UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

DESARROLLO DE UN SISTEMA BASADO EN UN CLASIFICADOR NEURONAL PARA RECONOCIMIENTO DE ORUGAS

T E S I S

QUE PARA OPTAR POR EL TÍTULO DE:

INGENIERO EN COMPUTACIÓN

PRESENTA:

MARCO ANTONIO RODRÍGUEZ FLORES

DIRECTOR DE TESIS:

DRA. TETYANA BAYDYK

México, D.F., Ciudad Universitaria, abril 2010.

Este trabajo se desarrolló en el Laboratorio de Computación Neuronal del Centro de Ciencias Aplicadas y Desarrollo Tecnológico (CCADET) de la Universidad Nacional Autónoma de México (UNAM), bajo la tutoría de la Dra. Tetyana Baydyk y con el apoyo de los proyectos PAPIIT IN110510-3, PAPIIT IN119610 y al proyecto de ICyTDF 332/2009.

Jurado Asignado:

Presidente:	Dr. Felipe de Jesús Lara Rosano.
Vocal:	Dra. Tetyana Baydyk.
Secretario:	M. I. Juan Manuel Gómez González.
1er suplente:	Dr- Ernst Kussul.
2do suplente:	Dra. Graciela Velasco Herrera.

Agradecimientos.

Le doy gracias a Dios por darme la vida para poder concretar una más de mis metas.

A mis padres Antonio Rodríguez y Lourdes Flores por brindarme su comprensión, cariño y apoyo. Porque gracias a sus consejos y sacrificio no hubiese logrado llegar hasta este punto de culminar mi carrera. Los amo.

A mis hermanos Lidia, Verónica, Nicte y Víctor, por siempre estar conmigo en todo momento brindándome su apoyo. Sé que puedo contar con cada uno de ustedes cuando lo necesite.

A mis pequeños sobrinos Oscar, Diego y Fernanda, a los que quiero muchísimo.

A mis abuelitos, tíos, primos y demás familia por el apoyo que me han brindado, en especial la fam. Flores Gaytán.

A mi tutora de tesis la Dra. Tetyana Baydyk por la confianza, asesoramiento e inversión de tiempo para la propuesta, el desarrollo y la conclusión de este trabajo.

Al Dr. Ernst Kussul por brindarme la orientación y asesoramiento en la realización de este trabajo.

A la Dra. Graciela Velasco por abrirme las puertas de su laboratorio, brindándome el gran apoyo, motivación y confianza desde un inicio, así como los gratos momentos.

A mi amiga Lucero por los consejos, apoyo, confianza y motivación que me ha dado para seguir adelante, además por los dichosos momentos vividos no solo en la carrera. Realmente gracias.

A mis amig@s Elba, Estela, Karem, Leticia, Ruth, Abraham, Bando, Paty, Aris y Norma a quienes realmente aprecio. Así como los compañer@s que olvide mencionar pero con los cuales compartí diversos momentos a lo largo de la carrera.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPÍTULO 1.

INTRODUCCIÓN

El reconocimiento de imágenes es un problema importante en el campo de visión computacional, pues se requiere de la construcción de modelos con algoritmos y diseño de hardware influenciados por cómo estos problemas son resueltos naturalmente [1].

En la actualidad, existen diversos grupos de investigación que han desarrollado distintos trabajos en el área de reconocimiento de imágenes. Uno de los grupos de investigación en dicha área se localiza en el laboratorio de Computación Neuronal del Centro de Ciencias Aplicadas y Desarrollo Tecnológico (CCADET) de la UNAM, a cargo de los investigadores Dr. Ernst Kussul y Dr. Tetyana Baydyk. Éste grupo ha creado un sistema de visión computacional sustentado en redes neuronales artificiales.

El sistema desarrollado lleva por nombre Clasificador Neuronal LIRA *Grayscale,* el cual está basado en la red neuronal PERCEPTRON [2], [3]. Dicho clasificador ha sido probado con éxito en tareas de reconocimiento de dígitos escritos a mano [4], clasificación de imágenes de microtornillos [5], [6], y tareas de ensamble de microdispositivos [7].

El presente trabajo trata sobre el clasificador mencionado adaptado a la tarea de reconocimiento de orugas en los cultivos [8], [9]. La investigación parte del análisis del estado del arte de las redes neuronales, procesamiento de imágenes y clasificadores neuronales enfocados al problema en particular y las características requeridas del sistema para que le dé solución. Se presentan la descripción y análisis de los experimentos y resultados hechos.

1.1. Impacto de los pesticidas en México.

Entre los factores que limitan la producción agrícola y la calidad de las cosechas están las enfermedades y las plagas, las cuales pueden atacar a los cultivos desde que las plantas inician su crecimiento hasta la cosecha.

Si bien, los pesticidas (que comprenden un amplio grupo de compuestos químicos heterogéneos) producen un beneficio público al incrementar la productividad de la agricultura, al permitir controlar la proliferación de plagas y enfermedades en los cultivos, así como reducir o evitar las pérdidas en la producción de alimentos. La aplicación indiscriminada, el manejo incorrecto y sin control de estos, en nuestro país, ocasiona

daños al ambiente, por ejemplo, el deterioro de la flora y fauna silvestres, la contaminación del suelo, de mantos freáticos y aguas continentales y costeras. Así como la generación de plagas resistentes [10].

Los efectos de los plaguicidas sobre las diversas formas de vida se pueden clasificar como primarios y secundarios. En el primero el plaguicida actúa directamente sobre una especie dada, en la segunda no actúa directamente sobre la especie, pero destruye su sustrato o hábitat [10].

Pero no solo resulta perjudicial al medio ambiente, sino también a la salud del hombre, principalmente en los fetos y los niños debido a sus efectos adversos [11], lo cual puede manifestarse por intoxicaciones nocivas de diferente grado que pueden presentarse a mediano o largo plazo, tales como carcinogénesis, mutagénesis, esterilidad entre otros.

La exposición ocupacional a los pesticidas puede alterar la fecundidad de los agricultores, los cuales pueden presentar un número reducido de espermatozoides, aumento en el número de espermatozoides morfológicamente anormales y porcentaje reducido de espermatozoides viables en la eyaculación. Esto significa que la exposición a los pesticidas en los agricultores es un factor de riesgo significativo de infertilidad [11].

Los efectos tóxicos observados en humanos se obtienen a partir de estudios en poblaciones ocupacionalmente expuestas y en individuos cuya exposición fue incidental. Existe una amplia variedad de efectos tanto crónicos como agudos reportados, y van desde efectos sistémicos hasta la muerte [10].

La información estadística de la figura 1.1, contiene los casos reportados de exposición y/o intoxicación por pesticidas durante el periodo de enero a septiembre del 2006. Donde se observa que los estados con mayor número de casos reportados son el estado de México, Jalisco, Chiapas, Puebla y Michoacán.

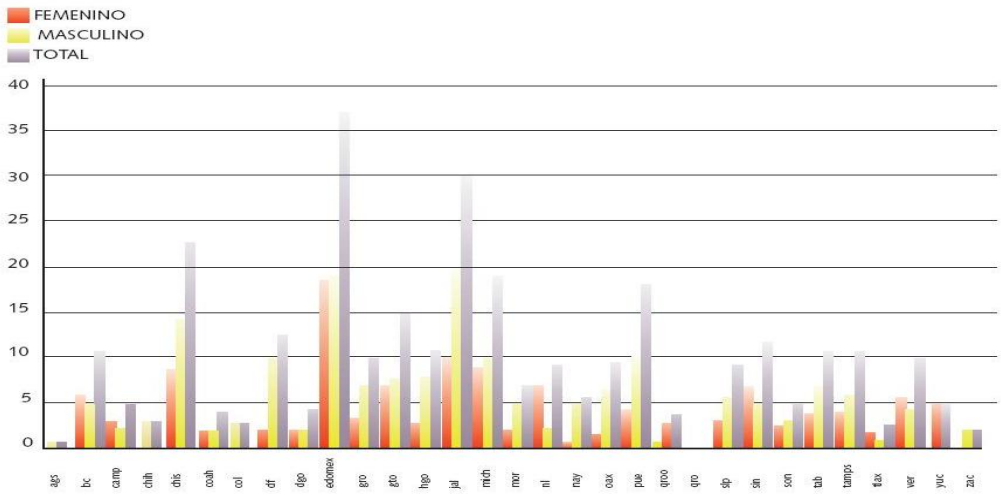


Figura 1.1. Casos reportados por entidad federativa y por sexo de enero a septiembre 2006 [12]

Los fetos y los niños tienen mayor riesgo de exposición ambiental a los pesticidas en el aire, el agua y el suelo que los adultos. El feto está en riesgo a partir de la exposición materna debido a la transferencia placentaria de estos agentes. Ya que su sistema de detoxificación hepático es inmaduro [11].

En la gráfica de la figura 1.2 se observa que las edades con mayor número de casos reportados son de 1 a 4 años y de 20 a 29 años, predominando el sexo masculino.

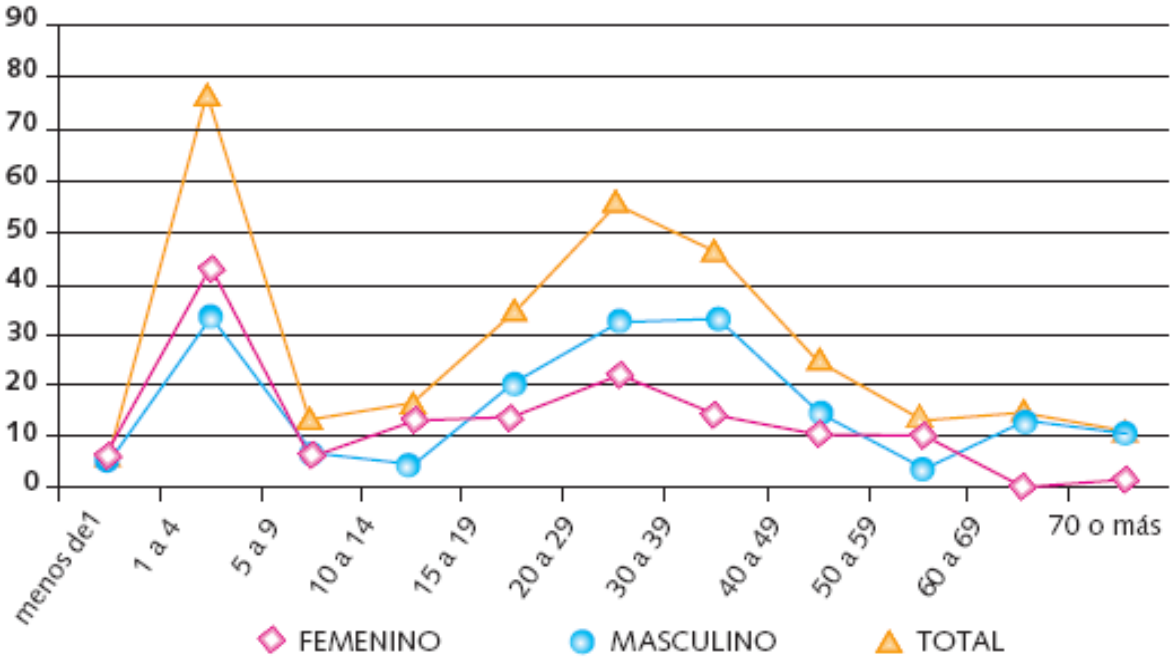


Figura 1.2. Casos reportados por edad y sexo de enero a septiembre 2006 [12]

Actualmente, no hay un consenso internacional con respecto a los niveles de seguridad de los residuos de pesticidas en las comidas, pero en los EE.UU. se prohibió la utilización de algunos pesticidas organoclorados como el DDT o se restringieron (heptacloro).

Los únicos plaguicidas cuya importación, comercialización y uso están permitidos en México, son los que han sido registrados por la Comisión Intersecretarial para el Control del Proceso y Uso de Plaguicidas, Fertilizantes y Sustancias Tóxicas (CICOPLAFEST).

En el siguiente mapa del país (figura 1.3) se ilustra el uso del DDT por entidad federativa entre los años de 1988-1999.

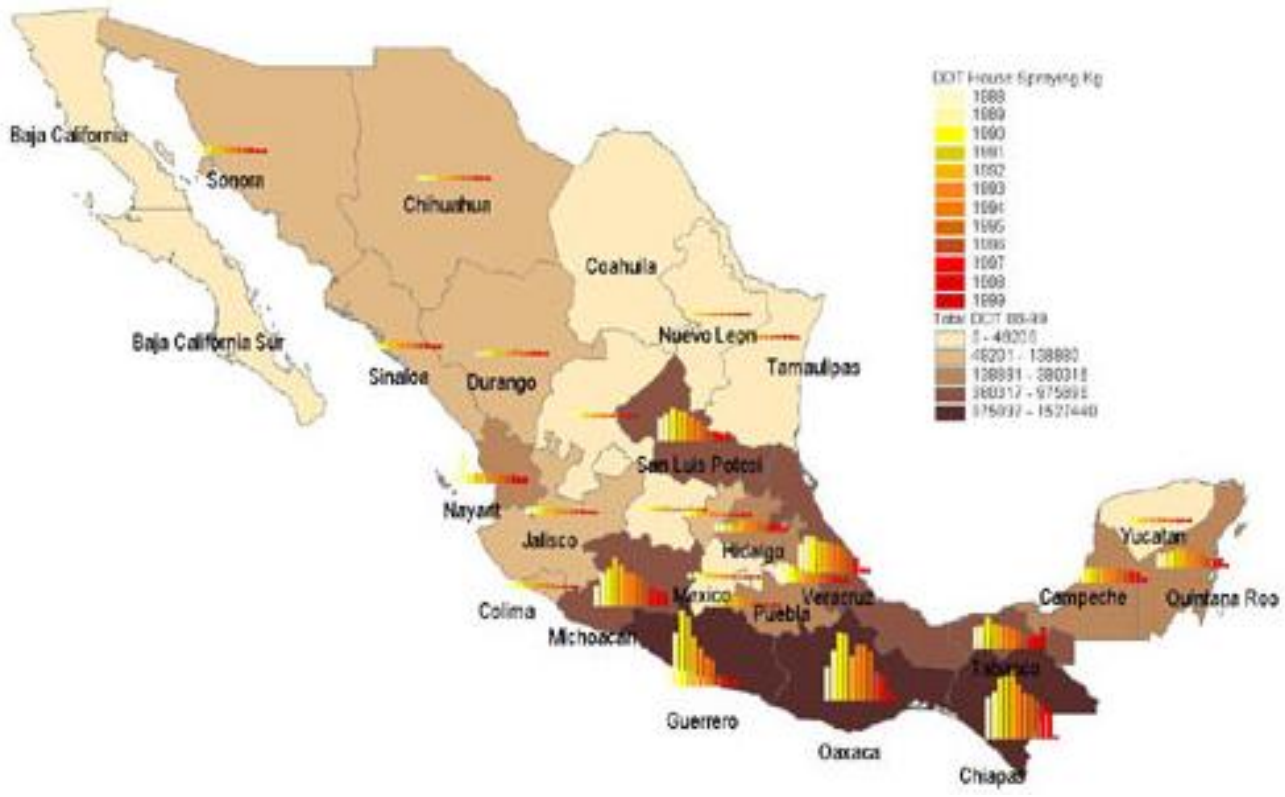


Figura 1.3. Uso del DDT en México por entidad federativa. Años 1988-1999 [13]

El DDT es uno de los muchos pesticidas utilizados en agricultura, pero que tienen efectos adversos documentados sobre la reproducción. Si bien ya no se utilizan en diversas regiones, aún se emplean en países subdesarrollados como el caso de México.

Es por eso, la necesidad de implementar prácticas ecológicas con tecnologías limpias que reduzcan, en lo posible, el uso de los pesticidas, tomando en cuenta los efectos en la salud humana, la sustentabilidad agrícola, el medio ambiente y la economía.

1.2. Objetivos generales.

El objetivo de este trabajo es proponer un sistema de reconocimiento de orugas basado en un clasificador neuronal como una alternativa para el control de plagas en los cultivos, monitoreando las áreas de localización para saber cómo están distribuidas y así evitar el uso de grandes cantidades de pesticidas en el campo.

1.2.1. Justificación.

Durante los últimos años se ha utilizado gran cantidad de agroquímicos (pesticidas, herbicidas, plaguicidas, insecticidas) para el control de plagas en México. Esto provoca grandes daños a la salud de las personas y al medio ambiente, que están en contacto con cualquier tipo de ellos.

Para la reducción de las cantidades de pesticidas se necesita localizar más precisamente la distribución de las plagas en los cultivos, con el propósito de dosificarlos de manera eficaz y lograr el menor daño. Por este motivo necesitamos desarrollar un sistema de reconocimiento de imágenes con la intención de lograr la menor exposición posible, al contacto que las personas dedicadas al sector agrícola, puedan tener con cualquier tipo de pesticida, así como la disminución de las áreas de los cultivos donde se aplican.

La tarea no es fácil ya que muchos insectos utilizan la técnica de mimesis (imitación) para esconderse entre el follaje, pero para el desarrollo del sistema propuesto en este trabajo, se dan las características de las formas y texturas que dan la posibilidad de encontrarlos. Por esta razón se recurre al uso de visión computacional con el apoyo de redes neuronales para poder localizarlos. De esta manera se evita utilizar grandes volúmenes de pesticidas que resultan nocivos tanto para los agricultores como para los cultivos.

1.2.2. Metodología.

La realización del presente trabajo en primer lugar se efectuó una investigación con la literatura relacionada para tener un esbozo general de las redes neuronales artificiales y su funcionamiento. Aunado a la investigación sobre el diseño y estructura de un sistema de clasificación para el reconocimiento de imágenes. Describiendo algunos de los clasificadores neuronales ya desarrollados, haciendo énfasis en el clasificador neuronal LIRA.

Enseguida, en base a estos fundamentos teóricos, se lleva a cabo el desarrollo y programación de un sistema de visión por computadora basado en el clasificador neuronal LIRA para el reconocimiento de imágenes de orugas.

Posteriormente, se procede a realizar los experimentos necesarios para el entrenamiento y validación del sistema con el conjunto de imágenes especificas de la tarea en cuestión. Esta parte se divide en dos fases: primero el entrenamiento de la red neuronal artificial usando una base de imágenes de orugas de diferentes formas y colores, diferentes

cantidades y diferentes posiciones. La segunda fase es la verificación del sistema con la otra parte de la base de imágenes de orugas.

Al finalizar se realiza un análisis detallado de los resultados obtenidos, así como las conclusiones.

1.3. Objetivo de la tesis.

El objetivo principal de la tesis es el diseño y programación de un sistema de visión computacional con uso de redes neuronales, basado en el clasificador neuronal LIRA *Grayscale*, que sea útil para la tarea reconocer orugas en los cultivos, considerando las características reales y diversas en tamaño y forma que puedan tener.

CAPÍTULO 2.

REDES NEURONALES ARTIFICIALES

Las redes neuronales artificiales, RNA (denominadas en inglés *Artificial Neural Networks*, ANN) son los paradigmas de aprendizaje y procesamiento automático inspirados en la forma en que funciona el sistema nervioso biológico [14]. Se conforman por la interconexión de elementos simples de proceso (neuronas), que operan de forma paralela para dar una solución alterna a problemas complejos relacionados con el reconocimiento de formas o texturas, predicción, codificación, control y optimización.

Históricamente dos grupos de investigadores han trabajado con las redes neuronales artificiales. Un grupo motivado por el objetivo de usar RNA's para estudiar y modelar procesos de aprendizaje biológicos, y otro grupo motivado por el objetivo de obtener máquinas altamente eficaces con algoritmos de aprendizaje, independientemente si éstos reflejan procesos biológicos [15].

En la actualidad las RNA's constituyen un activo campo multidisciplinario, en el que participan investigadores de diferentes áreas, como la electrónica, física, matemáticas, ingeniería, biología y psicología ya que son una forma de emular ciertas características propias de los humanos, como la capacidad de memorizar y de asociar hechos para la ejecución de ciertas tareas en particular.

2.1. Historia.

Los primeros estudios en la historia de las redes neuronales comienza con el científico español Santiago Ramón y Cajal, descubridor de la estructura neuronal del sistema nervioso, demostrando que éste estaba compuesto por una red de células individuales, las neuronas, interconectadas entre sí [16].

A partir de la década de los 40, con el desarrollo de las computadoras, comienza el estudio sobre redes neuronales artificiales y su simulación. A partir de entonces se han ido incrementando con fuerza hasta la actualidad. Diversos científicos han conseguido que las redes neuronales tomen un lugar relevante en la ciencia, por eso cabe mencionar algunos de los que forman parte en la historia de las redes neuronales artificiales [17].

Año	Autor(es)	Aportación
1943	Warren McCulloch y Walter Pitts	Desarrollan el primer modelo de una neurona artificial [18]. Este modelo introduce la idea de una función de paso por umbral utilizada posteriormente por muchos otros modelos.
1949	Donald Hebb	Desarrolló un paradigma de entrenamiento: *Regla de Hebb* [19]. Sus trabajos formaron las bases de la teoría de las RNA's.
1951	Marvin Minsky	Diseñó una máquina con 40 neuronas, construida con tubos, motores y relés, cuyas conexiones se ajustaban automáticamente mediante procesos hebbianos [20].
1956	Albert Uttley	Desarrolló una máquina teórica compuesta de elementos de proceso, para simular fenómenos atmosféricos y el reconocimiento adaptivo de patrones [21], [22].
1958	Frank Rosenblatt	Desarrolló el PERCEPTRON [2], [3], un modelo 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.
1960	Bernand Widrow y Hoff	Desarrolló la ley de *Widrow-Hoff*, y las RNA´s llamadas ADALINE (Adaptive Linear Element) y MADALINE (Multiple ADALINE) [23]. Éstas han sido utilizadas para procesamiento adaptivo de señales, sistemas de control y sistemas adaptivos de antenas [24].
1961	Steinbuch	Creó métodos de codificación de información en RNA's [25], para el reconocimiento de escritura a mano distorsionada, mecanismos de diagnostico de fallos de maquinarias y control de múltiples procesos en producción [26].
1964	Stephen Grossberg	Realizó estudios sobre el procesamiento humano de la información. Sus trabajos incluyen un estricto análisis matemático que permiten tener un acceso directo a la información mientras se opera en tiempo real [27].
1969	Minsky y Papert	Realizaron una crítica del PERCEPTRON, revelando serias limitaciones, como su incapacidad de representar la función XOR, debido a su naturaleza lineal, provocando una crisis en las investigaciones [28].
1972	Shun-Ichi Amari	Combinó la actividad de redes neuronales biológicas con rigurosos modelos matemáticos de RNA's [29]. Sus estudios incluyen el tratamiento de RNA´s dinámicas y aleatoriamente conectadas, aprendizaje competitivo, y análisis matemático de memorias asociativas.
1974	Paul Werbos	Desarrolló el método de entrenamiento *Backpropagation*; algoritmo más utilizado en las aplicaciones de RNA´s [30].

Año	Autor(es)	Aportación
1977	James Anderson	Desarrolló el Asociador Lineal [31] y su extensión *Brain-state-in-a-Box* para modelar funciones complejas. Empleando un nuevo método de corrección de error, y la función umbral rampa [32].
1978-1980	Kunihiko Fukushima	Realizó estudios sobre modelos espaciales y espacio-temporales para sistemas de visión con RNA's, desarrollando un paradigma multicapa llamado COGNITRON [33], y su versión mejorada el NEOCOGNITRON [34].
1980-1984	Teuvo Kohonen	Desarrolló el modelo SOM (*Self-Organizing Maps*) [35], capaz de formar mapas de características a través de una organización matricial de neuronas artificiales. El aprendizaje de éste modelo es de tipo no supervisado.
1982	Feldman y Ballard	Realizaron trabajos sobre visión por computadora, basándose en modelos de memoria visual, lenguaje natural y representación de conceptos abstractos [36], [37].
	John Hopfield	Provocó el renacimiento de las redes neuronales con su trabajo: "Computación neuronal de decisiones en problemas de optimización." [38]. Elaboró un modelo de red neuronal auto-asociativa, aplicando los principios de estabilidad desarrollados por Grossberg [39].
1985	Bart Kosko	Creó memorias asociativas bidimensionales, utilizando aprendizaje sin supervisión, capaces de converger a una solución mínima dada una matriz arbitraria [40].
1985-1986	Sejnowski y Hinton	Desarrollaron el algoritmo de la máquina de Boltzmann que reconoce patrones y optimiza funciones simples [41].
1986	Carpenter y Grossberg	Desarrollaron el modelo ART (*Adaptive Resonance Theory*) para aplicaciones de reconocimiento de patrones [42].
	Robert Hecht-Nielsen	Fue el principal diseñador de uno de los primeros computadores neuronales, dedicado al procesamiento de paradigmas de RNA's. Este neuro-computador, el TRW MARK III, está soportado por un computador VAX de DIGITAL, y estuvo comercialmente disponible [43].
	Rumelhart	Perfecciona el algoritmo *Backpropagation*, ofreciendo una poderosa solución para la construcción de RNA´s más complejas [44].
1995	Vapnik	Desarrolló la teoría de aprendizaje de Vapnik conocido como SVM (*Support Vector Machine*), la cual proporciona las bases teóricas para el desarrollo de métodos de aprendizaje dando datos finitos [45].

En la actualidad las investigaciones y el desarrollo sobre las RNA's han continuado su progreso, es por eso que son numerosos los trabajos que se realizan y publican cada año.

2.2. La neurona biológica.

La neurona es la célula fundamental del sistema nervioso especializada en procesar información. El cerebro humano se compone de 10^{11} neuronas interconectadas entre sí por 10^{13} conexiones formando redes que desarrollan funciones complejas específicas [14].

Una neurona consta de tres elementos: cuerpo celular, dendritas, y un axón, que termina en las conexiones sinápticas (Figura 2.1).

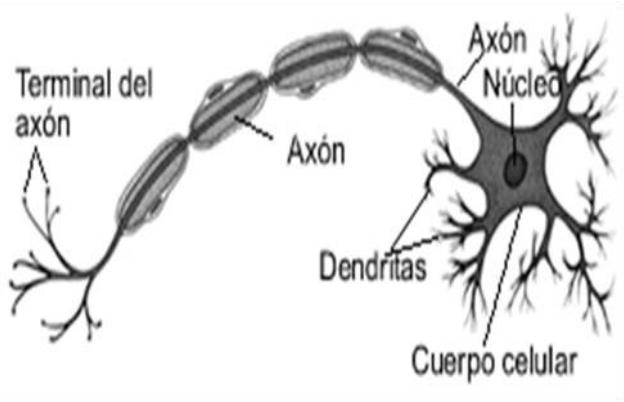


Figura 2.1. Estructura de una neurona biológica [46]

Esquemáticamente una neurona recoge las señales procedentes de otras neuronas, a través de las dendritas. Estas señales se combinan, y en función de la estimulación total recibida, la neurona toma cierto nivel de activación, que traduce en la generación de breves impulsos nerviosos con una determinada frecuencia o tasa de disparo.

Si estos impulsos son superiores al "umbral de activación" la neurona propaga su señal a través del axón. Las extremidades de estas ramificaciones llegan hasta las dendritas de otras neuronas y establecen una conexión llamada sinapsis, que transforman el impulso eléctrico en un mensaje neuroquímico mediante la liberación de una sustancia llamada neurotransmisor.

El efecto sobre la neurona receptora puede ser excitador o inhibidor. El aprendizaje se produce mediante la variación de la efectividad de la sinapsis.

2.3. Definición de red neuronal artificial y sus elementos.

Una red neuronal artificial es ampliamente una estructura de procesamiento paralelo distribuido, en forma de grafo dirigido, compuesta por unidades de procesamiento simple (neuronas artificiales), las cuales son utilizadas para almacenar información y hacerlo disponible para su uso [43].

Existen diversos modelos de RNA's en los cuales se siguen filosofías de diseño, reglas de aprendizaje y funciones de construcción de las respuestas muy distintas. Pero las tres características principales, las cuales describen a una red neuronal, y con lo cual contribuye a sus habilidades funcionales son: estructura, dinámica y entrenamiento [14].

Los elementos básicos de una RNA son [16]:

- Neuronas artificiales.
- Conjunto de entradas x_j.
- Pesos sinápticos w_j.
- Regla de propagación u.
- Función de activación $f(u)$.
- Función de salida y.

2.3.1. Neurona artificial

Una neurona artificial (o elemento de procesamiento) es una simplificación de la neurona biológica (Figura 2.2), que recibe información de otras neuronas artificiales o sensores, ejecutan operaciones simples en los datos y proporcionan una salida [14].

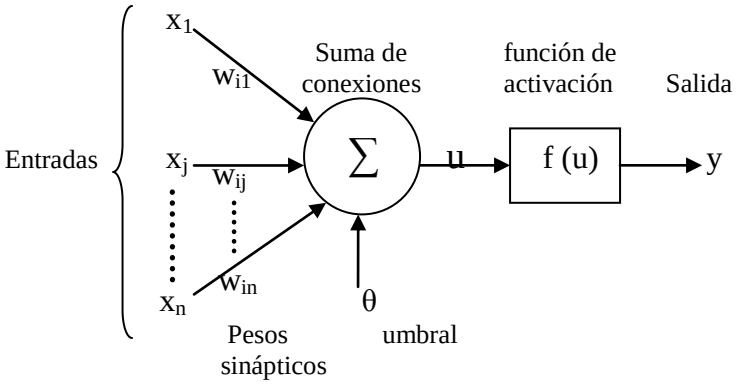


Figura 2.2. Modelo de una neurona artificial.

En la figura 2.2 todas las señales de entrada en la neurona $(x_1, x_j, \ldots, x_n)$ se multiplican por sus respectivos pesos sinápticos $(w_{i1}, w_{ij}, \ldots, w_{in})$ formando una entrada total u, lo que constituye la función de activación $f(u)$.

$$f(u) = f\left(\sum_{j=1}^{n} w_{ij}x_j + \theta\right) \qquad (2.1)$$

El termino θ (umbral de activación) que es añadido a la suma de las entradas, específica cierto desplazamiento debido a las características internas de la propia neurona, y no es igual en todas ellas.

El resultado de la función de activación es la salida y de la neurona artificial.

$$y = f(u) \qquad (2.2)$$

Esta salida se distribuye entre los pesos de las conexiones de las otras neuronas artificiales.

2.3.2. Conjunto de entradas.

El conjunto de entradas x_j es un vector formado por las señales que son recibidas desde el entorno, es decir, aquellas procedentes de sensores u otras neuronas en la misma red. Pueden ser digitales o analógicas.

2.3.3. Pesos sinápticos.

Los pesos sinápticos w_j, son valores numéricos (fraccionarios o enteros) con los que se ponderan las señales de entrada y representan la intensidad de interacción entre cada neurona presináptica j y la neurona postsináptica i. Las salidas basan su información en los valores de dichos pesos. Si el peso es positivo $w_{ij} > 0$, la conexión entre las neuronas es excitadora, si el peso es negativo $w_{ij} < 0$ la conexión es inhibidora.

2.3.4. Regla de propagación.

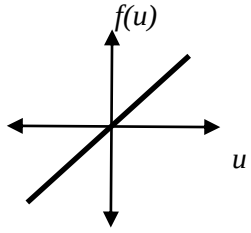
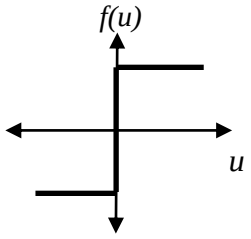
La regla de propagación $u(w_{ij}, x_j)$ también conocida como función de ponderación o de excitación, proporciona el valor del potencial postsináptico de la neurona i en función de sus pesos y entradas.

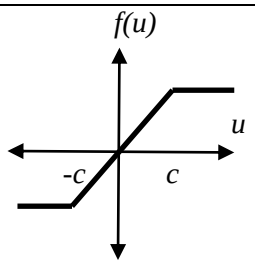
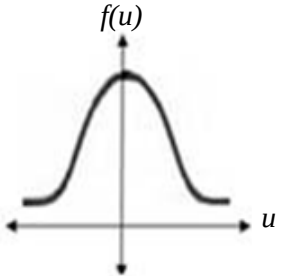
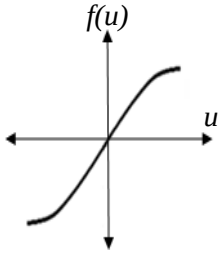
La función más habitual es de tipo lineal, y se basa en la suma ponderada de las entradas con los pesos sinápticos, que formalmente puede interpretarse como el producto escalar de los vectores de entrada y los pesos sinápticos.

2.3.5. Función de activación.

La función de activación (o función de transferencia) $f(u)$, filtra el valor de la regla de propagación, limitando el rango de amplitud de la señal de salida hacia valores finitos, por medio de una comparación con algún valor umbral para determinar la salida final de la neurona. Los tipos de función de activación más comunes que pueden ser utilizadas se muestran en la Tabla 2.1.

Tabla 2.1. Funciones de activación más comunes

Función	Fórmula	Gráfica
Lineal	$f(u) = u$	
Paso	$f(u) = \begin{cases} 1 & si \quad u \geq \theta \\ -1 & si \quad u < \theta \end{cases}$	

Rampa	$f(u) = \begin{cases} 1 & si\ u > c \\ u & c \leq u \leq -c \\ -1 & si\ u < -c \end{cases}$	
Gaussiana	$f(u) = ce^{-\frac{u}{\sigma}}$	
Sigmuidal	$f(u) = \frac{1}{1 + e^{-\frac{u}{\sigma}}}$	

2.3.6. Función de salida.

La función de salida *y = f(u)* proporciona la salida global de la neurona *i* en función del estado de activación actual. Frecuentemente la función de salida es simplemente la función identidad, de modo que el estado de activación de la neurona se considera la propia salida.

2.4. Entrenamiento.

La parte más importante de las RNA's es el esquema de entrenamiento. Éste consiste en la determinación de los valores precisos de los pesos para todas sus conexiones, que la capacite para la resolución eficiente de un problema.

Las RNA's son sistemas de entrenamiento basados en ejemplos. La capacidad de una red para resolver cierta tarea en particular estará ligada de forma fundamental al tipo de ejemplos de que dispone en el proceso de entrenamiento. Desde el punto de vista de los ejemplos, el conjunto de entrenamiento debe poseer las siguientes características [17]:

- *Ser representativo*. Los componentes del conjunto de entrenamiento deberán ser diversos. Porque si se tienen muchos más ejemplos de un tipo que del resto, la red se especializará en dicho subconjunto de datos y no será de aplicación general. Es importante que todas las regiones significativas del espacio de estados estén suficientemente representadas en el conjunto de entrenamiento.

- *Ser significativo*. Debe haber un número suficiente de ejemplos. Si el conjunto de entrenamiento es reducido, la red no será capaz de adaptar sus pesos de forma eficaz.

El proceso general de entrenamiento consiste en ir introduciendo paulatinamente todos los ejemplos de un conjunto, y modificar los pesos de las conexiones (después de introducir cada ejemplo del conjunto, o una vez introducidos todos ellos), siguiendo un determinado esquema de aprendizaje. Posteriormente se comprueba si se ha cumplido cierto criterio de convergencia; de no ser así se repite el proceso y todos los ejemplos del conjunto vuelven a ser introducidos.

Dependiendo del esquema de entrenamiento y del problema a resolver, se pueden distinguir dos tipos de aprendizaje [17]:

- Aprendizaje supervisado.
- Aprendizaje no supervisado.

2.4.1. Aprendizaje supervisado.

El aprendizaje supervisado (Figura 2.3) se caracteriza porque el proceso de entrenamiento es controlado por un agente externo (supervisor, maestro) que determina la respuesta que debería generar la red a partir de una entrada dada.

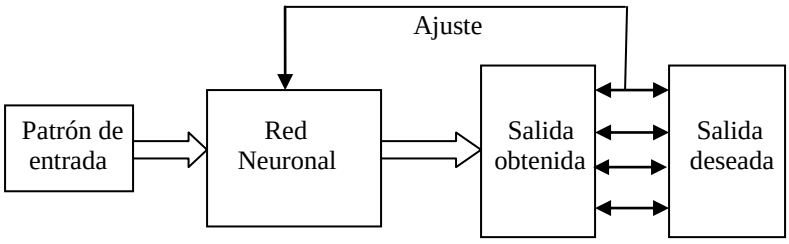


Figura 2.3. Esquema del aprendizaje supervisado.

El supervisor comprueba la salida de la red y en el caso de que ésta no coincida con la deseada, se procederá a modificar los pesos de las conexiones, con el fin de conseguir que la salida obtenida se aproxime a la deseada.

Se consideran tres formas para llevar a cabo el aprendizaje.

- Aprendizaje por corrección de error.
- Aprendizaje por refuerzo.
- Aprendizaje estocástico.

2.4.1.1. Aprendizaje por corrección de error.

Consiste en ajustar los pesos de las conexiones de la red, en función de la diferencia entre los valores deseados y los obtenidos en la salida de la red; es decir, en función del error cometido en la salida. La regla de aprendizaje simple por error es la siguiente:

$$\Delta w_{ij} = \alpha y_i (d_j - y_j) \qquad (2.3)$$

Algoritmos de este tipo de aprendizaje son el denominado regla delta o regla del error mínimo cuadrado que trata de encontrar los pesos de las conexiones que minimicen la función de error. Y el denominado regla delta generalizada o algoritmo de retropropagación del error que, como su nombre lo indica, es una generalización de la regla delta para poder aplicarlas a redes con conexiones hacia adelante en capas ocultas.

2.4.1.2. Aprendizaje por refuerzo.

Se basa en la idea de no disponer de un ejemplo completo del comportamiento deseado; es decir de no indicar durante el entrenamiento exactamente la salida que se desea que proporcione la red ante una determinada entrada. El supervisor solo le indica a la red, mediante una señal de refuerzo, si la salida obtenida se ajusta a la deseada (éxito = +1 ó fracaso = -1), y en función de ello se ajustan los pesos basándose en un mecanismo de probabilidades.

2.4.1.3. Aprendizaje estocástico.

Este aprendizaje consiste en realizar cambios aleatorios en los valores de los pesos de las conexiones de la red y evaluar su efecto a partir del objetivo deseado mediante distribuciones de probabilidad.

2.4.2. Aprendizaje no supervisado.

En este aprendizaje (también conocido como auto-supervisado), la red no recibe ninguna información por parte de un agente externo que le indique si la salda generada en respuesta a una determinada entrada es o no correcta (Figura 2.4); únicamente los datos del conjunto de entrenamiento tienen la información de los ejemplos.

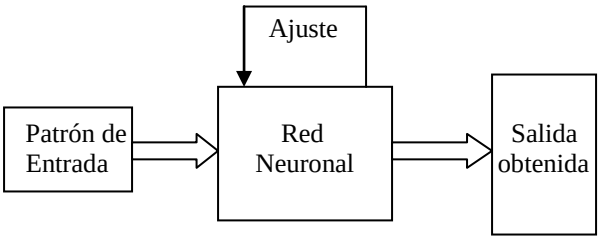


Figura 2.4. Esquema del aprendizaje no supervisado

La red tratará de determinar características de los datos del conjunto de entrenamiento (rasgos significativos, regularidades o redundancias) para modificar los valores de los pesos en sus conexiones.

2.4.2.1. Aprendizaje competitivo y cooperativo.

En este tipo de aprendizaje las neuronas compiten y/o cooperan unas con otras. Y consiste en que cuando se presente a la red cierta información de entrada, solo una o un grupo de la neuronas de salida se active (alcance su valor de respuesta máximo). Por tanto las neuronas compiten por activarse, quedando finalmente una, o un grupo, como neurona vencedora, anulando el resto, que son forzadas a sus valores de respuesta mínimo.

La competición entre neuronas se realiza en todas las capas de la red, existiendo en estas neuronas conexiones recurrentes de auto excitación y conexiones de inhibición (signo negativo) por parte de neuronas vecinas. Si el aprendizaje es cooperativo, estas conexiones con las neuronas vecinas serán de excitación (signo positivo).

El objetivo de este aprendizaje es categorizar los datos que se introducen en la red. De esta forma, las informaciones similares son clasificadas formando parte de la misma categoría y por tanto deben activar la misma neurona de salida.

2.4.3. Regla de aprendizaje de Hebb.

Se trata de uno de los modelos clásicos de aprendizaje en RNA's. Se basa en el postulado formulado por Donald O. Hebb en 1949: *"Cuando un axón de una celda A está suficientemente cerca como para conseguir excitar a una celda B y repetida o persistentemente toma parte en su activación, algún proceso de crecimiento o cambio metabólico tiene lugar en una o ambas celdas, de tal forma que la eficiencia de A, cuando la celda a activar es B, aumenta"*.

Así, según este método de aprendizaje aplicado a las RNA's, las conexiones entre las neuronas de entrada activas y las neuronas de salida activas se refuerzan durante el entrenamiento: coincidencias entre actividad de entrada y actividad de salida se intensifican. Mientras que las conexiones entre neuronas de entrada inactivas y neuronas de salida (activas o inactivas) no se refuerzan.

Este método de aprendizaje puede ser tanto supervisado como no supervisado.

Cuando es supervisado, la respuesta correcta para el dato de entrada es introducida para cada neurona de salida, y los pesos sinápticos entre las neuronas activas se incrementan, mientras que los pesos entre neuronas que no estén activas simultáneamente permanecen igual como estaban.

El problema de este método es que no tiene en cuenta la eficacia de la red. Así, aunque la red ya este entrenada y los valores de entrada generen valores de salida correctos, la regla de aprendizaje continua incrementando los pesos sinápticos entre neuronas activas.

Es por eso que, se usa una regla de aprendizaje derivada que tome en cuenta la eficacia de la red en cada momento es decir:

- Si la salida de la neurona j es la correcta, no se realizan ajustes de los pesos sinápticos.

- Si la salida de la neurona *j* es 1 pero debería ser 0, se reducen sólo los pesos de las conexiones activas según una constante C.
- Si la salida es 0 pero debería ser 1, entonces se aumentan sólo los pesos de las conexiones activas según la misma constante C.

En el aprendizaje no supervisado, lo que sucede es que un número elevado de neuronas de salida pueden activarse simultáneamente.

2.5. Capas o niveles en una red neuronal.

La distribución de neuronas dentro de la red se realiza formando niveles o capas de un número determinado de neuronas cada una (Figura 2.5). A partir de su situación dentro de la red, se pueden distinguir tres tipos de capas [47]:

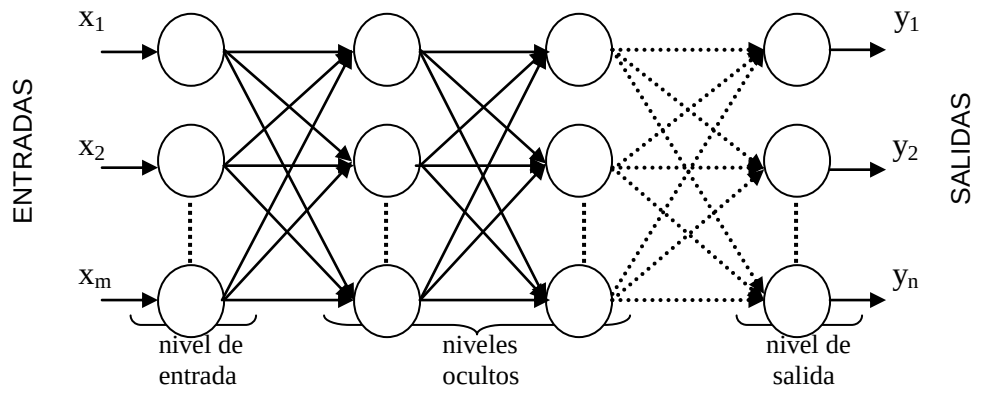


Figura 2.5. Estructura de las capas en una RNA

- *Entrada*: Es la capa que recibe directamente la información proveniente de las fuentes externas de la red.

- *Ocultas:* Son internas a la red y no tienen contacto directo con el entorno exterior. El número de niveles ocultos puede estar entre cero y un número elevado. Las neuronas de las capas ocultas pueden estar interconectadas de distintas maneras, lo que determina, junto con su número, las distintas topologías de redes neuronales.

- *Salidas:* Transfieren información de la red hacia el exterior.

2.5.1. Redes monocapa.

En las redes monocapa se establecen conexiones laterales entre las neuronas que pertenecen a la única capa que constituye la red. Este tipo de redes se utilizan típicamente en tareas relacionadas con autoasociación; por ejemplo, para regenerar información de entrada que presenta distorsiones o está incompleta.

2.5.2. Redes multicapa.

Las redes multicapa son aquellas que disponen de conjuntos de neuronas agrupadas en varios niveles o capas. Normalmente, todas las neuronas de una capa reciben señales de entrada de otra capa anterior, y envían las señales de salida una capa posterior, más cercana a la salida de la red.

2.6. Formas de conexión entre neuronas.

La conectividad entre las neuronas de una red está relacionada con la forma en que las salidas de las neuronas están canalizadas para convertirse en entradas de otras neuronas.

Si la señal de salida es conexión de entrada a la misma neurona, se denomina conexión autorrecurrente. Cuando la señal de salida es entrada de otras neuronas en una capa posterior, se describe como propagación hacia adelante. Cuando las salidas pueden ser conectadas como entradas de neuronas en niveles previos o del mismo nivel, incluyéndose ellas mismas, la red es de propagación hacia atrás.

2.7. Ventajas de las redes neuronales artificiales.

Las redes neuronales artificiales ofrecen ventajas de procesamiento específicas, las cuales la convierten en la tecnología de elección en múltiples áreas de aplicación [14]. Estas ventajas son:

- Procesamiento paralelo.
- Aprendizaje adaptivo.
- Auto-organización.
- Tolerancia a fallos a través de codificación de la información redundante.
- Operación en tiempo real.
- Fácil inserción dentro de la tecnología existente.

2.7.1. Procesamiento paralelo.

Es una característica innata de las RNA´s, ya que hay un paralelismo inherente en la forma en que operan los elementos de proceso que las constituyen. La estructura y modo de operación las hace especialmente adecuadas para el procesamiento paralelo real mediante multiprocesadores.

2.7.2. Aprendizaje adaptivo.

Es la propiedad de aprender a adaptarse, mediante la modificación de sus pesos sinápticos, a las nuevas condiciones del entorno. Esto lo logran por medio de un entrenamiento con ejemplos ilustrativos. Porque las redes neuronales pueden aprender a discriminar patrones basados en ejemplos y entrenamiento.

El diseñador solamente se encarga de la arquitectura apropiada, y no como la RNA aprenderá a discriminar entre las posibles elecciones; sin embargo, si es necesario el desarrollo de un buen algoritmo de entrenamiento, que le proporcione la capacidad de discriminar correctamente.

Algunas redes continúan su aprendizaje aún después de terminar su periodo inicial de entrenamiento.

2.7.3. Auto-organización.

Las RNA´s usan su capacidad de aprendizaje adaptivo para auto-organizar la información que reciben durante el entrenamiento y/o la operación. Esto es, crean representaciones de distintas características en los datos presentados para llevar a cabo un objetivo específico.

La auto-organización provoca la generalización, lo cual les permite responder apropiadamente cuando se presentan datos o situaciones a los que no habían sido expuestas anteriormente. Esta característica es muy útil cuando la información de entrada es poco clara, con ruido o está incompleta.

2.7.4. Tolerancia a fallos.

La razón por la que las RNA´s sean tolerantes a fallos es que distribuyen y almacenan la información aprendida, de forma redundante, en las conexiones sinápticas de las neuronas. Hay dos aspectos distintos de tolerancia de fallos:

1ª Tolerancia a fallos en los datos. Las redes neuronales pueden aprender a reconocer patrones con ruido, distorsiones o incompletos.

2ª Tolerancia a fallos a daño con ella misma. Ellas pueden seguir realizando su función (con cierta degradación) aunque se destruya parte de la red.

2.7.5. Operación en tiempo real.

Una de las mayores prioridades en las áreas de aplicación, es la necesidad de realizar grandes procesos con los datos de forma muy rápida. Debido a su estructura y su capacidad de procesamiento paralelo entre los elementos de proceso, las RNA´s pueden operar en un entorno de tiempo real con una mínima variación en los pesos de las conexiones. Por lo tanto, de todos los métodos posibles, las RNA’s son la mejor alternativa para reconocimiento y clasificación de patrones en tiempo real.

2.7.6. Fácil inserción en la tecnología existente.

Una sola red puede ser entrenada, comprobada y verificada para desarrollar tareas específicas. Así puede ser trasladada a una implementación en hardware de bajo costo, para su inserción dentro de sistemas existentes. De esta manera, las redes neuronales se pueden utilizar para mejorar sistemas de forma incremental.

2.8. Aplicaciones.

Existen diferentes tipos de RNA, cada una de las cuales tiene una aplicación particular más apropiada sobre tecnologías convencionales existentes [47], algunas aplicaciones son:

- ***Biología:***

  Aprender más acerca del cerebro y otros sistemas.

  Obtención de modelos de la retina.

- ***Empresa:***

  Evaluación de probabilidad de formaciones geológicas y petrolíferas.

  Identificación de candidatos para posiciones específicas.

  Explotación de bases de datos.

  Optimización de plazas y horarios en líneas de vuelo.

  Reconocimiento de caracteres escritos.

  Modelado de sistemas para automatización y control.

- ***Finanzas:***

  Previsión de la evolución de los precios.

  Valoración del riesgo de los créditos.

  Identificación de falsificaciones.

  Interpretación de firmas.

- ***Medio ambiente***:

  Analizar tendencias y patrones.

Previsión del tiempo.

- ***Manufacturación:***

  Robots automatizados y sistemas de control (visión artificial y censores de presión, temperatura, gas, etc.).

  Control de producción en líneas de procesos.

  Inspección de la calidad.

- ***Militares:***

  Clasificación de las señales de radar.

  Creación de armas inteligentes.

  Optimización del uso de recursos escasos.

  Reconocimiento y seguimiento en el tiro al blanco.

- ***Medicina:***

  Analizadores del habla para ayudar en la audición de sordos profundos.

  Diagnóstico y tratamiento a partir de síntomas y/o de datos analíticos (electrocardiograma, encefalogramas, análisis sanguíneo).

  Monitorización en cirugías.

  Predicción de reacciones adversas en los medicamentos.

  Entendimiento de la causa de los ataques cardíacos.

CAPÍTULO 3.

PROCESAMIENTO Y CLASIFICACIÓN DE IMÁGENES

Un área de la ciencia e ingeniería que se ha desarrollado rápidamente en los últimos años es el procesamiento digital de señales, tales como el de voz, audio e imágenes [1].

El procesamiento digital de imágenes es un conjunto de técnicas aplicadas a la representación digital de una imagen que conforma una escena, con el objeto de destacar algunos elementos de interés, que facilite su posterior análisis por parte de un sistema de visión computarizado [48].

El procesamiento digital de imágenes consiste en hacer un mapeo de una imagen a puntos bien definidos discretamente, a los cuales se les asigna un par de coordenadas y un valor de intensidad. La alteración de los valores de intensidad por medio de una computadora permite efectuar con gran facilidad operaciones de realce de análisis de la imagen [49].

Las técnicas de procesamiento de imágenes son aplicadas cuando resulta necesario:

- Modificar una imagen para mejorar su apariencia.

- Destacar algún aspecto de la información contenida en la misma.

- Medir, contrastar o clasificar algún elemento en la imagen.

- Se requiere combinar imágenes o porciones de las mismas para reorganizar su contenido.

3.1. Elementos de la imagen digital.

Pixel: Es el acrónimo del inglés *picture element*. Es un elemento de la imagen digital, el cual tiene una posición asociada [1].

Imagen: es una función de intensidad bidimensional $f(x, y)$, donde x e y son las coordenadas espaciales, y el valor de f en algún par de coordenadas (x, y) es proporcional a la intensidad o nivel de gris de la imagen en ese punto, que está comprendido en el intervalo de [0, 255]. Cuando x, y y f son todos cantidades discretas, se conoce como imagen digital [1].

Contraste: es la diferencia de brillos entre dos pixeles vecinos [1].

Resolución: Representa la calidad de la imagen, ya que describe el número total de pixeles en la imagen [1].

3.2. Sistemas para la percepción de imágenes.

Existen métodos de procesamiento de imágenes construidos sobre bases matemáticas y probabilísticas. En la vida real hay problemas tales como identificación de huellas digitales, reconocimiento de caracteres ópticos, clasificación de objetos, que requieren el diseño y construcción de sistemas para dar solución a estos. El reconocimiento de patrones es un ejemplo de estos sistemas.

Los sistemas de reconocimiento de patrones son capaces de tomar datos para procesar y realizar ellos una acción en base a la categoría del patrón. Para la solución de problemas como reconocimiento visual y de voz, algunos modelos de propósito particular basan sus algoritmos y diseño de hardware en cómo son resueltos naturalmente [1].

La clasificación de patrones difiere del procesamiento de imágenes, ya que en que ésta ultima la entrada es una imagen y la salida es una imagen, además que frecuentemente incluyen pasos de rotación, mejoramiento del contraste y otras transformaciones las cuales preservan toda la información original [1]. Sin embargo el procesamiento de imágenes es necesario en el reconocimiento de patrones visuales para poder realizar una clasificación.

3.2.1. Estructura de un sistema de clasificación.

Aunque en los sistemas de reconocimiento de patrones surgen un gran número de transformaciones altamente complejas, y algunas son de dominio específico, se pueden distinguir tres operaciones diferentes: de preprocesamiento, extracción de propiedades y

clasificación. Los componentes de un sistema prototipo para el reconocimiento de patrones se muestra en el siguiente diagrama (Figura 3.1)

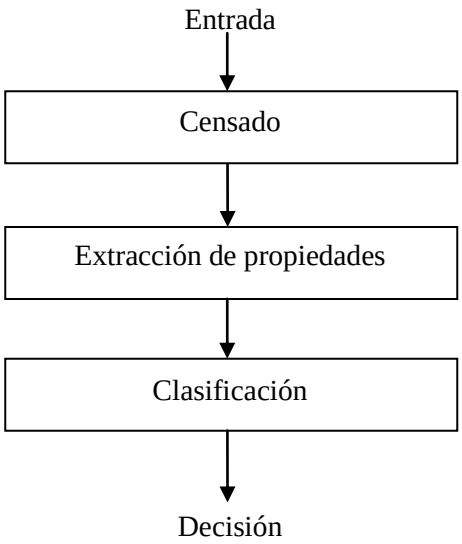


Figura 3.1. Esquema prototipo de un sistema de reconocimiento de patrones [1]

Para entender el problema de diseñar tal sistema, se debe entender el problema de las operaciones que cada componente en turno debe resolver.

3.2.1.1. Censado.

En esta parte, la entrada del sistema es captada por un sensor (p. ej. una cámara web) que convierte las entradas físicas en señales de datos, éstas son pre-procesadas para simplificar operaciones subsecuentes sin pérdida relevante de información. Frecuentemente para este propósito se usa alguna clase de transductor, el cual es un arreglo de cámaras o micrófonos. La dificultad del problema bien puede depender en las características y limitaciones del transductor –estos son ancho de banda, resolución, sensibilidad, distorsión, relación señal a ruido, latencia, etc.

3.2.1.2. Extracción de propiedades y segmentación.

La extracción de propiedades es un proceso arbitrario, ya que las características del objeto son tomadas por un extractor, quien produce valores característicos y genera un patrón, que es usado para la clasificación. El número de características es virtualmente siempre

elegido para ser menor que el total necesario para describir completamente el objeto de interés, dejando una perdida en la información. Esto da la idea de buscar características distinguibles que sean invariantes para transformaciones irrelevantes de la entrada, como rotaciones, traslaciones o escalamientos.

El principal objetivo de la extracción de propiedades es, que los valores medidos para caracterizar un objeto a ser reconocido, sean muy similares para objetos en la misma categoría y muy diferentes para objetos otras en categorías. Este acto se le conoce como memoria asociativa, es decir, el sistema toma un patrón y emite otros patrones los cuales son representativos de un grupo general de patrones. Esto provoca la reducción de la información hasta cierto punto

La segmentación consiste en aislar los objetos censados del fondo o de otros objetos. El problema que puede surgir aquí, es el de reconocimiento o agrupamiento de varias partes de un objeto compuesto. Esto aparece aunque el sistema trate de incorporar la mayor cantidad de entradas para la categorización.

3.2.1.3. Clasificación.

La tarea propia de un clasificador es el uso del vector de características, proporcionado por el extractor de propiedades, para asignar al objeto a una categoría. Este paso representa incluso más pérdida de datos, ya que se reduce la información original a un poco de bits que representen la categoría elegida. Porque la representación de una clasificación perfecta es prácticamente imposible, una tarea más general es determinar la probabilidad para cada categoría de todas las categorías posibles.

3.2.2. Ruido.

El grado de dificultad del problema de clasificación depende de la variabilidad en los valores característicos para objetos en la misma categoría, relativa a las diferencias entre valores característicos para objetos en categorías diferentes. La variabilidad de estos valores puede ser debida a la complejidad y al ruido.

El ruido en términos generales es cualquier propiedad aleatoria ajena al patrón censado, la cual no es verdadera según el modelo. Toda decisión no trivial y problemas de reconocimiento de patrones envuelven ruido de alguna forma. Un problema que surge en

la práctica es que no siempre es posible determinar los valores de todas las características para una entrada en particular.

3.3. Diseño del ciclo para un sistema de clasificación.

El diseño de un sistema de reconocimiento de patrones usualmente conlleva la repetición de un número de actividades diferentes (Figura 3.2) [1].

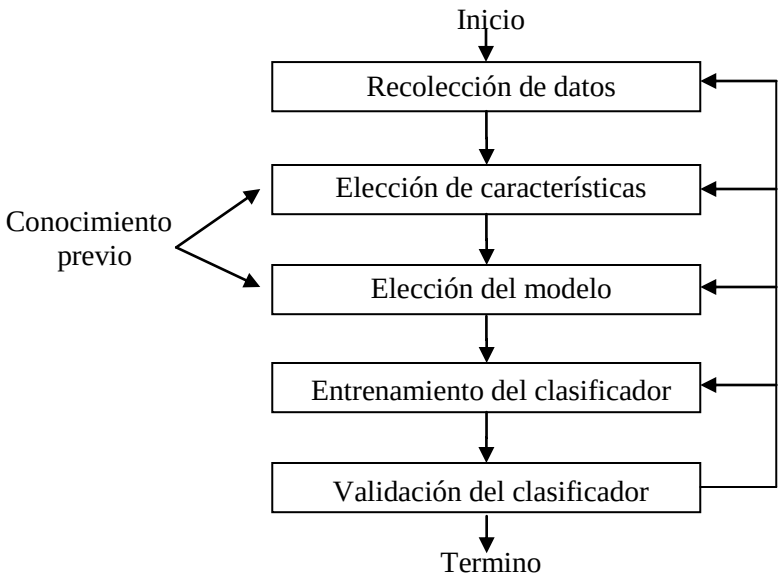


Figura 3.2. Diagrama del diseño del ciclo para un sistema de reconocimiento de patrones.

Aunque esta descripción acentúa de una manera ascendente el flujo de los datos, algunos sistemas emplean la retroalimentación desde los niveles más altos hacia atrás a los niveles más bajos.

3.3.1. Recolección de datos.

La recolección de datos en el desarrollo del sistema, puede informar en gran parte el costo del sistema, ya que hace posible un estudio de viabilidad preliminar con un pequeño conjunto de ejemplos típicos, pero para asegurar una buena representación en los campos del sistema, usualmente serán necesarios muchos más datos.

3.3.2. Elección de propiedades.

La elección de las propiedades distinguibles es un paso crítico del diseño y depende sobre el dominio del problema. Hay que encontrar propiedades simples de extraer, transformaciones invariantes o irrelevantes, insensibles al ruido y ayuden para la discriminación de patrones en categorías diferentes.

Tener acceso a los datos de ejemplo previamente será válido para la elección del conjunto de características. Este conocimiento previo de los datos servirá en el diseño del sistema para sugerir características prometedoras que acerquen a la forma del modelo de la categoría o especificar atributos en los patrones.

3.3.3. Elección del modelo.

Realmente nunca se tendrá conocimiento seguro si la representación del modelo elegido es adecuado o no. Como diseñadores hay que tratar de reducir la aleatoriedad, los procesos tediosos y el error, para que el modelo hipotético no difiera significativamente del verdadero según nuestros patrones, para así encontrar el modelo más adecuado.

3.3.4. Entrenamiento del clasificador.

El entrenamiento es el proceso de usar datos para determinar el tipo de clasificador. Diferentes métodos no universales han sido encontrados para resolver problemas de reconocimiento de patrones. La experiencia de las últimas décadas ha sido que los métodos más efectivos para el desarrollo de clasificadores envuelven entrenamiento por medio de ejemplos de patrones.

3.3.5. Validación del clasificador.

La evaluación es importante para medir la representación del sistema y para identificar lo necesario para mejorar sus componentes. Un sistema demasiado entrenado puede permitir una clasificación perfecta de los ejemplos de entrenamiento, pero es poco probable dar una buena representación para nuevos patrones. Esta situación se conoce como sobreentrenamiento. Una de las principales áreas de investigación en clasificación de patrones es determinar cómo ajustar la complejidad del modelo

CAPÍTULO 4.

CLASIFICADORES NEURONALES

En este capítulo se analiza la aplicación de las redes neuronales para procesamiento de información como son los clasificadores neuronales. Presentando de manera general la estructura y funcionamiento de los clasificadores neuronales RTC, RSC, PCN y LIRA, que dan solución a problemas de reconocimiento de imágenes de diferentes clases, por ejemplo, cifras escritas, rostros humanos y tareas de micromecánica [50]. A continuación se presenta el primero de ellos.

4.1. Clasificador de Umbrales Aleatorios, RTC.

El Clasificador de Umbrales Aleatorios (*Random Threshold Classifier*, RTC) [51], es un sistema propuesto para resolver problemas relacionados con el reconocimiento de imágenes, ya que posee un alto rendimiento en las etapas de entrenamiento y en la de procesamiento.

La estructura del RTC (Figura 4.1), consiste en una sola capa de entrenamiento, y otras capas compuestas de neuronas binarias, con conexiones no modificables, que permitan transformaciones no lineales de parámetros de entrada espaciales a parámetros espaciales binarios de altas dimensiones.

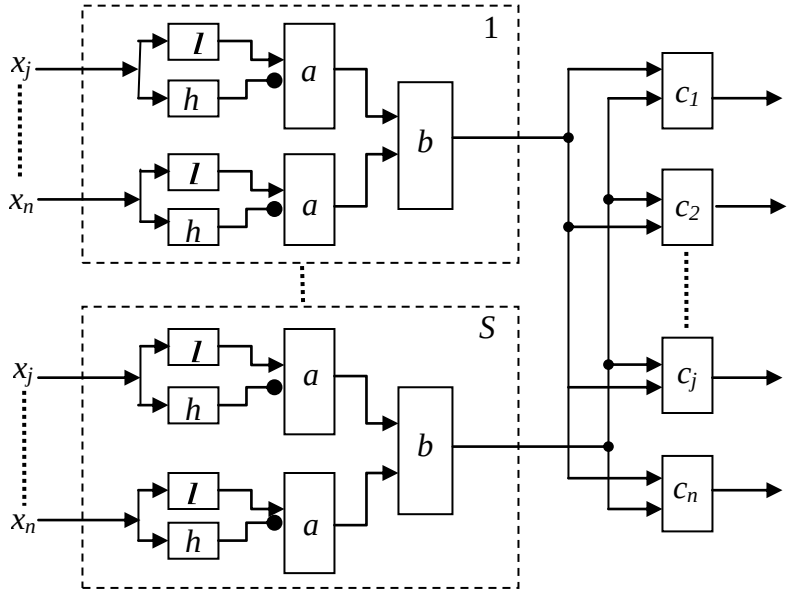


Figura 4.1. Estructura del clasificador RTC [50].

La red está formada por S grupos similares con una salida neuronal en cada grupo ($b_{1,\ldots,}b_s$). Las entradas de cada grupo están formadas por un juego de características ($x_{1,\ldots,}x_n$) que describen a la imagen. Cada característica x se conecta a dos neuronas, hi_j y l_{ij}, donde i(i=1,2,..,n) representa el número de característica, y j (j = 1, 2,..,n) representa el número de grupo neuronal.

El valor umbral de l_{ij} es menor que el umbral de h_{ij}, y son fijados por medio de un proceso aleatorio. La salida de la neurona l_{ij} es conectada a la entrada excitada de la siguiente neurona a_{ij}, y la salida de la neurona l_{ij} es conectada a una salida inhibida de una neurona a_{ij}.

La señal de salida de una neurona a_{ij}, solo ocurre cuando la señal de una entrada excitada es igual a 1 y cuando la inhibida es 0. Todas las salidas de las neuronas a, en un grupo j, se conectan a las entradas excitadas de una neurona b_j, que representa la salida de todo el grupo neuronal.

El valor umbral de la neurona b_j es igual al número de características; por ejemplo, para la neurona b_j la señal de salida ocurre cuando son excitadas todas las neuronas a_i en el grupo. Las salidas de cada grupo son conectadas a todas las neuronas del clasificador c mediante conexiones entrenables.

Entonces las neuronas c_i en la capa de respuesta tienen asociado un peso w_i, el cual se ve modificado por las neuronas b_j durante la etapa de entrenamiento. Cada neurona en esta capa representa una respuesta de clasificación del sistema, y se elige aquella que tenga el valor de excitación más alto.

En la fase de entrenamiento, el objetivo de la red es reconocer correctamente el mayor número de patrones. Por eso en esta etapa, se trata de disminuir todos los pesos de las conexiones de la clase incorrecta y aumentar los de la clase correcta, lo cual conlleva a un entrenamiento supervisado. Se utiliza la regla de Hebb para las conexiones entrenables.

El entrenamiento termina cuando se cumpla cualquiera de las dos condiciones:

1ª El número de errores es menor al prefijado.

2ª Se llegue al último de un número fijo de iteraciones de entrenamiento.

Este clasificador ya ha sido probado para diversos problemas de reconocimiento como son de texturas naturales, escritura, palabras y señales, obteniendo buenos resultados.

4.2. Clasificador de Espacio Aleatorio, RSC.

El clasificador neuronal con espacio aleatorio (Random Subspace Classifier, RSC) se ha desarrollado a partir del clasificador neuronal RTC. Su característica principal es que no todos los parámetros $(x_1,\ldots,x_n)$ participan en los procesos de entrenamiento y reconocimiento de imágenes, sino que, solo eligen aleatoriamente un subconjunto de todo el conjunto. Este procedimiento permite disminuir el tamaño del espacio paramétrico y acelerar los procesos de tratamiento de la información.

4.3. Clasificador de Permutación de Códigos, PCNC.

El clasificador PCNC (*Permutation Coding Neural Classifier*) [52], trabaja con imágenes en escala de grises y está basado en la estructura genérica del paradigma APNN [53] [54]. Ha sido probado en la tarea de reconocimiento de caracteres escritos con la base de datos MNIST con tasa de error de 0.37% [4], reconocimiento de rostros con la base de datos ORL mostrando una tasa promedio de 0.1% y reconocimiento de microobjetos [55].

Su estructura consta de tres partes que trabajan de forma serial: un extractor de propiedades, un codificador y un clasificador (Figura 4.2).

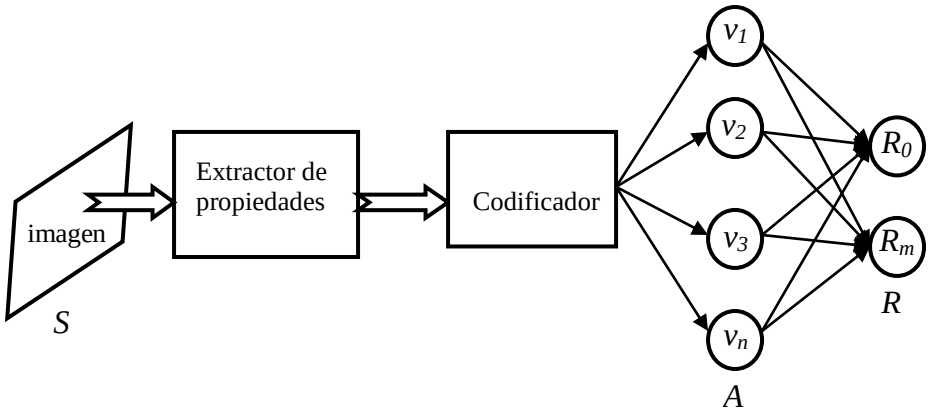


Figura 4.2. Estructura del clasificador PCNC [50].

A continuación se explica cada una de las partes de la estructura.

Extractor de propiedades.

A partir de una imagen en escala de grises, el extractor de propiedades selecciona una serie de puntos específicos que representen las propiedades de la imagen para su clasificación. Los métodos para definir dichos puntos son: por umbral de brillo y por extracción de contornos. Para el primer método se selecciona un umbral de brillo determinado B y los puntos específicos de la imagen serán todos aquellos pixeles cuyo brillo b_{ij} sea mayor que B. En el segundo método se utiliza un algoritmo de extracción de bordes sobre la imagen.

Todas las propiedades extraídas de todos los puntos específicos definidos son entregadas al codificador.

Codificador.

El codificador transforma las propiedades dadas por el extractor de propiedades a un vector binario V y a uno adicional U para cada propiedad extraída, llamado mascara de la propiedad F_k. Se definen S propiedades, cada una mediante p puntos positivos y n puntos negativos aleatoriamente distribuidos sobre un rectángulo $w\ x\ h$ pixeles. Para cada punto se prueba la existencia de las S propiedades, entonces el vector U se transforma al vector auxiliar U* por cada propiedad localizada en la imagen. Esta transformación se hace mediante las permutaciones del vector U. El número de permutaciones depende de la localización de la propiedad en la imagen.

Una vez calculados todos los vectores U_r* de todas las propiedades detectadas en la imagen se crea el vector código V. Para reconocer alguna pieza en una imagen se es necesario utilizar una combinación de propiedades, es decir, combinar la existencia de ciertas propiedades con la ausencia de otras. Para este propósito se utiliza el enrarecimiento dependiente de contexto CDT, que prueba cada uno de los valores correctos del vector V , obteniendo el vector V' de dimensión N que representa el código de la imagen presentada originalmente al extractor de propiedades. Este resultado permite pasar al clasificador neuronal.

Clasificador neuronal.

Es necesario que en el espacio paramétrico, las clases tengan separabilidad lineal. Es por eso que de las etapas anteriores (extractor de propiedades y codificador) se obtiene la separabilidad lineal y de un espacio paramétrico de dimensión $W\ x\ H$ se convierte a uno de dimensión N, donde W y H son el ancho y alto de las imágenes que va a procesar el clasificador PCNC, y N es igual a la dimensión del vector V'.

4.4. Clasificador LIRA.

El clasificador neuronal LIRA (*Limited Receptive Area*) está basado en los principios del PERCEPTRON de Rosenblat [2], [3]. Este clasificador ha sido probado en tareas de microensamble y reconocimiento de dígitos escritos a mano mostrando buenos resultados.

Este clasificador y el PCNC son similares en su método de clasificación, pero difieren en los procesos de extracción de propiedades y codificación de la imagen que se va a clasificar [50].

El clasificador neuronal LIRA (Figura 4.3) consiste de cuatro capas: capa de entrada *S*, capa intermedia *I*, capa asociativa *A* y capa de salida *R*.

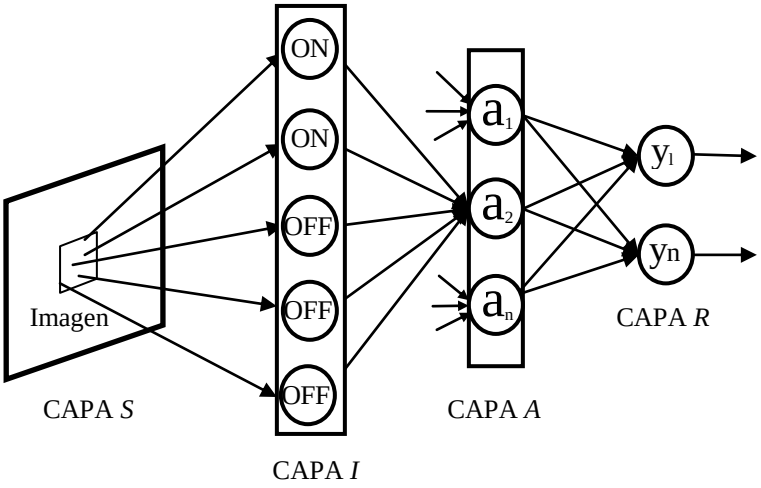


Figura 4.3. Estructura del clasificador LIRA.

Las neuronas de la capa de entrada *S* corresponden al valor de un pixel en la imagen y tienen salidas en el rango [0, 255].

La capa intermedia *I* contiene dos tipos de neuronas: neuronas ON y neuronas OFF, que tienen dos estados como valores de salida {0, 1}. Esta capa conecta a la capa *S* y la capa *A* por medio de un del siguiente procedimiento:

Para cada neurona a_k de la capa asociativa, se selecciona aleatoriamente un área rectangular, definida como ventana de tamaño $h \times w$ neuronas sobre la capa *S*. De esta ventana se eligen aleatoriamente m puntos que son divididos en p puntos positivos y n puntos negativos que se conectan cada uno a una neurona *ON y OFF* de la capa *I*, respectivamente, los cuales tendrán relacionado un umbral T_{mk} en el rango [0, 255]. Este grupo de m neuronas de la capa intermedia se conecta a cada neurona a_k.

La capa asociativa *A* corresponde a un extractor de propiedades que contiene neuronas con dos estados de salida {0, 1} que corresponden a si la característica está presente o ausente en la imagen. Todas las salidas de las neuronas de esta capa se conectan a todas las entradas de las neuronas de la capa *R*, cuyas conexiones son entrenables. Una neurona asociativa tiene salida 1(estado activo) si todas las neuronas del grupo conectado a ella están en estado activo, de lo contrario la salida es 0.

En la capa de salida *R* el número de neuronas, con función de activación de tipo lineal, corresponden al número de clases por reconocer. La neurona con excitación máxima en esta capa, es seleccionada como ganadora.

La principal desventaja del PCNC sobre el clasificador LIRA consiste en que los requerimientos de cómputo son mayores. Ello se debe a la necesidad del PCNC de realizar gran cantidad de cálculos con vectores durante el proceso de codificación de una imagen [50].

En el siguiente capítulo se describirá más a detalle la estructura del clasificador neuronal

CAPÍTULO 5.

PARADIGMA DEL CLASIFICADOR NEURONAL LIRA Y SU ESTRUCTURA

Existen dos tipos de sistemas de reconocimiento de imágenes. El primer tipo contiene un extractor de características y un clasificador (Figura 5.1). El extractor de características transforma la imagen a un vector paramétrico. Cada componente de este vector corresponde a una característica específica, estás son usadas para resolver diferentes problemas de reconocimiento.

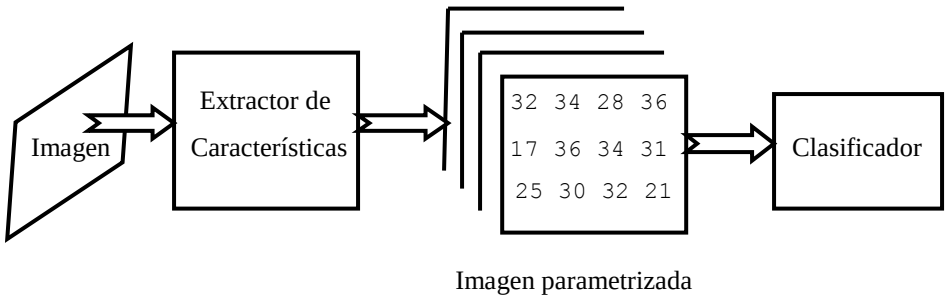


Figura 5.1. Sistema de reconocimiento compuesto por un extractor de características y un clasificador.

El segundo tipo de sistema de reconocimiento de imágenes contiene sólo el clasificador. Los sistemas para el reconocimiento de imágenes tales como el PERCEPTRON de Rosenblatt [2], [3], NEOCOGNITRON de Fukushima [34], *LeNet* de Yan LeCun [56], *MWC* de Hoque [57] entre otros, pertenecen a este tipo de sistema. Estos sistemas se caracterizan porque incorporan el extractor de características en su estructura interna.

El clasificador neuronal *Limited Receptive Area* (LIRA) perteneciente al segundo tipo de sistema, fue desarrollado por los investigadores Ernst Kussul, Tatiana Baidyk y Dimitri Rachkovskij está basado en el PERCEPTRON de Rosenblatt [53], [54].

Existen dos variantes principales del clasificador neuronal LIRA, aunque son similares en estructura, la diferencia radica de acuerdo a cómo es presentada una imagen en la entrada del clasificador y en el codificador. Si la imagen es presentada en formato binario, es decir, únicamente con valores de entrada negro y blanco se denomina LIRA binario*;* si la imagen de entrada se presenta con valores de gris se denomina LIRA *Grayscale*.

El clasificador neuronal LIRA ha sido probado en tareas de reconocimiento de dígitos escritos a mano [4], clasificación de imágenes de microtornillos [5], [6], tareas de ensamble de microdispositivos [7], y ha permitido obtener buenos resultados.

La estructura general para el clasificador LIRA *Grayscale* se describe a continuación.

5.1. Estructura.

La estructura del clasificador neuronal LIRA *Grayscale* es similar a la estructura del PERCEPTRON de 3 capas de Rosenblatt [50]. La principal modificación está en la conexión al tamaño del área de la imagen, la cual es usada para la extracción de cada característica. Esta modificación es necesaria para obtener las características locales, las cuales describen mejor a la imagen que las características globales del PERCEPTRON de Rosenblatt.

El clasificador neuronal LIRA *Grayscale* consiste de cuatro capas: de entrada o de sensor (*S*), intermedia (*I*), asociativa (*A*) y de salida o de reacción (*R*). Su estructura y conexión entre capas se muestra en la figura 5.2.

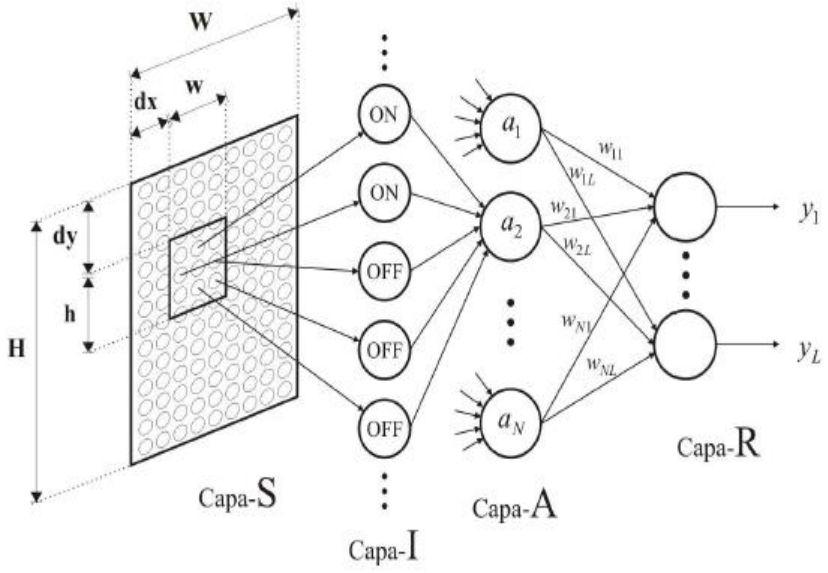


Figura 5.2. Arquitectura del clasificador LIRA *Grayscale.*

En la capa de entrada *S* se presenta la imagen que se desea clasificar. Ésta capa contiene $W \, x \, H$ neuronas, donde W es el ancho y H es el alto de la imagen a ser clasificada. Las neuronas de ésta capa corresponden al valor de un pixel de la imagen y los valores que tienen en las salidas están en el rango [0, 255], donde 0 indica el valor mínimo de brillo (negro) y 255 el valor máximo de brillo (blanco).

La capa *S* está conectada a la capa *A* a través de la capa *I* por medio de un proceso aleatorio que se describirá posteriormente. Estas conexiones serán no entrenables, es decir, no pueden modificar el valor de su peso durante el proceso de entrenamiento.

La capa intermedia *I* contiene N grupos de neuronas, donde N es el número total de neuronas de la capa asociativa *A*. La capa *I* tiene dos tipos de neuronas: neuronas ON que se activan con la presencia de un estimulo y neuronas OFF que se activan con la ausencia del estimulo. Estas neuronas tienen dos estados como valores de salida {0, 1}. Las conexiones entre las capas *A* e *I* tampoco son entrenables.

La capa asociativa *A* corresponde a un extractor de características que contiene neuronas con dos estados de salida igual a 1 (estado activo) ó 0 (estado no activo). Todas las neuronas de la capa *A* están conectadas con todas las neuronas de la capa *R*, mediante conexiones cuyos pesos podrán ser modificados durante el proceso de entrenamiento. En el valor de todos los pesos de estas conexiones se ubica la memoria del clasificador neuronal.

La capa de salida *R* se compone de neuronas con función de activación de tipo lineal. El número de neuronas en esta capa corresponde al número de clases por reconocer.

5.2. Conexiones entre capas y activación de neuronas.

El procedimiento para conectar la capa de entrada *S* con la capa asociativa *A* a través de la capa intermedia *I* es el siguiente. Sea N el número de total de neuronas asociativas. Para cada neurona asociativa a_k, donde $k = 1,...,N$, se selecciona aleatoriamente un área rectangular en la capa *S* (definida como ventana) de $h \, x \, w$ neuronas (ver figura 5.2). Los valores de w y h son seleccionados experimentalmente en el rango $[0, W]$ y $[0, H]$. Los valores para los parámetros dx y dy son seleccionados aleatoriamente en el intervalo $[0,W\text{-}w]$ y $[0,H\text{-}h]$, respectivamente para obtener las coordenadas de la esquina superior izquierda de la ventana.

De la ventana resultante se eligen aleatoriamente m puntos (neuronas) que serán divididos de manera aleatoria en dos conjuntos, uno de p puntos “positivos” y otro de n puntos “negativos” donde $p + n = m$. Cada punto positivo y cada punto negativo será conectado a una neurona ON y OFF de la capa *I*, respectivamente, y cada uno de estos puntos tendrá

relacionado un umbral T_{mk} seleccionado de manera aleatoria del rango [0, 255]. Este grupo de m neuronas en la capa intermedia se conectará a la neurona a_k de la capa *A*.

En el proceso de conexión entre las capas *S, I* y *A,* los valores de los parámetros se mantienen fijos antes de iniciar el entrenamiento del clasificador neuronal, excepto los valores de los pesos sinápticos entrenables de las conexiones entre la capa *A* y la capa *R*.

Para la activación de las neuronas se procede de la siguiente manera:

Sea x_{ij} una neurona de entrada de la capa *S,* la salida de una neurona ON será igual a 1 (estado activo) si su valor de entrada x_{ij} es mayor o igual al umbral T_{pk}; y será 0 en otro caso, esto es:

$$\varphi_{ON}(x_{ij}) = \begin{cases} 1, & x_{ij} \geq T_{pk} \\ 0, & x_{ij} < T_{pk} \end{cases} \qquad (5.1)$$

La salida de una neurona OFF será igual a 1 (estado activo) si su valor de entrada x_{ij} es menor o igual al umbral T_{nk}; y será igual a 0 en otro caso, esto es:

$$\varphi_{OFF}(x_{ij}) = \begin{cases} 1, & x_{ij} \leq T_{nk} \\ 0, & x_{ij} > T_{nk} \end{cases} \qquad (5.2)$$

Una neurona asociativa tendrá salida igual a 1 (estado activo) si todas las neuronas ON y OFF del grupo conectado a ella están en estado activo, de cualquier otra manera su salida será 0. Cada neurona asociativa actúa como una característica de la imagen, cuya salida indica si dicha característica está presente o ausente en la imagen.

5.3. Proceso de entrenamiento.

Una vez constituido el clasificador neuronal LIRA *Grayscale*, junto con los conjuntos de imágenes de entrenamiento y prueba, comenzará una etapa durante la cual se llevarán a cabo experimentos preliminares destinados a encontrar los valores óptimos para los parámetros del sistema que no serán modificables durante la etapa de aprendizaje de la red neuronal. Nos referimos al número de neuronas asociativas, dimensiones de ventana y la excitación adicional de la neurona ganadora, entre otros.

Al término de la mencionada fase, comenzará la de entrenamiento del sistema, la cual requerirá supervisión. En esta etapa se realizaran los ajustes de los parámetros internos de la red neuronal propuesta, de manera que el clasificador reconozca correctamente todas las imágenes que le sean presentadas durante el proceso. Por tanto el clasificador neuronal LIRA *Grayscale* emplea un proceso de entrenamiento supervisado.

El conjunto de imágenes de esta fase será distinto al utilizado durante la etapa de prueba, es decir, se repetirán los pasos descritos anteriormente para crear nuevos conjuntos de entrenamiento y de prueba.

Antes de iniciar el entrenamiento, los pesos de las conexiones entre neuronas de las capas *A* y *R* son inicializados a cero. El proceso de aprendizaje consiste en tres fases que se describen a continuación:

5.3.1. Fase inicial.

El entrenamiento inicia con la presentación de la imagen a la capa de entrada *S* de la red neuronal. Las características de la imagen son extraídas y codificadas a través de la capa *I* y la capa *A*, respectivamente.

Las salidas de las neuronas de la capa *R* son adecuadas mediante la siguiente fórmula:

$$y_i = \sum_{k=1}^{N} w_{ki} \cdot a_k \qquad (5.3)$$

donde y_i es la salida (excitación) de la i-ésima neurona de la capa *R*;
a_k es la salida (0 ó 1) de la k-ésima neurona de la capa *A*;
w_{ij} es el peso de la conexión entre la k-ésima neuronas de la capa *A* y la i-ésima neurona de la capa *R*.

5.3.2. Método de selección del ganador.

En este proceso, una vez calculada la salida de todas las neuronas de la capa *R*, se elegirá como clase reconocida el índice de la neurona de salida de la capa R que posea el valor más alto de excitación, es decir, el índice de la neurona que indique si el vector de entrada presentada a la red neuronal pertenece o no a la clase reconocida.

$$y_g = max_i(E_i) \qquad (5.4)$$

donde y_g es la salida de la neurona ganadora de la capa R;
E_i es la neurona con la más alta excitación.

5.3.3. Adaptación de pesos.

Una vez que se ha obtenido la neurona ganadora (clase reconocida por la red neuronal), si la neurona r correspondiente a la clase real es igual a la neurona g correspondiente a la clase ganadora, esto es, si $r = g$ entonces no se modifica ningún peso en las conexiones; pero si $r \neq g$, entonces:

$$\forall k,\ w_{kr}(t+1) = w_{kr}(t) + a_k \quad (5.5)$$

$$\forall k,\ w_{kg}(t+1) = w_{kg}(t) - a_k \quad (5.6)$$

$$si\ \left(w_{kg}(t+1) < 0\right) \rightarrow w_{kg}(t+1) = 0 \quad (5.7)$$

donde $w_{ki}(t)$ es el peso de la conexión entre la k-ésima neurona de la capa *A* y la i-ésima neurona de la capa *R* antes del refuerzo, $w_{ki}(t+1)$ es el peso de la misma conexión después del refuerzo, a_k es el valor de salida de la k-ésima neurona de la capa *A*.

Este proceso será repetido hasta alcanzar un criterio de convergencia, en este caso hasta llegar al número máximo de ciclos de entrenamiento.

5.4. Proceso de validación.

Una vez concluida la etapa de entrenamiento, se realiza el proceso de validación del clasificador neuronal. Para poder determinar si la red produce salidas adecuadas, de la base de imágenes se toma el conjunto de validación, que se utiliza para medir la eficacia de la red para resolver el problema. Se emplean datos que no han sido utilizados para su entrenamiento. Si el error sobre el conjunto de validación es pequeño, entonces quedará garantizada la capacidad de generalización de la red.

Para que este proceso sea eficaz el conjunto de validación debe tener las siguientes características:

- El conjunto de validación debe ser independiente del de entrenamiento para asegurar un correcto resultado. No puede haber ningún tipo de sesgo en el proceso de selección de los datos de validación.

- El conjunto de validación debe cumplir las propiedades de un conjunto de entrenamiento.

Además, el conjunto de validación puede utilizarse durante el aprendizaje para guiarlo en conjunción con el de entrenamiento. En este caso el proceso sería el siguiente:

1. asignar a los pesos valores aleatorios.
2. Introducir todos los ejemplos del conjunto de entrenamiento, modificando los pesos de acuerdo con el esquema de aprendizaje supervisado elegido.
3. Introducir todos los ejemplos del conjunto de validación. Obtener el error producido al predecir dichos ejemplos.
4. Si el error calculado en el paso anterior está por encima de cierto valor umbral ir a 2
5. Acabar el proceso de aprendizaje y dar como salida la red obtenida.

En este proceso, también se emplea la regla de selección del ganador para obtener la salida del clasificador neuronal, esto es, se elige como clase reconocida el índice de la neurona de la capa R que posea el mayor valor de excitación.

CAPÍTULO 6.

DESARROLLO, IMPLEMENTACIÓN Y VERIFICACIÓN DEL SISTEMA BASADO EN EL CLASIFICADOR LIRA

En este capítulo se expondrá la estructura, el algoritmo de reconocimiento y los resultados obtenidos con el clasificador neuronal LIRA *Grayscale* para la tarea de reconocimiento de orugas [58], [59].

El problema de reconocimiento de las orugas en los cultivos con diferentes formas, tamaños, texturas y posiciones no es trivial. Es por eso que, a partir de un conjunto de imágenes de orugas para entrenamiento y validación se construye el clasificador neuronal LIRA *Grayscale* para dar solución a este problema [8], [9].

En la construcción del clasificador neuronal se utilizó una base de imágenes de orugas en formato BMP con una resolución de 768 x 512 pixeles, tomadas de internet [60].

Las imágenes contenidas en la base son diversas, por ejemplo al número de orugas (Fig.6.1)

a) Image Number: 2089019
Polyphemus moth
Image Citation:
Lacy L. Hyche, Auburn University,
www.forestryimages.org

b) Image Number: 3057061
Pinkstriped oakworm
Image Citation:
James Solomon, USDA Forest Service,
www.forestryimages.org

Figura 6.1. Ejemplos de imágenes con diferentes números de orugas

Otras de las orugas tienen texturas muy diferentes (Figura 6.2). Algunas de ellas contienen cerdas, otras son lisas.

a) Image Number: 1368001
American dagger moth
Image Citation:
Joseph Berger,
www.forestryimages.org

b) Image Number: 1791015
Copper underwing
Image Citation:
Lance S. Risley, William Paterson University
www.forestryimages.org

Figura 6.2. Ejemplos de imágenes de orugas de diferentes texturas

Para trabajar con esta base de imágenes tenemos que formar dos conjuntos de imágenes: uno para el entrenamiento del sistema y otro conjunto para validar el sistema.

Las orugas pueden variar en color (Figura 6.3)

a) Image Number: 2721072
Peigler's oakworm moth
Image Citation:
Paul M. Choate, University of Florida
www.forestryimages.org

b) Image Number: 1160018
Luna moth
Image Citation:
David J. Moorhead, The University of Georgia,
www.forestryimages.org

Figura 6.3. Ejemplos de imágenes de orugas de diferente color

y en tamaño (Figura 6.4)

a) Image Number: 1791018
Eightspotted forester
Image Citation:
Lance S. Risley, William Paterson University,
www.forestryimages.org

b) Image Number: 1748032
Polyphemus moth
Image Citation:
Robert L. Anderson, USDA Forest Service,
www.forestryimages.org

Figura 6.4. Ejemplos de imágenes de orugas de diferentes tamaños.

Estos ejemplos presentados son algunos de los contenidos en la base de imágenes, que servirán para el óptimo desarrollo del clasificador.

6.1. Algoritmo de trabajo.

En general lo que hará el sistema del clasificador neuronal basado en LIRA *Grayscale* expuesto en este trabajo es:

Una imagen con de una oruga es presentada como entrada en la capa sensorial *S* del clasificador. Se codifica la imagen por medio de la capa intermedia *I.* La capa asociativa *A* extrae las características, para que la salida de la capa *R* sea calculada, es decir, obtener el tipo de clase reconocida por el sistema (si en la imagen hay presencia o ausencia de oruga).

La metodología para la programación del clasificador neuronal será el siguiente:

PASO #1. Inicializar los pesos de las conexiones entre las capas *A* y *R* a cero, es decir, $w=0$.
PASO #2. Dar la imagen a la capa de entrada *S* del sistema.
PASO #3. Codificar la imagen por medio de la capa *I* para obtener el vector de características.

PASO #4. Presentar el vector de características extraído de la imagen a la capa *A*.
PASO #5. Obtener el tipo de clase reconocida en salida de la capa *R*.
PASO #6. Modificar los pesos w_{ij} de acuerdo a la salida obtenida.
PASO #7. Repetir los paso #2 a #6 hasta que se alcance el número máximo de iteraciones.

El código de la imagen se almacenará para que en todos los sucesivos ciclos procesen este código y no la imagen desde la entrada ahorrando importantes recursos de cómputo y tiempo.

6.2. Criterios para el análisis.

En el desarrollo de este sistema, es necesario seguir ciertos criterios que ayuden a la construcción del clasificador neuronal. Es por eso que, de acuerdo a la arquitectura del clasificador neuronal LIRA *Grayscale*, ésta se debe adaptar e implementar para el reconocimiento de orugas [8], [9].

La construcción del sistema se realiza a partir de un conjunto de imágenes de entrenamiento y prueba, el cual debe ser significativo y representativo para una resolución eficiente del problema.

La fase de entrenamiento será con supervisión, es decir, con maestro. Esto es para comparar si la salida obtenida del clasificador y la salida deseada son adecuadas, sino se deben modificar apropiadamente los valores de los parámetros del sistema.

En el proceso de entrenamiento el sistema “sabe” a que clase pertenece la imagen, es decir si existe la presencia de oruga o es solamente fondo.

El proceso de validación del clasificador neuronal se hará con el otro conjunto de imágenes, esto con el propósito de obtener el porcentaje de reconocimiento alcanzado por el sistema.

Al término de estos procesos se deben realizar las modificaciones y/o correcciones al sistema de acuerdo con los resultados obtenidos.

6.3. Implementación del sistema.

El sistema de clasificación de imágenes fue desarrollado a través de C++ Builder 6 para *Windows XP*. Este software es un lenguaje de programación orientado a objetos que permite crear programas eficientes con un mínimo de código manual, provee las librerías con componentes que permiten un fácil manejo de imágenes, así como las herramientas necesarias para desarrollar y probar aplicaciones, favoreciendo un desarrollo rápido de las aplicaciones.

En el desarrollo del clasificador neuronal, se realiza un tratamiento preliminar de las imágenes [7], [8], [59], es decir, se marcan las orugas con color blanco (Figura 6.5) con la ayuda de un software para editar de imágenes.

a) Imagen inicial. b) Imagen marcada.

Image Number: 1748032
Polyphemus moth.
Image Citation:
Robert L. Anderson, USDA Forest Service,
www.forestryimages.org

Figura 6.5. Tratamiento preliminar de las imágenes de orugas

El objetivo de esto es, codificar y extraer características de la imagen necesarias para el proceso de entrenamiento de la red neuronal. La imagen contiene $H \; x \; W$ neuronas, que corresponden respectivamente al alto y ancho de la imagen. Las neuronas corresponden al brillo de cada uno de los pixeles.

El brillo de los pixeles nos ayuda a poder distinguir entre el fondo y la oruga marcada. Esto mediante el establecimiento de un umbral B, el cual sirve para discriminar a neuronas con brillo $x_i < B$ de las que tengan brillo $x_i > B$.

Las características de la imagen se obtienen por medio del siguiente procedimiento:

Se crea experimentalmente una ventana de tamaño $h \; x \; w$, (llamada también área local) la cual debe ser menor al tamaño de la imagen de entrada, esto con la finalidad de aumentar el número de muestras en la imagen durante el escaneo.

Se define el punto central de la ventana creada, ya que por medio de éste se localizaran las características de interés para el clasificador neuronal, es decir, donde haya presencia o ausencia de alguna parte de la oruga.

Mediante este procedimiento se genera el vector de características, que representa a la imagen codificada, el cual es necesario para la etapa de entrenamiento.

6.4. Entrenamiento del sistema.

La fase de entrenamiento, es con supervisión, para poder realizar los ajustes de los parámetros internos de la red y el clasificador reconozca correctamente todas las imágenes.

En este proceso se requiere contar con un conjunto de imágenes que representen las clases de interés con que se desea entrenar al clasificador. Cada una de estas imágenes debe tener asociada una etiqueta con la clase a la cual representa, de modo que el clasificador pueda ser entrenado.

El proceso de entrenamiento consta de varios ciclos, en cada ciclo el vector de características generado en la etapa de codificación, es presentado al clasificador junto con su etiqueta correspondiente, con el propósito de hacer que el sistema “aprenda” a generalizar para la etapa posterior de validación.

Después de calcular la salida de la capa *R* del clasificador, la clase correcta que corresponde a la imagen de entrada es leída a través de la etiqueta asociada a esta.

6.5. Criterios para terminar el entrenamiento.

El proceso de entrenamiento se repite hasta alcanzar un criterio de convergencia, éste depende del tipo de red utilizada o del tipo de problema a resolver. El término del ciclo de entrenamiento se basa en cualquiera de las siguientes premisas [17]:

1ª Mediante un número fijo de ciclos. Se decide a priori cuántas veces será introducido todo el conjunto de entrenamiento y una vez superado dicho número se detiene el proceso y se da por aceptada la red resultante.

2ª Cuando el error descienda por debajo de una cantidad preestablecida. En este caso habrá que definir en primer lugar una función de error, bien a nivel de patrón individual, bien a nivel de la totalidad del conjunto de entrenamiento. Se decide a priori un valor aceptable para dicho error, y solo se detiene el proceso de entrenamiento cuando la red produzca un valor de error por debajo del nivel prefijado. Para este criterio puede suceder que la red jamás consiga bajar por debajo del nivel prefijado, en cuyo caso se debe disponer de un criterio adicional para detener el proceso.

3ª Cuando la modificación de los pesos sea irrelevante. En algunos modelos se define un esquema de aprendizaje que hace que las conexiones vayan modificándose cada vez con menor intensidad. Si el proceso de aprendizaje continúa, llegará un momento en que ya no se producirán variaciones de los valores de los pesos de ninguna conexión; en ese momento se dice que la red ha convergido y se detiene el proceso de aprendizaje.

Una vez que se cumple algún criterio de convergencia, el proceso de entrenamiento termina. Para el término del proceso de entrenamiento del sistema presentado en este trabajo, se tomó como base la primera premisa.

6.6. Validación del sistema.

El objetivo de esta etapa, es verificar la eficiencia del clasificador neuronal mediante un conjunto de imágenes que no ha sido presentada aún al sistema en ninguna de las etapas anteriores. El propósito de esto es, obtener el porcentaje de reconocimiento alcanzado por el sistema.

Después de esto, mediante la fórmula 6.1 se detecta la neurona de la capa *R* con el más alto valor de salida, llamada neurona ganadora. Esta neurona representa la clase reconocida para la imagen dada como entrada.

$$y_g = max_i(E_i) \qquad (6.1)$$

De acuerdo a los resultados obtenidos, se podrá validar la eficiencia del sistema para la tarea propuesta.

CAPÍTULO 7.

EXPERIMENTOS Y RESULTADOS

En la evaluación del sistema propuesto y desarrollado en este trabajo, se crearon dos bases de datos de imágenes de orugas. Una base de 55 imágenes, donde cada imagen contiene solo una oruga. Otra base aumentando el número de imágenes de la primera hasta 79 imágenes, las cuales contienen una o más orugas.

La estructura de la red neuronal empleada para los propósitos de este trabajo, contiene las siguientes características: se utilizaron 32000 neuronas asociativas, 2 neuronas ON, 3 neuronas OFF. Las pruebas consistieron en codificar todas las imágenes de la base de datos modificando el tamaño de la ventana para escanear, es decir primero usan ventanas de 20 x 20 pixeles hasta llegar a una ventana de 120 x 120 pixeles.

Todos los experimentos fueron realizados en una computadora con procesador Intel Pentium Dual Core @2.20GHz con 1Gb de memoria RAM, con sistema operativo de *Windows XP*.

Las pruebas realizadas para los experimentos de ambos conjuntos de imágenes, se efectuaron en dos fases:

- Variando el número de imágenes para entrenamiento y dejar un número fijo de ciclos de entrenamiento.
- Variando el número de ciclos de entrenamiento y dejar fijo el número de imágenes en el conjunto de entrenamiento.

Para ambos conjuntos de imágenes, el cálculo del porcentaje de error de reconocimiento del clasificador, se realizó lo siguiente:

se obtuvo el número de ventanas totales Nv, para cada escaneo que se pueden generar sobre una imagen, de acuerdo a la siguiente fórmula:

$$Nv = \left(\frac{W}{\frac{w}{2}} - 1\right) \cdot \left(\frac{H}{\frac{h}{2}} - 1\right) \qquad (7.1)$$

donde Nv es el número total de ventanas sobre la imagen; W y H corresponden al ancho y alto respectivamente de la imagen; w y h corresponden al ancho y alto respectivamente de la ventana.

El resultado anterior se multiplica por el número total de imágenes NIm, de la base de datos para obtener el número total de ventanas Vt generadas, es decir:

$$Vt = Nv \cdot NIm \qquad (7.2)$$

donde: Vt es el número total de ventanas en toda la base imágenes; NIm es el número total de imágenes de cada conjunto.

Esto es, con el propósito de saber cuál es el número total de ventanas Vt por reconocer el sistema al momento de escanear las imágenes, para diferenciar entre el fondo y la oruga.

El porcentaje de error $\%error$ se calcula mediante una regla de tres, es decir, si sabemos que el número total de ventanas Vt por reconocer representa el 100%, entonces se tiene que obtener el porcentaje de error para el número de ventanas en las cuales el sistema erró al reconocer entre el fondo y la imagen en cuestión. Por lo tanto el porcentaje de error se calculó mediante la siguiente fórmula:

$$\%error = \frac{(100 \cdot NError)}{Vt} \qquad (7.3)$$

A continuación se presentan los resultados correspondientes a cada base de datos.

7.1. Resultados con Base de Datos de 55 imágenes.

En la primera etapa de investigación, del conjunto total de imágenes de orugas, se eligieron solo aquellas que presentaran únicamente una oruga en la imagen, independientemente de su forma, color, textura o tamaño. Obteniendo así una base de 55 imágenes en total con solo una oruga. Los experimentos para este conjunto se realizaron en dos fases.

La primera fase consistió en variar el número de imágenes para entrenamiento, es decir, tomar cierto número de imágenes de la base de 55. En este caso se tomaron 10/55 imágenes (Número Imágenes Entrenamiento entre Número Imágenes Totales), 20/55 imágenes, 30/55 imágenes para cada prueba. El resto de las imágenes de la base de datos se utilizó para la etapa de reconocimiento. En todas estas pruebas se dejó un número fijo de 150 ciclos de entrenamiento.

La segunda fase se modifico el número de ciclos de entrenamiento, es decir, para 50 ciclos y 300 ciclos, dejando fijo un número de 10 imágenes para el entrenamiento en ambos casos.

A continuación, los resultados obtenidos para 10 imágenes de entrenamiento son presentados en la Tabla 7.1.

Tabla 7.1. Error obtenido modificando el tamaño de ventana (10/55 imágenes)

Ventana	Nv	Vt	NError	%error
20 x 20	3749	296171	24562	8.293181979
40 x 40	887	70073	6105	8.712342842
60 x 60	383	30257	2528	8.355091384
80 x 80	197	15563	1584	10.17798625
100 x 100	125	9875	1254	12.69873418
120 x 120	76	6004	619	10.30979347

El mejor resultado de estos experimentos es 8.29% de errores para ventanas con tamaño 20x20. Esto significa que el porcentaje de reconocimiento correcto es de 91.71%, que para este tipo de tareas es bueno.

Los resultados de la Tabla 7.1 si bien no son de todo mal respecto al porcentaje de error, se observa que, conforme se va aumentando el tamaño de la ventana para el escaneo de la imagen, el porcentaje de error también aumenta poco. La razón de esto es porque la extracción de las características de la imagen se realiza de forma más detallada con ventanas de menor tamaño que en ventanas de tamaño mayor, al momento de efectuar el escaneo.

También se puede observar que, la diferencia entre los resultados mostrados en la Tabla 7.1 es relativamente pequeña. En algunos casos el porcentaje de error no asciende de forma regular. Una razón de esta inestabilidad en el sistema es porque, el número de imágenes utilizadas para el entrenamiento no es proporcional respecto al número de imágenes de reconocimiento, sino que es menor la cantidad del primer conjunto que el del segundo. Lo cual provoca que el sistema no pueda realizar una representación general más completa para los nuevos datos presentados.

En la siguiente prueba, se aumentó en diez el número de imágenes para entrenamiento, es decir, hasta 20 imágenes. Los resultados se presentan a continuación en la Tabla 7.2.

Tabla 7.2. Error obtenido modificando el tamaño de la ventana (20/55 imágenes).

ventana	Nv	Vt	NError	%error
20 x 20	3749	296171	24700	8.339776683
40 x 40	887	70073	5935	8.469738701
60 x 60	383	30257	2528	8.355091384
80 x 80	197	15563	1390	8.931439954
100 x 100	125	9875	870	8.810126582
120 x 120	76	6004	619	10.30979347

Al igual que para la Tabla 7.1, se observa que, los resultados para el porcentaje de error en la Tabla 7.2, es bajo para ventanas de menor tamaño al comparado con el medido en ventanas de mayor tamaño, y la diferencia entre estos errores es relativamente mínimo bajo uno respecto al otro.

En este experimento también se presenta, que con el aumento del tamaño de la ventana, aumenta el porcentaje de error. Una razón es, como en el caso anterior, la proporcionalidad entre los dos conjuntos utilizados no es la misma, ya que el de entrenamiento esta reducido respecto al conjunto de reconocimiento. Esto ocasiona que la red no sea capaz de adaptar sus pesos en las conexiones de forma eficaz.

A pesar de este inconveniente, se puede apreciar que, si hay un mejoramiento en los valores del porcentaje de error respecto a los valores medidos en la Tabla 7.1.

En la tercera y última prueba de la primera fase, se volvió a aumentar en diez el número de imágenes de entrenamiento, es decir, hasta 30 imágenes. Los resultados de este experimento se presentan en la Tabla 7.3.

Tabla 7.3. Error obtenido variando el tamaño de la ventana (30/55 imágenes)

ventana	Nv	Vt	NError	%error
20 x 20	3749	296171	24686	8.335049684
40 x 40	887	70073	6105	8.712342842
60 x 60	383	30257	2528	8.355091384
80 x 80	197	15563	1390	8.931439954
100 x 100	125	9875	962	9.741772152
120 x 120	76	6004	663	11.04263824

En la Tabla 7.3, al igual que en las Tablas 7.1 y 7.2, se presenta que el valor del porcentaje de error es relativamente bajo para ventanas de menor tamaño que el medido para ventanas de mayor tamaño.

En los conjuntos de entrenamiento y de reconocimiento en este experimento se presenta una proporción, contienen casi el mismo número de imágenes. Lo cual permite a la red poder adaptar correctamente los pesos de sus conexiones.

Al comparar los valores del porcentaje de error obtenidos en cada experimento de esta primera fase, unos respecto a los otros de las tablas anteriores, se deduce que, importa el tamaño de la ventana de escaneo, conforme esta aumenta o disminuye en tamaño, el porcentaje de error modifica su valor de la misma forma.

Otro punto por resaltar es, el número de imágenes del conjunto de entrenamiento si importa, porque dependiendo de este, la red es capaz de adaptar el valor de los pesos en sus conexiones. En estos experimentos se comprueba que a partir de 30/55 imágenes

para entrenamiento, la red comienza a tener una estabilidad en su comportamiento para dar resultados de forma confiable, es decir, es capaz de poder generalizar.

La segunda fase de los experimentos, consiste en cambiar el numero de ciclos de entrenamiento y dejar un número de 10/55 imágenes para el entrenamiento y 45/55 para el reconocimiento.

En la primera prueba se establecen 50 ciclos para el entrenamiento. Los resultados se presentan en la Tabla 7.4.

Tabla 7.4. Error obtenido utilizando 10/55 imágenes para 50 ciclos

ventana	Nv	Vt	NError	%error
20 x 20	3749	296171	24657	8.325258043
40 x 40	887	70073	5793	8.267092889
60 x 60	383	30257	2528	8.355091384
80 x 80	197	15563	1390	8.931439954
100 x 100	125	9875	870	8.810126582
120 x 120	76	6004	619	10.30979347

Los valores obtenidos para el porcentaje de error mostrados en la Tabla 7.4 son relativamente buenos ya que se mantienen casi dentro del mismo rango. Solo que la variación de los valores conforme se modifica el tamaño de la ventana de escaneo, no se aprecia mucho al comparar los valores del porcentaje de error, uno con respecto del otro, ya que no se presenta un aumento en los valores de forma gradual. Por lo tanto no se puede decidir qué ventana es conveniente para realizar el experimento.

La causa de esto, al igual que en los experimentos de la primera fase, es porque el número de imágenes utilizado para el entrenamiento no es significativo.

En la segunda prueba se aumenta el número de ciclos de entrenamiento hasta 300, dejando 10/55 imágenes para entrenamiento y 45/55 imágenes para el reconocimiento. Los resultados se presentan en la Tabla 7.5.

Tabla 7.5. Error obtenido utilizando 10/55 imágenes para 300 ciclos

Ventana	Nv	Vt	NError	%error
20 x 20	3749	296171	24658	8.325595686
40 x 40	887	70073	6087	8.686655345
60 x 60	383	30257	2725	9.006180388
80 x 80	197	15563	1500	9.638244554
100 x 100	125	9875	962	9.741772152
120 x 120	76	6004	663	11.04263824

Los valores presentados en la Tabla 7.5 respecto a los presentados en la tabla 7.4 son peores, ya que estos valores del porcentaje de error medido son más altos. Aunque en estos si se presenta un ascenso de manera regular conforme se aumenta la dimensión de la ventana de escaneo.

Al hacer la comparación de los resultados presentados en cada una de las tablas para ambas fases, se deduce que, para futuras investigaciones podemos usar 50 ciclos de entrenamiento porque la variación de los mismos no influye mucho en los valores de los resultados. Esto es bueno porque con un número de 50 ciclos se disminuye el tiempo total de trabajo del sistema en comparación de 300 ciclos.

Sin embargo, lo que si influye es el número de imágenes utilizadas para el entrenamiento de la red. Hay que tratar de que los conjuntos de entrenamiento y validación sean proporcionales, o el primero ser más significativo que el segundo.

Para ambas fases, a pesar de que el valor del porcentaje de error es menor para ventanas pequeñas, los recursos de tiempo y computo que se requiere para codificar las imágenes es mayor que para ventanas grandes.

7.2. Resultados con base de datos de 79 imágenes.

Para el estudio de casos más complejos, se realizó una segunda etapa de investigación. Aumentando hasta 79 el número de imágenes contenidas en el conjunto. Éstas contienen una o más de una oruga en la imagen también de diferentes texturas, formas y tamaño.

En esta base de imágenes se efectuaron las mismas pruebas de la primera etapa, también en dos fases: aumentando el número de imágenes para entrenamiento para 150 ciclos de entrenamiento y variando el número de ciclos de entrenamiento para un determinado número de imágenes de entrenamiento.

El primer experimento de esta etapa consistió en tomar 10/79 imágenes para el entrenamiento con un numero de 150 ciclos. Los resultaros obtenidos se muestran en la Tabla 7.6. a continuación.

Tabla 7.6. Error obtenido variando el tamaño de la ventana (10/79 imágenes)

ventana	Nv	Vt	NError	%error
20 x 20	3749	296171	31182	10.52837719
40 x 40	887	70073	7772	11.09129051
60 x 60	383	30257	3203	10.5859801
80 x 80	197	15563	1909	12.26627257
100 x 100	125	9875	1163	11.77721519
120 x 120	76	6004	1125	18.73750833

Los resultados de la Tabla 7.6 son peores a los presentados en la Tabla 7.1, ya que el porcentaje de error aumento en 2% o más. El aumento de los valores es de manera inestable. Al igual que en los casos anteriores se observa que, el porcentaje de error depende del tamaño de ventana para el escaneo, si la ventana aumenta el porcentaje de error también.

En el segundo experimento se aumentó en 10 el número de imágenes de entrenamiento, es decir, 20/79 imágenes para 150 ciclos. El resto de las imágenes del conjunto se utilizó para el reconocimiento.

Los resultados obtenidos se muestran en la Tabla 7.7.

Tabla 7.7. Error obtenido variando el tamaño de la ventana (20/79 imágenes)

ventana	Nv	Vt	NError	%error
20 x 20	3749	296171	31189	10.53074069
40 x 40	887	70073	7797	11.12696759
60 x 60	383	30257	3246	10.72809598
80 x 80	197	15563	1839	11.81648782
100 x 100	125	9875	1247	12.6278481
120 x 120	76	6004	793	13.20786143

Aunque ya se presenta una estabilidad en el aumento del porcentaje de error conforme aumenta el tamaño de la ventana. Los resultados de esta tabla también son pésimos respecto a los presentados en la Tabla 7.2.

En el tercer experimento otra vez se aumenta en 10 el número de imágenes de entrenamiento, es decir, 30/79 imágenes para 150 ciclos. Los resultados obtenidos se muestran en la Tabla 7.8.

Tabla 7.8. Error obtenido variando el tamaño de la ventana (30/79 imágenes).

ventana	Nv	Vt	NError	%error
20 x 20	3749	296171	31202	10.53513004
40 x 40	887	70073	7735	11.03848843
60 x 60	383	30257	3423	11.31308458
80 x 80	197	15563	1909	12.26627257
100 x 100	125	9875	1247	12.6278481
120 x 120	76	6004	843	14.04063957

Los resultados de la Tabla 7.8 no son tan buenos respecto a los presentados en las tablas de la primera etapa. Pero son aceptables comparados a los presentados en las Tablas 7.6 y 7.7 porque el porcentaje de error ya mantiene un aumento estable conforme aumenta el tamaño de la ventana. Razón de ello es porque el número de imágenes de entrenamiento es mayor respecto a los experimentos anteriores.

Con los resultados de las Tablas 7.6, 7.7 y 7.8, se deduce que, el porcentaje de error depende del tamaño de la ventana de escaneo.

La segunda fase, al igual que la primera etapa, consiste en variar el número de ciclos dejando fijo 10/79 imágenes para entrenamiento y el resto de las imágenes para reconocimiento.

El primer experimento se realizó para 50 ciclos de entrenamiento. Los resultados se muestran a continuación en la Tabla 7.9.

Tabla 7.9. Error obtenido utilizando 10/79 imágenes para 50 ciclos.

ventana	Nv	Vt	NError	%error
20 x 20	3749	296171	31179	10.5273643
40 x 40	887	70073	7797	11.1269676
60 x 60	383	30257	3423	11.3130846
80 x 80	197	15563	1913	12.2919746
100 x 100	125	9875	1163	11.7772152
120 x 120	76	6004	1118	18.6209194

Se puede observar que los resultados de la Tabla 7.9, el porcentaje de error aumenta de manera casi conforme la ventana de escaneo va aumentando. Sin embargo los resultados son peores a los presentados en la Tabla 7.4.

El segundo experimento consistió en aumentar a 300 ciclos el número de ciclos para 10/79 imágenes de entrenamiento. Los resultados obtenidos se presentan en la Tabla 7.10.

Tabla 7.10. Error obtenido utilizando 10/79 imágenes para 300 ciclos.

ventana	Nv	Vt	NError	%error
20 x 20	3749	296171	31171	10.5246631
40 x 40	887	70073	10880	15.526665
60 x 60	383	30257	3203	10.5859801
80 x 80	197	15563	1909	12.2662726
100 x 100	125	9875	1811	18.3392405
120 x 120	76	6004	793	13.2078614

Se observa que los resultados son pésimos porque no presentan estabilidad alguna conforme se aumenta el tamaño de ventana de escaneo. Comparados con los presentados en la Tabla 7.5 no son resultados aceptables.

Al comparar los resultados de cada una de las tablas de las dos etapas de investigación, se puede deducir que el número de ciclos de entrenamiento no es un factor importante para caracterizar el sistema. Con 150 ciclos es suficiente.

También se observa que el tipo de imágenes presentadas al clasificador influye bastante en la respuesta que dé ésta. Ya que por los resultados obtenidos en la segunda etapa se observa que no fue suficiente el número de imágenes que contienen más de una oruga, para el entrenamiento y validación del clasificador, porque no se logra el correcto ajuste de los pesos de las conexiones del sistema. Lo cual puede considerarse para los trabajos a futuro.

7.3. Tiempo de respuesta.

El tiempo de respuesta del clasificador neuronal es otro de los parámetros importantes, ya que nos permite conocer el tiempo necesario del sistema para reconocer una imagen y realizar una clasificación.

En el desarrollo de este trabajo se midió el tiempo de respuesta para el sistema con las siguientes características: ventanas de 40 x 40 pixeles, 150 ciclos y 10 imágenes para entrenamiento, 45 imágenes para reconocimiento.

El tiempo de cada operación se desglosa de la siguiente manera: siete minutos y diecisiete segundos para el entrenamiento del sistema y veintiséis segundos para el reconocimiento de 45 imágenes.

CONCLUSIONES

Se desarrolló y programo el sistema de reconocimiento de orugas en base al clasificador neuronal LIRA.

El entrenamiento y validación del sistema se realizó con dos bases datos distintas: una base de 55 imágenes la cual solo contenía una sola oruga en cada imagen, y otra base de 79 imágenes que contenían una o más orugas.

En base a los conjuntos de imágenes construidos se realizaron los experimentos de variar el número de imágenes o el número de ciclos para el entrenamiento, con diferentes tamaños de ventana de escaneo. Consiguiendo así el porcentaje de error de reconocimiento alcanzado por el clasificador neuronal creado.

De los resultados obtenidos el mejor porcentaje de reconocimiento es de 91.71% (10/55 imágenes de entrenamiento, 150 ciclos, ventana 20x20), que para este tipo de tareas es aceptable.

Se obtuvo el tiempo de respuesta del clasificador neuronal para dos procesos: entrenamiento y reconocimiento. Para el entrenamiento con 10 imágenes y 150 ciclos fue de siete minutos diecisiete segundos. Para el reconocimiento de 45 imágenes fue de veintiséis segundos.

Aunque los resultados obtenidos son muy aceptables, se debe considerar la implementación de otros experimentos para trabajos futuros con el fin de optimizar el funcionamiento del sistema.

TRABAJO A FUTURO

Los resultados obtenidos en el porcentaje de error son buenos, pero no perfectos, es por eso que se planea seguir haciendo algunas modificaciones al sistema para así mejorar la precisión de reconocimiento. Entre los objetivos para el trabajo a futuro están:

El aumento del número de imágenes de la base de datos para entrenamiento con la finalidad de mejorar la capacidad de generalización del sistema.

El incluir alguna técnica de extracción de bordes existentes en la etapa de censado, para automatizar el proceso de marcación de la localización de las orugas en las imágenes antes de que sean codificadas por la capa *I* del sistema.

Adaptar el sistema para que ejecute el reconocimiento hacia otro tipo de plagas que afectan los cultivos.

Cambiar la forma de proporcionar el nombre de clase de cada ventana en la imagen, en vez de hacerlo por medio del punto central de la ventana de escaneo, hacerlo por el cálculo de áreas máximas, es decir, el área de la clase que predomine más en la ventana será la clase reconocida por el sistema.

Otro objetivo será la programación de pausa en el sistema, en cualquiera de las fases, para que almacene la información procesada sin que haya perdida de ésta si se desea parar por algún momento la ejecución del sistema.

También lo que se pretende en el futuro es poder implementar el sistema desarrollado en algún chip para ser probado en las regiones agrícolas.

APÉNDICE

Código del programa.

```
#include <vcl.h>
#pragma hdrstop
#include <stdio.h>
#include <fcntl.h>
#include <io.h>
#include <conio.h>
#include <mem.h>
#include <string.h>
#include <stdlib.h>
#include <exception.h>
#include <time.h>
#include <iostream>
#include <wstring.h>
#include "codigo.h"

char ImageName[][7] = {"0014046","0014121","0014123","0014254","0014260",
                       "0355031","0590093","0795031","0805020","1160010",
                       "1160017","1160018","1178107","1222018","1226076",
                       "1227093","1228054","1235189","1235190","1274052",
                       "1297020","1329034","1368001","1368002","1388028",
                       "1430037","1430044","1430045","1430046","1430047",
                       "1430058","1430059","1430210","1430211","1430212",
                       "1432034","1435052","1748032","1791013","1791014",
                       "1791015","2089018","2089019","2089020","2089021",
                       "2089022","2089023","2721072","2722006","2722030",
                       "3057063","3225091","3226004","4064016","4387059",
                       };  //número de imagenes en base 55

const int FilterSize=7;
const int ImN=55;
unsigned int NumTr=10;
char buff1[7];
const char buff2[]= "LarvaeBMP";
char buff3[256];
const h=20; //alto de ventana
const w=20; //ancho de ventana
const NumOfNeurons=32000; //numero de neuronas
long int ActivSize=NumOfNeurons/4;
const int NumOfClasses=2; //0-no larva, 1-larva
int im1[h][w][NumOfClasses];
long int PositivPoint=2;
long int NegativPoint=3;
const SizeOfSubspace=PositivPoint+NegativPoint; //tamaño del subarea
//numero de componentes de la matriz de entrada aplicada a una neurona
float TDS=0.15;
const unsigned int PassN=150; //numero de ciclos
unsigned int iNum;
long int MatSize=NumOfClasses*NumOfNeurons;
```

```
long MaskSize=(PositivPoint+NegativPoint)*NumOfNeurons;
long PorogBrSize=(PositivPoint+NegativPoint)*NumOfNeurons;
unsigned short* mtr;  //[MatSize] = ClassSize*NetSize;
long unsigned* ActivNeuronNumber=new long unsigned[ActivSize];

Graphics::TBitmap *ImageA;
Graphics::TBitmap *ImageB;
Graphics::TBitmap *ImageBitmap2 = new Graphics::TBitmap();

const int WidthA=768;
const int HeightA=512;
const int WidthB=768;
const int HeightB=512;
long int Nokno=(WidthA/(w/2)-1)*(HeightA/(h/2)-1);  //numero de ventanas
                                                    //dependiendo del tamaño de ventana
long unsigned* SumNumber=new long unsigned[Nokno*ImN];

unsigned char GrayImageA[WidthA*HeightA];//A imagen inicial (original)
unsigned char DistImA[WidthA*HeightA];
unsigned char GrayImageB[WidthB*HeightB];//B oruga marcada
unsigned char DistImB[WidthB*HeightB];
unsigned oknoA[h*w],oknoB[h*w];

unsigned int pass;
long int Errors[PassN];
long int ErrorNumber,sumforwindow;
long unsigned offset, offset1;
int* SetFormation;
unsigned short true1,rec;

const unsigned char *MaskFile = "Maska.dat";
const unsigned char *PorogFile= "Porog.dat";
const unsigned char *ActiveNeuronFile = "ActiveNeuron.dat";
const unsigned char *WeightMatrixFile = "WeightMatrix.dat";

FILE *ActiveNeuron;
FILE *matr;
FILE *errors;

void ImageRead (int k);
void Ima(void);
void MaskCreation(void);
void MaskTest(void);
long int Coding(void);
void CodeGeneration(void);
void train(void);
void training(void);
unsigned short recognition_tr(void);
unsigned short recognition(void);

 //!!!Clasificador neuronal LIRA
long* ImMask;
long* PorogMask;
long* NazKl; //nombre de clase
int NumbOfActN;
```

```
FILE *st2;
FILE *fp;
FILE *mark;
FILE *remark;
FILE *nsum;
FILE *strips;
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;

unsigned char* BinaryImage;
unsigned char* BinaryImage2;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
randomize();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Coding1Click(TObject *Sender)
{
  int k,l,i1,j1,ih,jw,z,iH1,iW1,i2,j2,oknoK;
  int br, br1;
  int s[1];
  unsigned br2;
  long int iii;
  unsigned kk[1],mm[1];
  long c[1];

 oknoK=0;
 Form1->Button2->Caption="Coding_beg";
  ActiveNeuron=fopen("actn.dat","wb");
      if(ActiveNeuron==NULL)
        { Form1->Button1->Caption="Cannot open file of active neurons";
            exit(1);
          }
  fclose(ActiveNeuron);

    fp=fopen("KeySet.dat","wb");
        if(fp==NULL)
         { Form1->Button1->Caption="Cannot open KeySet with class marks";
            exit(1);
          }
    fclose(fp);

  nsum=fopen("sumN.dat","wb");
      if(nsum==NULL)
        { Form1->Button1->Caption="Cannot open file of sums";
            exit(1);
          }
  fclose(nsum);

 for(k=0;k<ImN;k++)
  {
   ImageRead(k);
```

```
z=1;   //definimos clase por punto central de ventana
for(i1=0;i1<(HeightB/(h/2)-1);i1++)//escanear imagen por punto de
                                                    //ventana
    for(j1=0;j1<(WidthB/(w/2)-1);j1++)//paso h/2 y w/2
    {
       //arreglo h*w
       for(i2=0;i2<h;i2++)
         for(j2=0;j2<w;j2++)
         {
          oknoA[i2*w+j2]=0;
          oknoB[i2*w+j2]=0;
         }
          for(i2=0;i2<h;i2++)
          for(j2=0;j2<w;j2++)
            {
             l=(i1*(h/2)+i2)*WidthB+j1*(w/2)+j2;
             oknoA[i2*w+j2]=GrayImageA[l];
             oknoB[i2*w+j2]=GrayImageB[l];
            }
             //definimos clase por punto central de ventana
             br2=oknoB[(h/2)*w+w/2];
             z=(int)br2;
             if(z==0) z=0;
             else z=1;
    fp=fopen("KeySet.dat","ab");
    if(fp==NULL)
      { Form1->Button1->Caption="Cannot open KeySet with marks";
          exit(1);
      }
    {
    mm[0]=z;
    fwrite(mm,sizeof(unsigned),1,fp);
    }
    fclose(fp);

    memset(im1, 0, sizeof int (w*h*z));
      for(i2=0;i2<h;i2++)
        for(j2=0;j2<w;j2++)
         {
          im1[i2][j2][z]=oknoA[i2*w+j2];
          br=im1[i2][j2][z];
          br1=(br<<8);
          ImageBitmap2->Canvas->Pixels[j1*(w/2)+j2][i1*(h/2)+i2]=br1;
          Form1->Canvas->Pixels[j1*(w/2)+j2][i1*(h/2)+i2]=br1;
         }
    Form1->Canvas->TextOut(25,25,"Image  "+AnsiString(k)+"  ");
    Form1->Canvas->TextOut(25,40,"Class  "+AnsiString(z)+"  ");
  iNum=k;
  ImMask=new long [MaskSize];
  PorogMask=new long[PorogBrSize];

 fp=fopen("maska.dat","rb");
       if(fp==NULL)
    { Form1->Caption="Can't open maska.dat";
       exit(1);
      }
```

```
      for(iii=0;iii<MaskSize;iii++)
      {
      fread(c,sizeof(long),1,fp);
      ImMask[iii]=c[0];
      }
      fclose(fp);

      fp=fopen("porog.dat","rb");
        if(fp==NULL)
         { Form1->Caption="Can't open porog.dat";
           exit(1);
         }

       for(iii=0;iii<PorogBrSize;iii++)
        {
         fread(c,sizeof(long),1,fp);
         PorogMask[iii]=c[0];
        }
       fclose(fp);

      CodeGeneration();
      oknoK++;
        }   //por ventana

  }    //por imagen
  delete ImMask;
  delete PorogMask;
 }
//---------------------------------------------------------------------------
   void CodeGeneration(void)
{
  int i,j,color;
  unsigned k[1];
  long int jjj,sum;

  //neuronas activas por ventana h*w
  sum=Coding();
  Form1->Canvas->TextOutA(30,200,AnsiString(sum));
  nsum=fopen("sumN.dat","ab");
   if(nsum==NULL)
    { Form1->Button1->Caption="Cannot open file of sums";
      exit(1);
    }
    {
      k[0]=sum;

   if(fwrite(k,sizeof(long),1,nsum)==NULL)
    { Form1->Button1->Caption="Cannot write to file sums";
      exit(1);
    }
    }
  fclose(nsum);

  ActiveNeuron=fopen("actn.dat","ab");
     if(ActiveNeuron==NULL)
      { Form1->Button1->Caption="Cannot open file of active neurons";
```

```
        exit(1);
       }

     for(jjj=0; jjj<sum; jjj++)
      {
       k[0]=ActivNeuronNumber[jjj];
       if(fwrite(k,sizeof(long),1,ActiveNeuron)==NULL)
      { Form1->Button1->Caption="Cannot write to file of active neurons";
         exit(1);
       }
      }
  fclose(ActiveNeuron);

  delete ImMask;
  delete PorogMask;
  delete BinaryImage;
  delete BinaryImage2;
}
//---------------------------------------------------------------------

long int Coding(void)
{
 long int i;
 long int ii,jj,kk,q,p;
 long int sum1;

  q=PositivPoint+NegativPoint;
  i=0; sum1=1;

  for(ii=0;ii<MaskSize;ii=ii+q)
    {
     jj=ii;
     for(kk=0;kk<PositivPoint;kk++)
      {
        p=ImMask[jj++];
      if(oknoA[p]<PorogMask[p]) goto m1;
      }
      for(kk=0;kk<NegativPoint;kk++)
      {
            p=ImMask[jj++];
      if(oknoA[p]>PorogMask[p]) goto m1;
      }

        ActivNeuronNumber[sum1++]=i;
        if(sum1>ActivSize-2) goto m2;
        m1: i++;
    }
    m2: ActivNeuronNumber[0]=sum1;//numero de neuronas activas por cada
                                        //ventana en la imagen

 return(sum1);
}
//---------------------------------------------------------------------

void ImageRead (int k)
{
```

```
   unsigned i,j,ii,jj,kk,hh,ww,color1,color2,sum1,sum2;
   int a1,b1,c1,d1,a2,b2,c2,d2;

 { for(i=0;i<7;i++)
      buff1[i]=ImageName[k][i];

  ImageA = new Graphics::TBitmap();
  ImageB = new Graphics::TBitmap();

  sprintf(buff3,"c:\\ProyLarva\\unaOrugaA\\a%s.bmp",buff1);
   i=0;
    ImageA->LoadFromFile(buff3);

   for(ii=0;ii<HeightA;ii++)
     for(jj=0;jj<WidthA;jj++)
     {
       color1=ImageA->Canvas->Pixels[jj][ii];
       color2=(((color1>>16)&0xff)+((color1>>8)&0xff)+(color1&0xff))/3;
       GrayImageA[ii*WidthA+jj]=color2;
       color1=(color2<<16)+(color2<<8)+color2;
       Form1->Canvas->Pixels[jj][ii]=TColor(color1);
     }
  sprintf(buff3,"c:\\ProyLarva\\unaOrugaB\\b%s.bmp",buff1);
   i=0;
    ImageB->LoadFromFile(buff3);

   for(ii=0;ii<HeightA;ii++)
     for(jj=0;jj<WidthA;jj++)
     {
       color1=ImageB->Canvas->Pixels[jj][ii];
         if(color1==ImageA->Canvas->Pixels[jj][ii])
           color2=0;     //0x00000000;
         else
           color2=255;    //   0x00FFFFFF;   clWhite
             GrayImageB[ii*WidthA+jj]=color2;

       color1=(color2<<16)+(color2<<8)+color2;
       Form1->Canvas->Pixels[jj+100][ii+30]=TColor(color1);
     }
  delete(ImageA);
  delete(ImageB);
  }
}
//---------------------------------------------------------------------

void Ima(void)
 {
   int i,j;
   long unsigned color;

     for(i=0;i<HeightA;i++)
      for(j=0;j<WidthA;j++)
       {
        color=DistImA[i*WidthA+j];
        color=(color<<8)+DistImA[i*WidthA+j];
        color=color*255;
```

```
        Form1->Canvas->Pixels[j+200][i+50]=(TColor)color;
       }

 }
//---------------------------------------------------------------------------

void MaskCreation(void)
{
 int k;
 long c[1];
 long int xx,yy,uu,vv,bb;
 long int ii,jj,q;
 ImMask=new long [MaskSize];
 PorogMask=new long [PorogBrSize];

  for(jj=0;jj<NumOfNeurons;jj++)
  {
   for(ii=0;ii<(PositivPoint+NegativPoint);ii++)
     {
       xx=random(w);   //+uu;
       yy=random(h);    //+vv;
       bb=random(255);
       q=yy*w + xx;
       ImMask[jj*(PositivPoint+NegativPoint)+ii]=q;
       PorogMask[jj*(PositivPoint+NegativPoint)+ii]=bb;
     }
  }

  fp=fopen("maska.dat","wb");
    if(fp==NULL)
     { Form1->Button1->Caption="Cannot open file mask";
       exit(1);
     }
    for(ii=0;ii<MaskSize;ii++)
    {
     c[0]=ImMask[ii];
     fwrite(c,sizeof(long),1,fp);
    }
  fclose(fp);
  delete ImMask;

 fp=fopen("porog.dat","wb");
  if(fp==NULL)
  { Form1->Button1->Caption="Cannot open file porog";
    exit(1);
  }
  for(ii=0;ii<PorogBrSize;ii++)
   {
    c[0]=PorogMask[ii];
    fwrite(c,sizeof(long),1,fp);
   }
 fclose(fp);
 delete PorogMask;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Masks1Click(TObject *Sender)
```

```
{
randomize();
MaskCreation();
MaskTest();
}
//---------------------------------------------------------------------------

void MaskTest(void)
{
 unsigned m,s;
 long k[1];
 unsigned* check=new unsigned[WidthA*HeightA];
 long int ii,jj;
 ImMask=new long [MaskSize];

 s=0;
 for(ii=0;ii<w*h/*WidthA*HeightA*/;ii++)
  check[ii]=0;

 fp=fopen("maska.dat","rb");
     if(fp==NULL)
       { Form1->Button1->Caption="Cannot open file maska.dat";
         exit(1);
       }
     for(ii=0;ii<MaskSize;ii++)
     {
      fread(k,sizeof(long),1,fp);
      ImMask[ii]=k[0];
     }
 fclose(fp);

   for(ii=0;ii<MaskSize;ii++)
   {
    m=ImMask[ii];
    check[m]++;
   }
 check[0]=0;
    for(ii=0;ii<h*w;ii++)
     {
      if (check[ii]>s)
      s=check[ii];
     }

    for(ii=0;ii<h*w;ii++)
     {
      m=(check[ii]*255)/s;
      check[ii]=m;
     }

    for(ii=0;ii<w;ii++)
      for(jj=0;jj<h;jj++)
        {
       m=check[ii*w+jj]+(check[ii*w+jj]<<8)+(check[ii*w+jj]<<16);
       Form1->Canvas->Pixels[jj+30][ii+10]=(TColor)m;
        }
 delete check;
```

```
 delete ImMask;
}
//---------------------------------------------------

void __fastcall TForm1::Begin1Click(TObject *Sender)
{
  long int ii,jj,jjj,j,sum,wini;
  int winperim;
  unsigned k[1];
  unsigned short kkk[1];
  NazKl = new long[Nokno*ImN];
  unsigned short NazKl2[((WidthA/(w/2)-1)*(HeightA/(h/2)-1))*ImN];
  mtr=new unsigned short [MatSize];

   Form1->Button2->Caption="train_beg";
    pass=0;
    ErrorNumber=0;
    winperim=0;

    offset=0;
    offset1=0;

    matr=fopen("matr.dat","wb");
      if(matr==NULL)
      {Form1->Button1->Caption="Cannot open matr";
       exit(1);
      }
    fclose(matr);

    errors=fopen("errors.dat","wt");
     if(errors==NULL)
     {Form1->Button1->Caption="Cannot open errors.dat";
      exit(1);
     }
    fclose(errors);

   for(ii=0;ii<PassN;ii++)
    Errors[ii]=0;

    for(ii=0;ii<MatSize;ii++)
     mtr[ii]=0;

    nsum=fopen("sumN.dat","rb");
      if(nsum==NULL)
        { Form1->Button1->Caption="Cannot open file of sums";
          exit(1);
        }
      for(ii=0;ii<ImN*Nokno;ii++)
       {
        if(fread(k,sizeof(long),1,nsum)==NULL)
        {Form1->Button1->Caption="Cannot read to file sums";
         exit(1);
        }
        SumNumber[ii]=k[0];
       }
    fclose(nsum);
```

```
   iNum=0;
     fp=fopen("KeySet.dat","rb");
       if(fp==NULL)
         { Form1->Button1->Caption="Cannot open KeySet with marks";
           exit(1);
         }
       for(j=0;j<ImN*Nokno;j++)
       {
        fread(k,sizeof(unsigned),1,fp);
        NazKl[j]=k[0];
        NazKl2[j]=k[0];
       }
     fclose(fp);

m3:     wini=(iNum*Nokno);  //+1;
        winperim=0;

m:  true1=NazKl2[iNum*Nokno+winperim];
    sumforwindow=SumNumber[wini+winperim];

  ActiveNeuron=fopen("actn.dat","rb");
    if(ActiveNeuron==NULL)
      { Form1->Button1->Caption="Cannot open file of active neurons";
        exit(1);
      }

    for(jjj=0; jjj<sumforwindow; jjj++)
     {
      if(fread(k,sizeof(long),1,ActiveNeuron)==NULL)
     { Form1->Button1->Caption="Cannot write to file of active neurons";
         exit(1);
       }
      ActivNeuronNumber[jjj]=k[0];
     }
  fclose(ActiveNeuron);

train();
   winperim++;
   if (winperim<Nokno)  goto m;

m2:    iNum++;
       if(iNum<NumTr) goto m3;

    Form1->Canvas->TextOutA(100,100,"wini=    "+AnsiString(wini)+"   ");
    Form1->Canvas->TextOutA(100,150,"iNum=    "+AnsiString(iNum)+"   ");

    Errors[pass]=ErrorNumber;

  Form1->Canvas->TextOutA(10,20,"Error= +AnsiString(ErrorNumber)+"   ");
  Form1->Canvas->TextOutA(10,50,"Pass=  "+AnsiString(pass)+"       ");

 matr=fopen("matr.dat","wb");
    if(matr==NULL){Form1->Button1->Caption="Cannot open matr.dat";
       exit(1);
      }
```

```
     for(jj=0; jj<MatSize; jj++)
     {
       kkk[0]=mtr[jj];
        if(fwrite(kkk,sizeof(unsigned short),1,matr)==NULL)
        { Form1->Button1->Caption="Cannot open matr.dat";
          exit(1);
        }
     }
  fclose(matr);

  pass++;
  ErrorNumber=0;

   if(pass==PassN)
    {
  errors=fopen("errors.dat","ab");
     if(errors==NULL)
     {Form1->Button1->Caption="Cannot open errors.dat";
        exit(1);
     }
      for(jj=0;jj<pass;jj++)
       {
        k[0]=Errors[jj];
        fwrite(k,sizeof(unsigned long),1,errors);
       }
  fclose(errors);
   }
    else
     {
       iNum=0;
       ErrorNumber=0;
       goto m3;
     }
   delete mtr;
    delete NazKl;
}
//---------------------------------------------
void train(void)
{
  long int jj,sum0,sum1,im,j;
  unsigned k[1];
  float q[3];

  rec=recognition_tr();
   if(rec!=true1)
    {
      ErrorNumber++;
      training();  //training
    }
}
//---------------------------------------------------------------------------

void training(void)
{
  long int j,ii,jj, sum;
```

```
  unsigned short c1;

  sum=sumforwindow;

  for(j=1;j<sum;j++)
  {
    ii=ActivNeuronNumber[j]*NumOfClasses;
      jj=ii+true1;
      c1=mtr[jj];
      if(c1<64000) c1++;
      mtr[jj]=c1;

      jj=ii+rec;
      c1=mtr[jj];
      if(c1>0) c1--;
      mtr[jj]=c1;
  }
}
//---------------------------------------------------------------------------

unsigned short recognition_tr(void)
{
  long int j,k,Active;
  unsigned short i,rec;
  long int jj,max;
  long int* sum=new long int[NumOfClasses];
  float f1,f2;

 for(i=0;i<NumOfClasses;i++)  sum[i]=0;
  for(j=1;j<sumforwindow;j++)
  {
    jj=(long)ActivNeuronNumber[j]*NumOfClasses;
      for(k=0;k<NumOfClasses;k++)
      sum[k]+=mtr[jj++];
  }
    f1=(float)(sum[true1]);
    sum[true1]=(long int)f1;
  max=0;
  rec=0;
  for(i=0;i<NumOfClasses;i++)
    if(sum[i]>max)
    {
      max=sum[i];
      rec=i;
    }
    delete sum;
    return(rec);
}
//---------------------------------------------

void __fastcall TForm1::Recognition1Click(TObject *Sender)
{
     long int i,ii,jj,jjj,sum,im,wini,i1;
     int winperim;
     int Err[1000];
     int dxx,dyy;
```

```
     unsigned k[1];
     unsigned short kkk[1];
     mtr=new unsigned short [MatSize];
     float q[3];
      NazKl = new long[Nokno*ImN];
     unsigned short NazKl2[((WidthA/(w/2)-1)*(HeightA/(h/2)-1))*ImN];
BinaryImage=new unsigned char[WidthA*HeightA];
BinaryImage2=new unsigned char[WidthB*HeightB];
ImMask=new long [MaskSize];

     iNum=0;
     ErrorNumber=0;
     i1=0;

     for(i=0;i<NumOfClasses;i++)
     {
       Err[i]=0;
     }

   nsum=fopen("sumN.dat","rb");
      if(nsum==NULL)
        { Form1->Button1->Caption="Cannot open file of sums";
            exit(1);
          }
      for(ii=0;ii<ImN*Nokno;ii++)
        {
          if(fread(k,sizeof(long),1,nsum)==NULL)
         {Form1->Button1->Caption="Cannot read to file sums";
          exit(1);
         }
         SumNumber[ii]=k[0];
        }
   fclose(nsum);

  matr=fopen("matr.dat","rb");
      for(jj=0; jj<MatSize; jj++)
      {
       if(fread(kkk,sizeof(unsigned short),1,matr)==NULL)
       { Form1->Button1->Caption="Cannot open matr.dat";
         exit(1);}
        mtr[jj]=kkk[0];
       }
  fclose(matr);

   for(jj=0; jj<MatSize; jj++)
     {
       if(mtr[jj]==1)
       i1++;
     }

  fp=fopen("maska.dat","rb");
      if(fp==NULL)
        { Form1->Button1->Caption="Cannot open file maska.dat";
          exit(1);}
       for(ii=0;ii<MaskSize;ii++)
     {
```

```
     fread(k,sizeof(long),1,fp);
     ImMask[ii]=k[0];
     }
  fclose(fp);

  remark=fopen("KeySet.dat","rb");
    if(remark==NULL)
      { Form1->Button1->Caption="Cannot open KeySet with marks";
        exit(1);
      }
    for(jj=0;jj<ImN*Nokno;jj++)
     {
      fread(k,sizeof(unsigned),1,fp);
        NazKl2[jj]=k[0];
     }
  fclose(remark);

  wini=(iNum*Nokno);  //+1;
  winperim=0;

cc1:   true1=NazKl2[iNum*Nokno+winperim];

  sumforwindow=SumNumber[wini+winperim];

  ActiveNeuron=fopen("actn.dat","rb");
    if(ActiveNeuron==NULL)
     { Form1->Button1->Caption="Cannot open file of active neurons";
       exit(1);
     }

    for(jjj=0; jjj<sumforwindow; jjj++)
      {
        if(fread(k,sizeof(long),1,ActiveNeuron)==NULL)
      { Form1->Button1->Caption="Cannot write to file of active neurons";
          exit(1);
        }
        ActivNeuronNumber[jjj]=k[0];
      }
  fclose(ActiveNeuron);

  rec=recognition();

     if(rec!=true1)
      {
        ErrorNumber++;
      }

      Form1->Canvas->TextOutA(10,10,AnsiString(iNum));
      Form1->Canvas->TextOutA(10,40,AnsiString(winperim));
      Form1->Canvas->TextOutA(10,60,AnsiString(ErrorNumber));

      winperim++;
      if (winperim<Nokno)  goto cc1;

      iNum++;
      if(iNum>ImN)goto end;
```

```
      else
        {
          winperim=0;    goto cc1;
        }

      end:  Form1->Canvas->TextOutA(100,20,AnsiString(ErrorNumber));
      delete BinaryImage;
      delete BinaryImage2;
      delete ImMask;
      delete mtr;
}
//---------------------------------------------------------------------------
  unsigned short recognition(void)
{
  long int j,k,Active;
  unsigned short i,rec;
  long int jj,max;
  long int* sum=new long int[NumOfClasses];

 Active=sumforwindow;//suma total de neuronas activas en la ventana de escaneo

  for(i=0;i<NumOfClasses;i++)  sum[i]=0;
  for(j=1;j<Active;j++)
  {
    jj=(long)ActivNeuronNumber[j]*NumOfClasses;
      for(k=0;k<NumOfClasses;k++)
      sum[k]+=mtr[jj++];
  }
  max=0;
  rec=0;
  for(i=0;i<NumOfClasses;i++)
    if(sum[i]>max)
    {
      max=sum[i];
      rec=i;
    }
  delete sum;
  return(rec);
}
```

BIBLIOGRAFÍA

[1] Duda, Richard O., Peter E. Hart, David G. Stork, *Pattern Classification.* 2ª edicion. John Wiley & Sons, USA, 2001.

[2] Rosenblatt F., *The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain,* Psychological Review, vol. 65, 1958, pp. 386-408.

[3] Rosenblatt F., *Principles of Neurodynamics*. Spartan Books, Washington, 1961.

[4] Kussul E., T. Baidyk, *Improved Method of Handwritten Digit Recognition Tested on MNIST Database,* Image and Vision Computing, vol. 22, no. 12, octubre 2004, pp. 971-981.

[5] Baidyk T., E. Kussul, *Neural Network Based Vision System for Micro Workpieces Manufacturing,* WSEAS Transactions on Systems, vol. 3, no. 2, abril, 2004, pp. 483-488.

[6] Toledo, G.K., E. Kussul, T. Baidyk, *Neural classifier for Micro WorkPiece Recognition.* Image and Vision Computing, Elsevier., vol. 24, no. 8, 2006, pp. 827-836.

[7] Baidyk, T., E. Kussul, O. Makeyev, A. Caballero, L. Ruiz, G. Carrera, G. Velasco, *Flat Image Recognition in the Process of Microdevice Assembly,* Pattern Recognition Letters, vol. 25, no. 1, enero, 2004, pp. 107-118.

[8] Baydyk T., Kussul E., Vega A. *Clasificador neuronal para reconocimiento de orugas*, Congreso de Instrumentación SOMI XXIII, Xalapa, Veracruz, México, del 1 al 3 de octubre de 2008, pp. 1-5.

[9] Baidyk T., Kussul E., *Reconocimiento de imágenes con clasificador neuronal para uso agrícola*, Congreso de Instrumentación SOMI XXII, Monterrey, Nuevo León, México, 30 de septiembre al 4 de octubre de 2007, pp. 1-6.

[10] Comisión Intersecretarial para el control del proceso y uso de plaguicidas, fertilizantes y sustancias toxicas. Catalogo de plaguicidas. México, 2004.

[11] Jurewicz J, Hanke W, Zetterström R., *Efectos de los pesticidas en los niños*. http://www.amiif.org/cms/index.php?option=com_content&task=view&id=278&Itemid=41.

[12] Servicio de información toxicológica SINTOX. Estadísticas y reportes. http://www.amifac.org.mx/sintox3.html.

[13] http://ais.paho.org/sigepi/index.asp?xml=AplicaSIG_Utecnicas/Mexico-BIG.htm.

[14] Maren Alianna, Craig Harston, Robert Pap. *Handbok of Neural Computing Applications*. USA, Academic Press, 1990.

[15] Mitchel, Tom M. *Machine Learning*. New York, McGraw Hill, 1997 pp 81-126

[16] Martin de Brio, Bonifacio, Alfredo Sanz Molina. *Redes neuronales y sistemas borrossos.* 3ª ed., México, Alfa Omega, 2007.

[17] Isasi Viñuela, Pedro. Inés M., Galván León. *Redes de neuronas artificiales. Un enfoque practico.* España, Prentice-Hall, 2004.

[18] McCulloch, W.S., and W. A. Pitts, *A logical calculus of the ideas immanent in nervous activity.* Bulletin of Mathematics and Biophysics, 5, 1943, pp. 115-133.

[19] Hebb, D. O., *The organization of behavior,* Wiley, New York, 1949.

[20] Minsky, M., *Neural-analog networks and the brain model problem.* Ph.D. Thesis, Pricenton University.

[21] Uttley, A., *Automata Studies, chapter Conditional probability machines and conditional reflexes*, Princenton University Press, Princenton, 1956, pp. 253-276.

[22] Uttley, A., *Automata Studies, chapter Temporal and spatial patterns in a conditional probability machines*, Princenton University Press, Princenton, 1956, pp. 277-285.

[23] Widrow, B., M. E. Hoff, *Adaptive switching circuits*, WEST Convention, Record part IV, New York, 1960, pp. 96-104.

[24] Widrow, B., S. D. Stearns, *Adaptive Signal Processing*, Prentice-Hall, Englewood Cliffs NJ, 1985.

[25] Steinbuch, K., *Die Lematrix. Kibernetik.* 1, 1961, pp. 36-45.

[26] Steinbuch, K., U. Piske, *Learning matricers and their application.* IEEE Transaction on Electronic Computers, EC-12, 1963, pp. 846-862.

[27] Grossberg, S., *The theory of embedding fields with applications to psychology and neuropsychology.* Rockfeller Institute of Medical Research, New York, 1964.

[28] Minsky, M., and S. Papert, *Perceptrons,* MIT Press, Cambridge MA, 1969.

[29] Amari, S., *Learning patterns and pattern sequences by self-organizing nets of threshold elements,* IEEE Transactions on Computers, 21, 1972, pp. 1197-1206.

[30] Werbos, P. J., *Beyond regression: New tools for prediction and analysis in the behavioral sciences,* Ph.D. Thesis, Harvard University, 1974.

[31] Anderson, J., *A theory for the recognition of items from short memorized list.* Psychological Review, 1973, pp. 417-438.

[32] Anderson, J., and G. Murphy, *Psychological concepts in a parallel system.* Physica, 1986, pp. 318-336.

[33] Fukushima K., *Cognitron: A self-organizing multilayered neural network.* Biological Cybernetics, 1975, pp. 121-136.

[34] Fukushima K., Neocognitrón: *A self-organizing neural network for a mechanism of pattern recognition unaffected by shift in position.* Biological Cybernetics, 36, 1980, pp. 193-202.

[35] Kohonen, T., *Self-organized formation of topologically correct feature maps.* Biological Cybernetics, 43, 1982, pp. 59-69.

[36] Feldman, J., *A distributed information processing model of visual memory.* Technical Report TR-52, University of Rochester, Departament of Computer Science Technical Report.

[37] Ballard, D., *Computer Vision.* Prentice-Hall, Englewood Cliffs.

[38] Hopfield, J., *Neural Computation of Decisions in Optimization Problems.* Biological Cyernetics, 52, 1985, pp. 141-152.

[39] Hopfield, J., *Neural Networks and physical system whit emergent collective computational abilities.* Proceedings of the National Academy of Sciences, 79, 1982, pp. 2554-2558.

[40] Kosko, B., *Competitive adaptive bidirectional associative memories.* In Proceeding of IEEE First International Conference on Neural Networks, vol. 2, San Diego, 1987, pp. 759-766

[41] Hinton, G., D. Ackley, T. Sejnowski, *Boltzmann machines: Constrain satisfaction networks that learn.* Carnegie-Mellon University, Departament of Computer Science Technical Report, 1984.

[42] Carpenter G., S. Grossberg, *The ART of Adaptive Pattern Recognition by a Self-Organizing Neural Network.* IEEE Computer, marzo 1987, pp. 77-88.

[43] Hecht-Nielsen, R., *Neurocomputing.* Addison-Wesley, Massachusetts, 1990.

[44] Rumelhart, D., G. Hinton, R. Williams. *Learning representations by back-propagating errors.* Nature, 1986, pp. 533-536.

[45] Vapnik, V. N., *Statistical learning theory.* Springer Verlag, 1995.

[46] http://www.correodelmaestro.com/anteriores/2004/febrero/1anteaula93.htm

[47] Hilera, José R., Victor J. Martinez. *Redes neuronales artificiales. Fundamentos, modelos y aplicaciones.* Wilmington, Delaware, Addison-Wesley, 1995.

[48] Muñoz Torres, Juan Carlos. *Apuntes para clase de procesamiento digital de imágenes medicas.* México 2008.

[49] Azpirov J., V. Medina, J. Lerallut, *Procesamiento Digital de Imágenes Biomedicas.* Universidad Autonoma Metropolitana, 2000.

[50] Baidyk, T., E. Kussul, *Redes Neuronales, visión computacional y micromecánica,* Itaca, México, 2009.

[51] Kussul E., T. Baidyk, V. V. Lukovitch, D. A. Rachkovskij, *Adaptive High Performance Classifier Based on Random Threshold Neurons,* Proceedings of Twelfth European Meeting on Cybernetics and Systems Research (EMCSR-94), Viena, abril 5-8, 1994, en R. Trappl (ed.), Cybernetics and System 94, World Scientific Publishing Co. Pte. Ltd, Singapur, pp. 1687-1695.

[52] Kussul E., T. Baidyk, *Permutative Coding Technique for Handwritten Digit Recognition*, Proceedings of the International Joint Conference on Neural Networks, vol.3, Portland, Oregon, julio 20-24, 2003, pp. 2163-2168.

[53] Kussul E., T. Baidyk, V. V. Lukovitch, D. A. Rachkovskij, *On Image Texture Recognition by Associative Projective Neurocomputer, en* Proceedings of the ANNIE'91 conference "Intelligent engineering system through artificial neural networks", ASME Press, C.H. Dagli and S. Kumara and Y.C, Shin, 1991, pp. 453-458.

[54] Kussul E., T. Baidyk, D. A. Rachkovskij, *Associative-projective Neural Networks: Architecture, Implementation, Applications,* Nimes, noviembre 4-8, 1991, pp. 463-476.

[55] Kussul E., T. Baidyk, D. Wunsch, *Image Recognition Systems whit Permutative Coding,* International Joint Conference on Neural Networks, Montreal, julio 2005, pp. 1788-1793.

[56] LeCun, Y., L. Bottou, Y. Bengio, P. Haffnier, *Gradient-based Learning Applied to document recognition,* Proceeding of the IEEE, vol. 86, no. 11, November 1998, pp. 2278-2344.

[57] Hoque M.S., M.C. Fairhurst, *A moving window classifier for off-line character recognition,* en: Proceedings of the Seventh International Workshop on Frontiers in Handwriting Recognition, 2000, pp. 595-600.

[58] Baidyk T., Kussul E., Makeyev O., Vega A., *Limited receptive area neural classifier based image recognition in micromechanics and agriculture*, International Journal of Applied Mathematics and Informatics, Issue 3, Vol. 2, 2008, pp.96-103.

[59] Baidyk T., Kussul E., Makeyev O., Vega A., *Image recognition with neural classifiers in micromechanics and agriculture*, Proceedings of the 10th WSEAS International Conference on Neural Networks NN'09, Prague, Czech Republic, March 23-25, 2009, pp.100-103.

[60] http://www.forestryimages.org/browse/catsubject.cfm?cat=15.