CLASIFICACIÓN AUTOMÁTICA DE PANELES DE MADERA

Nicolás E. M. Jedlicka
Director: Jorge A. Fiora - *INTI Energía*

- 2007 -

INSTITUTO UNIVERSITARIO GASTÓN DACHARY
Ingeniería en Informática

AGRADECIMIENTOS

A Jorge y Marga por la ayuda y las atenciones brindadas.

A Diego, Raúl, Andy y a todos los que de una u otra manera contribuyeron a que este trabajo se pueda terminar.

ÍNDICE DE CONTENIDO

LISTA DE SIGLAS Y ABREVIATURAS

t	número de iteración.
T	número total de iteraciones.
v	vector de entrada del SOM / LVQ.
w	vector de valores asociados a una neurona u.
n	dimensión de los vectores v y w.
u	unidad (neurona) del SOM.
u2	unidad (neurona) perteneciente al entorno de u.
dist	medida de distancia entre v y w (generalmente la distancia euclídea).
BMU	unidad ganadora (*Best Matching Unit*).
σ	radio del entorno que decrece a medida que aumenta t.
α	tasa de aprendizaje que decrece a medida que aumenta t.
λ	parámetro de aprendizaje dependiente de t. Se utiliza para hacer decrecer el radio del entorno y para hacer decrecer la tasa de aprendizaje α.

RESUMEN

La madera es una de las principales fuentes de ingresos de la industria local, pero la falta de normas que aseguren la calidad en el proceso de producción de los diferentes productos derivados de la madera plantea muchos inconvenientes, para los productores como así también para los consumidores. Las maderas terciadas, como así también los tableros recubiertos en madera natural, tienen una amplia gama de aplicaciones y son utilizadas en grandes cantidades. En la mayoría de las aplicaciones es importante que las partes visibles de la madera utilizada contengan la menor cantidad de defectos posible, o por lo menos los defectos contemplados en una calidad determinada. Al no existir un estándar para la clasificación de las placas, cada fábrica aplica una nomenclatura diferente para etiquetar sus productos, lo cual hace muy difícil que los consumidores sepan de antemano cómo es el aspecto del producto, o que fallas posee. Otro problema importante es que la clasificación es realizada por operadores humanos, los cuales no poseen criterios homogéneos entre sí, y aún peor, un mismo operario puede variar su criterio influenciado por la fatiga de realizar un trabajo repetitivo por varias horas.

En este trabajo se presentan una serie de técnicas y procedimientos computacionales para tratar de resolver el problema de la clasificación de tableros de madera. Principalmente se utilizan técnicas de Inteligencia Artificial, como redes neuronales artificiales (RNA) y técnicas de procesamiento de imágenes digitales. Combinando estas técnicas, se puede implementar un sistema que permita la clasificación automática y homogénea de paneles de madera según el aspecto de sus caras.

CAPÍTULO 1 - MARCO TEÓRICO

1.1. Propósito y resultados principales

El propósito de este trabajo es el reconocimiento de defectos en placas de madera, a través del análisis de imágenes digitales por medio de una computadora. Esto con el fin de clasificarlas de una manera semejante a como se realiza actualmente de manera manual. Se quiere decir que el sistema automático debe reproducir cercanamente la clasificación que realizaría un buen operador especializado.

Por otra parte las clasificaciones manuales no son muy uniformes y de hecho un mismo operador, después de algún tiempo de trabajo, gradualmente tiende a cometer más y más errores debido al cansancio. El uso de un sistema automático daría como resultado al menos una clasificación uniforme, claro esta que esta clasificación debe corresponderse con los criterios usuales vigentes actualmente en la industria para la calidad de las placas.

No se pretende aquí presentar un sistema terminado y operativo a escala industrial, sino una serie de procedimientos computacionales que permitan el reconocimiento automático de defectos con cierto grado de certeza. Estos procedimientos, podrían constituir la base de un sistema real que necesariamente debería plasmarse en una máquina concreta que realice la clasificación.

Con respecto a la implantación de tal sistema en una línea real de producción se cree que hay asimismo al menos dos niveles de automatización :

- Semiautomático: donde el sistema de análisis indica a los operadores por medio de señales visuales (luces por ejemplo) el grado de calidad de la placa, la separación es manual. Esta modalidad permitiría que el operador separe las placas sobre las cuales no esta de acuerdo con la clasificación automática. Una posterior revisión de estos casos serviría para perfeccionar el método. Ésta se cree que sería la forma aconsejable en una primera instancia.
- Automático: el sistema de análisis comanda directamente una separadora mecánica. Es posible asimismo una supervisión humana que marque los posibles defectos para su posterior revisión.

Se presenta pues un sistema (prototipo) que opera en una computadora personal analizando imágenes digitales previamente adquiridas e indica las áreas

con defectos, distinguiéndolos entre si con un color característico previamente asignado. De hecho se analizan diversos algoritmos de clasificación y se muestran análisis comparativos.

El principio general de funcionamiento es como sigue:

- La imagen se divide en pequeños cuadrados de $n \times n$ píxeles (de color, ver [1]) que llamaremos Regiones Elementales (RREE).
- Se tiene entonces para cada canal de color (rojo, verde y azul) n^2 valores que representan la intensidad del canal para cada píxel, es decir se tienen $3n^2$ valores (de 1 byte) para cada RE.
- Estos valores son los únicos datos a partir de los cuales, en una primera etapa, se clasifica cada RE como sana o defectuosa mediante una técnica de bajo costo computacional.
- Todas las RREE marcadas como defectuosas son objeto de un análisis ulterior cuya primara etapa consiste en agrupar todas las RE contiguas y considerar el menor rectángulo (unión de RREE) que las contiene. Este rectángulo será denominado una Zona Defectuosa (ZD).
- Las ZZDD son analizadas para clasificar el tipo de defecto (eventualmente podrían ser reclasificadas como sanas). Dado el menor volumen de información es posible aquí utilizar técnicas más complejas.

La clasificación de las RREE se realiza en principio con una técnica denominada "Mapa Auto-organizado", aunque, como se verá, es posible obtener resultados interesantes por una aplicación funcional directa de las RREE en un rectángulo.

La clasificación posterior se hace usando un clasificador del tipo LVQ (Learning Vector Quantization).

1.2. Imágenes digitales

Las imágenes analógicas (del mundo real) que vemos con nuestros ojos cotidianamente pueden ser definidas como una función de intensidad dependiente de dos variables $f(x,y)$, cuya salida es un valor de intensidad para la coordenada espacial (x,y). Este valor de intensidad se puede representar como un número entero o real, cuanto mas pequeño sea el valor obtenido, mas oscuro será el punto en cuestión, el menor valor posible representa el negro absoluto. Inversamente,

cuanto mas grande sea el valor obtenido, mas claro será el punto en cuestión y el valor máximo representa el blanco absoluto. Se obtiene así una imagen con diferentes niveles de intensidad o niveles de gris. La cantidad de niveles intermedios depende del rango de la función $f(x,y)$, siendo generalmente 256 valores en total por cada canal de color. [1]

Las imágenes digitales se obtienen a partir de las imágenes analógicas a través de un proceso denominado cuantización, el que consiste en transformar los valores analógicos continuos en valores discretos. Este valor discreto es asignado al punto (x,y) correspondiente en la imagen digital. Cada uno de estos puntos es denominado *píxel*[1] en una imagen digital.

Se puede definir entonces una imagen digital como un conjunto de píxeles, cuyas únicas propiedades son su posición en la imagen y su nivel de intensidad. La cantidad de píxeles depende del ancho de la imagen, definido como n, y la altura de la misma, definida como m. Representando una imagen digital en un sistema de coordenadas cartesiano, como en la Ilustración 1.1, se puede ver fácilmente como es posible obtener la posición de un píxel mediante sus coordenadas (x,y). La esquina superior derecha de la imagen representa el origen de coordenadas, la única diferencia con los ejes de coordenadas cartesianas convencionales es que en el eje y los valores están invertidos (crecen para abajo y no para arriba como en los ejes convencionales). En el caso de la Ilustración 1.1, se tienen únicamente dos valores de intensidad posibles, blanco o negro. Este tipo de imágenes se denomina imágenes binarias.

1 Del inglés *Picture Element*.

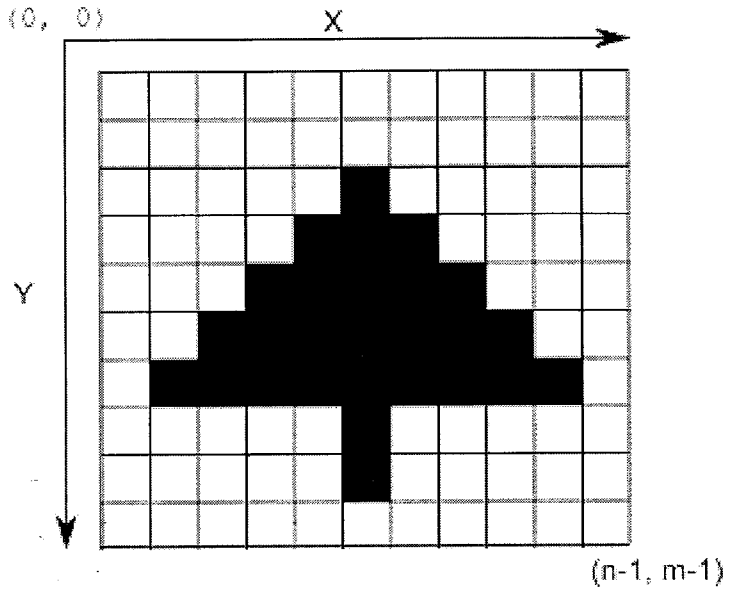


Ilustración 1.1: Esquema de una imagen digital

La cantidad de píxeles que contiene la imagen se denomina resolución en píxeles, o simplemente resolución y se suele especificar indicando el número de filas y de columnas (en píxeles). Cabe destacar que los píxeles no tienen un tamaño físico, sino que contienen únicamente información acerca de su posición dentro de la imagen y su nivel de intensidad, como se mencionó anteriormente. El tamaño físico de un píxel dependerá del dispositivo en el cual este es representado. Para aclarar este concepto se considera una imagen de 1024 x 768 píxeles. Esta imagen puede ser vista en un monitor de computadora LCD de 1024 x 768 píxeles de resolución, el cual tiene un tamaño físico de 14 pulgadas (medidas diagonalmente). En este monitor de 14 pulgadas cada punto del monitor que representa un píxel tendrá un tamaño determinado. Ahora considérese la misma imagen, pero esta vez visualizada en un monitor LCD de 17 pulgadas, el cual también tiene una resolución de 1024 x 768 píxeles. En este caso, la misma imagen se verá físicamente mas grande debido a que el dispositivo que representa la imagen es mas grande, pero su resolución sigue siendo la misma. A medida que aumenta la cantidad de píxeles, aumenta la resolución y por lo tanto la cantidad de datos contenidos en la imagen, pero el tamaño físico de la imagen dependerá del dispositivo en la cual esta sea representada.

La *resolución espacial* de una imagen, generalmente indicada en *PPI* (pixels per inch, píxeles por pulgada), depende del dispositivo en el que se la represente. Otra "resolución espacial" es la empleada en sistemas de representación de superficies, tales como los sistemas geoespaciales, donde se indica el área que representa cada píxel, o sus dimensiones. En este sentido es que se utilizará el

término resolución espacial de la imagen en este trabajo. La resolución espacial de las imágenes determina el tamaño de los defectos que pueden resolverse. Es claro que el tamaño de un píxel debe ser bastante menor que el del mínimo defecto a detectar. En este caso (circunstancialmente, debido a la resolución de la cámara fotográfica) cada píxel representa una región de 1,04 x 1,04 mm de la placa.

Se define una región de una imagen digital como un subconjunto de sus píxeles definido por alguna propiedad de la imagen, como la adyacencia de otros píxeles o sus valores de intensidad. La utilidad de las regiones radica en que a veces es interesante procesar únicamente una porción de la imagen en lugar de procesar la imagen completa, como se verá mas adelante en este trabajo. Estas porciones se denominan regiones de interés, ROI[2], o simplemente regiones.

1.2.1. Entorno de un píxel

El entorno de un píxel es un conjunto de píxeles vecinos determinado por una relación de adyacencia que existe entre este píxel y los píxeles que están conectados con el. Las relaciones mas utilizadas en el mundo de las imágenes digitales son las denominadas entorno-4 y entorno-8.

Si se tiene un píxel con coordenadas (x,y), el entorno-4 es el conjunto de píxeles adyacentes situados horizontalmente y verticalmente, teniendo así cuatro vecinos, de ahí el nombre del entorno.

(x - 1, y - 1)	(x, y - 1)	(x + 1, y - 1)
(x - 1, y)	(x, y)	(x + 1, y)
(x - 1, y + 1)	(x, y + 1)	(x + 1, y + 1)

Ilustración 1.2: Entorno de un píxel

El entorno-8 es el conjunto de píxeles del entorno-4 más los píxeles adyacentes situados diagonalmente. En la siguiente ilustración se puede apreciar el concepto de ambos tipos de entorno. El casillero de color rojo representa el píxel

2 Del inglés *Region Of Interest*

(x, y), los de color azul representan el entorno-4 y los de color azul mas los de color verde representan el entorno-8.

1.2.2. Histograma de una imagen

El histograma de una imagen es una función discreta que representa la proporción o frecuencia de píxeles de la imagen para cada nivel de intensidad. [2]

La frecuencia de ocurrencia de un determinado nivel de intensidad g está definido por:

$$P(g)=\frac{N(g)}{T} \tag{1}$$

donde $N(g)$ es la cantidad de píxeles de la imagen que tienen valor de intensidad g y T es el total de píxeles de la imagen.

Como con cualquier distribución de frecuencias los valores de $P(g)$ son menores o iguales a 1 y la suma de todos los valores de $P(g)$ da como resultado 1.

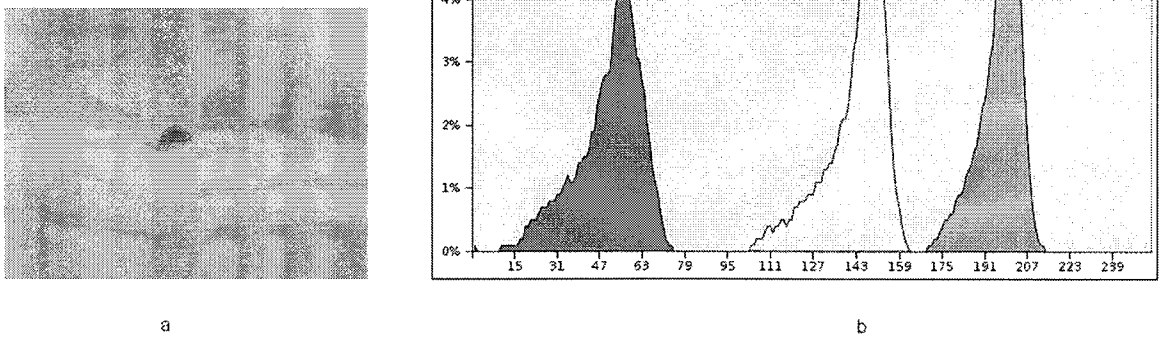


Ilustración 1.3: Histograma de una imagen digital

En la Ilustración 1.3 se presenta una imagen digital con su correspondiente histograma. La imagen de muestra contiene la información de color en tres canales separados: rojo, verde y azul. Al mezclar estos colores en diferentes proporciones se pueden obtener otros colores, formando así el color final para cada píxel (ver Apéndice A). En el caso de la imagen utilizada para la muestra y también en las demás imágenes utilizadas en este trabajo, la información de color para cada canal tiene la longitud de 8 bits. O sea que se tienen $2^8=256$ niveles de intensidad para

cada canal, variando desde 0 hasta 255 inclusive. Calculando todas las combinaciones de colores posibles entre los tres canales se obtiene que $256^3=16777216$, que es la cantidad de colores posibles que se pueden expresar en este tipo de imágenes, las cuales son conocidas como imágenes de 24 bits[3] o a veces llamadas de color verdadero. Existen también imágenes de color verdadero de 32 bits, las cuales poseen la misma cantidad de colores posibles que las de 24 bits, pero se le agregan 8 bits mas de información de transparencia.

1.3. Redes neuronales artificiales

Las redes neuronales artificiales (RNA) son un paradigma para el procesamiento de información, donde la neurona es el elemento central o la unidad básica de procesamiento. Este paradigma está basado en la manera en la cual trabajan los sistemas nerviosos biológicos, como el cerebro humano.

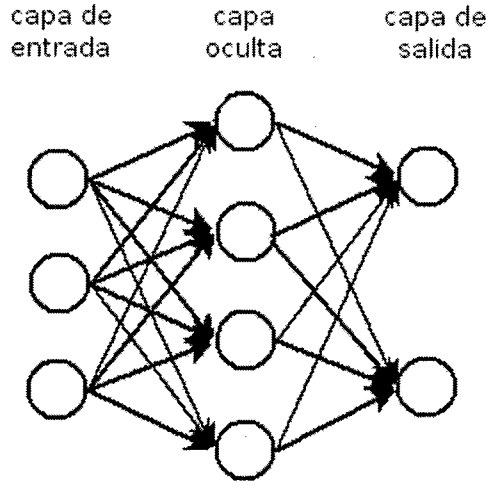


Ilustración 1.4: Estructura de una RNA

1.3.1. Estructura de una red neuronal artificial

Al igual que el cerebro humano, las RNA realizan una simplificación de la información con la cual trabajan, ya sea porque es excesiva o porque es redundante para el problema a resolver. Cualquier modelo de RNA consta de unidades elementales de procesamiento: las neuronas (representadas por círculos en la Ilustración 1.4). En 1943, Warren McCulloch y Walter Pitts realizaron el primer modelo matemático de una Red de Neuronas Artificiales, este modelo está basado en la idea de que las neuronas operan mediante impulsos binarios. Generalmente,

3 Por sus 3 componentes de 8 bits.

se pueden encontrar tres tipos de neuronas, dispuestas en diferentes capas:

1. Las que reciben estímulos del exterior, pertenecientes a la capa de entrada. Son las que en el modelo biológico están relacionadas con el aparato sensorial. En el caso de las RNA son las que obtienen la información de entrada a la red.

2. Esta información se transmite a los elementos internos de la red, los cuales se encargan de su procesamiento. Es en esta etapa en donde se genera cualquier representación interna de la información. Como estos elementos no tienen relación directa con la información de entrada ni la de salida, se denominan *unidades ocultas*. Estas unidades están dispuestas en una o varias capas ocultas.

3. Una vez hecho el procesamiento de la información, las unidades de salida se encargan de dar respuesta al sistema. Estas unidades de salida se agrupan en la capa de salida.

La neurona artificial es un elemento que posee un estado interno, llamado *estado de activación*, y recibe señales que le permiten cambiar de estado. Estos estados pueden definirse como un conjunto E, el cual contiene valores que definen el estado en el cual se puede encontrar la neurona en un determinado instante de tiempo. Puede utilizarse, por ejemplo, un conjunto E que contenga únicamente dos valores $E=\{0,1\}$, siendo 0 el estado inactivo y 1 el estado activo. Puede también utilizarse un conjunto con una mayor cantidad de valores, $E=\{0,1,2,...,n\}$ o un intervalo continuo de valores para representar estados.

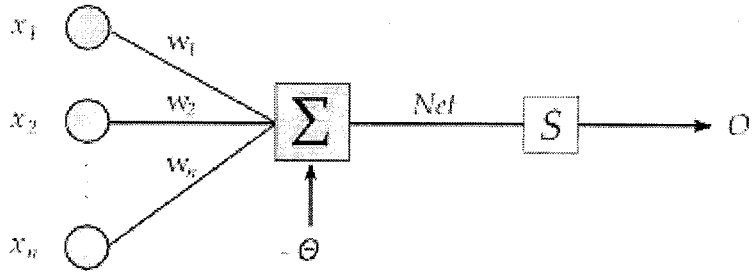


Ilustración 1.5: Esquema de una neurona artificial

Las neuronas poseen una función que les permite cambiar de estado de activación a partir de las señales que reciben; a dicha función se la denomina *función de activación*. El nivel o estado de activación de una neurona depende de las entradas recibidas por esta. Para calcular el estado de activación de una neurona, primero se debe calcular la entrada total de la misma. Esta entrada total, denominada Net, se calcula como la suma de todas las entradas de la neurona, ponderada por ciertos valores, denominados pesos, que son los que se modifican en el proceso de aprendizaje de la red neuronal. Si se definen las entradas como un

vector $\overline{X}$ de tamaño n, con elementos $x_1, x_2, \cdots, x_n$; y los pesos están definidos por un vector $\overline{W}$ el cual también es de tamaño n y con elementos $w_1, w_2, \cdots, w_n$; se define la entrada total como $Net = \left(\sum_{i=1}^{n} (x_i \cdot w_i) \right) - \theta$, donde θ es un valor numérico conocido como *umbral* y se utiliza para determinar a partir de que valor la neurona producirá una salida significativa. Para obtener el estado de activación de la neurona, se utiliza una función dependiente de esta entrada total, denominada *función de activación* ($F(Net)$). Generalmente la función de activación es la función identidad, es decir que la salida de la función es igual a la entrada $F(Net) = Net$, aunque pueden utilizarse otro tipo de funciones. La salida de la función de activación es utilizada luego para calcular el valor de salida de la neurona $O = S(F(Net))$, el cual en el caso de utilizar una función de activación del tipo identidad se reduce a $O = S(Net)$. En la Ilustración 1.5 se expone la estructura básica de una neurona artificial con función de activación del tipo identidad. La función $S(F(Net))$ determina el tipo de neurona que se está utilizando, existen algunas funciones que son utilizadas típicamente (como la escalón o la lineal), pero en la práctica se pueden utilizar otro tipo de funciones. [3] y [4]

Existen modelos muy diversos de redes neuronales artificiales, los cuales se diferencian entre si por sus filosofías de diseño, reglas de aprendizaje, topología de la red (en que forma las neuronas se conectan entre sí), entre otros factores. Cada modelo tiene sus ventajas y desventajas, y cada uno se adapta mejor a la resolución de un tipo específico de problema.

Las redes neuronales artificiales pueden aprender acerca del problema que se trata de resolver con ellas, al igual que las redes neuronales biológicas aprenden nuevos conocimientos.

Este aprendizaje puede ser de dos tipos: supervisado o no supervisado. Como cuando un niño aprende a sumar o restar en la escuela con la ayuda de un maestro que enseña y supervisa si lo que el niño esta haciendo es correcto o no, el aprendizaje supervisado en las redes neuronales artificiales trabaja con el mismo principio. Un operador humano debe "decirle" a la red si lo que esta haciendo es correcto o no.

El aprendizaje no supervisado al contrario, no requiera la intervención de un agente externo, sino que la red trata de determinar características de los datos de entrada: rasgos significativos, regularidades o redundancias. A este tipo de redes se las conoce también como sistemas autoorganizados, debido a que la red se ajusta dependiendo únicamente de los valores recibidos como entrada, como es el caso del modelo de Kohonen, el cual se describirá mas adelante.

Las redes neuronales cuentan con un gran número de aplicaciones hoy en día, ya que se pueden utilizar en cualquier problema donde principalmente se necesite aprender, razonar y tomar decisiones en base a lo aprendido, valiéndose de la clasificación de patrones y/o la aproximación de funciones. Una de las aplicaciones en las cuales se está dando gran importancia a las redes neuronales es la visión por computadora, ya que las redes neuronales en conjunto con técnicas de procesado de imágenes forman una herramienta de gran potencial para ser aplicada en este área.

1.3.2. Breve historia de las RNA

El interés por las redes neuronales data de los años 40, a partir del trabajo de McCulloch y Pitts (1943), donde propusieron modelos de neuronas en forma de dispositivos binarios basados en un umbral y algoritmos estocásticos que implicaban cambios binarios 0-1 y 1-0 en los estados de las neuronas como la base para el modelado de sistemas neuronales. En 1957, Frank Rosenblatt generalizó el modelo de células de McCulloch y Pitts añadiéndole aprendizaje, y llamó a este modelo *Perceptrón*. Primero desarrolló un modelo de dos niveles, que ajustaba los pesos de las conexiones entre los niveles de entrada y salida, en proporción al error entre la salida deseada y la salida obtenida. Rosenblatt intentó extender su modelo de aprendizaje a un perceptrón de tres niveles, pero no encontró un método matemático sólido para entrenar la capa de conexiones intermedia, conocida como capa oculta. En 1969, Minsky y Papert escribieron un libro en el cual describían las limitaciones de los perceptrones de una única capa. El libro tuvo un impacto tremendo, causando que muchos investigadores en el campo de las redes neuronales pierdan interés en el tema. El libro exponía matemáticamente que los perceptrones de una única capa no podían realizar algunas operaciones básicas de reconocimiento de patrones, como en el caso de que los patrones dados no sean separables linealmente. A pesar de que el interés en general y los fondos disponibles para la investigación sobre redes neuronales eran mínimos, algunos pocos investigadores continuaron trabajando en el tema. Teuvo Kohonen comenzó sus investigaciones sobre redes neuronales en 1971, estas investigaciones se centraron en memorias asociativas. Mas tarde realizó investigaciones en métodos de aprendizaje y desarrolló el LVQ (Learning Vector Quantization), un sistema de aprendizaje competitivo. En 1974 Werbos desarrolló la red *back-propagation*, la cual es en esencia un perceptrón con múltiples capas y con una función de umbral (que puede ser diferente en cada neurona) y una regla de aprendizaje que permiten resolver el problema de la separabilidad lineal de patrones. Estos y otros avances hicieron que el interés en el tema resurgiera en la década de 1980 [4].

1.3.3. Aplicaciones de las RNA

Las RNA son utilizadas actualmente en un gran número de aplicaciones diferentes y en diferentes campos, como la medicina, investigación, aplicaciones industriales, entre otros. Un área de gran interés es la aplicación de las RNA para el reconocimiento de imágenes, a continuación se exponen varios ejemplos diferentes de estas aplicaciones:

- Reconocimiento de huellas dactilares.
- Reconocimiento óptico de caracteres, sistemas conocidos por sus siglas en inglés como OCR (Optical Character Recognition). Esto puede ser utilizado en varias aplicaciones, como por ejemplo el reconocimiento de matrículas de vehículos a partir de imágenes de las matrículas o reconocimiento de caracteres en documentos manuscritos (para esto se han utilizado en algunos casos técnicas como SOM y LVQ, las cuales se describirán mas adelante).
- Reconocimiento de objetos basándose en su contorno.
- Reconocimiento de caras de personas.
- Sistemas médicos que ayudan a detectar irregularidades en imágenes, como por ejemplo radiografías.
- Sistemas industriales de clasificación y control de calidad.

Las RNA son utilizadas en estos y otros campos en donde el conocimiento adquirido con el tiempo sea de ayuda para resolver un problema. En una aplicación como la que se presenta en este documento y similares, las RNA ayudan a resolver el problema aprendiendo de los ejemplos (en este caso muestras de fallas en la superficie de la madera). Una vez que se adquirió el conocimiento necesario, se pueden aplicar para clasificar nuevos ejemplos en base a los conocimientos adquiridos previamente. Cabe destacar que no existen soluciones "mágicas" a estos problemas, si los datos de entrada a las RNA no representan características que puedan ayudar a resolver el problema, no se obtendrán posteriormente resultados satisfactorios.

En [5] se puede ver la aplicación de redes neuronales en conjunto con imágenes digitales para el control de calidad en la fabricación de textiles.

En [6] se expone la utilización de redes neuronales para la aplicación en un sistema de aprendizaje para el soporte a las decisiones médicas.

Para mas ejemplos de aplicaciones de las RNA, consultar [2].

1.4. Mapas Auto organizados de Kohonen (SOM)

Los *SOM (Self organizing maps)* o mapas auto organizados son un tipo de RNA de aprendizaje no supervisado, inventados por el profesor Teuvo Kohonen [7]. Estos mapas sirven para representar datos multidimensionales en menores dimensiones, como 2 o 3, las cuales pueden ser comprendidas de una forma mas clara por la mente humana. Esto es esencialmente una técnica de compresión de datos conocida como cuantización vectorial. Además de esto, la técnica de Kohonen crea una red que guarda la información de tal manera que también se mantienen las relaciones métricas entre la información, esto significa que la información que es similar se sitúa cerca en el mapa y a medida que crece la diferencia, se sitúa cada vez mas lejos.

1.4.1. La estructura del SOM

A continuación se explicará la estructura de un SOM bidimensional para hacerlo de forma mas simple, pero cabe recordar que puede tener cualquier número de dimensiones y el concepto sigue siendo aplicable.

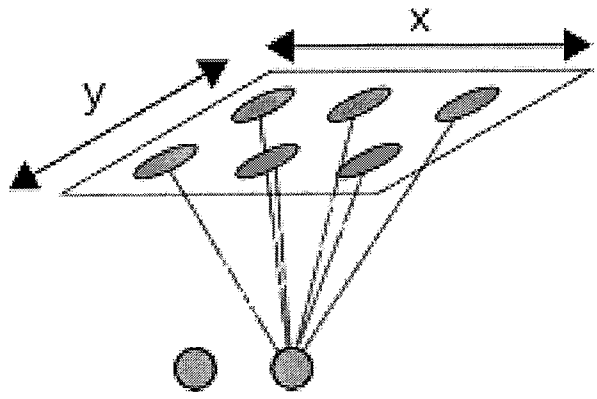


Ilustración 1.6: Estructura de un SOM bidimensional

En la Ilustración 1.6 se puede ver la estructura de un SOM bidimensional simple. Los elipses dentro del recuadro representan las *unidades* o *neuronas* del SOM, los cuales tienen una posición topológica específica en el mapa, en este caso bidimensional, la cual puede ser denotada como (x,y). Los círculos que están fuera del recuadro representan las entradas a la red, esto es un valor por cada círculo. Las líneas que unen las entradas con las unidades representan las conexiones entre ambas y tienen asociado un valor numérico real denominado *peso*, el cual representa la intensidad de la conexión. Cada entrada está conectada con todas las neuronas del mapa. [8]

Refinando el esquema anterior, se definen las entradas del mapa como un

conjunto de vectores v de n dimensiones o elementos. Los pesos de las conexiones entre un vector v y una unidad u son representados como un vector w el cual también es de n dimensiones, lo cual permite comparar ambos mediante una *función de distancia* que determinará el grado de semejanza que existe entre uno y otro.

Una introducción a la arquitectura y funcionamiento del SOM puede encontrarse en [9].

1.4.2. Funcionamiento del SOM

El SOM se puede utilizar para descubrir la relación que existe entre los datos de entrada, agrupándolos en diferentes clases. Cada neurona del mapa representa una clase, cuanto mas grande sea el tamaño del mapa, mas clases se tendrán. El mapa asocia cada entrada a una neurona a través de la función de distancia, la neurona que menor distancia tenga con respecto a la entrada en cuestión es la ganadora para esa entrada. Si a cada neurona se le asigna previamente una etiqueta que defina el nombre de la clase, se puede determinar la clase a la que corresponde la entrada simplemente verificando cual es su neurona asociada. Además de agrupar los datos de entrada en diferentes clases, el mapa mantiene una relación topológica entre estas clases manteniendo las clases similares juntas y a medida que aumenta la diferencia entre las clases estas son situadas cada vez mas lejos en el mapa.

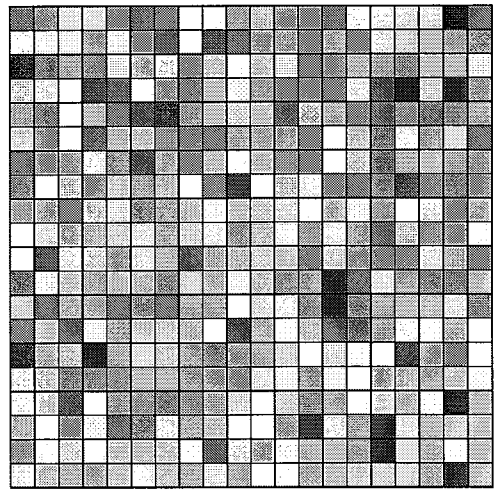


Ilustración 1.7: SOM sin entrenar

En la ilustración anterior, se presenta un SOM bidimensional de 20 x 20. Cada cuadrado en la ilustración representa una neurona del SOM. El color de relleno del cuadrado representa su valor. Este color es la combinación de tres valores numéricos, uno para el color rojo, uno para el verde y uno para el azul. Al mezclar

estos tres valores se obtiene el color final con el cual está relleno el cuadrado.

Al principio las neuronas son inicializadas con valores aleatorios, por eso aparecen colores al azar en el mapa. Suponiendo que las entradas a la red son cuatro y que estas tienen los valores que representan los colores rojo, verde, azul y naranja, se ve el resultado del entrenamiento del mapa en la Ilustración 1.8[4].

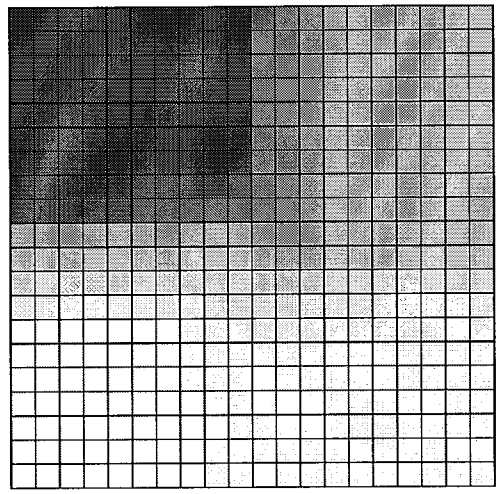


Ilustración 1.8: SOM entrenado

Como se ve en la ilustración, el mapa agrupa todos los colores similares en una misma región. A medida que se va cambiando de región el color cambia gradualmente, esto es debido a que existen solo cuatro clases diferentes en la entrada y sin embargo el mapa está configurado para diferenciar $20 \cdot 20 = 400$ clases (colores) diferentes, por lo que se muestran varias clases intermedias. El concepto a tener en cuenta es que todas las clases parecidas son agrupadas muy cerca unas de otras, mientras que clases diferentes están alejadas en el mapa.

Teniendo esto en cuenta, si se entrena el mapa con un conjunto determinado de datos, la ubicación de las regiones no varía si se repite posteriormente el entrenamiento con otro conjunto de datos similar, lo cual es una propiedad muy útil ya que permite etiquetar el mapa dividiéndolo en regiones y asignándole un nombre o etiqueta a cada región. Una vez que el mapa está entrenado basta con ver a que región del mapa (previamente etiquetada) corresponde la entrada presentada. Esta entrada será entonces de la clase definida por la etiqueta, como se ve en la Ilustración 1.9.

4 Captura de pantalla de un software propio, realizado para comprender el funcionamiento del SOM

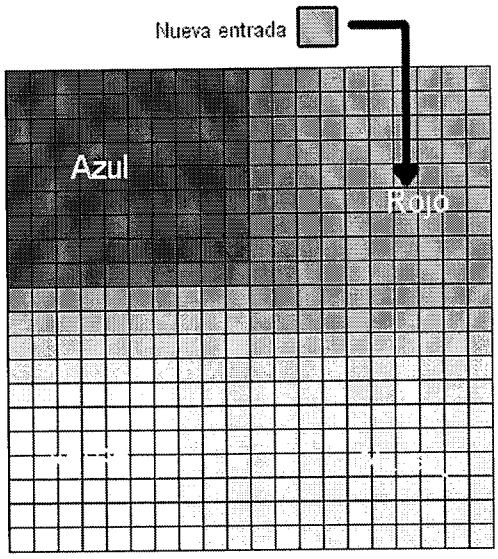


Ilustración 1.9: SOM etiquetado

1.4.3. La Matriz-U y la calidad del mapa

La utilidad principal del SOM es agrupar datos similares en regiones contiguas, por lo tanto el mapa será útil cuando esta agrupación sea correcta en el sentido de que resuelva el problema para el cual fue diseñado. En el ejemplo anterior se puede ver claramente como los colores están agrupados en diferentes regiones del mapa y que las fronteras entre cada agrupación están bien definidas. Para problemas mas complicados en los que intervienen mas de 3 variables puede ser muy difícil determinar si el mapa está correctamente configurado o si las entradas para este son correctas, debido a que resulta imposible en algunos casos representar gráficamente este tipo de información.

Para estos casos se utiliza una herramienta denominada *Matriz-U* (U-Matrix[5]) [10], la cual es una matriz que tiene las mismas dimensiones que el mapa a analizar. En cada posición de la matriz se encuentra un valor que mide la distancia que tiene la neurona que se encuentra en la misma posición del SOM con respecto a sus vecinos. Estos valores se calculan de acuerdo a la siguiente fórmula:

$$dist = \sum_{u2 \in NN(u)} d(w(u) - w(u2)) \qquad (2)$$

donde u es la neurona actual, $NN(u)$ es el entorno de la neurona u y $d(w(u)-w(u2))$ es la función de distancia utilizada en el SOM.

La matriz obtenida se puede representar como un gráfico de alturas, donde los valles representan un grupo homogéneo de datos y los picos representan las

5 Del inglés Unified Distance Matrix (Matriz unificada de distancias)

fronteras entre las clases.

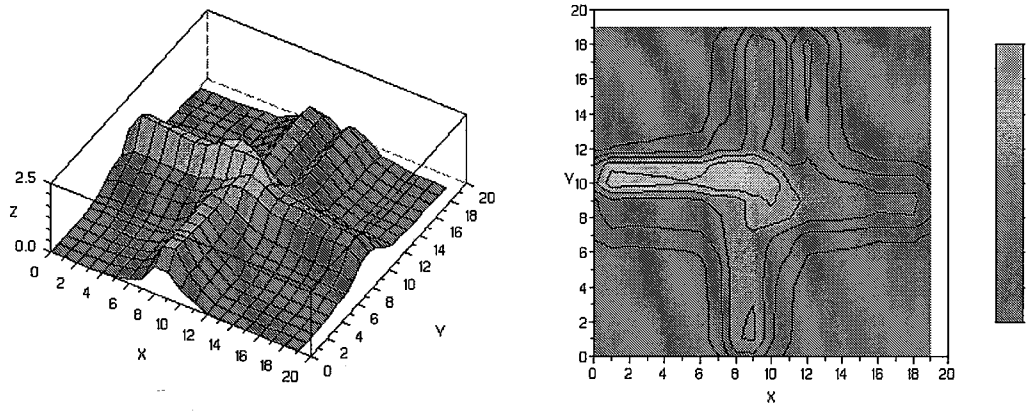


Ilustración 1.10: U-Matrix del SOM clasificador de colores

En la Ilustración 1.10 se puede ver la representación gráfica de la Matriz-U que se calculó en base al ejemplo de la sección anterior. Como se ve en el SOM, los colores están agrupados en las esquinas y las fronteras entre cada clase se encuentran en el centro del mapa, formando una figura similar a una cruz. En la Matriz-U se observa este mismo comportamiento. En la figura se muestran en color azul las zonas con menor altura, mientras que en rojo se ven las zonas mas altas. En la Matriz-U las zonas en azul se corresponden con las agrupaciones de clases, mientras que las zonas en rojo con las fronteras entre clases. A mayor altura, mas grande es la diferencia entre las clases.

En el ejemplo anterior la calidad del SOM se puede verificar a simple vista, pero como se verá mas adelante, la Matriz-U servirá para determinar la calidad de un SOM en el cual los datos de entrada no se pueden visualizar con tanta facilidad.

1.4.4. El algoritmo de aprendizaje

1.4.4.1. Introducción al algoritmo de aprendizaje del SOM

Los SOM utilizan un método de aprendizaje que es conocido como aprendizaje no supervisado, esto significa que no requiere supervisión por parte de una persona en su etapa de entrenamiento.

El entrenamiento consiste en una serie de iteraciones predefinidas en las cuales se selecciona una muestra al azar del conjunto de entrada y se la presenta al mapa. Mediante una función de distancia que determina el grado de semejanza entre una unidad del mapa y una muestra, se calcula la distancia que existe entre la muestra y todas las unidades del mapa, de a una por vez, y se selecciona la que

menor distancia tiene. Esta unidad es denominada *BMU* (*Best Matching Unit*) o *ganadora*. La BMU y el área de unidades que la rodean son modificadas para asemejarse a la muestra. Luego de varias iteraciones el mapa se estabiliza en un conjunto de zonas, donde las características similares se encuentran en lugares geométricamente cercanos. Estas zonas se pueden tomar como un clasificador, y cuanto mas grande sea el tamaño de la red (cantidad de unidades), mas clases existirán. [11]

Resumiendo, el algoritmo de aprendizaje puede plantearse en los siguientes pasos:

1. Se inicializan aleatoriamente los vectores de pesos.
2. Un vector del conjunto de muestras es seleccionado al azar y se lo presenta al mapa.
3. Se comparan las unidades, una a una, con la muestra presentada mediante una función de distancia y se selecciona la que tenga menor distancia. Esta unidad es etiquetada como BMU o ganadora.
4. Para saber que otras unidades entrenar, además de la BMU, se calcula el radio del entorno a entrenar. El radio comienza con un valor alto que abarca varias unidades del mapa, y a medida que avanzan las iteraciones se va disminuyendo, hasta que sólo la BMU pertenezca al entorno.
5. Los pesos asociados a cada unidad del entorno calculado en el paso anterior se modifican para hacer que la unidad en cuestión sea mas parecida a la muestra presentada. Cuanto más lejos esté una unidad de la BMU menos es modificada.
6. Se repite el algoritmo desde el paso 2 un número T de iteraciones.

A continuación se explicará el algoritmo de entrenamiento del mapa con un mayor grado de detalle.

1.4.5. El algoritmo de entrenamiento en detalle

1.4.5.1. Inicialización de los pesos

Los vectores de pesos son inicializados con valores aleatorios, típicamente con valores pequeños reales entre 0 y 1, pero esto puede variar dependiendo los valores utilizados y si las muestras están normalizadas o no. Se recomienda normalizar el conjunto de entrenamiento en un rango de [0, 1]. Si se conoce de antemano los valores aproximados de cada clase, se puede inicializar el mapa de manera tal que las clases se ubiquen siempre en el mismo lugar asignando los

valores del prototipo de cada clase en la ubicación deseada.

1.4.5.2. Calculando la BMU

Se selecciona al azar una muestra del conjunto de entrada, la cual es comparada con todas las unidades del mapa para ver cuál es mas semejante a la muestra presentada. Se mencionó anteriormente que esto se realizaba mediante una función de distancia. Esta función puede ser cualquier función que pueda medir el grado de semejanza entre una muestra y una unidad del mapa, con la condición de que si ambas son iguales la función se defina como 0, y aumente a medida que disminuye la semejanza entre la muestra y la unidad.

Típicamente se utiliza la distancia euclídea para realizar esta comparación. Se define esta distancia como:

$$dist=\sqrt{\sum_{i=1}^{n}(v_i-w_i)^2} \tag{3}$$

Donde $v=(v_1, v_2, \ldots, v_n)$ es el vector de entrada o muestra y $w=(w_1, w_2, \ldots, w_n)$ es el vector de pesos asociado a la unidad actual. En realidad por razones de economía de cálculo se utiliza en la práctica el cuadrado de esta distancia (con lo que se evita tomar la raíz cuadrada).

1.4.5.3. Calculando el entorno de la BMU

Una vez que la BMU es encontrada, es necesario encontrar las unidades cercanas para poder entrenarlas también. Este conjunto de unidades obtenidas se denomina entorno de la BMU o simplemente entorno. Existen dos formas de hacer esto, la primera es definiendo una relación entre las unidades, la cual definirá una estructura de entorno, como por ejemplo un entorno podría ser las unidades que están arriba, abajo, a la derecha e izquierda de la BMU.

Otra forma de hacerlo es calculando una función que determinará el radio del entorno, y luego buscar todas las unidades que estén dentro de este radio, donde la BMU será el centro del entorno. Además de esto, el radio del entorno debe disminuir en cada iteración, esto se logra mediante la siguiente función:

$$\sigma(t)=\sigma_0 e^{\left(-\frac{t}{\lambda}\right)} \quad 0 \leqslant t < T \tag{4}$$

donde la letra griega minúscula sigma (σ) representa denota el radio del entorno en la iteración t, y la letra lambda (λ) es una constante dependiente del radio inicial del entorno y el número de iteraciones (T) del algoritmo:

$$\lambda = \frac{T}{log(\sigma_0)} \quad (5)$$

Si se toma por ejemplo un ancho inicial de 20, y se realizaran 100 iteraciones, la función sería la siguiente:

$$\lambda = \frac{100}{log(20)} = 33.381$$

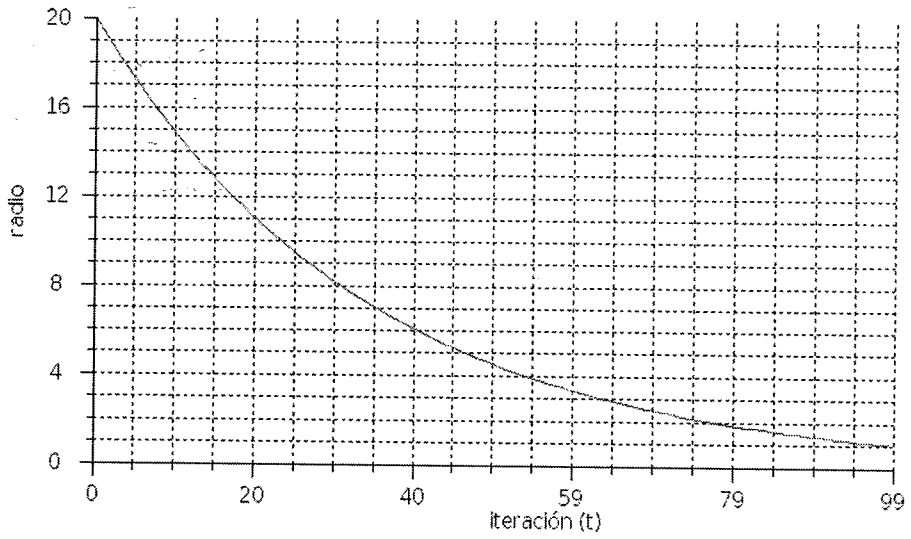


Ilustración 1.11: Función de variación del radio del entorno

$$\sigma(t) = 20e^{\left(-\frac{t}{33.381}\right)} \quad 0 \leq t < 100$$

Se puede apreciar como el radio del entorno decae en cada iteración, hasta que sólo queda una unidad en el entorno, la BMU.

Una vez que se sabe el radio, se buscan las unidades que están ubicadas dentro de éste y esas son las unidades a entrenar.

1.4.5.4. Entrenamiento de las unidades

Una vez encontrada la BMU y su entorno, se ajustan los vectores de pesos asociados a cada unidad según la siguiente fórmula:

$$w(t+1) = w(t) + \Theta(t, dist)\alpha(t)(v(t) - w(t)) \quad (6)$$

donde $w(t+1)$ es el nuevo peso, $w(t)$ es el peso actual, y $v(t)$ es el valor de la muestra. $\Theta(t, dist)$ es una función representa la influencia que tiene la distancia en la que se encuentra una unidad de la BMU en el aprendizaje, cabe destacar que cuanto mas cerca se encuentre, mas se deberán ajustar sus pesos para hacerla semejante a la muestra. Esta función se conoce como función de entorno. En este trabajo se utilizará una función de entorno de tipo gaussiana[6] (se pueden utilizar otros tipos de funciones de entorno), de la forma:

$$\Theta(t, dist) = e^{\left(-\frac{dist^2}{2\sigma(t)^2}\right)} \quad 0 \leqslant t < T \tag{7}$$

en donde $dist$ es la distancia en que una unidad se encuentra de la BMU y $\sigma(t)$ es el radio del entorno en la iteración t. T representa la cantidad total de iteraciones. Con esta función se logra que la tasa de aprendizaje varíe según la distancia que tiene una unidad de la BMU y que también varíe según la iteración del algoritmo, es decir, el tiempo. En otras palabras, con esta función se logra que cuanto mas lejos se encuentre una unidad de la BMU aprenda menos y cuanto mas cerca se encuentre aprenda mas. A medida que el tiempo avanza, menos unidades logran aprender algo debido a que el radio del entorno disminuye.

6 Llamada así en honor a Carl Friedrich Gauss

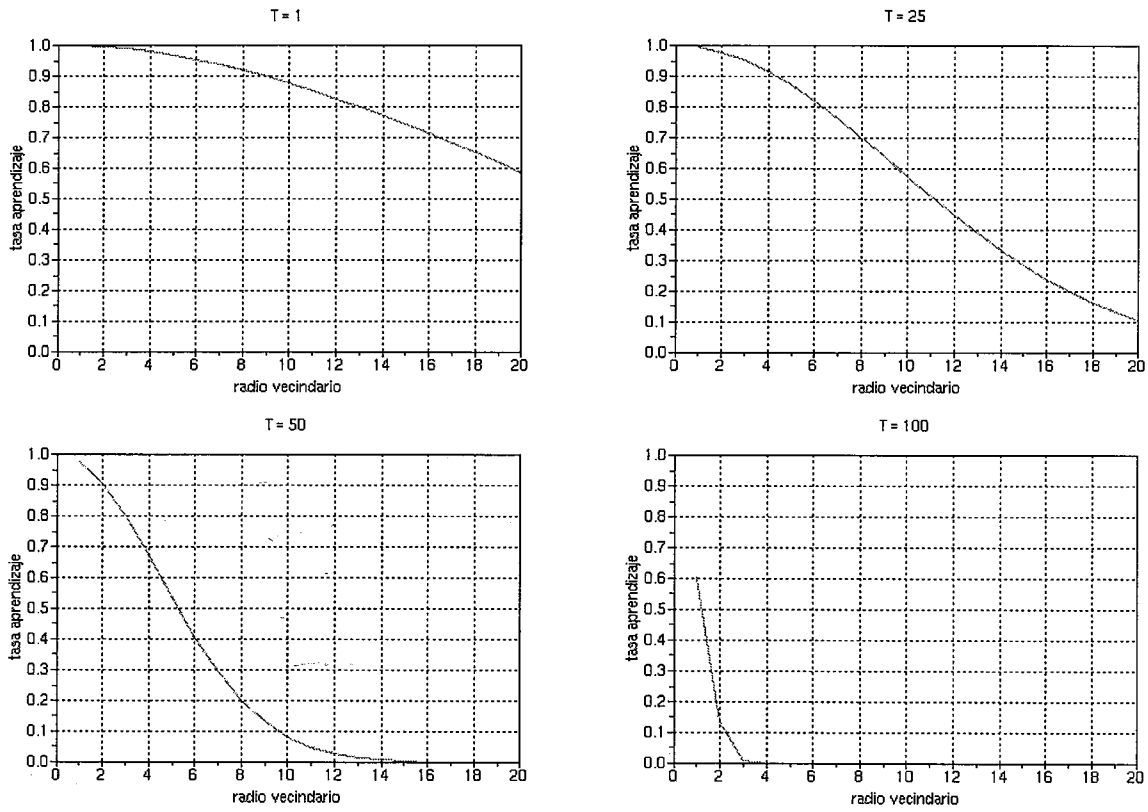


Ilustración 1.12: Influencia de la distancia y el tiempo en el aprendizaje

$\alpha(t)$ representa la tasa de aprendizaje en la iteración t, esto sería asignarle sólo un porcentaje del error (diferencia entre la muestra y la unidad). Esta tasa de aprendizaje decae en cada iteración según la siguiente fórmula:

$$\alpha(t)=\frac{\alpha_0}{1+t} \quad 0 \leqslant t < T \tag{8}$$

El algoritmo de entrenamiento se repite una cantidad de iteraciones predeterminada (T). Una vez finalizado el proceso de entrenamiento el mapa se encuentra listo para clasificar nuevos datos. La calidad del mapa depende de los parámetros utilizados en su entrenamiento, como ser el tamaño del mapa, la cantidad de iteraciones, la tasa de aprendizaje utilizada (α) y la función de distancia utilizada, pero la componente mas importante es el conjunto de datos de entrada. Se debe contar con un conjunto de muestras que sea representativo del dominio del problema a resolver. El mapa no es nada mas que un clasificador, si los datos a clasificar no se pueden dividir correctamente, el mapa producirá salidas acordes a las muestras de entrada que tiene; por lo tanto, si las entradas al mapa no son buenas, las salidas tampoco lo serán.

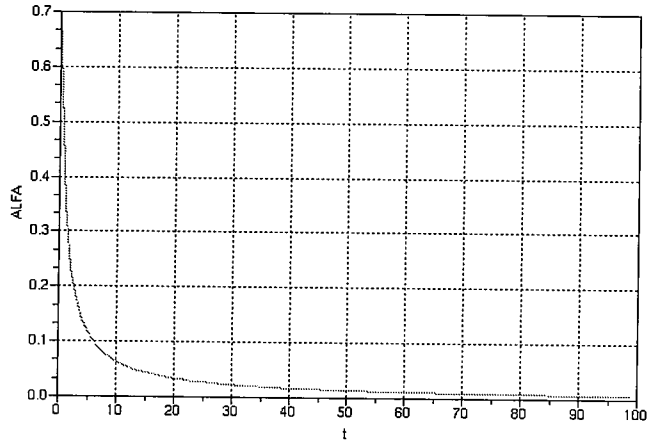


Ilustración 1.13: Alfa(t) con valor inicial 0.7

1.4.6. Pseudo Código del entrenamiento

```
EntrenarSOM(Ejemplos, Mapa){
      InicializarSOM(Ejemplos,lambda);
      t=0;
      Para cada ejemplo V en Ejemplos
      {
            G=DeterminarCeldaGanadora(Mapa,V);
            E=DeterminarEntorno(Mapa,G,lambda,t)
            Para cada celda W en E
                  NuevoValorSegunEjemplo(W,G,t,lambda);
            t=t+1;
      }
}
```

```
InicializarSOM(Ejemplos,lambda)
{
      N=NumeroDeEjemplosEn(Ejemplos);
      lambda=CalcularParametroDeAprendizaje(N);
}
```

```
DeterminarCeldaGanadora(Mapa,U)
{
      G= una celda cualquiera en Mapa;
      d2m=Distancia2(U,G);
      Para cada celda X en Mapa
      {
            d2=Distancia2(U,X);
            Si d2<d2m
            {
                  G=X;
                  d2m=d2;
            }
      }
      retornar G;
}
```

```
DeterminarEntorno(Mapa,G,lambda,t)
{
      Por economía se determinan las celdas en las que el nuevo
      ejemplo tendrá una influencia apreciable. El entorno en
      cuestión, centrado en G, es más pequeño a medida que crece
      t. El parámetro lambda es una medida de su radio inicial.
      En todo caso con cierto gasto puede obviarse este
      procedimiento considerando que el entorno es todo el mapa.
}
```

```
Distancia2(V,W)
{
     // esta función devuelve el cuadrado de la distancia
     // entre V y W

     N=dimensión del espacio de las celdas;
     d2=0;

     Para i=1 hasta N
          d2=d2+(V(i)-W(i))*(V(i)-W(i));
     retornar d2; // el cuadrado de la distancia
}
```

```
NuevoValorSegunEjemplo(W,V,t,lambda)
{
     // Este procedimiento modifica el valor de W de acuerdo
     // con el ejemplo V, el tiempo t y el parámetro lambda

     N=dimensión del espacio de las celdas;
     d2=Distancia2(W,V);
     beta=theta(lambda,t,d2)*alfa(t);

     Para i=1 hasta N
          W(i)=W(i)+(V(i)-W(i))*beta;
     retornar;
}
```

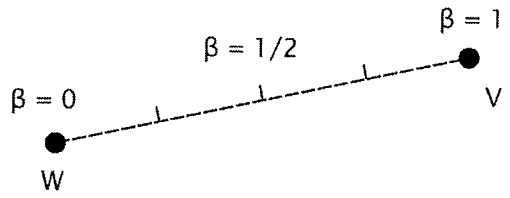


Ilustración 1.14: Porcentaje de aprendizaje

Se puede encontrar mas información acerca del SOM y su algoritmo de aprendizaje en [12].

1.5. Aprendizaje por cuantización vectorial

Las redes *LVQ* (*Learning Vector Quantization*) o *Redes de cuantización vectorial* [4] son un tipo híbrido de redes muy similares a los Mapas Auto organizados de Kohonen, pero estas utilizan aprendizaje supervisado, lo que significa que es necesario contar con "algo" que esté supervisando constantemente la fase de entrenamiento de la red. Por "algo" se entiende un sistema automático externo o una persona que esté monitorizando el proceso de aprendizaje y realizando correcciones a medida que estas sean necesarias.

1.5.1. Arquitectura de las redes LVQ

Este tipo de redes están compuestas por dos capas, la primera funciona de la misma manera que la única capa en las redes SOM, con una diferencia en la interpretación de su salida. En las redes SOM, la unidad ganadora o BMU representa una clase o grupo, en las redes LVQ, la unidad ganadora de esta capa representa una subclase, la cual pertenecerá a una clase o grupo final que está definida como una neurona o unidad de la segunda capa de la red, la cual es denominada capa de clasificación o clasificador. Esta capa actúa combinando varias subclases dentro de una misma clase. El número de neuronas en la primer capa debe ser mayor o igual que el número de neuronas en la segunda capa, esto es, se deben tener mayor número de subclases que de clases. Al igual que los SOM, cada valor de los vectores de entrada están conectados con todas las neuronas de la primera capa, y todas las conexiones de la red tienen un peso asociado que determina la fuerza de esa conexión. Los pesos de las conexiones entre las entradas y la primer capa son inicializados aleatoriamente y luego son ajustados mediante la regla de aprendizaje que se explicará mas adelante. Los pesos entre la primer capa y la segunda representan si una subclase (neurona de la primer capa) pertenece o no a una clase (neurona de la segunda capa). Esto se logra estableciendo como peso un valor de 1 (existe conexión) si la subclase pertenece a la clase o estableciendo este valor como 0 en caso contrario [13].

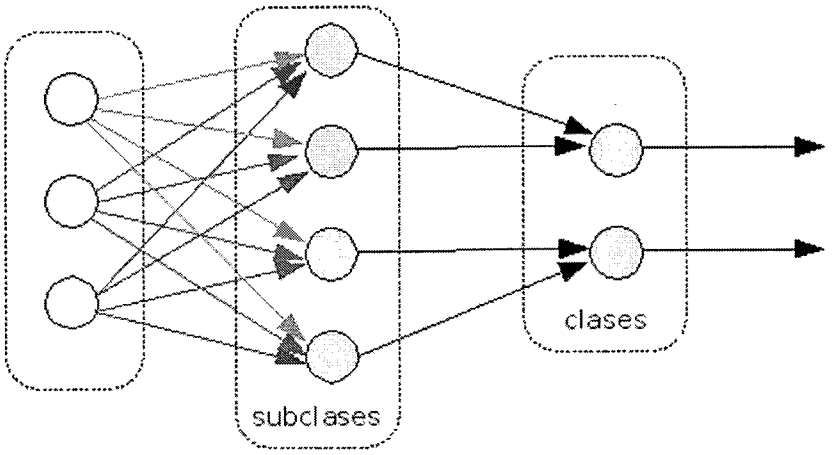


Ilustración 1.15: esquema de una red LVQ

Cada neurona de la segunda capa lleva una etiqueta asociada que determina el nombre de la clase que ésta representa. La neurona ganadora de la segunda capa determina entonces a que clase pertenece el vector de entrada presentado a la red. En este momento entra en juego el aprendizaje supervisado, si la clasificación es correcta se refuerzan las conexiones entre las entradas y la primer capa haciendo que la neurona ganadora sea mas parecida al vector de entrada. En caso contrario se penaliza a la neurona ganadora modificando sus pesos de forma tal de que sea diferente al vector de entrada presentado.

Como se puede apreciar, se elimina el concepto de entorno al entrenar únicamente la neurona ganadora, simplificando así el modelo.

La red LVQ es básicamente un clasificador lineal, es decir que utiliza ecuaciones lineales para separar el espacio de entrada. El clasificador actúa dividiendo el espacio de entrada mediante hiperplanos, en donde cada región conformada entre estos representa una clase. En el caso de tener dos neuronas de entrada, el clasificador trabaja con datos bidimensionales, lo cual permite visualizarlos como puntos en el plano, las regiones que forman las diferentes clases se representan como líneas. En el caso de tener tres neuronas se pueden visualizar los datos en un espacio tridimensional con las regiones separadas por planos. Para mas de tres dimensiones se utilizan hiperplanos para dividir el espacio de entrada.

En la Ilustración 1.16 se puede apreciar un ejemplo de una red LVQ bidimensional. Los puntos de diferentes colores representan los prototipos de cada subclase, cada color representa una clase. A mayor cantidad de subclases, mayor será la complejidad del clasificador, permitiendo discriminar datos con menor diferencia entre si. Las líneas en azul representan las fronteras entre las subclases y

las áreas coloreadas representan el espacio que abarca la clase correspondiente al color.

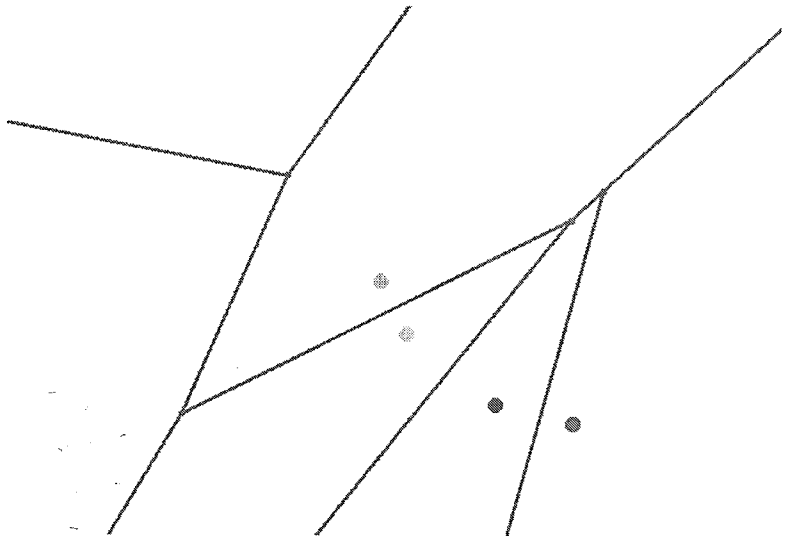


Ilustración 1.16: Clasificador LVQ

El diagrama formado en la figura anterior se conoce como *diagrama de Voronoi*, denominado así por su creador Gregory Voronoi [14]. Las regiones conformadas entre las fronteras se denominan regiones o *celdas de voronoi*. Para clasificar un nuevo punto, basta con determinar sus coordenadas en el plano y verificar a que celda de voronoi corresponde dicho punto. El punto entonces será de la clase representada por la celda de voronoi correspondiente. La forma en la cual son confeccionados estos diagramas se puede encontrar en [15].

La determinación de la celda es un proceso sencillo que consiste esencialmente en encontrar el prototipo más cercano. En términos generales puede describirse como sigue :

El espacio R^n se pone como la unión disjunta de un número finito de clases $C_1, C_2, \ldots, C_p$. Cada clase a su vez está definida por un conjunto finito de puntos de R^n que denominaremos sus representantes o prototipos, y un procedimiento que describiremos en seguida.

El conjunto de los representantes de la clase C_k será notado $R(C_k)=\{P_{k1}, P_{k2}, \ldots, P_{kq_k}\}$

La distancia de un punto cualquiera $x \in R^n$ a la clase C_k se define como la mínima distancia de x a un representante de C_k es decir

$$d(x, C_k)=d_k(x)=mín\{d(x, P)\, P \in c_K\}=mín\{d(x, P_{kj})\, j=1,2,\ldots,q_k\}$$

Cada punto pertenece a la clase que se encuentre a la menor distancia, y si hay más de una, a la de menor índice, o sea:

$$x \in C_k \Leftrightarrow ' \forall\, j=1,2,\ldots,p; d_k(x) \leqslant d_j(x),\ y\, si\ d_k(x)=d_j(x) \Rightarrow k \leqslant j '$$

1.5.2. Algoritmo de aprendizaje de la red LVQ

Se cuenta con un conjunto de muestras, compuesto por vectores $v=(v_1, v_2, \ldots, v_n)$, los cuales están etiquetados con el nombre de la clase a la cual representan.

También se cuenta el conjunto de neuronas de la primer capa de la red, $w=(w_1, w_2, \ldots, w_n)$. Estas neuronas están asociadas con las neuronas de la capa de salida, las cuales contienen la etiqueta de la clase que representan.

Se define como N la dimensión de los dos vectores.

Al igual que en el SOM, se cuenta con una función de distancia, típicamente la distancia euclídea, utilizada para comparar una muestra con una neurona. Cabe destacar que se puede utilizar otra función de distancia, de la misma forma que en el SOM.

El algoritmo de aprendizaje de la red se resume en los siguientes pasos:

1. Se selecciona un ejemplo del conjunto de muestras.
2. Se busca la neurona que tenga la menor distancia al ejemplo presentado. Esta es etiquetada como la ganadora.
3. Se comparan las clases que representan a la neurona ganadora y al ejemplo. Si las clases son iguales se refuerza el aprendizaje haciendo la neurona ganadora mas parecida a la clase de entrada. En el caso de que las clases sean distintas, se penaliza a la ganadora haciéndola mas distinta al ejemplo.
4. Se repiten los pasos 1 hasta el 3 hasta que se hayan seleccionado todos los ejemplos del conjunto.
5. Se repiten todos los pasos por un número T de iteraciones.

1.5.2.1. Selección de un ejemplo

Del conjunto de ejemplos de entrada a la red, se van seleccionando uno a uno los ejemplos de forma consecutiva y se los presentan a la red, hasta haberlos

seleccionados todos. Se completa una iteración al haber presentado todos los ejemplos a la red.

1.5.2.2. Búsqueda de la neurona ganadora

El ejemplo de entrada es comparado, mediante una función de distancia predefinida, con todas las neuronas de la primer capa de la red. La neurona que tenga la menor distancia al ejemplo presentado es marcada como la neurona ganadora.

1.5.2.3. Entrenamiento de la neurona ganadora

Se comparan las clases de la neurona ganadora y la del ejemplo de entrada. Si las clases son iguales se debe reforzar el aprendizaje, esto se logra con la siguiente fórmula:

$$w(t+1)=w(t)+\alpha(t)(v(t)-w(t)) \tag{9}$$

Si las clases son diferentes se penaliza el aprendizaje mediante la fórmula:

$$w(t+1)=w(t)-\alpha(t)(v(t)-w(t)) \tag{10}$$

Donde $0<\alpha(t)<1$, y puede ser un valor constante o decrecer monotonicamente en cada iteración.

Como se puede apreciar, el algoritmo de aprendizaje es muy similar al del SOM, con la diferencia de estar simplificado eliminando el concepto de entorno. Para mas información acerca del algoritmo de aprendizaje de la red LVQ y sus variaciones, ver [16].

1.5.2.4. Pseudo código

```
EntrenarLVQ(v, w){
      T = número total de iteraciones;
      t = iteración actual;
      N = dimensión de los vectores v y w;
      E = cantidad de entradas a utilizar;

      /** iteraciones del algoritmo */
      Para t = 0 hasta T - 1 {

            /** presenta todas las entradas a la red */
            Para e = 0 hasta E - 1 {
                  u = seleccionarSiguienteEntrada();
                  u2 = buscarGanadora(u, w);
                  entrenarNeurona(u, u2, w);
            }

      }
}
```

```
buscarGanadora(u, w) {
      // busca la neurona que sea mas parecida a la entrada u

      G = una neurona cualquiera de la red;
      d = Distancia(u, G);

      Para cada unidad u2 en la red
      {
            d2=Distancia(u, u2);
            Si d2<d
            {
                  G=u2;
                  d=d2;
            }
      }
      retornar G;
}
```

```
EntrenarNeurona(v, w){
      N = dimensión de las neuronas;
      alfa = tasa de aprendizaje; // 0 < alfa < 1

      Si clase(v) = clase(w){
            Para i = 0 hasta N - 1
                  W(i) = W(i) + alfa * (V(i) - W(i));
      }
      Sino {
            Para i = 0 hasta N - 1
                  W(i) = W(i) - alfa * (V(i) - W(i));
      }
}
```

CAPÍTULO 2 - MÉTODOS

En este capítulo se describen las diferentes técnicas utilizadas durante el desarrollo del trabajo.

2.1. Adquisición de las muestras

Las muestras utilizadas para realizar las pruebas con el sistema fueron tomadas utilizando una cámara digital de uso doméstico. El tamaño de las placas de muestra es de 2,10 x 1,60 metros y la resolución original de las imágenes es de 2048 x 1536 píxeles. Estas imágenes fueron luego recortadas utilizando un software de procesamiento de imágenes digitales para eliminar algunos bordes y fallas de iluminación (se requieren equipos mas adecuados para obtener las muestras, los utilizados son de uso doméstico). El tamaño final de las imágenes a utilizar es de 1792 x 1280 píxeles[7], lo cual es equivalente a una resolución de 2,3 megapixels, que está dentro de los valores de cualquier cámara digital de uso doméstico que se puede conseguir en el mercado hoy en día. El área analizada de las placas es de entonces 1,86 x 1,33 metros (se recuerda que cada píxel representa una región de 1,04 x 1,04 mm de la placa). La elección de esta resolución es un balance entre el tamaño de los defectos que se intentan reconocer y el tamaño total de la imagen. Cabe destacar que cuanto mas grande es la imagen, mas tiempo de procesamiento se necesita para llevar a cabo la tarea de clasificación, pero cuanto mas pequeña es se pueden detectar menor cantidad de defectos.

Para obtener las muestras, las placas son posicionadas en forma vertical, valiéndose de un apoyo en su parte posterior para permanecer en pié. La iluminación se realiza con dos reflectores de 300 watts ubicados detrás de la cámara, la cual está montada en un trípode a 2.5 metros de la placa.

7 Cada pixel consta de 24 bits de manera que se tienen $2^{24}=16777216$ colores.

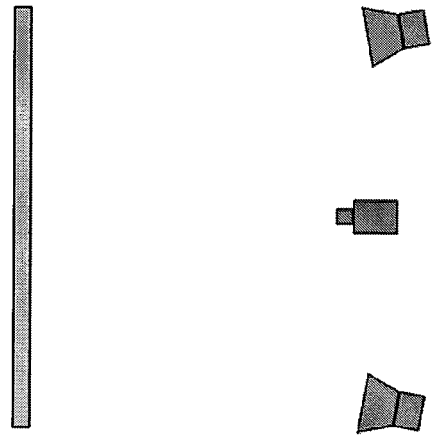


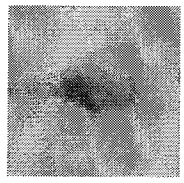
Ilustración 2.1: Adquisición de datos

La muestra total consiste en 60 imágenes, 30 para paneles de pino y 30 para paneles de guatambú. A partir de estas muestras de placas se tomaron muestras de cada tipo de defecto que se busca reconocer con el sistema. Se obtuvieron 16 muestras para cada tipo de defecto, seleccionadas de placas al azar.

2.1.1. Clasificación de las muestras en tipos de fallas

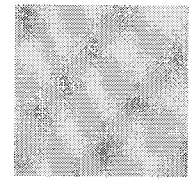
Las muestras de fallas en placas fueron etiquetadas con el tipo de defecto correspondiente.

Para las placas de pino estas clases son:



Nudo

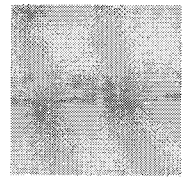
Hueco



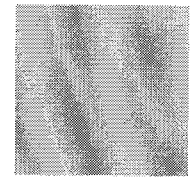
Sano

Para las placas de guatambú:

Nudo



Mancha



Sano

Las muestras que se ven en las imágenes anteriores tienen un tamaño de 60

milímetros de lado en la placa de madera. Estas muestras fueron obtenidas a partir de la imagen de la placa completa, y no por separado.

El conjunto de muestras para entrenar a la red LVQ es dividido en las clases descriptas anteriormente.

Actualmente los expertos realizan la clasificación basándose en estas clases y algunas otras cuestiones que no son inherentes a la madera en sí, como por ejemplo alguna mancha ocasionada por el mal funcionamiento de alguna máquina o fallas en el pegado o la unión de diferentes láminas que forman la cara del tablero, entre otras.

Las fallas agrupadas en las clases anteriores son fácilmente identificables por un operador especializado, pero puede darse el caso de una falla como un nudo que se parezca a un hueco. La manera de resolver este tipo de semejanzas varía dependiendo del criterio del operador, algunos simplemente clasifican la placa como de calidad inferior sin importar que tipo de falla contenga, otros verifican la severidad de la falla (como por ejemplo el tamaño de la misma) y si no es de mayor importancia, dejan pasar la falla y clasifican la placa como de mayor calidad. Este es uno de los principales problemas en la actualidad, la falta de criterios homogéneos a la hora de realizar la clasificación.

2.2. Implementación del prototipo

El prototipo del sistema fue implementado en Java, por su simplicidad en el manejo de objetos y las facilidades para la creación de interfaces gráficas de usuario. A pesar de que los programas en Java son interpretados por una máquina virtual, haciéndolos mas lentos que programas compilados, al momento de ejecutarse los resultados de performance son bastante satisfactorios. En el caso de implementarse un sistema de producción, podemos observar dos alternativas:

- Re-Implementar el sistema en otro lenguaje que brinde una mayor performance, como por ejemplo C++.
- Re-Implementar únicamente los métodos que son intensivos en cálculo (que demandan mucho poder de procesamiento) en otro lenguaje como C++ o Ensamblador, aprovechando las facilidades que brinda Java para implementar métodos nativos en estos lenguajes.

El prototipo fue desarrollado y probado en una PC Pentium 4 con 512mb de memoria RAM, bajo una plataforma Linux. El entorno de desarrollo utilizado fue NetBeans (www.netbeans.org), el cual es de código abierto y gratuito.

2.3. Puesta en marcha del sistema

La puesta en marcha del prototipo se divide básicamente en dos etapas:

1. Entrenamiento de las redes neuronales utilizadas para la clasificación y el reconocimiento

2. Utilización propiamente dicha del sistema para clasificar alguna especie determinada de madera.

2.3.1. Etapa de entrenamiento

El sistema debe ser configurado y entrenado antes de pasar al proceso de clasificación. Esta configuración consiste en crear las clases en el sistema para cada tipo de defecto que se pretende detectar. A cada tipo de defecto se le asigna un color con el cual este será identificado posteriormente.

Una vez creadas las clases, el sistema está listo para entrenarse. Tanto el SOM como la red LVQ deben ser entrenados.

El entrenamiento del SOM se realiza con muestras de placas completas, una vez cargada la muestra, el sistema se entrena basándose en sus parámetros internos. Al terminar el entrenamiento con una placa en particular, el operador debe marcar las zonas con defectos haciendo click sobre ellas con el mouse para diferenciarlas de las sanas. Hay dos formas diferentes de marcar las zonas defectuosas. La primera consiste en marcarlas sobre la imagen de la placa, en este caso el sistema verifica automáticamente a que neurona del SOM corresponde esa entrada y marca todas las demás entradas asociadas a esa neurona. La segunda forma es marcando sobre el mapa la neurona que se considera que representa una zona defectuosa. El entrenamiento se repite con la cantidad de placas deseada hasta que la separación de zonas con defectos de las sanas sea satisfactoria para el operador. Al cargar una nueva imagen el sistema marca automáticamente las regiones que corresponden a zonas defectuosas en el mapa.

Para el entrenamiento de la red LVQ, se cargan muestras en el sistema para cada tipo de defecto. Al momento de la carga de las muestras el operador selecciona a que defecto corresponde la muestra, de una lista en la cual aparecen las clases previamente creadas. Una vez cargadas las muestras se entrena la red LVQ hasta que la clasificación de las muestras sea satisfactoria.

Se verá mas adelante la influencia que tiene la variación de los parámetros internos del sistema en el resultado de la clasificación.

2.3.2. Etapa de clasificación

En la etapa de clasificación el sistema ya está entrenado y listo para realizar la detección y el reconocimiento de cada tipo de falla.

La etapa de clasificación se divide en dos fases. Primero se detectan las regiones con posibles defectos, lo cual se realiza en la fase de detección de fallas. Luego estas regiones pasan a una segunda inspección con el fin de reconocer a que tipo de falla pertenece la región con posibles defectos. Esta es la fase de reconocimiento de fallas.

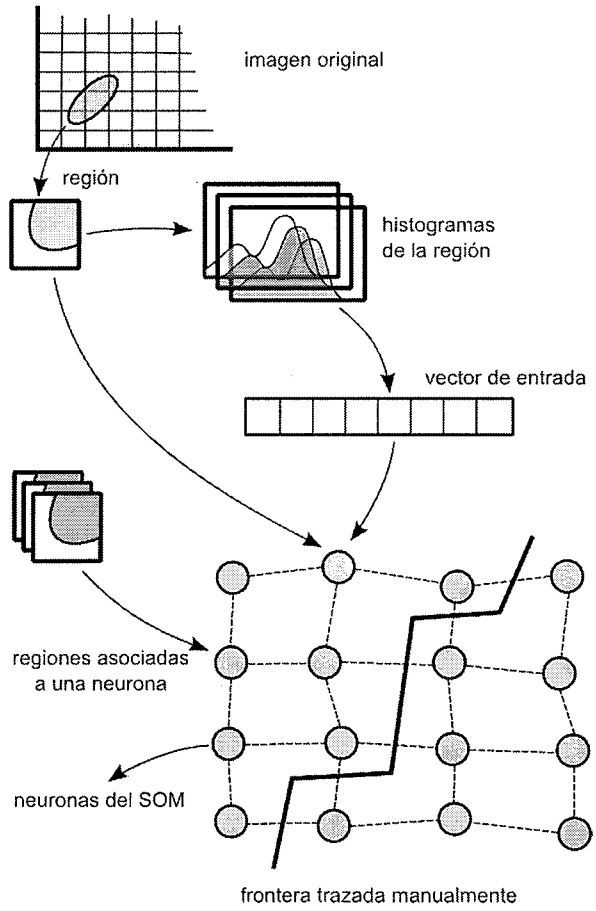


Ilustración 2.2: Esquema de la fase de detección

La Ilustración 2.2 presenta un esquema del funcionamiento de la fase de detección de fallas. Como entrada se cuenta con la imagen original de la placa a analizar. Esta imagen es dividida en regiones uniformes de un tamaño determinado, en el caso del sistema presentado de 24 x 24 píxeles. Estos 24 x 24 píxeles equivalen a una región de 24,9 x 24,9 milímetros en las imágenes utilizadas. Luego se calculan los histogramas para cada canal de color, correspondientes a cada una de estas regiones. Basándose en la información de los histogramas, se extraen diferentes características, las cuales se agrupan en un

vector de entrada. Se obtiene entonces un vector de entrada por cada región de la imagen original. Estos vectores sirven como entradas al SOM. Cada una de las regiones es presentada al SOM, con el fin de encontrar la BMU para esa región. Una vez encontrada la BMU para una región determinada, se asocia la imagen de la región correspondiente con la BMU. Se obtiene así un mapa en el cual todas las regiones de la imagen están asociadas con una neurona del mapa. Se mencionó anteriormente que el operador marca las zonas defectuosas de la placa, internamente esto equivale a definir una frontera entre las neuronas marcadas como sanas y las neuronas marcadas como defectuosas. Se puede apreciar esta división en la Ilustración 2.2[8].

Una vez finalizada la fase de detección de fallas, se pasa a la fase de reconocimiento. Las entradas de esta fase se obtienen a partir de un análisis de conectividad, aplicado a las regiones posiblemente defectuosas obtenidas en la fase de detección. El análisis de conectividad es simplemente un algoritmo que analiza que regiones son contiguas unas de otras. Esto se realiza porque una falla puede abarcar mas de una región de 24 x 24 píxeles, por lo tanto se asume que todas las regiones marcadas como defectuosas y que son contiguas constituyen una sola falla. A partir de este análisis se generan las imágenes de las posibles fallas uniendo las regiones contiguas en una sola imagen la cual se denomina zona defectuosa. A continuación se presenta el pseudo-código del algoritmo utilizado para realizar esta tarea. Este algoritmo es para analizar componentes conectadas en una imagen binaria, en el sistema propuesto cada píxel de esta imagen sería una región, y cada región está marcada con un valor binario (sana o defectuosa), por lo que el algoritmo es igualmente aplicable.

Se dispone de una imagen binaria I de NLINEAS y cada línea con NPIXELS y la etiqueta ETIQUETA. La función NUEVAETIQUETA genera una nueva etiqueta con valor entero cada vez que es llamada. La función VECINOS devuelve el conjunto de los vecinos ya etiquetados de un determinado píxel (región) en su misma línea o en la línea previa. La función ETIQUETAS, cuando se le proporciona el conjunto de píxeles ya etiquetados, devuelve el conjunto de sus etiquetas. Finalmente, la función MIN devuelve la etiqueta con menor valor de un conjunto de etiquetas. La letra "U" representa la unión de conjuntos.

8 Imagen confeccionada utilizando la imagen presentada en [17] como referencia.

```
Procedimiento iterar
        "inicialización de cada píxel de valor 1 a una etiqueta"
        for L = 1 to NLINEAS
                for P = 1 to NPIXELES
                        if I(L, P) == 1
                                ETIQUETA(L, P) = NUEVAETIQUETA()
                        else
                                ETIQUETA(L, P) = 0
                        end if
                end for
        end for

        "iteración de arriba-abajo seguida por pasos de abajo-arriba"

        REPETIR hasta que la variable CAMBIO sea FALSO

            "Paso de arriba-abajo"
            CAMBIO = FALSO

            for L = 1 to NLINEAS
                    for P = 1 to NPIXELES
                            if ETIQUETA(L, P) != 0
                                    M = MIN(ETIQUETAS(VECINOS((L, P)) U (L, P)))
                                    if M != ETIQUETA(L, P)
                                            CAMBIO = VERDADERO
                                    end if
                            end if
                    end for
            end for

            "Paso de abajo-arriba"
            for L = NLINEAS to 1 (decremento -1)
                    for P = NPIXELES to 1 (decremento -1)
                            if ETIQUETA(L, P) != 0

                                    M = MIN(ETIQUETAS(VECINOS((L, P)) U (L, P)))
                                    if M != ETIQUETA(L, P)
                                            CAMBIO = VERDADERO
                                            ETIQUETA(L, P) = M
                                    end if
                            end if
                    end for
            end for
        fin REPETIR
Fin iterar
```

Para mas información acerca del análisis de conectividad, se explican en detalle dos algoritmos en [2].

A cada imagen obtenida por el análisis de conectividad, se le aplican filtros de realce de imagen, para aislar las partes de la imagen que pertenecen a la falla de las que pertenecen al fondo de la imagen.

La imagen filtrada es luego procesada para obtener los vectores de entrada

para la red neuronal de reconocimiento (LVQ). Para obtener estas características se pueden utilizar técnicas de un mayor costo computacional que las utilizadas en para el SOM, ya que la cantidad de datos a procesar es mucho menor. Se puede analizar cada imagen para obtener diferentes medidas de la falla, como su relación de aspecto, rotación, tamaño, etc.

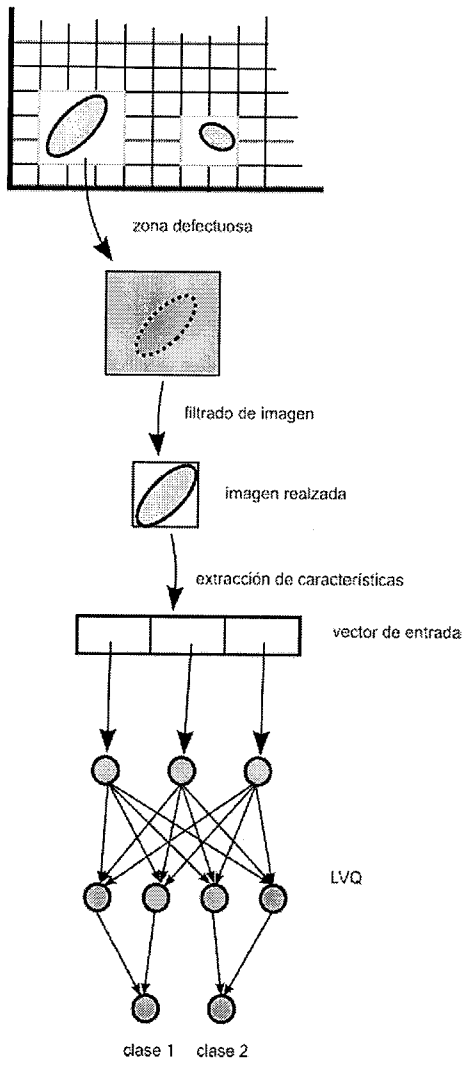


Ilustración 2.3: Fase de reconocimiento

Cada zona defectuosa es analizada por la red neuronal de reconocimiento (LVQ). El resultado del reconocimiento es marcado en una imagen de la placa, con el color previamente asignado al tipo de defecto asociado al resultado. De esta manera se puede apreciar en la imagen original el resultado del proceso de clasificación.

El motivo por el cual se realiza una "doble clasificación" (separar partes con posibles defectos de las sanas y luego reconocerlas) es que al eliminar las partes sanas de la imagen original, se reduce enormemente la cantidad de regiones a

analizar. Gracias a esto, en la etapa de reconocimiento se pueden utilizar algoritmos y técnicas de reconocimiento de imágenes mas avanzadas y que consumen mayor cantidad de recursos de procesamiento y memoria, haciéndolas por lo tanto mucho mas lentas.

2.4. Vectores de entrada de las redes

Los valores (o características) extraídos para cada red se agrupan en un conjunto de vectores de tamaño n, donde n representa el número de características extraídas. Estos vectores representan las entradas a cada una de las redes.

Para obtener las entradas del SOM, como se mencionó con anterioridad, se divide la imagen de una placa completa en pequeñas regiones de tamaño uniforme, denominadas regiones elementales (RE), las cuales son de 24 x 24 píxeles en el caso del sistema desarrollado. Luego se extraen las características para cada una de estas regiones conformando así el conjunto de entrenamiento para la imagen de placa en cuestión. Se mencionó que el tamaño de las imágenes de las placas es de 1792 x 1280 píxeles, por lo tanto se tendrán 1792 / 24 = 74 regiones a lo ancho y 1280 / 24 = 53 regiones a lo alto, sin tener en cuenta los decimales depreciados en la división no entera. Estos decimales equivalen a un pequeño margen de 16 píxeles en el borde derecho de la imagen y de 8 píxeles en el borde inferior de la imagen. Teniendo esto en cuenta se obtienen $74 \cdot 53 = 3922$ vectores de entrada por cada imagen a utilizar.

En el caso de la red LVQ, las entradas son imágenes de zonas defectuosas de la placa (ZD), las cuales son obtenidas a partir de la imagen original de una placa, tomando únicamente la zona de interés que contiene el defecto. Para la fase de entrenamiento de la red, esto se realiza manualmente por un operador. En la fase de operación, estas entradas se obtienen a partir de las RE marcadas como defectuosas en el SOM.

2.4.1. Características utilizadas

Las características a utilizar como entradas dependen del problema que se quiera resolver, es decir que hay que tener cuidado al definir estos conjuntos. En nuestro caso, se puede pensar en una primera instancia que un defecto es una zona visualmente anormal en la superficie de la placa de madera, es decir, que si la madera tiene una superficie de color homogéneo o con vetas regulares, un defecto es "algo" que tiene un color bien diferenciable con respecto al fondo. Una primera aproximación sería tomar esta propiedad y utilizarla para clasificar los defectos.

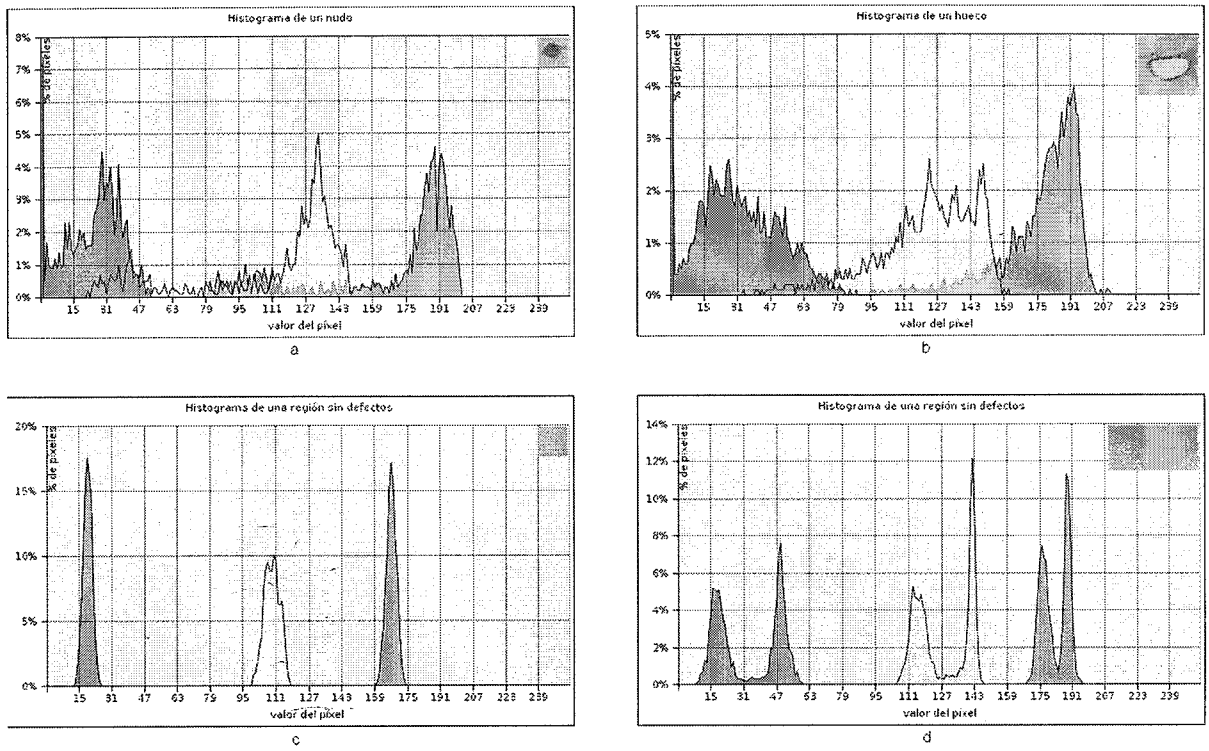


Ilustración 2.4: Histogramas de distintas regiones de una placa. Cada pixel consta de tres bytes correspondientes a los colores primarios (rojo, verde y azul, o RGB en inglés). Los Histogramas indican el porcentaje de pixels en cada zona para cada uno de los canales, es decir los colores.

Existen varios tipos de propiedades o descriptores de color para una región determinada, como el promedio del nivel de intensidad, la desviación estándar, percentiles y otras propiedades estadísticas aplicables al histograma de la imagen.

Todos los descriptores utilizados en este trabajo se extraen del histograma de la imagen. Para obtener el histograma de una imagen, basta con recorrerla, píxel a píxel, y obtener el nivel de intensidad para cada canal del píxel actual. Se tienen vectores que representan los canales que posee la imagen, la longitud de los mismos está determinada por la cantidad de intensidad posibles para un píxel Suponiendo que se trabaja con una imagen RGB de 24 bits de profundidad de color, se obtienen tres vectores (R, G y B) de 8 bits cada uno ($2^8=256$ niveles de intensidad por vector). Todas las posiciones de los vectores son inicializadas con un valor de 0. El cálculo del histograma consiste en recorrer la imagen, de a un píxel por vez, y determinar el nivel de intensidad para cada canal. Una vez determinado el nivel de intensidad, se incrementa en 1 el vector que representa el canal en cuestión, en la posición indicada por el valor del nivel de intensidad correspondiente al píxel actual. Una vez recorrida toda la imagen, se obtienen tres vectores que representan el histograma para cada canal de la imagen.

Como se puede ver, el cálculo del histograma es una técnica de bajo costo

computacional, ya que requiere un solo acceso a cada píxel y los cálculos que intervienen en la operación son únicamente sumas de números enteros.

Se recuerda que el histograma es una función de distribución de frecuencias, teniendo esto en cuenta, se pueden utilizar las medidas estadísticas basadas en esta función como descriptores, todas con sus ventajas y desventajas. A continuación se explicarán algunas medidas estadísticas estudiadas para la detección de fallas.

2.4.1.1. Promedio de color

Una propiedad interesante del histograma es el promedio del nivel de intensidad. Un promedio alto significa que la imagen es bastante clara (tiene mucho brillo), mientras que un promedio bajo indica una imagen oscura. Teniendo en cuenta que la mayoría de los defectos tienen un color bien diferenciado (generalmente oscuro) al resto de la madera, este descriptor podría ser utilizado para la discriminación. El problema del promedio es que es muy sensible a valores extremos, por lo tanto, si existen regiones con vetas el promedio no sería parecido a una región homogénea; o lo que es aún peor, podría parecerse al promedio de una región con defectos. Las regiones sanas tienen un promedio de intensidad generalmente alto, otro problema del promedio es que una región sana puede tener un promedio muy bajo en el caso de que la iluminación no sea homogénea y por lo tanto confundir una región sana con una falla. Esto se puede solucionar con un buen esquema de iluminación.

2.4.1.2. Desviación estándar

En la Ilustración 2.4 se puede apreciar cuatro histogramas y las imágenes correspondientes a cada uno. En 2.4(a) y 2.4(b) se presentan los histogramas de dos regiones defectuosas, correspondientes a un nudo y a un hueco respectivamente. En 2.4(c) y 2.4(d) se ven los histogramas correspondientes a regiones sin defectos, la primera no contiene vetas pero la última si. A simple vista se puede observar que en los histogramas correspondientes a las imágenes sin defectos, los valores para los canales R, G y B tienen muy poca dispersión (son muy similares); mientras que en los histogramas correspondientes a las imágenes con defectos esta dispersión es mucho mayor. Podría entonces utilizarse una medida de dispersión, como la desviación estándar para realizar una separación entre regiones sanas y defectuosas. El problema de las medidas de dispersión es que al haber un cambio en el color, como en una región con vetas, la dispersión aumenta, pudiendo asemejarse a los valores obtenidos para una región con defectos, y por lo tanto la

clasificación no sería buena.

2.4.1.3. Segmentos del histograma

Se puede utilizar también directamente el histograma de la imagen como descriptor. Obtendríamos así un vector de entradas de 256 posiciones por cada canal para una imagen con 8 bits de información por canal. Al tener 3 canales (R, G y B) obtenemos $256 \cdot 3 = 768$ valores de entrada. Esto no resulta muy práctico ya que se requiere mucho poder de cómputo para procesar todos los datos, y además existirán muchas entradas con valor 0, como se puede apreciar en los histogramas de la Ilustración 2.4. Esta desventaja es también una ventaja ya que se puede dividir el histograma en particiones y utilizar a estas como descriptores. Si se tiene un histograma de 256 niveles podemos dividirlo, por ejemplo en 4, tomando particiones de 64 valores $256/4 = 64$. Viendo el histograma de una imagen sin defectos, se puede observar que en la primer partición de 64 valores ($[0,63]$) para el canal verde no hay muestras; mientras que en una imagen con defectos en el mismo rango si se obtienen valores. Lo mismo ocurre con el canal rojo, pero en el segmento comprendido entre los valores $[64,127]$. El tamaño y la cantidad de las particiones a utilizar se calcula en base a las muestras de los datos que se tiene. Los valores aquí propuestos fueron obtenidos mediante la observación de los histogramas de varias muestras de maderas de diferente tamaño y en todas las observaciones se pudo apreciar que las muestras correspondientes a regiones sin defectos no contienen píxeles para los valores propuestos.

2.4.1.4. Percentiles

El $p-ésimo$ percentil es un valor tal que por lo menos un p por ciento de los elementos tienen dicho valor o menos, y al menos, un $(100-p)$ por ciento de los elementos tienen este valor o mas. En el histograma de una imagen, un percentil p representa el valor para el cual el $p\%$ de los píxeles tienen dicho valor de color. Los percentiles también se pueden utilizar para nuestro propósito, como se puede ver en [17].

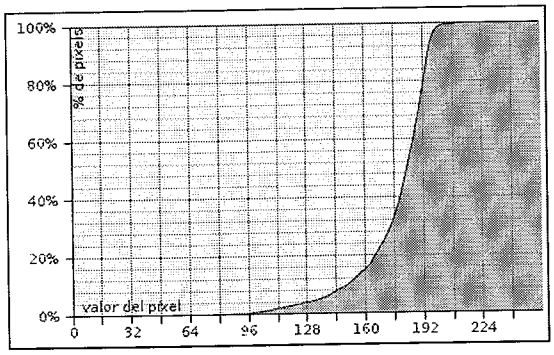


Ilustración 2.5: Histograma acumulativo

El cálculo de los percentiles es una operación sencilla. Si se cuenta con un histograma acumulativo como el de la Ilustración 2.5. Este tipo de histograma tiene valores que siempre son crecientes ya que un valor determinado se define como la suma de todos los valores anteriores. Como en cualquier función de distribución de probabilidades, la suma de los todos los valores debe ser 1 (el 100% de los datos). Una vez que se tiene el histograma acumulativo, basta con realizar una búsqueda en el mismo para saber el valor de un percentil determinado. Por ejemplo, en la Ilustración 2.5, se puede observar que el 80-percentil (eje de las abscisas) tiene el valor 192 (eje de las ordenadas). En este caso se puede observar que el 80% de los datos tienen un valor menor a 192.

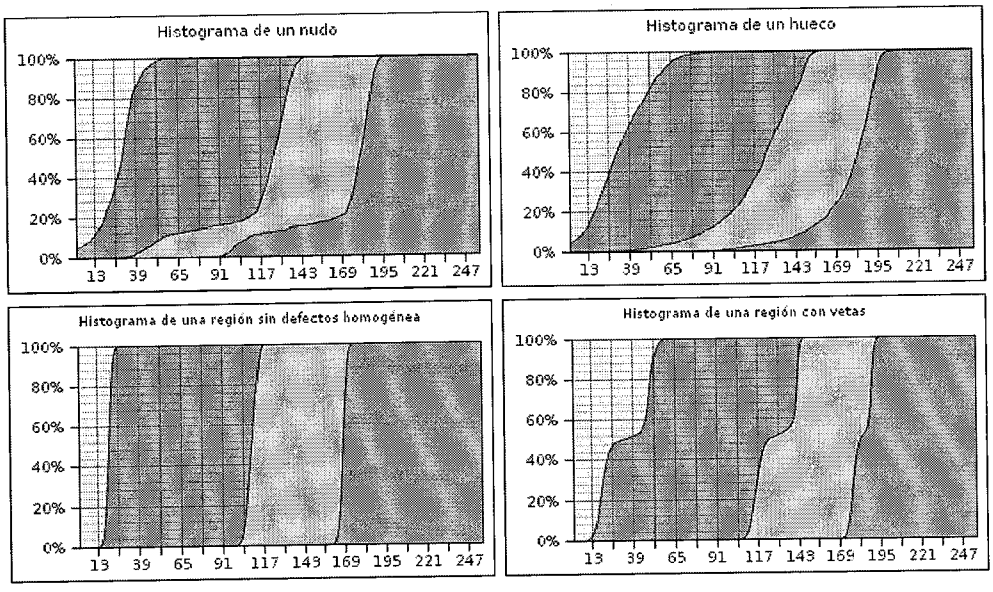


Ilustración 2.6: Histogramas acumulativos

Los histogramas acumulativos de la Ilustración 2.6 fueron calculados en base a los histogramas de la Ilustración 2.4. Observando los dos primeros histogramas (los cuales representan regiones con defectos) se puede verificar que estos tienen una gran variación hasta el 20-percentil, y a partir de ese valor los valores para los

percentiles restantes son muy similares. En el histograma correspondiente a una región sin defectos homogénea se observa que existe muy poca variación en todos los percentiles, esto se debe a que la variación de los niveles de color en la zona son muy pequeños. En el histograma correspondiente a una región sana pero con vetas se observa una variación aproximadamente en el 50-percentil.

A partir de estas observaciones se pueden determinar los valores de los percentiles a utilizar para la discriminación de las regiones. En [17] se mencionan algunos valores a utilizar, que son los siguientes: 0,2%, 1%, 10%, 50% y 95%. Estos valores son utilizados para los tres canales: R, G y B, obteniendo un vector de características con 15 valores. Se ve claramente como se utilizan mas valores en el rango 0% - 10% y se le da menor importancia a valores mayores.

Teniendo esto en cuenta, se podrían utilizar mas percentiles para determinar si contribuyen a mejorar el proceso de clasificación o si por el contrario no se comprueban mejoras y sólo lo hacen mas lento.

CAPÍTULO 3 - RESULTADOS

En este capítulo se presentarán los resultados obtenidos a partir de las diferentes pruebas realizadas con distintos parámetros del sistema.

Estas pruebas se dividen en dos etapas, la primera consta de un conjunto de pruebas para poner a punto el sistema, en las cuales se entrenan las dos redes (SOM y LVQ) con un conjunto de muestras diseñado para cada una de ellas. El conjunto de muestras de entrenamiento para el SOM consta de 10 placas de guatambú y 10 placas de pino. Cabe destacar que la clasificación de placas de guatambú y la clasificación de placas de pino son experimentos diferentes.

El conjunto de entrenamiento para la red LVQ consta de 16 muestras (en el caso de placas de pino) para cada una de las clases a discriminar: nudo, hueco y sin defecto. En el caso de las placas de guatambú, se obtuvieron 8 muestras para cada clase: nudo, mancha y sin defecto. Estas muestras se pueden observar en el Apéndice E.

La segunda etapa de pruebas consiste en una prueba integral del sistema, es decir, una vez que el sistema (las redes SOM y LVQ) está entrenado, se presentan muestras de placas y se cuentan la cantidad de aciertos en la clasificación que obtuvo el sistema, teniendo en cuenta la cantidad total de fallas y el tipo de estas. Se realizaron dos pruebas por separado, una para las placas de pino y otra para las placas de guatambú. Se presentaron al sistema 10 muestras de cada tipo de placa.

3.1. Fase de detección de fallas

Se comparan dos versiones del sistema de clasificación, la primera utiliza percentiles como en el sistema propuesto en [17]. La segunda versión utiliza menor cantidad de características, las cuales también se extraen de los histogramas de la imagen.

El clasificador utilizado en la etapa de detección deberá determinar si una región contiene fallas o no, por lo tanto solamente será necesario discriminar entre dos clases: sana o defectuosa. Teniendo esto en cuenta se puede utilizar el gráfico de la U-Matrix para determinar si un SOM es apropiado o no para realizar dicha tarea.

Para encontrar la configuración del SOM a través de la visualización de los resultados en la U-Matrix se utilizaron 20 muestras de diferentes placas completas.

10 muestras fueron de placas de guatambú y 10 muestras correspondientes a placas de pino.

3.1.1. Utilización de percentiles para la detección de fallas

Se realizaron distintas pruebas utilizando percentiles para la fase de detección de fallas, variando la cantidad de percentiles como así también su valor.

3.1.1.1. Utilización de 15 percentiles

En principio se utilizaron 15 valores diferentes, los cuales son los mismos que se utilizan en [17].

Para cada canal los valores de los percentiles son: 0.2%, 1%, 10%, 50% y 95%.

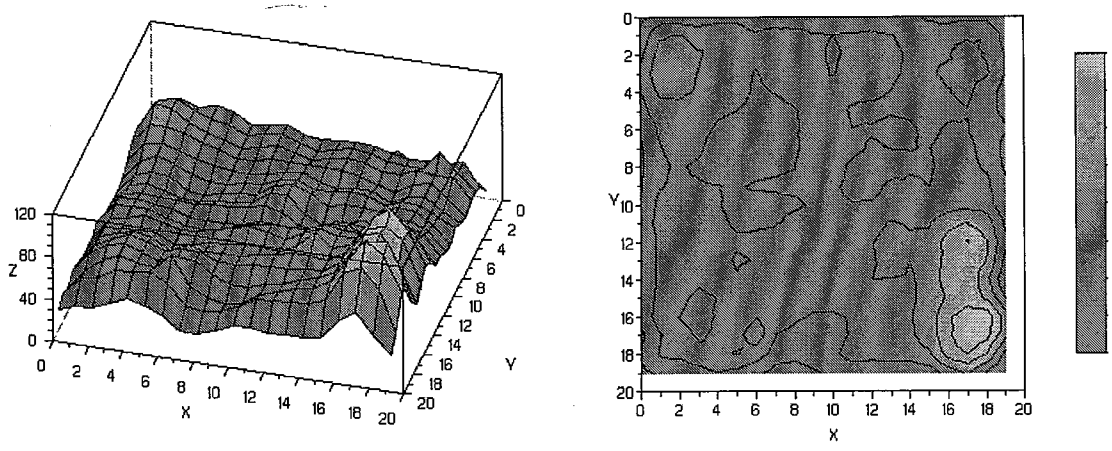


Ilustración 3.1: U-Matrix para SOM utilizando 15 percentiles

En la Ilustración 3.1 se representa la U-Matrix del SOM ya entrenado con los valores mencionados anteriormente. Se puede ver que no existe una frontera bien definida que divida las regiones sanas de las defectuosas. Existen sin embargo varias clases intermedias. Esto resulta en un mapa donde las fallas se agrupan según su clase en diferentes regiones del mismo. En este caso las regiones defectuosas están agrupadas en la zona inferior derecha del mapa, justo donde se registra la mayor altura. Esto significa que las regiones que se agrupan en ese lugar serán diferentes unas de otras. Verificando los resultados obtenidos por el sistema se comprobó que en la región mencionada se agrupan las fallas, pero también se filtran regiones sin defectos pero en las que su brillo general es mas bajo que las demás regiones sanas, esto se debe a que la iluminación no fue homogénea al momento de realizar la captura de las imágenes.

En las demás zonas del mapa se distribuyen las regiones sanas pero con diferentes características, como ser la cantidad de vetas que contiene la región o su brillo general en el caso de las regiones sin vetas.

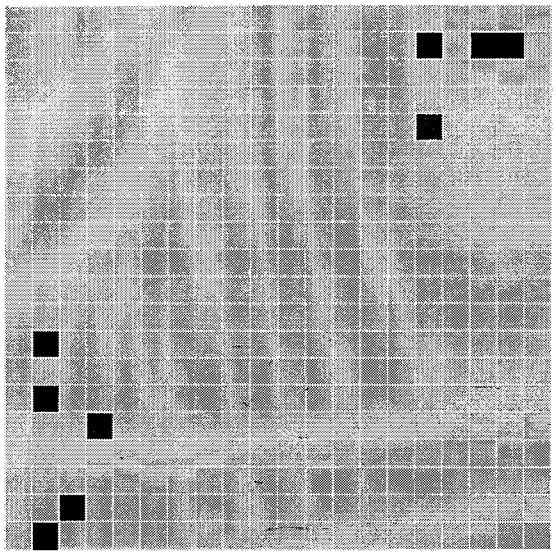


Ilustración 3.2: SOM con 15 percentiles

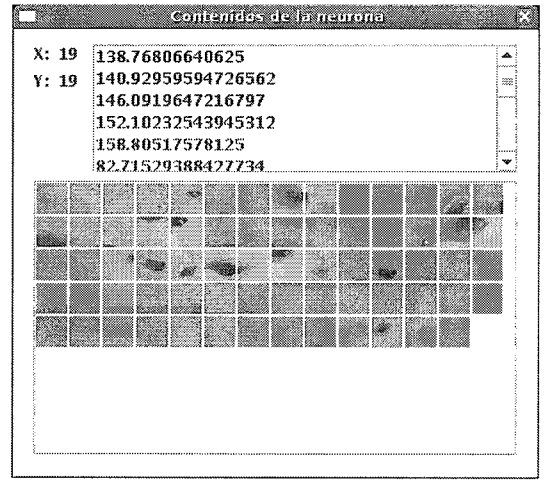


Ilustración 3.3: Contenidos de una neurona

La Ilustración 3.2 es una representación del SOM en la cual cada región de la retícula representa una neurona del mapa. Las regiones marcadas en naranja son las que el operador define como defectuosas. La imagen mostrada para la región es una de las regiones de la placa para la cual la neurona en esa posición es la BMU. Así una neurona podrá tener varias regiones de la placa asociadas a ella, en este caso se mostrará una sola de las imágenes. Las partes en negro que se pueden ver en la Ilustración 3.2 representan neuronas que no tienen regiones de la placa asociadas. También se puede ver como el mapa agrupa las partes parecidas en sus diferentes regiones, por ejemplo, en la esquina superior izquierda se encuentran las regiones sanas y homogéneas. En la parte central y mas cerca de los bordes inferior y derecho se encuentran las regiones con vetas, y en la esquina inferior derecha las regiones con defectos.

Para poder ver todas las regiones de la placa asociadas a una neurona del mapa, el operador debe hacer click con el botón derecho del mouse en la neurona a inspeccionar. De esta manera, aparecerá una ventana como la que se ve en la Ilustración 3.3, la cual representa la neurona situada en la esquina inferior derecha del mapa. En el caso del SOM con 15 percentiles, se puede ver que los contenidos de esta neurona contiene las regiones defectuosas de la placa, pero como se mencionó con anterioridad, también contiene regiones sanas que son mas oscuras.

3.1.1.2. Utilización de 30 percentiles

A los valores utilizados en las pruebas con 15 percentiles, se le agregaron 5 valores mas por canal, obteniendo así un total de 30 valores. Los valores para cada canal en estas pruebas fueron: 0.1%, 0.5%, 1%, 5%, 10%, 15%, 20%, 25%, 50% y 95%.

Los resultados obtenidos en las pruebas con 30 percentiles no muestran una

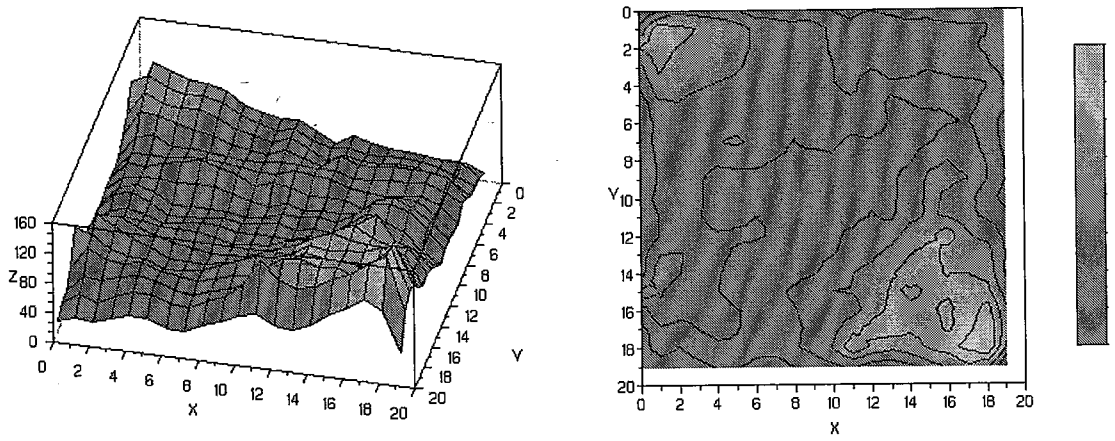


Ilustración 3.4: U-Matrix del SOM de 30 percentiles

mejora significativa con respecto a las pruebas con 15 percentiles.

Se puede apreciar que la Ilustración 3.4 que representa la U-Matrix del SOM de 30 percentiles es muy similar a la U-Matrix del SOM de 15 percentiles. No existen fronteras bien definidas entre las clases, salvo en la parte central del mapa, donde se puede ver una agrupación importante, la cual pertenece a regiones sin defectos. Nuevamente, las regiones defectuosas se presentan en la esquina inferior derecha, siendo estas diferentes entre si. Se encontraron en la agrupación regiones correspondientes a partes de nudos y huecos. También en este caso se filtraron en la clasificación regiones sanas pero de color mas oscuro (debido a la iluminación) que la mayoría de las demás regiones sanas.

En la Ilustración 3.5 se puede ver la distribución de las neuronas del mapa y los prototipos de las regiones asociadas a cada una. En color naranja están marcadas las neuronas que el operador considera como representantes de regiones defectuosas. La Ilustración 3.6 representa el contenido de la neurona de la esquina inferior izquierda. Como se puede apreciar, la neurona contiene regiones con fallas pero también contiene algunas pocas regiones sanas. Como se mencionó anteriormente, no existe una diferencia significativa utilizando mayor cantidad de

percentiles.

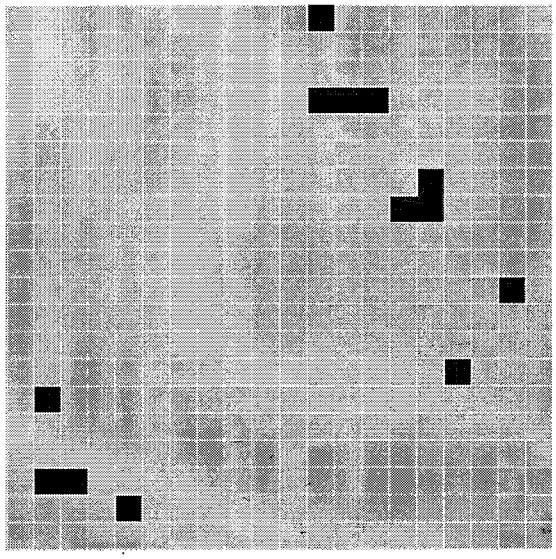


Ilustración 3.5: SOM con 30 percentiles

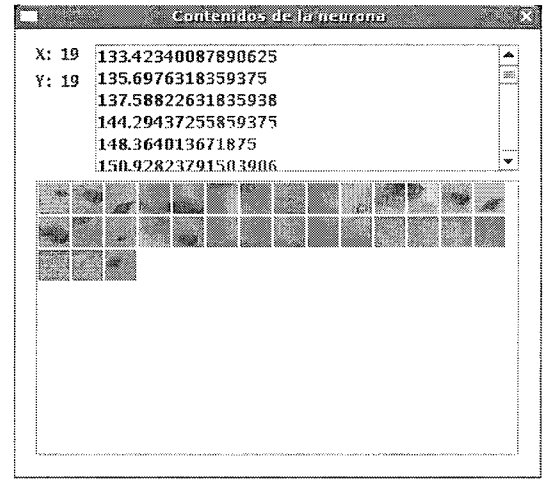


Ilustración 3.6: Contenidos de una neurona

3.1.2. Utilización de otras características

El método propuesto en nuestro trabajo reduce drásticamente la cantidad de características de entrada utilizadas en las redes, haciendo el sistema mas veloz. Para la red SOM de detección se utilizan únicamente 2 características: el valor del histograma del canal verde para el intervalo comprendido entre [0-63] y el valor del histograma del canal rojo para el intervalo comprendido entre [64-127].

La utilización de estos segmentos del histograma mejora los resultados de la clasificación como así también la velocidad del sistema, debido a que debe procesar una cantidad de datos mucho menor.

En la Ilustración 3.7 se puede ver claramente una frontera bien definida. Las partes sin defectos se ubican en la esquina superior izquierda, mientras que las regiones con defectos se encuentran dispersas por el resto del mapa, delimitadas entre las zonas encerradas entre curvas. Estas zonas se representan en el gráfico 3D como zonas con altura. Se puede ver también que aproximadamente la mitad del mapa (delimitada diagonalmente) contiene regiones sin altura, lo que significa que los datos que se encuentren en esa zona pertenecerán a la misma clase. En este caso esta zona sin altura contiene las regiones sin defectos. Se recuerda que el tamaño del mapa define la cantidad de clases diferentes que se quieren obtener. Si se tiene esto en cuenta, se puede reducir el tamaño del mapa, debido a que nuestro propósito no es diferenciar entre varias regiones sin defectos, sino dividir las zonas

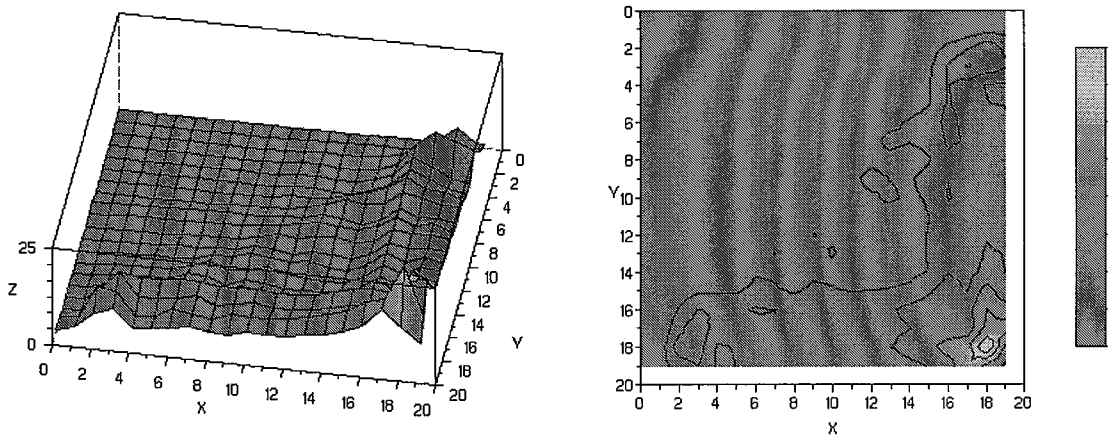


Ilustración 3.7: U-Matrix del SOM con 2 segmentos de histograma

sin defectos de las que sí los tienen, por lo tanto se tienen únicamente dos clases. Estas clases están divididas claramente en el mapa.

En las ilustraciones 3.8 y 3.9 se puede apreciar la distribución de las neuronas del mapa como así también el contenido de una de ellas. Se puede ver como la separación entre las zonas sanas y las zonas con defectos está claramente marcada por una diagonal, la cual se corresponde con la que se vio en la representación de la U-Matrix. En la Ilustración 3.9 se puede ver el contenido de la neurona ubicada en la esquina inferior derecha. A diferencia de las pruebas realizadas utilizando percentiles, en este caso se puede ver que la neurona contiene únicamente regiones con defectos. Utilizando los segmentos del histograma se obtiene una división bien definida entre las dos clases. También se puede ver en la Ilustración 3.8 que la mayoría de las neuronas del mapa no contienen zonas asociadas, lo que también da un indicio de que el tamaño del mapa es innecesariamente grande para nuestro propósito.

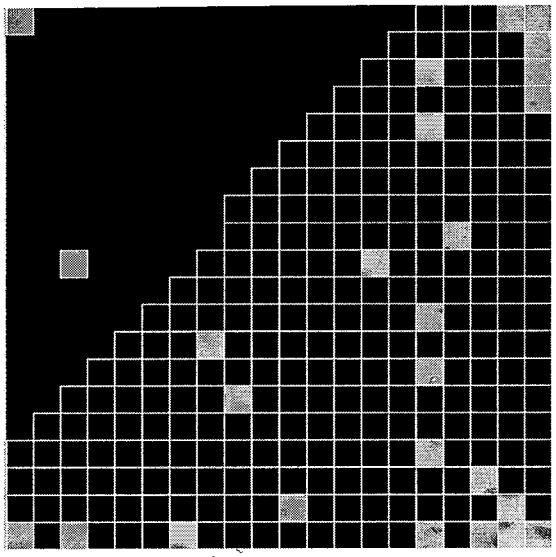


Ilustración 3.8: SOM de 2 segmentos

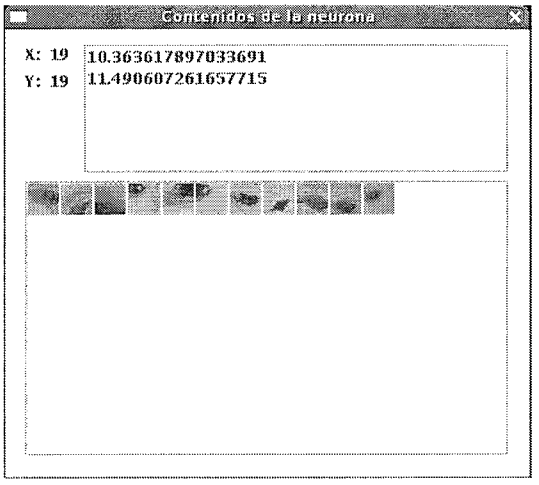


Ilustración 3.9: Contenidos de una neurona

En todas las pruebas realizadas, el tamaño del mapa era de $20x20=400$ neuronas. Como se ve en las ilustraciones anteriores una reducción en el tamaño del mapa es posible debido a la cantidad de neuronas que no tienen regiones asociadas y a que la cantidad de clases a obtener son únicamente dos. Además con la utilización de segmentos del histograma la división entre las dos clases está mucho mejor definida que con la utilización de percentiles.

El problema es definir el tamaño óptimo para el mapa a utilizar. Éste debe ser lo suficientemente pequeño como para que no existan muchas neuronas sin regiones asociadas, pero lo suficientemente grande como para mantener una frontera bien definida entre las dos clases. No existe ninguna fórmula para determinar el tamaño del mapa, éste debe definirse dependiendo de la cantidad de clases que se quieren obtener y teniendo en cuenta el tipo de problema que se quiere resolver. También hay que tener en cuenta las características utilizadas para la clasificación. Como se vio en los ejemplos de los percentiles, los mapas utilizados con los percentiles tenían regiones asociadas para casi todas las neuronas del mapa. Para el mismo problema, y únicamente variando las características utilizadas y no el tamaño del mapa, cambia la distribución del mapa drásticamente.

Una forma de determinar el tamaño del mapa es probando las diferentes configuraciones y comparando los resultados obtenidos hasta obtener la configuración óptima.

Reduciendo el tamaño del mapa a 5 x 5 se mantiene el nivel de discriminación mientras que se reduce aún mas el tiempo necesario para procesar los datos. La cantidad de clases a obtener es de $5x5=25$. Como se ve en la Ilustración 3.10, la

frontera entre las dos clases sigue siendo bien definida como en el caso del SOM de 20 x 20. Las clases también están distribuidas de la misma manera, las regiones sin defectos en la parte superior izquierda y las con defectos en la inferior derecha. Las clases están divididas aproximadamente por la diagonal que va desde la esquina inferior izquierda hasta la esquina superior derecha.

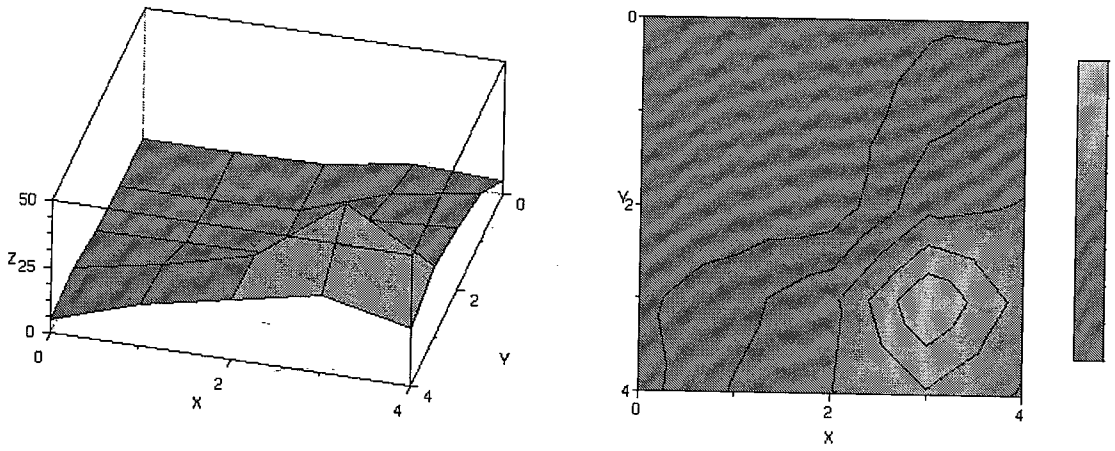


Ilustración 3.10: U-Matrix del SOM de 5 x 5

En la Ilustración 3.11 se puede ver como la distribución del mapa se corresponde con el gráfico de la U-Matrix. La zona en negro no contiene regiones asociadas a las neuronas. La neurona de la esquina superior izquierda contiene todas las regiones sin defectos. Los defectos están agrupados cerca de la neurona ubicada en la parte inferior derecha.

En la Ilustración 3.12 se ve el contenido de la neurona ubicada en la parte inferior derecha del mapa. Se siguen agrupando únicamente zonas con defectos en esta parte del mapa.

Con una menor cantidad de características y un mapa de mucho menor tamaño se logró mejorar la clasificación de zonas defectuosas y se solucionó el problema de la distribución de brillo en la imagen. Las zonas sanas pero que contienen problemas de iluminación (disminución del brillo con respecto a otras zonas sanas) no son clasificadas como defectuosas utilizando este tipo de descriptores.

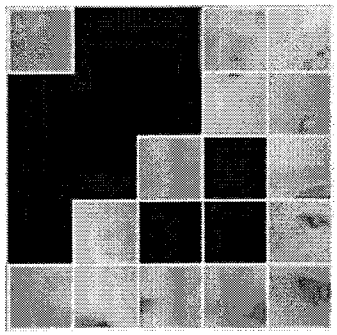


Ilustración 3.11: SOM 5x5

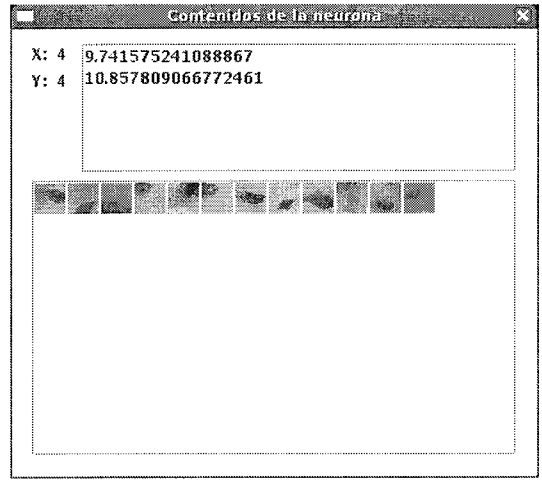


Ilustración 3.12: Contenidos de una neurona

3.2. Fase de reconocimiento de fallas

La fase de reconocimiento de fallas consiste en clasificar las regiones de la imagen original que fueron detectadas como defectuosas por el SOM en las diferentes clases definidas por el operador. Estas imágenes son obtenidas mediante un análisis de conectividad de regiones, el cual comprueba si las regiones marcadas como defectuosas son contiguas o no. Se supone que todas las regiones contiguas marcadas como defectuosas pertenecen a una misma falla en la placa, por lo que se genera una imagen por cada conjunto de regiones contiguas. Cada una de estas imágenes representa una falla y por lo tanto una entrada para la fase de reconocimiento de fallas.

Antes de extraer características de entrada para la red LVQ, las imágenes de entrada son filtradas mediante algoritmos de realce y procesamiento de imágenes d\`itales. En el caso del sistema desarrollado, se trata de eliminar las zonas de la imagen que no pertenecen a la falla en si, mediante un algoritmo de detección de bordes. Este algoritmo funciona transformando la imagen de entrada mediante un proceso de binarización[9] [1], el cual consiste en convertir los píxeles que posean un valor mayor a un umbral determinado en dos valores posibles: blanco o negro. Si el píxel en cuestión supera el umbral se lo convierte al blanco, en caso contrario en negro. Una vez convertida la imagen, se buscan los cuatro puntos extremos: superior, inferior, derecho e izquierdo. Suponiendo que el fondo de la madera es de un color relativamente claro comparado con los bordes del defecto, los puntos extremos se aproximarán a la parte externa de los bordes del defecto. Una vez obtenidas las coordenadas de los puntos extremos, se recorta la imagen original (en color) utilizando estas coordenadas. El algoritmo es bastante sencillo y de bajo costo

9 Que puede tener únicamente dos valores posibles: 0 o 1 (Negro o Blanco)

computacional. Los resultados obtenidos mediante este algoritmo son similares a los obtenidos mediante técnicas mas complejas de detección de bordes, como por ejemplo la detección de bordes mediante *operadores de Sobel* [2], en la cual intervienen varias operaciones de convolución, haciéndola por lo tanto mucho mas lenta.

Una vez obtenida la imagen que contiene únicamente la falla, se procede a la extracción de características para su utilización con la red de clasificación LVQ.

3.2.1. Percentiles en el reconocimiento de fallas

A diferencia de la fase de detección de fallas, en la cual existen únicamente dos clases posibles (con o sin defectos); en la fase de reconocimiento existen varias clases, las cuales son definidas por el operador del sistema. El problema consiste en separar los elementos correspondientes a cada clase, se deben definir las características a utilizar en la red LVQ dependiendo de que elementos se quieran separar. En la sección 2.1.1 se definieron los diferentes tipos de fallas que el sistema deberá clasificar.

A continuación se presentan las diferentes características utilizadas por la red LVQ para realizar la clasificación de las fallas. Cabe destacar que puede darse el caso en el que una región sin defectos sea clasificada como defectuosa por el SOM. Para solucionar este inconveniente la red LVQ debe contener una clase que represente estas regiones sin defectos para que puedan ser clasificadas como sanas al utilizar otro tipo de descriptores que no se usaron en el SOM.

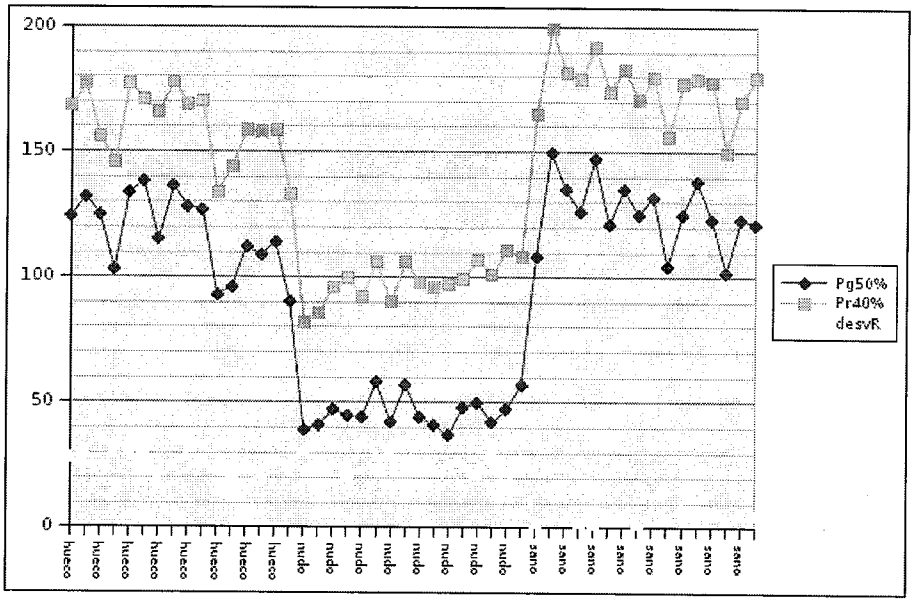


Ilustración 3.13: Posibles características a utilizar con la red LVQ

Los percentiles pueden utilizarse para el reconocimiento de las fallas. Estos descriptores resultaron ser muy útiles para dividir nudos de huecos, como se verá a continuación.

En la Ilustración 3.13 se representan diferentes características extraídas del conjunto de entrenamiento para la red LVQ. La línea azul representa el 50-percentil para el histograma del canal verde, la magenta el 40-percentil para el histograma del canal rojo y la amarilla representa la desviación estándar para el histograma del canal rojo. Como se puede ver en el gráfico, la información provista por los percentiles es casi igual, es decir por mas que los valores para cada percentil sean diferentes, las líneas que los representan tienen una forma muy similar. En el eje de las ordenadas se puede ver la clase que está representando un punto. A la izquierda se sitúan los huecos, en el centro los nudos y a la derecha las regiones sanas. Los percentiles resultan ser útiles para diferenciar entre nudos y huecos o entre nudos y partes sanas, pero no son útiles para distinguir entre huecos y regiones sanas.

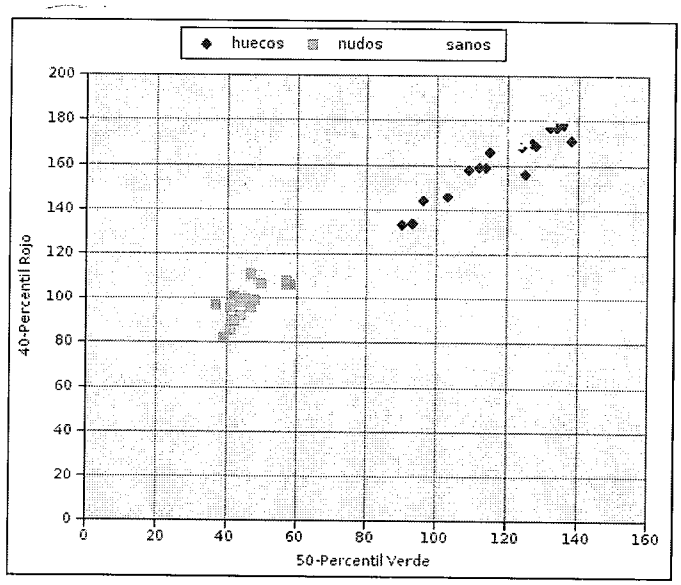


Ilustración 3.14: 40-Percentil Rojo y 50-Percentil Verde

La Ilustración 3.14, muestra la separación entre los huecos y nudos, como así también se ve la superposición entre los huecos y las partes sanas.

Es necesario un descriptor que permita separar entre huecos y regiones sanas. A diferencia de los percentiles, la desviación estándar no es útil para distinguir entre huecos y nudos, pero sí para distinguir entre nudos o huecos y regiones sanas. Combinando el 50-Percentil verde y la desviación estándar para el canal rojo se obtienen los resultados que se ven en la Ilustración 3.15.

Como se puede ver, las zonas en las que se distribuye cada clase se pueden separar fácilmente por el clasificador LVQ.

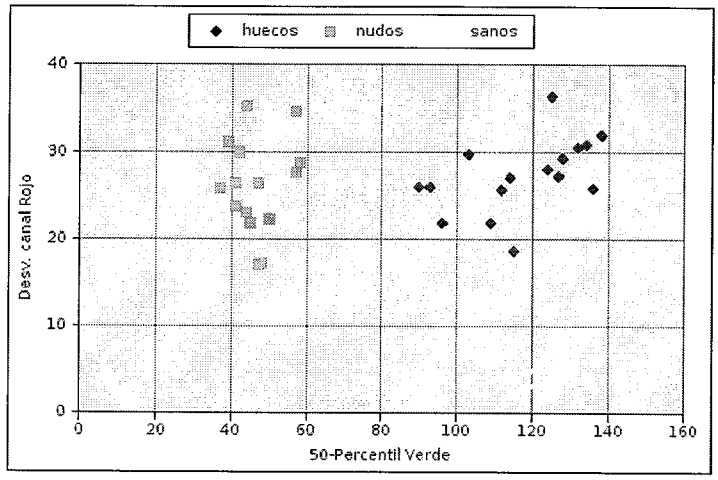


Ilustración 3.15: 50-Perc. Verde y Desviación para canal rojo

3.3. Pruebas integrales del sistema

Para las pruebas integrales del sistema, se etiquetaron previamente las fallas en un conjunto de imágenes de placas con la ayuda de un experto. En el caso de las placas de pino, estas clases se dividieron en nudos y huecos; mientras que en las placas de guatambú se dividieron en nudos y manchas. En ambos casos se agregó una clase mas denominada "sin defectos", esto es necesario ya que pueden existir zonas de las placas que sean detectadas como posiblemente defectuosas por el SOM y pasan luego a ser analizadas por la red LVQ. Esta última define la clasificación final y en el caso de zonas que no posean defectos, debería clasificarlas como "sin defectos".

Una vez que etiquetadas las fallas en las muestras de las placas, se confeccionó una tabla, la que contiene la información de las observaciones.

Posteriormente se analizó el conjunto de muestras con el prototipo del sistema de clasificación, de a una por vez. El sistema asigna a cada tipo de falla un color, y marca las fallas detectadas sobre la imagen de la placa con el color correspondiente.

Una vez obtenidos los resultados del sistema, se cuentan las fallas detectadas, y con la ayuda de un experto, se anotan los resultados dividiéndolos en tres clases: clasificaciones correctas, clasificaciones incorrectas y clasificaciones dudosas. Si el sistema clasifica correctamente una falla previamente etiquetada, esta clasificación es contada como correcta, y en caso contrario es contada como incorrecta. Existen casos en los cuales la falla es un caso híbrido, como puede darse con nudos que están desprendidos a medias y que visualmente parecen ser "mitad nudo" y "mitad hueco". Otro ejemplo se da cuando el sistema detecta dos fallas en la madera como una sola, por lo tanto trata de clasificar una imagen que contiene dos

fallas diferentes. Estos casos híbridos son contabilizados como clasificaciones dudosas.

Este procedimiento se aplica para las muestras de placas de guatambú y para las muestras de placas de pino. Cada grupo contiene 15 muestras de placas. Los resultados obtenidos fueron los siguientes:

	NUDOS ENCONTRADOS	HUECOS ENCONTRADOS	SANOS ENCONTRADOS	TOTAL ENCONTRADO	Correctas	Incorrectas	Dudosas
PLACA 1	6	2	5	13	11	0	2
PLACA 2	12	2	9	23	21	1	1
PLACA 3	4	2	2	8	4	3	1
PLACA 4	5	3	7	15	13	1	1
PLACA 5	7	2	6	15	11	2	2
PLACA 6	3	3	4	10	6	0	4
PLACA 7	0	2	5	7	5	0	2
PLACA 8	17	1	4	22	22	0	0
PLACA 9	10	1	1	12	12	0	0
PLACA 10	6	0	2	8	8	0	0
PLACA 11	2	0	5	7	5	1	1
PLACA 12	4	1	3	8	5	0	3
PLACA 13	2	0	3	5	5	0	0
PLACA 14	2	1	4	7	3	2	2
PLACA 15	1	2	1	4	4	0	0
TOTALES	**81**	**22**	**61**	**164**	**135**	**10**	**19**

Tabla 1: Resultados de clasificación para placas de pino

Ilustración 3.16: Resultados de clasificación para placas de pino

	NUDOS ENCONTRADOS	MANCHAS ENCONTRADOS	SANOS ENCONTRADOS	TOTAL ENCONTRADO	Correctas	Incorrectas	Dudosas
PLACA 1	1	5	5	11	11	0	0
PLACA 2	3	0	1	4	4	0	0
PLACA 3	2	0	1	3	3	0	0
PLACA 4	2	1	0	3	2	1	0
PLACA 5	3	4	1	8	4	4	0
PLACA 6	1	6	6	13	6	6	1
PLACA 7	2	2	2	6	4	1	1
PLACA 8	2	1	3	6	3	1	2
PLACA 9	3	0	1	4	3	1	0
PLACA 10	3	0	0	3	2	1	0
PLACA 11	5	0	0	5	5	0	0
PLACA 12	2	0	1	3	2	1	0
PLACA 13	1	1	1	3	2	1	0
PLACA 14	3	1	0	4	4	0	0
PLACA 15	2	0	3	5	5	0	0
TOTALES	**35**	**21**	**25**	**81**	**60**	**17**	**4**

Tabla 2: Resultados de clasificación para placas de guatambú

Ilustración 3.17: Resultados de clasificación para placas de guatambú

CAPÍTULO 4 - CONCLUSIONES

Teniendo en cuenta los resultados obtenidos mediante los métodos propuestos a lo largo de este trabajo, se puede observar que la realización de un sistema de escala industrial para la clasificación de paneles de maderas según el aspecto de sus caras es posible, siempre y cuando se tengan ciertas consideraciones en el desarrollo del mismo. La utilización de percentiles y otros valores calculados en base al histograma de las imágenes brindan un método rápido de calcular los vectores de entradas a las redes, pero en algunas circunstancias se requieren otro tipo de descriptores mas complejos. El problema principal es separar las partes de la madera que son sanas de las que no lo son, como los defectos y las partes sanas varían con el tipo de madera a analizar, es muy difícil que un mismo método sea aplicable a todos los casos. Se podrían aplicar filtros a las imágenes de entrada para separar el fondo o partes con vetas de la madera de las partes con fallas, esto mejoraría enormemente la detección y el posterior reconocimiento de cada tipo de falla. El problema con estos tipos de filtros es que requieren un alto poder de cómputo para procesar imágenes tan grandes como las utilizadas en el trabajo.

Otro problema importante es la iluminación de las placas en el momento de adquisición de las muestras. En un sistema de producción, se deberían tomar todas las precauciones para que la iluminación de las placas sea completamente homogénea, ya que al haber alteraciones en la iluminación, los valores obtenidos a partir del histograma de la imagen suelen variar demasiado en regiones parecidas pero en las que la iluminación varía.

Los equipos utilizados para la adquisición de las imágenes también influyen mucho al momento de procesar las mismas. En este trabajo se utilizó una cámara digital de uso doméstico para capturar las imágenes. Se debería contar con equipos mas sofisticados y especialmente diseñados para la captura industrial de imágenes, o realizar un estudio de otro tipo de sensores que puedan utilizarse para adquirir datos de las placas.

En la clasificación manual realizada por operadores humanos intervienen factores como la fatiga o las diferencias de criterio entre un operador y otro, en el caso del sistema automático se solucionan estos problemas ya que define criterios homogéneos en la clasificación de cada placa.

Actualmente no existe una norma o estándar en la Argentina que defina cómo debe hacerse la clasificación, aunque hay un borrador de norma IRAM [18] que

todavía no ha sido aprobada. Esto presenta un problema a la hora de comparar los resultados obtenidos por el sistema con los resultados obtenidos mediante una clasificación manual, ya que los operadores se basan en su experiencia personal a la hora de resolver ciertos conflictos en la clasificación. Un ejemplo de esto es que en el caso de las maderas de pino hay nudos que son parecidos a huecos, si esta falla es de tamaño muy grande los operadores clasifican la placa como de calidad inferior, pero no importa si la falla es en realidad un nudo o un hueco y ni siquiera importa su tamaño exacto, sino que basándose en su experiencia los operadores definen que esa placa no reúne las condiciones necesarias para ser de una calidad superior. Esta falta de criterios homogéneos hace que no se puedan comparar directamente los resultados obtenidos en ambos casos. En los casos en que las muestras de pino fueron previamente etiquetadas el sistema siempre detectó fallas en donde las hubo, y en la fase de reconocimiento detectó correctamente todos los casos en los que las muestras contenían iluminación homogénea y eran bien distinguibles. Para las muestras de guatambú el sistema clasifica de forma errónea partes con manchas como partes sanas, esto es debido a que los algoritmos utilizados para encontrar los límites de las partes con manchas no son lo suficientemente adecuados para aislar las partes con manchas del fondo de la madera. Al aplicar otro tipo de algoritmo de procesamiento de imágenes mas avanzados se podrían resolver este tipo de problemas.

4.1. Mejoras sugeridas

Como se mencionó anteriormente, el problema principal del sistema es aislar las partes de la madera con fallas de las vetas o el fondo de la misma. En [19] se realiza un estudio de la utilización de redes neuronales artificiales para el reconocimiento de fallas en la madera. En ese trabajo se utilizan *filtros de Gabor* para el filtrado de las imágenes utilizadas. Podría implementarse en el sistema propuesto este tipo de filtros u otros similares para mejorar los resultados obtenidos. Otra posible mejora consiste en utilizar algoritmos mas sofisticados en la detección de bordes de las fallas, esto es con el fin de aislar la falla en sí del fondo de la madera, especialmente cuando los límites de la falla son difíciles de detectar.

APÉNDICE A - FUNDAMENTOS DE COLOR

El color es un fenómeno físico que cuenta con una gran cantidad de combinaciones de luz, relacionado con las diferentes longitudes de onda del espectro electromagnético, que perciben las personas y algunos animales a través de los órganos de la visión. Todo cuerpo iluminado absorbe todas, o parte, de las ondas electromagnéticas y refleja las restantes. Estas ondas reflejadas, son analizadas por el ojo e interpretadas como colores según las longitudes de onda correspondientes. El color blanco resulta de la combinación de todos los colores, mientras que el negro es la ausencia de la luz. Es bien conocido el fenómeno mediante el cual un rayo de luz blanca que atraviesa un prisma se descompone en los siete colores básicos del asociados al espectro visible, este fenómeno se puede apreciar en la naturaleza en la forma del arco iris, el cual aparece en el cielo cuando los rayos de luz atraviesan gotas de agua que están suspendidas en la atmósfera, las cuales actúan como un prisma.

La utilización del color en el procesamiento de imágenes digitales está motivado por dos factores fundamentales. En primer lugar, el color es un potente descriptor que simplifica la identificación y extracción de objetos de una escena. En segundo lugar, el ojo humano puede distinguir una amplia gama de colores comparado con los niveles de gris.

Desde el punto de vista del ojo humano, todos los colores son vistos como combinaciones variables de los tres colores primarios de la luz: rojo (R), Verde (G) y Azul (B), dando así el nombre al modelo de color RGB (por sus siglas en inglés R = red, G = green, B = blue). Estos tres colores primarios pueden mezclarse para producir los colores secundarios de la luz: magenta (rojo y azul), cyan (verde y azul) y amarillo (rojo y verde).

Las cantidades de rojo, verde y azul necesarias para formar un color en particular se denominan colores tri-estímulos, denominados X, Y, Z respectivamente. Un color puede especificarse por sus coeficientes tri-cromáticos, definidos como:

$$x=\frac{X}{X+Y+Z} \quad y=\frac{Y}{X+Y+Z} \quad z=\frac{Z}{X+Y+Z}$$

$$donde\ x+y+z=1$$

A.1. Modelos de color

El propósito de un modelo de color es facilitar la especificación de los colores de alguna forma estándar. En esencia, un modelo de color es una especificación de un sistema de coordenadas 3D y un sub-espacio dentro de dicho sistema donde cada color es representado por un punto en el espacio con coordenadas x, y, z.

Existen varios modelos diferentes de color, esto es porque cada uno está orientado hacia el hardware que utiliza colores (como monitores a color o impresoras) o bien hacia aplicaciones en las cuales la manipulación del color es un objetivo (paquetes de edición de imágenes o video). Los modelos mas comunes utilizados hoy en día orientados al hardware son el RGB (Rojo, Verde y Azul) para monitores a color y cámaras de video; el CMY (Cyan, Magenta y Amarillo) para impresoras a color; y el YIQ para TV a color. A pesar de que existen otros varios modelos, se profundizará únicamente en el modelo RGB por ser el de mas importancia para el propósito del trabajo.

A.1.1. El modelo RGB

Este modelo obtiene su nombre por las siglas en inglés R = red, G = green y B = blue, las cuales hacen referencia a los tres colores primarios utilizados en el modelo y se lo conoce como un modelo de color aditivo porque cuando la luz de dos diferentes frecuencias viajan juntas, desde el punto de vista del observador, estos colores son sumados para crear nuevos tipos de colores. Los colores rojo, verde y azul fueron escogidos porque cada uno corresponde aproximadamente con uno de los tres tipos de conos sensitivos al color en el ojo humano (65% sensibles al rojo, 33% sensibles al verde y 2% sensibles al azul). El modelo está basado en el sistema de coordenadas cartesianas y el sub-espacio de color de interés es el tetraedro mostrado en la Ilustración A.1. Los valores RGB están situados en tres vértices; el cyan, el magenta y el amarillo se sitúan en otros tres vértices, el negro corresponde al origen de coordenadas y el blanco al punto más alejado del origen. La escala de grises se extiende por la línea recta que une a los puntos que representan al negro y al blanco, y los colores son los puntos dentro del tetraedro representados por vectores desde el origen. Se asume que todos los vectores han sido normalizados, de modo que sus valores se encuentran en el rango [0, 1], esto se realiza para generalizar la aplicación del modelo a cualquier tipo de dispositivo. Para obtener los valores reales del dispositivo utilizado, se deben escalar los valores a la cantidad de bits que puede representar el dispositivo mediante la siguiente fórmula: $x \cdot (2^n - 1)$.

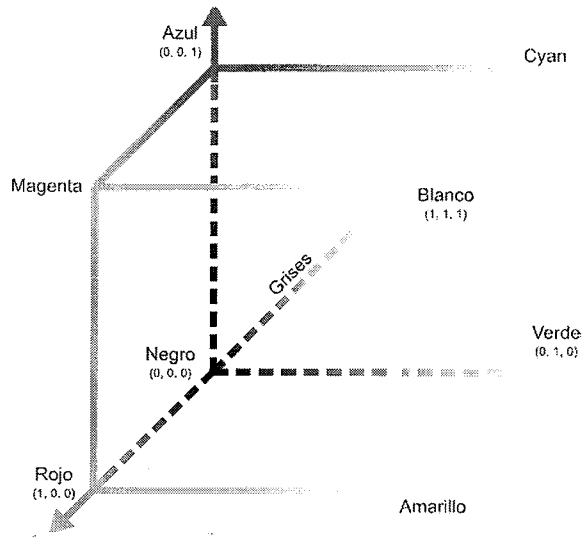


Ilustración A.1: Tetraedro de color RGB

A.1.2. El modelo CMY

El modelo CMY es generalmente utilizado en la impresión de imágenes a color y no es de gran utilidad en el tratamiento de imágenes digitales. El hardware de impresión suele contar con una entrada CMY o realiza una conversión interna de RGB a CMY. Esta conversión se realiza de la siguiente manera:

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

La conversión de CMY a RGB puede lograrse de forma inversa.

APÉNDICE B - DIAGRAMAS DEL SISTEMA

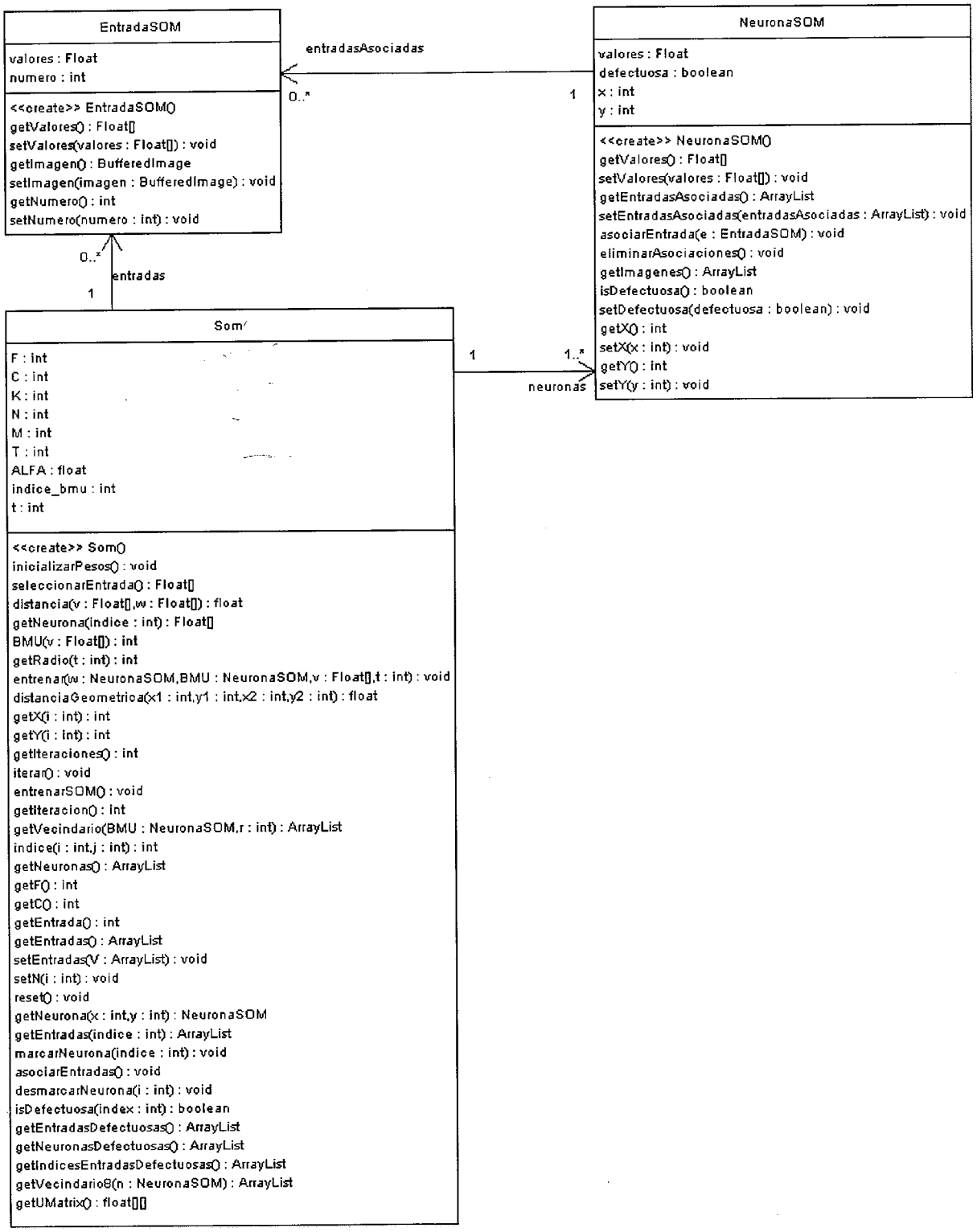


Ilustración B.1: Diagrama de clases para la implementación del SOM

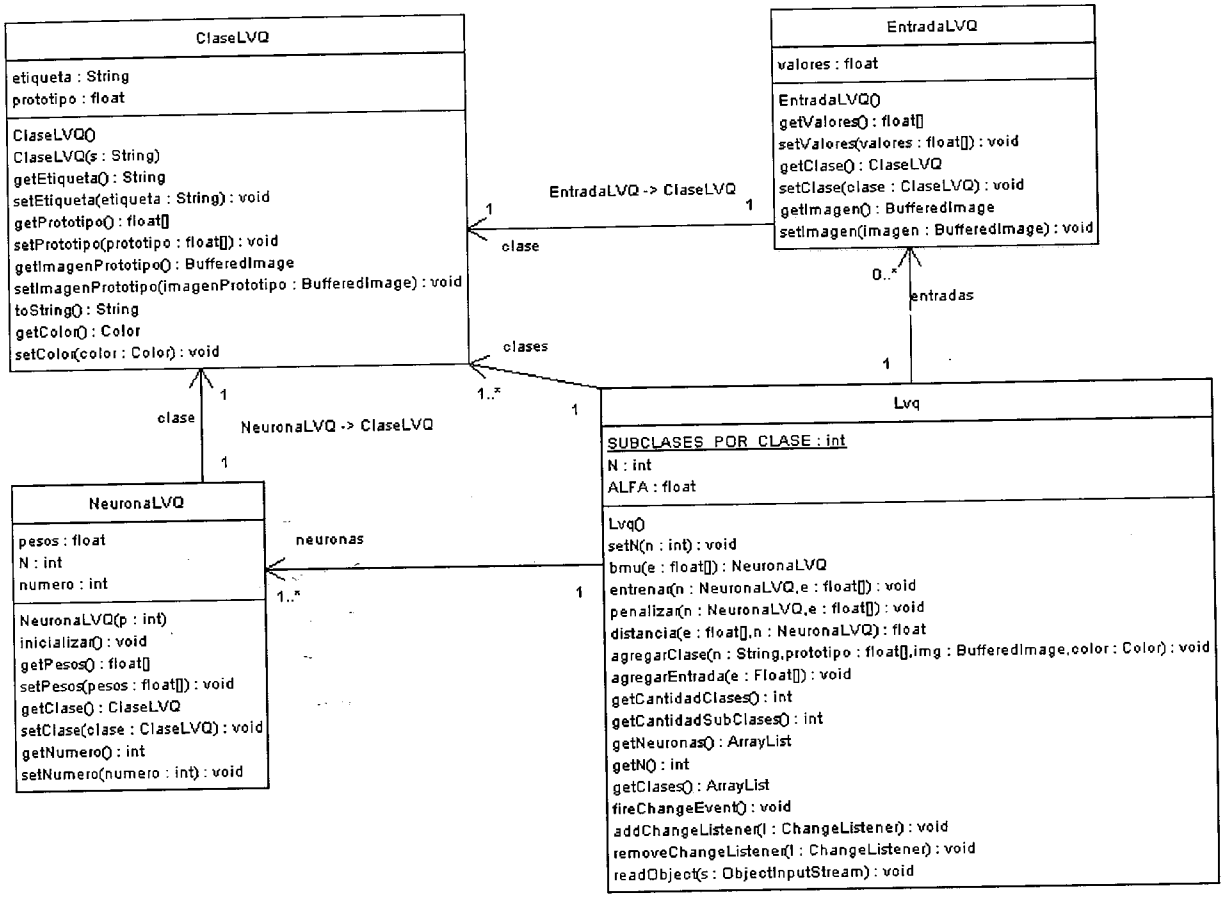


Ilustración B.2: Diagrama de clases para la implementación de la red LVQ

APÉNDICE C - CÓDIGO FUENTE

En este apéndice se presenta el código fuente de las clases que figuran en los diagramas del Apéndice B. No se incluye el código fuente de las interfaces gráficas ni de otras clases de utilidad debido a la longitud de este código adicional.

```
/*
 * Som.java
 */

package nn;

import java.io.Serializable;
import java.util.ArrayList;


/**
 * Representa una red tipo SOM (self organizing map)
 */
public class Som implements Serializable {

    private final int F = 5; // numero de filas de la capa de competicion
    private final int C = 5; // numero de columnas de la capa de competicion
    private int K = 4; // cantidad de entradas de la red
    private int N = 2; // numero de elementos de las neuronas y entradas
    private final int M = F * C; // cantidad de neuronas en la capa de competicion
    public final int T = 1000; // cantidad total de iteraciones a realizar
    public final float ALFA = 0.1f; // tasa de aprendizaje inicial

    // entradas de la red
    private transient ArrayList<EntradaSOM> entradas = new ArrayList<EntradaSOM>(0);

    // capa de competicion
    private ArrayList<NeuronaSOM> neuronas = new ArrayList<NeuronaSOM>(0);

    public int entrada = 0; // indice de la entrada seleccionada para el entrenamiento
    public int indice_bmu = 0; // indice de la BMU para la entrada seleccionada

    private int t = 0; // iteracion actual

    /**
     * Crea e inicializa una nueva red SOM
     */
    public Som(){
        inicializarPesos();
```

```
}

private void inicializarPesos(){
    Float[] muestra;
    NeuronaSOM neu;
    float max = F + C - 2;

    neuronas.clear();

    /** se inicializan los pesos de las neuronas y se les asigna su posicion en la SOM
        bidimensional */
    for(int f = 0; f < F; f++){
        for(int c = 0; c < C; c++){
            muestra = new Float[N];

            for(int n=0; n < N; n++)
                muestra[n] = (float)(f + c) / max;

            neu = new NeuronaSOM();
            neu.setValores(muestra);
            neu.setX(c);
            neu.setY(f);

            neuronas.add(neu);
        }
    }
}

/** Selecciona aleatoriamente un vector de entrada de la matriz de vectores de entrada
 *  El vector seleccionado corresponde a una fila de la matriz V */
private Float[] seleccionarEntrada(){
    entrada = ((int)(K * Math.random()));
    return entradas.get(entrada).getValores();
}

/** calcula la distancia euclidea entre un vector v y una neurona w */
private static float distancia(Float[] v, Float[] w){
    float d = 0;

    /** se calcula la sumatoria para todos los elementos de v y w (cantidad = N) elevados
        al cuadrado */
    for(int i=0; i < v.length; i++)
        d+= (v[i] - w[i]) * (v[i] - w[i]);

    /** se devuelve la raiz cuadrada de la sumatoria, esto podría obviarse para un mayor
        rendimiento */
    return (float)Math.sqrt(d);
}
```

```
/** devuelve una columna la matriz neuronas
 * @param indice índice de la columna a obtener */
private Float[] getNeurona(int indice){
    return neuronas.get(indice).getValores();
}

/**
 * devuelve la BMU
 * @param v vector con los valores de entrada
 * @return indice de la BMU
 */
public int BMU(Float[] v){
    int BMU = 0; // neuronas[j] con menor distancia al vector V
    float dist_min; // distancia minima actual
    float dist_actual; // distancia actua;

    dist_min = distancia(v, getNeurona(0));

    for(int i=1; i < M; i++){
        dist_actual = distancia(v, getNeurona(i));

        if(dist_actual < dist_min){
            dist_min = dist_actual;
            BMU = i;
        }
    }

    return BMU;
}

/** calcula el radio del vecindario de la BMU
 * @param t número de iteración */
 private int getRadio(int t){
     int r = 0; // radio del vecindario
     float lambda = 0; // constante de tiempo lambda

     lambda = ((float) T) / (float)Math.log(getC());

     r = (int) ((float) getC() * Math.exp((-((float) t / lambda))));

     return r;
 }

 /** calcula los nuevos pesos para neuronas
  * @param w columna de neuronas a entrenar
  * @param t número de iteración */
 private void entrenar(NeuronaSOM w, NeuronaSOM BMU, Float[] v,  int t){
```

```
        // la tasa de aprendizaje disminuye con cada iteracion
        float alfa = (float) (ALFA / (1f + t));

        float radioVecindario = getRadio(t);

        Float[] pesosW = w.getValores();

        int xw = w.getX(); // posicion x del elemento w;
        int yw = w.getY(); // posicion y del elemento w;

        int xb = BMU.getX(); // posicion x de la BMU;
        int yb = BMU.getY(); // posicion y de la BMU;

        float dist = distanciaGeometrica(xw, yw, xb, yb);
        dist = dist * dist;

        float tita = (float) Math.exp(-(dist) / (2d * (radioVecindario * radioVecindario)));

        // se actualizan los pesos
        for(int i=0; i < N; i++){
            pesosW[i] = pesosW[i] +  tita * ALFA * (v[i] - pesosW[i]);
        }

    }

    /** calcula la distancia geométrica entre dos puntos definidos por coordenadas X, Y
     * @param x1 coordenada x del punto 1
     * @param y1 coordenada y del punto 1
     * @param x2 coordenada x del punto 2
     * @param y2 coordenada y del punto 2 */
    private float distanciaGeometrica(int x1, int y1, int x2, int y2){
        float d = (float) Math.sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));

        return d;
    }

    /** calcula el indice x de una neurona en la matriz neuronas
     * @param i número de elemento correspondiente a la neurona */
    private int getX(int i){
        return i % C;
    }

    /** calcula el indice y de una neurona en la matrix neuronas
     * @param i número de elemento correspondiente a la neurona */
    private int getY(int i){
        return (i - (i % C)) / C;
    }
```

```
/** devuelve el numero maximo de iteraciones a realizar */
public int getIteraciones(){
    return T;
}

/** realiza una iteracion completa */
public void iterar(){
    Float[] entrada = seleccionarEntrada();
    indice_bmu = BMU(entrada);
    NeuronaSOM BMU = neuronas.get(indice_bmu);

    ArrayList<NeuronaSOM> vecindario = null;

    vecindario = getVecindario(BMU, getRadio(t));

    for(int i = 0; i < vecindario.size(); i++){
        entrenar(vecindario.get(i), BMU, entrada, t);
    }

    t++;
}

/** entrena la red SOM */
public void entrenarSOM(){
    for(int i = 0; i < T; i++)
        iterar();

    asociarEntradas();
}

/** devuelve la iteracion actual */
public int getIteracion(){
    return t;
}

/** devuelve el vecindario de la BMU
 * @param BMU índice de la BMU
 * @param r radio del vecindario */
private ArrayList<NeuronaSOM> getVecindario(NeuronaSOM BMU, int r) {

    java.util.ArrayList<NeuronaSOM> vecinos = new java.util.ArrayList<NeuronaSOM>(0);

    int bmux = BMU.getX();
    int bmuy = BMU.getY();

    for(int y = bmuy - (r-1); y <= bmuy + (r-1); y++){
```

```
            for(int x = bmux - (r-1); x <= bmux + (r-1); x++){
                if( x < getC() & y < getF() & x >=0 & y >= 0){
                    vecinos.add(neuronas.get(indice(x,y)));
                }
            }
        }

        return vecinos;
    }

    /** devuelve el indice de un elemento x, y en la matriz neuronas */
    public int indice(int i, int j){
        int k = i + j * getC();
        return k;
    }

    /** devuelve las neuronas de la capa de clasificacion */
    public ArrayList<NeuronaSOM> getNeuronas(){
        return neuronas;
    }

    public int getF() {
        return F;
    }

    public int getC() {
        return C;
    }

    public int getEntrada(){
        return entrada;
    }

    public ArrayList<EntradaSOM> getEntradas(){
        return entradas;
    }

    public void setEntradas(ArrayList<EntradaSOM> V) {
        this.entradas = V;
        K = V.size();
    }

    public void setN(int i) {
        this.N = i;
        inicializarPesos();
    }
```

```
/** establece la iteración actual en 0 */
public void reset(){
    t = 0;
}

// devuelve los valores de una neurona que está situada en la posicion x, y de la grilla
public NeuronaSOM getNeurona(int x, int y){
    int indice = indice(x, y);

    return neuronas.get(indice);
}

/**
 * devuelve todas las entradas asociadas a una neurona
 * (las entradas para las cuales la neurona es la BMU)
 * @param i índice de la BMU
 */
public java.util.ArrayList<Integer> getEntradas(int indice){
    // índices de las entradas a buscar dentro de la matriz de entradas
    java.util.ArrayList<Integer> indices = new java.util.ArrayList<Integer>(0);
    Float[] e;

    /** se recorren todas las entradas marcadas de la red */
    for(int i = 0; i < entradas.size(); i++){
        e = entradas.get(i).getValores();
        if(BMU(e) == indice)
            indices.add(i);
    }

    return indices;
}

/** marca una neurona como defectuosa (representa una entrada que contiene defectos)
 * @param i índice de la neurona a marcar */
public void marcarNeurona(int indice){
    int bmu = BMU(entradas.get(indice).getValores());
    neuronas.get(bmu).setDefectuosa(true);
}

/**  establece las entradas que estan asociadas con una neurona */
public void asociarEntradas(){
    int bmu;
    EntradaSOM e;

    /** elimina todas las asociaciones anteriores */
    for(NeuronaSOM neurona : neuronas)
        neurona.eliminarAsociaciones();
```

```
    for(int i = 0; i < entradas.size(); i++){
        e = entradas.get(i);

        bmu = BMU(e.getValores());

        neuronas.get(bmu).asociarEntrada(e);
    }
}

/** desmarca una neurona defectuosa
 * @param i índice de la neurona a desmarcar */
 public void desmarcarNeurona(int i){
    int bmu = BMU(entradas.get(i).getValores());
    neuronas.get(bmu).setDefectuosa(false);
 }

 /** devuelve true si una region (entrada de la SOM) esta asociada a una neurona
  * defectuosa o false en caso contrario */
 public boolean isDefectuosa(int index){
     ArrayList<EntradaSOM> defectuosas = getEntradasDefectuosas();
     EntradaSOM e = entradas.get(index);
     boolean retValue = defectuosas.contains(e);

     return retValue;
 }

 /** devuelve un array con las entradas que estan marcadas como defectuosas */
 public ArrayList<EntradaSOM> getEntradasDefectuosas(){
     ArrayList<EntradaSOM> retValue = new ArrayList<EntradaSOM>(0);

     for(NeuronaSOM n : neuronas)
         if(n.isDefectuosa())
             retValue.addAll(n.getEntradasAsociadas());

     return retValue;

 }

/** devuelve un array con las neuronas defectuosas */
public ArrayList<NeuronaSOM> getNeuronasDefectuosas() {
    ArrayList<NeuronaSOM> defectuosas = new ArrayList<NeuronaSOM>(0);

    for(NeuronaSOM n : neuronas)
        if(n.isDefectuosa())
            defectuosas.add(n);
```

```
        return defectuosas;
    }

    /** devuelve los indices de las entradas marcadas como defectuosas */
    public ArrayList<Integer> getIndicesEntradasDefectuosas() {
        ArrayList<Integer> indices = new ArrayList<Integer>(0);
        ArrayList<EntradaSOM> entradasDefectuosas = getEntradasDefectuosas();

        for(EntradaSOM e : entradasDefectuosas)
            indices.add(e.getNumero());

        return indices;
    }

    /** devuelve el entorno-8 de una neurona sin incluirla en su vecindario */
    public ArrayList<NeuronaSOM> getVecindario8(NeuronaSOM n){
        ArrayList<NeuronaSOM> vecindario = new ArrayList<NeuronaSOM>(0);

        int x = n.getX();
        int y = n.getY();

        /** busca el vecindario de la neurona actual */
        for(int f = y - 1; f <= y + 1; f++){
            for(int c = x - 1; c <= x + 1; c++){

                /** comprueba que los índices estén dentro del margen */
                if(f >= 0 & c >= 0 & f < F & c < C){

                    /** comprueba que el índice no sea igual al de la neurona */
                    if(f != y | c != x)
                        vecindario.add(getNeurona(c, f));
                }
            }
        }

        return vecindario;
    }

    /** genera la U-Matrix para este SOM */
    public float[][] getUMatrix(){
        float[][] umatrix = new float[F][C];

        float u_height = 0f;

        ArrayList<NeuronaSOM> vecinos;

        for(NeuronaSOM n : neuronas){
```

```
            vecinos = getVecindario8(n);

            for(NeuronaSOM v : vecinos)
                u_height+= distancia(n.getValores(), v.getValores());

            umatrix[n.getX()][n.getY()] = u_height;

            u_height = 0;
        }

        return umatrix;
    }
}


/*
 * NeuronaSOM.java
 */

package nn;

import java.awt.image.BufferedImage;
import java.io.Serializable;
import java.util.ArrayList;

public class NeuronaSOM implements Serializable{

    private Float[] valores;
    private ArrayList<EntradaSOM> entradasAsociadas = new ArrayList<EntradaSOM>(0);
    private boolean defectuosa = false;
    private int x; // posicion x en una som bidimensional
    private int y; // posicion y en una som bidimensional

    /** Creates a new instance of NeuronaSOM */
    public NeuronaSOM() {
    }

    public Float[] getValores() {
        return valores;
    }

    public void setValores(Float[] valores) {
        this.valores = valores;
    }

    public ArrayList<EntradaSOM> getEntradasAsociadas() {
        return entradasAsociadas;
```

```
}

public void setEntradasAsociadas(ArrayList<EntradaSOM> entradasAsociadas) {
    this.entradasAsociadas = entradasAsociadas;
}

public void asociarEntrada(EntradaSOM e){
    entradasAsociadas.add(e);
}

/** elimina todas las asociaciones de entradas */
public void eliminarAsociaciones(){
    entradasAsociadas.clear();
}

/** devuelve un ArrayList con las imagenes de las entradas asociadas a la neurona */
public ArrayList<java.awt.image.BufferedImage> getImagenes(){
    ArrayList<java.awt.image.BufferedImage> imagenes = new
                        ArrayList<java.awt.image.BufferedImage>(0);

    for(EntradaSOM e : entradasAsociadas)
        imagenes.add(e.getImagen());

    return imagenes;
}

public boolean isDefectuosa() {
    return defectuosa;
}

public void setDefectuosa(boolean defectuosa) {
    this.defectuosa = defectuosa;
}

public int getX() {
    return x;
}

public void setX(int x) {
    this.x = x;
}

public int getY() {
    return y;
}

public void setY(int y) {
    this.y = y;
```

```
    }
}

/*
 * EntradaSOM.java
 */
package nn;

import java.io.Serializable;

public class EntradaSOM implements Serializable{

    private Float[] valores; // valores extraidos a partir de una imagen

    // imagen de la cual se extraen los valores de la entrada
    private transient java.awt.image.BufferedImage imagen;

    private int numero; // numero de entrada

    /** Creates a new instance of EntradaSOM */
    public EntradaSOM() {
    }

    public Float[] getValores() {
        return valores;
    }

    public void setValores(Float[] valores) {
        this.valores = valores;
    }

    public java.awt.image.BufferedImage getImagen() {
        return imagen;
    }

    public void setImagen(java.awt.image.BufferedImage imagen) {
        this.imagen = imagen;
    }

    public int getNumero(){
        return numero;
    }

    public void setNumero(int numero) {
        this.numero = numero;
    }
}
```

```
/*
 * Lvq.java
 */

package nn;
import java.io.IOException;
import java.io.Serializable;
import java.util.ArrayList;
import java.util.Iterator;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

public class Lvq implements Serializable {

    // neuronas de la capa de competición
    private ArrayList<NeuronaLVQ> neuronas = new ArrayList<NeuronaLVQ>(0);

    // clases definidas en la red
    private ArrayList<ClaseLVQ> clases = new ArrayList<ClaseLVQ>(0);

    // entradas de la red
    private ArrayList<Float[]> entradas = new ArrayList<Float[]>(0);

    public static final int SUBCLASES_POR_CLASE = 4;
    private int N = 0; // tamaño de las muestras

    public final float ALFA = 0.1f; // factor de aprendizaje

    private transient ArrayList<ChangeListener> listeners = new ArrayList<ChangeListener>(0);

    /** crea una nueva instancia de LVQ */
    public Lvq(){

    }

    /** establece el tamanio de las muestras */
    public void setN(int n){
        this.N = n;
    }

    /** devuelve la bmu para la entrada pasada como parametro */
    public NeuronaLVQ bmu(float[] e){
        float dist = 0; // distancia actual
        float distMinima = 0; // minima distancia encontrada
        NeuronaLVQ bmu; // best matching unit
        NeuronaLVQ n; // neurona que se está procesando
```

```
        Iterator<NeuronaLVQ> it = neuronas.iterator();

        bmu = it.next();
        distMinima = distancia(e, bmu);

        while(it.hasNext()){
            n = it.next();
            dist = distancia(e, n);

            if(dist < distMinima){
                distMinima = dist;
                bmu = n;
            }
        }

        return bmu;
    }

    /** entrena la neurona para que se asemeje mas al ejemplo */
    public void entrenar(NeuronaLVQ n, float[] e){
        float[] pesosNeurona = n.getPesos();

        for(int i = 0; i < N; i++)
            pesosNeurona[i] = pesosNeurona[i] + ALFA * (e[i] - pesosNeurona[i]);

        n.setPesos(pesosNeurona);

        fireChangeEvent();

    }

    /** penaliza la neurona para que se aleje del ejemplo */
    public void penalizar(NeuronaLVQ n, float[] e){
        float[] pesosNeurona = n.getPesos();

        for(int i = 0; i < N; i++)
            pesosNeurona[i] = pesosNeurona[i] - ALFA * (e[i] - pesosNeurona[i]);

        n.setPesos(pesosNeurona);

        fireChangeEvent();
    }


    /** devuelve la distancia euclídea entre una neurona y un vector de entrada */
    public float distancia(float[] e, NeuronaLVQ n){
        float dist = 0; // distancia entre una neurona y una entrada
```

```
        float[] pesos = n.getPesos(); // pesos de la neurona

        for(int i = 0; i < N; i++){
            dist+= (e[i] - pesos[i]) * (e[i] - pesos[i]);
        }

        return (float) Math.sqrt(dist);
    }

    /** agrega una clase con el nombre especificado */
    public void agregarClase(String n, float[] prototipo, java.awt.image.BufferedImage img,
                             java.awt.Color color){
        ClaseLVQ c = new ClaseLVQ(n);

        c.setPrototipo(prototipo);
        c.setImagenPrototipo(img);
        c.setColor(color);

        clases.add(c);

        NeuronaLVQ neurona;
        int numeroNeurona = neuronas.size();

        for(int i = 0; i < SUBCLASES_POR_CLASE; i++){
            neurona = new NeuronaLVQ(N);
            neurona.setPesos(prototipo);
            neurona.setClase(c);
            neurona.setNumero(numeroNeurona);
            numeroNeurona++;
            neuronas.add(neurona);
        }

        fireChangeEvent();
    }

    /** agrega una entrada a la red */
    public void agregarEntrada(Float[] e){
        entradas.add(e);
    }

    /** devuelve la cantidad de clases que existen en la red */
    public int getCantidadClases(){
        return neuronas.size() / SUBCLASES_POR_CLASE;
    }

    /** devuelve la cantidad de subclases */
    public int getCantidadSubClases(){
        return neuronas.size();
```

```
    }

    /** devuelve las neuronas de la capa de competicion */
    public ArrayList<NeuronaLVQ> getNeuronas(){
        return neuronas;
    }

    /** devuelve el tamanio de las muestras */
    public int getN(){
        return N;
    }

    /** devuelve las clases definidas */
    public ArrayList<ClaseLVQ> getClases(){
        return clases;
    }

    /** avisa a sus listeners que surgio un cambio en el modelo */
    private void fireChangeEvent(){
        ChangeEvent c = new ChangeEvent(this);

        for(ChangeListener l : listeners)
            l.stateChanged(c);
    }

    public void addChangeListener(ChangeListener l){
        listeners.add(l);
        fireChangeEvent();
    }

    public void removeChangeListener(ChangeListener l){
        listeners.remove(l);
        fireChangeEvent();
    }

    /** establece propiedades luego de recuperar el objeto de un archivo externo */
    private void readObject(java.io.ObjectInputStream s) throws java.io.IOException{
        try {
            s.defaultReadObject();

            listeners = new ArrayList<ChangeListener>(0);
        } catch (ClassNotFoundException ex) {
            ex.printStackTrace();
        }
    }
}
```

```
/*
 * NeuronaLVQ.java
 */

package nn;

import java.io.Serializable;


public class NeuronaLVQ implements Serializable {

    private float[] pesos; // pesos de la neurona
    private ClaseLVQ clase; // clase de la cual es subclase
    private int N; // tamanio del vector de pesos
    private int numero; // número identificatorio de la neurona

    /**
     * Creates a new instance of NeuronaLVQ
     */
    public NeuronaLVQ(int p) {
        N = p;
        pesos = new float[N];
        inicializar();
    }

    /** inicializa los pesos de la neurona */
    private void inicializar(){
        for(int i = 0; i < N; i++)
            pesos[i] = (float) Math.random();
    }

    /** devuelve los pesos de la neurona */
    public float[] getPesos() {
        return pesos.clone();
    }

    /** establece los pesos de la neurona */
    public void setPesos(float[] pesos) {
        this.pesos = pesos;
    }

    /** devuelve la clase a la cual representa la neurona */
    public ClaseLVQ getClase() {
        return clase;
    }

    /** establece la clase a la cual representa la neurona */
```

```
    public void setClase(ClaseLVQ clase) {
        this.clase = clase;
    }

    public int getNumero() {
        return numero;
    }

    public void setNumero(int numero) {
        this.numero = numero;
    }
}

/*
 * EntradaLVQ.java
 */

package nn;

public class EntradaLVQ{

    private float[] valores;
    private ClaseLVQ clase;
    private java.awt.image.BufferedImage imagen = null;

    /** Creates a new instance of EntradaLVQ */
    public EntradaLVQ() {
    }

    public float[] getValores() {
        return valores;
    }

    public void setValores(float[] valores) {
        this.valores = valores;
    }

    public ClaseLVQ getClase() {
        return clase;
    }

    public void setClase(ClaseLVQ clase) {
        this.clase = clase;
    }

    public java.awt.image.BufferedImage getImagen() {
        return imagen;
```

```
    }

    public void setImagen(java.awt.image.BufferedImage imagen) {
        this.imagen = imagen;
    }
}

/*
 * ClaseLVQ.java
 */

package nn;

import java.awt.Graphics2D;
import java.awt.image.BufferedImage;
import java.io.Serializable;

public class ClaseLVQ implements Serializable {

    private String etiqueta = null; // nombre de la clase

    // vector que representa una muestra prototipo de la clase
    private float[] prototipo = null;

    // color que se usa para marcar los defectos en la imagen
    private java.awt.Color color = null;

    javax.swing.ImageIcon imagen = null; // imagen de referencia de la clase

    /** Creates a new instance of ClaseLVQ */
    public ClaseLVQ() {
    }

    /** crea una nueva instancia de ClaseLVQ y le asigna una etiqueta */
    public ClaseLVQ(String s){
        etiqueta = s;
    }

    /** devuelve la etiqueta de la clase */
    public String getEtiqueta() {
        return etiqueta;
    }

    /** establece la etiqueta de la clase */
    public void setEtiqueta(String etiqueta) {
        this.etiqueta = etiqueta;
    }
```

```
    /** devuelve el prototipo de la clase */
    public float[] getPrototipo() {
        return prototipo;
    }

    /** establece el prototipo de la clase */
    public void setPrototipo(float[] prototipo) {
        this.prototipo = prototipo;
    }

    public BufferedImage getImagenPrototipo() {
        java.awt.Image image = imagen.getImage();

        BufferedImage bufferedImage = new BufferedImage(
            image.getWidth(null), image.getHeight(null),
            BufferedImage.TYPE_INT_RGB);

        Graphics2D g2 = bufferedImage.createGraphics();

        g2.drawImage(image, null, null);

        return bufferedImage;
    }

    public void setImagenPrototipo(java.awt.image.BufferedImage imagenPrototipo) {
        imagen = new javax.swing.ImageIcon(imagenPrototipo);

    }

    public String toString() {
        return etiqueta;
    }

    public java.awt.Color getColor() {
        return color;
    }

    public void setColor(java.awt.Color color) {
        this.color = color;
    }
}
```

APÉNDICE D - INTERFACES GRÁFICAS

En este apéndice se muestran las interfaces gráficas utilizadas en el prototipo del sistema de clasificación.

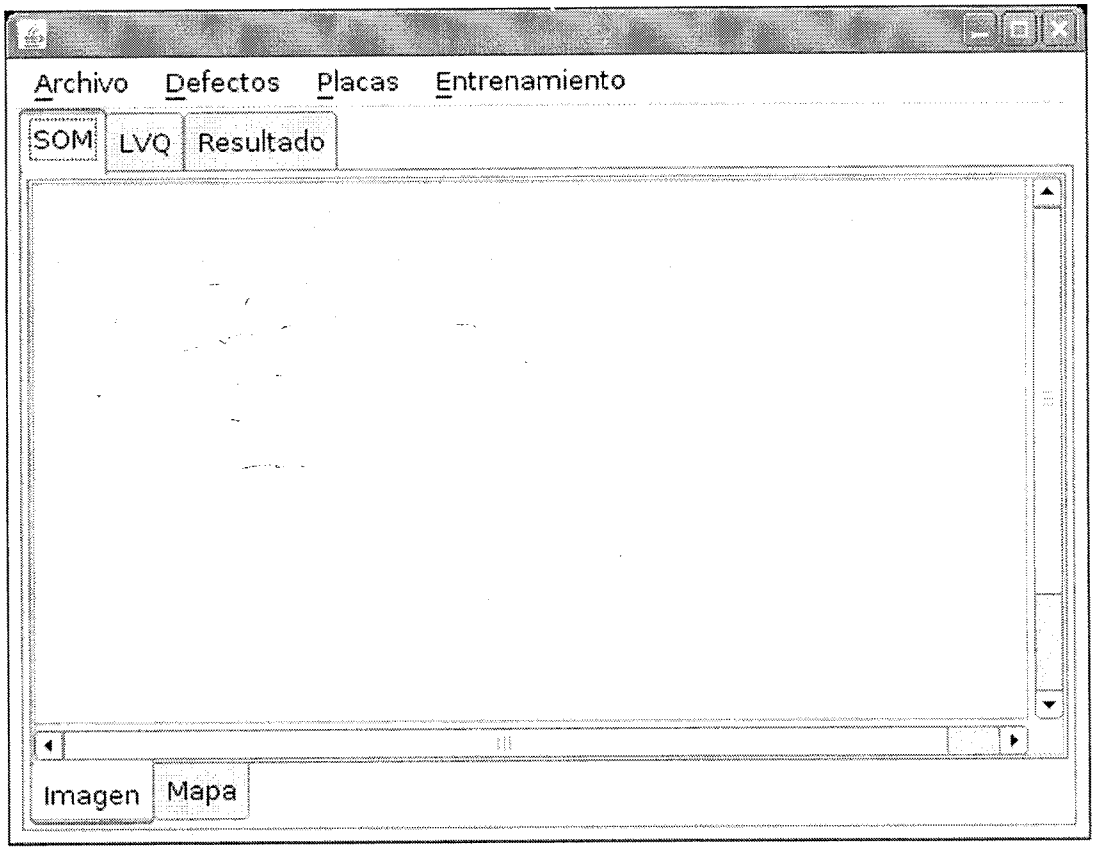


Ilustración D.1: Pantalla principal del sistema

En la Ilustración D.1 se pude apreciar la ventana principal del sistema. Esta ventana contiene una barra de menú y un área principal, la cual tiene tres pestañas en su parte superior. Las pestañas ubicadas en la parte inferior del área principal permiten cambiar entre dos modos diferentes de visualización del SOM (imagen y mapa) y sólo aparecen cuando la pestaña superior SOM está activa. Estos modos de visualización serán explicados mas adelante. La pestaña LVQ permite ver información acerca de la red LVQ, como los pesos actuales de las neuronas y las clases disponibles (creadas por el usuario). En la pestaña resultado se visualizan los resultados de una clasificación realizada por el sistema.

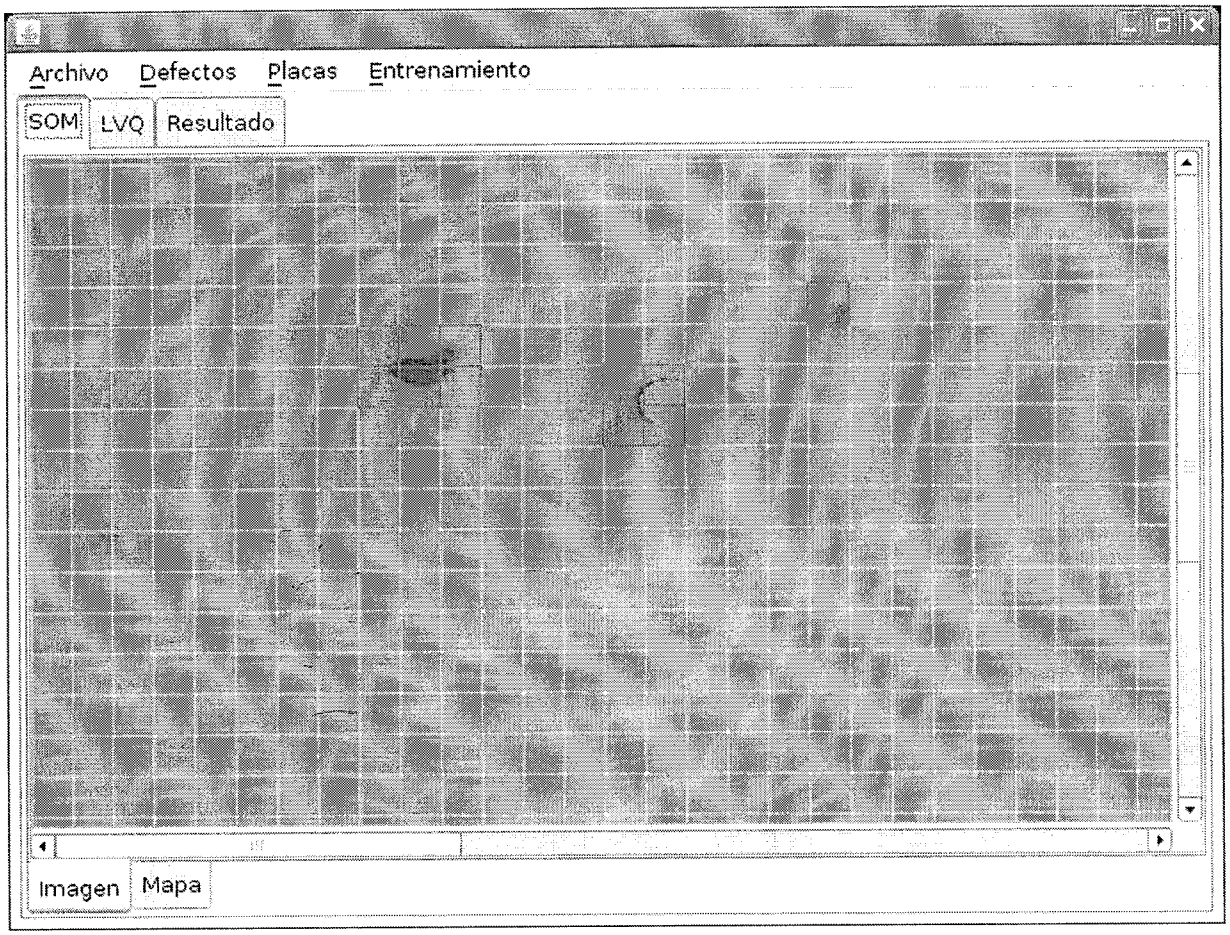


Ilustración D.2: Entrenamiento del SOM

La Ilustración D.2 muestra la pantalla principal en modo de entrenamiento del SOM. Para realizar el entrenamiento del SOM se debe seleccionar la opción SOM del menú entrenamiento. Al seleccionar esta opción, se desplegará un navegador de archivos en el cual el usuario debe seleccionar la imagen a utilizar en el entrenamiento. En este caso la imagen pertenece a una placa de pino. La cuadrícula mostrada en color verde representa la división de la imagen en regiones de 24 x 24 píxeles. Cada una de estas regiones representa una entrada al SOM. Al hacer click con el mouse sobre una región, el sistema marca esta región como defectuosa y busca además en el SOM la BMU para esa región y marca todas las demás entradas asociadas con esa BMU como defectuosas en la imagen. Estas regiones defectuosas aparecen en la imagen con un recuadro de color rojo. En el caso de esta imagen, se realizó un click en una sola región y el sistema automáticamente marcó todas las demás zonas defectuosas.

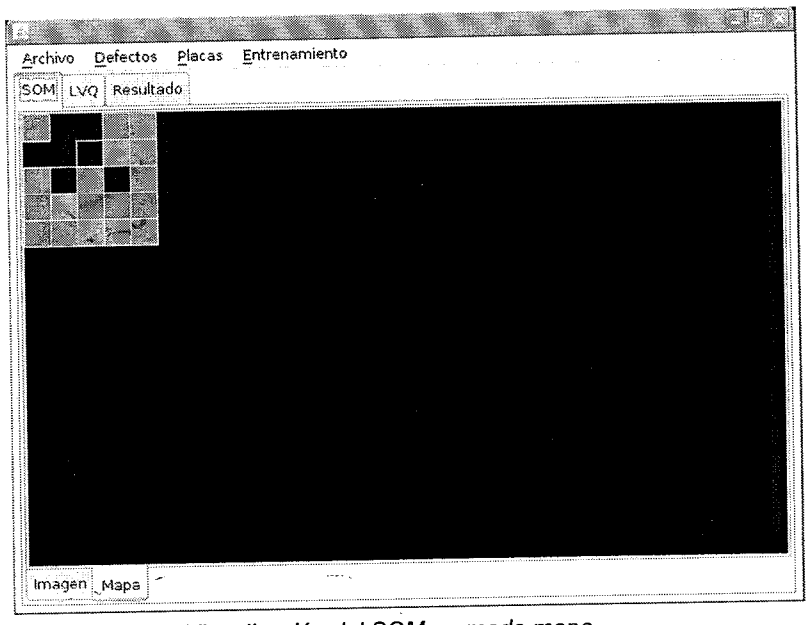


Ilustración D.3: Visualización del SOM en modo mapa

La Ilustración D.3 muestra el entrenamiento del SOM en modo mapa. Esta es otra forma de entrenamiento del SOM. Como se puede ver, las zonas con defectos se ubican en la parte inferior derecha del mapa. En este caso, cada cuadro representa una neurona del SOM. Al hacer click en una neurona con el botón izquierdo del mouse, el sistema marca la neurona seleccionada (y todas las entradas asociadas con ella en la vista Imagen) como defectuosa. Si se hace click nuevamente, la neurona es desmarcada. Las neuronas marcadas como defectuosas representan las zonas de la imagen que contienen fallas y están representadas en la imagen con un recuadro de color naranja. Las neuronas que no contienen fallas se visualizan con un recuadro verde. La imagen mostrada en cada neurona es una de las imágenes asociadas a esta.

Al hacer click con el botón derecho del mouse sobre una neurona se despliega una ventana como la de la Ilustración D.4. En esta ventana se pueden ver todas las regiones de la imagen asociadas con la neurona seleccionada, como así también los valores actuales de esa neurona. A medida que el entrenamiento del SOM continúa, estos valores generalmente cambiarán.

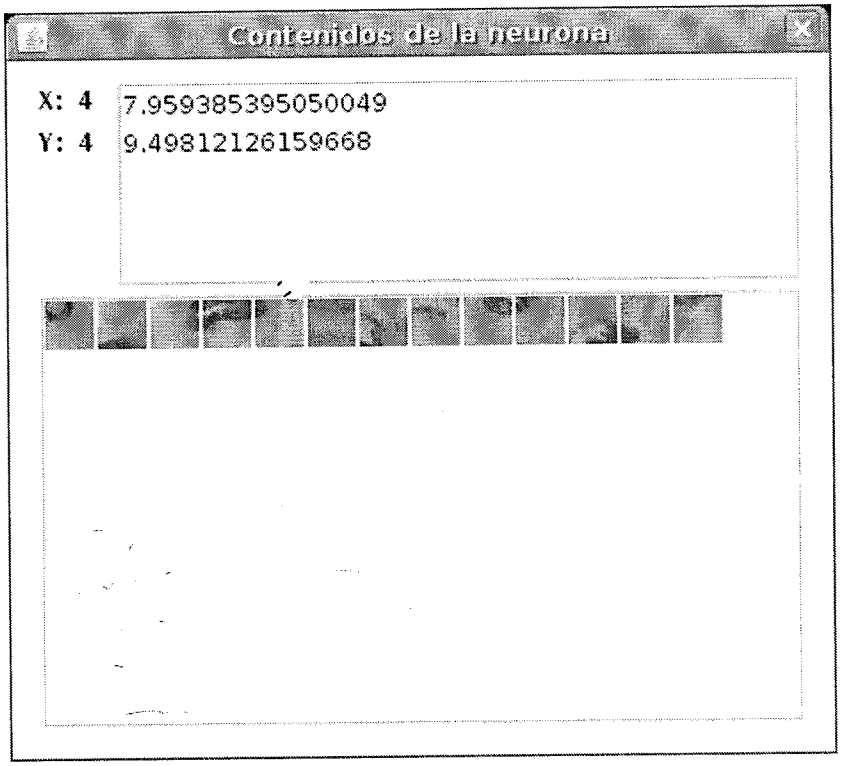


Ilustración D.4: Contenidos de una neurona

El procedimiento de entrenamiento del SOM puede repetirse con la cantidad de placas deseadas. Se debe cargar una imagen para el entrenamiento y luego seleccionar las regiones defectuosas en la imagen o en el mapa.

El entrenamiento del SOM se puede realizar antes o después de entrenar la red LVQ, no importa el orden en que se hagan los entrenamientos.

Antes de entrenar la red LVQ, se deben crear las clases a utilizar. Además de crear las clases con defectos, se deberá crear una que represente las partes sin defectos de las imágenes, esto es por si el SOM permite pasar alguna región sana a la red LVQ. Para crear una nueva clase se selecciona la opción nueva clase, del menú defectos. Esta operación mostrará una ventana como la de la Ilustración D.5.

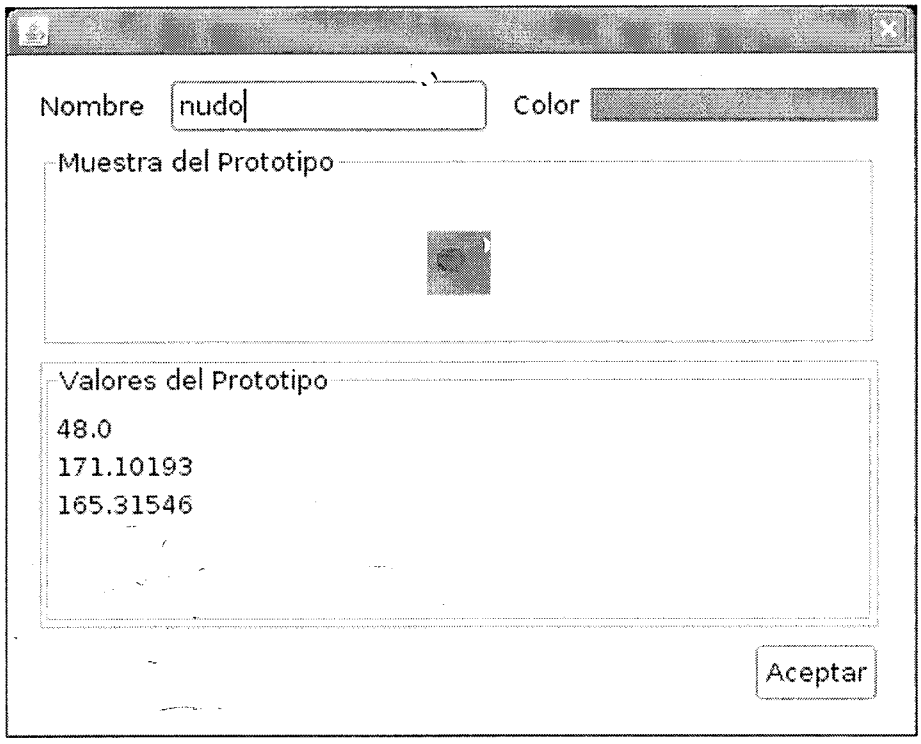


Ilustración D.5: Nuevo defecto

En la ventana para crear una nueva clase se deben completar los datos requeridos por el sistema. Se debe asignar un nombre a la clase, como así también un color. Este color se asigna haciendo click en el cuadro que aparece a la derecha de la etiqueta color. Esta acción despliega un diálogo de selección de color, donde el usuario puede asignar cualquier combinación de colores RGB a la clase a crear. En este caso se seleccionó el color rojo para la clase nudo. Este color se utilizará mas adelante para marcar todas las fallas detectadas como nudo por el sistema en la imagen a analizar. Además del color y el nombre de la clase, el usuario debe seleccionar una imagen prototipo que represente a la clase. De esta imagen se extraerán los valores de iniciales de la clase, los cuales se muestran en la lista ubicada en la parte inferior de la ventana.

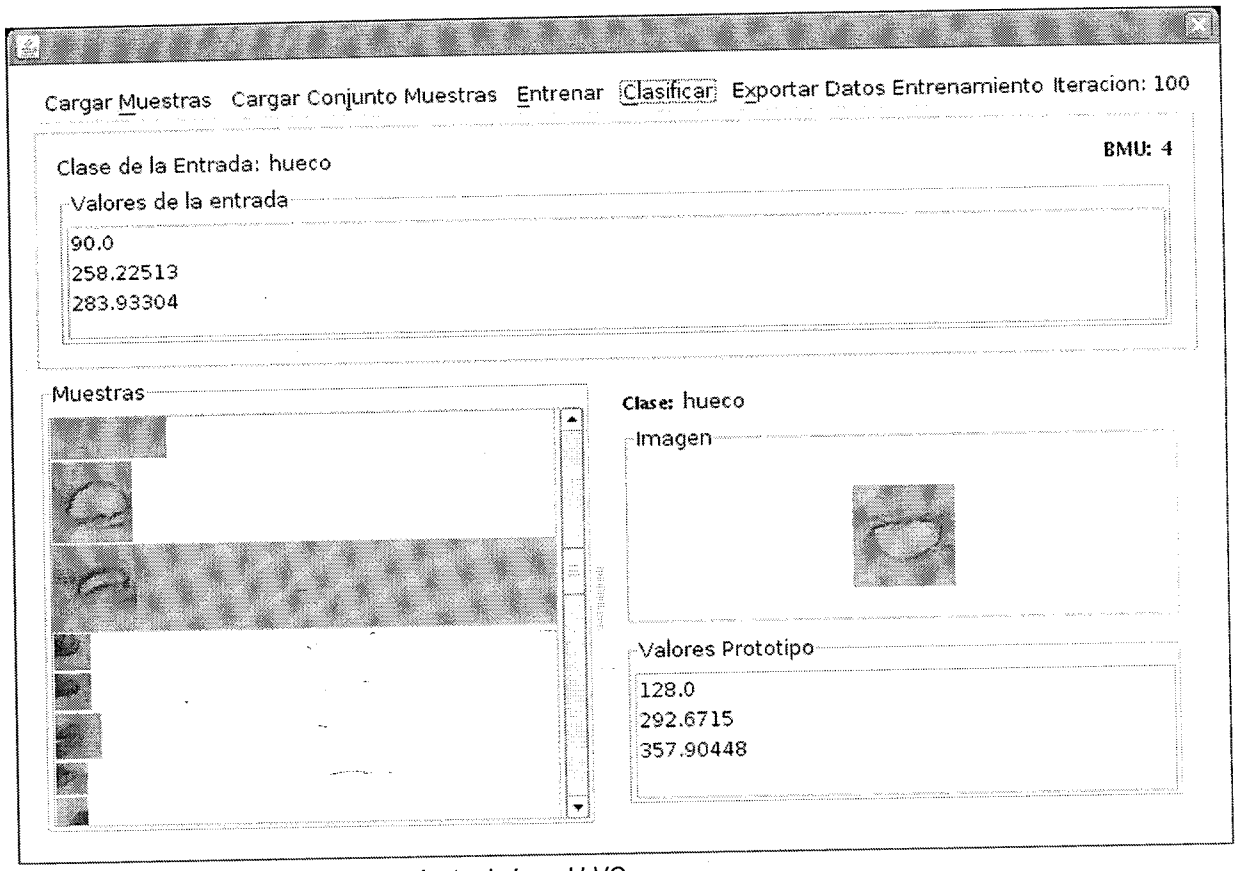


Ilustración D.6: Ventana de entrenamiento de la red LVQ

Una vez creadas todas las clases a utilizar, se puede proceder al entrenamiento de la red LVQ. Para esto se debe seleccionar la opción LVQ del menú entrenamiento. Esto despliega una ventana como la de la Ilustración D.6, la cual está dividida en cuatro partes. En la parte superior se encuentra una barra en donde se muestran las acciones disponibles. Debajo de esta barra, se muestran los datos acerca de la muestra seleccionada en la lista de muestras, ubicada en la esquina inferior izquierda de la ventana. En la esquina inferior derecha se muestran los resultados actuales de clasificación de la red LVQ.

Las acciones disponibles en la barra superior son las siguientes:

- **Cargar muestras:** Despliega un diálogo que permite seleccionar varios archivos de imagen al usuario. Luego muestra cada imagen y pregunta al usuario a que clase (creada previamente) pertenece.

- **Cargar conjunto de muestras:** Ídem a la opción anterior, con la diferencia de que el usuario deberá seleccionar varias imágenes que representen a una misma clase, el sistema luego preguntará una sola vez a que clase pertenecen las imágenes y asignará a todas las imágenes seleccionadas esta clase.

- **Entrenar:** Entrena la red LVQ con las muestras cargadas por el usuario. Estas muestras se ven el la lista de muestras. Por defecto, por cada vez que se presione sobre el botón entrenar, el sistema realizará 100 iteraciones en el entrenamiento de la red LVQ.

- **Clasificar:** Al seleccionar esta opción, el sistema clasifica la muestra seleccionada en la lista de muestras y muestra el resultado de la clasificación para la muestra seleccionada en la parte inferior derecha de la ventana.

- **Exportar datos entrenamiento:** Exporta los valores de las neuronas de la red LVQ y las entradas utilizadas para su entrenamiento a un archivo de texto separado por comas, para poder analizar los resultados.

En la parte derecha de la barra superior se muestra la iteración actual.

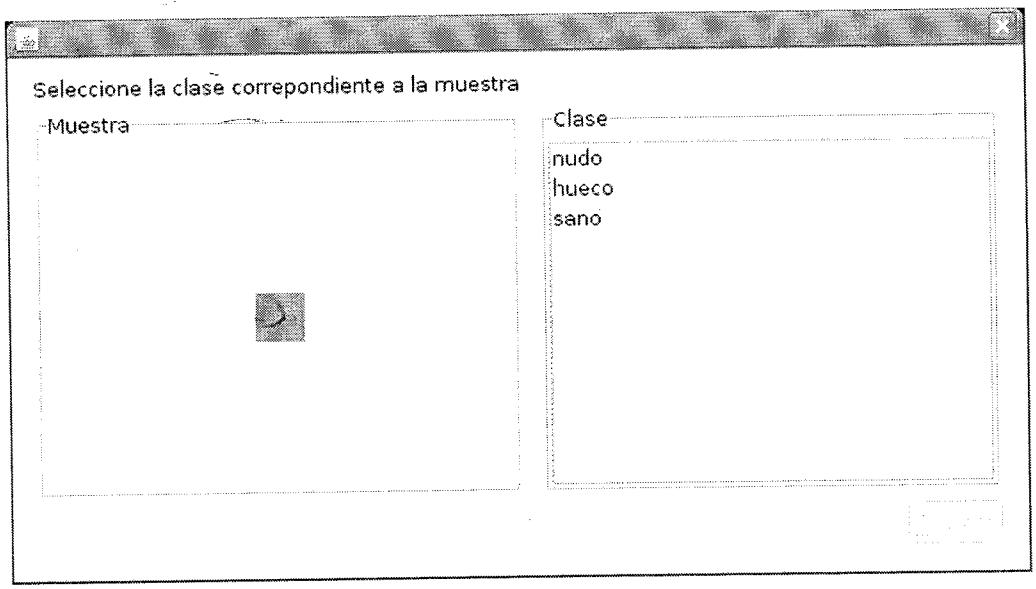


Ilustración D.7: Ventana de carga de muestras

En la Ilustración D.7 se puede ver la ventana que permite seleccionar al usuario la clase que representa una muestra. En el caso de cargar las muestras con la opción "cargar muestras", esta ventana aparecerá una vez por cada muestra seleccionada. En el caso de cargar las muestras mediante la opción "cargar conjunto de muestras" la ventana aparecerá una sola vez y asignará a todas las muestras la clase seleccionada.

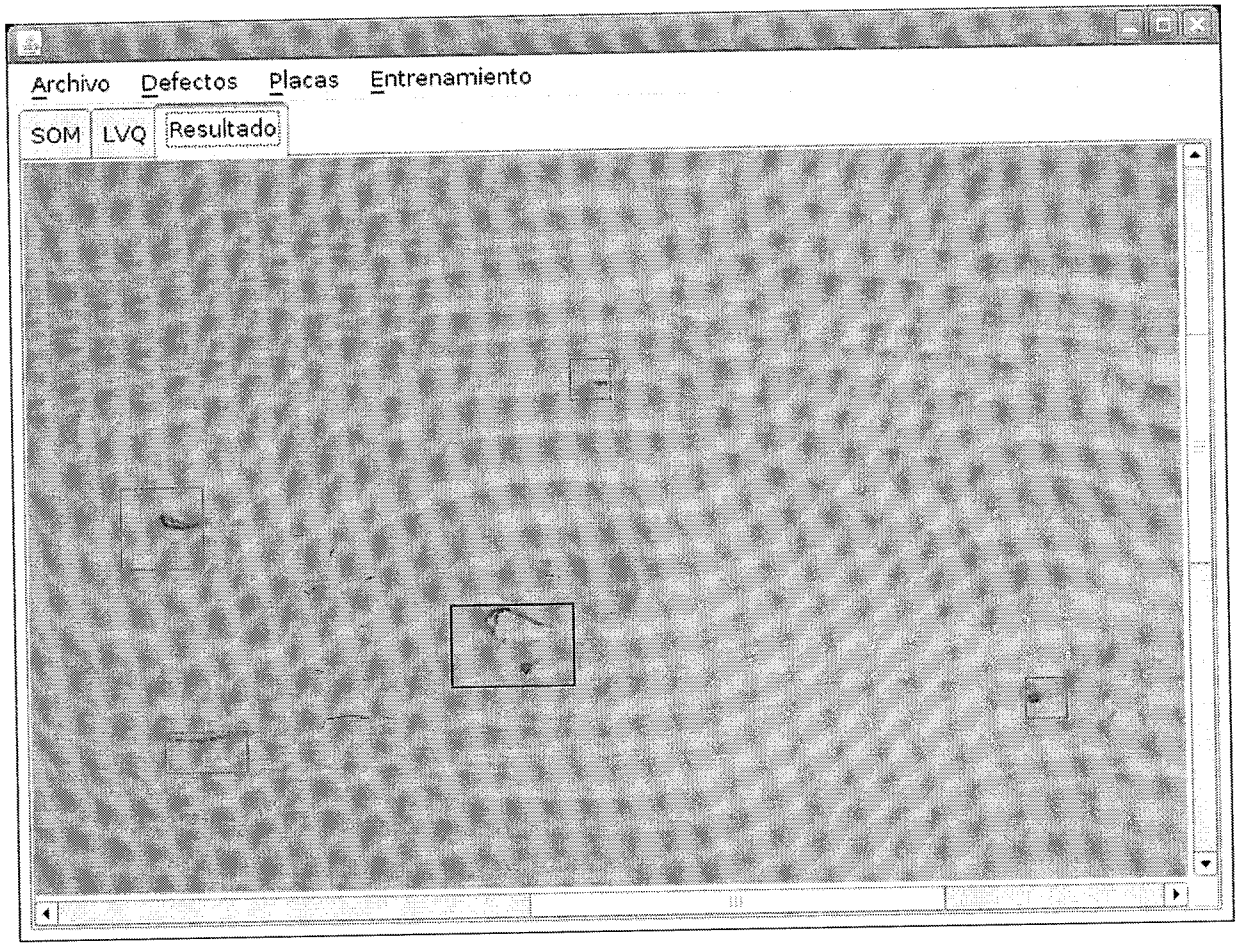


Ilustración D.8: Resultados de clasificación

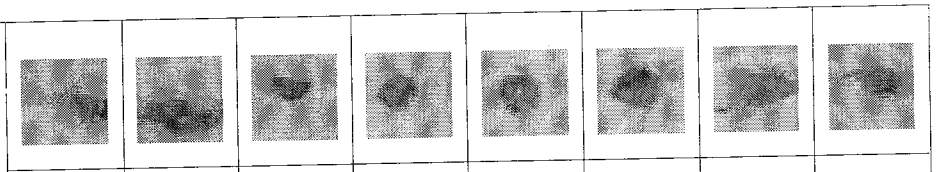
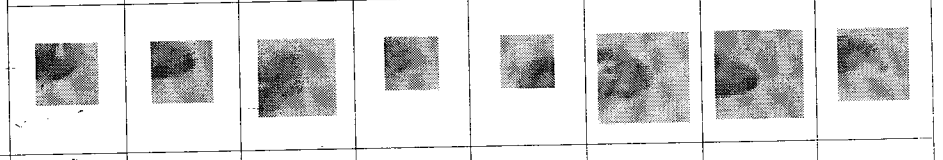
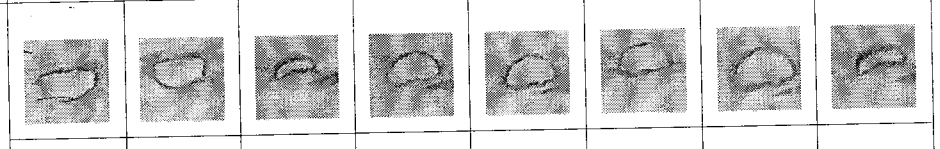
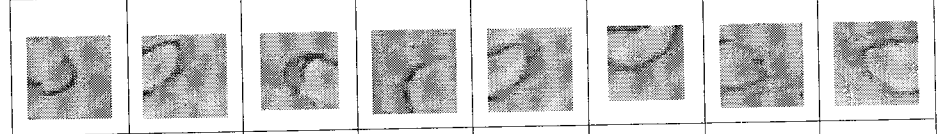
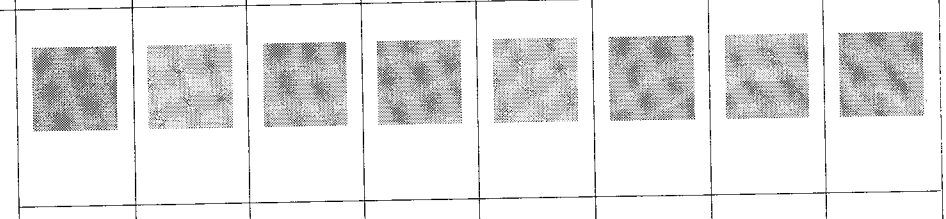
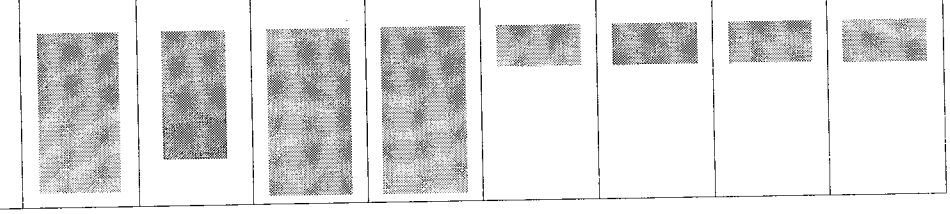
La Ilustración D.8 muestra los resultados de realizar una clasificación luego de haber entrenado el sistema. En este caso se tienen tres clases: nudo, hueco y sin defectos. Los nudos son marcados en la imagen con un recuadro rojo, los huecos con un recuadro azul, y las partes sanas en negro. En la imagen mostrada no se detectaron zonas sanas, esto es debido a que el SOM no permitió que pasaran regiones sin defectos a la clasificación por la red LVQ. En caso de que la imagen no quepa en pantalla, el usuario puede desplazarse con las barras de desplazamiento horizontal y vertical. Para realizar una clasificación, el usuario debe seleccionar la opción cargar del menú Placa.

Además de las funciones descriptas anteriormente, el sistema cuenta con varias opciones adicionales, como guardar un entrenamiento realizado para utilizarlo posteriormente, cargar un entrenamiento guardado y exportar la u-matrix del SOM utilizado.

APÉNDICE E - RESULTADOS DE CLASIFICACIÓN

Se presentan en esté apéndice algunos resultados obtenidos por el sistema de clasificación.

E.1. Muestras de Pino

Nudo	
	
Hueco	
	
Sin defecto	
	

Estas muestras fueron obtenidas de imágenes de placas completas y seleccionadas manualmente para el entrenamiento de la red LVQ, en este caso, todas las muestras fueron clasificadas correctamente, como se ve en la Ilustración E.1.

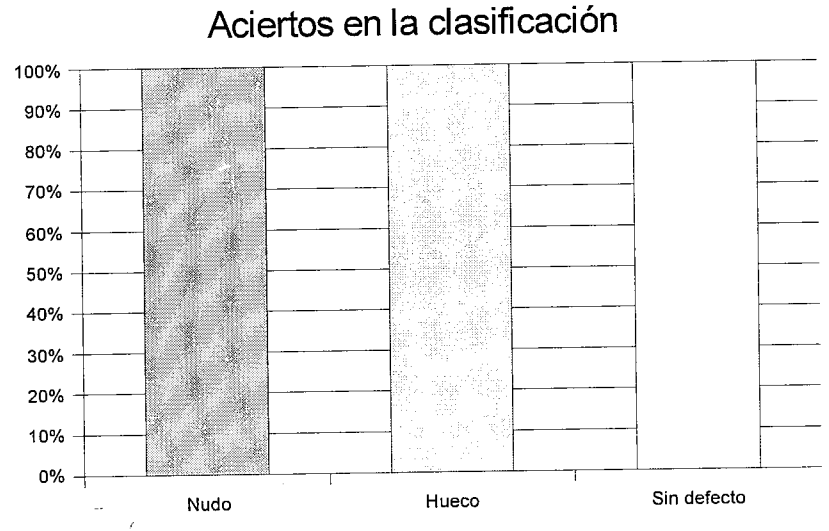
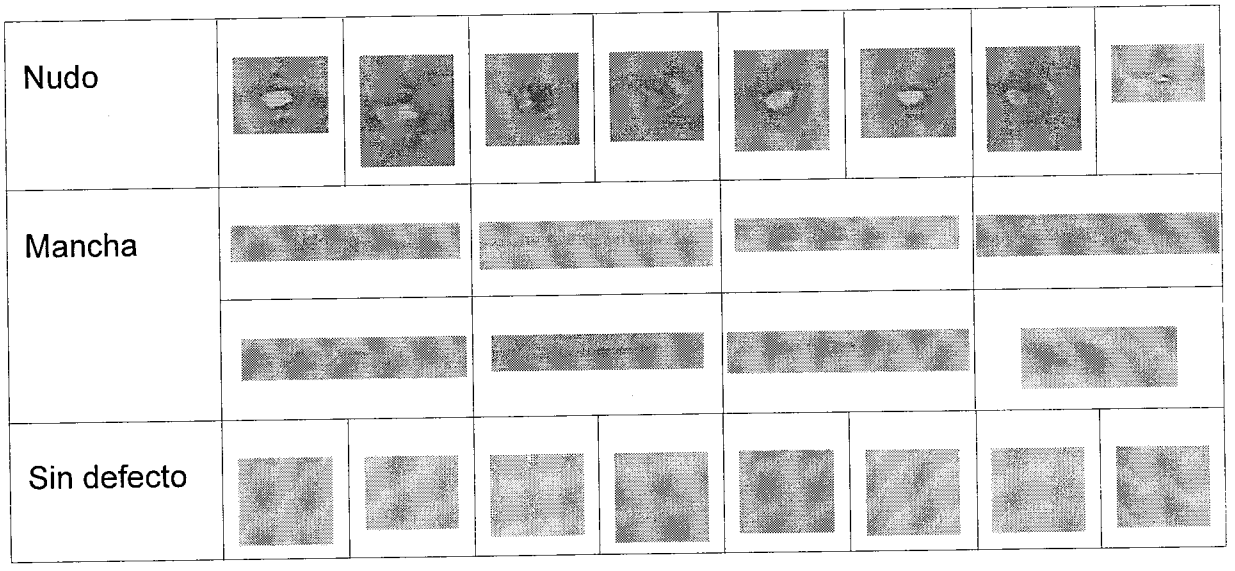


Ilustración E.1: Aciertos en la clasificación para muestras de pino

E.2. Muestras de Guatambú



En el caso de las muestras de guatambú, se reconocieron correctamente las clases nudo y mancha, pero algunas muestras sin defectos fueron clasificadas erróneamente como manchas.

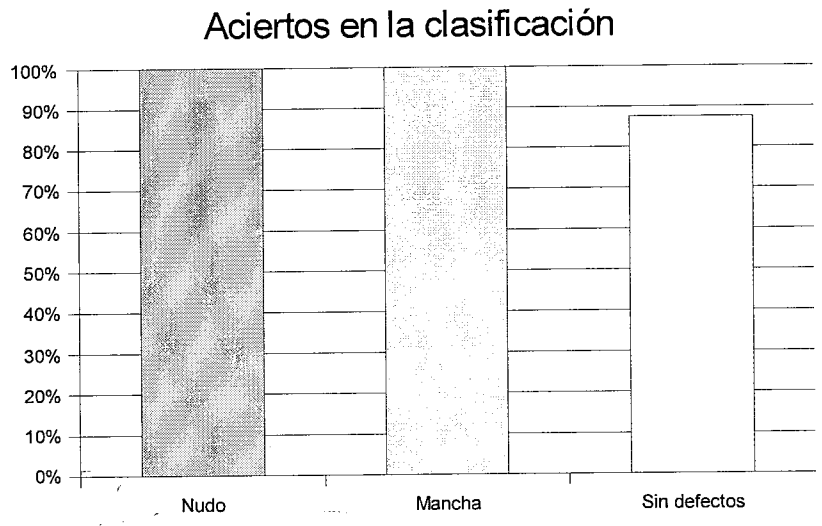


Ilustración E.2: Aciertos en la clasificación para muestras de guatambú

Las muestras obtenidas para las clasificaciones fueron tomadas de imágenes de placas, en el caso del sistema estas imágenes son adquiridas teniendo en cuenta las salidas del SOM y luego recortadas utilizando algoritmos de detección de bordes. En algunos casos, estos algoritmos no son lo suficientemente buenos como para aislar las regiones con defectos del fondo, por lo que al momento del reconocimiento de las fallas, se producen mas errores.

BIBLIOGRAFÍA

[1]: Libro - Gonzalo Pajares, et al.; "Imágenes Digitales" , Edición Original, Alfaomega, Madrid - España, 2004

[2]: Libro - Gonzalo Pajares, Jesús M. de la Cruz; "Visión por Computador" , Edición Original, Alfaomega, Madrid - España, 2002

[3]: Libro - José R. Hilera González, Victor J. Martínez Hernando; "Redes Neuronales Artificiales. Fundamentos, Modelos y Aplicaciones" , Edición Original, RA-MA, Madrid - España, 1995

[4]: Libro - Isasi Viñuela, P; Galván León, I.M.; "Redes de Neuronas Artificiales, un enfoque práctico" , Edición Original, Pearson Educación, Madrid - España, 2004

[5]: Informe de investigación - Claus Bahlmann, Gunther Heidemann, Helge Ritter, Artificial Neural Networks for Automated Quality Control of Textile Seams, AG Neuroinformatik - Universität Bielefeld, Universitätstrasse 25, Bielefeld, Alemania, 1999

[6]: Informe de investigación - L.I. Passoni, et al., Sistema de Soporte a las Decisiones Médicas utilizando herr. Neuro-Fuzzy, Facultad de Ingeniería - Universidad Nacional de Mar del Plata, , 2002

[7]: Documento WWW - http://www.cis.hut.fi/teuvo/, 2007

[8]: Libro - Isasi Viñuela, P; Galván León, I.M.; "Redes de Neuronas Artificiales, un enfoque práctico" , Edición Original, Pearson Educación, Madrid - España, 2004

[9]: Documento WWW - http://davis.wpi.edu/~matt/courses/soms/, 1999

[10]: Informe de investigación - Alfred Ultsch, U*-Matrix: A Tool to visualize Clusters in high dimensional data, Department of Computer Science - University of Marburg, Alemania, 2003

[11]: Documento WWW - http://www.ai-junkie.com/ann/som/som1.html, 2005

[12]: Documentación técnica - Teuvo Kohonen, et al., The Self-Organizing Map Program Package, http://www.cis.hut.fi/research/som_pak/som_doc.ps, 1995

[13]: Documento WWW - http://www.neural-forecasting.com/lvq_neural_nets., 2005

[14]: Documento WWW - http://en.wikipedia.org/wiki/Gregory_Voronoi, 2007

[15]: Informe de investigación - Oswin Aichholzer, Franz Aurenhammer, Skew Voronoi Diagrams, Institute for Theoretical Computer Science - Graz University of Technology, Austria, 1998

[16]: Documentación técnica - Teuvo Kohonen, et al., The Learning Vector Quantization Program Package, http://www.cis.hut.fi/research/lvq_pak/lvq_doc.ps, 1995

[17]: Informe de investigación - Olli Silvén, Matti Niskanen, Hannu Kauppinen, Wood inspection with non-supervised clustering, Department of Electrical Engineering - University of Oulu, Finlandia, 2001

[18]: Informe de reunión - Instituto Argentino de Normalización, Compensados de madera: clasificación según el aspecto de las caras, Buenos Aires - Argentina, 2000

[19]: Informe de investigación - Graciela M. de Jesús Ramírez, Mario I. Chacón Murgía, Clasificación de Defectos en Madera utilizando Redes Neuronales Artificiales, Laboratorio de DSP y Visión - Instituto Tecnológico de Chihuahua, Chihuahua - México, 2005

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS