de símbolos de acuerdo a su valor, activándose la salida correspondiente, ya sea `grp1`, `grp2`, `grp3` o `grp4`. Estas señales se utilizan para conformar el código, en el bloque `codgen` y para determinar la cantidad de desplazamientos en el registro `rdesp` que entrega en su salida los valores codificados en serie. La unidad de control saca una señal que indica la presencia de un bit válido (`bv`).

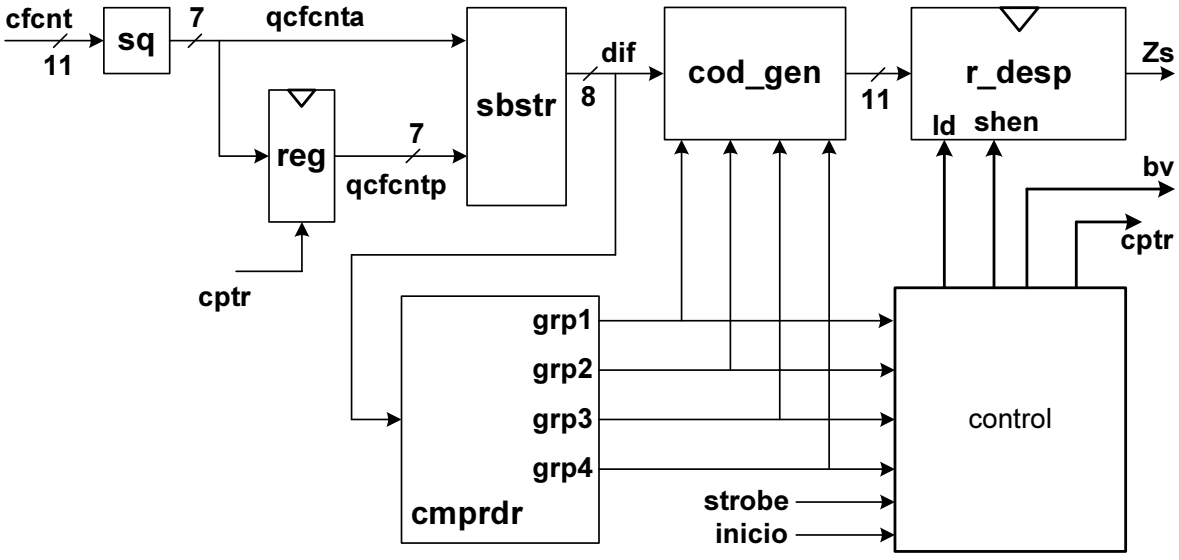


Fig. 5.28: Diagrama a bloques del codificador

En la Fig. 5.29 se muestra el diagrama de la máquina de estados que coordina la operación del codificador. Una vez iniciado el proceso, la máquina espera que se active la señal `strobe`. Entonces carga el registro de desplazamiento de salida con el símbolo y pasa al estado en el que se van desplazando los bits hasta que la cantidad de desplazamientos sea igual al tamaño del símbolo de acuerdo al grupo donde se ubicó la diferencia. Al concluir los desplazamientos, se captura en el registro el coeficiente actual para ser utilizado en el siguiente procesamiento

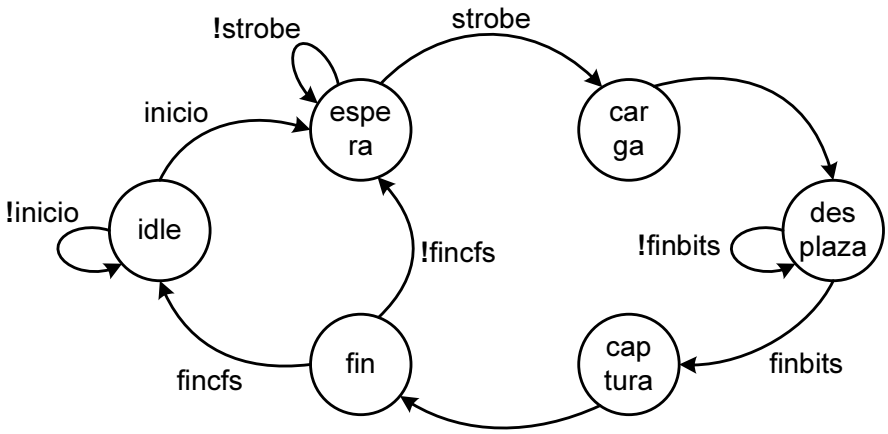


Fig. 5.29: Diagrama de transición de estados para el codificador

como coeficiente previo. La cantidad de ciclos requeridos para procesar un coeficiente es igual a 4 más el número de bits en el símbolo, la cual varía entre 4 para los símbolos del grupo uno hasta 11 para los del cuatro. Por lo tanto, la cadencia del codificador en el peor caso es de 15 ciclos por coeficiente.

El codificador se implementó sobre la FPGA de la placa. La frecuencia de operación alcanzada es de 184 MHz. La ocupación de recursos es poco significativa, sin llegar siquiera al 1% de los disponibles.

5.2.5 Implementación, experimentación y resultados

Implementación

El sistema de compresión completo se implementó sobre la Virtex 4 LX80. La Tabla 5.7 muestra la ocupación de recursos.

Recurso	Utilizado	Disponible	Utilización
Flip-flops	16970	71680	24%
LUTs	43930	71680	61%
Slices	24443	35840	68%
Bloques de RAM	42	200	21%

Tabla 5.7: Recursos de la FPGA ocupados por el sistema

Para una frecuencia de operación de 50 Mhz, considerando las tres etapas del sistema, la temporización del flujo de datos se muestra en la Fig. 5.30. El procesamiento completo de una imagen de 512 x 512 pixeles requiere 12,22 milisegundos. Esto representa más del doble de la tasa de procesamiento de tramas requerida por el video estándar.

Es importante analizar la relación que existe entre la cadencia de la etapa 2D-DWT y la de la VQ. Las sub-bandas del nivel de resolución uno contienen 256 x 256 coeficientes. De acuerdo con la Ec. 5.1, el cómputo de cuatro filas de coeficientes del nivel dos ($M = 256$ y $L = 16$) requiere de 4644 ciclos. Con estas cuatro filas de coeficientes se forman 32 vectores

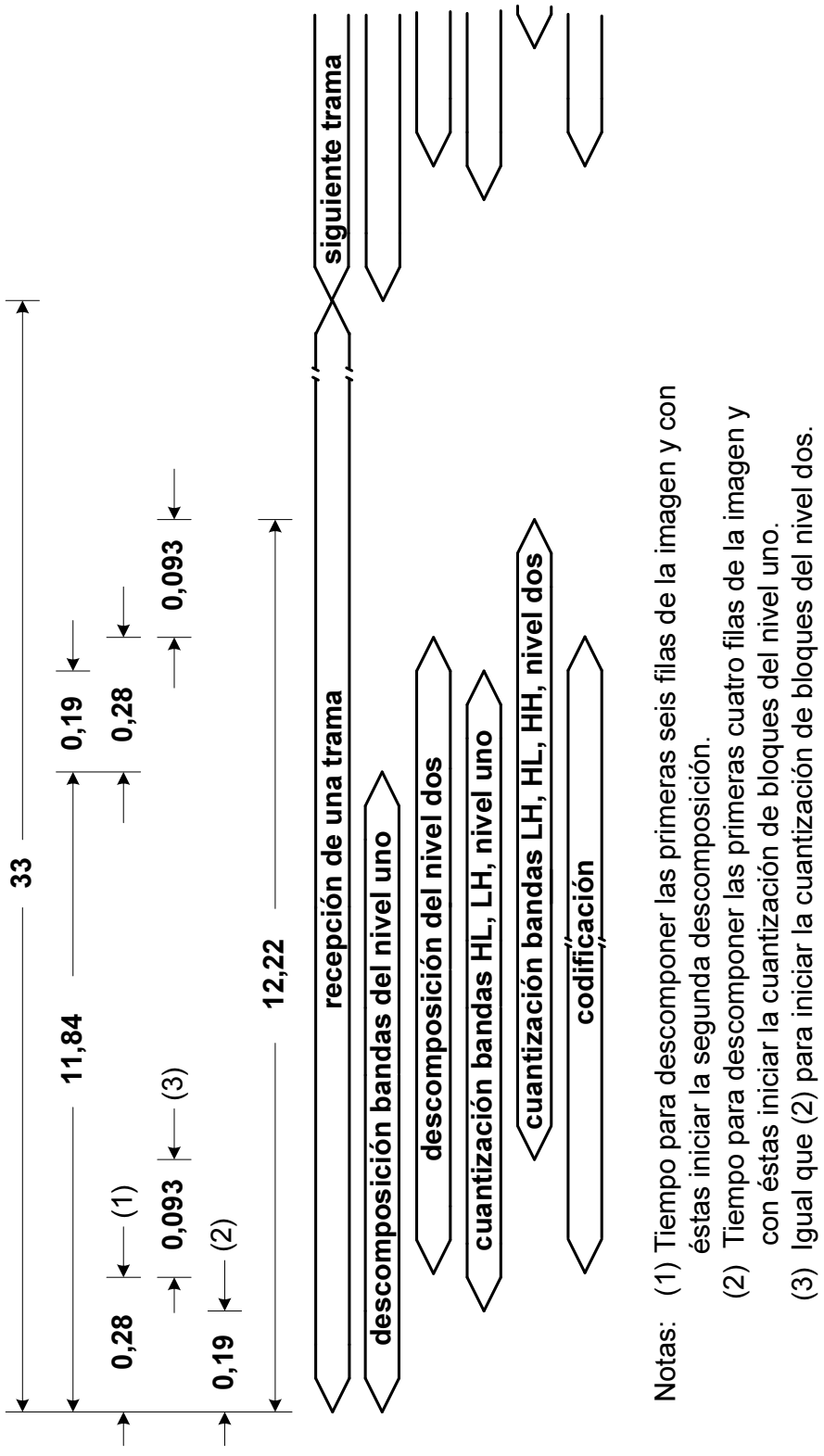


Fig. 5.30: Temporización del flujo de datos

que el VQ deberá cuantizar antes de que la 2D-DWT genere otras cuatro filas. Debido a que la cadencia de la VQ es de 24 ciclos por vector, se requieren 768 ciclos para cuantizar las cuatro filas de coeficientes correspondientes a una sub-banda y, por tanto, 2304 ciclos para las tres sub-bandas (LH2, HL2 y HH2) utilizando la misma red como VQ. Por lo tanto, la VQ2 (Fig. 5.22) se puede reutilizar para cuantizar las tres sub-bandas, renglón por renglón, de manera alternada.

Respecto a la cadencia del codificador, aunque en el peor caso es menor que la del 2D-DWT (15 contra 9 ciclos por coeficiente), esto no representa un problema mayor puesto que el 2D-DWT para completar el procesamiento de una fila del nivel uno (64 coeficientes) necesita haber completado previamente el procesamiento de una fila de la imagen (128 coeficientes). En otras palabras, el tiempo disponible para que el codificador procese una fila de coeficientes del nivel de resolución dos está determinado por el tiempo de procesamiento de la imagen original, el cual es el doble que la del nivel uno, debido a la cantidad de coeficientes.

Experimentación

La experimentación se realizó trabajando con las cinco imágenes de 512 x 512 pixeles en tonos de gris que se listan en la Tabla 5.6. Para generar los diccionarios multiresolución, se usaron todas las imágenes, con excepción de Peppers. A cada imagen se le aplicó la descomposición wavelet de dos niveles y todos los coeficientes de la misma sub-banda se usaron para construir el conjunto de entrenamiento respectivo. Finalmente, cada conjunto se aplicó a la red con 256 neuronas para generar el diccionario correspondiente. El entrenamiento se realizó fuera de línea utilizando el banco desarrollado.

Cada vector de entrenamiento se formó con un bloque de 4 x 4 coeficientes. De este modo, cada conjunto de entrenamiento para las sub-bandas del nivel de resolución uno tuvo 16384 vectores y para las del nivel de resolución dos constó de 4096 vectores. Los diccionarios multiresolución generados de esta manera se incorporaron al sistema por medio de los bloques de memoria ROM que se muestran sombreados en negro en la Fig. 5.22.

Dos imágenes fueron sometidas al proceso de compresión-reconstrucción para evaluar la calidad del sistema completo. Se utilizó el valor pico de la relación señal a ruido (PSNR) como métrica. Para Lena, incluida en el conjunto de entrenamiento, se obtuvo un valor de 34,78 dB y para Peppers, no incluida, 31,28 dB. Para estimar la calidad del sistema mediante la percepción visual, en la Fig. 5.31 se muestran las imágenes original y reconstruida para el caso de Lena y en la Fig. 5.32 para el de Peppers.

Se calculó la razón de compresión para las dos imágenes, obteniéndose 0,624 bpp para Lena y 0,628 para Peppers.

Comparación

En la Tabla 5.8 el sistema desarrollado se compara con tres previos que también utilizan la combinación 2D-DWT/VQ. Se toman en consideración los siguientes aspectos: tipo de implementación, ya sea software (SW) o hardware (HW); el método para la cuantización vectorial, ya sea basado en una red SOM o no; la manera como se generan los diccionarios, es decir, con el mismo sistema o con algún software independiente; en su caso, la tecnología empleada para la implementación hardware; la razón de compresión y la calidad de la compresión como valor del PSNR. El sistema desarrollado destaca debido a que cubre las tres etapas en la implementación hardware usando una tecnología específica; también porque la VQ está basada

Fig. 5.31: Lena original y reconstruida

Fig. 5.32: Peppers original y reconstruida

en una red SOM y, como consecuencia, los diccionarios se generan con la red en un tiempo relativamente corto, y porque la calidad de la imagen reconstruida es alta para una razón de compresión aceptable. Los resultados de la compresión para las referencias [16] y [113] se obtuvieron experimentando con imágenes típicas de 512 x 512 pixeles, tales como las usadas para esta tesis. En la referencia [78] se utilizaron imágenes médicas de color de 128 x 128.

Diseño	Ref. [16]	Ref. [113]	Ref. [78]	Desarrollado
Implementación	SW	HW	SW	HW
VQ basada en SOM	No	No	Sí	Sí
Diccionarios en diseño	No	No	Sí	Sí
Tiempo real	No	Si	No	Sí
Tecnología	—	—	—	FPGA
CR (bpp)	0,78	0,48	nd	0,62
PSNR (dB)	32,1	32	38	34,78

Tabla 5.8: Comparación con trabajos previos

5.3 Conclusiones

Se estructuró y diseñó el banco de entrenamiento para la red SOM con lo cual se tiene la posibilidad de establecer en tiempo de ejecución:

- El tamaño del conjunto de entrenamiento.
- El valor inicial de la tasa de aprendizaje.
- La rapidez de decaimiento de la tasa de aprendizaje.
- El tipo de entrenamiento, ya sea por lotes o adaptativo.
- La cantidad de epochs

Se demostró la funcionalidad del banco de entrenamiento mediante dos breves aplicaciones de investigación y se utilizó para generar los diccionarios multiresolución de un sistema que integra la transformada wavelet bidimensional y la cuantización vectorial.

Para este sistema cabe destacar lo siguiente:

- Los resultados experimentales prueban que el sistema tiene capacidad para procesar video en tiempo real para imágenes en tonos de gris de 512 x 512 pixeles ya que supera ampliamente la tasa de 30 imágenes por segundo.
- El sistema completo ocupa 68% de los recursos de lógica del dispositivo FPGA Virtex 4 LX80 y 20% de sus bloques de memoria. Los siete diccionarios multiresolución fueron generados con la misma red SOM operando en la fase de entrenamiento en sólo aproximadamente 4 segundos cada uno.

- Usando el PSNR como métrica, la calidad numérica de la compresión fue de 34,78 dB para una imagen incluida en el conjunto de entrenamiento y 31,28 para una imagen no incluida, con una tasa de compresión de 0,62 bpp.
- La reducción de los efectos de bloques por la 2D-DWT mejora significativamente la percepción visual de la imagen reconstruida.
- Como producto de la investigación se publicó el artículo con la siguiente referencia:
  A. Ramirez, R. Gadea, R. Colom, J. Diaz; "A wavelet-VQ system for real-time video compression"; *Springer Science Journal of Real-Time Image Processing*, **2**(4), 271-280, 2007.

Capítulo 6

Conclusiones y trabajos futuros

6.1 Conclusiones

En esta Tesis Doctoral se elaboró el diseño hardware de una red neuronal SOM con la finalidad de ser utilizada en un sistema para compresión de imágenes y video. El diseño es modular, escalable y proporciona flexibilidad para establecer algunas características de la red en tiempo de implementación, así como el tipo de entrenamiento y sus parámetros en el tiempo de ejecución.

Como preámbulo del diseño se realizó una revisión de las técnicas para compresión de imágenes basadas en redes neuronales y se efectuó una experimentación software con las que se consideraron más viables para su implementación hardware.

El diseño de la red SOM partió del análisis de cinco arquitecturas factibles de ser utilizadas, considerando como factores de comparación la ocupación de recursos y aspectos temporales como son la cadencia y la latencia.

El diseño físico de la red SOM incluye su síntesis, implementación y configuración en un dispositivo FPGA alojado en una placa de aplicación con interface PCI. De este modo, esta placa se configuró como la base de un neurocoprocesador para un banco de entrenamiento hardware-software que permite integrar en un mismo ambiente las tareas de preparación del conjunto de vectores de entrenamiento, la ejecución del entrenamiento y del recall, y el análisis de los resultados del proceso de compresión-reconstrucción de imágenes para evaluar el comportamiento de la red.

Para la evaluación de la red se emplean dos métodos. Uno consiste en la comparación con otros diseños relacionados publicados recientemente. El otro tiene como base la comparación de los resultados con respecto a los obtenidos mediante el algoritmo LBG, tradicionalmente empleado para la cuantización vectorial. Por tal razón, como parte de las funciones del software desarrollado se incorporan algunas vinculadas con este algoritmo.

A continuación se enumeran las principales aportaciones de esta Tesis:

1. El resultado de la revisión y experimentación de las arquitecturas de redes neuronales aplicadas para la compresión de imágenes señaló que el camino a seguir para la investigación en este campo está por el lado de las redes SOM para la cuantización vectorial y los predictores no lineales mediante el perceptrón multicapa, así como la combinación de ambas técnicas.

2. Se realizó un análisis de las arquitecturas apropiadas para la implementación hardware de las redes SOM.

3. Se desarrolló la red SOM incorporando todas las funcionalidades requeridas para su operación en las fases de entrenamiento y de recall.

4. Se incluyó en la red la técnica de sensibilidad a la frecuencia ganadora con la finalidad de reducir la probabilidad de que en el entrenamiento ocurran neuronas inactivas.

5. La red se integró en un banco de entrenamiento hardware-software que permite establecer en tiempo de ejecución el valor inicial de la razón de aprendizaje, la rapidez con la que este valor decae y el número de epochs.

6. El entorno del banco de entrenamiento desarrollado proporciona las funciones para realizar directamente las tareas previas y posteriores al entrenamiento, siendo éstas la conformación de los pesos iniciales de la red, la estructuración de los vectores del conjunto de entrenamiento y la recuperación y almacenamiento de los resultados.

7. Las funciones para el entorno del banco de entrenamiento incluyen también las tareas para la cuantización vectorial de una imagen, su posterior reconstrucción y la evaluación de la calidad del proceso de compresión-reconstrucción.

8. La red desarrollada es capaz de procesar video en tiempo real satisfaciendo los requerimientos de velocidad para imágenes de hasta 1024 x 768 pixeles.

9. La red superó en velocidad a otros diseños similares reportados recientemente. La calidad del compresor medida por los valores del PSNR después del proceso codificación-reconstrucción es ligeramente superior a la obtenida usando el algoritmo LBG con la misma función de distancia. Esta aportación y la del punto anterior fueron publicadas en [114].

10. Se estableció la estructura para un sistema de compresión de imágenes que incluye la transformada wavelet y la cuantización vectorial, así como una técnica de codificación estadística para los coeficientes de la sub-banda con la mayor cantidad de información. Esta estructura está publicada en [115].

11. El sistema se implementó físicamente y los resultados de la experimentación comprueban su capacidad para realizar el procesamiento en tiempo real de imágenes en tonos de gris de 512 x 512 pixeles con valores de distorsión equiparables a los de otros diseños reportados recientemente, con una favorable calidad perceptual originada por la inclusión de la etapa de transformación.

12. El análisis de los resultados obtenidos pone en evidencia que el empleo de la función Manhattan como medida de la distancia entre vectores representa un costo en la calidad del codificador en algunas décimas de decibelio.

6.2 Líneas para trabajos futuros

Al revisar el trabajo realizado durante la investigación, se observan tareas que quedan pendientes, ya sea por motivo de falta de tiempo para escudriñar de manera más detallada algunas

variantes de estudio, o porque están fuera del alcance original del tema. Entre ellas se sugieren como trabajos futuros los siguientes:

1. Efectuar un análisis de los recursos de la FPGA requeridos para utilizar en la red la distancia euclidiana y rediseñar la red con tal propósito.
2. Retomar el sistema de compresión expuesto en §5.2 en el cual quedan tres trabajos pendientes:
   - Estudiar el reemplazo del codificador utilizado para los coeficientes de la sub-banda de baja frecuencia por uno basado en un predictor no lineal por medio de un MLP como el propuesto en [64].
   - Aplicar un tercer nivel de descomposición para la 2D-DWT y estudiar los posibles esquemas de cuantización vectorial multiresolución.
   - Incorporar alguna técnica de codificación entrópica para codificar los índices que resultan de la cuantización vectorial.
3. Aplicar la red SOM para la reconstrucción de imágenes PET bidimensionales, trabajo sugerido someramente en [115].
4. Modificar la estructura del banco de entrenamiento con la finalidad de permitir el monitoreo del proceso de aprendizaje de manera dinámica.
5. Rediseñar la red SOM para obtener una red LVQ con entrenamiento supervisado y de este modo utilizarla para alguna aplicación de diagnóstico, por ejemplo en vibraciones de máquinas.
6. Analizar la factibilidad y conveniencia de un rediseño de la red hacia una solución híbrida hardware-software al nivel SoC.
7. Modificar el diseño del bloque de la red SOM que realiza la función de sensibilidad a la frecuencia ganadora de las neuronas con el propósito de que tenga flexibilidad en cuanto al grado de penalización que ejerce sobre las neuronas ganadoras.

Bibliografía

[1] R.C. Gonzalez, R.E. Woods; *Digital image processing*, Addison-Wesley Publishing Company, New York, 1992.

[2] C.E. Shannon; "A mathematical theory of communication", *The Bell System Technical Journal*, **27**, 379-423, 623-656, 1948.

[3] D.A. Huffman; "A method for the construction of minimum-redundancy codes", *Proceedings of the Institute of Radio Engineers*, **40**(9), 1098-1101, 1952.

[4] J. Rissanen, G. Langdon; "Arithmetic coding", *IBM J. Res. Develop.*, **23**, 149-162, 1979.

[5] C. Jones; "An efficient coding system for long source sequences", *IEEE Transactions on Information Theory*, **27**(3), 280-291, 1981.

[6] B.M. Oliver, J. R. Pierce, C. E. Shannon; "The philosophy of PCM", *Proceedings of the Institute of Radio Engineers*, **36**, 1324-1331, 1948.

[7] B.M. Oliver; "Efficient coding", *The Bell Systems Technical Journal*, **31**, 724-750, 1952.

[8] P. Elias; "Predictive coding", *IRE Transactions on Information Theory*, **1**(1), 16-33, 1955.

[9] The Joint Photographic Experts Group; "Digital compression and coding of continuous-tone still images", *International Telecommunication Union, Recommendation T.81*, 1992.

[10] J. Hadamard; "Résolution d'une question relative aux déterminants", *Bull. Sci. Math.*, **17**, 240-246, 1893.

[11] J.L. Walsh; "A closed set of normal orthogonal functions", *Amer. J. Math.*, **45**, 5-24, 1923.

[12] J. Ziv, A. Lempel; "A universal algorithm for sequential data compression", *IEEE Transactions on Information Theory*, **23**, 337-343, 1977.

[13] T.A. Welch; "A technique for high performance data compression", *IEEE Computer*, **17**(6), 8-19, 1984.

[14] Y. Ming, N. Bourbakis; "An overview of lossless digital image compression techniques", *IEEE Midwest Symposium on Circuits and Systems*, **2**, 1099-1102, 2005.

[15] J. Miao, D. Huo, D. Wilson; "Perceptual difference model (Case-PDM) for evaluation of MR images: validation and calibration", *en Y. Jiang, B. Sahiner (eds.), Medical Imaging 2007: Image Perception, Observer Performance, and Technology Assessment*, SPIE Digital Library, **6515**, 2007.

[16] M. Antonini, M. Barlaud, P. Mathieu, I. Daubechies; "Image coding using wavelet transform", *IEEE Transactions on Image Processing*, **1**(2), 205-220, 1992.

[17] G. Ginesu, F. Massida, D. Giusto; "A multi-factors approach for image quality assessment based on a human visual system model", *Elsevier, Signal Processing: Image Communication*, **21**(4), 316-333, 2006.

[18] K. Karhunen; "Uber lineare Methoden in der Wahrscheinlichkeitsrechnung", *Ann. Acad. Sci. Fennicae, Ser. A. I. Math-Phys*, **37**, 1-79, 1947.

[19] N.T. Ahmed, T. Natrajan, K.R. Rao; "Discrete Cosine Transform", *IEEE Transactions on Computers*, **23**(1), 90-93, 1974.

[20] A. Haar; "Zur Theorie der orthogolanen Funktionen-Systeme", *Mathematische Annalen*, **69**, 331-371, 1910.

[21] Y. Sheng; "Wavelet transform", *en A.D. Poularikas (ed.), The Transforms and Applications Handbook*, CRC, Press, Inc., 1996.

[22] R.J. Colom; "Estudio e implementación de la transformada wavelet para la compresión de imágenes y video", *Tesis Doctoral, Departamento de Ingeniería Electrónica*, Universidad Politécnica de Valencia, 2001.

[23] A. Gersho, R.M. Gray; *Vector quantization and signal compression*, Kluwer Academic Publishers, Norwell, MA. 1992.

[24] M. Fujibayashi, T. Nozawa, T. Nakayama, K. Mochizuki, M. Konda, K. Kotani, S. Sugawa, T. Ohmi; "A still-image encoder based on adaptive resolution vector quantization featuring needless calculation elimination architecture", *IEEE Journal of Solid-State Circuits*, **38**(5), 726-733, 2003.

[25] H. Sun, K. Lam, S. Chung, W. Dong, M. Gu, J. Sun; "Efficient vector quantization using genetic algorithm", *Springer Neural Computing and Applications*, **14**, 203-211, 2005.

[26] P. Franti, J. Kivijarvi, O. Nevalainen; "Tabu search algorithm for codebook generation in vector quantization", *Elsevier Pattern recognition*, **31**(8), 1139-1148, 1998.

[27] S. Pan, K. Cheng; "An evolution-based tabu search approach to codebook design", *Elsevier Pattern recognition*, **40**, 476-491, 2007.

[28] S.P. Lloyd; "Least squares quantization in pcm", *IEEE Transactions on Information Theory*, **28**, 127-135, 1982.

[29] Y. Linde, A. Buzo, R. Gray; "An algorithm for vector quantizer design", *IEEE Transactions on Communications*, **28**(1), 84-95, 1980.

[30] C.C. Aggarwal, A. Hinneburg, D.A. Keim; "On the surprising behavior of distance metrics in high dimensional space", *Springer Lecture Notes in Computer Science*, **1973**, 420-434, 2001.

[31] Y. Yano, T. Koide, H.J. Mattausch; "Associative memory with fully parallel nearest-Manhattan-distance search for low-power real-time single-chip application", *IEEE Proceedings of the 2004 Conference on Asia South Pacific design automation*, 543-544, 2004.

[32] Ch. Chang, Ch. Tai, W.P. Chung, Ch.T. Shou, Ch.H. Yi; "Reducing computation for vector quantization by using bit-mapped look-up tables", *IEEE Transactions on Networking, Sensing and Control*, **2**, 1229-1234, 2004.

[33] A. Nakada, T. Shibata, M. Konda, T. Morimoto, T. Ohmi; "A fully parallel vector-quantization processor for real-time motion-picture compression", *IEEE Journal of Solid-State Circuits*, **34**(6), 822-830, 1999.

[34] M. Ogawa, K. Ito, T. Shibata; "A general-purpose vector-quantization processor employing two-dimensional bit-propagating winner-take-all", *IEEE Symposium on VLSI Circuits, Digest of Technical Papers*, 244-247, 2002.

[35] W.S. McCulloch, W. Pitts; "A logical calculus of the ideas immanent in nervous activity", *Bulletin of Mathematical Biophysics*, **5**, 115-133, 1943.

[36] D.O. Hebb; *The organization of behavior: a neuropsychological theory*, John Wiley and Sons Inc, New York, 1949.

[37] F. Rosenblatt; "The perceptron: a probabilistic model for information storage and organization in the brain", *Psychology Review*, **65**(6), 386-408, 1958.

[38] J. Hertz, A. Krogh, R.G. Palmer; *Introduction to the theory of Neural Computation*, Addison-Wesley, Redwood City, CA, 1991.

[39] R.M. Tallam, T.G. Habetler, R.G. Harley; "Continual on-line training of neural networks with applications to electric machine fault diagnostics", *IEEE Power Electronics Specialists Conference*, **4**, 17-21, 2001.

[40] B. Linares, A. G. Andreou, G. Indiveri, T. Shibata; "Guest editorial, special issue on neural networks hardware implementations", *IEEE Transactions on Neural Networks*, **14**(5), 976-979, 2003.

[41] U. Rückert; "Microelectronic implementation of neural networks", *Proceedings of Workshop on Neural Networks*, 77-86, 1993.

[42] P.C. Treleaven; "Neurocomputers", *International Journal of Neurocomputing*, **1**, 4-31, 1989.

[43] U. Ramacher, U. Rückert; *VLSI design of neural networks*, Kluwer Academic Publishers, 1991.

[44] T. Nordström, B. Svensson; "Using and designing massively parallel computers for artificial neural networks", *Journal of Parallel and Distributed Computers*, **14**, 260-285, 1992.

[45] M. Vellasco; "A VLSI architecture for neural networks chips", *PhD Thesis, Department of Computer Science*, University of London, 1992.

[46] P. Ienne; "Quantitative comparison of architectures for digital neurocomputers", *Proceedings of the IJCNN*, 1987-1990, 1993.

[47] M. Glesner, M. Pöchmüller; *Neurocomputers: an overview of neural networks in VLSI*, Chapman & Hall Neural Computing, London, 1994.

[48] J. Heemskerk; "Neurocomputers for brain style processing. Design, implementation and application", *Phd thesis, Unit of Experimental and Theoretical Psychology*, Leiden University, 1995.

[49] C.S. Lindsey, T. Lindblad; "Review of hardware neural networks: a user's perspective", *Third Workshop on Neural Networks: From Biology to High Energy Physics*, 26-30, 1994.

[50] D.E. Rumelhart, G.E. Hinton, R.J. Williams; "Learning representations by back-propagating errors", *Nature*, **323**, 533-536, 1986.

[51] G.W. Cottrell, P. Munro, D. Zipser; "Learning internal representations from gray-scale images: an example of extensional programming", *9th Annu. Conf. of the Cognitive Soc.*, 462-473, 1987.

[52] R.D. Dony, S. Haykin; "Neural networks approaches to image compression", *Proceedings of the IEEE*, **83**(2), 288-303, 1995.

[53] O. Abdel, M.M. Fahmy; "Image Compression using multi-layer neural networks", *IEE Proceedings of Vision, Image and Signal Processing*, **144**(5), 307-312, 1997

[54] Y. Benbenisti, D. Kornreich, H. Mitchell, P. Schaefer; "Fixed bit-rate image compression using a parallel-structure multilayer neural network", *IEEE Transactions on Neural Networks*, **10**(5), 1166-1172, 1999.

[55] E. Watanabe, K. Mori; "Lossy image compression using a modular structured neural network", *Neural Networks for Signal Processing*, **XI**, 403-412, 2001.

[56] N. Charif, H. Salam; "Neural networks-based image compression system", *IEEE Proceedings of the 43rd Midwest Symposium on Circuits and Systems*, 846-849, 2000.

[57] A. Steudel, S. Ortmann, M. Glesner; "Medical Image Compression with Neural Nets" *IEEE Proceedings of the International Symposium on Uncertainty Modelling and Analysis*, **1**, 571-576, 1995.

[58] N.M. Nasrabadi, S.A. Dianat, S. Venkataraman; "Non-linear prediction using a three-layer neural network", *IEEE International Joint Conference on Neural Networks*, **1**, 689-694, 1991.

[59] C.N. Manikopoulos; "Neural network approach to DPCM system design for image coding", *IEE Proceedings of Communications, Speech and Vision*, **139**(5), 501-507, 1992.

[60] Ch. Burges, H.S. Malvar, P. Simard; "Improving wavelet image compression with neural networks", *Microsoft Research Technical Report-2001-47*, 1-18, 2001.

[61] D-CH. Park, T-H. Park; "DPCM with recurrent neural network predictor for image compression", *IEEE Proceedings of International Joint Conference on Neural Networks*, **2**, 826-831, 1998.

[62] S. Marusic, G. Deng; "A neural network based adaptive non-linear lossless predictive coding technique", *IEEE Proceedings of the International Symposium on Signal Processing and its Applications*, **2**, 653-656, 1999.

[63] S. Marusic, G. Deng; "Adaptive prediction for lossless image compression", *Elsevier Signal Processing: Image Communication*, **17**, 363-372, 2002.

[64] R. Gadea, A. Ramirez; "FPGA implementation of non-linear predictors", *en A.R. Omondi, J.C. Rajapakse (eds.), FPGA Implementations of Neural Networks*, 297-323, Springer, Netherlands, 2006.

[65] A. Rizvi, L.CH. Wang, N.M. Nasrabadi; "Nonlinear vector prediction using feed-forward neural networks", *IEEE Transactions on Image Processing*, **6**(10), 1431-1436, 1997.

[66] C. Von der Malsburg; "Self-organization of orientation sensitive cells in the striate cortex", *Kybernetik*, **14**, 85-100, 1973.

[67] T.N. Wiesel, D.H. Hubel; "Comparison of effects of unilateral and bilateral eye closure on cortical unit responses in kittens", *Journal of Neurophysiology*, **28**(6), 1029-1040, 1965.

[68] G. Carpenter, S. Grossberg; "Absolutely stable learning of recognition codes by a self-organizing neural network", *AIP, Conference Proceedings*, **151**(1), 77-85, 1986.

[69] G. Carpenter, S. Grossberg; "ART2: Self-organization of stable category recognition codes for analog input patterns", *Applied Optics*, **26**(23) 4919-4930, 1987.

[70] G. Carpenter, S. Grossberg, J. Reynolds; "ARTMAP: a self-organizing neural network architecture for fast supervised learning and pattern recognition", *IEEE International Joint Conference on Neural Networks*, **1**, 863-888, 1991.

[71] T. Kohonen; "Learning vector quantization", *Neural Networks*, **1**(23), 214-220, 1981.

[72] T. Kohonen; "The self-organizing map", *Proceedings of the IEEE*, **78**(9), 1464-1480, 1990.

[73] T. Kohonen; "Self-organized formation of topologically correct feature maps", *Biological Cybernetics*, **43**, 59-69, 1982.

[74] T. Kohonen, K. Makisara, T. Saramaki; "Phonotopic maps-insightful representation of phonological features for speech recognition", *IEEE Proceedings of the 7th Int. Conference on Pattern Recognition*, 182-185, 1984.

[75] T. Kohonen; "Automatic formation of topological maps of patterns in a self-organizing system", *Proceedings of 2SCIA, Scand. Conference on Image Analysis*, 214-220, 1981.

[76] N. Nasrabadi, Y. Feng; "Vector quantization of images based upon the Kohonen self-organizing feature maps"; *IEEE International Conference on Neural Networks* **1**, 101-108, 1988.

[77] F. Madeiro, M. Vajapeyam, M. Morais, B. Aguiar, M. Alencar; "Multiresolution codebook design for wavelet/VQ image coding", *IEEE Proceedings of 15th International Conference on Pattern Recognition*, **3**, 75-78, 2000.

[78] K. Kim, S. Kim, G. Kim; "Vector quantizer of medical image using wavelet transform and enhanced SOM algorithm", *Springer Neural Computing and Applications*, **15**(3), 245-251, 2006.

[79] R. Li, J. Kim; "Image compression using fast transformed vector quantization", *IEEE Proceedings of 29th Applied Imagery Pattern Recognition Workshop*, 141-145, 2000.

[80] R. Li, E. Sherrod, J. Kim, G. Pan; "Fast image vector quantization using a modified learning neural network approach", *Wiley International Journal of Imaging Systems and Technology*, **8**(4), 413-418, 1997.

[81] C. Amerijckx, M. Verleysen, P. Thissen, J. Legat; "Image compression by self-organized Kohonen map", *IEEE Transactions on Neural Networks*, **9**(3), 503-507, 1998.

[82] H. Soliman; "Neural net simulation: SFSN model for image compression", *IEEE, Proceedings of 34th Annual Simulation Symposium*, 325-332, 2001.

[83] J. Barbalho, A. Duarte, D. Neto, J. Costa, M. Netto; "Hierarchical SOM applied to image compression", *IEEE, Proceedings of International Joint Conference on Neural Networks*, **1**, 442-447, 2001.

[84] W. Fang, B. Sheu, O. Chen, J. Choi; "A neural network based VLSI vector quantizer for real-time image compression", *IEEE Transactions on Neural Networks*, **3**(3), 506-518, 1992.

[85] I. Kuon, J. Rose; "Measuring the gap between FPGAs and ASICs", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **26**(2), 203-215, 2007.

[86] H. Hikawa; "FPGA implementation of self organizing map with digital phase locked loops", *Elsevier Neural Networks*, **18**(5), 514-522, 2005.

[87] D.C. Hendry, A.A. Duncan, N. Lightowler; "IP core implementation of a self-organizing neural network", *IEEE Transactions on Neural Networks*, **14**(5), 1085-1096, 2003.

[88] H. Tamukoh, T. Aso, K. Horio, T. Yamakawa; "Self-organizing map hardware accelerator system and its application to realtime image enlargement", *Proceedings of the IJCNN2004*, 2683-2687, 2004.

[89] T. Kohonen "Self-organized formation of topologically correct feature maps", *MIT Press Neurocomputing: foundations of research*, 509-521, 1988.

[90] M. Franzmeier, C. Pohl, M. Porrmann, U. Ruckert; "Hardware accelerated data analysis", *International Conference on Parallel Computing in Electrical Engineering*, 309-314, 2004.

[91] G.E. Moore; "Progress in digital integrated electronics", *IEEE Technical Digest of International Electron Devices Meeting*, 11-13, 1975.

[92] Xilinx; "Virtex-4 Family overview", Versión 3.0, 2007, disponible en: www.xilinx.com.

[93] Xilinx; "Virtex-5 Family overview", Versión 4.0, 2008, disponible en: www.xilinx.com.

[94] Synopsys Inc.; "SystemC users guide", Versión 2.0, 2003, disponible en: www.systemc.org

[95] Celoxica; "DK design suite user guide", 2005, disponible en: www.celoxica.com/support/

[96] N. Calazans, E. Moreno, F. Hessel, V. Rosa, F. Moraes, E. Carara; "From VHDL register transfer level to SystemC transaction level modelling: a comparative case study", *IEEE Proceeding of Symposium on Integrated Circuits and Systems Design*, **1**, 355-360, 2003.

[97] L.M. Reyneri; "A simulink-based hybrid codesign tool for rapid prototyping of FPGA's in signal processing systems", *Elsevier Microprocessors and Microsystems*, **28**(5-6), 273-289, 2004.

[98] K. Banovic, M.A. Khalid, E. Abdel; "FPGA-Based rapid prototyping of digital signal processing systems", *IEEE Proceedings of Midwest Symposium on Circuit and Systems*, **1**, 648-650, 2005.

[99] S. Tsasakou, N.S. Voros, M. Koziotis, D. Verkest, A. Prayati, A. Birbas; "Hardware-software co-design of embedded systems using CoWares's N2C methodology for application development", *IEEE Proceedings of International Conference on Electronics, Circuits and Systems*, **1**, 59-62, 1999.

[100] M. Lícko, J. Schier, M. Tichý, Markus Kühl; "MATLAB/Simulink based methodology for rapid-FPGA-prototyping ", *Springer Lecture Notes in Computer Science*, **2778**, 984-987, 2003.

[101] J. Hwang, B. Milne, N. Shirazi, J.D. Stroomer; "System level tools for DSP in FPGAs", *Springer Lecture Notes in Computer Science*, **2147**, 534-543, 2001.

[102] N. Shirazi, J. Ballagh; "Put hardware in the loop with Xilinx System Generator for DSP", *Xilinx Xcell Journal*, **Fall**, 2003.

[103] Mathworks, Inc.; "Simulink", http://www.mathworks.com/productos/simulink/

[104] Alpha Data; "ADM-XRC-4 (LX/SX) User Guide", version 1.4, 2008, disponible en www.alpha-data.com.

[105] Alpha Data; "XRC board level application library", versión 2.1d, 2004, disponible en www.alpha-data.com.

[106] Alpha Data; "XRC Matlab toolbox", version 1.8, 2006, disponible en www.alpha-data.com.

[107] X. Wu, L. Guan; "Acceleration of the LBG algorithm", *IEEE Transactions on Communications*, **42**(234-3), 1518-1523, 1994.

[108] Y-Ch. Lin, S-Sh. Tai; "A fast Linde-Buzo-Gray algorithm in image vector quantization", *IEEE Transactions on Circuits and Systems-II: Analog and Digital Signal Processing*, **45**(3), 432-435, 1998.

[109] P-Ch. Wang, Ch-L. Lee, H-Y. Chang, T-Sh. Chen; "Hardware accelerator for vector quantization by using pruned lookup table", *Springer Lecture Notes in Computer Science* **3483**, 1007-1016, 2005.

[110] J. Vesanto, J. Himberg, E. Alhoniemi, J. Parhankangas; "Self-organizing map in Matlab: the SOM Toolbox", *Proceedings of the Matlab DSP Conference*, 35-40, Espoo Finland, 1999.

[111] R. Colom, R. Gadea, A. Sebastia; "A novel FPGA architecture of a 2-D wavelet transform", *Springer Science Journal of VLSI Signal Processing*, **2**, 273-284, 2006.

[112] P.P. Vaidyanathan; *Multirate systems and filter banks*, Prentice Hall International Inc., 1993.

[113] S.K. Paek, L.S. Kim; "A real-time wavelet vector quantization algorithm and its VLSI architecture", *IEEE Transactions on Circuits and Systems for Video Technology*, **10**(3), 475-489, 2000.

[114] A. Ramirez, R. Gadea, R. Colom; "A hardware design of a massive-parallel, modular NN-based vector quantizer for real-time video coding"; *Elsevier Microprocessors and Microsystems*; **32**(1), 33-44, 2008.

[115] A. Ramirez, R. Gadea, R. Colom, J. Diaz; "A wavelet-VQ system for real-time video compression", *Springer Science Journal of Real-Time Image Processing*, **2**(4), 271-280, 2007.