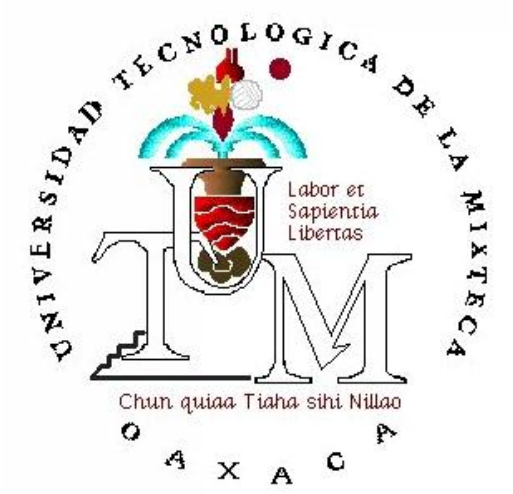

UNIVERSIDAD TECNOLÓGICA DE LA MIXTECA

Cuantificación Vectorial de Imágenes con base en Memorias Asociativas Extendidas

T E S I S

QUE PARA OBTENER EL GRADO DE

MAESTRO EN CIENCIAS EN ELECTRÓNICA Y COMPUTACIÓN

PRESENTA

MIGUEL ANGEL RAMÍREZ JIMÉNEZ

DIRECTOR: DR. ENRIQUE GUZMÁN RAMÍREZ

HUAJUAPAN DE LEON, OAXACA. MAYO, 2013.

Resumen

El desarrollo de nuevas técnicas de compresión de imágenes de forma eficaz y eficiente continúa. Es casi seguro que la mayoría, sino es que todas las imágenes que se observan en el internet utilicen algún tipo de formato de compresión para que al momento de reproducirlas se pueda tener mayor velocidad de carga, menor espacio utilizado en memoria y mayor capacidad de almacenamiento en disco para su resguardo.

Este trabajo de tesis propone un esquema de cuantificación vectorial que basa su funcionamiento en el uso del algoritmo LBG y las Memorias Asociativas Extendidas (MAE), denotado como VQ-MAE. El esquema propuesto utiliza uno de cuatro operadores en que basan su operación las MAE, operador **prom**, operador **med**, operador **pmed** y operador **sum**, tanto en el proceso de generación del libro de códigos como en el proceso de búsqueda del vector de reconstrucción. Mediante resultados experimentales se muestra el desempeño del esquema VQ-MAE con cada uno de estos operadores, este desempeño se midió utilizando el popular criterio objetivo de relación señal a ruido.

La mayor parte de los operadores utilizados en la operación de las MAE basan su funcionamiento en comparaciones y restas. La simplicidad de estas operaciones permite a las MAE ofrecer una alta velocidad de procesamiento y baja demanda de recursos (memoria del sistema), características que son heredadas al esquema VQ-MAE. Se muestran los análisis de complejidad temporal y espacial de las variantes del VQ-MAE.

Los resultados obtenidos de estos experimentos demuestran que el esquema propuesto se mantiene competitivo con los métodos tradicionales con lo que respecta a los parámetros de relación de compresión y relación señal a ruido, mientras que en velocidad de procesamiento y recursos utilizados durante su operación los supera.

Finalmente, para comparar el desempeño ofrecido por el VQ-MAE con respecto a métodos de cuantificación existentes, se diseñó e implementó un compresor de imágenes modular, en el cual fueron probadas las variantes del VQ-MAE y algoritmos de cuantificación tradicionales.

Abstract

In the field of computing sciences, effective and efficient techniques for image compression are still in development. For example, almost all internet images use some kind of compressing format, making uploading, downloading, storing and memory consumption more efficient.

In this work a new scheme for Vector Quantization (VQ) is proposed that is based on the use of the LBG and Extended Associative Memories algorithms, named as VQ-MAE. This scheme was achieved using one of the four MAE operators — **prom**, **med**, **pmed** and **sum** — as much as in the creation of the new codebook as in the process of searching for the reconstruction vector. Through experimental results the performance of the VQ-MAE scheme is shown with each of these operators; this performance was measured using the objective criteria of the pick signal to noise ratio.

Most of the operators used in the operation of MAE base their function on comparisons and subtraction. The simplicity of these operations allows MAE to offer high-speed processing and low demand for resources (system memory), characteristics that are inherited from the VQ-MAE scheme. The temporal and spatial complexity analyses of VQ-MAE variations are shown.

The results obtained from these experiments demonstrate that the VQ-MAE achieved similar performance with traditional methods in regard to the parameters of the compression ratio and the signal to noise ratio, while in regard to processing speed and consumed resources during operation it exceeds the parameters.

Finally, to compare the performance that VQ-MAE offers in regard to existing methods of quantization, a modular compression scheme of images was designed and implemented, in which the variations of VQ-MAE and traditional algorithms were tested.

The final aim for these processes was the reconstruction of the original image with the best quality possible but with less memory consumption and better processing speed.

Agradecimientos

Durante la formación profesional y académica que me ha brindado y he recibido en esta etapa del posgrado, expreso mi agradecimiento a la **Universidad Tecnológica de la Mixteca,** institución que durante estos últimos años ha forjado mis convicciones académicas. En este tiempo tuve el privilegio de conocer a destacados investigadores, excelentes profesores y personas muy valiosas; a todos ellos gracias por las enseñanzas, consejos y apoyo que me brindaron para culminar esta última fase de forma exitosa.

Particularmente quiero agradecer de manera sincera a mi director de Tesis, el **Dr. Enrique Guzmán Ramírez** por brindarme su tiempo, dedicación, apoyo e invaluable experiencia científica durante el desarrollo de esta tesis. Así también, expresar su valiosa e incondicional amistad que he tenido en más de una década de conocerle y en este último año en el desarrollo de este trabajo. Al **Dr. José Aníbal Arias Aguilar**, **Dr. Antonio Orantes Molina**, **Dr. Ricardo Pérez Águila** y al **Dr. Rosebet Miranda Luna** por el tiempo invertido en la revisión, corrección y comentarios vertidos a este escrito.

Dedicatoria

Este trabajo lo dedico para aquellas personas que en principio me dieron la vida y durante mi crecimiento y formación entregaron todo su amor, esfuerzos y recursos para educarme y ser una persona de provecho. A quienes nunca podré corresponder por todo su apoyo incondicional entregado, ni aún con las mayores riquezas del mundo. A **Celina Jiménez Arellano** y **Eustolio Ramírez Méndez**, mis Padres.

De manera muy especial, hago esta dedicatoria a la persona que me ha acompañado en todo momento esta última década y media y que ha sabido entender, apoyar, alentar y estimular cada uno de mis sueños y metas personales y que se convirtieron también de ella, a **Reyna**, mi Esposa, porque ha sido mi principal motivación para persistir en este logro para beneficio en nuestras vidas.

A quienes he tenido durante cada una de las etapas de mi existir y hasta el día de hoy. Y que me han regalado su amor, cariño y hermandad, **Nestor**, **Jorge**, **Norma**, **Merced, Guillermo** y **Guadalupe**, mis Hermanos.

Índice

Índice de figuras

Índice de tablas

Lista de Acrónimos

VQ	Cuantificación Vectorial (*Vector Quantization*).
LBG	Algoritmo Linde-Buzo-Gray (LBG).
GLA	Algoritmo de Lloyd generalizado (*Generalized Loyd Algorithm*).
ANN	Redes Neuronales Artificiales (*Artificial Neuronal Networks*).
SOM	Mapas auto-organizados (*Self-Organizing Maps*).
CODEBOOK	Libro de Códigos.
CODEWORD	Vectores de Reconstrucción.
KLA	Algoritmo de Aprendizaje de Kohonen (*Kohonen Learning Algorithm*).
SCS	Diseño de Competencias Suaves (*Soft Competition Scheme*).
CNN-ART	Red Neuronal Centroide con Teoría de resonancia Adaptiva (*Centroid Neuronal Network-Adaptive Resonance Theory*).
NN	Red Neuronal (*Neuronal Network*).
MS	Corrimiento Medio (*Mean Shift*).
PCA	Análisis de Componentes Principal (*Principal Component Analysis*).
LOSSLESS	Método de compresión sin pérdida.
LOSSY	Método de compresión con pérdida.
ENTROPIA	Redundancia de datos
FLVQ	Cuantificación Vectorial de Aprendizaje Difuso (*Fuzzy Learning Vector Quantization*).
IAFC	Agrupamiento Difuso Adaptivo Integrado (*Integrated Adaptive Fuzzy Clustering*).
MAE	Memoria Asociativa Extendida.
PROM	Operador Promedio aritmético usado en las MAE.
MED	Operador Mediana usado en las MAE.
PMED	Operador Punto-Medio usado en las MAE.
SUM	Operador Suma usado en las MAE.
VQ-MAE	Cuantificación Vectorial basada en Memorias Asociativas Extendidas.
MAE-codebook	Libro de códigos generado usando el proceso de entrenamiento o aprendizaje de las MAE.
SVH	Sistema Visual Humano.
IGS	Técnica de Cuantificación con escala de Grises Mejorada (*Improved Gray-Scale Quantization*).
DCT	Transformada Discreta del Coseno (*Discrete Cosine Transform*).
DWT	Transformada Discreta Wavelet (*Discrete Wavelet Transform*).
SQ	Cuantificación Escalar (*Scalar Quantization*).
MSE	Error Cuadrático Medio (*Means Square Error*).
MIDTREAD	Cuantificador Uniforme asimétrico.
MIDRISE	Cuantificador uniforme simétrico.
COMPANDING	Proceso conjunto de compresión y expansión en imágenes.
PSRN	Parámetro de Relación Señal a Ruido Pico (*Pick Signal-Noise Ratio*).

BAM	Memoria Asociativa Bidireccional (*Bidirectional Associative Memory*).
ADAM	Memoria Asociativa Distribuida Avanzada (*Advanced Distributed Associative Memory*).

Capítulo 1

Introducción

La Cuantificación Vectorial (VQ, *Vector Quantization*) se ha usado como un método popular y eficiente en el área de compresión con pérdidas de imágenes y voz [22], [43], [19]. En estas áreas, VQ es una técnica que puede producir resultados muy cercanos a los límites teóricos. La técnica más simple y ampliamente usada para diseñar cuantificadores vectoriales es el algoritmo Linde-Buzo-Gray (LBG) creado por Y. Linde *et al.* (1980) [40]. Se trata de un algoritmo iterativo descendente el cual decrece monotónicamente su función de distorsión hacia un mínimo local. Algunas veces, también es referenciado como algoritmo de Lloyd generalizado (GLA, *Generalized Lloyd Algorithm*), debido a que se trata de una generalización de un algoritmo de clasificación propuesto por Lloyd [41].

Un novedoso enfoque en el diseño de esquemas de VQ basados en memorias asociativas o redes neuronales artificiales (ANNs, *Artificial Neuronal Networks*) ha surgido como una alternativa a métodos tradicionales. Dentro de este enfoque, muchos algoritmos VQ han sido propuestos o métodos tradicionales han sido mejorados. Ejemplos de estos son los Mapas Auto-Organizados (SOM, *Self-Organizing Maps*) propuestos por el Prof. Teuvo Kohonen como una red de aprendizaje competitiva ampliamente utilizada en la creación de esquemas VQ [37], [38]. Con base en los SOM, C. Amerijckx *et al.* propusieron un esquema de compresión con pérdida para imágenes digitales [2] y un esquema de compresión de imagen para la compresión sin pérdida [3]. Otro esquema VQ, que incorpora principios estocásticos al algoritmo de aprendizaje de Kohonen, nombrado “Diseño de

Competencia Suave" (SCS, *Soft Competition Scheme*) fue propuesto por Eyal Yair *et al.* [17]. Una propuesta que integra ventajas presentadas por el algoritmo LBG y redes neuronales de aprendizaje competitivo y aplicado a la compresión de imágenes fue presentado por Basil and Jiang [10]. Más recientemente, persiguiendo como objetivo generar un libro de códigos más eficiente, Chin-Chuan Han *et al.* [12] presentaron un algoritmo que combina la teoría de CNN-ART (*Centroid Neuronal Network -Adaptive Resonance Theory*) y un enfoque LBG mejorado. Los avances que el área de las redes neuronales experimenta se ven también reflejados en los esquemas VQ, como lo muestra la propuesta de Yong-Soo & Sung-Ihl, donde presentan una cuantificación vectorial de aprendizaje difuso (FLVQ, *Fuzzy Learning Vector Quantization*) la cual está basada en la fuzzificación de LVQ [62].

Un estudio acerca de esquemas VQ basados en un enfoque neuronal y/o mejoras de algoritmos tradicionales siguiendo este mismo enfoque es presentado en el apartado 2.3.

En el presente trabajo se propone un algoritmo eficiente de búsqueda de alta velocidad aplicado al proceso cuantificación de imágenes con base en Memorias Asociativas Extendidas (MAE). Este nuevo algoritmo reduce significativamente la necesidad computacional sin sacrificar rendimiento.

1.1 Problema a resolver

La VQ es un proceso en el cual los datos son divididos en bloques pequeños, formando vectores, que son codificados secuencialmente. El proceso VQ es dividido en dos fases: diseño del libro de códigos y fase de codificación.

En la fase de diseño del libro de códigos la idea principal es identificar un conjunto de vectores de reconstrucción que conformarán el libro de códigos, es decir, los vectores de reconstrucción son los vectores representativos de la información a codificar. La fase de codificación busca la coincidencia más cercana de cada vector de entrada con los vectores de reconstrucción. La codificación final entonces, se reduce a un simple proceso de listado secuencial de vectores de reconstrucción indexados.

Aunque la VQ con vectores de dimensión grande presenta un mejor desempeño en comparación con aquellos de dimensión pequeña, uno de los problemas más serios para este proceso, especialmente para vectores dimensionalmente grandes, es la alta complejidad computacional en el proceso de búsqueda de la coincidencia más cercana entre los vectores de entrada y los vectores de reconstrucción incluidos en el libro de códigos diseñado; es decir, el tiempo de cómputo requerido para la cuantificación del vector crece exponencialmente con la dimensión del vector. Esto hace a los vectores dimensionalmente grandes inadecuados para la cuantificación vectorial.

Por lo tanto, dado un libro de códigos de *m* vectores de reconstrucción $\{\mathbf{c}_1, \mathbf{c}_2, \cdots, \mathbf{c}_i, \cdots, \mathbf{c}_m\}$, el problema es encontrar el *i*-ésimo elemento que ofrezca la mejor coincidencia con un vector de entrada $\mathbf{x}$, cuando $\mathbf{x}, \mathbf{c}_i \in \mathbf{R}^d \ \forall \ i$ en una forma rápida y eficiente.

Para dar solución al problema planteado, el presente trabajo propone la combinación de dos algoritmos, LBG y Memorias Asociativas Extendidas (MAE), con la finalidad de crear un nuevo esquema de cuantificación vectorial con un enfoque en la compresión de imágenes

estáticas. Este nuevo esquema reduce significativamente la necesidad computacional sin sacrificar el rendimiento.

1.2 Justificación

La cuantificación vectorial es un método usado en la compresión con pérdida de datos, que puede producir resultados muy próximos a los límites teóricos; sin embargo, su desventaja principal es que el proceso de búsqueda, al basar su funcionamiento en un algoritmo de exploración total de igualación, lo hace un proceso lento y de una complejidad computacional considerable.

Como se mencionó en el apartado anterior, el esquema VQ propuesto hace uso del algoritmo LBG y de las MAE. Primero, se aplica la fase de aprendizaje de las MAE, utilizando un operador **prom**, **med, pmed** o **sum**, a un libro de códigos generado mediante el algoritmo LBG; el resultado de esta etapa es una red asociativa cuyo objetivo es establecer una relación entre el conjunto de entrenamiento y el libro de códigos generado por el algoritmo LBG; esta red asociativa es un nuevo libro de códigos (MAE-codebook) usado por el esquema para la cuantificación vectorial propuesto. Segundo, teniendo como elemento central al MAE-codebook, se utiliza la fase de clasificación de las MAE para obtener un proceso de búsqueda rápida, este proceso tiene por función generar eficientemente el conjunto de los índices de clases a los cuales cada vector de entrada pertenece, completando así la cuantificación vectorial.

El proceso de generación del libro de códigos de un cuantificador mediante el algoritmo LBG es la técnica más simple y ampliamente usada. Debido a esto, se ha decidido emplear este algoritmo para generar el libro de códigos inicial que usará el esquema propuesto en este trabajo.

La propuesta de usar Memorias Asociativas Extendidas en este trabajo se fundamenta principalmente en la alta velocidad de procesamiento, baja demanda de recursos, alta inmunidad al ruido, y de que se trata de una técnica usada en la clasificación de patrones que a diferencia de otras técnicas no requiere converger para realizar una clasificación perfecta.

La alta velocidad de procesamiento que una MAE puede generar se debe a que éstas basan su funcionamiento en las operaciones morfológicas de erosión y dilatación, así, la complejidad computacional se ve reducida drásticamente ya que las operaciones utilizadas son muy simples: máximos o mínimos de sumas.

1.3 Objetivo

El objetivo principal de este trabajo es el de diseñar e implementar en un lenguaje de alto nivel (C++ Builder) un esquema de cuantificación de imágenes con base en Memorias Asociativas Extendidas y el algoritmo LBG y su aplicación a la compresión de imágenes.

Para ello se contemplan las siguientes metas particulares:

- Implementar el algoritmo LBG para la generación del libro de códigos inicial.
- Implementar la fase de aprendizaje de las MAE con los operadores **prom**, **med**, **pmed** y **sum**.

- Implementar la fase de clasificación de las MAE con los operadores **prom**, **med**, **pmed** y **sum**.
- Conjuntar las implementaciones anteriores para obtener el esquema de cuantificación propuesto.
- Implementar una cuantificación vectorial con base en el algoritmo LBG.
- Implementar medidas de desempeño que permitan evaluar el rendimiento, con base en los parámetros de relación señal a ruido y relación de compresión, de los algoritmos de cuantificación utilizados en este trabajo.
- Implementar una interfaz de usuario que facilite manipular en forma simple el esquema propuesto, permita visualizar los resultados obtenidos y permita compararlos con esquemas de cuantificación tradicionales.

1.4 Solución propuesta

Como en cualquier esquema de VQ, la propuesta de este trabajo consta de dos procesos: la generación del libro de códigos y la búsqueda del vector de reconstrucción (proceso de codificación o proceso de cuantificación vectorial). En la generación del libro de códigos se aplicará la fase de aprendizaje de la MAE sobre un libro de códigos previamente generado mediante el algoritmo LBG, con la finalidad de crear un libro de códigos nuevo denominado MAE-codebook. En el proceso de codificación, se propone un algoritmo de búsqueda de alta velocidad con base en la fase de clasificación de las MAE, este proceso generará un conjunto de índices de clases donde cada índice tiene una correspondencia con cada vector de entrada.

1.4.1 Generación del libro de códigos usando Memorias Asociativas Extendidas

El proceso de generación del libro de códigos, con base en las MAE y el algoritmo LBG, es fundamental en la obtención del algoritmo de búsqueda rápida, VQ-MAE**,** propuesto en este trabajo. La Figura 1.1 muestra el diagrama de bloques del proceso de generación del libro de códigos.

En este proceso se genera una red asociativa aplicando la fase de aprendizaje de la MAE sobre un libro de códigos generado con el algoritmo LBG y un conjunto de vectores de entrenamiento. Esta red asociativa representa el llamado MAE-codebook y tiene por función establecer una relación entre el conjunto de entrenamiento y el libro de códigos LBG. El MAE-codebook se obtiene en tres fases: Generación del libro de códigos, Determinación del vector de reconstrucción asociado a cada bloque de imagen (conjunto de entrenamiento) usando la distancia Euclidiana y Generación de un nuevo libro de códigos basado en MAE.

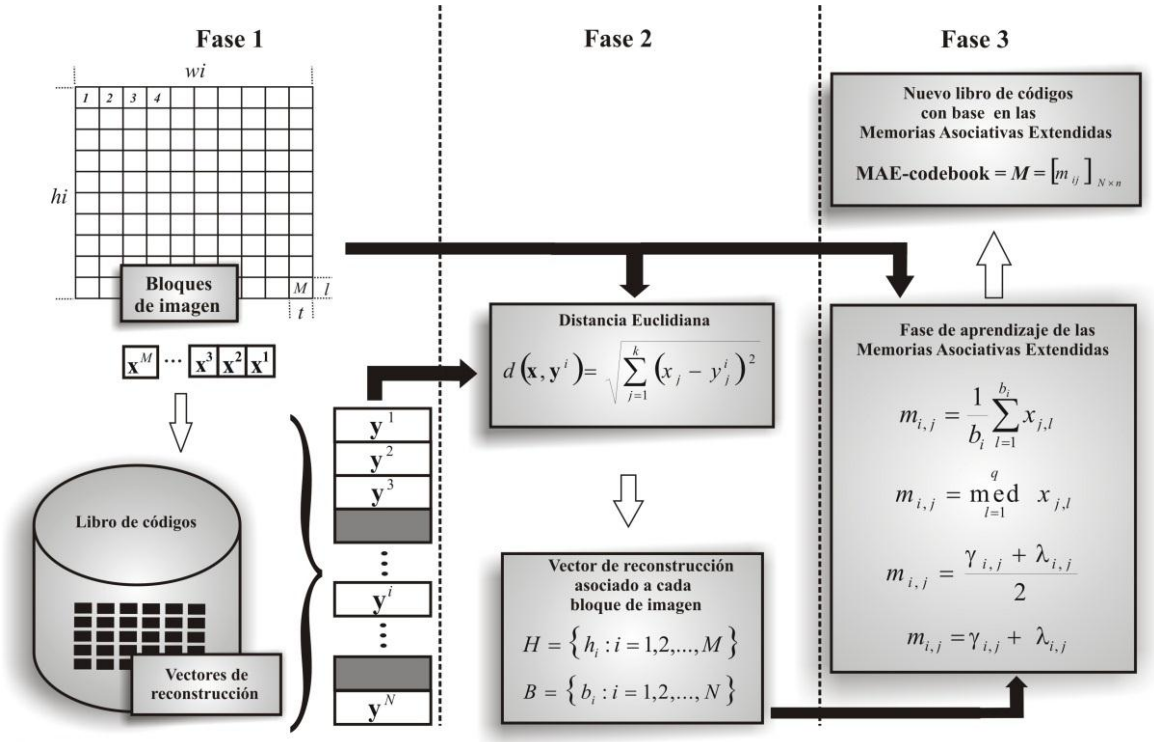

Figura 1.1. Esquema del algoritmo de generación del libro de códigos basado en MAE.

1.4.2 Cuantificación Vectorial VQ-MAE.

En un proceso VQ basado en un criterio de similaridad entre vectores de entrada y vectores de reconstrucción, cada vector de entrada es reemplazado por el índice del vector de reconstrucción que presenta la correspondencia más cercana.

En contraste con el proceso que genera al MAE-Codebook, la cuantificación vectorial usando el algoritmo VQ-MAE es un proceso en demasía simple. Debido a que la generación del MAE-Codebook es realizada usando el proceso de aprendizaje de las MAE, entonces, el VQ-MAE es llevado a cabo usando el proceso de clasificación de las MAE. La Figura 1.2 muestra el diagrama de bloques del algoritmo de búsqueda rápida para VQ con base en las MAE.

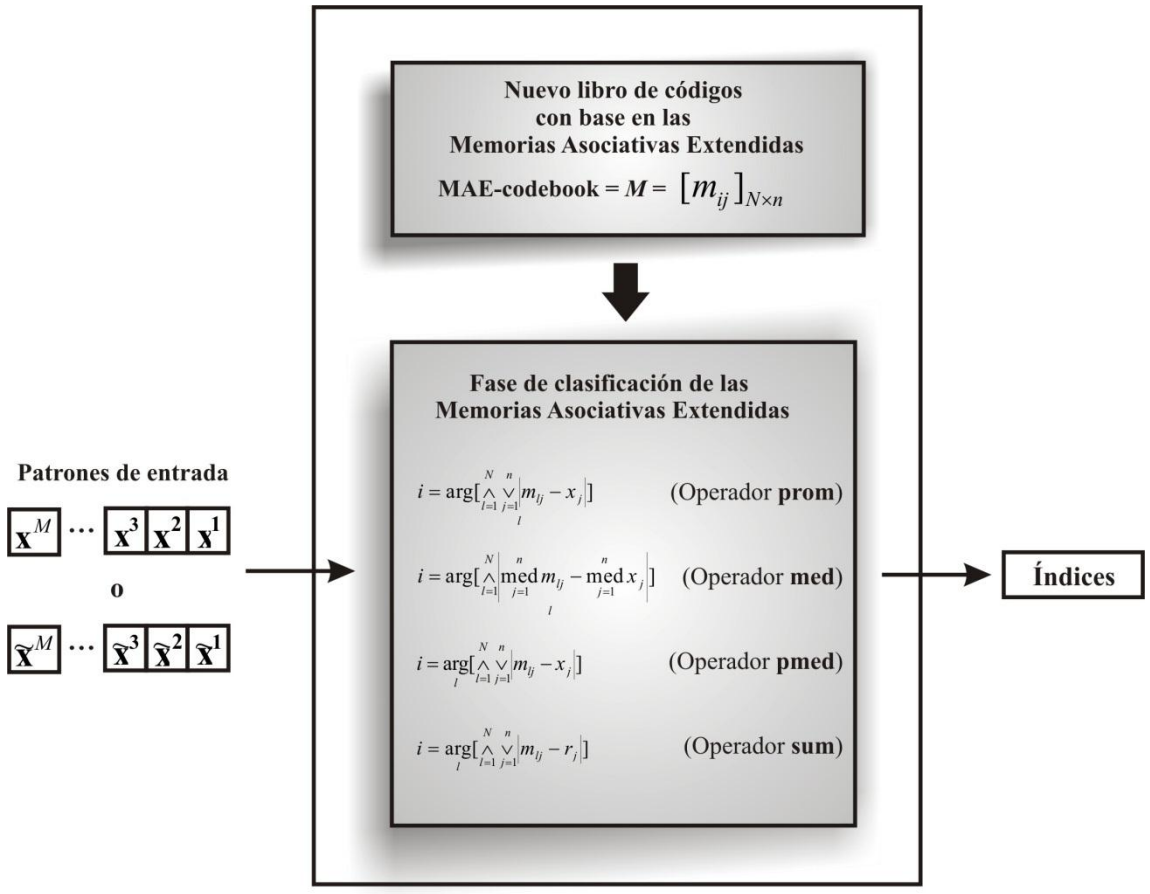

Figura 1.2. Esquema del algoritmo de búsqueda rápida basado en MAE.

1.5 Contribuciones

La principal contribución de este trabajo de tesis es la propuesta de un cuantificador vectorial con base en las Memorias Asociativas Extendidas, VQ-MAE, y su aplicación a la compresión de imágenes. VQ-MAE se mantiene competitivo en los parámetros de relación señal a ruido y relación de compresión con respecto a algoritmos de VQ clásica como el LBG.

1.6 Organización del documento

Seis capítulos forman la estructura de la tesis, los cuales se describen a continuación.

Capítulo 1. Introducción. Este capítulo es un breve resumen del documento, los principales temas tratados en este capítulo son la definición de problema a resolver, la justificación y la solución propuesta.

Capítulo 2. Antecedentes. El contenido inicial de este capítulo versa sobre los fundamentos de la compresión de imágenes haciendo énfasis en la etapa de cuantificación y termina con el obligatorio estado del arte que permite deducir cuál es el estado actual del área de cuantificación vectorial.

Capítulo 3. Memorias asociativas, antecedentes. En este capítulo se incluye un estudio referente a las memorias asociativas, presentando sus aspectos generales y describiendo algunos de los modelos representativos de esta área.

Capítulo 4. Solución propuesta, esquema VQ-MAE. Inicialmente, en este capítulo se presentan los métodos utilizados en la solución propuesta, para posteriormente describir detalladamente el esquema VQ-MAE. El capítulo termina con un análisis de complejidad, temporal y espacial, del VQ-MAE.

Capítulo 5. Resultados experimentales. En este capítulo se muestra el desempeño del VQ-MAE tras ser sometido a una serie de experimentos.

Capítulo 6. Conclusiones y trabajo futuro. Este capítulo contiene las conclusiones que esta investigación ha derivado y propuestas de trabajos que le darán continuidad.

1.7 Artículos publicados

1. Guzmán-Ramírez, E., Ramírez, Miguel A. and Progrebnyak, O. (2011) *"Applied Extended Associative Memories to High-Speed Search Algorithm for Image Quantization"*. Search Algorithms and Applications book, ISBN: 978-953-307-156-5. INTECH Publisher, pp. 151-174.
2. Guzmán-Ramírez, E., Selene Alavarado, Miguel A. Ramírez and Luis A. Rosario (2010). "*An Efficient Morphological Associative Memories Hardware Implementation for Pattern Recognition Applications*". IEEE Computer Society Press, CERMA 2010 Proceedings, IEEE Xplore. pp. 457-462. ISBN 978-0-7695-4204-1.

Capítulo 2

Antecedentes

Este trabajo de tesis presenta un esquema de cuantificación vectorial mediante memorias asociativas. Un cuantificador vectorial es parte fundamental de un sistema de compresión de datos, por lo que en las primeras secciones de este capítulo se introducirán los fundamentos de la compresión de imágenes, en secciones siguientes se profundizará en la etapa de cuantificación para finalizar con la obligatoria sección referente al estado del arte, lo que permitirá determinar cómo se encuentra el tema de la cuantificación vectorial al momento de realizar la presente investigación y cuáles son las tendencias actuales en esta área.

2.1 Fundamentos de la compresión de imágenes

Una imagen es una representación de objetos de diversa índole que estimula el sentido visual de los seres vivos en diferentes maneras. Se puede observar que la palabra clave dentro de la definición de imagen es "representación". Ahora, la representación de dicha información se hace mediante elementos llamados datos. Hay que puntualizar bien la diferencia entre estos dos términos (*datos* e *información*), ya que la mayoría de las veces se utilizan como sinónimos y no lo son [20]. Entonces, los datos son una forma de representar la información. Así, una misma información puede ser representada por distintas cantidades de datos [20]. Por ejemplo una misma noticia de algún acontecimiento, puede ser redactada tanto en una cuartilla como en seis de ellas, siendo que las dos redacciones poseen la idea

principal del evento. Sin embargo, algunas veces, estas representaciones de la información contienen datos repetidos o redundantes, es decir, datos con poca o nula relevancia.

La compresión de datos se puede definir como el proceso de reducir la cantidad de datos necesarios para representar eficazmente una información, en otras palabras, la eliminación de datos sin relevancia o redundantes.

La compresión de imágenes comprende un conjunto de técnicas que se aplican a las imágenes para almacenarlas o transmitirlas de manera eficiente. Según [21], "*es el proceso de reducción del volumen en datos para representar una determinada cantidad de información"*. Es decir, un conjunto de datos puede contener datos redundantes que son de poca relevancia o son datos que se repiten en el conjunto, los cuales si se identifican pueden ser eliminados.

En el caso de las imágenes, se identifican varias formas de representación de la misma, de acuerdo a la forma utilizada se aplican al menos tres maneras posibles de reducir o eliminar el número de datos redundantes: *Eliminación de código redundante, Eliminación de pixeles redundantes y Eliminación de redundancia visual*.

2.1.1 Representación discreta de una imagen

Una *imagen digital* es un arreglo rectangular de puntos, o elementos de fotografía, organizado en *hi* filas y *wi* columnas. La expresión $hi \times wi$ es llamada resolución de la imagen, y los puntos son llamados pixeles. El término resolución es también usado para indicar el número de pixeles por unidad de longitud de la imagen, siendo común utilizar el término *dpi* que significa *puntos por pulgada*. Para el propósito de la compresión de imagen esto es útil para distinguir los siguientes tipos de imágenes: imagen de dos niveles (binaria), en escala de grises ó imagen en tono continuo.

Las imágenes constituyen una distribución espacial de la intensidad de luz de la escena capturada. Matemáticamente hablando, ésta distribución espacial puede ser descrita como una función continua de dos variables espaciales:

$$F(z) = f(x, y) \tag{2.1}$$

Las computadoras no pueden manejar imágenes continuas, solamente arreglos de números digitales. Por lo tanto, se requiere representar a las imágenes como arreglos de puntos de dos dimensiones. Un punto sobre la cuadricula bidimensional es un *pixel*. Un pixel representa la intensidad de luz en la cuadricula correspondiente. En el caso más simple, los pixeles son localizados sobre una cuadricula rectangular (Figura 2.1). La posición del pixel esta dado en la notación común para matrices.

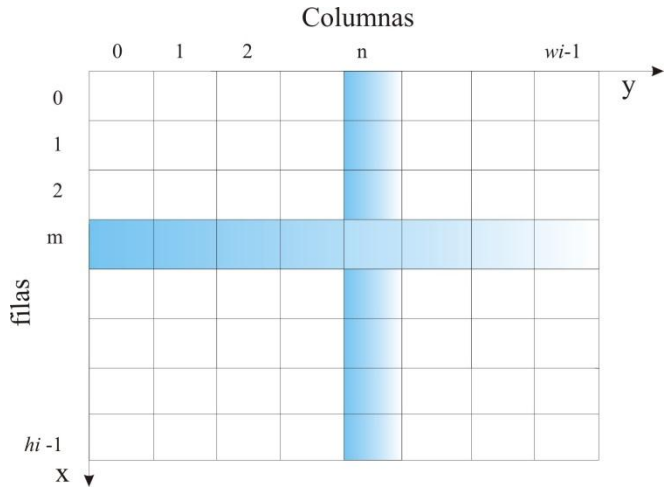

Figura 2.1. Representación discreta de una imagen digital en 2-D por arreglos de puntos discretos sobre una cuadricula rectangular.

2.1.2 Tipos de imágenes

En esta subsección se describen en forma concreta los tipos de imágenes más comunes, ejemplos de ellos son mostrados en la Figura 2.2.

1. Una *imagen de dos niveles (imagen binaria)* es una imagen donde los pixeles pueden tener uno de dos posibles valores, normalmente referido como blanco y negro. Cada pixel en tal imagen es representado por un bit, haciendo a este el tipo de imagen más simple.
2. Una *imagen en escala de grises (monocromática)* es aquella donde un pixel de la misma es representado por *m* bits pudiendo tomar 1 de 2^m tonos de gris posibles (de 0 a 2^m-1). El valor de *m* es normalmente compatible con un tamaño de byte; por ejemplo, 4, 8, 12, 16 ,24 o algún otro múltiplo conveniente de 4 o de 8.
3. Una *imagen en tono continuo* es aquella que puede tener muchos colores similares (o escala de grises). Cuando los pixeles adyacentes son diferentes por sólo una unidad, es difícil para el ojo humano distinguirlos. Por consiguiente, tal imagen podría contener áreas con los colores que parecen variar de forma continua cuando el ojo se mueve a lo largo de la zona. Un pixel en tal imagen es representado por un número grande único o tres componentes (en el caso de una imagen a color). Una imagen con tono continuo es normalmente una imagen natural y es obtenida por tomar una fotografía con una cámara digital, o escaneando una fotografía o una pintura.

Figura 2.2. Clasificación de las imágenes. (a) Imagen a 2 niveles, imagen binaria, (b) Imagen en escala de grises, monocromática, (c) Imagen en tono continuo.

2.1.3 Redundancias en las imágenes

El concepto de redundancia hace referencia a aquellos datos que proporcionan información sin relevancia. Dentro del área de compresión de imágenes, se pueden distinguir y aprovechar tres tipos básicos de redundancias.

2.1.3.1 Redundancia de codificación

Eliminación de código redundante. El código de una imagen representa el cuerpo de la información mediante un conjunto de símbolos. Luego entonces, el objetivo para eliminar código redundante consiste en utilizar el menor número de símbolos para representar la información (imagen). Una palabra de código (símbolo) permite representar cierta cantidad de información. Se produce redundancia cuando las palabras del código presentan distintas probabilidades de ocurrencia, pero posee un tamaño constante. Idealmente, a las palabras de un código se les debiera asignar una longitud proporcional a su frecuencia de ocurrencia. En este caso es usual utilizar las técnicas de compresión por codificación de Huffman y codificación aritmética los cuales utilizan cálculos estadísticos para lograr eliminar este tipo de redundancia y reducir la ocupación original de los datos.

2.1.3.2 Redundancia entre píxeles

Eliminación de pixeles redundantes. La mayoría de las imágenes presentan semejanzas o correlaciones entre sus pixeles. Estas correlaciones se deben a la existencia de relaciones estructurales o geométricas entre los objetos de la imagen, ya que al ser estos objetos reales, tienen propiedades morfológicas de consistencia y continuidad. Estas propiedades determinan correlaciones internas entre diferentes pixeles de una imagen. Estas correlaciones pueden ser: de *primer orden*, debidas a continuidad en los niveles locales de luminosidad; de *segundo orden*, debidas a continuidad en los contornos de los objetos; de *orden superior*, debidas, por ejemplo, a la homogeneidad de los patrones de textura de los objetos presentes.

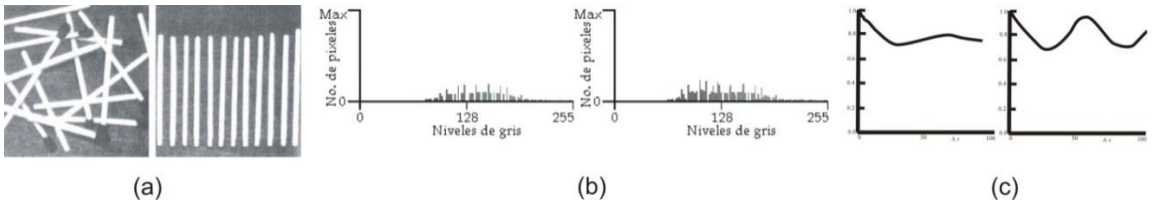

Figura 2.3. Redundancia de datos. (a) Imágenes de fósforos en diferentes posiciones, (b) Histogramas de brillo de las imágenes de fósforos, (c) Coeficientes de auto-correlación normalizados a lo largo de una línea.

La Figura 2.3, tomada de [20], muestra una forma importante de redundancia de los datos (una directamente relaciona con las correlaciones entre los pixeles de una imagen). Puesto que no son completamente aleatorias, la mayoría de sus pixeles se relacionan entre sí y es posible predecir razonablemente el valor de un determinado pixel a partir del valor de sus vecinos o viceversa, la información que aporta individualmente un pixel es relativamente pequeña. La mayor parte de la contribución visual de un único pixel a una imagen es redundante; podría haberse inferido de acuerdo con los valores de sus vecinos. La técnica de compresión Lempel-Ziv implementa algoritmos basados en sustituciones para lograr la eliminación de esta redundancia.

2.1.3.3 Redundancia psicovisual

Eliminación de redundancia visual. Esta redundancia se debe a las características y limitaciones del sistema visual humano (SVH) y que éste responde con diferente sensibilidad a la información visual que recibe, es decir, existe información o cambios en la calidad de la información que no pueden ser distinguidas por el SVH o a la que es menos sensible. Esta información tiene menor importancia relativa que otra en el proceso visual normal. Se dice que esta información es psicovisualmente redundante, y se puede eliminar sin que se altere o afecte significativamente la calidad de la percepción de la imagen. Por tanto, se elimina así lo que se conoce como redundancia visual, produciéndose a la vez la pérdida de ciertas características de la imagen.

A modo de ejemplo, el sistema visual es mucho más receptivo a detalles finos en la señal de luminancia de la imagen (*brillo*), que a detalles en la señal de crominancia (*color*). Otra característica del sistema visual es que es menos sensitivo a señales de alta frecuencia que a señales de baja frecuencia, por lo cual se codifican los coeficientes de alta frecuencia con menos bits y los de baja con mayor número de bits.

La eliminación de la redundancia psicovisual está relacionada con la cuantificación de los datos que permiten representar una cierta imagen, lo que conlleva a una pérdida de datos irreversible. Técnicas de compresión como JPEG, TIFF, EZW o SPIHT hacen uso de la cuantificación, permitiendo usar técnicas que generan pérdida de datos.

2.1.4 Clasificación de los sistemas de compresión de imágenes

Los métodos de compresión se pueden agrupar en dos grandes categorías: métodos sin pérdida de información y métodos con pérdida de información. Su aplicación se realiza de acuerdo al tipo de imagen que se vaya a tratar, es decir, en algunas imágenes no es permisible la pérdida de información en el proceso de compresión como por ejemplo en imágenes médicas, legales o imágenes de astronomía. Mientras que en otras imágenes es posible permitir cierto grado de error, aunque manteniendo la calidad de la imagen, con la

finalidad de optimizar la compresión de imágenes, por ejemplo en imágenes de videoconferencias.

Los métodos de compresión sin pérdida de información (*lossless*) se caracterizan porque la tasa de compresión que proporcionan está limitada por la entropía (redundancia de datos) de la señal original. Entre estas técnicas destacan las que emplean métodos estadísticos, basados en la teoría de Shannon, que permite la compresión sin pérdida. Por ejemplo: codificación de Huffman, codificación aritmética y Lempel-Ziv.

Los métodos de compresión con pérdida de información (*lossy*) logran alcanzar unas tasas de compresión más elevadas a costa de sufrir una pérdida de información sobre la imagen original. Por ejemplo: JPEG, compresión fractal, EZW, SPIHT, etc. Para la compresión de imágenes se emplean métodos *lossy*, ya que se busca alcanzar una tasa de compresión considerable, pero que se adapte a la calidad deseada que la aplicación exige sobre la imagen objeto de compresión.

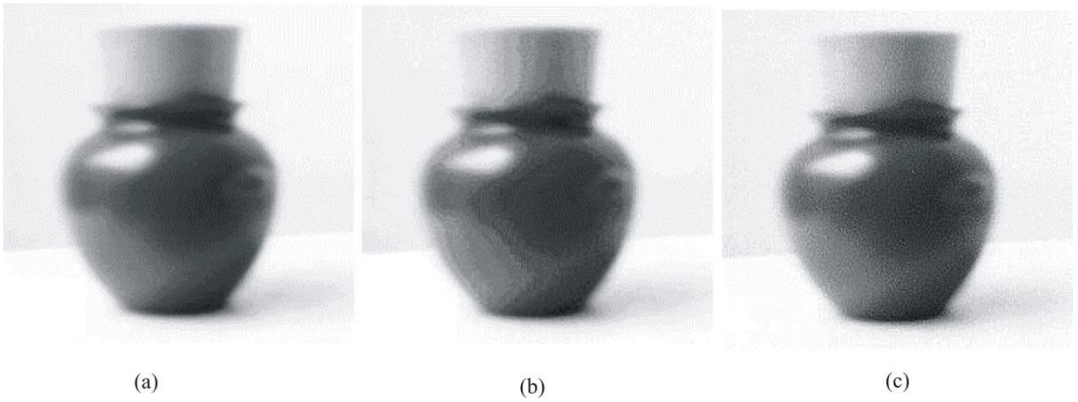

Figura 2.4. (a) Imagen original monocromática con 256 niveles de gris posibles, (b) Imagen con cuantificación uniforme a 16 niveles de gris, (c) Imagen cuantificada a 16 niveles con escala de grises mejorada (IGS).

La Figura 2.4(a) muestra una imagen monocromática con 256 niveles de gris. La Figura 2.4(b) muestra esta misma imagen después de una cuantificación uniforme de 16 niveles posibles. La relación de compresión resultante es de 2:1. Se observa que aparecen unos falsos contornos en las regiones que eran suaves en la imagen original. Este es el efecto visual natural de una representación más basta de los niveles de gris de la imagen. La Figura 2.4(c) ilustra las importantes mejoras que se pueden obtener cuando el proceso de cuantificación aprovecha las particularidades del sistema visual humano. Aunque la relación de compresión resultante de esta segunda cuantificación también es 2:1, se reduce ampliamente la presencia de falsos contornos a costa de la introducción de cierta granulosidad.

El método utilizado para producir este resultado se conoce como Cuantificación con escala de grises Mejorada (IGS, *Improved Gray-Scale Quantization*). Este método reconoce la sensibilidad inherente del ojo a los bordes y los suaviza añadiendo a cada pixel un número pseudo-aleatorio, que se genera a partir de los bits de menor peso de los pixeles vecinos, antes de cuantificar el resultado. Puesto que los bits de menor peso son bastante aleatorios, esto equivale a añadir cierta aleatoriedad, en función de las características locales de la imagen, a los bordes artificiales que aparecerían debido a los falsos contornos.

2.1.5 Compresor de imágenes

La compresión de datos es el proceso mediante el cual se consigue representar cierto volumen de información usando la mínima cantidad de datos. Este tipo de transformaciones son útiles porque permiten ahorrar recursos de almacenamiento y de transmisión. Los compresores de datos son codificadores universales, capaces de comprimir cualquier secuencia de datos si en ella se detecta y elimina redundancia estadística.

La compresión de imágenes es un proceso semejante, pero los datos comprimidos siempre son representaciones digitales de señales bidimensionales. Los compresores de imágenes explotan la redundancia de codificación, entre pixeles y psicovisual. De esta forma, los niveles de compresión que se pueden conseguir aumentan sensiblemente. Incluso comprimidas, las imágenes digitales requieren grandes cantidades de memoria. Debido a este factor, existen básicamente dos tipos de compresores de imágenes: compresores reversibles o compresores sin pérdida (*lossless* compressor) y compresores irreversibles o compresores con pérdidas (*lossy* compressors). Estos últimos, a costa de eliminar la información menos relevante para el observador, alcanzan cotas de compresión muy superiores. El mejor ejemplo lo tenemos en el estándar de compresión JPEG2000.

Como ya se mencionó en el subtema precedente, existen situaciones en la que no es aceptable ningún deterioro de la imagen. Esta restricción viene impuesta porque a priori toda la información contenida, incluso la que es considerada como ruido, se considera imprescindible. Precisamente por la existencia de este ruido los niveles de compresión sin pérdida que se esperan son en general bastante modestos. Típicamente, los ficheros comprimidos son la mitad de largos que el fichero original.

En muchas de las aplicaciones donde se manejen imágenes, la necesidad de compresión proviene principalmente de problemas de transmisión y manipulación de la información, y en menos ocasiones por problemas de almacenamiento. Esta situación es especialmente cierta en el caso de las imágenes. Evidentemente, almacenar el doble de imágenes ya es una gran ventaja, pero donde realmente se obtiene mayor provecho de su compresión es en la transmisión ya que transmitir una imagen comprimida puede dividir el tiempo por la mitad, lo que sin duda constituye un gran ahorro. Esto es especialmente cierto cuando se transmite a través de canales lentos como el internet. La Figura 2.5 muestra un diagrama de bloques de un compresor de imágenes, donde se pueden distinguir 3 etapas: Transformación, Cuantificación y Codificación.

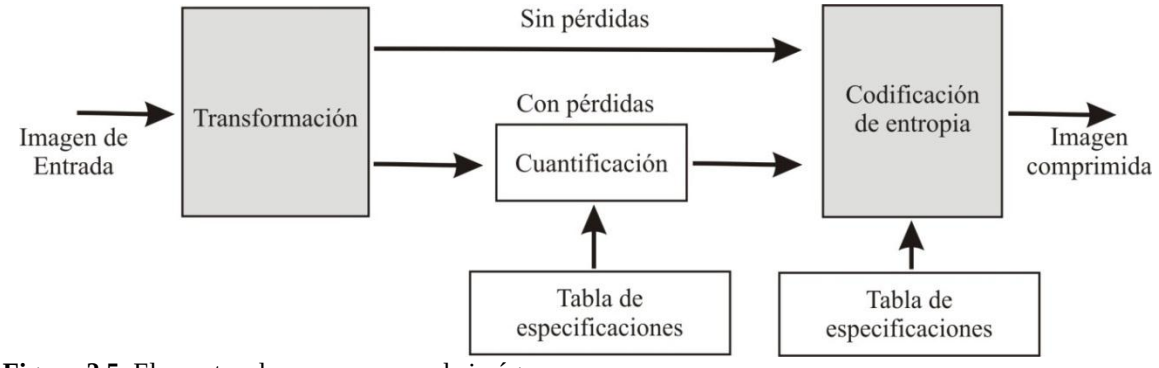

Figura 2.5. Elementos de un compresor de imágenes.

2.1.5.1 Transformación

Dentro de un sistema de compresión de imágenes el primer proceso aplicado a la imagen original es la **transformación**. La transformación tiene como función primordial la reducción de la dependencia entre pixeles de la imagen logrando con ello reducir la redundancia espacial. En esta primera fase no se logra ningún tipo de compresión de los datos procesados, por tanto es un proceso reversible; la función principal de la transformación es cambiar la información de dominio, lo que facilitará la compresión en las etapas siguientes [27].

Se dispone de varias técnicas ó métodos matemáticos que pueden ser usados en esta fase, pero dos de ellas han sido las más utilizadas para este propósito: la transformada discreta del coseno (DCT, *Discrete Cosine Transform*) y la transformada discreta wavelet (DWT, *Discrete Wavelet Transform*).

La DCT pertenece a la familia de las transformadas sinusoidales desarrolladas por Jain A. K. [35]. Su lógica trata de mapear los pixeles uno a uno y presenta las siguientes características: evitar pérdida de información, lo que la hace una técnica reversible, y nula compresión de datos. La transformación usando la técnica DCT es un proceso que se lleva a cabo dividiendo la imagen en bloques, siendo los tamaños de bloques más comúnmente utilizados los de 8x8 y 16x16 [46].

La segunda técnica matemática ampliamente empleada en la compresión de imágenes es la DWT, difiriendo de la DCT en que ésta se aplica a imágenes completas, hecho que hace que esta transformada requiera de una gran cantidad de recursos de memoria durante su operación. Esta técnica procesa la imagen en diferentes etapas que permiten obtener la descomposición *Wavelet* de la imagen.

Algunos aspectos que pueden ser considerados en la elección de la técnica de transformación que se utilizará en la compresión de algún tipo específico de imagen son [27]:

- Los datos en el dominio de la transformada deben de estar separados en componentes con una mínima interdependencia.
- La mayor parte de la energía en los datos transformados debe estar concentrada en un mínimo de valores.
- La transformada debe ser reversible.
- La transformada debe ser computacionalmente tratable (usar un bajo requerimiento de memoria almacenable, usar una precisión aritmética limitada y un bajo número de operaciones aritméticas).

2.1.5.2 Cuantificación

El trabajo presentado en esta investigación versa sobre este tema, por lo que se dedicará el apartado 2.2 para hacer una exhaustiva descripción de esta etapa de un compresor de imágenes.

2.1.5.3 Codificación

La tercera etapa que se desarrolla durante la compresión de imágenes es el *codificador*, éste es un método de codificación de información basado en parámetros estadísticos y tiene la

función de asignar un código a cada símbolo que entrega el cuantificador. La etapa de codificación es capaz de explotar o reducir la redundancia de codificación. En forma general, al codificador se le puede definir como una función fc que asocia un conjunto de símbolos X, a un alfabeto de códigos C, con el propósito de reducir el número de bits utilizados en su representación; la expresión matemática que define a esta función es: $fc : X \rightarrow C.$ Este proceso mantiene una correspondencia uno a uno entre los símbolos fuente y los símbolos del alfabeto de códigos, garantizando con esto la reversibilidad del proceso [27].

Una vez que se tiene la imagen transformada y cuantificada, ésta contiene típicamente una cantidad pequeña de coeficientes distintos de cero y una gran cantidad de coeficientes con valor cero. Una imagen con estas características o estructura se puede comprimir eficientemente de acuerdo al siguiente **proceso**:

- *Reordenamiento de los coeficientes cuantificados.*
- *Codificación por longitud de series o codificación de la carrera (RLE run-length encoding).*
- *Codificación por entropía.*

En base al proceso anterior que involucran 3 pasos; se usan diferentes algoritmos en cada uno de ellos como se mencionan a continuación:

Cuando la imagen es sometida al proceso de transformación, este proceso deja una distribución de coeficientes con una estructura fija. Si se utiliza la DCT, el coeficiente de la esquina superior izquierda representa el promedio de la intensidad del bloque; a partir de este punto, desplazándose hacia la derecha, los coeficientes van representando frecuencias horizontales más altas y con un desplazamiento hacia abajo los coeficientes van representando frecuencias verticales más altas. Ahora bien, si se utiliza la DWT la relación existe entre coeficientes de diferentes niveles, debido a que cada coeficiente tiene relación con los coeficientes de niveles anteriores.

Además, se conoce que usando la DCT y una vez aplicado el proceso de cuantificación, los coeficientes distintos de cero en cada bloque de la imagen generalmente quedarán agrupados en la esquina superior izquierda del mismo, es decir, la DCT concentra su energía en los coeficientes en esta esquina. Conociendo esto, se recomienda *reordenar la distribución de estos coeficientes* mediante la técnica de "exploración" permitiendo adecuar los coeficientes de forma descendente según su probabilidad permitiendo así una codificación más eficiente. Se pueden usar dos esquemas de reorganización: zig-zag o zig-zag modificado si se utilizó la DCT.

Por otro lado, si se utiliza la DWT, las técnicas de *reordenamiento de coeficientes* se apoyan de algoritmos que explotan la correlación entre coeficientes de diferentes niveles; debido a que en la trasformación wavelet cada coeficiente tiene relación con coeficientes de niveles anteriores. Los algoritmos que aprovechan estas propiedades en este paso son: EZW, SPIHT y EBCOT.

El paso definido como *codificación por longitud de series*, es un método muy sencillo, puesto que su funcionamiento se basa en la búsqueda de repeticiones consecutivas de un símbolo para posteriormente sustituirlo por un símbolo especial, el símbolo a codificar y el

número de veces que se repite [18]. Algoritmos que aprovechan esta codificación son: BMP, PCX, JPEG, aunque usan cada uno métodos diferentes de implementación.

Finalmente para la *codificación por entropía* donde entropía indica el límite teórico para la compresión de datos, así como también medirá la cantidad de información media de una fuente. El valor de la entropía está íntimamente ligado con la eficiencia que pueden alcanzar los algoritmos de codificación, entendiendo por eficiencia a las representaciones que requieran menor cantidad de información. Los codificadores por entropía son aquellos que consideran las características estadísticas de la fuente de información a codificar, es decir, toman en cuenta la probabilidad de aparición de los símbolos al momento de asignarles los códigos del alfabeto [27].

La codificación por entropía es un mapeo de un alfabeto fuente de dimensión *m*, representado por $X = \{x_i \mid 0 \le i < m\}$, a un alfabeto de códigos, $\{c_i = C(x_i) \mid 0 \le i < m\}$, con el propósito de minimizar el tamaño del alfabeto fuente. Si la probabilidad del símbolo x_i en el alfabeto fuente $p(x_i)$, la longitud esperada (en bits) de un mensaje compuesto de *n* símbolos es:

$$R = n\sum_i p(x_i)l(C(x_i)) \qquad \textbf{(2.2)}$$

Donde $l(c_i)$ es la longitud (en bits) del código c_i. Para que el proceso sea reversible es necesario que exista un código por cada símbolo fuente, es decir, que se trate de un mapeo uno a uno. Shanon en sus trabajos acerca de la teoría de la información estableció que la entropía de una fuente es el límite inferior del número medio de bits que son necesarios para codificar las salidas de dicha fuente [49]. Existen diversos algoritmos que se aproximan a este límite, entre los que destacan son la codificación Huffman [33] y la codificación aritmética [55], estos algoritmos también son considerados codificadores de longitud variable en virtud de que asignan códigos más pequeños a los símbolos fuente más probables.

2.2 Cuantificación

El procesamiento de imágenes estáticas así como el video y la visión por computadora se han convertido paulatinamente en un área de investigación importante debido a las nuevas tecnologías disponibles. Sus campos de aplicaciones son diversos y se extienden al procesamiento de video, fotografía digital, visión industrial, imágenes médicas, restauración e interpretación de imágenes por satélite o de telescopios, así como el arte por mencionar sólo algunas. Dependiendo del campo de estudio, el término cuantificación puede tomar diferentes connotaciones. En los siguientes párrafos se explicará el término cuantificación cuando se usa en la compresión de imágenes.

2.2.1 Introducción

Como se ha mencionado en párrafos anteriores, la cuantificación es la segunda etapa del proceso final y para ello se espera obtener una alta relación de compresión por medio de la reducción del número de bits necesarios requeridos para almacenar la matriz cuantificada ya que el transformador no fue capaz de reducir el espacio de almacenaje que utiliza la matriz

de pixeles original. Y en esta etapa del proceso, los datos de entrada que recibe el bloque del cuantificador es una matriz de coeficientes transformados. Es de importancia mencionar que este proceso generará pérdidas de información irreversibles, por lo que se tendrán implicaciones de importancia tanto en la compresión como en la calidad de la imagen reconstruida.

Como ejemplo simple de cuantificación se puede mencionar que el redondeo es en sí un tipo elemental de cuantificación sobre un valor o conjunto de valores [23]. Cualquier número real *x* puede ser redondeado al entero más cercano.

2.2.2 Definición de un proceso de cuantificación (compresión de imágenes)

Se entiende por cuantificación, al proceso a través del cual se toman un conjunto de valores continuos y estos valores son aproximados a otro conjunto de valores finitos. Este proceso de cuantificación se puede desarrollar de dos maneras: Sobre coeficientes individuales (cuantificación escalar, SQ, *Scalar Quantization*) y sobre un grupo de coeficientes (cuantificación vectorial, VQ, *Vectorial Quantization*).

Basándose en la definición anterior se puede puntualizar que la función principal de esta fase es lograr una alta relación de compresión a la imagen transformada (matriz de datos de salida), eliminando parte de la precisión de los valores que contiene cada coeficiente transformado en la matriz, logrando por tanto, una reducción de bits que sirven para representar cada coeficiente transformado [44]. El cuantificador está diseñado para disminuir la redundancia psicovisual o también llamada redundancia espacial.

2.2.3 Tipos de cuantificación

Para poder minimizar los efectos negativos del error de cuantificación en el procesamiento de señales digitales (audio y video), se utilizan las distintas técnicas de cuantificación que a continuación se describen:

Cuantificación Uniforme. Es el cuantificador más simple. En los cuantificadores uniformes (o lineales) la distancia entre los niveles de reconstrucción es siempre la misma. No hace ninguna suposición acerca de la naturaleza de la señal a cuantificar, de ahí que no proporcione los mejores resultados. Sin embargo, tiene como ventaja que son los más fáciles y menos costosos de implementar (Figura 2.6).

Todos sus intervalos de cuantificación tienen el mismo tamaño (se suele denotar como Δ); con la salvedad del primer y último intervalo si la señal de entrada no está acotada.

Los valores de reconstrucción están igualmente espaciados, con el mismo tamaño, que las fronteras de decisión Δ.

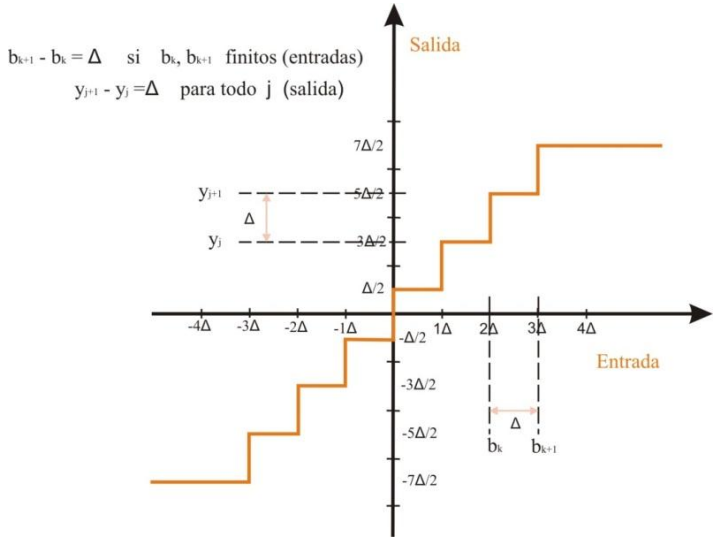

Figura 2.6. Cuantificador escalar uniforme.

De los cuantificadores uniformes existen dos tipos particulares: Primero se tiene al *Cuantificador uniforme asimétrico (Midtread),* donde el valor 0 pertenece al conjunto de salidas. Y normalmente se usa este tipo de cuantificador si el número de intervalos es impar o si necesitamos representar al 0 en los valores de salida.

Y en segundo término se tiene al *Cuantificador uniforme simétrico (Midrise),* donde el 0 no está en el conjunto de salidas. (Ver Figura 2.7).

Así mismo, se encuentran los cuantificadores uniformes sobre fuentes uniformemente distribuidas y sobre fuentes no uniformemente distribuidas.

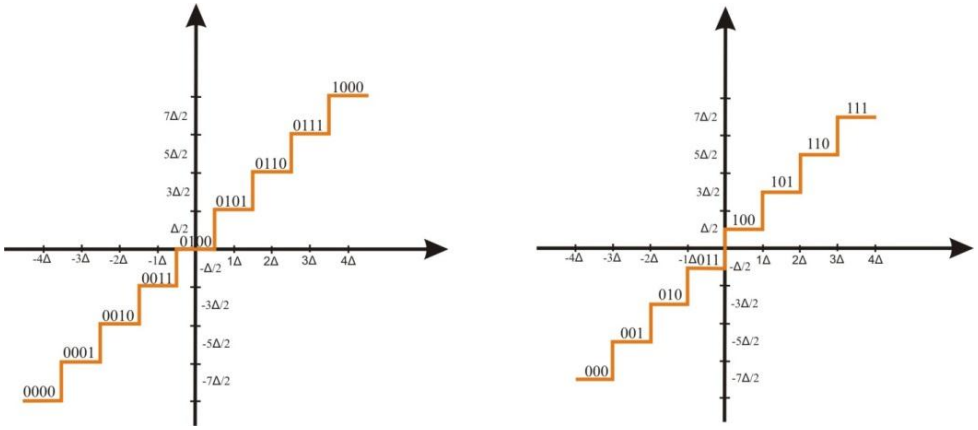

Figura 2.7. Cuantificador escalar uniforme asimétrico y simétrico.

Cuantificación no uniforme. La cuantificación no uniforme (o no lineal) se aplica cuando se procesan señales no homogéneas que se sabe que van a ser más sensibles en una determinada banda concreta de frecuencias (señal de voz). En este caso, lo que se hace es estudiar la entropía de la señal y asignar niveles de cuantificación de manera no uniforme (utilizando un *bit rate* variable), de tal manera que se asigne un mayor número de niveles para aquellos márgenes en que la amplitud cambia más rápidamente (contienen mayor densidad de información, Figura 2.8). Si se ha usado este tipo de cuantificador, deberá usarse el mismo proceso no lineal en la decodificación, para poder reconstruir la señal en forma adecuada.

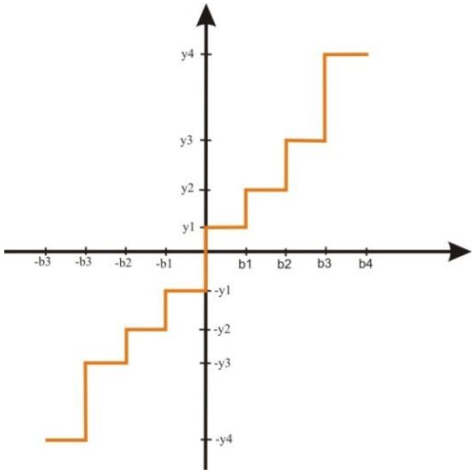

Figura 2.8. Cuantificador escalar no uniforme.

Cuantificación Logarítmica ó escalar. Es un tipo de cuantificación digital en el que se utiliza una tasa de datos constante, pero se diferencia de la cuantificación uniforme en que como paso previo a la cuantificación se hace pasar la señal por un compresor logarítmico.

Cuantificación Vectorial. La cuantificación vectorial puede combinarse con una cuantificación uniforme (utiliza un bit rate constante) o una no uniforme (utiliza un bit rate variable). La particularidad radica, en que, en lugar de cuantificar los elementos en forma individual, estos son cuantificados en bloques. Con ello se logra una cuantificación más eficaz. Cada bloque de muestras es tratado como si fuera un vector, de ahí su nombre de esta tipología.

La cuantificación vectorial es la más eficiente de todas las modalidades de cuantificación en lo referente al error de cuantificación. No obstante, está más predispuesta a verse afectada por errores de transmisión. Otro inconveniente, es que los procesos informáticos para lograr esta codificación resultan muy complejos.

A continuación se ampliarán las explicaciones acerca de cuantificación escalar y vectorial.

2.2.4 Cuantificación escalar

2.2.4.1 Definición

Se entiende por *cuantificación escalar o logarítmica* al proceso de representar los elementos de un conjunto de valores de forma separada. Es decir, el valor discreto asignado a un elemento es independiente del resto de los elementos del conjunto de datos a cuantificar. También se hace referencia al término escalar en base a las (E/S) del cuantificador: si las E/S son escalares entonces se hace referencia a un cuantificador escalar; si las E/S son vectores entonces se trata de un cuantificador vectorial. A continuación exponemos algunos ejemplos que auxiliarán a entender a la cuantificación escalar.

Ejemplo1. En la Tabla 2.1 tenemos como rango de posibles valores de cuantificación a todos los enteros comprendidos entre 0 y 100, se podría diseñar el siguiente esquema de cuantificación:

Tabla 2.1. Equivalencia de valores cuantificados respecto a valores reales en la Cuantificación escalar.

Valor original	2.58	5.56	12.25	23.56	196.58	153.69	52.99	500.52
Valor cuantificado	2	5	12	23	100	100	52	100

Se observa que las apariciones del valor 100 provienen de valores originales distintos, además, una vez obtenido un valor de salida del cuantificador, no hay forma de saber de qué valor provenía.

Ejemplo2. En la Figura 2.9 se observa como una señal ha sido discretizada, tomando muestras a intervalos regulares y posteriormente se han cuantificado sus valores de forma escalar.

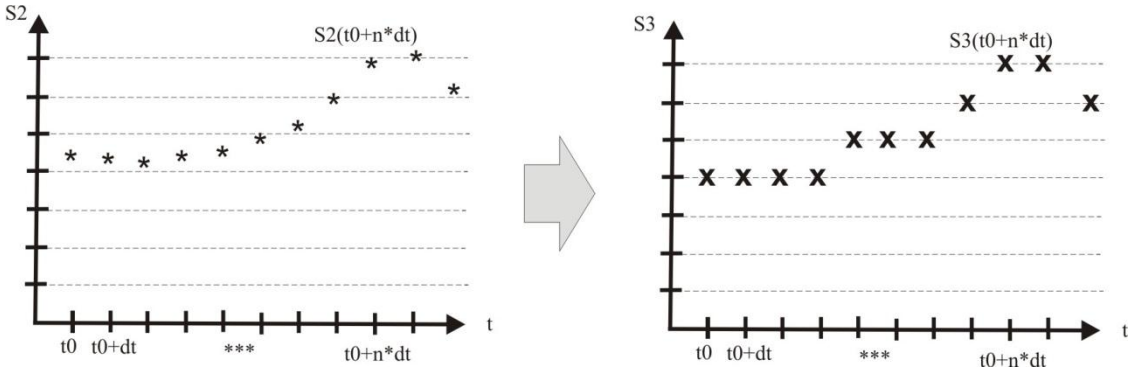

Figura 2.9. Proceso de cuantificación escalar.

Existen varias formas de mapear el rango de valores continuo a un conjunto definido de niveles discretos en función de la naturaleza de la señal a digitalizar.

En primer lugar, el método más simple de codificación por forma de onda se denomina cuantificación escalar y consiste en dividir el rango dinámico en *k* niveles de amplitud uniforme, de modo que el valor de cada muestra queda representado por el cardinal del nivel al que pertenece. Este método recibe el nombre de cuantificación uniforme.

Este tipo de cuantificación digital utiliza una tasa de datos constante, pero que se diferencia de la cuantificación uniforme en que como paso previo a la cuantificación se hace pasar la señal por un compresor logarítmico. Como en la señal resultante la amplitud de la señal sufre variaciones menos abruptas, la posibilidad de que se produzca un ruido de cuantificación grande disminuye y antes de poder reproducir la señal digital, ésta debe pasarse por un expansor que realiza la función inversa al compresor logarítmico. El procedimiento conjunto de compresión y expansión se denomina ***companding***.

Un método alternativo consiste en dividir el rango de valores en intervalos de amplitud no uniforme, de manera que se asigna mayor resolución (pasos de cuantificación más pequeños) a los rangos de amplitud más probables.

En esta cuantificación se tienen pequeños pasos de cuantificación para los valores pequeños de amplitud y pasos de cuantificación grandes para los valores grandes de amplitud, lo que proporcionan mayor resolución en señales débiles al compararse con una cuantificación uniforme de igual bit rate, pero menor resolución en señales de gran amplitud.

Los algoritmos Ley μ y Ley A son ejemplos de cuantificadores escalares ó logarítmicos [11].

2.2.5 Cuantificación vectorial

La *cuantificación vectorial* tiene en cuenta los valores de forma conjunta. Es decir, el valor discreto asignado a cada elemento depende directa o indirectamente de todo el conjunto de datos a cuantificar [45]. Por consiguiente, la cuantificación vectorial, al procesar un conjunto ordenado de escalares, puede ser vista como una generalización de la cuantificación de un escalar. Existen interesantes paralelos entre la cuantificación escalar y la vectorial y muchas de las técnicas de análisis y de diseño son generalizaciones naturales de aquellas ya vistas para el caso escalar. Sin embargo, el salto de una dimensión a varias dimensiones es en realidad un importante cambio de enfoque y descubre gran cantidad de nuevas ideas, conceptos, técnicas y aplicaciones.

Una diferencia importante entre la cuantificación escalar y la vectorial es la siguiente: mientras que la cuantificación escalar es usada principalmente para la conversión A/D, la cuantificación vectorial es usada junto a un procesamiento digital de señales sofisticadas donde en general la señal de entrada ya tiene algún tipo de representación digital y la salida deseada es una versión comprimida de la señal original.

Un vector puede ser usado para describir casi cualquier tipo de patrón, como podría ser un segmento de una señal de voz o de una imagen, simplemente formando un vector de muestras de la forma de onda o de la imagen. Pero también podría formarse un vector con un conjunto de parámetros usados para representar, por ejemplo, la envolvente espectral de una señal de voz.

La cuantificación vectorial puede verse como una forma de reconocimiento de patrones donde un patrón es asociado con un patrón seleccionado de un conjunto almacenado de vectores de reconstrucción. De ahí que la cuantificación vectorial es mucho más que una generalización formal de la cuantificación escalar. En los últimos años se ha convertido en una importante técnica en reconocimiento de voz como, también en compresión de audio y video, y su importancia y sus aplicaciones van en aumento.

La idea base de la *cuantificación vectorial* es la de representar un vector *N*-dimensional que puede tomar cualquier valor continuo en cada una de sus *N* componentes mediante otro vector seleccionado de entre un conjunto finito de vectores *N*-dimensionales.

Es decir, sea **x** un vector *N*-dimensional cuyas componentes son x_k, $1 \le k \le N$ variables aleatorias que toman valores continuos de una determinada magnitud física (por ejemplo, los voltajes correspondientes a *N* sensores de un cierto sistema físico).

La cuantificación vectorial transforma el vector *N*-dimensional **x**, en otro vector *N*-dimensional **y**, mediante una operación de cuantificación:

$$\mathbf{y} = q(\mathbf{x}) \qquad \textbf{(2.3)}$$

Se dice entonces que **x** se cuantifica como **y**, y que el operador *q* es el *operador de cuantificación***.**

El vector **y** no puede tomar cualquier valor, sino que sólo puede tomar un conjunto finito de valores *N*-dimensionales $\mathbf{y}_i$, $1 \le i \le L$.

Al conjunto $\mathbf{Y} = \mathbf{y}_i$, $1 \le i \le L$, compuesto por los posibles valores *N*-dimensionales se le conoce como *libro de códigos***,** ó *codebook* y a los $\mathbf{y}_i$ vectores como los *vectores de reconstrucción*.

Al número de elementos que componen el libro de vectores, que ha de ser un número finito, (en nuestro caso *L*), se le llama tamaño del libro de códigos.

Al proceso de encontrar los *L* vectores que forman el conjunto **Y** de forma óptima en función del universo de posibles valores de **x** se le conoce con el nombre de *entrenamiento o diseño del libro de códigos.*

Básicamente se puede esquematizar el proceso de cuantificación vectorial con la Figura 2.10.

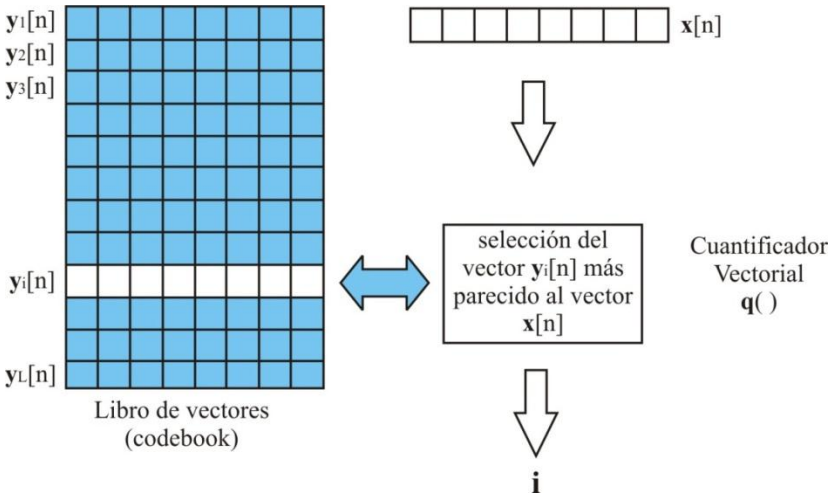

Figura 2.10. Proceso de cuantificación vectorial.

La entrada al cuantificador es el vector **x** que se desea cuantificar; el cuantificador compara este vector con cada uno de los vectores del libro de códigos en base a una *medida de distorsión,* que permite decidir cuál es la distancia del vector **x** con cada vector del libro de códigos. La salida del cuantificador es simplemente, el índice *i,* que corresponde al elemento del libro de códigos más cercano al dado.

Al momento de diseñar un cuantificador vectorial dos aspectos de relevante importancia deben ser tomados en cuenta:

- Diseñar el libro de códigos que represente lo más fiel posible a los vectores que formarán la información a procesar.
- Seleccionar una medida de distorsión adecuada a la aplicación.

2.2.6 Métodos de diseño del Libro de Códigos

Una vez seleccionada la medida de distorsión más adecuada a la aplicación y basada en ésta, se procede a la creación del libro de códigos.

Dado un cuantificador vectorial definido por el conjunto de vectores del libro de códigos $\{y_1, y_2, y_3, ..., y_i, ..., y_k\}$, y la medida de distorsión $d(\mathbf{x}, \mathbf{y})$, se define la *distorsión media* como:

$$D_{media} = \lim_{M \to \infty} \frac{1}{M} \sum_{n=1}^{M} \min_{k=1}^{L} \left[d(x_n, y_k) \right] \quad \textbf{(2.4)}$$

Se dice que un *cuantificador* es *óptimo* cuando el conjunto de vectores del libro de códigos produce el mínimo global en la función de distorsión media, de entre todos los posibles cuantificadores.

Existen algoritmos de diseño del libro de códigos, que si bien no encuentran el cuantificador óptimo para un determinado universo de vectores de entrada, obtienen *cuantificadores sub-óptimos* que corresponden a mínimos locales en la función de distorsión media, pero que en la mayoría de los casos pueden resolver nuestro problema. Para ello se precisa de un conjunto suficientemente amplio de valores de vectores de datos **x**, y haber definido una medida de distorsión adecuada.

2.2.7 Medidas de Distorsión

Cuando un cierto vector **x** se cuantifica como otro vector **y**, indudablemente estamos introduciendo un error, llamado *error de cuantificación*. Hemos de definir una función que nos permita obtener dicho error. A esta función se le llama *medida de distorsión*.

Así, definimos la medida de distorsión como:

$$d(\mathbf{x}, \mathbf{y}) \quad \textbf{(2.5)}$$

donde **x** es el vector a cuantificar, e **y** es una posible cuantificación de **x**.

El proceso de cuantificación consiste en seleccionar del libro de códigos aquel vector **y**, que produzca el menor error de cuantificación. Vemos por tanto la importancia que tiene la medida de distorsión en el diseño de cualquier cuantificador vectorial.

La selección de la medida de distorsión depende por supuesto de la naturaleza de las componentes de los vectores a cuantificar. Para que una medida de distorsión sea útil ha de ser posible su cálculo, y además ha de ser relevante a los datos que maneja.

Existen múltiples medidas de distorsión, y no es fácil decidir la más adecuada a un problema específico. Medidas de distorsión de uso común son las siguientes:

Error Cuadrático medio (MSE). Es la medida de distorsión más frecuentemente utilizada, debido a su simplicidad de cálculo. Se define como:

$$d(\mathbf{x}, \mathbf{y}) = \frac{1}{N} (\mathbf{x} - \mathbf{y})^T (\mathbf{x} - \mathbf{y}) = \frac{1}{N} \sum_{k=1}^{N} [x_k - y_k] \quad \textbf{(2.6)}$$

Error Cuadrático medio ponderado. Es la medida anterior, todos los componentes contribuyen en igual medida al valor final de la distorsión. A veces es conveniente que esto no sea así, posiblemente porque ciertos componentes son más “fiables” que otros. Para solventar este problema se utiliza la ponderación de los diferentes componentes:

$$d(\mathbf{x},\mathbf{y}) = \frac{1}{N}(\mathbf{x}-\mathbf{y})^T \boldsymbol{W} (\mathbf{x}-\mathbf{y}) \quad \textbf{(2.7)}$$

donde $\boldsymbol{W}$ es la matriz de ponderación de componentes. Una elección para $\boldsymbol{W}$ muy común es la inversa de la *matriz de covarianza* del vector aleatorio **x.**

$$\boldsymbol{W} = \Gamma^{-1} \quad \textbf{(2.8)}$$

Donde:

$$\Gamma = \mathbf{E}[(\mathbf{x}-\mathbf{x})(\mathbf{x}-\mathbf{x})^T] \quad \textbf{(2.9)}$$

En ese caso, la medida de distorsión se conoce como la *distancia de Mahalanobis*, y se calcula como:

$$d(\mathbf{x},\mathbf{y}) = \frac{1}{N}(\mathbf{x}-\mathbf{y})^T \Gamma^{-1} (\mathbf{x}-\mathbf{y}) \quad \textbf{(2.10)}$$

Si **x** y **y** son incorreladas, esta medida de distorsión coincide con la distorsión cuadrática media. La medida de *Mahalanobis* es la óptima, según el criterio de *Bayes*, para clasificar vectores distribuidos de acuerdo a una función de densidad de probabilidad gaussiana.

Relación señal a ruido (PSNR). Definida por la siguiente expresión:

$$d(\mathbf{x},\mathbf{y}) = 10\log_{10}\left(\frac{(2^n-1)}{\mathbf{MSE}}\right) \quad \textbf{(2.11)}$$

Medidas basadas en predicción lineal. Se basan en utilizar como componentes de cada vector **x**, los coeficientes $a[k]$ que son el resultado de minimizar la energía del error de predicción lineal. Para un estudio en profundidad de la predicción lineal, consultar [45]. Entre estas, destaca la distancia llamada de *Itakura-Saito* [34], por su extendida utilización, y por su simplicidad de cálculo.

Medidas de distorsión con motivación perceptual. Se basan en utilizar medidas de distorsión relacionadas con el comportamiento humano. Por ejemplo, en cuantificación de señal de voz, se realiza una transformación de los vectores de características de manera que representen señales muy precisas en la baja frecuencia (la que el ser humano percibe mejor), y poco precisas en la alta frecuencia. Para un estudio detallado de dichas medidas, consultar [42].

2.3 Estado del arte

La Cuantificación Vectorial (VQ, *Vector Quantization*) se ha usado como un método popular y eficiente en el área de compresión con pérdidas de imágenes y voz [22], [43], [19]. En estas áreas, VQ es una técnica que puede producir resultados muy cercanos a los límites teóricos. La técnica más simple y ampliamente usada para diseñar cuantificadores vectoriales es el algoritmo Linde-Buzo-Gray (LBG) creado por [40]. Se trata de un

algoritmo iterativo descendente el cual decrece monotónicamente su función de distorsión hacia un mínimo local. Algunas veces, también es referenciado como algoritmo de Lloyd generalizado (GLA, *Generalized Lloyd Algorithm*), debido a que se trata de una generalización de un algoritmo de clasificación propuesto por [41].

Un novedoso enfoque en el diseño de esquemas de VQ basados en memorias asociativas o redes neuronales artificiales (ANNs, *Artificial Neuronal Networks*) ha surgido como una alternativa a métodos tradicionales. Dentro de este enfoque, muchos algoritmos VQ han sido propuestos por lo que se hará mención de algunos de ellos.

Los Mapas Auto-Organizados (SOM, *Self-Organizing Maps*) es una red de aprendizaje competitiva desarrollada por el Prof. Teuvo Kohonen a inicio de la década de los 80´s [37], [38], que ha sido utilizado con mucho éxito en la creación de nuevos esquemas para VQ. [2] propusieron un esquema de compresión con pérdida para imágenes digitales utilizando el algoritmo de la red neuronal de Kohonen. Los SOM fueron aplicados en las etapas de cuantificación y codificación del compresor de imágenes. En la etapa de cuantificación, el algoritmo SOM crea una correspondencia entre el espacio de entrada de estímulo y el espacio de salida derivado y este a su vez constituido por los elementos del **libro de códigos** (*codebook*) llamados **vectores de reconstrucción** (*codewords*) ó **neuronas** usando la distancia euclidiana. Después del aprendizaje de la red, estos elementos del libro de códigos aproximan los vectores en el espacio de entrada de la mejor forma posible. En la etapa de codificación de entropía, un codificador de entropía diferencial utiliza la propiedad topológica de preservación de las SOMs, como resultado del proceso de aprendizaje y la hipótesis de que los bloques consecutivos en la imagen son a menudo similares. En [3], los mismos autores propusieron un esquema de compresión de imagen para la compresión sin pérdida utilizando SOMs y los mismos principios.

[17], proporciona un análisis de convergencia para el algoritmo de aprendizaje de Kohonen (KLA, *Kohonen Learning Algorithm*) con respecto al criterio de optimización para un esquema VQ e introduce una técnica de relajación estocástica que produce el mínimo global, pero que resulta computacionalmente costosa. Al incorporar los principios del enfoque estocástico dentro del KLA, un nuevo algoritmo de diseño VQ determinístico fue introducido el cual fue nombrado “Diseño de Competencia Suave” (SCS, *Soft Competition Scheme*). Este algoritmo es un método *on-line* que actualiza el vector de reconstrucción mediante una competencia suave con cada vector de entrenamiento.

Un nuevo esquema de VQ basado en redes neuronales de aprendizaje competitivo y cuantificación Vectorial LBG fue presentado por [10]. El algoritmo integra ventajas presentadas en ambos cuantificadores vectoriales, LBG y redes neuronales de aprendizaje competitivo, para lograr un rendimiento mejorado en comparación con sus formas existentes tradicionales cuando éste es aplicado a la compresión de imagen.

[12] propusieron un enfoque hibrido para VQ con el objeto de obtener un mejor libro de códigos. Este libro de códigos es modificado y mejorado basado en la teoría de CNN-ART (*Centroid Neuronal Network -Adaptive Resonance Theory*) y el enfoque mejorado LBG. Se utilizan en varias ocasiones tres módulos, una red neuronal NN (*Neuronal Network*), basada en agrupamientos, un corrimiento medio MS (*Mean Shift*) basado en refinamiento y un análisis de componentes principal (PCA, *Principal Component Analysis*) basado en asignación de neuronas (*codewords*). Básicamente, el módulo de asignación de neuronas genera un nuevo libro de códigos inicial que reemplaza el libro de códigos utilizado durante

la iteración. La NN basada en agrupamientos, agrupa el vector de entrenamiento usando un enfoque de aprendizaje competitivo. Los resultados agrupados son refinados usando la operación de corrimiento mínimo.

Una cuantificación vectorial de aprendizaje difuso (FLVQ, *Fuzzy Learning Vector Quantization*) la cual está basada en la fuzzificación de LVQ fue propuesto por [62]. El FLVQ propuesto utiliza un porcentaje de aprendizaje difuso que dependiendo de si la clasificación es correcta o no. Cuando la clasificación es correcta, se usa la combinación de una función de la distancia entre el vector de entrada y los prototipos de clases y una función del número de iteraciones como un porcentaje de aprendizaje difuso. Por otro lado, cuando la clasificación es incorrecta, usa la combinación del valor de pertenencia difusa y una función del número de iteraciones como una tasa de aprendizaje difuso. El FLVQ propuesto está integrada dentro de un IAFC *(Integrated Adaptive Fuzzy Clustering)* supervisado, llamado "red neuronal 5". El conjunto de datos del iris fue utilizado para comparar el rendimiento del IAFC supervisado red neuronal 5 con aquellos del algoritmo LVQ y la red neuronal de propagación inversa.

Se ha observado en [13] que en la mayoría de los VQ en una red neuronal aplicados a almacenamiento y transmisión de señales de voz, video o aplicaciones multimedia utilizan técnicas de predicción no lineal en sus simulaciones, y además, presentan inconvenientes respecto a sus parámetros implementados usando diferentes algoritmos aplicables al diseño del cuantificador vectorial. Plantea también la existencia y clasificación de esquemas de VQ óptimos y sub-óptimos y los describe con base en los resultados obtenidos respecto al equilibrio entre desempeño y complejidad del esquema implementado.

En [30] se plantea el uso exitoso de un algoritmo genético (GA, *Genetic Algorithm*) combinado con la técnica de recocido simulado (SA, *Simulated Annealing*) para obtener un mejor libro de códigos utilizando el menor tiempo de uso de recursos del CPU para un VQ llamado GSAVQ (*Genetic Simulated Annealing Vector Quantization*) y a su vez, comparado con métodos de diseño de libro de códigos convencionales LBG (*Conventional Codebook Design Algorithm*). Esta técnica hibrida arroja en sus resultados experimentales un consumo del 71 - 87 % de uso de CPU comparado con el LBG tradicional, al realizar operaciones de mutación para obtener el libro de códigos óptimo y obtiene también mejoras en el parámetro PSNR con valores de 1.1 - 2.1 dB usando diferentes tamaños de libros de códigos.

En [31] se propuso un método para el diseño del libro de códigos para un VQ llamado Algoritmo Descendente Máximo (MD, *Maximum Descent)*, que comparado con el algoritmo tradicional de Lloyd generalizado (GLA, *Generalized Lloyd Algorithm*). El algoritmo MD conserva mejor desempeño con mucho menor tiempo de cálculo. Sin embargo, buscar el hiperplano para el particionamiento óptimo de un conjunto multidimensional es un problema difícil en este algoritmo. Existen tres técnicas de partición para el método MD y por consiguiente se ha planteado una nueva técnicas de particionamiento basada en el enfoque de Búsqueda Tabú (TS, *Tabu Search*) y se presenta para utilizarse en el algoritmo MD. Los resultados experimentales muestran que el algoritmo MD basado en búsqueda tabú produce un mejor libro de códigos respecto al algoritmo MD convencional.

En [28] se propone una exposición teórica sobre la combinación de dos algoritmos para la creación de un nuevo esquema de VQ. Primero se obtiene una red asociativa aplicando el

algoritmo de aprendizaje para el análisis de datos multivariados (LAMDA, *Learning Algorithm for Multivariate Data Analysis*) a un libro de códigos generado por medio del algoritmo LBG tradicional. El propósito de esta red es establecer una relación entre el conjunto de entrenamiento y el libro de códigos generado por el algoritmo LBG; esta red asociativa será el nuevo libro de códigos LAMDA-codebook que es usado en el nuevo esquema propuesto (VQ-LAMDA). Segundo, considerando a LAMDA-codebook como el elemento central se usa una fase de clasificación de la metodología LAMDA para obtener un proceso de búsqueda rápida, la función de este proceso es generar el conjunto de índices de clases a la cual cada vector de entrada le corresponde, completando así la cuantificación vectorial donde este esquema presenta un balance en el tiempo de procesado del algoritmo y la calidad de la imagen procesada.

A lo largo de los años se han venido proponiendo nuevas variantes sobre los esquemas planteados sobre VQ. Aquí en [7] se propuso una mejora en el algoritmo LBG tradicional donde la idea básica se centra en la clasificación de patrones de entrada basados en la división de espacio con un umbral de distancia variable. En primer lugar, se establece el factor del valor de ajuste de la distancia y el umbral de cifrado de la distancia inicial, entonces, los vectores de entrada se clasifican en celdas de correspondencia para ser nuevos vectores de agrupamiento. Después de eso, se seleccionan los vectores de agrupamiento que tienen menos vectores de entrada en sus células que el número del vector promedio y elimina esas células. En tercer lugar, actualiza el conjunto de vectores de entrada y disminuye el umbral de distancia, de esta manera continúa a la siguiente iteración hasta que el número de vectores de agrupamiento seleccionados cumplan el requisito del algoritmo. Finalmente, se considera a estos vectores agrupados como el libro de códigos inicial para el algoritmo LBG. Los resultados en la simulación muestras que la reducción de los tiempos de las iteraciones excepcional y el efecto de las imágenes reconstruidas mejorada.

Capítulo 3

Memorias asociativas, antecedentes

Este capítulo tiene por finalidad introducir al lector al tema de las memorias asociativas, por lo que se presentan los conceptos básicos relacionados con este tema y una breve descripción de algunos de los modelos más destacados.

3.1 Memorias asociativas, conceptos generales

Los seres humanos tienen la capacidad de reconocer los rostros de una persona, no obstante que sólo observe una parte del rostro, o que la persona se haya colocado peluca o lentes, o se haya quitado la barba o el bigote, por mencionar algunos ejemplos.

En relación con este tipo de habilidades, se afirma que la mente humana tiene características asociativas; es decir, que los seres humanos poseemos memoria asociativa, dado que podemos recordar mediante asociaciones lo que hemos aprendido: objetos, seres vivos, conceptos e ideas abstractas, a pesar de que éstas estén contaminadas con "ruido" (como en el caso de los rostros con lentes o sin barba, o las melodías incompletas [59]).

El concepto de memoria puede ser entendido de diferentes maneras. Generalmente, involucra un *mecanismo de almacenamiento* el cual utiliza un *medio* de almacenamiento. La memoria asociativa parece ser una de las funciones primordiales del cerebro. Nosotros asociamos fácilmente la cara de un amigo con su nombre, o un nombre con un número de teléfono.

Las memorias asociativas son una de las redes neuronales artificiales más importantes con un amplio rango de aplicaciones en áreas tales como memorias de acceso por contenido, reconocimiento de patrones y control inteligente. Una memoria asociativa puede almacenar información y puede recuperarla cuando sea necesario. Puede recuperar dicha información basándose en el conocimiento de parte de ésta (clave). El patrón clave puede ser una versión con ruido de un patrón memorizado, es decir, que difiere de él en pocas componentes.

Una memoria asociativa es un sistema de entrada y salida, cuyo propósito fundamental es recuperar patrones completos a partir de patrones de entrada que pueden estar alterados con ruido aditivo, sustractivo o combinado.

Figura 3.1. Esquema de una memoria asociativa.

En la Figura 3.1 se observa el esquema general de una memoria asociativa, donde **x** y **y** son el vector o patrón de entrada y salida respectivamente.

Cada uno de los patrones de entrada forma una asociación con el correspondiente patrón de salida. La notación es similar a una pareja ordenada; por ejemplo, los patrones **x** y **y** de la Figura 3.1 forman la asociación $(\mathbf{x}, \mathbf{y})$.

En este trabajo, para denotar un patrón de entrada o de salida se denotarán con las letras negrillas, **x** y **y**, y se usarán números naturales como superíndices para efectos de discriminación simbólica. A un patrón de entrada $\mathbf{x}^1$ le corresponderá un patrón de salida $\mathbf{y}^1$, y ambos formarán la asociación $(\mathbf{x}^1, \mathbf{y}^1)$.

Una memoria asociativa ***M*** se representa por una matriz cuya componente *ij*-ésima es m_{ij}, la matriz ***M*** se genera a partir de un conjunto finito de asociaciones conocidas de antemano: éste es el *conjunto fundamental de asociaciones*, o simplemente *conjunto fundamental*. Sea *p* la cardinalidad del conjunto fundamental y μ un índice, el conjunto fundamental se representa de la siguiente manera:

$$(\mathbf{x}^{\mu}, \mathbf{y}^{\mu}) \mid \mu = 1,2,...,p \qquad \textbf{(3.1)}$$

A los patrones que conforman las asociaciones del conjunto fundamental, se les llama *patrones fundamentales*. La naturaleza del conjunto fundamental proporciona un importante criterio para clasificar las memorias asociativas. Si se cumple que $\mathbf{x}^{\mu} = \mathbf{y}^{\mu}\ \forall \mu \in \{1,2,...,p\}$, se dice que la memoria es *autoasociativa*. Por otro lado, si $\exists\ \mu \in \{1,2,\ldots,p\}$ para el que se cumple que $\mathbf{x}^{\mu} \neq \mathbf{y}^{\mu}$ se dice que la memoria es *heteroasociativa*.

Es posible que los patrones fundamentales sean alterados con diferentes tipos de ruido. Para diferenciar un patrón alterado del correspondiente patrón fundamental, en este trabajo usaremos el tilde en la parte superior; así el patrón $\tilde{\mathbf{x}}^k$ es una versión alterada del patrón fundamental $\mathbf{x}^k$.

Si al presentarse a la memoria $\boldsymbol{M}$ un patrón $\tilde{\mathbf{x}}^w$ como entrada $(w \in \{1,2,...,p\})$, $\boldsymbol{M}$ responde con el correspondiente patrón fundamental de salida $\mathbf{y}^w$, se dice que la recuperación es perfecta. Una *memoria perfecta* es aquella que realiza recuperaciones perfectas para todos los patrones fundamentales.

En las ciencias de la computación es de particular interés crear modelos matemáticos que se comporten como memorias asociativas y, con base en estos modelos, crear, diseñar y operar sistemas (software o hardware) que sean capaces de aprender y recordar objetos, seres vivos, conceptos e ideas abstractas [36].

En las memorias asociativas y su aplicación al reconocimiento de patrones hay dos fases claramente distinguibles [59], [15], [5]:

1. Fase de *aprendizaje* (generación de la memoria asociativa).
2. Fase de *reconocimiento* (operación de la memoria asociativa).

Se considerará a *A* como el conjunto de valores que pueden tomar los elementos de los patrones de entrada y salida.

3.1.1 Fase de aprendizaje

La fase de aprendizaje consiste en encontrar los operadores adecuados y una manera de generar una matriz $\boldsymbol{M}$ que almacene las *p* asociaciones del conjunto fundamental $\{(\mathbf{x}^\mu, \mathbf{y}^\mu) \mid \mu = 1,2,\ldots, p\}$, donde $\mathbf{x}^\mu \in A^n$ y $\mathbf{y}^\mu \in A^n \ \forall \mu \in \{1,2,...,p\}$.

3.1.2 Fase de recuperación

La fase de recuperación consiste en hallar los operadores adecuados y las condiciones suficientes para obtener el patrón fundamental de salida $\mathbf{y}^\mu$, cuando se opera la memoria $\boldsymbol{M}$ con el patrón fundamental de entrada $\mathbf{x}^\mu$. Lo anterior para todos los elementos del conjunto fundamental [60], [47].

3.2 Evolución de las memorias asociativas

3.2.1 Lernmatrix

La Lernmatrix es una memoria heteroasociativa que puede funcionar como un clasificador de patrones binarios si se escoge de forma adecuada los patrones de salida, es un sistema de entrada y salida que al operar acepta como entrada un patrón binario $\mathbf{x}^\mu \in A^n$, $A = \{0,1\}$ y produce como salida la clase $\mathbf{y}^n \in A^p$ que le corresponde (de entre *p* clases diferentes), la memoria es codificada con un método simple, que para representar la clase $k \in \{1,2,...,p\}$, se asignan a las componentes del vector de salida $\mathbf{y}^\mu$ los siguientes valores: $y_k^\mu = 1$ y $y_j^\mu = 0$ para $j = 1,2,...,k-1,k+1,...,p$ [52], [56].

En la Tabla 3.1 se esquematiza la fase de aprendizaje para la Lernmatrix de Steinbuch, al incorporar la pareja de patrones $(\mathbf{x}^\mu, \mathbf{y}^\mu) \in A^n \times A^p$.

Tabla 3.1 Fase de aprendizaje de la Lernmatrix

	x_1^μ	x_2^μ	$\cdots$	x_j^μ	$\cdots$	x_n^μ
y_1^μ	m_{11}	m_{12}	$\cdots$	m_{1j}	$\cdots$	m_{1n}
y_2^μ	m_{21}	m_{22}	$\cdots$	m_{2j}	$\cdots$	m_{2n}
$\vdots$	$\vdots$	$\vdots$		$\vdots$		$\vdots$
y_i^μ	m_{i1}	m_{i2}	$\cdots$	m_{ij}	$\cdots$	m_{in}
$\vdots$	$\vdots$	$\vdots$		$\vdots$		$\vdots$
y_p^μ	m_{p1}	m_{p2}	$\cdots$	m_{pj}	$\cdots$	m_{pn}

Cada uno de los componentes m_{ij} de $\boldsymbol{M}$, la Lernmatrix de Steinbuch, tiene valor cero al inicio, y se actualiza de acuerdo con la regla $m_{ij} + \Delta m_{ij}$, donde:

$$\Delta m_{ij} = \begin{cases} +\xi \ \text{ si } x_j^\mu = 1 = y_i^\mu \\ -\xi \ \text{ si } x_j^\mu = 0 \quad \text{y} \quad y_i^\mu = 1 \\ 0 \quad \textit{en otro caso} \end{cases} \tag{3.2}$$

Siendo ξ una constante positiva escogida previamente.

La fase de recuperación consiste en encontrar la clase a la que pertenece un vector de entrada $\mathbf{x}^\omega \in A^n$ dado. Encontrar la clase significa obtener las coordenadas del vector $\mathbf{y}^\omega \in A^p$ que le corresponde al patrón $\mathbf{x}^\omega$ en virtud del método de construcción de los vectores $\mathbf{y}^\omega$ la clase debería obtenerse sin ambigüedad [56].

La *i*-ésima coordenada y_i^ω del vector de clase $\mathbf{y}^\omega \in A^p$ se obtiene como lo indica la siguiente expresión, donde $\vee$ es el operador máximo:

$$y_i^\omega = \begin{cases} 1 \ \text{ si } \ \sum_{\text{j=1}}^{\text{n}} m_{ij} \cdot x_j^\omega = \bigvee_{h=1}^{p} \left(\sum_{j=1}^{n} m_{hj} \cdot x_j^\omega \right) \\ 0 \qquad\qquad\qquad \textit{en otro caso} \end{cases} \tag{3.3}$$

3.2.2 Asociador lineal de Anderson-Kohonen

El *Asociador Lineal* (Figura 3.2) tiene su origen en los trabajos pioneros de los años 1972 publicados por Andeson y Kohonen [4], [36], [57].

Para presentar el *Asociador Lineal* se considera nuevamente el conjunto fundamental:

$$\{(\mathbf{x}^{\mu},\mathbf{y}^{\mu}) \mid \mu = 1,2,...,p\}, \quad con \quad A=\{0,1\}, \quad \mathbf{x}^{\mu} \in A^{n} \;\; y \;\; \mathbf{y}^{\mu} \in A^{m}$$

En el *Asociador lineal* la **fase de aprendizaje** consiste en dos etapas:

1. Para cada una de las *p* asociaciones $(\mathbf{x}^{\mu},\mathbf{y}^{\mu})$ se encuentra la matriz $\mathbf{y}^{\mu}\cdot(\mathbf{x}^{\mu})^{t}$ de dimensiones $m\times n$.

$$\mathbf{y}^{\mu}\cdot(\mathbf{x}^{\mu})^{t} = \begin{pmatrix} y_1^{\mu} \\ y_2^{\mu} \\ \vdots \\ y_m^{\mu} \end{pmatrix} \cdot (x_1^{\mu} \quad x_2^{\mu} \quad \cdots \quad x_n^{\mu})^{t}$$

$$\mathbf{y}^{\mu}\cdot(\mathbf{x}^{\mu})^{t} = \begin{pmatrix} y_1^{\mu}x_1^{\mu} & y_1^{\mu}x_2^{\mu} & \cdots & y_1^{\mu}x_j^{\mu} & \cdots & y_1^{\mu}x_n^{\mu} \\ y_2^{\mu}x_1^{\mu} & y_2^{\mu}x_2^{\mu} & \cdots & y_2^{\mu}x_j^{\mu} & \cdots & y_2^{\mu}x_n^{\mu} \\ \vdots & \vdots & & \vdots & & \vdots \\ y_i^{\mu}x_1^{\mu} & y_i^{\mu}x_2^{\mu} & \cdots & y_i^{\mu}x_j^{\mu} & \cdots & y_i^{\mu}x_n^{\mu} \\ \vdots & \vdots & & \vdots & & \vdots \\ y_m^{\mu}x_1^{\mu} & y_m^{\mu}x_2^{\mu} & \cdots & y_m^{\mu}x_j^{\mu} & \cdots & y_m^{\mu}x_n^{\mu} \end{pmatrix} \tag{3.4}$$

2. Se suman las *p* matrices para obtener la memoria

$$\boldsymbol{M} = \sum_{\mu=1}^{p} \mathbf{y}^{\mu}\cdot(\mathbf{x}^{\mu})^{t} = [m_{ij}]_{m\times n} \tag{3.5}$$

De manera que la *ij*-ésima componente de la memoria ***M*** se expresa así:

$$m_{ij} = \sum_{\mu=1}^{p} y_i^{\mu}x_j^{\mu} \tag{3.6}$$

La **fase de recuperación** consiste en presentarle a la memoria un patrón de entrada $\mathbf{x}^{\omega}$, donde $\omega \in \{1,2,...,p\}$ y realizar la operación:

$$\boldsymbol{M}\cdot\mathbf{x}^{\omega} = \left[\sum_{\mu=1}^{p} \mathbf{y}^{\mu}\cdot(\mathbf{x}^{\mu})^{t}\right]\cdot\mathbf{x}^{\omega} \tag{3.7}$$

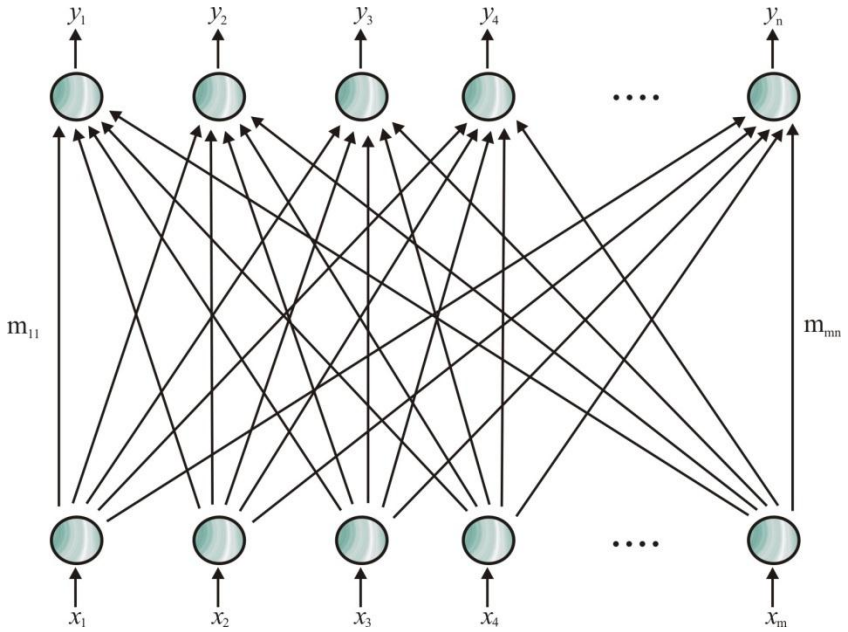

Figura 3.2. Modelo del Asociador lineal.

3.2.3 Red de auto-organización de elementos de umbral, Amari

En 1977 Shun-ichi Amari publicó un nuevo modelo de redes neuronales que establecía una nueva perspectiva, la auto-organización. Este modelo recurrente se muestra en la Figura 3.3.

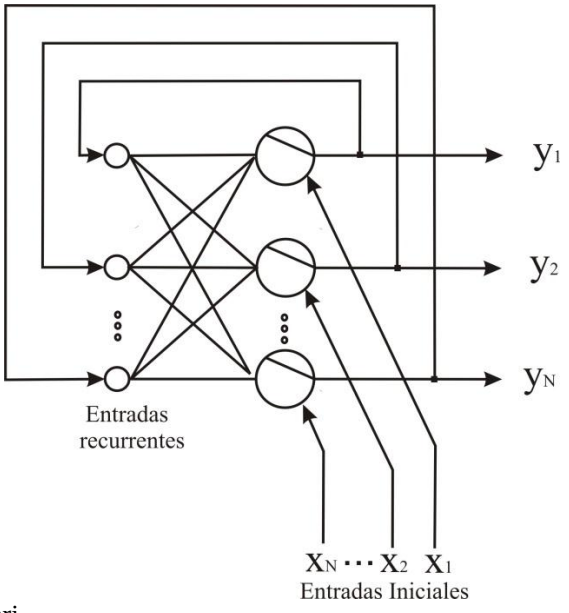

Figura 3.3. Modelo de Amari.

Aunque existe una sola capa de neuro-nodos en la arquitectura, la recurrencia tiene el efecto de proveer un número ilimitado de capas: una para cada lazo recurrente a través de la capa única. Esta red es derivada del modelo McCulloch-Pitts, y fue el precursor de la red Hopfield. Las entradas iniciales mostradas en la Figura 3.3 como: $\mathbf{x} = (x_1, x_2, ..., x_N)$, entran directamente a las neuronas y se convierten en las salidas iniciales $\mathbf{y} = (y_1, y_2, ..., y_N)$ en un tiempo $t = 0$. Las salidas entonces son retroalimentadas a unos nodos de ramificación

donde son distribuidos a cada uno de los neuro-nodos. Así cada neuro-nodo recibe las *N* entradas de retroalimentación $y_1^0, y_2^0, ..., y_N^0$ simultáneamente. En el $(r+1)$-ésimo lazo (iteración), los neuro-nodos han sacado las salidas $\mathbf{y} = y_1^r, y_2^r, ..., y_N^r$ en el tiempo $t = r$, que son alimentados y distribuidos nuevamente a los neuro-nodos. Cada *n*-ésimo nodo calcula su siguiente salida por medio de un conjunto de pesos sinápticos $\{w_{pn}\}$ $(1 \le p \le N,\ \ 1 \le n \le N)$, un conjunto de umbrales $\{\tau_n\}$, y una función de umbral. El nuevo valor de salida en el *n*-ésimo neuro-nodo es:

$$y_n^{(r+1)} = f(s) = f\left(\sum_{(p=1,2,,,,N)} w_{pn}[y_n^{(r)} - \tau_n] \right) \qquad \textbf{(3.8)}$$

Donde $f(-)$ es la función de umbral $f(s) = 1$ si $s \ge 0$; ó $f(s) = 0$ si $s < 0$ (el principio de todos o ninguno).

El estado del sistema en el tiempo $t = r$ es la pareja $\mathbf{y}^r = y_1^r, y_2^r, ..., y_N^r$ de ceros y unos. Si la red converge sobre los lazos de retroalimentación iterativos en un estado estable donde $y^{r+1} = y^r$ para $r > r_0$ para algún r_0, entonces este estado fijo es el identificador correspondiente al vector de características $\mathbf{x}$ que fue ingresado inicialmente. La *n*-ésima neurona también retroalimentará su valor y_n^r a una *n*-ésima neurona (por ejemplo, a si misma), pero esto tiende a tener un efecto dominante luego de un número de iteraciones. Estudios muestran que cuando la convergencia a un estado estable ocurre, es más rápida que otras redes recurrentes similares, desafortunadamente, la convergencia no necesita ocurrir.

3.2.4 Memoria Hopfield

El artículo de John J. Hopfield de 1982, publicado por la prestigiosa y respetada *National Academy of Sciences* (en sus *Proceedings*), impactó positivamente y trajo a la palestra internacional su famosa memoria asociativa [29].

En el modelo que originalmente propuso Hopfield, donde cada neurona x_i tiene dos posibles estados, a la manera de las neuronas de McCulloch-Pitts: $x_i = 0$ y $x_i = 1$. Sin embargo, Hopfield observa que, para un nivel dado de exactitud en la recuperación de patrones, la capacidad de almacenamiento de información de la memoria se puede incrementar por factor de 2, si se escogen como posibles estados de las neuronas los valores $x_i = -1$ y $x_i = 1$ en lugar de los valores originales $x_i = 0$ y $x_i = 1$.

Al utilizar el conjunto $\{-1,1\}$ y el valor de umbral cero, la fase de aprendizaje para la memoria Hopfield será similar, en cierta forma, a la fase de aprendizaje del *Asociador Lineal*. La intensidad de la fuerza de conexión de la neurona x_i a la neurona x_j se representa por el valor de m_{ij}, y se considera que hay simetría, es decir, $m_{ij} = m_{ji}$. Si x_i no está conectada con x_j entonces $m_{ij} = 0$. El estado instantáneo del sistema está completamente especificado por el vector columna de dimensión *n* cuyas componentes son los valores de las *n* neuronas.

La memoria Hopfield es autoasociativa, simétrica, con ceros en la diagonal principal. En virtud de que la memoria es autoasociativa, el conjunto fundamental para la memoria Hopfield es:

$$\{(\mathbf{x}^{\mu},\mathbf{x}^{\mu}) \mid \mu = 1,2,...,p\}, \quad con \quad \mathbf{x}^{\mu} \in A^{n} \quad y \quad A = \{-1,1\} \tag{3.9}$$

Fase de Aprendizaje

La fase de aprendizaje para la memoria Hopfield es similar a la fase de aprendizaje de la memoria *Asociador Lineal*, con una ligera diferencia relacionada con la diagonal principal en ceros, como se muestra en la siguiente regla para obtener la *ij*-ésima componente de la memoria Hopfield ***M***.

$$m_{ij} = \begin{cases} \sum_{\mu=1}^{p} x_i^{\mu} x_j^{\mu} & \text{si} \quad i \neq j \\ 0 \qquad \text{si} \qquad i = j \end{cases} \tag{3.10}$$

Fase de Recuperación

Si se le presenta un patrón de entrada $\tilde{\mathbf{x}}$ a la memoria Hopfield, ésta cambiará su estado con el tiempo, de modo que cada neurona x_i ajuste su valor de acuerdo con el resultado que arroje la comparación de la cantidad $\sum_{j=1}^{n} m_{ij} x_j$ con un valor de umbral, el cual normalmente se coloca en cero.

Se representa el estado de la memoria Hopfield en el tiempo *t* por $\mathbf{x}(t)$; entonces $x_1(t)$ representa el valor de la neurona x_i en el tiempo *t* y $x_i(t+1)$ el valor x_i en el tiempo siguiente $(t+1)$.

Dado un vector columna de entrada $\tilde{\mathbf{x}}$, la fase de recuperación consta de tres pasos:

1) Para $t = 0$ se hace $\mathbf{x}(t) = \tilde{\mathbf{x}}$, es decir, $x_1(0) = \tilde{x}_i, \quad \forall i \in \{1,2,3,...,n\}$.

2) $\forall i \in \{1,2,3,...,n\}$ se calcula $x_i(t+1)$ de acuerdo con la condición siguiente:

$$x_i(t+1) = \begin{cases} 1 \quad \text{si} \quad \sum_{j=1}^{n} m_{ij} x_j(t) > 0 \\ x_i(t) \quad \text{si} \quad \sum_{j=1}^{n} m_{ij} x_j(t) = 0 \\ -1 \quad \text{si} \quad \sum_{j=1}^{n} m_{ij} x_j(t) < 0 \end{cases} \tag{3.11}$$

3) Se compara $x_i(t+1)$ con $x_i(t) \quad \forall i \in \{1,2,3,...,n\}$. Si $\mathbf{x}(t+1) = \mathbf{x}(t)$ el proceso termina y el vector recuperado es $\mathbf{x}(0) = \tilde{\mathbf{x}}$. De otro modo, el proceso continúa de la siguiente manera: los pasos 2 y 3 se iteran tantas veces como sea necesario hasta llegar a un valor $t = \tau$ para el cual $x_i(\tau+1) = x_i(\tau) \quad \forall i \in \{1,2,3,...,n\}$; el proceso termina y el patrón recuperado es $\mathbf{x}(\tau)$.

En el artículo original de 1982, Hopfield había estimado empíricamente que su memoria tenía una capacidad de recuperar 0.15*n* patrones, y en el trabajo de [1] se estableció formalmente que una cota superior para el número de vectores de estado arbitrarios estables en una memoria Hopfield es *n*.

3.2.5 Memoria Asociativa Bidireccional de Kosko

Bart Kosko, investigador de la *University of Soutthern California*, propuso en 1988 la Memoria Asociativa Bidireccional (*Bidirectional Associative Memory*, BAM) [39] para subsanar la clara desventaja de la autoasociatividad de la memoria Hopfield. La BAM maneja pares de vectores $(A_1, B_1),...,(A_m, B_m)$, donde $A \in \{0,1\}^n$ y $B \in \{0,1\}^p$.

Del mismo modo que Austin ensambló dos redes asociativas de Willshaw para diseñar su ADAM (*Advanced Distributed Associative Memory*) [6], Kosko ideó un arreglo de dos memorias Hopfield, y demostró que este diseño es capaz de asociar patrones de manera heteroasociativa.

La matriz $\boldsymbol{M}$ es una memoria Hopfield con la única diferencia que la diagonal principal es diferente de cero. $\boldsymbol{M}^T$ es la matriz transpuesta de $\boldsymbol{M}$ que, ahora como entrada, recibe a *B* y la salida será *A*. El proceso bidireccional anteriormente ilustrado continúa hasta que *A* y *B* convergen a una pareja estable (A_i, B_i).

$$\begin{aligned} A &\rightarrow \boldsymbol{M} \rightarrow B \\ A' &\leftarrow \boldsymbol{M}^T \leftarrow B \\ A' &\rightarrow \boldsymbol{M} \rightarrow B' \\ A'' &\leftarrow \boldsymbol{M}^T \leftarrow B' \\ &\cdots \\ A_i &\rightarrow \boldsymbol{M} \rightarrow B_i \\ A_i &\leftarrow \boldsymbol{M}^T \leftarrow B_i \end{aligned} \qquad \textbf{(3.12)}$$

Kosko descubrió que su memoria funcionaba mejor con patrones bipolares que con patrones binarios (a la manera Hopfield), de ahí que: $A \in \{-1,1\}^n$ y $B \in \{-1,1\}^p$.

Para la codificación de la BAM se consideran las *m* asociaciones; sumándolas para formar la matriz de correlación, de acuerdo a la siguiente expresión:

$$\boldsymbol{M} = \sum_i A_i^T B_i \tag{3.13}$$

y la memoria dual $\boldsymbol{M}^T$ que está dada por:

$$\boldsymbol{M}^T = \sum_i (A_i^T B_i)^t = \sum_i B_i^T A_i \tag{3.14}$$

En el proceso de decodificación cada neurona a_i que se encuentra en el campo *A* y cada neurona b_i localizada en el campo *B*, de forma independiente y asíncrona, examina la suma de entrada de las neuronas del otro campo, entonces puede o no cambiar su estado si la suma de entrada es mayor, igual o menor que un umbral dado. Si la suma de entrada es igual al umbral, entonces la neurona no cambia su estado. La suma de entrada para b_i es el producto interno columna:

$$A\boldsymbol{M}^j = \sum_i a_i m_{ij} \tag{3.15}$$

Donde $\boldsymbol{M}^j$ es la *j*-ésima columna de $\boldsymbol{M}$. La suma de entrada para a_i es, de manera similar,

$$B\boldsymbol{M}_i^T = \sum_j b_j m_{ij} \tag{3.16}$$

Donde $\boldsymbol{M}_i$ es la *i*-ésima fila de $\boldsymbol{M}$. Se toma el 0 como el umbral para todas las neuronas. Las funciones de umbral para a_i y b_j son:

$$\begin{aligned} a_i &= \begin{cases} 1, & \text{si} \quad B\boldsymbol{M}_i^T > 0 \\ -1, & \text{si} \quad B\boldsymbol{M}_i^T < 0 \end{cases} \\ b_j &= \begin{cases} 1, & \text{si} \quad A\boldsymbol{M}^j > 0 \\ -1, & \text{si} \quad A\boldsymbol{M}^j < 0 \end{cases} \end{aligned} \tag{3.17}$$

Cuando se le presenta un patrón (*A*, *B*) a la BAM, las neuronas en los campos *A* y *B* se prenden o se apagan de acuerdo a la ocurrencia de 1's y 0's en los vectores de estado *A* y *B*. Las neuronas continúan sus cambios de estado hasta que se alcance un estado estable bidireccional (A_f, B_f).

3.2.6 Memorias Asociativas Morfológicas

Las memorias asociativas morfológicas fueron propuestas por G. Ritter *et al* [47]. La diferencia fundamental entre las memorias asociativas clásicas (*Lernmatrix, Correlograph, Asociador Lineal y la memoria Asociativa Hopfield*) y las memorias asociativas morfológicas radica en los fundamentos operacionales de éstas últimas, que son las

operaciones morfológicas de *dilatación* y *erosión*; el nombre de las memorias asociativas morfológicas está inspirado precisamente en estas dos operaciones básicas.

Estas memorias rompieron con el esquema utilizado a través de los años en los modelos de memorias asociativas clásicas, que utilizan operaciones convencionales entre vectores y matrices para la fase de aprendizaje y suma de productos para la recuperación de patrones. Las memorias asociativas morfológicas cambian los productos por sumas y las sumas por máximos o mínimos en ambas fases, tanto de aprendizaje como de recuperación de patrones [47], [61], [58].

Hay dos tipos de memorias asociativas morfológicas: las memorias *max*, simbolizadas con $\boldsymbol{M}$, y las memorias *min*, cuyo símbolo es $\boldsymbol{W}$; en cada uno de los dos tipos, las memorias pueden funcionar en ambos modos: heteroasociativo y autoasociativo.

Para entender el funcionamiento de este tipo de memoria, es necesario definir los siguientes productos matriciales:

El *producto máximo* entre **D** y **H**, denotado por $\mathbf{C} = \mathbf{D} \nabla \mathbf{H}$, es una matriz $[c_{ij}]_{m \times n}$ cuya *ij*-ésima componente c_{ij} es:

$$c_{ij} = \bigvee_{k=1}^{r} d_{ik} + h_{kj} \qquad \textbf{(3.18)}$$

El *producto mínimo* entre **D** y **H**, denotado por $\mathbf{C} = \mathbf{D} \Delta \mathbf{H}$ es una matriz $[c_{ij}]_{m \times n}$ cuya *ij*-ésima componente c_{ij} es:

$$c_{ij} = \bigwedge_{k=1}^{r} d_{ik} + h_{kj} \qquad \textbf{(3.19)}$$

Los *productos máximo* y *mínimo* contienen a los operadores *máximo* y *mínimo*, los cuales están íntimamente ligados con los conceptos de las dos operaciones básicas de la morfología matemática: *dilatación* y *erosión*, respectivamente.

Las memorias asociativas morfológicas pueden ser de carácter heteroasociativo o autoasociativo; además, cada una de ellas puede ser de tipo *max* o *min*.

3.2.6.1 Memorias Heteroasociativas Morfológicas

Algoritmo de las memorias heteroasociativas morfológicas *max*

Fase de Aprendizaje

1. Para cada una de las *p* asociaciones $\left(\mathbf{x}^{\mu}, \mathbf{y}^{\mu}\right)$ se usa el producto mínimo para crear la matriz $\mathbf{y}^{\mu} \Delta \left(-\mathbf{x}^{\mu}\right)^{t}$ de dimensiones $m \times n$, donde el negado transpuesto del patrón de entrada $\mathbf{x}^{\mu}$ se define como $\left(-\mathbf{x}^{\mu}\right)^{t} = \left(-x_1^{\mu}, -x_2^{\mu}, ..., -x_n^{\mu}\right)$.
2. Se aplica el operador máximo $\vee$ a las *p* matrices para obtener la memoria $\boldsymbol{M}$.

$$\boldsymbol{M} = \bigvee_{\mu=1}^{p} \left[\mathbf{y}^{\mu} \Delta \left(-\mathbf{x}^{\mu} \right)^{t} \right] \tag{3.20}$$

Fase de Recuperación

Esta fase consiste en realizar el producto mínimo Δ de la memoria $\boldsymbol{M}$ con el patrón entrada $\mathbf{x}^{\omega}$, donde $\omega \in (1,2,...,p)$, para obtener un vector columna **y** de dimensión *m*:

$$\mathbf{y} = \boldsymbol{M} \ \Delta \ \mathbf{x}^{\omega} \tag{3.21}$$

Las fases de aprendizaje y de recuperación de las memorias heteroasociativas morfológicas *min* se obtienen por dualidad [16].

3.2.6.2 Memorias Autoasociativas Morfológicas

Para este tipo de memorias se utilizan los mismos algoritmos descritos anteriormente y que son aplicados a las memorias heteroasociativas; lo único que cambia es el conjunto fundamental. Para este caso, se considera el siguiente conjunto fundamental:

$$\left\{ \left(\mathbf{x}^{\mu}, \mathbf{x}^{\mu} \right) \mid \mathbf{x}^{\mu} \in A^{n}, \text{ donde } \mu = 1,2,...,p \right\} \tag{3.22}$$

3.2.7 Memorias Asociativas $\alpha\beta$

En esta sección se presenta el fundamento teórico generalizado que sustenta a las memorias asociativas alfa-Beta $(\alpha\beta)$ [59]; para ello se presentan las definiciones y propiedades de las operaciones binarias originales α y β; así como la descripción de las fases de aprendizaje y recuperación de las memorias heteroasociativas y autoasociativas $(\alpha\beta)$ que utilizan máximos $(\vee)$ y mínimos $(\wedge)$.

3.2.7.1 Operaciones binarias $(\alpha\beta)$: Definición y propiedades

Las memorias Alfa-Beta utilizan máximos y mínimos, y dos operaciones binarias originales α y β de las cuales heredan el nombre.

Para la definición de las operaciones binarias $(\alpha\beta)$ se deben especificar los conjuntos *A* y *B*, los cuales son: $A = \{0, 1\}$ y $B = \{0, 1, 2\}$

La operación binaria α: $A \times A \rightarrow B$ se define como se muestra en la Tabla 3.2.

Los conjuntos *A* y *B*, las operaciones binarias α y β junto con los operadores $\wedge$ (*mínimo*) y $\vee$ (*máximo*) usuales conforman el sistema algebraico (*A*, *B*, α, β, $\wedge$, $\vee$) en el que están inmersas las memorias asociativas Alfa-Beta [59], [48].

Tabla 3.2. Operación binaria $\alpha : A \times A \to B$

x	y	$\alpha(x,y)$
0	0	1
0	1	0
1	0	2
1	1	1

La operación binaria $\beta:\ B \times A \to A$ se define como se muestra en la Tabla 3.3.

Tabla 3.3. Operación binaria $\beta : B \times A \to A$

x	y	$\beta(x,y)$
0	0	0
0	1	0
1	0	0
1	1	1
2	0	1
2	1	1

Se requiere la definición de cuatro operaciones matriciales, de las cuales se usarán sólo 4 casos particulares:

$$\begin{aligned}
&\text{Operación} \quad \alpha\max: \quad P_{m\times r}\nabla_{\alpha} Q_{r\times n} = [f_{ij}^{\alpha}]_{m\times n}, \quad \text{donde} \quad f_{ij}^{\alpha} = \underset{k=1}{\overset{r}{\vee}} \alpha(p_{ik}, q_{kj}) \\
&\text{Operación} \quad \beta\max: \quad P_{m\times r}\nabla_{\beta} Q_{r\times n} = [f_{ij}^{\beta}]_{m\times n}, \quad \text{donde} \quad f_{ij}^{\beta} = \underset{k=1}{\overset{r}{\vee}} \beta(p_{ik}, q_{kj}) \\
&\text{Operación} \quad \alpha\min: \quad P_{m\times r}\Delta_{\alpha} Q_{r\times n} = [h_{ij}^{\alpha}]_{m\times n}, \quad \text{donde} \quad h_{ij}^{\alpha} = \underset{k=1}{\overset{r}{\wedge}} \alpha(p_{ik}, q_{kj}) \\
&\text{Operación} \quad \beta\min: \quad P_{m\times r}\Delta_{\beta} Q_{r\times n} = [h_{ij}^{\beta}]_{m\times n}, \quad \text{donde} \quad h_{ij}^{\beta} = \underset{k=1}{\overset{r}{\wedge}} \beta(p_{ik}, q_{kj})
\end{aligned} \qquad \textbf{(3.23)}$$

3.2.7.2 Memorias heteroasociativas $(\alpha\beta)$

Se tienen dos tipos de memorias heteroasociativas Alfa-Beta: tipo $\vee$ (*máximo*) y $\wedge$ (*mínimo*). En la generación de ambos tipos de memorias se usará el operador $\otimes$ el cual tiene la siguiente forma:

$$[\mathbf{y}^{\mu} \otimes (\mathbf{x}^{\mu})^{t}]_{ij} = \alpha(y_i^{\mu}, x_j^{\mu}); \quad \mu \in \{1,2,...,p\}, \quad i \in \{1,2,...m\}, \quad j \in \{1,2,...,n\} \qquad \textbf{(3.24)}$$

Algoritmo Memoria heteroasociativa Alfa-Beta tipo $\vee$ (*máximo*)

Fase de Aprendizaje

Paso 1. Para cada $\mu = 1,2,...,p$, a partir de la pareja $(\mathbf{x}^{\mu}, \mathbf{y}^{\mu})$ se construye la matriz:

$$[\mathbf{y}^{\mu} \otimes (\mathbf{x}^{\mu})^{t}]_{m\times n} \tag{3.25}$$

Paso 2. Se aplica el operador binario máximo $\vee$ a las matrices obtenidas en el paso 1:

$$v_{ij} = \bigvee_{\mu=1}^{p} \alpha(y_i^{\mu}, x_j^{\mu}) \tag{3.26}$$

Fase de Recuperación

Se presenta un patrón $\mathbf{x}^{\omega}$, con $\omega \in \{1,2,...,p\}$, a la memoria heteroasociativa $\alpha\beta$ tipo $\vee$ y se realiza la operación $\Delta_{\beta} : \mathbf{V}\ \Delta_{\beta}\ \mathbf{x}^{\omega}$.

Dado que las dimensiones de la matriz $\mathbf{V}$ son de $m\times n$ y $\mathbf{x}^{\omega}$ es un vector columna de dimensión *n*, el resultado de la operación anterior debe ser un vector columna de dimensión *m*, cuya *i*-ésima componente es:

$$(\mathbf{V}\ \Delta_{\beta}\ \mathbf{x}^{\omega})_i = \bigwedge_{j=1}^{n} \beta(v_{ij}, x_j^{\mu}) \tag{3.27}$$

Algoritmo Memoria heteroasociativa Alfa-Beta tipo $\wedge$ (*mínimo*)

Fase de Aprendizaje

Paso 1. Para cada $\mu = 1,2,...,p$, a partir de la pareja $(\mathbf{x}^{\mu}, \mathbf{y}^{\mu})$ se construye la matriz:

$$[\mathbf{y}^{\mu} \otimes (\mathbf{x}^{\mu})^{t}]_{m\times n} \tag{3.28}$$

Paso 2. Se aplica el operador binario mínimo $\wedge$ a las matrices obtenidas en el paso 1:

$$\Lambda = \bigwedge_{\mu=1}^{p} [\mathbf{y}^{\mu} \otimes (\mathbf{x}^{\mu})^{t}] \tag{3.29}$$

La entrada *ij*-ésima está dada por la siguiente expresión:

$$\lambda = \bigwedge_{\mu=1}^{p} \alpha(y_i^{\mu}, x_j^{\mu}) \tag{3.30}$$

Fase de Recuperación

Se presenta un patrón $\mathbf{x}^{\omega}$, con $\omega \in \{1,2,...,p\}$, a la memoria heteroasociativa $\alpha\beta$ tipo $\wedge$ y se realiza la operación $\nabla_{\beta} : \mathbf{V}\ \nabla_{\beta}\ \mathbf{x}^{\omega}$.

Dado que las dimensiones de la matriz Λ son de $m\times n$ y $\mathbf{x}^{\omega}$ es un vector columna de dimensión *n*, el resultado de la operación anterior debe ser un vector columna de dimensión *m*, cuya *i*-ésima componente es:

$$(\Lambda \nabla_{\beta} \mathbf{x}^{\omega})_i = \bigvee_{j=1}^{n} \beta(\lambda_{ij}, x_j^{\mu}) \qquad \textbf{(3.31)}$$

3.2.7.3 Memorias autoasociativas ($\alpha\beta$)

Si a una memoria heteroasociativa se le impone la condición de que $\mathbf{y}^{\mu} = \mathbf{x}^{\mu} \quad \forall \mu \in \{1,2,...,p\}$ entonces, deja de ser heteroasociativa y ahora se le denomina autoasociativa.

A continuación se enlistan algunas de las características de las memorias autoasociativas Alfa-Beta:

1. El conjunto fundamental toma la forma $\{(\mathbf{x}^{\mu}, \mathbf{x}^{\mu}) \mid \mu = 1,2,..., p\}$
2. Los patrones fundamentales de entrada y salida son de la misma dimensión; denotada por *n*.
3. La memoria es una matriz cuadrada, para ambos tipos, $\mathbf{V}$ y $\mathbf{\Lambda}$. Si $\mathbf{x}^{\mu} \in A^n$ entonces:

$$\mathbf{V} = [v_{ij}]_{n \times n} \text{ y } \mathbf{\Lambda} = [\lambda_{ij}]_{n \times n} \qquad \textbf{(3.32)}$$

Algoritmo Memoria Autoasociativa Alfa-Beta tipo $\vee$ (*máximo*)

Las fases de aprendizaje y recuperación son similares a las memorias heteroasociativas Alfa-Beta.

Fase de Aprendizaje

Paso 1. Para cada $\mu = 1,2,..., p$, a partir de la pareja $(\mathbf{x}^{\mu}, \mathbf{x}^{\mu})$ se construye la matriz:

$$[\mathbf{x}^{\mu} \otimes (\mathbf{x}^{\mu})^t]_{n \times n} \qquad \textbf{(3.33)}$$

Paso 2. Se aplica el operador binario $\vee$ (*máximo*) a las matrices obtenidas en el paso 1:

$$\mathbf{V} = \bigvee_{\mu=1}^{p} [\mathbf{x}^{\mu} \otimes (\mathbf{x}^{\mu})^t] \qquad \textbf{(3.34)}$$

La entrada *ij*-ésima de la memoria está dada así:

$$v_{ij} = \bigvee_{\mu=1}^{p} \alpha(x_i^{\mu}, x_j^{\mu}) \qquad \textbf{(3.35)}$$

y puesto que $\alpha: \ A \times A \rightarrow B$, se tiene que $\forall j \in \{1,2,...,n\}$.

Fase de Recuperación

La fase de recuperación de las memorias autoasociativas Alfa-Beta tipo $\vee$ (*máximo*) tienen dos casos posibles. En el primer caso el patrón de entrada es un patrón fundamental; es

decir, la entrada es un patrón $\mathbf{x}^{\omega}$, con $\omega \in \{1,2,...,p\}$. En el segundo caso, el patrón de entrada no es un patrón fundamental, sino la versión distorsionada de por lo menos uno de los patrones fundamentales; lo anterior significa que si el patrón de entrada es $\tilde{x}$, debe existir al menos un valor de índice $\omega \in \{1,2,...,p\}$, que corresponde al patrón fundamental respecto del cual $\tilde{x}$ es una versión alterada con alguno de los tres tipos de ruido: *aditivo, sustractivo* o *mezclado.*

Algoritmo Memoria Autoasociativa Alfa-Beta tipo $\wedge$ (*mínimo*)

Fase de Aprendizaje

Paso 1. Para cada $\mu = 1,2,...,p$, a partir de la pareja $(\mathbf{x}^{\mu}, \mathbf{x}^{\mu})$ se construye la matriz:

$$[\mathbf{x}^{\mu} \otimes (\mathbf{x}^{\mu})^{t}]_{n \times n} \tag{3.36}$$

Paso 2. Se aplica el operador binario $\wedge$ (*mínimo*) a las matrices obtenidas en el paso 1:

$$\mathbf{\Lambda} = \bigwedge_{\mu=1}^{p} [\mathbf{x}^{\mu} \otimes (\mathbf{x}^{\mu})^{t}] \tag{3.37}$$

La entrada *ij*-ésima de la memoria está dada así:

$$\lambda_{ij} = \bigwedge_{\mu=1}^{p} \alpha(x_i^{\mu}, x_j^{\mu}) \tag{3.38}$$

y de acuerdo con que $\alpha: A \times A \rightarrow B$, se tiene que $\lambda_{ij} \in B, \ \forall i \in \{1,2,...,n\}. \ \forall j \in \{1,2,...,n\}.$

Fase de Recuperación

La fase de recuperación de las memorias autoasociativas Alfa-Beta tipo $\wedge$ (*mínimo*) tienen dos casos posibles. En el primer caso el patrón de entrada es un patrón fundamental; es decir, la entrada es un patrón $\mathbf{x}^{\omega}$, con $\omega \in \{1,2,...,p\}$. En el segundo caso, el patrón de entrada no es un patrón fundamental, sino la versión distorsionada de por lo menos uno de los patrones fundamentales; lo anterior significa que si el patrón de entrada es $\tilde{x}$, debe existir al menos un valor de índice $\omega \in \{1,2,...,p\}$, que corresponde al patrón fundamental respecto del cual $\tilde{x}$ es una versión alterada con alguno de los tres tipos de ruido: *aditivo, sustractivo* o *mezclado.*

3.2.8 Memorias Asociativas tipo Mediana

Las memorias asociativas Mediana [50] utilizan los operadores A y B, definidos de la siguiente forma:

$$\begin{aligned} \mathbf{A}(x,y) &= x - y \\ \mathbf{B}(x,y) &= x + y \end{aligned} \tag{3.39}$$

Las operaciones utilizadas se describen a continuación.

Sean $\mathbf{P} = [p_{ij}]_{m\times r}$ y $\mathbf{Q} = [q_{ij}]_{r\times n}$ dos matrices.

$$\text{Operación } \langle\rangle_{\mathrm{A}}: \ \mathbf{P}_{m\times r} \ \langle\rangle_{\mathrm{A}} \ \mathbf{Q}_{r\times n} = [f_{ij}^{\ \mathrm{A}}]_{\mathrm{m\times n}} \text{ donde } f_{ij}^{\ \mathrm{A}} = \underset{k=1}{\overset{r}{\mathbf{med}}} \ \mathrm{A}(p_{ik}, q_{k,j})$$
$$\text{Operación } \langle\rangle_{\mathrm{B}}: \ \mathbf{P}_{m\times r} \ \langle\rangle_{\mathrm{B}} \ \mathbf{Q}_{r\times n} = [f_{ij}^{\ \mathrm{B}}]_{m\times n} \text{ donde } f_{ij}^{\ \mathrm{B}} = \underset{k=1}{\overset{r}{\mathbf{med}}} \ \mathrm{B}(p_{ik}, q_{k,j}) \qquad \textbf{(3.40)}$$

3.2.8.1 Algoritmo de las Memorias Mediana

Fase de Aprendizaje

Paso 1. Para cada $\xi = 1,2,..., p$, de cada pareja $(\mathbf{x}^{\xi}, \ \mathbf{y}^{\xi})$ se construye la matriz:

$$[\mathbf{y}^{\xi} \ \langle\rangle_{\mathrm{A}} (\mathbf{x}^{\xi})^{t}]_{m\times n} \qquad \textbf{(3.41)}$$

Paso 2. Se aplica el operador media a las matrices obtenidas en el *paso 1* para obtener la matriz ***M***, como sigue:

$$[\boldsymbol{M} = \underset{\xi=1}{\overset{p}{\mathbf{med}}} [y^{\xi} \ \langle\rangle_{\mathrm{A}} \ (x^{\xi})^{t}] \qquad \textbf{(3.42)}$$

El *ij*-ésimo componente ***M*** está dado como sigue:

$$m_{ij} = \underset{\xi=1}{\overset{p}{\mathbf{med}}} \ \mathrm{A}(y_i^{\xi}, \ x_j^{\xi}) \qquad \textbf{(3.43)}$$

En la fase de recuperación se tienen dos casos:

Caso 1. Recuperación de un patrón fundamental. Un patrón $\mathbf{x}^{w}$, con $w \in \{1,2,..., p\}$ se le presenta a la memoria ***M*** y se realiza la siguiente operación:

$$\boldsymbol{M} \ \langle\rangle_{\mathrm{B}} \ \mathbf{x}^{w} \qquad \textbf{(3.44)}$$

El resultado es un vector columna de dimensión *n*, con la *i*-ésima componente dada como:

$$(\boldsymbol{M} \ \langle\rangle_{\mathrm{B}} \ \mathbf{x}^{w})_i = \underset{j=1}{\overset{n}{\mathbf{med}}} \ \mathrm{B}(m_{ij}, x_j^{w}) \qquad \textbf{(3.45)}$$

Caso 2. Recuperación de un patrón alterado. Un patrón $\tilde{\mathbf{x}}$, que es una versión alterada de un patrón $\mathbf{x}^{w}$, se le presenta a la memoria ***M***, y se realiza la siguiente operación:

$$\boldsymbol{M} \ \langle\rangle_{\mathrm{B}} \ \tilde{\mathbf{x}} \qquad \textbf{(3.46)}$$

De nuevo, el resultado es un vector de dimensión *n*, con la *i*-ésima componente dada como:

$$(\boldsymbol{M} \ \langle\rangle_{\mathrm{B}} \ \tilde{\mathbf{x}})_i = \underset{j=1}{\overset{n}{\mathbf{med}}} \ \mathrm{B}(m_{ij}, x_j) \qquad \textbf{(3.47)}$$

Capítulo 4

Solución propuesta, esquema VQ-MAE

El presente trabajo propone la combinación de dos algoritmos, LBG y MAE, con la finalidad de crear un nuevo esquema de cuantificación vectorial con aplicación en la compresión de imágenes estáticas. Este nuevo esquema reduce significativamente la complejidad computacional sin sacrificar el rendimiento.

4.1 Métodos utilizados

4.1.1 Algoritmo LBG (Linde-Buzo-Gray)

El algoritmo LBG está formado por dos procesos: 1) Generación del libro de códigos y 2) Búsqueda de los vectores de reconstrucción. En el proceso de generación del libro de códigos, el algoritmo LBG iterativamente hace uso de un conjunto de vectores de entrenamiento de forma aleatoria y de un libro de códigos inicial para generar un libro de códigos final localmente óptimo [40], [32]. El libro de códigos, formado por un conjunto finito de vectores $\mathbf{y} = [y_j]$, *n*-dimensionales, es denotado por $\mathbf{C} = \{\mathbf{y}^i : i = 1,2,3,...,N\}$, y cada uno de sus vectores es llamado vector de reconstrucción. El algoritmo de generación del libro de códigos consiste de la siguiente secuencia de pasos [54]:

Paso 1. Inicialización. Toma un conjunto de *N* vectores de entrenamiento y genera un libro de códigos inicial.

Paso 2. Particionamiento. Determina la distribución del conjunto de entrenamiento usando la condición del vecino más próximo (Distancia Euclidiana) con respecto a los vectores de reconstrucción.

Paso 3. Actualización del libro de códigos. Genera un nuevo libro de códigos aplicando la condición del centroide, definida como:

$$y_i = \frac{1}{m}\sum_{j=1}^{m} x_i^j \qquad \textbf{(4.1)}$$

donde: y_i es el componente *i*-ésimo del vector de reconstrucción, y x_i^j es el *i*-ésimo componente del *j*-ésimo vector de entrenamiento y *m* es el número de vectores de entrenamiento asociados al vector de reconstrucción.

Paso 4. Chequeo de Convergencia. Parar los cálculos iterativos cuando el proceso converja. De no ser así, regresar al paso 2.

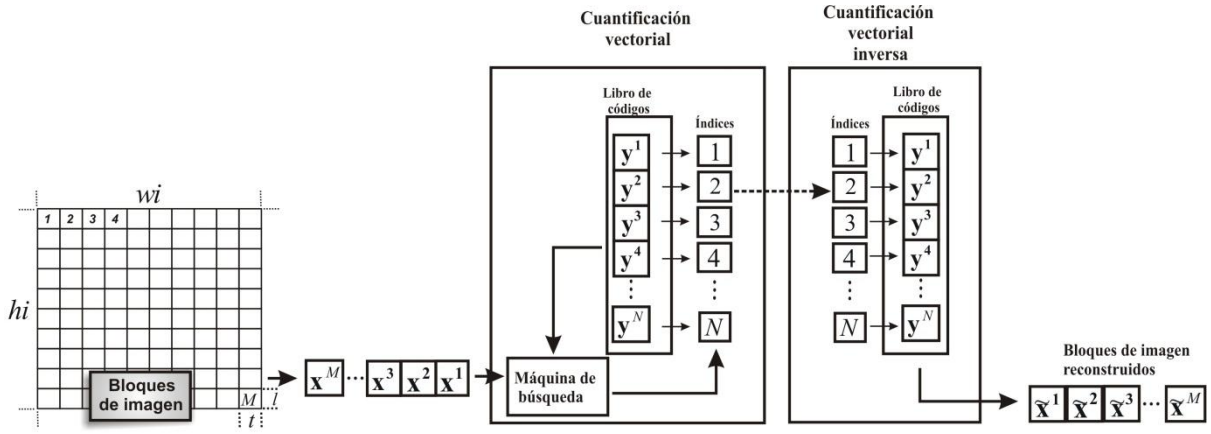

Figura 4.1. Esquema de cuantificación vectorial LBG.

La Figura 4.1 muestra un esquema básico de VQ por medio del algoritmo LBG (VQ-LBG). En este esquema, la imagen transformada de $hi \times wi$ píxeles es dividida en *M* bloques de tamaño $n = t \times l$. Estos bloques representan los vectores de entrada $\mathbf{x} = [x_j]$ *n*-dimensionales, denotados por $X = \{\mathbf{x}^i : i = 1,2,...,M\}$, $n = t \times l$. El algoritmo VQ-LBG consiste de un mapeo de los vectores de entrada sobre un conjunto finito de vectores de reconstrucción: $\mathbf{y} = [y_j]$, $j = 1,2,3,...,16$, $q : \mathbf{R}^n \to C$, $C = \{\mathbf{y}^i : i = 1,2,...,N\}$, $\mathbf{y}^i \in \mathbf{R}^n$.

Asociado a cada uno de los vectores de reconstrucción del libro de códigos se tiene una celda denominada región de "*Voronoi*", v_i; la *i*-ésima región está definida por:

$$V_i = \{\mathbf{x} \in \mathbf{R}^n : \ q(\mathbf{x}) = \mathbf{y}^i\}$$
$$V_i = \{\mathbf{x} \in \mathbf{R}^n : \ \|\mathbf{x} - \mathbf{y}^i\| \le \|\mathbf{x} - \mathbf{y}^j\|, \ \forall \ j \ne i\} \qquad \textbf{(4.2)}$$

El conjunto de *regiones de Voronoi* (Figura 4.2) forman una división de $\mathbf{R}^k$ tal que:

$$\bigcup_{i=1}^{N} V_i = \mathbf{R}^n$$
$$V_i \bigcap_{i=1}^{N} V_j = \phi, \quad \forall \ j \neq i \tag{4.3}$$

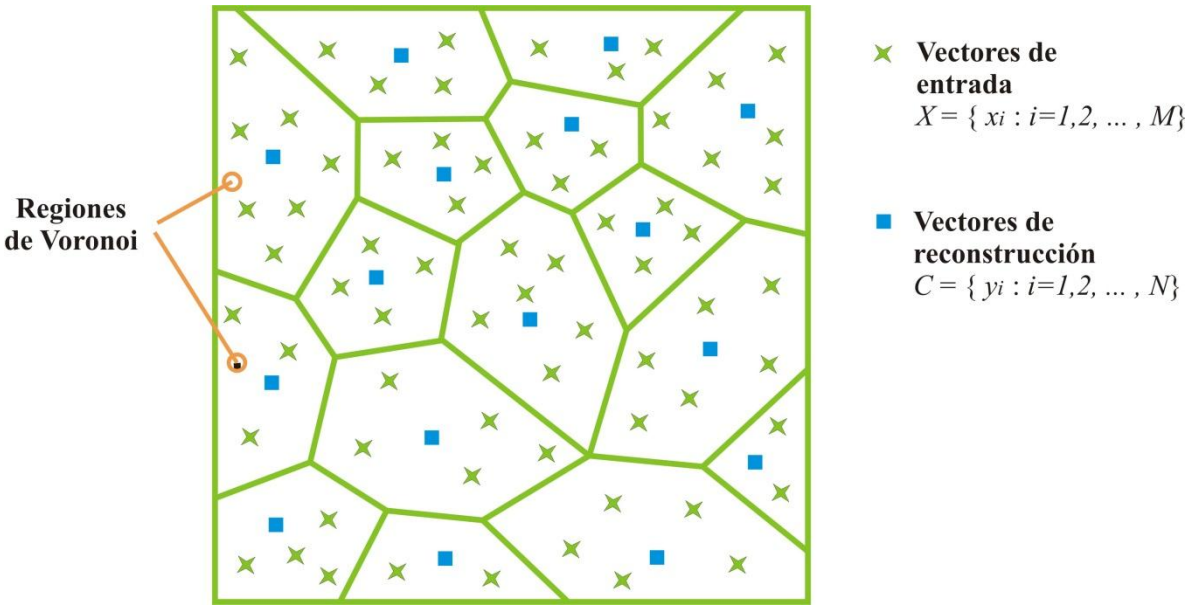

Figura 4.2. Representación gráfica de los vectores de entrada, de los vectores de reconstrucción y de la región de Voronoi.

Con base en un criterio de similitud entre el vector de entrada y los vectores de reconstrucción, cada vector de entrada perteneciente a la región V_i es substituido por el índice del vector de reconstrucción $\mathbf{y}^i$. La distancia Euclidiana es un popular criterio de similitud y es definida por:

$$d(\mathbf{x}, \mathbf{y}^i) = \sqrt{\sum_{j=1}^{n} (x_j - y_j^i)^2} \tag{4.4}$$

Donde x_j es el *j*-ésimo componente vector de entrada y y_j^i es el *j*-ésimo componente del vector de reconstrucción $\mathbf{y}^i$.

4.1.2 Memorias Asociativas Extendidas

En este apartado se estudiará el elemento principal de la propuesta de este trabajo, las Memorias Asociativas Extendidas. Este modelo de memoria está orientado a la clasificación de patrones. Una memoria Asociativa diseñada para la clasificación de patrones es un elemento cuyo propósito fundamental es establecer una relación de un patrón de entrada $\mathbf{x} = [x_i]_n$ con el índice *i* de una clase c_i (ver Figura 4.3)

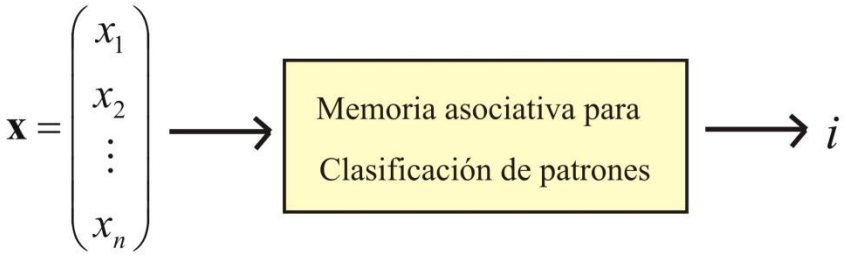

Figura 4.3. Esquema de una Memoria Asociativa para la clasificación de patrones.

Un conjunto de N pares ordenados (tuplas) $\{(\mathbf{x}^1, c_1), (\mathbf{x}^2, c_2), ..., (\mathbf{x}^N, c_N)\}$, donde cada elemento está integrado por un patrón $\mathbf{x}^\mu$ y su correspondiente índice de clase c_μ, define al "conjunto fundamental de parejas" (tuplas fundamentales). Así, una representación general del conjunto fundamental de parejas para una memoria asociativa orientada a la clasificación es representado por:

$$\{(\mathbf{x}^\mu, c_\mu) \quad | \quad \mu = 1,2,3,..., N\} \tag{4.5}$$

donde: $\mathbf{x}^\mu \in \mathbf{R}^n$ y $c_\mu = \{1,2,3,\cdots,N\}$.

La memoria asociativa es representada por una matriz generada a partir del conjunto fundamental de parejas y es denotada ***M***.

En [51] propusieron un modelo de memoria asociativa para la clasificación de patrones con valores reales. Este modelo, llamado Memoria Asociativa Extendida, es una extensión del modelo Lernmatrix propuesto por K. Steinbuch (1961). La MAE está basada en el concepto general de la función de aprendizaje de la memoria asociativa y presenta un alto desempeño en la clasificación de patrones de datos con valores reales en sus componentes y con versiones alteradas de patrones pertenecientes al conjunto fundamental de parejas [53], [8], [9].

En una MAE, al ser una memoria asociativa utilizada en la clasificación de patrones, todos los pesos sinápticos que pertenecen a la salida *i* son ordenados en la *i*-ésima fila de la matriz ***M***. El valor final de esta fila es una función ϕ de todos los patrones pertenecientes a la clase *i*. La función ϕ actúa como un mecanismo de aprendizaje generalizado que refleja la flexibilidad de la memoria [51].

La fase de aprendizaje consiste en evaluar la función ϕ para cada una de las clases. Entonces, la matriz ***M*** puede ser estructurada como:

$$\boldsymbol{M} = \begin{bmatrix} \phi_1 \\ \vdots \\ \phi_N \end{bmatrix} \tag{4.6}$$

donde ϕ_i es la evaluación de la función ϕ para todos los patrones de la clase *i*, $i = 1,2,\cdots,N$. La función ϕ puede ser evaluada de varias formas. Frecuentemente, los operadores promedio aritmético (**prom**) y mediana (**med**) son usados en el procesamiento de señales e

imágenes. En [51], los autores analizan el desempeño de estos operadores cuando se usan en las MAE para evaluar la función ϕ. Además, en [53] se propone el uso de los operadores punto medio (**pmed**) y suma (**sum**) con el mismo propósito.

4.1.2.1 Fase de Entrenamiento de la MAE

El objetivo de la fase de entrenamiento es establecer una relación entre un patrón de entrada $\mathbf{x} = [x_i]_n$, y el índice i de la clase c_i.

Considerando que cada clase se compone de *q* patrones $\mathbf{x} = [x_i]_n$ y que $\phi_i = [\phi_{i,1}, \phi_{i,2}, \cdots, \phi_{i,n}]$. Entonces, la fase de entrenamiento de la MAE, cuando usa el operador **prom** para evaluar la función $\phi_{i,j}$, está definida por:

$$\phi_{i,j} = \tfrac{1}{q} \sum_{l=1}^{q} x_{j,l}, \quad j = 1,2,3,\cdots,n \tag{4.7}$$

La fase de entrenamiento de la MAE, cuando se usa el operador **med** para evaluar la función $\phi_{i,j}$, está definida por:

$$\phi_{i,j} = \underset{l=1}{\overset{q}{\text{med}}} \; x_{j,l}, \quad j = 1,2,3,\cdots,n \tag{4.8}$$

Cuando el operador **pmed** se usa para evaluar la función $\phi_{i,j}$, la fase de entrenamiento de la MAE está definida por:

$$\phi_{i,j} = \frac{\gamma_{i,j} + \lambda_{i,j}}{2}, \quad j = 1,2,3,\cdots,n \tag{4.9}$$

Cuando el operador **sum** se usa para evaluar la función $\phi_{i,j}$, la fase de entrenamiento de la MAE está definida por:

$$\phi_{i,j} = \gamma_{i,j} + \lambda_{i,j}, \quad j = 1,2,3,\cdots,n \tag{4.10}$$

donde $\gamma_{i,j}$ y $\lambda_{i,j}$ representan los vectores máximos y mínimos de la clase *i* respectivamente. Estos vectores están definidos por:

$$\gamma_{i,j} = \bigvee_{l=1}^{q} (x_j^{i,l}) \tag{4.11}$$

$$\lambda_{i,j} = \bigwedge_{l=1}^{q} (x_j^{i,l}) \tag{4.12}$$

Los operadores max($\vee$) y min($\wedge$) ejecutan operaciones morfológicas en los patrones de cada clase. La memoria asociativa $\boldsymbol{M}$, se obtiene después de evaluar todas las funciones $\phi_{i,j}$. En el caso cuando existen las *N* clases y los vectores a clasificar son *n*-dimensionales, la memoria resultante $\boldsymbol{M} = [m_{i,j}]_{N\times n}$ está denotada por:

$$\boldsymbol{M} = \begin{bmatrix} \phi_{1,1} & \phi_{1,2} & \cdots & \phi_{1,n} \\ \phi_{2,1} & \phi_{2,2} & \cdots & \phi_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ \phi_{N,1} & \phi_{N,2} & \cdots & \phi_{N,n} \end{bmatrix} = \begin{bmatrix} m_{1,1} & m_{1,2} & \cdots & m_{1,n} \\ m_{2,1} & m_{2,2} & \cdots & m_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ m_{N,1} & m_{N,2} & \cdots & m_{N,n} \end{bmatrix} \tag{4.13}$$

4.1.2.2 Fase de Clasificación de la MAE

El objetivo de la fase de clasificación es la generación del *índice de la clase* a la que pertenece cada patrón de entrada. La clasificación de patrones por medio de la MAE se realiza cuando un patrón $\mathbf{x}^{\mu} \in \boldsymbol{R}^n$ se presenta a la memoria $\boldsymbol{M}$ generada en la etapa de entrenamiento. La fase de clasificación de la MAE tiene la propiedad de asignar el índice del patrón de reconstrucción más cercano al vector de entrada sin necesidad que este último haya sido utilizado en la generación de la memoria $\boldsymbol{M}$.

Cuando se usan los operadores **prom** o **pmed**, la clase a la que pertenece el vector **x** está dada por:

$$i = \underset{l}{\mathbf{arg}}\ [\overset{N}{\underset{l=1}{\wedge}}\ \overset{n}{\underset{j=1}{\vee}} |m_{l,j} - x_j|] \tag{4.14}$$

En este caso, los operadores $\vee \equiv \max$ y $\wedge \equiv \min$ ejecutan operaciones morfológicas sobre el valor absoluto de la diferencia entre los elementos $m_{l,j}$ de $\boldsymbol{M}$ y los componentes x_j del patrón **x** a ser clasificado.

Cuando la MAE utiliza el operador **med,** la clase a la cual el vector **x** pertenece está dada por:

$$i = \underset{l}{\mathbf{arg}}\ [\overset{N}{\underset{l=1}{\wedge}} \left| \underset{j=1}{\overset{n}{\mathrm{med}}}\, m_{l,j} - \underset{j=1}{\overset{n}{\mathrm{med}}}\, x_j \right|] \tag{4.15}$$

En este caso, el operador $\wedge \equiv \min$ ejecuta una erosión morfológica sobre el valor absoluto de la diferencia entre la mediana de los elementos $m_{l,j}$ de $\boldsymbol{M}$ y la mediana del componente x_j del patrón **x** a ser clasificado.

Cuando se usa el operador **sum**, se debe generar la matriz de recuperación $\mathbf{r} = [r_{ij}]_{N\times n}$. Los componentes de la matriz de recuperación se pueden calcular de dos formas:

$$r_{ij} = x_j + \gamma_{l,j} \tag{4.16}$$

$$r_{ij} = x_j + \lambda_{l,j} \tag{4.17}$$

Entonces, la clase a la cual pertenece el vector **x** está dada por:

$$i = \underset{l}{\mathbf{arg}}\ [\overset{N}{\underset{l=1}{\wedge}}\ \overset{n}{\underset{j=1}{\vee}} \left| m_{l,j} - r_{l,j} \right|] \qquad \textbf{(4.18)}$$

En este caso, los operadores $\vee \equiv \max$ y $\wedge \equiv \min$ ejecutan operaciones morfológicas sobre el valor absoluto de la diferencia entre los elementos $m_{l,j}$ de $\boldsymbol{M}$ y los componentes $r_{l,j}$ de la matriz de recuperación.

El **Teorema 1** y los **Corolarios 1-5** de [51] gobiernan las condiciones que deben ser satisfechas para obtener una clasificación perfecta, ya sea que el patrón pertenezca al conjunto fundamental de parejas o sea una versión alterada de un patrón de dicho conjunto.

Aquí los reproduciremos a ellos.

Teorema 1. Dado que $d_i = \underset{\mathrm{x\ class}\ i}{\vee} d(\mathbf{x}, \phi_i)$ y $\boldsymbol{R}_i = \{\mathbf{x} :\ d(\mathbf{x}, \phi_i) \le d_i\}$ hiper cubos centrados en ϕ_i en un semilado $d_i,\ i = 1,2,3,\cdots,N$. Si $d(\phi_i, \phi_j) > 2\ \max\ \{d_i, d_j\},$ entonces:

$\boldsymbol{R}_i \bigcap \boldsymbol{R}_j = \phi,\ \ 1 \le i, j \le N,\ \ i \ne j$.

$\mathbf{x} \in \boldsymbol{R}_i$ *implica* $d(\mathbf{x}, \phi_i) \le d(\mathbf{x}, \phi_j)$.

$\mathbf{x} \in \boldsymbol{R}_j$ *implica* $d(\mathbf{x}, \phi_j) \le d(\mathbf{x}, \phi_i)$.

Corolario 1. Si las condiciones del Teorema 1 se aplican a todo $1 \le i, j \le m,\ i \ne j$ entonces:

$\boldsymbol{R}_i \bigcap \boldsymbol{R}_j = \phi,\ \ 1 \le i, j \le N,\ \ i \ne j.$

si $\mathbf{x} \in \boldsymbol{R}_i$ *entonces* $d(\mathbf{x}, \phi_i) \le d(\mathbf{x}, \phi_j),\ 1 \le i, j \le N,\ \ i \ne j$.

Corolario 2. La clasificación perfecta del conjunto fundamental de patrones se debe a cualquier patrón fundamental **x**, construido, perteneciente a su región correspondiente $\boldsymbol{R}_i$ y si las condiciones del Teorema 1 se mantienen, entonces $d(\mathbf{x}, \phi_i) \le d(\mathbf{x}, \phi_j),\ 1 \le j \le N,\ \ i \ne j.$ Por lo tanto, la memoria asigna el patrón fundamental **x** a la clase *i* como es debido.

Corolario 3. Si $\tilde{\mathbf{x}}$ es una versión alterada del patrón fundamental $\mathbf{x} \in \boldsymbol{R}_i$, $\tilde{\mathbf{x}}$ será clasificada correctamente si $\tilde{\mathbf{x}} \in \boldsymbol{R}_i$.

Corolario 4. La cuenca de atracción de la *i*-ésima clase es al menos $\boldsymbol{R}_i$ y la convergencia de la clasificación correcta se realiza en un paso.

Corolario 5. Dejar a **x** ser un patrón a clasificar. Si $\mathbf{x} \notin \boldsymbol{R}_i$ y $\mathbf{x} \notin \boldsymbol{R}_j$ entonces deberá clasificar en la clase para la cual:

$i = \underset{l}{\mathbf{arg}}\ [\overset{N}{\underset{l=1}{\wedge}}\ \overset{n}{\underset{j=1}{\vee}} \left| m_{lj} - x_j \right|]$ operadores **prom** y **pmed**.

$i = \underset{l}{\mathbf{arg}}\ [\overset{N}{\underset{l=1}{\wedge}}\left|\overset{n}{\underset{j=1}{\mathrm{med}}}\, m_{lj} - \overset{n}{\underset{j=1}{\mathrm{med}}}\, x_j\right|]$ operador **med**.

$i = \underset{l}{\mathbf{arg}}\ [\overset{N}{\underset{l=1}{\wedge}}\overset{n}{\underset{j=1}{\vee}}\left|m_{lj} - r_{lj}\right|]$ operador **sum**.

4.2 Esquema VQ-MAE

La VQ es un método usado en la compresión con pérdida de datos, que puede producir resultados muy próximos a los límites teóricos. Sin embargo, su desventaja principal es que el proceso de búsqueda al basar su funcionamiento en un algoritmo de exploración total de igualación, lo hace un proceso lento y de una complejidad computacional considerable.

En este apartado se describe el esquema de cuantificación vectorial de imágenes con base en las MAE (VQ-MAE). Como en cualquier esquema de VQ, la propuesta de este trabajo consta de dos procesos: la generación del libro de códigos y la búsqueda del vector de reconstrucción (proceso de codificación o proceso de cuantificación vectorial).

El esquema VQ propuesto hace uso del algoritmo LBG y de las MAE. En la generación del libro de códigos, se aplica la fase de aprendizaje de las MAE, utilizando un operador **prom**, **med, pmed o sum**, a un libro de códigos generado mediante el algoritmo LBG; el resultado de esta etapa es una red asociativa cuyo objetivo es establecer una relación entre el conjunto de entrenamiento y el libro de códigos generado por el algoritmo LBG; esta red asociativa es un nuevo libro de códigos, denominado **MAE-codebook**, usado por el esquema para la cuantificación vectorial propuesto. En el proceso de codificación, teniendo como elemento central al **MAE-codebook**, se utiliza la fase de clasificación de las MAE para obtener un proceso de búsqueda rápida, este proceso tiene por función generar eficientemente el conjunto de los índices de clases a los cuales cada vector de entrada pertenece, completando así la cuantificación vectorial.

El proceso de generación del libro de códigos de un cuantificador mediante el algoritmo LBG es la técnica más simple y ampliamente usada. Debido a esto, se ha decidido emplear este algoritmo para generar el libro de códigos inicial que usará el esquema propuesto en este trabajo.

La propuesta de usar Memorias Asociativas Extendidas en este trabajo se fundamenta principalmente en la alta velocidad de procesamiento, baja demanda de recursos, alta inmunidad al ruido, y de que se trata de una técnica usada en la clasificación de patrones que a diferencia de otras técnicas no requiere converger para realizar una clasificación perfecta.

La alta velocidad de procesamiento que una MAE puede generar se debe a que éstas basan su funcionamiento en las operaciones morfológicas de erosión y dilatación, así, la complejidad computacional se ve reducida drásticamente ya que las operaciones utilizadas son muy simples: máximos o mínimos de sumas.

4.3 Generación del libro de códigos MAE-codebook

En este proceso se genera una red asociativa aplicando la fase de aprendizaje de la MAE sobre un libro de códigos generado con el algoritmo LBG y un conjunto de vectores de entrenamiento. Esta red asociativa representa el llamado **MAE-codebook** y tiene por

función establecer una relación entre el conjunto de entrenamiento y el libro de códigos LBG.

El proceso de generación del libro de códigos basado en el algoritmo LBG y las MAE es fundamental en la obtención de un algoritmo de búsqueda rápida para VQ de imágenes propuesto en este trabajo. El **MAE-codebook** se obtiene en tres fases:

La Figura 4.4 muestra el diagrama de bloques del proceso de generación del libro de códigos **MAE-codebook**.

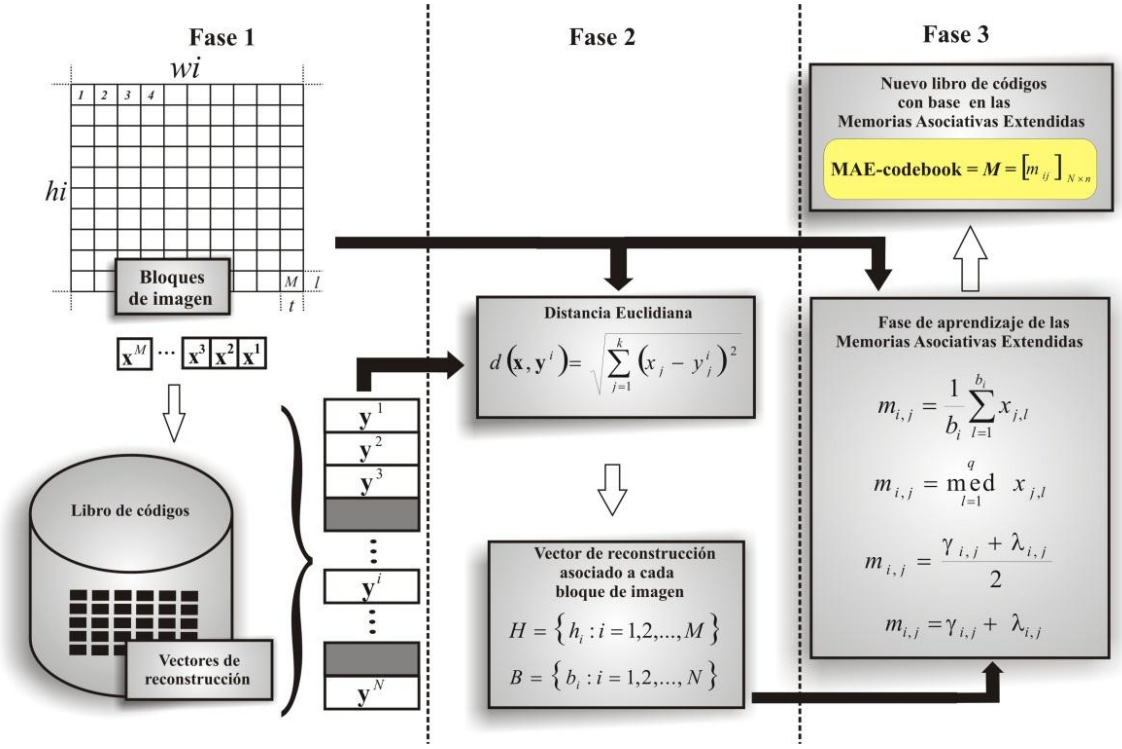

Figura 4.4. Esquema del algoritmo de generación del libro de códigos basado en MAE (**MAE-codebook**).

Fase 1. *Generación del libro de códigos LBG***.** Esta fase usa el algoritmo LBG y los bloques de imagen (conjunto de entrenamiento) $\mathbf{X}=\{\mathbf{x}^i:\ i=1,2,3,\cdots,M\}$ para generar un libro de códigos inicial $\mathbf{C}=\{\mathbf{y}^i:\ i=1,2,3,\cdots,N\}$.

Este libro de códigos será formado por un conjunto de vectores *n*-dimensionales $\mathbf{y}=[y_j]$, llamados vectores de reconstrucción.

Fase 2. *Determinación del vector de reconstrucción asociado a cada bloque de imagen (conjunto de entrenamiento) usando la distancia Euclidiana***.** Esta fase usa el libro de códigos generado en la fase previa y los *M* bloques de la imagen usada como conjunto de entrenamiento. Cada bloque representa un vector de entrada **x** *n*-dimensional y el conjunto de vectores de entrada se denota por $\mathbf{X}=\{\mathbf{x}^i:\ i=1,2,3,\cdots,M\}$. Esta etapa diseña un mapeo $\boldsymbol{Q}$ y asigna un índice $\boldsymbol{i}$ a cada vector de entrada *n*-dimensional $\mathbf{x}=(x_1,x_2,\cdots,x_n)$, con $\boldsymbol{Q}(\mathbf{x})=\mathbf{y}^i=(y_{i,1},y_{i,2},\cdots,y_{i,n})$. El mapeo $\boldsymbol{Q}$ es diseñado para asignar **x** a $\mathbf{y}^i$ cuando $\mathbf{y}^i$ satisface la siguiente condición:

$$d(\mathbf{x},\mathbf{y}^i) = \min_j \; d(\mathbf{x},\mathbf{y}^j), \quad para \quad j = 1,2,3,\cdots,N \tag{4.19}$$

donde $d(\mathbf{x},\mathbf{y}^j)$ es la distorsión, medida mediante la distancia euclidiana, entre el vector de entrada $\mathbf{x}$ y el vector de reconstrucción $\mathbf{y}^j$. Cada vector de reconstrucción $\mathbf{y}^i$ representa una clase (existen *N* clases), entonces el resultado de esta etapa generará un conjunto de *M* índices que indican la clase a la que pertenecen cada vector de entrada **x**, se denota a este conjunto de índices por $\boldsymbol{H} = \{h_i : \quad i = 1,2,3,\cdots,M\}$. Esta etapa también genera un conjunto de *N* índices que indican el número de vectores de entrada que integran cada clase, este conjunto es denotado por $\boldsymbol{B} = \{b_i : \quad i = 1,2,3,\cdots,N\}$.

Con base en los conjuntos $\boldsymbol{H}$ y $\boldsymbol{B}$**,** y considerando que el conjunto de vectores de entrada está conformado de *M* bloques de imagen, $\mathbf{X} = \{\mathbf{x}^i : \; i = 1,2,3,\cdots,M\}$, el conjunto fundamental de asociaciones queda definido por:

$$\begin{array}{cccc} \mathbf{x}^{h_{11}} \in C_1 & \mathbf{x}^{h_{21}} \in C_2 & \cdots & \mathbf{x}^{h_{M1}} \in C_N \\ \mathbf{x}^{h_{12}} \in C_1 & \mathbf{x}^{h_{22}} \in C_2 & \cdots & \mathbf{x}^{h_{M2}} \in C_N \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{x}^{h_{1b_1}} \in C_1 & \mathbf{x}^{h_{2b_2}} \in C_2 & \cdots & \mathbf{x}^{h_{Mb_N}} \in C_N \end{array} \tag{4.20}$$

donde, $b_1, b_2, \cdots, b_N$ pueden tener valores diferentes y $b_1 + b_2 + \cdots + b_N = M$.

La Figura 4.5 muestra la definición del conjunto fundamental de asociaciones.

Fase 3. *Generación un nuevo libro de códigos basado en MAE***.** El objetivo de esta tercera etapa es generar un nuevo libro de códigos, **MAE-codebook**.

Encontrar una solución óptima del problema de cuantificación vectorial característico requiere un proceso de entrenamiento que involucre el aprendizaje de la distribución de probabilidad de los datos de entrada. Para este propósito, la fase de entrenamiento de la MAE se aplica sobre el conjunto fundamental de asociaciones denotado por la ecuación (4.20).

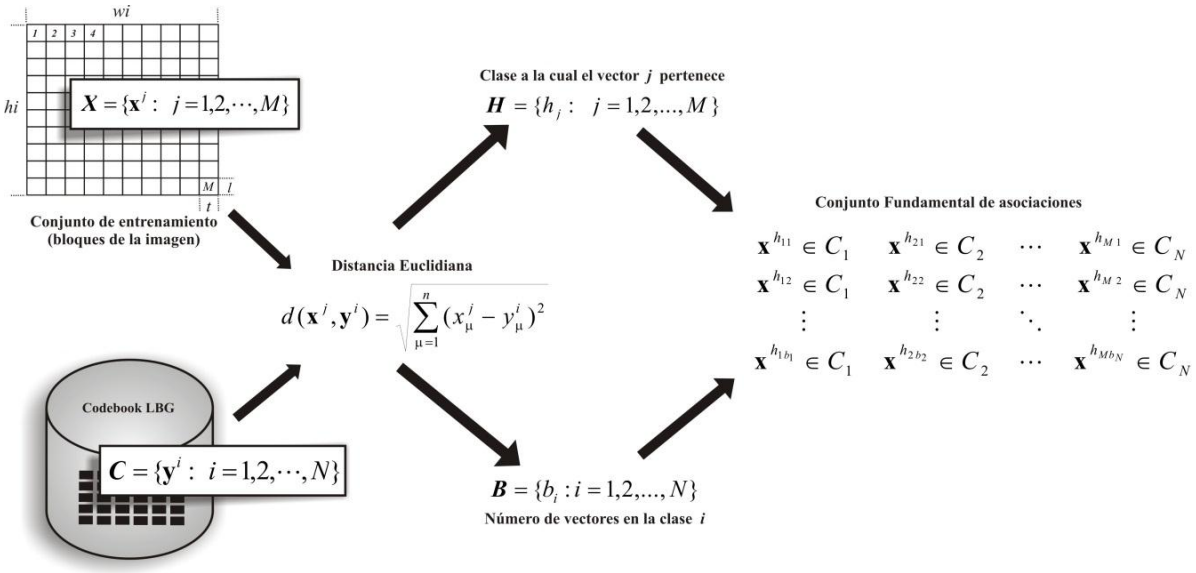

Figura 4.5. Definición del conjunto fundamental de asociaciones.

Considerando que cada clase agrupa $b_1, b_2, \cdots, b_N$ vectores de entrada, $\mathbf{x} = [x_i]_n$, el mecanismo de aprendizaje generalizado de las MAE queda definido por las expresiones (4.7), (4.8), (4.9) y (4.10) para los operadores **prom**, **med**, **pmed** y **sum** respectivamente.

$$m_{uv} = \frac{1}{q} \sum_{g=1}^{q} x_v^g$$

$$m_{uv} = \underset{g=1}{\overset{q}{\mathbf{med}}} x_v^g$$

$$m_{uv} = \frac{\underset{g=1}{\overset{q}{\vee}} x_v^{u,g} + \underset{g=1}{\overset{q}{\wedge}} x_v^{u,g}}{2}$$

$$m_{uv} = \underset{g=1}{\overset{q}{\vee}} x_v^{u,g} + \underset{g=1}{\overset{q}{\wedge}} x_v^{u,g}$$

Donde *q* puede tomar los valores de $b_1, b_2, b_3 \cdots, b_N$, $v = 1,2,3,\cdots,n$ y $u = 1,2,3,\cdots,N$.

El resultado de este paso es una red asociativa, **MAE- codebook**, que establece una relación entre los elementos del conjunto de entrenamiento y el libro de códigos generados por el algoritmo LBG. Al existir *N* clases y dado que los vectores de entrada son *n*-dimensionales, entonces, la red asociativa se representará por una matriz $\boldsymbol{M} = [m_{uv}]_{N \times n}$.

$$\textbf{MAE - codebook} = \boldsymbol{M} = \begin{bmatrix} m_{11} & m_{12} & \cdots & m_{1n} \\ m_{21} & m_{22} & \cdots & m_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ m_{N1} & m_{N2} & \cdots & m_{Nn} \end{bmatrix} = [m_{uv}]_{N \times n} \qquad \textbf{(4.21)}$$

La columna $(m_{u,1}, m_{u,2}, \cdots, m_{u,n})$ de $\boldsymbol{M}$ representa el centroide de la clase u, donde $u = 1,2,3,\cdots,N$. Es decir, el peso sináptico, $m_{u,v}$, es una función ϕ de todos los elementos pertenecientes a la clase u. Estos pesos sinápticos determinan el comportamiento de la red.

4.4 Cuantificación vectorial VQ-MAE

En un proceso VQ basado en un criterio de similitud entre vectores de entrada y vectores de reconstrucción, cada vector de entrada es reemplazado por el índice del vector de reconstrucción que presenta la correspondencia más cercana.

En contraste con el proceso que genera al **MAE-Codebook**, la cuantificación vectorial usando el algoritmo VQ-MAE es un proceso en demasía simple. Debido a que la generación del **MAE-Codebook** es realizada usando el proceso de aprendizaje de las MAE, entonces, el VQ-MAE es llevado a cabo usando el proceso de clasificación de las MAE.

La matriz $\boldsymbol{M} = [m_{i,j}]_{N \times n}$ es el **MAE-codebook**. Cuando un patrón de entrada $\mathbf{x}^i \in \boldsymbol{R}^n$ es presentado al **MAE-codebook**, el índice de la clase a la que pertenece es generado. Cuando se usa el operador **prom** o **pmed** en la generación del **MAE-codebook**, el índice de la clase a la cual pertenece **x** está dado por:

$$i = \arg_{l} \left[\bigwedge_{l=1}^{N} \bigvee_{j=1}^{n} \left| m_{l,j} - x_j \right| \right]$$

La operación morfológica $\bigvee_{j=1}^{n} \left| m_{l,j} - x_j \right| \equiv d(\mathbf{x}, m_l)$ es la métrica que indica la distancia entre cada fila de $\boldsymbol{M}$ y la característica del vector **x** que se refiere al grado de correspondencia de la característica del vector **x** con cada una de las N clases, esto es:

$$d(\mathbf{x}, m_l) = (\left| m_{l,1} - x_1 \right|) \vee (\left| m_{l,2} - x_2 \right|) \vee \cdots \vee (\left| m_{l,n} - x_n \right|) \qquad \textbf{(4.22)}$$

Por otro lado, cuando se usa el operador **med** en la generación del **MAE-codebook**, la región (índice de clase) a la cual el vector **x** pertenece, se determina usando la *mediana* y la operación morfológica *min* está definida por:

$$i = \arg_{l} \left[\bigwedge_{l=1}^{N} \left| \operatorname{med}_{j=1}^{n} m_{l,j} - \operatorname{med}_{j=1}^{n} x_j \right| \right]$$

Aquí $\left| \operatorname{med}_{j=1}^{n} m_{l,j} - \operatorname{med}_{j=1}^{n} x_j \right| \equiv d(\mathbf{x}, m_l)$ es la métrica que indica el grado de correspondencia en la característica del vector **x** con cada una de las *N* clases.

Considerando que $m_{l,1}, m_{l,2}, \cdots, m_{l,n}$ y $x_1, x_2, \cdots, x_n$ son conjuntos de valores finitos y cualquiera de los dos en orden ascendente o descendente, la mediana tiene dos casos: si *n* es un número impar, entonces la mediana es el valor definido como $m_{l,((n+1)/2)}$ y $x_{((n+1)/2)}$, si *n* es un número par, entonces la *mediana* es el valor definido como $(m_{l,(n/2)} + m_{l,((n/2)+1)})/2$ y $(x_{n/2} + x_{(n/2)+1})/2$, entonces:

$$d(\mathbf{x}, m_l) = \left| m_{l,((n+1)/2)} - x_{((n+1)/2)} \right|$$
$$d(\mathbf{x}, m_l) = \left| (m_{l,(n/2)} + m_{l,((n/2)+1)})/2 - (x_{n/2} + x_{(n/2)+1})/2 \right| \quad \textbf{(4.23)}$$

Cuando se usa el operador **sum** en la generación del **MAE-codebook**, el índice de la clase a la cual pertenece $\mathbf{x}$ está dado por:

$$i = \arg_l \left[\bigwedge_{l=1}^{N} \bigvee_{j=1}^{n} \left| m_{l,j} - r_{l,j} \right| \right]$$

donde $r_{i,j} = x_j + \gamma_{l,j}$ ó $r_{i,j} = x_j + \lambda_{l,j}$.

Los cálculos de $d(\mathbf{x}, m_l)$ son simples cuando se utiliza el operador **prom** porque los valores de **x** y m_l no deben ser ordenados, este hecho implica que la velocidad de procesamiento es más alta. Por el otro lado, cuando se usa el operador **med**, los valores extremos no tienen efectos de importancia en los resultados.

El resultado de los cálculos de $d(\mathbf{x}, m_l)$ para los cuatro casos, los operadores **prom**, **med**, **pmed** y **sum** son un vector columna donde cada elemento indica el grado de correspondencia del vector de entrada **x**, con cada una de las *N* clases.

$$\begin{bmatrix} d(\mathbf{x}, m_1) \\ d(\mathbf{x}, m_2) \\ \vdots \\ d(\mathbf{x}, m_N) \end{bmatrix} \quad \textbf{(4.24)}$$

Finalmente, para establecer cual clase corresponde a un nuevo vector de entrada, se aplica el operador *min* al vector definido en la expresión (4.24). La clase se indica por medio del índice de la fila de la matriz ***M*** que presenta el valor más próximo con el vector de entrada **x**.

$$i = \arg_l \left| \bigwedge_{l=1}^{N} d(\mathbf{x}, m_l) \right| \quad \textbf{(4.25)}$$

El proceso de VQ finaliza cuando todos los vectores de entrada (bloques de la imagen) se han remplazado por el índice de clase a la cual les corresponde.

En la mayor parte del procesado de datos, el algoritmo propuesto hace uso de las operaciones morfológicas correspondientes a sumas y comparaciones. Por lo tanto, el algoritmo propuesto ofrece una velocidad de procesamiento muy alta. La Figura 4.6 muestra la estructura de la memoria asociativa que implementa un algoritmo de búsqueda de alta velocidad.

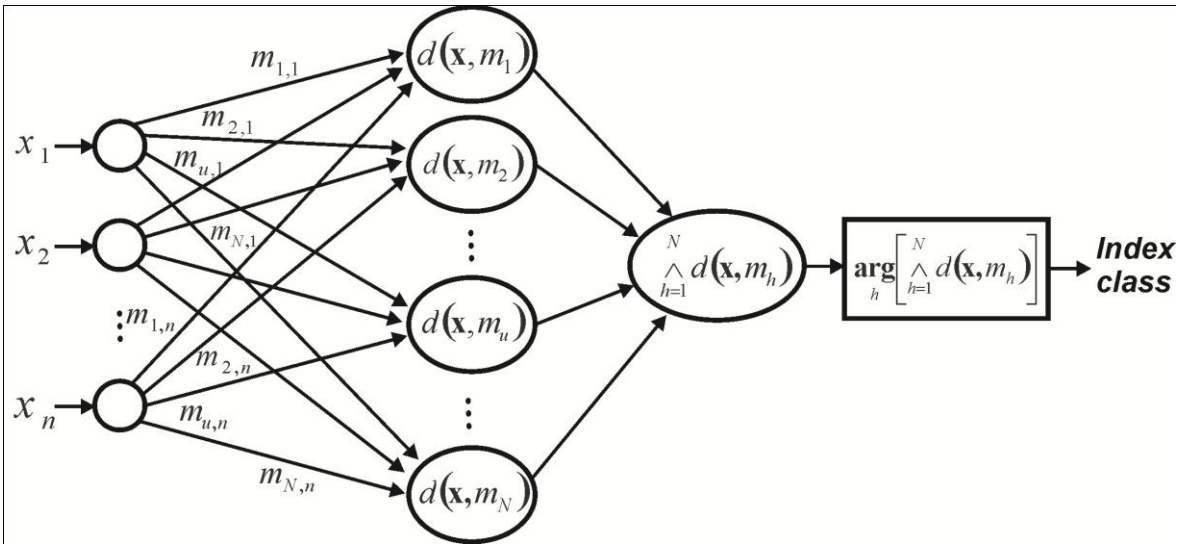

Figura 4.6. Estructura del algoritmo de búsqueda de alta velocidad basado en MAE.

Otra forma válida de representar el algoritmo propuesto es el que se muestra en la Figura 4.7 particularizando a cada uno de los operadores que pueden intervenir en el algoritmo propuesto de la VQ-MAE.

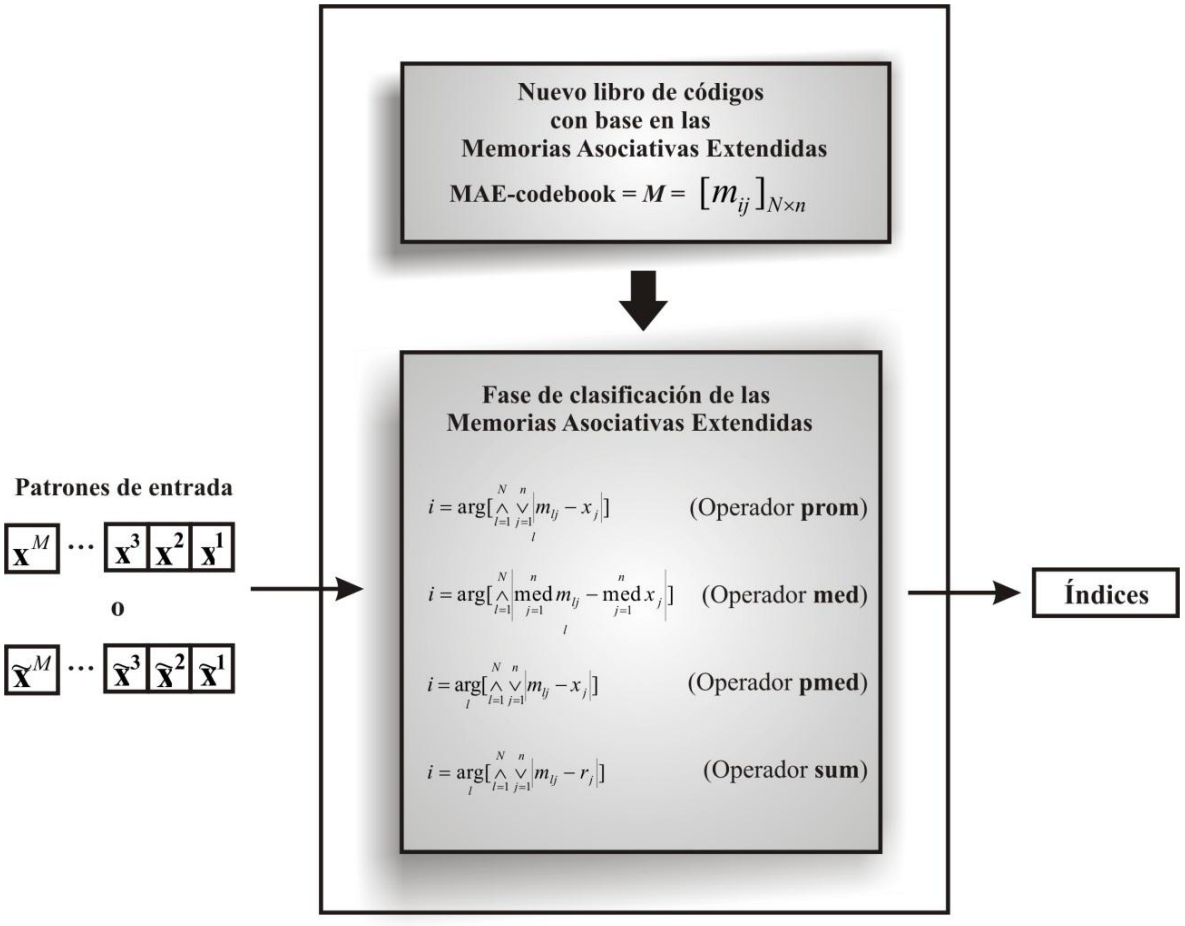

Figura 4.7. Esquema del algoritmo de búsqueda rápida basado en MAE.

Cuando un patrón no usado en la construcción de **MAE-codebook** es presentado al proceso de cuantificación vectorial VQ-MAE, éste tiene la propiedad de asignar el índice de la fila de $\boldsymbol{M}$ que presenta la correspondencia más cercana al patrón presentado. Esta propiedad permite al **VQ-MAE** cuantificar de forma eficiente los patrones que no pertenezcan al conjunto de entrenamiento.

4.5 Complejidad del algoritmo VQ-MAE

En un sentido amplio, si existe un problema y también existe un dispositivo donde resolverlo es necesario proporcionar un *método* preciso que lo solucione acorde al dispositivo. A tal método se le define como *algoritmo*. Una vez que se tiene diseñado el algoritmo de forma correcta y concreta, es imprescindible definir los criterios que permitan medir su eficiencia, la cual evalúe y mida con dos parámetros distintos el costo de consumo del algoritmo al ejecutarse, tanto en *tiempo* (cantidad de tiempo que tarda en ejecutarse) como en *espacio* (cantidad de memoria que utiliza el algoritmo al ejecutarse) para llegar a la solución y además ofrecer la posibilidad de comparar distintos algoritmos que resuelven el mismo problema. Por lo tanto, intrínsecamente nacen los conceptos de *complejidad en tiempo* y *complejidad en espacio.*

El tiempo de ejecución de un algoritmo depende de diversos factores como son: los datos de entrada que se le suministren, la calidad del código generado por el compilador para crear el programa objeto, la naturaleza y rapidez de las instrucciones máquina del procesador concreto que ejecute el programa, y la complejidad intrínseca del algoritmo.

Se entiende por *tamaño de la entrada*, al número de componentes sobre los que se va a ejecutar el algoritmo. Por ejemplo, la dimensión del vector a ordenar ó el tamaño de las matrices a multiplicar.

Respecto a la complejidad en tiempo, existen dos casos de estudio posibles:

- Tiempo de ejecución. Consiste en medir el tiempo de ejecución en función del número de operaciones que el algoritmo ejecuta.
- Cota de complejidad. Permite determinar el *orden de crecimiento* del algoritmo.

La unidad de tiempo utilizada en estas medidas de eficiencia no puede ser expresada en segundos u otra unidad de tiempo concreta, pues no existe un equipo de cómputo estándar al que pueda hacerse referencia todas las medidas. Por lo anterior, se denotará a $T(n)$ como el tiempo de ejecución de un algoritmo para una entrada de tamaño n.

Teóricamente $T(n)$, indica el número de instrucciones realizadas por un sistema de cómputo ideal. Se deben buscar por tanto medidas simples y abstractas, independientes al sistema de cómputo utilizado. Para ello es necesario acotar las posibles diferencias que se puedan presentar entre distintas implementaciones de un mismo algoritmo, ya sea del mismo código ejecutado por dos máquinas de distintas velocidades, como de dos códigos que implementen el mismo método (algoritmo) [24]. Esta diferencia es la que acota el siguiente principio.

Principio de Invariancia.

Dado un algoritmo y dos implementaciones suyas I_1 e I_2, que tardan $T_1(n)$ y $T_2(n)$ tiempos respectivamente, el principio de invariancia afirma que existen una constante real $c > 0$ y un número natural n_0 tales que para todo $n \geq n_o$ se verifica que $T_1(n) \leq cT_2(n)$. Es decir, el

tiempo de ejecución de dos implementaciones distintas de un algoritmo dado no va a diferir más que en una constante multiplicativa. Con esto se puede definir sin problemas que un algoritmo tarda un tiempo del orden de $T(n)$ si existe una constante real $c > 0$ y una implementación *I* del algoritmo que tarda menos que $cT(n)$, para todo *n* tamaño de entrada.

Existen 3 posibles casos que se pueden analizar para determinar la complejidad en el tiempo de ejecución de un mismo algoritmo: *caso peor*, *caso mejor* y *caso medio*.

El caso mejor corresponde a la traza (secuencia de instrucciones) del algoritmo que realiza menos operaciones. Análogamente, el caso peor corresponde a la traza del algoritmo que realiza más operaciones. Respecto al caso medio, corresponde a la traza del algoritmo que realiza un número de instrucciones igual a la esperanza matemática de la variable aleatoria definida por todas las posibles trazas del algoritmo para un tamaño de entrada dado, con las probabilidades de que éstas ocurran para esa entrada.

La medición del tiempo de ejecución se hace en función del número de *operaciones elementales* (OE) que realiza el algoritmo. Las OE son aquellas que una computadora realiza en un tiempo acotado por una constante. Se consideran como OE las operaciones aritméticas básicas, asignación a variables de tipo predefinido por el compilador, los saltos (llamadas a funciones y procedimientos, retorno de ellos), comparaciones lógicas y el acceso a estructuras indexadas básicas, como son los vectores y matrices. Cada una de ellas se contabiliza como 1 OE.

Resumiendo, el tiempo de ejecución de un algoritmo va a ser una función que mide el número de operaciones elementales que realiza el algoritmo para un tamaño de entrada dado.

Una vez determinada la forma de calcular el tiempo de ejecución *T* de un algoritmo es posible clasificarlo con la finalidad de tener un parámetro comparativo. Para ello se hace uso de clases de equivalencia correspondientes a funciones que "crecen de la misma forma".

Existen varias cotas que permiten caracterizar la complejidad de un algoritmo; una de las más usadas es la función asintótica O(*omicrón*), empleada como cota superior. Dada una función $f(n)$, se desea estudiar aquellas funciones $g(n)$ que a lo sumo crecen tan de prisa como $f(n)$. Al conjunto de tales funciones se le llama cota superior de $f(n)$ y se representa por $O(f(n))$. Conociendo la cota superior de un algoritmo se puede asegurar que, en ningún caso, el tiempo empleado por éste será de un orden superior al de la cota. La función O es definida a continuación.

En las siguientes definiciones **N** denotará el conjunto de los números naturales y **R** el de los reales.

Cota superior, notación O. Sea $f(n): \mathbf{N} \to [0, \infty)$, se define el conjunto de funciones O de $f(n)$ como [14], [24]:

$$O(f(n)) = \{g(n): \mathbf{N} \to [0,\infty) \mid \exists\ c \in \mathbf{R},\ \ c > 0,\ \ \exists n_0 \in \mathbf{N} \bullet g(n) \le cf(n)\ \ \forall\ \ n \ge n_0\} \qquad \textbf{(4.26)}$$

Se dice entonces, que una función $t(n): \mathbf{N} \to [0, \infty)$ es de orden O de $f(n)$ sí $t(n) \in O(f(n))$.

A continuación, se presentan los análisis de complejidad temporal y espacial del algoritmo propuesto. Con este fin, se hace uso de los pseudocódigos del VQ-MAE para los operadores **prom**, **pmed** y **med** mostrados en el Algoritmo1(a), (b) de la Tabla 4.1. Cabe hacer mención que no se han implementado variantes más eficientes, en la etapa de ordenamiento, para el VQ-MAE cuando se usa un operador **med** debido a que esto no repercutiría en los resultados de PSNR y relación de compresión donde la variante que usa el operador **prom** presenta un mejor desempeño.

Tabla 4.1. Algoritmo 1 Pseudocódigos del algoritmo VQ-MAE (a) Operadores **prom** y **pmed**, (b) Operador **med**.

(a)

```
      // prom and pmed operators
01| subroutine prom()
02| variables
03|    x,k,j,l,aux: integer
04|    arg[N]='0': integer
05| begin
06|     for k←1 to N [operations k=k+1] do
07|        for j←1 to n [operations j=j+1]) do
08|            aux=abs(prom_codebook[k][j]-x[j]);
09|            if (aux>arg[k]) then
10|                arg[k]=aux;
11|            end_if
12|        end_for
13|     end_for
14|     aux=max(x);
15|     for l←1 to N [operations l=l+1] do
16|         if(arg[l]<aux)
17|            aux=arg[l];
18|            index=l;
19|         end_if
20|     end_for
21| end_subroutine
```

(b)

```
      // med operator
01| subroutine med()
02| variables
03|    z,aux,w,aux2,index: integer
04|    median: float
05| begin
      // Arranges the vector input elements
06|     for z←1 to n [operations z=z+1] do
07|        aux=x[z];
08|        w=z-1;
09|        while (w>=0 && x[w]>aux) do
10|            x[w+1]=x[w];
11|            w=w-1;
12|        end_while
13|        x[w+1]=aux;
14|     end_for
      // Compute the median
15|    if (n%2==0) then
16|        median=(x[n/2]+x[(n/2)-1])/2;
17|    else
18|        median=x[n/2];
19|    end_if
     // Compute an element of M
20|     aux2=max(x);
21|     for k←1 to N [operations k=k+1] do
22|         aux=abs(med_codebook[k]-median);
23|         if(aux<aux2)
24|            aux2=aux;
25|            index=k;
26|         end_if
27|      end_for
28| end_subroutine
```

4.5.1 Complejidad temporal del algoritmo VQ-MAE

Para medir la complejidad en tiempo del algoritmo, primero se obtiene el tiempo de ejecución basado en el número de operaciones elementales (OE) que el algoritmo propuesto realiza para clasificar un vector de entrada, la subrutina `Prom()` es el elemento más representativo que lleva a cabo esta tarea. Para los operadores **prom** y **pmed**, se toma el pseudocódigo del Algoritmo 1(a), tomando en cuenta las siguientes consideraciones:

- En la línea 06 se realizan 3 OE (una asignación, una comparación y una suma) para cada iteración de la primera estructura cíclica *for*, más otras 3 OE cuando se realiza la salida del mismo.
- En la línea 07 sucede lo mismo, realiza 3 OE en cada iteración de la segunda estructura cíclica *for* (anidada), más otras 3 OE cuando se realiza la salida de la estructura *for*.
- En la línea 08 se realizan 5 OE: 2 accesos al vector, 1 resta, 1 llamada a función y 1 asignación.
- En la línea 09 se realizan 2 OE: 1 acceso al vector y 1 comparación.
- En la línea 10 se realizan 2 OE: 1 asignación y 1 acceso al vector.
- En la línea 14 se realizan 2 OE: 1 llamada a función y 1 asignación.
- En la línea 15 se realizan 3 OE en cada iteración de la estructura cíclica *for*, más otras 3 OE cuando se realiza la salida de la estructura *for*.
- En la línea 16 se realizan 2 OE: 1 acceso al vector y 1 comparación.
- En las líneas 17 y 18 se realizan 3 OE: 1 acceso al vector y 2 asignaciones.

En base a las consideraciones anteriores se desprende el siguiente análisis:

En el *caso peor*, las condiciones de las líneas 9 y 16 siempre serán verdaderas, por lo tanto, las líneas 10, 17 y 18 se ejecutarán en todas las iteraciones, y entonces el ciclo interno de la línea 7 controlado por el índice *j=1* hasta *n*; y el ciclo de la línea 15 controlado por el índice *l=1* hasta *N* realiza el siguiente número de OE:

$$\left(\sum_{j=1}^{n}(9+3)\right)+3=12\left(\sum_{j=1}^{n}1\right)+3=12n+3$$

$$\left(\sum_{l=1}^{N}(5+3)\right)+3=8\left(\sum_{l=1}^{N}1\right)+3=8N+3$$

El siguiente ciclo de la línea 6 repetirá $12n+3$ OE en cada iteración:

$$\left(\sum_{k=1}^{N}(12n+3)+3\right)+3=\left(\sum_{k=1}^{N}12n+6\right)+3=N(12n+6)+3=12Nn+6N+3$$

Por lo tanto, la ecuación (4.27) define el número total de OE que realiza el algoritmo.

$$T(n)=(12Nn+6N+3)+(8N+3)+2=12Nn+14N+8 \qquad \textbf{(4.27)}$$

En el *caso mejor*, las condiciones de las líneas 9 y 16 siempre serán falsas, por lo tanto, las líneas 10, 17 y 18 nunca se ejecutarán en todas las iteraciones, y entonces el ciclo interno de

la línea 7 controlado por el índice *j=1* hasta *n*; y el ciclo de la línea 15 controlado por el índice *l=1* hasta *N* realiza el siguiente número de OE respectivamente:

$$\left(\sum_{j=1}^{n}(7+3)\right)+3=10\left(\sum_{j=1}^{n}1\right)+3=10n+3$$

$$\left(\sum_{l=1}^{N}(2+3)\right)+3=5\left(\sum_{l=1}^{N}1\right)+3=5N+3$$

El siguiente ciclo de la línea 6 repetirá $10n+3$ OE en cada iteración:

$$\left(\sum_{k=1}^{N}(10n+3)+3\right)+3=\left(\sum_{k=1}^{N}10n+6\right)+3=N(10n+6)+3=10Nn+6N+3$$

Por lo tanto, la siguiente ecuación define el número total de OE que realiza el algoritmo.

$$T(n)=(10Nn+6N+3)+(5N+3)+2=10Nn+11N+8 \qquad \textbf{(4.28)}$$

Donde *N* es el tamaño del libro de códigos (codebook), y *n* es la dimensión del patrón.

Una vez que se ha expresado el algoritmo en base a las OE que realiza la función $f(n)$, el siguiente paso es determinar cuál es su orden de crecimiento. Para este propósito se hace uso de la cota superior ***O***.

Considerando el *peor caso* y fijando a *N*=16, el número de OE que realiza el algoritmo es:

$$\begin{aligned} f(n)&=12(16)n+14n+8=192n+14n+8\\ f(n)&=206n+8 \end{aligned} \qquad \textbf{(4.29)}$$

Para determinar el orden de crecimiento de la función $f(n)$ se hace uso de las reglas definidas en [24].

1. Si $f(n)$ es un polinomio de grado *x*, entonces $f(n)$ es $O(n^x)$ como *x*=1; por lo tanto $f(n)$ se simplifica a $O(n)$.
   i. Prescindir de los términos de orden menor.
   ii. Prescindir de los factores constantes.
2. Usar la clase más pequeña posible de funciones, es decir, "$2n$ es $O(n)$" en vez de "$2n$ es $O(n^2)$".
3. Usar la expresión más simple de la clase, es decir, "$3n+5$ es $O(n)$" en vez de "$3n+5$ es $O(3n)$".

Con base en estas reglas se concluye que el orden de crecimiento del algoritmo propuesto es $O(n)$.

Ahora, para medir la complejidad en tiempo del algoritmo 1(b) para el operador **med**, nuevamente se obtiene el tiempo de ejecución basado en el número de OE que el algoritmo propuesto realiza para clasificar un vector de entrada, la subrutina `med()` es el elemento

más representativo que lleva a cabo esta tarea con este operador. A continuación se toman en cuenta las siguientes consideraciones:

- En la línea 06 se realizan 3 OE: (una asignación, una comparación y una suma) para cada iteración de la estructura cíclica *for*, más otras 3 OE cuando se realiza la salida del mismo.
- En la línea 07 se realizan 2 OE: 1 acceso al vector y 1 asignación.
- En la línea 08 se realizan 2 OE: 1 resta y 1 asignación.
- En la línea 09 se realizan 5 OE: (1 acceso al vector, 3 comparaciones y 1 comparación lógica) para cada iteración de la estructura cíclica *while*.
- En la línea 10 se realizan 4 OE: 1 suma, 2 accesos al vector y 1 asignación.
- En la línea 11 se realizan 2 OE: 1 resta y 1 asignación.
- En la línea 12 se realizan 1 OE: 1 retorno de loop while.
- En la línea 13 se realizan 3 OE: 1 suma, 1 acceso al vector y 1 asignación.
- En la línea 15 se realizan 2 OE: 1 división-módulo y 1 comparación.
- En la línea 16 se realizan 8 OE: 3 divisiones, 1 resta, 2 accesos al vector, 1 suma, y 1 asignación.
- En la línea 18 se realizan 3 OE: 1 división, 1 acceso al vector y 1 asignación.
- En la línea 20 se realizan 2 OE: 1 llamada a función y 1 asignación.
- En la línea 21 se realizan 3 OE: (una asignación, una comparación y una suma) para cada iteración de la estructura cíclica *for*, más otras 3 OE cuando se realiza la salida del mismo.
- En la línea 22 se realizan 4 OE: 1 acceso al vector, 1 resta, 1 llamada a función y 1 asignación.
- En la línea 23 se realizan 1 OE: 1 comparación.
- En la línea 24 se realizan 1 OE: 1 asignación.
- En la línea 25 se realizan 1 OE: 1 asignación.

En base a las consideraciones anteriores se desprende el siguiente análisis:

En el *caso peor*, se presenta cuando las condiciones de las líneas 15 y 23 siempre serán verdaderas y el *"loop while"* será ejecutado *z* veces en cada iteración. Del Algoritmo 1(b), se pueden distinguir 3 fases: "arreglos de los elementos del vector de entrada", "cálculo de la mediana" y "cálculo de los elementos de $\boldsymbol{M}$".

La fase del arreglo de los elementos del vector de entrada realiza el siguiente número de OE:

$$\left(\sum_{z=1}^{n}(7+3)+3\right)+3=13\left(\sum_{z=1}^{n}1\right)+3=13n+3, \quad \text{sin} \quad \text{"}loop \quad while\text{"}$$

$$\frac{n(n+1)}{2}(12)=6n(n+1)=6n^2+6n, \quad sólamente\ con\ el \quad \text{"}loop \quad while\text{"}$$

La fase del cálculo de la mediana realiza 10 OE y la fase del cálculo de los elementos de $\boldsymbol{M}$ realiza el siguiente número de OE:

$$\left(\sum_{k=1}^{N}(7+3)+3\right)+3+2=13\sum_{k=1}^{N}1+5=13N+5$$

Finalmente, la ecuación (4.30) define el número total de OE que realiza el algoritmo.

$$T(n) = ((13n+3)+(6n^2+6n))+(10)+(13N+5) = 6n^2+19n+13N+18 \quad \textbf{(4.30)}$$

En el *caso mejor*, se presenta cuando las condiciones de las líneas 15 y 23 siempre serán falsas y el *"loop while"* nunca será ejecutado las *z* veces de cada iteración. De las 3 fases: identificadas anteriormente "arreglos de los elementos del vector de entrada", "cálculo de la mediana" y "cálculo de los elementos de ***M***" se contabilizan las siguientes OE.

La fase del arreglo de los elementos del vector de entrada realiza el siguiente número de OE:

$$\left(\sum_{z=1}^{n}(7+3)+3\right)+3 = 13\left(\sum_{z=1}^{n}1\right)+3 = 13n+3, \quad \text{sin} \quad \text{"}loop \quad while\text{"}$$

La fase del cálculo de la mediana realiza 5 OE y la fase del cálculo de los elementos de ***M*** realiza el siguiente número de OE:

$$\left(\sum_{k=1}^{N}(7+1)+3\right)+3+2 = 11\sum_{k=1}^{N}1+5 = 11N+5$$

Finalmente, la siguiente ecuación define el número total de OE que realiza el algoritmo.

$$T(n) = (13n+3)+(5)+(11N+5) = 13n+11N+13 \quad \textbf{(4.31)}$$

Donde *N* es el tamaño del libro de códigos (codebook), y *n* es la dimensión del patrón.

Una vez que se ha expresado el algoritmo en base a las OE que realiza la función $f(n)$, el siguiente paso es determinar cuál es su orden de crecimiento. Para este propósito se hace uso de la cota superior ***O***.

Considerando el *peor caso* y fijando a *N*=128, el número de OE que realiza el algoritmo es:

$$\begin{aligned} f(n) &= 6n^2+19n+13(128)+18 = 6n^2+19n+1664+18 \\ f(n) &= 6n^2+19n+1682 \end{aligned} \quad \textbf{(4.32)}$$

Ahora, considerando la expresión 4.32 y haciendo uso nuevamente de las reglas definidas en [24], se concluye que el orden de crecimiento del algoritmo propuesto es $O(n^2)$.

También, se analiza el número y tipo de operaciones aritméticas usadas por el algoritmo durante el proceso de VQ de una imagen. Entonces, para los operadores **prom** y **pmed**, se considera el pseudocódigo del Algoritmo 1(a). Además, se conoce que este proceso se aplica a una imagen de tamaño $hi \times wi$. Por tanto, el número de operaciones que el algoritmo propuesto necesita usando los operadores **prom** o **pmed** para cuantificar una imagen depende del tamaño de la imagen, el tamaño del libro de códigos *N* y la dimensión del patrón *n*, ecuación (4.33).

$$\left(\frac{hi \times wi}{n}\right)[Nn(op2) + N(op1)] \quad \textbf{(4.33)}$$

donde $op1 = 1$ comparación y $op2 = 1$ suma y 1 comparación; $(hi \times wi)/n$ es el número de patrones a cuantificar.

$Nn(op2)$ es el número y tipo de operaciones aritméticas utilizadas para ejecutar una dilatación morfológica sobre la diferencia absoluta de los elementos del **MAE-codebook** y los componentes del patrón de entrada; $N(op1)$ es el número y tipo de operaciones aritméticas utilizadas para ejecutar una erosión morfológica sobre los resultados del proceso previo.

Ahora, para el operador **med**, se considera el pseudocódigo del Algoritmo 1(b). Por lo tanto, el número de operaciones que el algoritmo propuesto necesita usando el operador **med** para cuantificar la imagen depende del tamaño de la imagen, el tamaño del libro de códigos *N*, la dimensión del patrón *n* y las operaciones de la *mediana*.

$$\left(\frac{hi \times wi}{n}\right)[n(op1) + (op2) + N(op3)] \quad \textbf{(4.34)}$$

donde $op1 = 2$ sumas y 2 comparaciones, $op2 = 2$ sumas, 1 comparación y 3 corrimientos, $op2 = 1$ suma y 1 comparación.

$n(op1)$ es el número y tipo de operaciones aritméticas utilizadas para acomodar los elementos del vector de entrada; $(op2)$ es el número y tipo de operaciones aritméticas utilizadas para calcular la *mediana* y $N(op3)$ es el número y tipo de operaciones aritméticas utilizadas para calcular un elemento de $\boldsymbol{M}$.

4.5.2 Complejidad espacial del algoritmo VQ-MAE

Referirse a la *complejidad en espacio* del algoritmo se dice que se desea medir la cantidad de memoria requerida para su ejecución. La complejidad en espacio tiene por función determinar la cantidad de memoria del sistema que utiliza el algoritmo en la solución del problema planteado, dependiendo exclusivamente del tipo de datos utilizados por el algoritmo.

Para cuantificar una imagen de tamaño $hi \times wi$, la cual contiene $\boldsymbol{M} = (hi \times wi)/n$ patrones *n*-dimensionales, la implementación del algoritmo propuesto para los operadores **prom** y **pmed**, requieren dos vectores $arg[N]$ y $indexes[M]$, y una matriz $prom\_codebook[N][n]$. Por lo tanto, el número de unidades de memoria (um) requeridas para este proceso es:

$$\text{um}\_arg + \text{um}\_indexes + \text{um}\_prom\_codebook = N + M + N(n) = M + N(n+1) \quad \textbf{(4.35)}$$

Cuando se utiliza el operador **med**, el algoritmo solo requiere dos vectores $indexes[M]$ y $med\_codebook[N]$. Por lo tanto, el número de (um) requeridas son:

$$\text{um}\_indexes + \text{um}\_med\_codebook = M + N \quad \textbf{(4.36)}$$

Para imágenes en escala de grises (8 bits/pixel), las variables `arg`, `prom_codebook` y `med_codebook` están declaradas de tipo byte, en la que la variable `indexes` depende del tamaño del libro de códigos. Por lo tanto, si $N \leq 256$ entonces `indexes` es de tipo byte y si $N > 256$ entonces `indexes` es de tipo short (número enteros con signo de 16 bits).

Entonces, el número total de bytes requeridos para la implementación del algoritmo usando los operadores **prom** y **pmed** es:

$$\begin{aligned} &M + N(n+1) \quad si \quad N \leq 256 \\ &2M + N(n+1) \quad si \quad N > 256 \end{aligned} \quad \textbf{(4.37)}$$

Y, el número total de bytes requeridos para la implementación del algoritmo con el operador **med** es:

$$\begin{aligned} &M + N \quad si \quad N \leq 256 \\ &2M + N \quad si \quad N > 256 \end{aligned} \quad \textbf{(4.38)}$$

El número de unidades de memoria dependen del tamaño de la imagen, el tamaño del libro de códigos y el tamaño de los vectores de reconstrucción elegidos para el proceso de VQ.

Capítulo 5

Resultados experimentales

Esta sección presenta los resultados experimentales obtenidos de aplicar el esquema de cuantificación vectorial propuesto (VQ-MAE) a una imagen a la vez. Primeramente se presenta el desempeño del algoritmo propuesto cuando se utilizaron los operadores **prom**, **med**, **pmed** y **sum**. Entonces, se compara el desempeño del algoritmo propuesto con el algoritmo LBG tradicional. Los parámetros evaluados son, la *distorsión* generada en la imagen reconstruida y la *influencia* que el proceso de VQ tiene en la compresión de la imagen cuando se usan diversos métodos de codificación. Finalmente, se analiza el número y tipo de operaciones así como la cantidad de memoria utilizada por el algoritmo propuesto y el algoritmo LBG. Para evaluar estos parámetros se usaron un conjunto de imágenes de prueba estándar de tamaño de 512 x 512 pixeles con una escala de gris de 256 tonos (Figura 5.1).

Figura 5.1. Conjunto de imágenes de prueba (a) Lena, (b) Peppers, (c) Elaine, (d) Man, (e) Barbara, (f) Baboon.

Para medir el rendimiento de ambos algoritmos, tanto el propuesto como el LBG, se utilizó el criterio de rendimiento objetivo llamado parámetro de relación señal a ruido pico (peak signal-to-noise ratio, PSNR), que está definido como:

$$PSNR = 10 \ \log_{10}\left(\frac{(2^n - 1)^2}{\frac{1}{M}\sum_{i=1}^{M}(p_i - \tilde{p}_i)^2}\right) \qquad \textbf{(5.1)}$$

Donde *n* es el número de bits por pixel, M es el número de pixeles en la imagen, p_i es el *i*-ésimo pixel en la imagen original, y $\tilde{p}_i$ es el *i*-ésimo pixel en la imagen reconstruida.

El primer experimento tuvo el propósito de determinar el rendimiento que el algoritmo propuesto presenta con cada uno de los operadores (**prom**, **med**, **pmed** y **sum**) cuando se aplica en las imágenes de prueba. La Tabla 5.1 muestra la distorsión que cada operador agrega a la imagen durante el proceso de cuantificación cuando se usan diferentes tamaños de libros de códigos (Codebook). Los resultados muestran que el operador **prom** genera la menor distorsión cuando se aplica el algoritmo a las imágenes de prueba.

En este experimento, se generó el **MAE-codebook** utilizando cada una de las imágenes del conjunto de prueba individualmente, resultando el codebook generado con la imagen Lena el que presentó un mejor desempeño durante el proceso de cuantificación. Los resultados mostrados en la Tabla 5.1 fueron obtenidos cuando la imagen Lena fue utilizada para generar el **MAE-codebook**.

Para minimizar la distorsión generada en el proceso de cuantificación de las imágenes que no se usaron para generar el **MAE-codebook**, es posible usar un método evolutivo que permite agregar información de nuevas imágenes al libro. Este aspecto es el tema de trabajos futuros basados en investigaciones donde de los autores propusieron el diseño de un libro de códigos evolutivo usando memorias asociativas morfológicas [25].

Tabla 5.1. Desempeño del algoritmo propuesto con los operadores prom, med, pmed y sum (la imagen Lena se utilizó para generar el codebook)

		Algoritmo propuesto VQ-MAE Tamaño del Codebook			
		64	**128**	**256**	**512**
Operador	**Imágenes**	**PSNR**	**PSNR**	**PSNR**	**PSNR**
prom	Lena	26.4166	27.4702	28.3959	29.2524
	Peppers	25.2016	26.0691	26.5467	27.0489
	Elaine	28.3715	29.1042	29.6899	30.1218
	Man	23.2950	23.9506	24.5434	25.0942
	Barbara	21.3870	21.7271	22.0764	22.4347
	Baboon	18.1937	18.5259	18.8603	19.1508
med	Lena	18.2766	18.4610	18.5554	19.6281
	Peppers	18.1225	18.2008	17.9674	18.6897
	Elaine	18.8188	19.1405	19.1743	20.6147
	Man	17.5697	17.7426	17.7681	18.3887
	Barbara	17.0985	17.1972	17.1063	17.8108
	Baboon	14.5220	14.7361	14.5346	15.0243
pmed	Lena	24.5133	26.4137	27.6557	28.7976
	Peppers	23.5998	25.0989	26.1539	26.8024
	Elaine	25.6682	27.4973	28.7723	29.6578
	Man	22.0482	23.3018	24.3623	24.9216
	Barbara	20.8122	21.5167	22.0609	22.4012
	Baboon	18.1270	18.5977	18.9179	19.2329
sum matriz de recuperación **gamma**	Lena	10.1138	9.6350	9.3987	9.3573
	Peppers	11.0020	9.9208	9.5918	9.5217
	Elaine	10.8094	9.4984	8.8848	8.7020
	Man	9.7254	9.4844	9.4353	9.5057
	Barbara	9.6697	8.9992	8.7903	8.8261
	Baboon	11.0305	9.9756	9.3046	9.0201
sum matriz de recuperación **lamda**	Lena	9.9806	9.7097	9.6631	9.7185
	Peppers	9.1677	9.4578	9.6734	9.9001
	Elaine	9.7869	9.2173	8.7770	8.7429
	Man	10.3245	10.1555	10.1681	10.1290
	Barbara	10.2239	9.9361	9.8830	9.8589
	Baboon	8.9656	8.5817	8.3717	8.3949

El segundo experimento tiene como objetivo comparar el desempeño del algoritmo propuesto, con sus diferentes operadores **prom**, **med**, **pmed** y **sum**, con el del algoritmo LBG (método de cuantificación ampliamente utilizado). Para este propósito, el libro de códigos fue generado usando la imagen Lena como conjunto de entrenamiento y los resultados fueron obtenidos usando diferentes tamaños de libros de códigos. Entonces, se aplicó la VQ al conjunto de imágenes de prueba para determinar el comportamiento de los

algoritmos con patrones que no corresponden al conjunto de entrenamiento. La Tabla 5.2 muestra los resultados de este experimento.

Tabla 5.2. Comparación del desempeño del algoritmo propuesto y el algoritmo LBG.

	Algoritmo LBG				**Algoritmo propuesto VQ-MAE (operador prom)**			
	Tamaño del Codebook				**Tamaño del Codebook**			
	64	**128**	**256**	**512**	**64**	**128**	**256**	**512**
Imágenes	**PSNR**	**PSNR**	**PSNR**	**PSNR**	**PSNR**	**PSNR**	**PSNR**	**PSNR**
Lena	27.1448	28.2131	29.0855	29.9825	26.4166	27.4702	28.3959	29.2524
Peppers	26.3830	27.1805	27.6496	28.2132	25.2016	26.0691	26.5467	27.0489
Elaine	28.9191	29.6845	30.3035	30.7453	28.3715	29.1042	29.6899	30.1218
Man	24.2034	24.9395	25.4963	26.0266	23.2950	23.9506	24.5434	25.0942
Barbara	21.8144	22.2457	22.6960	23.1359	21.3870	21.7271	22.0764	22.4347
Baboon	18.8927	19.3438	19.6829	20.0105	18.1937	18.5259	18.8603	19.1508
	Algoritmo propuesto VQ-MAE (operador med)				**Algoritmo propuesto VQ-MAE (operador pmed)**			
	Tamaño del Codebook				**Tamaño del Codebook**			
	64	**128**	**256**	**512**	**64**	**128**	**256**	**512**
Imágenes	**PSNR**	**PSNR**	**PSNR**	**PSNR**	**PSNR**	**PSNR**	**PSNR**	**PSNR**
Lena	18.2766	18.4610	18.5554	19.6281	24.5133	26.4137	27.6557	28.7976
Peppers	18.1225	18.2008	17.9674	18.6897	23.5998	25.0989	26.1539	26.8024
Elaine	18.8188	19.1405	19.1743	20.6147	25.6682	27.4973	28.7723	29.6578
Man	17.5697	17.7426	17.7681	18.3887	22.0482	23.3018	24.3623	24.9216
Barbara	17.0985	17.1972	17.1063	17.8108	20.8122	21.5167	22.0609	22.4012
Baboon	14.5220	14.7361	14.5346	15.0243	18.1270	18.5977	18.9179	19.2329
	Algoritmo propuesto VQ-MAE (operador sum) con matriz de recuperación Gamma				**Algoritmo propuesto VQ-MAE (operador sum) con matriz de recuperación Lamda**			
	Tamaño del Codebook				**Tamaño del Codebook**			
	64	**128**	**256**	**512**	**64**	**128**	**256**	**512**
Imágenes	**PSNR**	**PSNR**	**PSNR**	**PSNR**	**PSNR**	**PSNR**	**PSNR**	**PSNR**
Lena	10.1138	9.6350	9.3987	9.3573	9.9806	9.7097	9.6631	9.7185
Peppers	11.0020	9.9208	9.5918	9.5217	9.1677	9.4578	9.6734	9.9001
Elaine	10.8094	9.4984	8.8848	8.7020	9.7869	9.2173	8.7770	8.7429
Man	9.7254	9.4844	9.4353	9.5057	10.3245	10.1555	10.1681	10.1290
Barbara	9.6697	8.9992	8.7903	8.8261	10.2239	9.9361	9.8830	9.8589
Baboon	11.0305	9.9756	9.3046	9.0201	8.9656	8.5817	8.3717	8.3949

De la Tabla 5.2, se pueden hacer las siguientes observaciones: 1) los resultados obtenidos muestran que el método propuesto es competitivo con el algoritmo LBG en el parámetro PSNR, 2) el algoritmo propuesto remplaza un patrón de entrada por el índice del vector de reconstrucción (codeword) que presenta su valor más cercano, no es necesario que uno de estos vectores de reconstrucción se hayan usado en la generación de la memoria $\boldsymbol{M}$. Esta propiedad le permite al algoritmo propuesto cuantificar eficientemente las imágenes que no se han usado en la generación del **MAE-codebook**.

En el tercer experimento, se evaluó el desempeño de diversos métodos de codificadores estándar aplicados a las imágenes cuantificadas mediante el algoritmo propuesto. Estos métodos incluyen técnicas de modelado estadístico tales como: *aritmética*, *Huffman*, *range*, transformación *Burrows Wheeler*, *PPM* y técnicas de diccionario como: *LZ77* y *LZP*. El propósito de este tercer experimento es analizar el desempeño del algoritmo propuesto aplicado a la compresión de imágenes. Para este propósito, se desarrolló un esquema de compresión de imágenes que incluye nuestro algoritmo, el algoritmo LBG, así como diversas técnicas de codificación de entropía. La Tabla 5.3 muestra los resultados de compresión obtenidos expresados como bits por pixel (bpp).

Tabla 5.3. Resultados de compresión obtenidos después de aplicar varias técnicas de codificación de entropía a la información generada del algoritmo propuesto.

Imagen	Técnica de codificación de entropia	Algoritmo LBG				Algoritmo Propuesto VQ-EAM (operador prom)			
		Tamaño del codebook				Tamaño del codebook			
		64	128	256	512	64	128	256	512
		bpp	bpp	bpp	bpp	bpp	bpp	bpp	bpp
Lena	PPM	0.209	0.262	0.339	0.462	0.214	0.269	0.348	0.470
	SZIP	0.217	0.269	0.347	0.464	0.224	0.276	0.356	0.472
	Burrown	0.225	0.276	0.359	0.475	0.227	0.284	0.368	0.483
	LZP	0.227	0.276	0.354	0.522	0.234	0.284	0.364	0.532
	LZ77	0.241	0.300	0.386	0.563	0.248	0.308	0.397	0.575
	Range	0.332	0.392	0.459	0.661	0.330	0.388	0.457	0.657
Peppers	PPM	0.185	0.234	0.307	0.414	0.194	0.245	0.323	0.429
	SZIP	0.196	0.243	0.317	0.415	0.206	0.255	0.333	0.431
	Burrown	0.267	0.254	0.329	0.429	0.215	0.264	0.346	0.446
	LZP	0.207	0.254	0.326	0.473	0.218	0.267	0.342	0.492
	LZ77	0.225	0.279	0.363	0.523	0.237	0.292	0.379	0.546
	Range	0.313	0.364	0.429	0.636	0.310	0.363	0.430	0.635
Elaine	PPM	0.168	0.216	0.290	0.394	0.174	0.225	0.304	0.410
	SZIP	0.179	0.227	0.301	0.403	0.185	0.237	0.316	0.417
	Burrown	0.190	0.236	0.313	0.419	0.191	0.245	0.328	0.433
	LZP	0.191	0.239	0.312	0.452	0.197	0.248	0.328	0.470
	LZ77	0.211	0.268	0.352	0.520	0.218	0.279	0.366	0.538
	Range	0.285	0.340	0.409	0.624	0.284	0.340	0.410	0.621
Man	PPM	0.240	0.307	0.386	0.493	0.244	0.312	0.393	0.499
	SZIP	0.255	0.320	0.395	0.493	0.259	0.325	0.402	0.499
	Burrown	0.259	0.322	0.402	0.502	0.261	0.328	0.410	0.507
	LZP	0.271	0.334	0.410	0.569	0.276	0.340	0.418	0.575
	LZ77	0.284	0.351	0.428	0.605	0.289	0.357	0.433	0.612
	Range	0.328	0.389	0.452	0.651	0.326	0.387	0.452	0.650
Barbara	PPM	0.229	0.292	0.366	0.474	0.228	0.291	0.374	0.488
	SZIP	0.246	0.307	0.379	0.480	0.244	0.306	0.387	0.491
	Burrown	0.251	0.312	0.389	0.490	0.251	0.311	0.398	0.503
	LZP	0.257	0.315	0.387	0.541	0.258	0.316	0.397	0.559
	LZ77	0.265	0.329	0.408	0.575	0.270	0.331	0.415	0.597
	Range	0.333	0.394	0.463	0.660	0.325	0.385	0.455	0.656
Baboon	PPM	0.284	0.356	0.443	0.536	0.281	0.360	0.449	0.544
	SZIP	0.302	0.371	0.450	0.536	0.299	0.375	0.455	0.542
	Burrown	0.301	0.373	0.473	0.547	0.298	0.376	0.477	0.552
	LZP	0.317	0.388	0.470	0.619	0.315	0.393	0.478	0.628
	LZ77	0.322	0.391	0.470	0.651	0.322	0.390	0.471	0.664
	Range	0.336	0.402	0.470	0.663	0.328	0.397	0.470	0.664

Tabla 5.3. Resultados de compresión obtenidos después de aplicar varias técnicas de codificación de entropía a la información generada del algoritmo propuesto (continuación).

Imagen	Técnica de codificación de entropia	Algoritmo Propuesto VQ-EAM (operador med)				Algoritmo Propuesto VQ-EAM (operador pmed)			
		Tamaño del codebook				Tamaño del codebook			
		64	128	256	512	64	128	256	512
		bpp	bpp	bpp	bpp	bpp	bpp	bpp	bpp
Lena	PPM	0.254	0.300	0.330	0.363	0.214	0.267	0.339	0.452
	SZIP	0.271	0.315	0.345	0.372	0.223	0.274	0.345	0.451
	Burrown	0.277	0.320	0.351	0.385	0.230	0.281	0.355	0.461
	LZP	0.283	0.327	0.355	0.419	0.234	0.283	0.356	0.512
	LZ77	0.292	0.339	0.370	0.487	0.248	0.305	0.382	0.559
	Range	0.338	0.375	0.398	0.560	0.314	0.371	0.436	0.641
Peppers	PPM	0.232	0.274	0.308	0.344	0.198	0.242	0.309	0.412
	SZIP	0.245	0.287	0.322	0.354	0.209	0.252	0.320	0.413
	Burrown	0.250	0.292	0.330	0.360	0.216	0.260	0.330	0.424
	LZP	0.259	0.301	0.333	0.398	0.223	0.266	0.331	0.474
	LZ77	0.275	0.320	0.358	0.473	0.241	0.289	0.362	0.527
	Range	0.327	0.362	0.392	0.560	0.302	0.349	0.411	0.619
Elaine	PPM	0.238	0.280	0.315	0.372	0.181	0.230	0.296	0.391
	SZIP	0.254	0.297	0.331	0.382	0.193	0.242	0.309	0.399
	Burrown	0.258	0.298	0.337	0.392	0.201	0.250	0.320	0.413
	LZP	0.269	0.311	0.345	0.429	0.207	0.255	0.322	0.449
	LZ77	0.289	0.332	0.366	0.498	0.226	0.281	0.355	0.515
	Range	0.324	0.360	0.387	0.559	0.277	0.330	0.396	0.603
Man	PPM	0.261	0.307	0.335	0.373	0.245	0.305	0.378	0.488
	SZIP	0.280	0.324	0.351	0.381	0.259	0.318	0.390	0.489
	Burrown	0.279	0.323	0.355	0.385	0.330	0.321	0.395	0.499
	LZP	0.300	0.344	0.370	0.436	0.274	0.333	0.404	0.565
	LZ77	0.301	0.345	0.373	0.500	0.288	0.347	0.418	0.604
	Range	0.323	0.361	0.385	0.556	0.320	0.376	0.438	0.641
Barbara	PPM	0.266	0.310	0.340	0.374	0.236	0.292	0.364	0.473
	SZIP	0.283	0.324	0.356	0.384	0.252	0.307	0.376	0.477
	Burrown	0.285	0.328	0.361	0.395	0.253	0.313	0.384	0.489
	LZP	0.299	0.341	0.371	0.437	0.266	0.318	0.388	0.545
	LZ77	0.303	0.346	0.376	0.501	0.276	0.331	0.401	0.585
	Range	0.334	0.370	0.396	0.558	0.322	0.377	0.440	0.644
Baboon	PPM	0.313	0.353	0.381	0.416	0.286	0.359	0.446	0.542
	SZIP	0.331	0.368	0.394	0.421	0.303	0.374	0.454	0.541
	Burrown	0.405	0.367	0.397	0.424	0.301	0.373	0.462	0.552
	LZP	0.350	0.391	0.418	0.488	0.320	0.392	0.476	0.627
	LZ77	0.339	0.373	0.397	0.544	0.324	0.388	0.470	0.662
	Range	0.334	0.368	0.392	0.560	0.330	0.395	0.469	0.663

Tabla 5.3. Resultados de compresión obtenidos después de aplicar varias técnicas de codificación de entropía a la información generada del algoritmo propuesto (continuación).

Imagen	Técnica de codificación de entropia	Algoritmo Propuesto VQ-EAM (operador sum - gamma)				Algoritmo Propuesto VQ-EAM (operador sum - Lamda)			
		Tamaño del codebook				Tamaño del codebook			
		64	128	256	512	64	128	256	512
		bpp	bpp	bpp	bpp	bpp	bpp	bpp	bpp
Lena	PPM	0.195	0.250	0.324	0.422	0.177	0.235	0.303	0.413
	SZIP	0.201	0.256	0.331	0.422	0.184	0.247	0.311	0.415
	Burrown	0.204	0.262	0.341	0.436	0.191	0.253	0.321	0.428
	LZP	0.216	0.267	0.342	0.484	0.197	0.258	0.321	0.473
	LZ77	0.222	0.280	0.365	0.528	0.207	0.274	0.343	0.520
	Range	0.267	0.341	0.417	0.611	0.265	0.332	0.397	0.609
Peppers	PPM	0.166	0.212	0.298	0.388	0.167	0.230	0.283	0.383
	SZIP	0.174	0.220	0.309	0.390	0.179	0.243	0.295	0.386
	Burrown	0.176	0.227	0.319	0.405	0.185	0.251	0.305	0.400
	LZP	0.189	0.233	0.320	0.449	0.194	0.256	0.306	0.446
	LZ77	0.199	0.249	0.348	0.500	0.204	0.278	0.329	0.494
	Range	0.245	0.310	0.396	0.595	0.263	0.329	0.383	0.595
Elaine	PPM	0.143	0.204	0.288	0.380	0.150	0.215	0.278	0.381
	SZIP	0.153	0.217	0.299	0.389	0.160	0.229	0.292	0.390
	Burrown	0.159	0.222	0.307	0.401	0.167	0.238	0.301	0.398
	LZP	0.170	0.231	0.314	0.439	0.177	0.242	0.304	0.439
	LZ77	0.180	0.251	0.345	0.506	0.188	0.264	0.332	0.504
	Range	0.213	0.299	0.383	0.598	0.238	0.307	0.374	0.590
Man	PPM	0.233	0.299	0.383	0.484	0.213	0.278	0.341	0.455
	SZIP	0.245	0.312	0.394	0.486	0.224	0.288	0.351	0.457
	Burrown	0.246	0.316	0.401	0.495	0.230	0.295	0.355	0.463
	LZP	0.261	0.327	0.409	0.559	0.239	0.303	0.366	0.528
	LZ77	0.274	0.341	0.423	0.599	0.252	0.318	0.377	0.571
	Range	0.299	0.369	0.442	0.634	0.274	0.341	0.402	0.612
Barbara	PPM	0.212	0.276	0.358	0.455	0.204	0.270	0.336	0.448
	SZIP	0.227	0.290	0.370	0.459	0.217	0.285	0.348	0.451
	Burrown	0.230	0.295	0.380	0.466	0.223	0.290	0.356	0.459
	LZP	0.241	0.303	0.382	0.524	0.232	0.297	0.361	0.519
	LZ77	0.250	0.311	0.397	0.566	0.239	0.307	0.370	0.560
	Range	0.286	0.357	0.431	0.620	0.278	0.345	0.410	0.622
Baboon	PPM	0.225	0.318	0.418	0.518	0.275	0.355	0.441	0.541
	SZIP	0.239	0.335	0.428	0.519	0.293	0.371	0.450	0.539
	Burrown	0.308	0.336	0.437	0.532	0.293	0.372	0.472	0.549
	LZP	0.255	0.352	0.450	0.603	0.310	0.387	0.471	0.627
	LZ77	0.265	0.351	0.441	0.643	0.313	0.385	0.464	0.661
	Range	0.263	0.356	0.444	0.648	0.318	0.389	0.465	0.656

Estos resultados muestran que cuando estos métodos de codificación se aplican a los resultados obtenidos por el algoritmo propuesto, VQ-MAE, la técnica de codificación de entropía que ofrece el mejor resultado en su tasa de compresión es el codificador PPM. El codificador PPM es un método estadístico adaptativo; sus operaciones se basan en ecuaciones parciales de cadenas; es decir, la codificación PPM predice el valor de un elemento basada en la secuencia de elementos previos.

Por lo tanto, estos resultados muestran que el algoritmo propuesto permanece competitivo con el algoritmo LBG en el parámetro de compresión.

Finalmente, se realizó un análisis de complejidad a ambos algoritmos con base en los resultados obtenidos en la sección de *complejidad del algoritmo* VQ-MAE y del estudio del algoritmo LBG presentado en [26]. La Tabla 5.4 resume la complejidad computacional de la fase de codificación de ambos algoritmos tanto del algoritmo propuesto como del algoritmo LBG en términos del tipo y número promedio de operaciones por pixeles. Este experimento se realizó para n=16 y 64, que representan las dimensiones de patrones más populares. Con respecto a la complejidad temporal, esta tabla muestra que el algoritmo propuesto proporciona mejoras considerables sobre el algoritmo LBG. La Tabla 5.4 también muestra que la cantidad de memoria requerida para el algoritmo propuesto es más pequeña que la cantidad de memoria requerida para el algoritmo LBG cuando el proceso de VQ se ejecuta.

Tabla 5.4. Complejidad computacional en la fase de codificación para el algoritmo propuesto VQ-MAE y el algoritmo LBG.

Algoritmo	**Dimensión del patrón (*n*)**	**Tamaño del codebook (*N*)**	**Memoria requerida (bytes)**	**Número promedio de operaciones por pixel**				
				CMP	±	×	**SQRT**	**Shift**
Algoritmo LBG	16	128	41088	8	387	129	8.06	-
		256	49280	16	771	257	16.06	-
		512	65664	32	1539	513	32.06	-
	64	128	41472	2	387	129	2.015	-
		256	74240	4	771	257	4.015	-
		512	139776	8	1539	513	8.015	-
Algoritmo propuesto con operadores **prom** y **pmed**.	16	128	18560	136	128	-	-	-
		256	20736	272	256	-	-	-
		512	41472	544	512	-	-	-
	64	128	12416	130	128	-	-	-
		256	20736	260	256	-	-	-
		512	37376	520	512	-	-	-
Algoritmo propuesto con operadores **med**.	16	128	16512	10.062	10.125	-	-	0.1875
		256	16640	18.062	18.125	-	-	0.1875
		512	33280	34.062	34.125	-	-	0.1875
	64	128	4224	4.015	4.031	-	-	0.0468
		256	4352	6.015	6.031	-	-	0.0468
		512	8704	10.015	10.031	-	-	0.0468

Como parte de uno de los objetivos a alcanzar en la presente investigación se desarrolló la interface gráfica que permite manipular de forma simple y efectiva un conjunto de imágenes de prueba que pueden ser usadas para el proceso de cuantificación vectorial. Esta interface cuenta con una barra de menú con las opciones: **Archivo** (Abrir, Salvar, Salir), **Compresión** (Genera codebook, Cuantificación) y **Descompresión** (Cuantificación inversa). En la Figura 5.2 se muestras las distintas opciones que contiene la barra de menú según se mencionó.

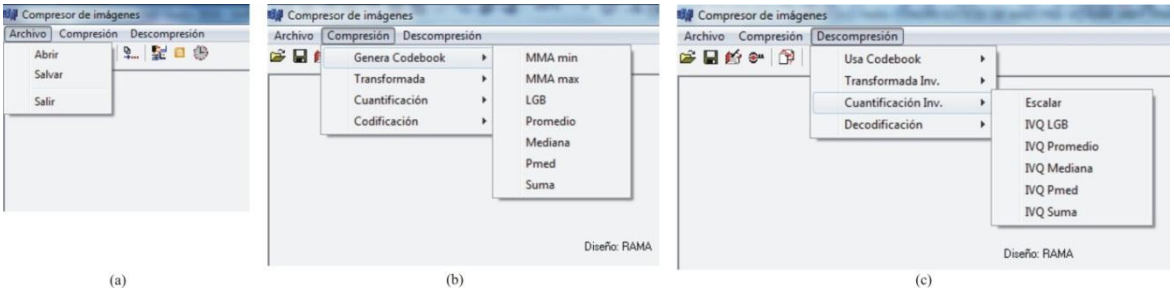

Figura 5.2. Opciones disponibles en cada menú de la barra de menú de la interface del compresor de imágenes VQ-MAE. (a) Menú Archivo, (b) Menú Compresión y (c) Menú Descompresión.

Así mismo la misma interface del compresor de imágenes basado en VQ-MAE cuenta con una barra de herramientas que permite al usuario acceder a las siguientes opciones (Abrir_imagen, Guardar_imagen, Leer_datos, Salvar_datos, imagen_original – imagen_procesada, Cerrar_todas_las_ventanas, Salir). La figura 5.3 describe estos iconos.

Figura 5.3. Opciones disponibles en la barra de herramientas de la interface del compresor de imágenes VQ-MAE.

A continuación se muestra en la Figura 5.4 una imagen de prueba (Lena.bmp) sometida al proceso de compresión usando el VQ-MAE con el operador **prom** y mostrando la imagen original, la imagen cuantificada y la pérdida generada usando el PSNR como criterio objetivo de medida.

Finalmente, en las Figuras 5.5, 5.6 y 5.7, se muestran la imágenes de prueba (*Lena.bmp*, *Elaine.bmp* y *Peppers.bmp*) habiendo experimentado el proceso de VQ con VQ-LBG (Algoritmo clásico) y VQ-MAE usando el operador promedio aritmético **prom** (Algoritmo propuesto) usando dos de los cuatro tamaños de libro de códigos sugeridos: 128 y 256 vectores de reconstrucción y habiéndose realizado la compresión en dos momentos distintos uno a la vez. El resultado visual de estas tres figuras muestra de forma subjetiva del proceso VQ y el error calculado mediante el PSNR.

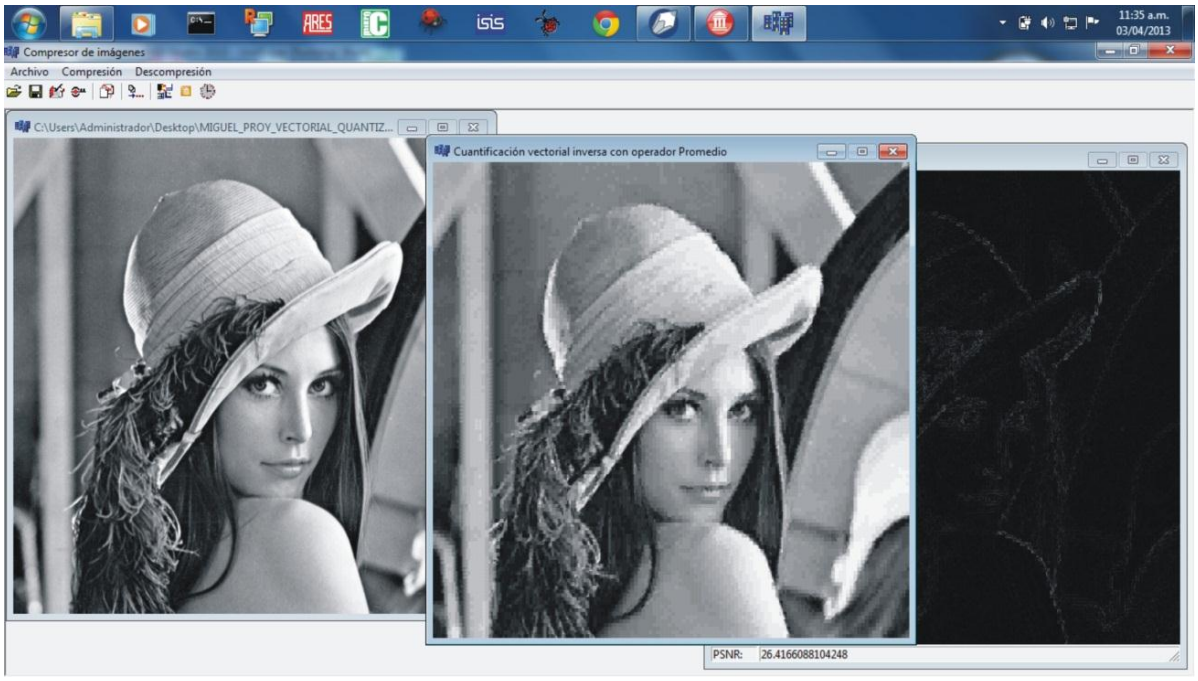

Figura 5.4. Interface gráfica que permite manipular imágenes del conjunto de imágenes de prueba para el procesamiento de VQ-LBG y VQ-MAE.

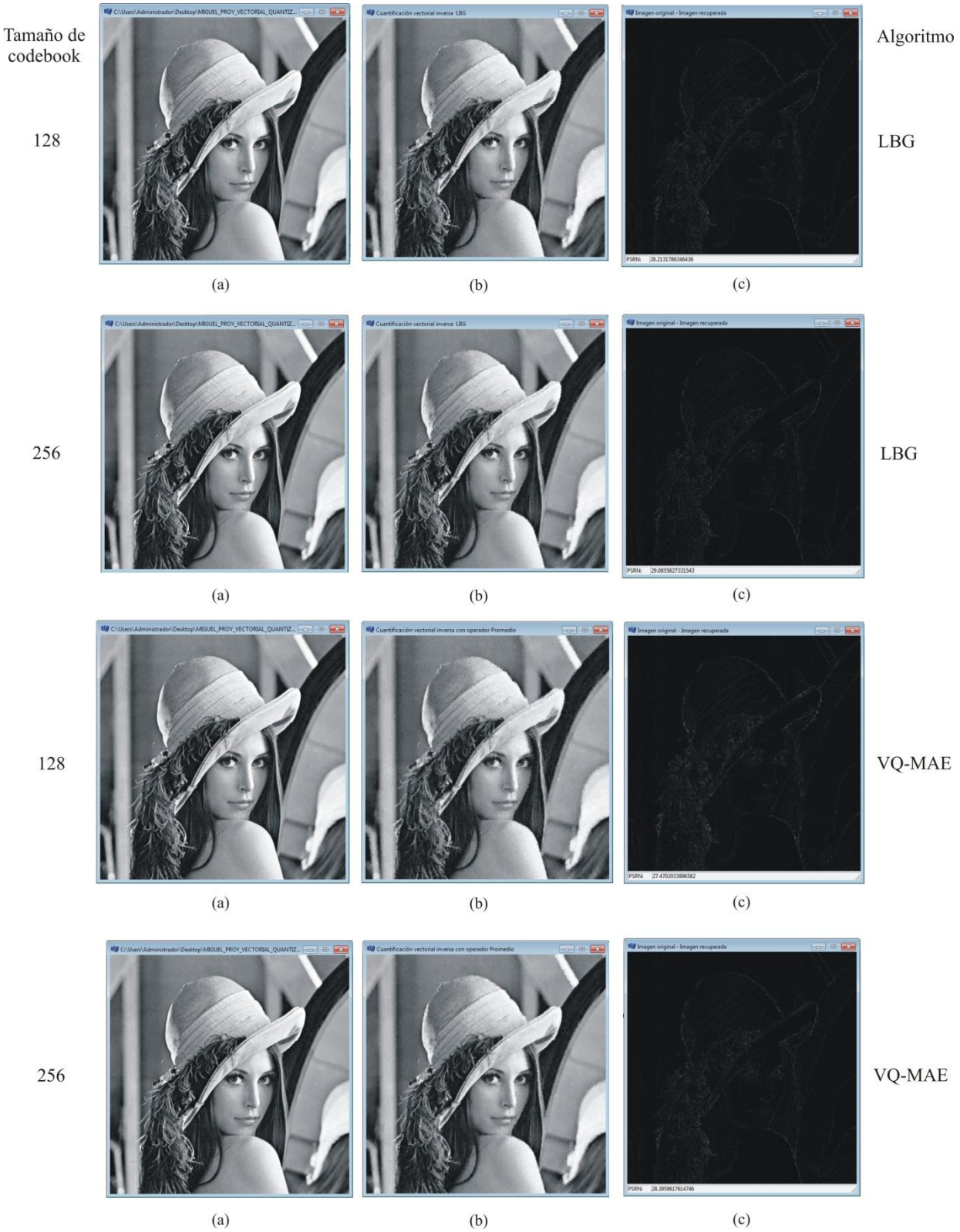

Figura 5.5. Imagen "Lena.bmp" con dos de los cuatro tamaños de codebook usados : 128, 256. (a) Imagen original, (b) Imagen cuantificada con LBG y VQ-MAE, (c) Error generado en el proceso de VQ según el algoritmo usado.

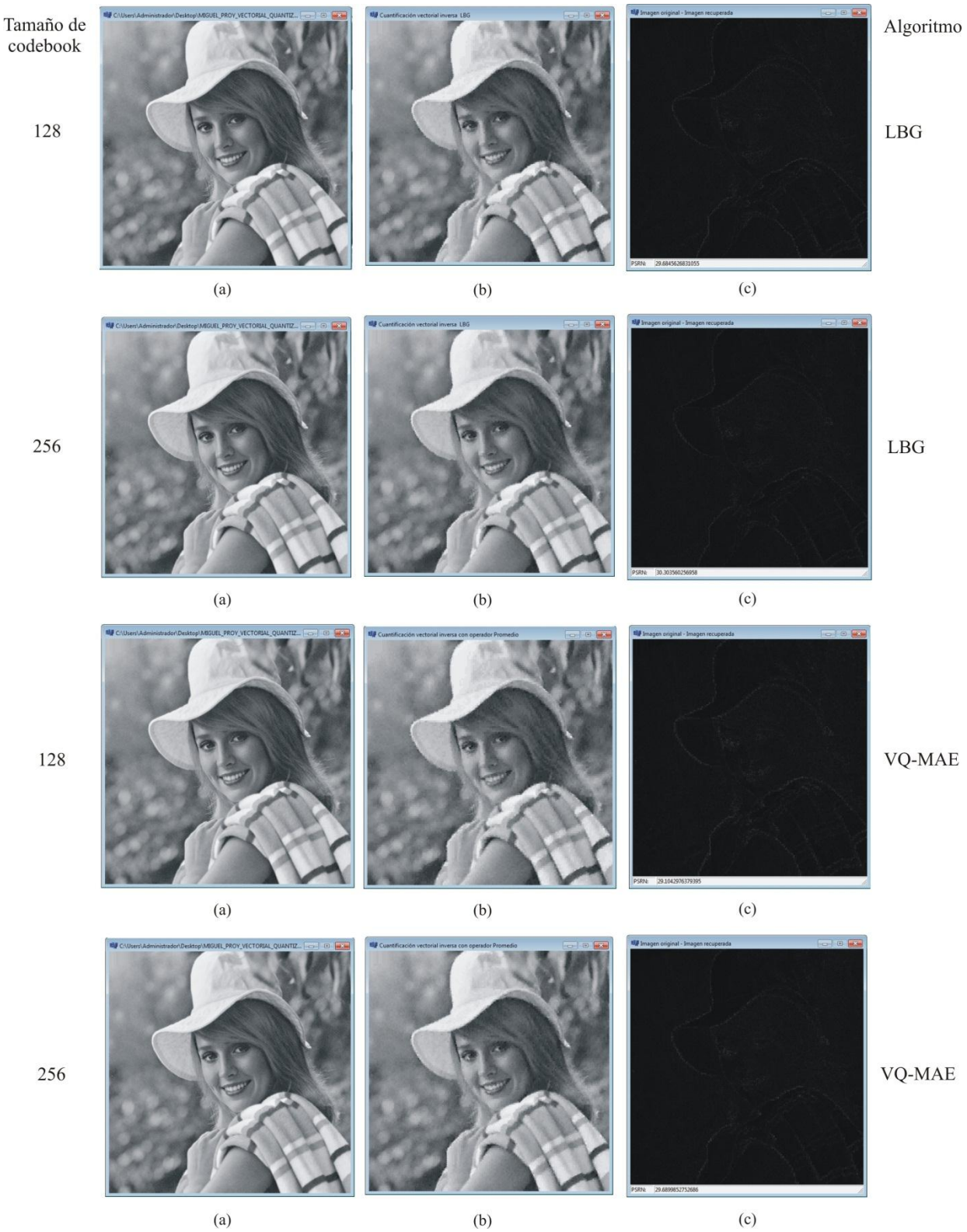

Figura 5.6. Imagen "Elaine.bmp" con dos de los cuatro tamaños de codebook usados : 128, 256. (a) Imagen original, (b) Imagen cuantificada con LBG y VQ-MAE, (c) Error generado en el proceso de VQ según el algoritmo usado.

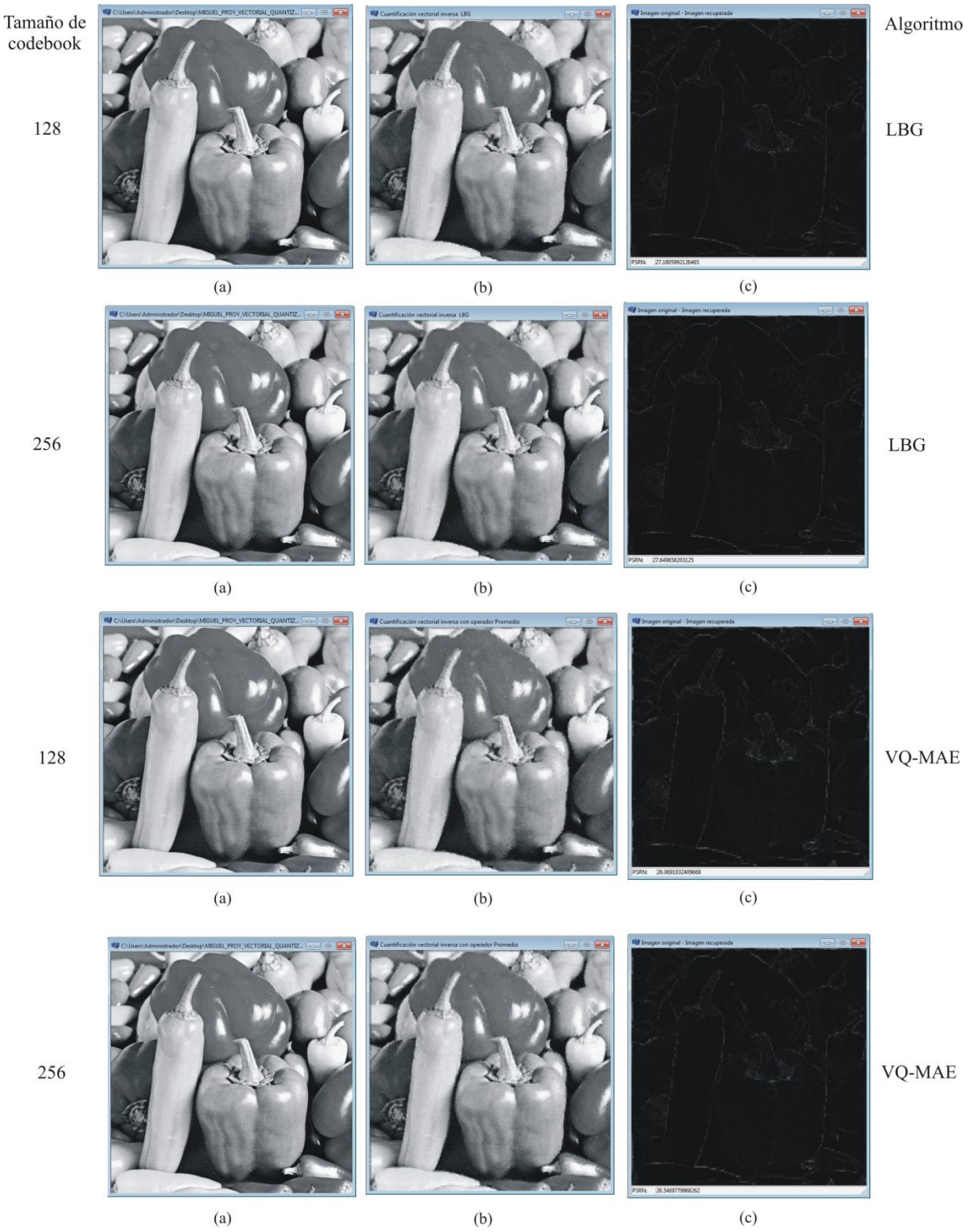

Figura 5.7. Imagen "Peppers.bmp" con dos de los cuatro tamaños de codebook usados : 128, 256. (a) Imagen original, (b) Imagen cuantificada con LBG y VQ-MAE, (c) Error generado en el proceso de VQ según el algoritmo usado.

Capítulo 6

Conclusiones y trabajo futuro

En este capítulo se incluyen las conclusiones generadas por este trabajo después de haber analizado el desempeño del algoritmo propuesto y compararlo con algoritmos de cuantificación vectorial clásicos. Posibles trabajos que darán continuación a la presente investigación son también discutidos.

6.1 Conclusiones

1. Una MAE tiene la propiedad de remplazar un vector de entrada por el índice del vector de reconstrucción que presenta mayor similitud, sin que sea necesario que este vector de entrada haya sido usado en la generación de la memoria. Esta propiedad es heredada por el VQ-MAE permitiendo cuantificar eficientemente imágenes que no se utilizaron en la generación del **MAE-codebook**.
2. La operación de las MAE puede llevarse a cabo con 1 de 4 posibles operadores, estos son el **prom**, **med**, **pmed** y **sum**. Con respecto a estos operadores, los resultados experimentales muestran que el operador **prom** genera la menor distorsión sobre las imágenes de prueba. Dicho de otra forma, el mejor desempeño en el parámetro relación señal a ruido se presenta cuando VQ-MAE utiliza el operador **prom**.
3. Con la finalidad de determinar el desempeño de VQ-MAE en el parámetro relación de compresión, un esquema de compresión fue implementado. Este esquema incluye

diversos algoritmos de codificación. Los resultados permiten concluir que al comparar el desempeño de VQ-MAE cuando usa cada uno de los operadores, nuevamente es el operador **prom** quien presenta un mejor rendimiento.

4. Además, los resultados muestran que cuando estos métodos de codificación se aplican a los resultados obtenidos por VQ-MAE la técnica de codificación de entropía que ofrece el mejor resultado en su tasa de compresión es el codificador PPM.
5. Un análisis de complejidad, temporal y espacial, fue realizado sobre el VQ-MAE con cada uno de los operadores. Los resultados obtenidos muestran que los operadores **prom** y **pmed** son los que menor número de operaciones aritméticas realizan durante su operación y también son operaciones de baja complejidad, solo sumas y comparaciones. Estos resultados permiten concluir que VQ-MAE presentará una mayor velocidad de procesamiento cuando utilice el operador **prom** o el **pmed**.
6. El algoritmo VQ-MAE, cuando usa el operador **prom**, se mantiene competitivo con algoritmos tradicionales como el LBG en los parámetros relación señal a ruido y relación de compresión y que los supera en velocidad de procesamiento y recursos consumidos durante su ejecución.

6.2 Trabajo futuro

1. Para minimizar la distorsión generada en el proceso de cuantificación de las imágenes que no se usaron para generar el **MAE-codebook**, es posible usar un método evolutivo que permite agregar información de nuevas imágenes al libro de códigos.
2. Implementar el VQ-MAE utilizando otros modelos asociativos y comparar los resultados con los presentados en esta investigación.
3. Proponer, fundamentar e implementar nuevos operadores para las MAE.

Apéndice A

Simbología

hi	dimensión de los patrones de entrada, alto de una imagen.
wi	dimensión de los patrones de entrada, ancho de una imagen.
$\mathbf{x}$	patrón de entrada, vector fila.
$\mathbf{y}$	patrón de salida, vector fila.
x_j	j-ésimo elemento de uno de los M patrones de entrada $X = \{\mathbf{x}^i : \ i = 1,2,...,M\}$.
y_j	j-ésimo elemento de uno de los N vector de reconstrucción $C = \{\mathbf{y}^i : \ i = 1,2,...,N\}$.
x_i^{φ}	i-ésimo elemento del patrón de entrada x^{φ}.
$\boldsymbol{c}_i$	i-ésimo vector de reconstrucción, codeword.
V_i	i-ésima región de Voronoi asociado a cada uno de los vectores de reconstrucción del libro de códigos.

$\mathbf{R}^n$	Espacio dimensional *n* donde para nuestro caso *n* = 16.
$\tilde{\mathbf{x}}$	versión alterada del vector x.
$\{(\mathbf{x}^1, c_1), (\mathbf{x}^2, c_2), ..., (\mathbf{x}^N, c_N)\}$	asociaciones entre un patrón de entrada y su correspondiente índice de clase.
$\{(\mathbf{x}^\mu, c_\mu) \quad \mid \quad \mu = 1,2,3,..., N\}$	μ-ésimo elemento del conjunto fundamental de asociaciones.
$(\mathbf{x}^\mu)^t$	transpuesta del vector $\mathbf{x}^\mu$.
$\mathbf{MAE}$ - ***codebook*** $= \boldsymbol{M} = [m_{ij}]_{\mathrm{N}\otimes\mathrm{n}}$	Matriz que representa la memoria asociativa conteniendo los pesos sinápticos que pertenecen a la salida i ordenados en la matriz *M* en la *i*-ésima fila utilizada tanto en la fase de aprendizaje como en la fase de clasificación de patrones.
$i = index$	índice de clase a la cual pertenece el vector x.
$\vee$	operador morfológico que representa el máximo usados en los operadores de la MAE.
$\wedge$	operador morfológico que representa el mínimo usados en los operadores de la MAE.
$\phi_{ij} = m_{ij}$	función que actúa como mecanismo de aprendizaje generalizado y esta puede ser evaluada de varias formas según cada uno de los 4 operadores de la MAE propuesta.
$\boldsymbol{Q}(\mathbf{x}) = \mathbf{y}^i = (y_{i,1}, y_{i,2}, \cdots, y_{i,n})$	mapeo donde se asigna un índice i a cada vector de entrada $\mathbf{x}$ a $\mathbf{y}^i$ cuando satisface la condición de la distancia euclidiana.
$d(\mathbf{x}, \mathbf{y}^j)$	distorsión medida mediante la distancia euclidiana.
$\boldsymbol{H} = \{h_i : \quad i = 1,2,3,\cdots, M\}$	Conjunto de *M* índices que indica el vector de reconstrucción o índice de clase a la que pertenece cada vector de entrada.
$\boldsymbol{B} = \{b_i : \quad i = 1,2,3,\cdots, N\}$	Conjunto de *N* índices que indica el número de vectores de entrada que integran cada índice de clase o vector de reconstrucción.

Referencias

[1] Abu-Mostafa, Y. S. and St Jacques, J. M. (1985). *"Information capacity of the Hopfield model"*. IEEE Transactions on Information Theory, 31, 461-464.

[2] Amerijckx, C., Verleysen, M., Thissen P. and Legat, J-D. (1998). *"Image Compression by Self-Organized Kohonen Map"*. IEEE Transactions. on Neural Networks, vol. 9, n. 3, pp. 503-507.

[3] Amerijckx, C., Legat, J.-D. and Verleysen, M. (2003). *"Image Compression Using Self-Organizing Maps"*. Systems Analysis Modelling Simulation, Vol. 43, No. 11, pp. 1529–1543.

[4] Anderson, J. A. (1972). *"A simple neural network generating an interactive memory"*. Mathematical Biosciences, 14, 197-220.

[5] Anderson, J. R. and Bower, G. (1977). "*Memoria Asociativa"*, México: LIMUSA.

[6] Austin, J. and Stonham. T. J. (1987). *"Distributed associative memory for use in scene analysis"*. Image and Vision Computing, 5, 251-260.

[7] Bang Huang, Linbo Xie. (2010). *"An Improved LBG Algorithm for Image Vector Quantization"*. 978-1-4244-5540-9/10. pp. 467-471.

[8] Barron. R. (2005). *"Associative Memories and Morphological Neural Networks for Patterns Recall (In Spanish)"*. PhD Thesis, Center for Computing Research.

[9] Barron, R. (2006). *"Associative Memories and Morphological Neural Networks for Patterns Recall"*. Ph.D. dissertation, Center for Computing Research of National Polytechnic Institute, México.

[10] Basil, G. and Jiang. J. (1999). *"An Improvement on Competitive Learning Neural Network by LBG Vector Quantization"*. Proceedings of IEEE International Conference on Multimedia Computing and Systems. Vol. 1, pp 244-249.

[11] Cattermole, K. (1962). *"Ley μ, Ley A"*; Dos tipos de codificación no uniforme: la ley μ (usada en USA y Japón) y la ley A (Europa y Sudamérica). Actualmente se encuentra en - ITU-T G.711.

[12] Chin-Chuan, H., Ying-Nong, C., Chih-Chung, L., Cheng-Tzu, W. and Kuo-Chin, F. (2006). *"A Novel Approach for Vector Quantization Using a Neural Network, Mean Shift, and Principal Component Analysis"*. Proceedings of IEEE Intelligent Vehicles Symposium, pp 244-249, ISBN: 4-901122-86-X, Tokio, Japan.

[13] Chu W. C. and Bose N. K. (1998). *"Vector Quantization of Neural Networks"*. IEEE Transactions on Neural Networks, Vol. 9, No. 8. pp. 1235-1245.

[14] Cormen T.H., Leiserson C. E. and Rivest R. L. (1995). *"Introductions to Algoritms"*. McGraw-Hill, USA. ISBN 0-07-013143-0.

[15] Díaz-de-León, J. L. and Yáñez, C. (1999). "*Memorias asociativas con respuesta perfecta y capacidad infinita"*. Memoria del TAINA´99, México, D.F., 23-38.

[16] Díaz-de-León Santiago, J. L. and Yañez Márquez, C. (2001). *"Memorias Morfológicas Heteroasociativas"*, IT-57, Serie Verde, ISBN: 970-18-6697-5, CIC-IPN, México.

[17] Eyal, Y., Zeger, K. and Gersho, A. (1992). *"Competitive Learning and Quantizer Design"*. IEEE Transactions on Signal Processing, vol. 40. n. 2. pp. 294-309. ISSN: 1053-587X.

[18] Ghanbari M. (2003). *"Standard Codecs: Image Compression to Advanced Video Coding"*.The Institution of Electrical Engineers, United Kingdom. ISBN: 0-85296-710-1.

[19] Gersho, A. and Gray R. M. (1992). *"Vector Quantization and Signal Compression"*. Kluwer Academic, ISBN 0-7923-9181-0, Norwell, Massachusetts USA.

[20] González M. y Ferrero R.(1998) *"Fundamentos de Compresión de Imágenes"*. Departamento de Matemáticas de la Universidad de Oviedo.

[21] Gonzalez R. Wood R.(2008) *"Tratamiento digital de imágenes"*. Addison-Wesley. USA.

[22] Gray, R., M. (1984). *"Vector Quantization"*. IEEE ASSP Magazine, Vol 1, pp. 4-9, ISSN:0740-7467.

[23] Gray, R., M. (1998). *"Quantization"*. IEEE Transactions ON Information Theory, vol 44, No. 6, pp. 2325-2383.

[24] Guerequeta R. y Vallecillo A. (1998). *"Técnicas de Diseño de Algoritmos"*. Servicio de Publicaciones de la Universidad de Málaga, España. ISBN 84-7496-666-3.

[25] Guzmán E., Pogrebnyak O. and Yañez, C. (2007). *"Design of an Evolutionary Codebook Based on Morphological Associative Memories"*. Proceeding of the 6th Mexican International Conference on Artificial Intelligence, LNAI 4827, Springer, Vol. 4827/2007, pp. 601-611, ISSN 0302-9743, Aguascalientes, México.

[26] Guzmán, E., Pogrebnyak, O., Yañez, C. (2008). *"A Fast Search Algorithm for Vector Quantization based on Associative Memories"*. Proceeding of the 13th Iberoamerican congress on Pattern Recognition, LNCS 5197, Springer, Vol. 5197/2008, pp. 487-495, ISSN 0302-9743, Habana, Cuba.

[27] Guzmán R., Enrique. (2008). *"Compresión de imágenes mediante Memorias Asociativas"*. Tesis doctoral, Centro de Investigación en Computación, IPN. México, D.F.

[28] Guzmán R. E., Zambrano J. G., Orantes A., Pogrebnyak O. (2009). *"A Theoretical Exposition to Aply the Lamda Methodology to Vector Quantization"*. IEEE Xplode, 978-1-4244-4480-9/09. pp. 743-746.

[29] Hopfield, J. J. (1982). *"Neural Networks and Physical Systems with Emergent Collective Computational Abilities"*. Proceedings of the National Academy of Sciences, 79, 2554-2558.

[30] Hsiang-Cheh Huang, Jeng-Shyang Pan, *et al.* (2001). *"Vector quantization based on genetic simulated annealing"*. Elsevier Signal Processing 81. pp. 1513-1523.

[31] Hsiang-Cheh Huang, Shu-Chuan Chu, *et al.* (2001). *"A Tabu Search Based Maximum Descent Algorithm for VQ Codebook Desing"*. Journal of Information Science and Engineering 17, pp. 753-762.

[32] Huang, C. M., and Harris, R. W. (1993). *"A Comparison of Several Vector Quantization Codebook Generations Approaches"*. IEEE Transactions on Image Processing, vol. 2, n. 1, pp. 108-112.

[33] Huffman D. A. (1952). *"A Method for the Construction of Minimun-Redundancy Codes"*. Proceedings of the Institute of Radio Engineers, 40 (9), pp. 1098-1101.

[34] Itakura, F. (1975). *"Minimum Prediction Residual Principle Applied to Speech Recognition"*. IEEE Transactions on ASSP, vol. ASSP-23, n. 1, Jan. 1980.

[35] Jain A. K. (1979). *"A sinusoidal family of unitary transforms"*. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 1, n. 4, pp. 356-365.

[36] Kohonen, T. (1972). *"Correlation Matrix Memories"*. IEEE Transactions on Computers", C-21, 4, 353-359.

[37] Kohonen. T. (1981). *"Automatic formation of topological maps of patterns in a self-organizing system"*. In Erkki Oja and Olli Simula, editors, Proc. 2SCIA, Scand. Conf. on Image Analysis, Helsinki, Finland, pp. 214-220.

[38] Kohonen T. (1982). *"Self-Organized formation of topologically correct feature maps"*. Biological Cybernetics, Vol. 43, No. 1, 59-69. ISSN: 0340-1200.

[39] Kosko, B. (1988). *"Bidirectional associative memories"*. IEEE Transactions on Systems, MAN and Cybernetics, 18, 49-60.

[40] Linde, Y., Buzo, A., and Gray, R. (1980). *"An algorithm for Vector Quantization Design"*. IEEE Transactions on Communications, vol. COM-28. n. 1., pp. 84-95, Jan. 1980.

[41] Lloyd, L. P. (1982). *"Least Squares Quantization in PCM"*. IEEE Transactions Inform. Theory, vol. IT-28, pp. 129-137.

[42] Makhoul, J., Roucos, S., and Gish, H. (1985). *"Vector Quantization in Speech Coding"*. Proceedings of the IEEE, vol. 73, n. 11, Noverber 1985.

[43] Nasrabadi N., M. and King R., A. (1988). *"Image Coding Using Vector Quantization: A Review"*. IEEE Transactions on Communications, Vol. 36, No. 8, pp. 957-971, ISSN: 0090-6778.

[44] Nelson M., and Gailly J-L. (1996). *"The Data Compression Book"*. Ed. M&T Books. Second edition. USA. ISBN 1558514341.

[45] Rabiner, L. R. and Shafer, R. W. (1978). "*Digital processing of Speech Signals"***.** Prentice-Hall. New Jersey.

[46] Rao K. R. and Yip P. C. (2001). *"The transform and data compression handbook"*. The Electrical and Engineering and Signal Processing Series. ISBN 0-8493-3692-9.

[47] Ritter, G. X., Sussner, P., and Díaz de León, J. L. (1998). *"Morphological associative memories"*. IEEE Transactions on Neural Networks, 9, 281-293.

[48] Rosen, K. H. (1996). *"Discrete Mathematics and Its Applications".* McGraw-Hill, Inc.

[49] Shanon C. E. (1948). *"A Mathematical Theory of Communication"*. Bell System Technical Journal, Vol. 27, pp. 379-423 and 623-656.

[50] Sossa H., Barrón R. and Vázquez, A. (2004). *"New Associative Memories to Recall Real-Valued Patterns"*. Lecture Notes in Computer Science, 3287, pp. 195-202.

[51] Sossa H., Barrón R. and Vázquez, A. (2004). *"Real-valued Patterns Classification based on Extended Associative Memory"*. In: Fifth Mexican International Conference on Computer Science, ENC, IEEE Computer Society, pp. 213-219.

[52] Steinbuch, K. (1961). *"Die Lernmatrix, Kybernetik"*,v. 1, n. 1, 36-45.

[53] Vázquez, R. (2005). *"Objects recognition in presence of traslapes by means of associative memories and invariants descriptions"*. Master dissertation, Center for Research in Computing of National Polytechnic Institute, México.

[54] Veprek P. and Bradley A. B. (2000). *"An Improved Algorithm for Vector Quantizer Design"*. IEEE Signal Processing Letters, vol. 7, n. 2., pp. 250-252.

[55] Witten I. H., Neal R. M. and Cleary J. G. (1987). *"Arithmetic Coding for Data Compression"*. Communications of ACM, Vol. 30, No. 6, pp. 520-540.

[56] Yáñez-Márquez, C. y J. L. Díaz-de-León Santiago, (2001). *"Lernmatrix de Steinbuch"*, No. 48 Serie VERDE, CIC-IPN, México.

[57] Yáñez-Márquez, C. y J. L. Díaz-de-León, (2001). *"Linear Associator de Anderson-Kohonen"*, No. 50 Serie VERDE, CIC-IPN, México.

[58] Yáñez-Márquez, C. y J. L. Díaz-de-León, (2001). *"Memorias Morfológicas Heteroasociativas"*, No. 57 Serie VERDE, ISBN 970-18-6697-5, CIC-IPN, México.

[59] Yáñez-Márquez C. (2002). *"Memorias Asociativas Basadas en relaciones de Orden y Operaciones Binarias"*. Tesis Doctoral, CIC-IPN, México.

[60] Yáñez, C. y Díaz de León, J. L., (2003). *"Introducción a las Memorias Asociativas",* Research in Computing Science, CIC-IPN, México.

[61] Yañez Márquez, C. and Díaz-de-León Santiago, J. L. (2003). *"Memorias Autoasociativas Morfológicas max: condiciones suficientes para convergencia, aprendizaje y recuperación de patrones"*, IT-175, Serie Azul, ISBN: 970-36-0035-2, CIC-IPN, México.

[62] Yong-Soon, K. and Sung-Ihl K. (2007). *"Fuzzy Neural Network Model Using a Fuzzy Learning Vector Quantization with the Relative Distance"*. Proceedings of IEEE 7th International Conference on Hybrid Intelligent Systems, pp. 90-94, ISBN: 978-0-7695-2946-2, Kaiserslautern, Germany.