INSTITUTO POLITÉCNICO NACIONAL

Centro de Investigación en Computación

APLICACIÓN DE LAS REDES NEURONALES PULSANTES EN EL RECONOCIMIENTO DE PATRONES Y ANÁLISIS DE IMÁGENES

TESIS QUE PRESENTA:

Idarh Claudio Matadamas Ortiz

Para obtener el grado de:

MAESTRO EN CIENCIAS EN INGENIERÍA DE CÓMPUTO

DIRECTORES DE TESIS:

Dr. Juan Humberto Sossa Azuela

M.C Osvaldo Espinosa Sosa

México D.F. Enero de 2014

SIP-14 bis

INSTITUTO POLITÉCNICO NACIONAL
SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

ACTA DE REVISIÓN DE TESIS

En la Ciudad de México, D.F. siendo las 12:00 horas del día once del mes de diciembre de 2013 se reunieron los miembros de la Comisión Revisora de la Tesis, designada por el Colegio de Profesores de Estudios de Posgrado e Investigación del:

Centro de Investigación en Computación

para examinar la tesis titulada:

"Aplicación de las redes neuronales pulsantes en el reconocimiento de patrones y análisis de imágenes"

Presentada por el alumno:

Matadamas	**Ortiz**	**Idarh Claudio**
Apellido paterno	Apellido materno	Nombre(s)

Con registro:

A	1	2	0	5	8	2

aspirante de: **MAESTRÍA EN CIENCIAS EN INGENIERÍA DE CÓMPUTO CON OPCIÓN EN SISTEMAS DIGITALES**

Después de intercambiar opiniones los miembros de la Comisión manifestaron ***APROBAR LA TESIS***, en virtud de que satisface los requisitos señalados por las disposiciones reglamentarias vigentes.

LA COMISIÓN REVISORA

Directores de Tesis

Dr. Juan Humberto Sossa Azuela

M. en C. Osvaldo Espinosa Sosa

Dr. Sergio Suárez Guerra

Dr. Herón Molina Lozano

Dra. Elsa Rubio Espino

Dr. Ricardo Barrón Fernández

PRESIDENTE DEL COLEGIO DE PROFESORES

Dr. Luis Alfonso Villa Vargas

INSTITUTO POLITECNICO NACIONAL
CENTRO DE INVESTIGACION EN COMPUTACION
DIRECCION

INSTITUTO POLITÉCNICO NACIONAL

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

CARTA CESIÓN DE DERECHOS

En la Ciudad de **México D.F**, el día **16** del mes de **Diciembre** del año **2013**, el (la) que suscribe **Idarh Claudio Matadamas Ortiz**, alumno (a) del Programa de **Maestría en Ciencias en Ingeniería de Cómputo con Opción en Sistemas Digitales** con número de registro **A120582**, adscrito al **Centro de Investigación en Computación**, manifiesta que es autor (a) intelectual del presente trabajo de Tesis bajo la dirección del **Dr. Juan Humberto Sossa Azuela** y el **M.C. Osvaldo Espinosa Sosa**, y cede los derechos del trabajo intitulado **"Aplicación de las Redes Neuronales Pulsantes en el Reconocimiento de Patrones y Análisis de Imágenes"**, al Instituto Politécnico Nacional para su difusión, con fines académicos y de investigación.

Los usuarios de la información no deben reproducir el contenido textual, gráficas o datos del trabajo sin el permiso expreso del autor y/o director del trabajo. Este puede ser obtenido escribiendo a la siguiente dirección **idarmo@hotmail.com**. Si el permiso se otorga, el usuario deberá dar el agradecimiento correspondiente y citar la fuente del mismo.

Idarh Claudio Matadamas Ortiz

Nombre y firma

Resumen

En la presente tesis se propone un método para la aplicación de Redes Neuronales Pulsantes a problemas de reconocimiento de patrones y análisis de imágenes, la arquitectura empleada para la red neuronal hace uso del modelo neuronal de Izhikevich como unidad de procesamiento; el entrenamiento no supervisado y ajuste de los parámetros libres, se realizan por medio de un algoritmo de Evolución Diferencial programado en lenguaje C++.

Usando una arquitectura *feedforward* con tres capas: capa de entrada (E), capa intermedia (O) y capa de salida (S); se logra la clasificación de distintos tipos patrones utilizando la frecuencia obtenida en la capa de salida de la red. El cálculo de la corriente inyectada a la entrada de las neuronas, se realiza por medio de funciones de transformación que permiten pasar de un vector de rasgos en *n* dimensiones a un valor escalar que facilite la separación. Se propone la utilización de la arquitectura y el método de entrenamiento en el problema de umbralado de imágenes utilizando como rasgos descriptores características de color y textura.

Abstract

In this thesis we propose a method for the application of pulsed neural networks to problems in pattern recognition and image analysis, the architecture used for the neural network makes use of the Izhikevich neuronal model as processing unit; the unsupervised training and free parameters adjustment performed by a Differential Evolution algorithm programed in C++ language.

Using a *feedforward* architecture with three layers: the input layer (E), intermediate layer (O) and output layer (S), the classification of different pattern types is achieved using the frequency obtained in the output layer of the network. The calculation of the current injected to the input neurons is realized by means of transformation functions which allow the pass of a feature vector in *n*-dimensional to a scalar value that facilitates separation. Proposes the use of the architecture and training method thresholding problem as features of images using descriptors of color and texture features.

Agradecimientos

En primer lugar quiero dedicar este logro y al mismo tiempo agradecer a toda mi familia que no ha dejado de apoyarme durante toda mi vida, gracias a mis padres y a mis hermanos.

A mi novia, quien estuvo a mi lado para apoyarme y darme ánimos día a día durante este proyecto, Emily muchas gracias por ser esa persona que siempre está ahí para mí, eres una parte importante de este logro.

Quiero agradecer a mis directores, el Dr. Juan Humberto Sossa Azuela y el M.C. Osvaldo Espinosa Sosa, por cada uno de sus consejos, sin ustedes nada de esto habría sido posible.

Al Instituto Politécnico Nacional por brindarme la oportunidad de crecer de forma académica y personal.

A cada uno de los integrantes del laboratorio de robótica y mecatrónica del Centro de Investigación en Computación que fueron mis compañeros durante este periodo y con quienes adquirí una muy buena amistad.

También, se agradece a la SIP-IPN a través de los proyectos: SIP 20121311, SIP 20131182, al CONACyT por el proyecto 155014 y al ICyTDF por el apoyo a través del proyecto 325/2011.

Finalmente agradezco al Consejo Nacional de Ciencia y Tecnología por haberme aceptado como becario de tiempo completo para el desarrollo de la siguiente tesis y por el soporte económico otorgado por dos años.

Muchas gracias a todos

ÍNDICE GENERAL

Página

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

ÍNDICE DE ANEXOS

GLOSARIO DE SIGLAS

RNA ..... Redes Neuronales Artificiales.
RNP ..... Redes Neuronales Pulsantes.
ED ..... Evolución Diferencial.
RP ..... Reconocimiento de Patrones.
AI ..... Análisis de imágenes.
MLP ..... MultiLayer Perceptron.
LSM ..... Least Mean Square algorithm.
STDP ..... Spike Time Dependet Plasticity.
ADDS ..... Active Dendrite and Dynamic Synapses
GPU ..... Graphics Processing Unit.
FPGA ..... Field Programmable Gate Array.
RS ..... Regular Spikes.
MFCC ..... Mel Frequency Cepstral Coefficients.
MVS ..... Máquinas de Vector Soporte.
CUDA ..... Compute Unified Device Architecture.

Capítulo 1
Introducción

El ser humano cuenta con uno de los sistemas de reconocimiento y clasificación de patrones más sofisticado dentro de la naturaleza, esto se aprecia al observar la facilidad con la que se realiza el reconocimiento y clasificación de distintos tipos de patrones, ya sean imágenes, sonidos, olores, etc., aún en presencia de ruidos y afectaciones; son procesos que se realizan de manera natural y que proporcionan información necesaria para la toma de decisiones de todos los días, no obstante la aparente sencillez de estos procesos de reconocimiento y clasificación, aún no se puede dar una explicación de cómo es que se realizan estas acciones dentro del cerebro humano.

Conforme se van generando nuevos conocimiento en la forma en que funciona el sistema nervioso, sobre todo el de los animales, surgen nuevos modelos de Redes Neuronales Artificiales (RNA). Un esquema que ha tomado mucha importancia en los últimos años es el de las Redes Neuronales Pulsantes (RNP), estas redes se inspiran a partir de que las células del sistema nervioso generan una serie de potenciales de acción en forma de trenes de pulsos o espigas que se pueden analizar como señales en el dominio del tiempo y cuya frecuencia y forma de los pulsos contienen la información necesaria para la percepción y procesamiento [8], [5], [19] y [15].

En este proyecto de tesis se presenta un método para utilizar una Red Neuronal Pulsante entrenada por medio de un algoritmo de Evolución Diferencial (ED). Este método es utilizado para realizar tareas de reconocimiento de patrones, además de probar su desempeño en el problema de umbralado de imágenes.

1.1 Antecedentes.

El cerebro humano es un sistema de extrema complejidad al estar compuesto de aproximadamente 10^{11} neuronas densamente interconectadas, además entre las neuronas existen alrededor de 10^{15} conexiones entre ellas, cada una con aproximadamente 10^{4} conexiones hacia otras neuronas, al mismo tiempo que recibe un número igual de conexiones desde otras neuronas [1].

Las neuronas son un grupo de células del sistema nervioso que tienen como característica básica la excitabilidad de su membrana plasmática; su principal función es la recepción de estímulos y la conducción del impulso nervioso en forma de potencial de acción, presentan estructuras morfológicas muy particulares que soportan sus funciones: un cuerpo celular o “pericarión” central, una o varias prolongaciones cortas, llamadas dendritas, que son las encargadas de recibir impulsos generados de otra neuronas y transmitirlos hacia el soma celular; y una prolongación larga denominada axón que conduce los impulsos desde el soma hacia otras neuronas u órgano.

En condiciones de reposo toda la neurona; hablando del cuerpo celular, el soma y las dendritas, está polarizado de tal manera que en el interior se tiene un potencial de -70 mv, también llamado potencial de reposo [1], como se mencionó anteriormente, la neurona recibe información de otras neuronas por medio de las dendritas, presentada como cambios en el potencial eléctrico de éstas (potenciales postsinápticos), estos cambios son sumados de manera temporal y espacial dentro del cuerpo celular y cuando se alcanza un cierto umbral la neurona genera una señal llamada potencial de acción. Este potencial

de acción es transmitido hacia el exterior por medio del axón y ayudará a la modificación del estado eléctrico en la membrana de la neurona siguiente, siendo éste de tipo excitatorio o inhibitorio, contribuyendo a que genere un potencial de acción más fácilmente o impidiéndolo.

Desde que McCulloch y Pitts en 1943, postularon en su trabajo que las neuronas funcionan como un dispositivo booleano, se cimentó la base para el surgimiento de las RNA, al modelar una neurona compuesta con una función lineal, que representa a las sinapsis, seguida de una función activación booleana, que representa a su vez al procesamiento que se realiza dentro de la neurona [8]. Basándose en estos trabajos comenzaron a aparecer variaciones del modelo como el perceptrón de Rosenblatt en 1958.

Gracias a los avances en las neurociencias y a las características anatómicas y funcionales de la corteza cerebral es posible la elaboración de modelos que permiten su estudio teórico a través de la modelación matemática, uno de los éxitos más famosos es el que lograron los neurofisiólogos A. Hodgkin A. Huxley en el año de 1952, que construyeron un modelo matemático para explicar el comportamiento del potencial de membrana de una neurona biológica, logrando una alta fidelidad en el comportamiento, comparado con las células nerviosas del calamar.

Numerosos modelos de Redes Neuronales Artificiales basadas en diversas teorías del funcionamiento de las neuronas biológicas han sido propuestos, dentro de éstos se puede hacer la clasificación de éstas en tres generaciones de RNA [7]. La primera generación está basada en la utilización de la neurona de McCulloch–Pitts; refiriéndose al perceptrón o compuertas de umbralado, de las cuales se desprenden una variedad de modelos de RNA como el Perceptrón Multi-Capa (MLP), redes de Hopfield y máquinas de Boltzmann, donde la característica principal de estos modelos es que sólo pueden entregar una salida binaria, es decir, a la salida sólo se puede tener como resultado 1 ó 0.

La segunda generación está basada en unidades de procesamiento que aplican una función activación con un conjunto continuo de posibles valores de salida. La función activación comúnmente utilizada en este tipo de redes neuronales es la función sigmoidea. Otra característica que presenta este tipo de modelos de RNA es que soportan algoritmos de aprendizaje como el algoritmo *Backpropagation*, con este último se demostró que con Redes Neuronales Artificiales, compuesta por múltiples capas se pueden resolver problemas de clasificación no lineales.

Conforme se tiene más conocimiento del funcionamiento del sistema nervioso de los animales, van surgiendo nuevos modelos inspirados en él, modelos que generan una serie de potenciales de acción que se pueden analizar como señales en el dominio del tiempo [1]. Ésta es la tercera generación de redes neuronales, las llamadas Redes Neuronales Pulsantes. Este esquema es muy distinto al modelo de McCulloch-Pitts ya que propone que la información que entrega una neurona, va más allá de una constante binaria, y sostiene que dicha información se encuentra contenida en la frecuencia y amplitud de los pulsos generados en el tiempo, y en forma de una señal conformada por una serie de picos o espigas [8] y [36]. En secciones posteriores se detallaran más las distintas generaciones de RNA.

Aunque en la actualidad se ha puesto un gran interés en el estudio de este tipo de redes neuronales y distintos autores han propuesto varias arquitecturas de RNP, en la literatura no se encuentra un modelo general que dé indicios del porqué de su estructura.

Sin embargo existen algoritmos de aprendizaje como la Regla de Plasticidad de Actividad Sináptica o el STDP (*Spike Time Dependet Plasticity*) [29], el algoritmo de *Backpropagation* para redes de neuronas pulsantes (*SpikeProp*) [23], además se han utilizado métodos estadísticos, métodos con algebra lineal, y evolutivos [26].

1.2 Planteamiento del problema.

Como se ha demostrado en distintos trabajos, la utilización de RNA en la investigación va más allá de sólo una simulación, los modelos de RNP han sido utilizados en distintas áreas tales como el reconocimiento de patrones, control de robots, reconocimiento de objetos, entre otras. Aunque en la literatura se encuentran distintas arquitecturas de RNP capaces de llevar a cabo cierta tarea, ya sea reconocimiento, clasificación, etc., así como un método de entrenamiento propio, no se ha presentado una arquitectura un poco más general que pueda ser utilizada en el procesamiento de distintas bases de datos.

Al analizar este problema se presentan distintas preguntas, por ejemplo, ¿cuántas neuronas y de qué tipo debo utilizar para mi problema?, ¿cómo debo utilizarlas y de qué manera puedo acoplarlas para que trabajen juntas?, ¿Qué método de entrenamiento debo utilizar?

En el presente trabajo de tesis se busca hacer un acercamiento, a una metodología para la utilización de una Red Neuronal Pulsante, que utilice un método automático de ajuste de sus parámetros libres en el entrenamiento, para realizar las tareas de reconocimiento de distintos tipos de patrones y que pueda ser utilizada en tareas de análisis de imágenes.

1.3 Motivación.

El campo del reconocimiento de patrones es un área muy estudiada en la actualidad, de tal manera que distintas propuestas han sido desarrolladas para resolver este problema, como es la red de perceptrones multicapa. Sin embargo, en los últimos años ha tomado mucho interés la utilización de las llamadas Redes Neuronales de tercera generación, también conocidas como Redes neuronales Pulsantes; estas redes neuronales poseen cualidades que

las hacen ser muy parecidas a la realidad biológica, es decir, su grado de realismo es mayor a las de primera y segunda generación [57].

En el presente trabajo se decidió utilizar el modelo neuronal de Izhikevich como base para llevar a cabo la solución de los problemas de reconocimiento de patrones y el análisis de imágenes, en particular el umbralado de éstas, por las siguientes razones.

- Su cercanía con las neuronas ubicadas en la corteza del cerebro humano ha sido de gran interés en el campo del análisis de imágenes, al grado de dar pie a la generación de redes neuronales complejas, basadas en el sistema visual humano, capaces de trabajar con base a una serie de estímulos de forma y color [25], [43] y [44].
- Pertenece a la generación de redes neuronales en las que se está prestando mucho interés, utilizadas en tareas tanto de robótica [16], [17] y [18], análisis de imágenes [14] y [34], reconocimiento de patrones [9], [10] y [28], entre otras.
- Gracias a sus características, las neuronas pulsantes permiten realizar la transformación de un problema en n dimensiones, a uno más simple en una sola dimensión por medio de la frecuencia en sus disparos, además de permitir crear una asociación entre diferentes clases de patrones, con la generación de diferentes frecuencias de disparo, como se ha visto en distintos trabajos, en donde se utiliza la frecuencia para la discriminación entre clases [9], [10] y [28].
- El modelo neuronal cuenta con una gran facilidad para reproducir diferentes patrones de disparo, que pueden ser utilizadas para codificar diferentes tipos de información [41].
- Se ha demostrado que una sola neurona es capaz de resolver problemas de clasificación no lineal, logrando con esto una disminución en cuanto al número de neuronas utilizadas en la resolución de problemas, y en la complejidad de la red neuronal [9], [10] y [28].

Esta forma de trabajo muy cercana a lo que lo en realidad hacen las neuronas biológicas da pie a realizar investigación y aplicación de este tipo de Redes Neuronales Artificiales para tareas de reconocimiento de patrones de distintos tipos, ya sea de imágenes, audio, etc., y observar el comportamiento y desempeño de la misma ante patrones desconocidos, es decir, su capacidad de generalización.

1.4 Hipótesis.

Teniendo un conjunto de patrones de entrada, pertenecientes a un conjunto de clases *K*, cada patrón es presentado a la capa de entrada de una Red Neuronal Pulsante, para posteriormente computar el tren de pulsos entregado en la capa de salida. Después de ajustar los parámetros de la red neuronal, los patrones pertenecientes a la misma clase entregarán un tren de pulsos parecido entre sí, logrando la clasificación correcta de éstos.

1.5 Objetivos.

1.5.1 Objetivo general

- Implementar en software una RNP, en conjunto con un método de entrenamiento de sus parámetros, para resolver algunos problemas en reconocimiento de patrones (RP) y análisis de imágenes (AI).

1.5.2 Objetivos particulares.

- Implementación de una RNP en software.
- Dado un conjunto de patrones $(\{x_i\},\{c_k\}), i = 1,\cdots,n \quad k = 1,\cdots,K$, poner en operación un método que permita el ajuste automático de los parámetros de la RNP, de forma que x_i pueda ser clasificado.
- Probar la capacidad de clasificación de la RNP con diversos conjuntos de patrones: imágenes y sonido.

1.6 Organización del trabajo.

El presente trabajo se organiza como se describe a continuación, comenzando con el capítulo dos el cual se refiere al estado del arte en cuanto a trabajos relacionados con el reconocimiento de patrones y la utilización de las Redes Neuronales Pulsantes; posteriormente, en el capítulo tres se hace una descripción de las herramientas utilizadas para el desarrollo del trabajo; la metodología seguida para desarrollar el trabajo se presenta en el capítulo cuatro; mientras que en el capítulo cinco se presentan los resultados obtenidos de la aplicación de las Redes Neuronales Pulsantes, en el reconocimiento de distintos tipos de patrones y en el análisis de imágenes; por último en el capítulo seis se muestran las conclusiones obtenidas, así como las recomendaciones para trabajos futuros.

Capítulo 2
Estado del arte

Desde la aparición de la tercera generación de RNA se han realizado trabajos en distintas áreas en las que se utilizan las RNP para el reconocimiento de patrones; análisis de imágenes [8], [14], [30], [34] y [35], controlar juegos de video [15], control de robots [16], [17] y [18], en [40] se presenta la utilización de una brazo robot capaz de reconocer objetos por medio de la forma. En cuanto a simulaciones a gran escala los modelos de Redes Neuronales Pulsantes han sido utilizados como base para trabajos sobre GPU (*Graphics Processing Unit*) [5] y [35] y en arquitecturas reconfigurables de tipo FPGA que aprovechan el bajo costo, su fácil acceso y su software flexible para desarrollar sistemas para el reconocimiento de caracteres y realización de simulaciones a gran escala [12], [43] y [45], gracias a su ciclo de desarrollo reducido a la facilidad para adaptarse a las características de la RNA [4].

Por otra parte se han realizado trabajos dedicados a proponer métodos de entrenamiento para RNP desde los métodos basados en plasticidad sináptica, uso de algoritmos evolutivos, hasta algoritmos basados en la retro-propagación del error [26]. Los métodos más utilizados en la literatura son los que hacen uso de un aprendizaje supervisado, es decir, encontrar un vector de pesos sinápticos que logren, obtener un tren de pulsos determinado con anterioridad.

En [23], [42] y [31] se proponen algoritmos de entrenamiento supervisado que utilizan una regla basada en la retro-propagación del error, demostrando con esto que el entrenamiento se puede hacer para problemas no lineales, en [33] se hacen mejoras al algoritmo logrando reducir el número de pesos necesarios. Si se habla de entrenamiento a través de métodos evolutivos se encuentran trabajos como [9], [10], [28], [32] y [39] en donde utilizan algoritmos de Evolución Diferencial y Algoritmos Genéticos para el entrenamiento de las RNP.

A continuación se presenta una revisión de los trabajos más relevantes en cuanto a la utilización de RNP para el reconocimiento de patrones y análisis de imágenes.

2.1 Reconocimiento de patrones a través de RNP.

Sander *et Al*., (2002) en [23] y [42] demuestran que un problema de clasificación no lineal puede ser resuelto por una RNP con arquitectura *feedforward*. En este trabajo se utiliza como unidad de procesamiento el modelo *Leaky Integrate-and-Fire*, además de proponer un método de entrenamiento análogo a la regla de entrenamiento *Backpropagation*. Usando una red de diez y nueve neuronas, se logra la separación del problema XOR usando el tren de pulsos de la neurona de salida. Se concluye que una RNP se desempeña de forma comparable con una RNA que utiliza la clásica neurona con función sigmoidea. En [33] se presenta una mejora del algoritmo de entrenamiento, llegando a la misma conclusión en cuanto a la clasificación del problema XOR.

En los trabajos presentados por Roberto A. Vásquez en [9], [10] y [28] se presentan resultados de experimentos de reconocimiento de patrones utilizando Redes Neuronales Pulsantes, en estos trabajos el ajuste los pesos de la neurona fue realizado por medio de un algoritmo evolutivo, la clasificación se realiza basada en los pulsos generados por la neurona después de ser estimulada por la corriente de entrada. También realiza la comparativa de sus

resultados entre dos modelos de neurona pulsante, el modelo neuronal de Izhikevich en [9] y [10], y el modelo *Integrate-and-Fire* [28].

En dichos experimentos de reconocimiento de patrones fueron utilizados los conjuntos de datos de la plata de Iris, Vino, Vidrio y desordenes del Hígado; además de realizar un experimento de reconocimiento de objetos, a partir de rasgos extraídos de imágenes, la figura 1 muestra los objetos utilizados para el experimento.

En este experimento se realiza un análisis de cada una de las imágenes para extraer los siete rasgos invariantes de Hu. En estos trabajos se demuestra que las neuronas pulsantes pueden ser utilizadas en problemas de reconocimiento de patrones de diferentes tipos, teniendo un desempeño comparable y en ocasiones mejor que las redes neuronales utilizadas comúnmente.

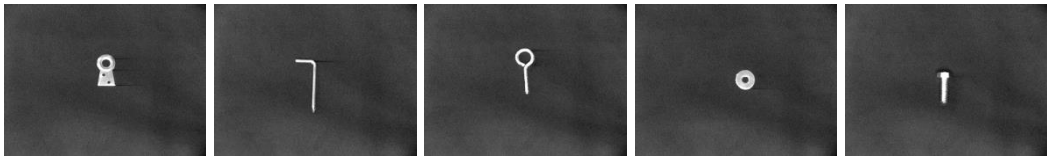


Figura 1. Conjunto de imágenes utilizadas en [8] y [10] para el reconocimiento de objetos utilizando RNP.

En [39] se presentan resultados de clasificación en bases de datos tales como el conjunto Iris Plant, Diabetes y el conocido problema XOR; los autores presentan un método de entrenamiento en el que se utilizan técnicas evolutivas, el algoritmo de Evolución Diferencial para ser precisos, además de proponer esquemas de RNP para cada uno de los problemas. Cada una de las arquitecturas propuestas está diseñada para trabajar únicamente con el problema dado, de esta forma, para el problema de la función booleana XOR se propone una red neuronal de la forma 2-2-1, para el problema de la planta del Iris se utiliza una red 4-4-1 y para el conjunto de datos de la Diabetes se tiene una arquitectura 8-4-4-2. Todas con una conexión de tipo *feedforward*. Los resultados reportados son comparables con los obtenidos con una red MLP.

Por otra parte trabajos relacionados con el reconocimiento de caracteres pueden ser encontrados en [12] y [13] donde se presenta la utilización de una Red Neuronal Pulsante para el reconocimiento de 48 caracteres de una dimensión de 24 x 24 pixeles. Fueron comparados dos modelos, el de Hodkin-Huxley y el modelo de Izhikevich. La arquitectura utilizada para la red neuronal consta de dos capas, una capa de entrada con un número igual al número de pixeles en la imagen y una de salida con un número igual al número de imágenes entrenadas, con una arquitectura *feedforward*, con todas las neuronas de la capa de entrada conectadas a todas las neuronas de la salida.

Una imagen es presentada en la capa de entrada y después de pasar por la red, una sola neurona de las que componen la capa de salida deberá disparar, dejando a las demás inactivas, identificando a la imagen correspondiente. La corriente de entrada hacia la capa de salida de la red es calculada utilizando el tren de pulsos generado por la capa de entrada. En este caso el método de entrenamiento es la regla STDP [37]. En la figura 2 se muestra en conjunto de caracteres utilizados en el trabajo.

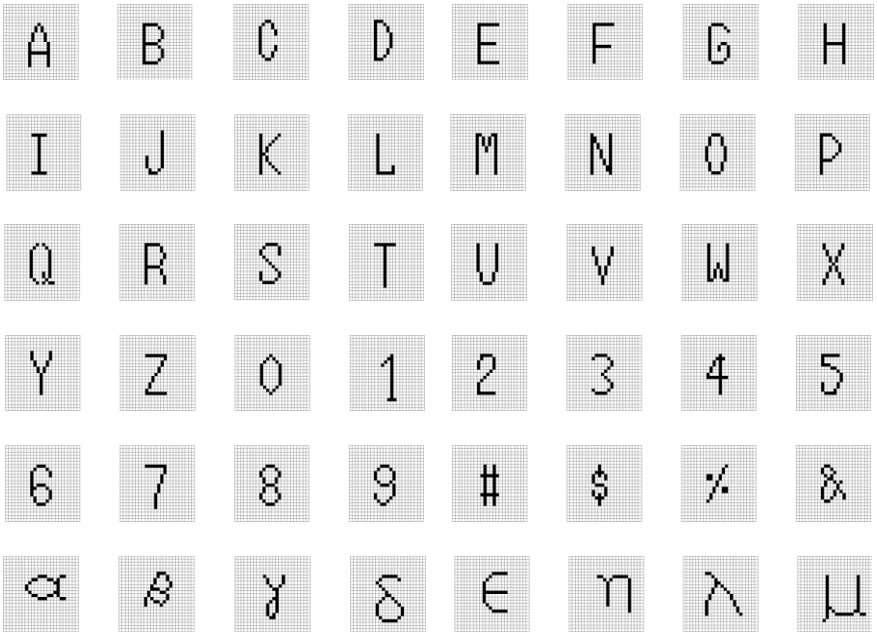


Figura 2. Conjunto de caracteres utilizado en [12] y [13].

Un trabajo que aborda la misma problemática del reconocimiento de caracteres es presentado por Gupta-Long en [19], los autores presentan una Red Neuronal Pulsante compuesta por dos capas, la capa de entrada constituida por un número de neuronas igual al número de pixeles en la imagen a reconocer, la segunda capa tiene un número de neuronas igual a los caracteres que se entrenarán. El set de imágenes utilizado es un conjunto de 48 caracteres, cada carácter es representado por una matriz de 3 x 5 pixeles. Para la construcción se usaron dos modelos neuronales, la capa de entrada está formada por neuronas de tipo *Integrate-and-Fire* mientras que la salida utiliza un modelo *Active Dendrite and Dynamic Synapses* (ADDS). El conjunto de caracteres utilizados en este trabajo es el presentado en la figura 3. El método de entrenamiento de la red neuronal se hizo por medio de STDP.

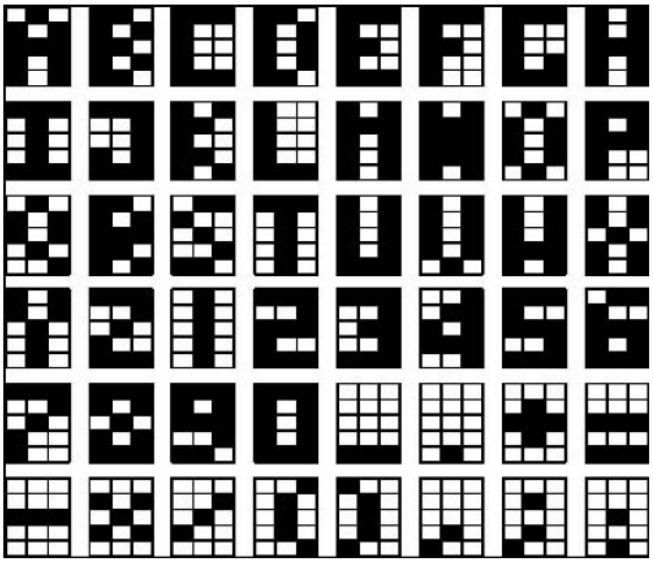


Figura 3. Conjunto de caracteres usados en [19].

En los resultados obtenidos se determinó que la red neuronal es capaz de reconocer la mayoría de los caracteres, ya sea de forma en que una sola neurona dispare ante la presencia de un carácter determinado o bien, por medio de su tasa de disparo. Los autores muestran que en los casos en que se presenta el disparo para una neurona con dos o más caracteres, éstos cuentan con el mismo número de pixeles además de contar con una forma muy similar.

En los trabajos anteriores se demuestra que una Red Neuronal Pulsante con tan sólo dos capas de neuronas es capaz de realizar el reconocimiento de caracteres, por medio de la tasa de disparo haciendo la discriminación entre clases de acuerdo al número de pulsos generados en las neuronas, y también por medio de la inhibición y excitación de neuronas de la capa de salida.

Un avance en cuanto a reconocimiento de objetos utilizando Redes Neuronales Pulsantes es el mostrado en [40], en donde las RNP son utilizadas para el reconocimiento de objetos a través de características de forma obtenidas mediante un brazo mecánico con una mano formada por tres dedos.

El sistema de reconocimiento utiliza tres redes separadas con una arquitectura *feedforward* 11-11-1, una para cada uno de los dedos, y que representan las tres dimensiones del objeto. El modelo neuronal utilizado es el *Leaky Integrate-and-Fire*. Los autores proponen un esquema de codificación de los ángulos obtenidos por cada uno de los grados de libertad de los dedos para convertirlos de valores de punto flotante a enteros. La clasificación se logra a través del método del vecino más cercano en tres dimensiones, tomando como valores la tasa de disparo de cada neurona de salida. Los autores muestran buenos resultados en la tarea del reconocimiento de objetos, tanto para los entrenados como para objetos con la misma forma, pero de diferentes dimensiones, ya sea mayor o menor.

2.2 Análisis de imágenes a través de RNP.

Se presenta en [34] simulaciones en donde se utiliza una RNP de modelo *Integrate-and-Fire* para la detección de líneas rectas. Con la idea de que neuronas individuales de la corteza visual responden a estímulos de luz en campos receptivos de la retina, los autores proponen una RNP en donde su capa de entrada es de las mismas dimensiones a la imagen, una línea está representada por un array de neuronas, en la capa de salida es reconocida por

medio de una neurona perteneciente a una matriz que determina el ángulo y la distancia de la línea al origen, su longitud es dada por la tasa de disparo.

Las conclusiones a las que llegan los autores son que el principio utilizado puede ser usado en sistemas de inteligencia artificial ya que el modelo propuesto puede desempeñar la detección de líneas rectas de forma comparable a la transformada de Hough.

En [43] se presenta un método para la detección de bordes por medio de una RNP de tres capas que usa el modelo de *Integrate-and-Fire* como unidad de procesamiento, la capa de entrada está compuesta por una matriz de neuronas, en donde cada neurona es un pixel de la imagen de entrada. La capa intermedia está dada cuatro matrices de neuronas del mismo tamaño que la capa de entrada encargadas de responder a los bordes con diferentes direcciones (arriba, abajo, izquierda y derecha), esta capa está conectada con la capa de salida por medio de diferentes matrices de pesos, en la capa de salida una neurona integra los resultados de las neuronas intermedias, respondiendo así a un borde en cualquier dirección.

La imagen resultado es obtenida mediante la frecuencia de disparo de las neuronas de salida, en conclusión los resultados son comparables con métodos de detección de bordes como Canny y Sobel. Los autores presentan en otro trabajo resultados utilizando el principio base anterior para la extracción de características de imágenes como los bordes y ángulos rectos [44].

Utilizando el conocimiento que se tiene sobre el sistema visual humano se propone una Red Neuronal Pulsante para realizar la segmentación de imágenes en color en [25]. La arquitectura de la RNP, basada en la retina, está dada de la siguiente forma: una capa de entrada se encarga de extraer un vector de ocho características de color, utilizando los valores de los canales R (Red), G (Green), B (Blue), además de la imagen en escala de grises, de cada pixel;

posteriormente estas características son enviadas a una red de tipo *Backpropagation* que se encarga de realizar la clasificación de las diferentes regiones en la imagen de salida. El modelo neuronal utilizado fue *Integrate-and-Fire*.

En un trabajo posterior, este mismo método es implementado en una arquitectura GPU para acelerar el procesamiento [35]. Los autores concluyen que la extracción de características de color facilita la identificación de áreas en una imagen, además de que el modelo RNP puede ser adecuado para la extracción de otro tipo de características como la textura y la forma.

Capítulo 3
Marco teórico

En el presente capítulo de describirán los elementos básicos de las herramientas utilizadas en el desarrollo del proyecto, iniciando con un acercamiento a las Redes Neuronales Artificiales y algunos de los diferentes modelos propuestos en la literatura, seguido de la descripción de funciones capaces de realizar la transformación de un conjunto de datos a dimensiones superiores a la original, posteriormente se hace una introducción a los métodos más utilizados para el umbralado de imágenes.

3.1 Redes Neuronales Artificiales.

Las Redes Neuronales Artificiales se basan en los conocimientos que existe del comportamiento y función del cerebro humano, en particular del sistema nervioso, el cual está compuesto por redes de neuronas biológicas que poseen bajas capacidades de procesamiento, sin embargo toda su capacidad se sustenta en la conectividad de estas [8], teniendo en cuenta esto, podemos definir una Red Neuronal Artificial como una estructura compuesta por unidades básicas llamadas neuronas, estas se encuentran interconectadas mediante distintos pesos que le dan la capacidad de almacenar su conocimiento experimental y hacerlo utilizable [5] y [45].

3.1.1 Neuronas biológicas.

Las neuronas biológicas presentan estructuras morfológicas particulares que soportan sus funciones, están compuestas por tres partes principales: un cuerpo celular o soma; una o varias prolongaciones cortas que generalmente transmiten impulsos al soma celular, denominadas dendritas; y una prolongación larga, denominada axón que conduce los impulsos desde el soma hacia otra neurona [1]. En la figura 4 se detalla la estructura básica de una neurona.

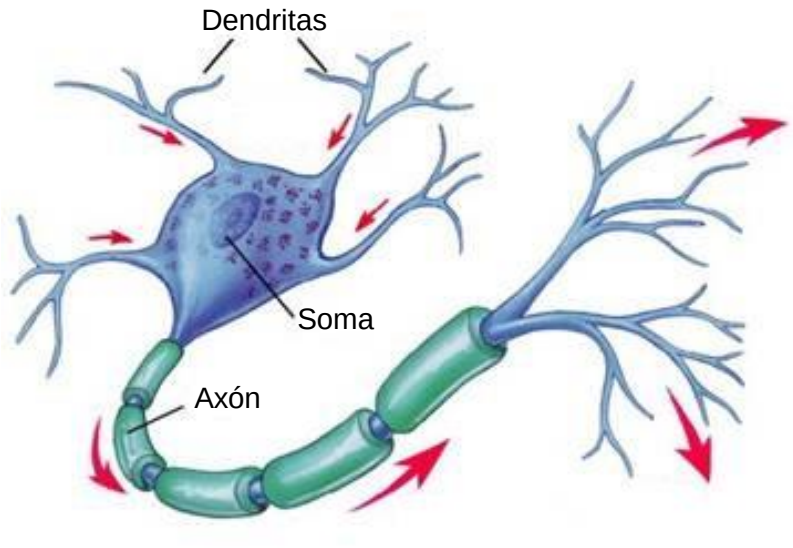


Figura 4. Estructura básica de una neurona biológica.

Como se mencionó anteriormente la neurona recibe información a través de las dendritas, esta información se presentan como un cambio en el potencial eléctrico en ellas; a estos cambios del potencial generalmente se les da el nombre de potenciales postsinápticos. El cuerpo celular se encarga de realizar la suma temporal y espacialmente y cuando se alcanza un cierto umbral la neurona genera un potencial de acción, este potencial es un impulso eléctrico que se propaga por el axón hasta los terminales axónicos, que a su vez llevarán estas señales a las dendritas de la neurona siguiente.

3.1.2 Neuronas artificiales.

La unidad principal de una RNA es un procesador elemental llamado neurona que posee una capacidad limitada de cálculo, McCulloch y Pitts en 1943 propusieron un modelo abstracto y simple de una neurona artificial, en general, la operación de esta neurona se basa en una suma ponderada de sus entradas seguida por la aplicación de una función de activación para obtener una señal que será transmitida a la próxima neurona En la figura 5 se presenta la estructura de una neurona artificial.

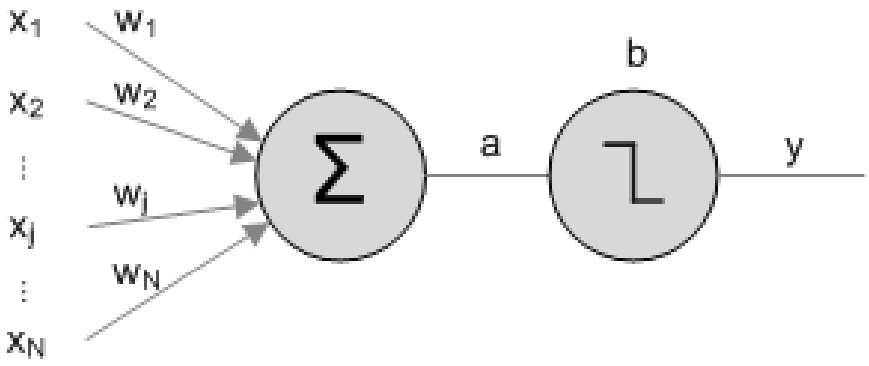


Figura 5. Esquema de una neurona artificial.

El modelo está compuesto por un vector de pesos $\bar{w} = (w_i \dots w_n)^T$ equivalente a las conexiones sinápticas en una neurona real, y un vector de entrada $\bar{x} = (x_1 \dots x_n)$, que corresponde a los estímulos externos. Al realizar el ajuste de los pesos sinápticos mediante algoritmos de aprendizaje se pueden obtener salidas deseadas ante entradas específicas, por lo general binarias [7] y [39].

3.2 Tres generaciones de redes neuronales.

De acuerdo a la unidad de procesamiento, los modelos neuronales pueden clasificarse en tres generaciones, en donde la primera generación está compuesta por los modelos que utilizan la neurona de McCulloch-Pitts, también llamadas perceptrón o compuerta de umbralado. La segunda generación está

compuesta por modelos que utilizan funciones de activación y cuentan con conjunto continuo de posibles salidas y la tercera está dada por las Redes Neuronales Pulsantes [7].

A continuación se presentan las características principales de cada una de las generaciones antes mencionadas [57].

Primera generación: La más simple unidad de procesamiento de las redes tradicionales artificiales, está basada en el modelo neuronal de McCulloch-Pitts. Una neurona de este tipo, con valores reales en sus pesos $\alpha_{i,j}$ y un umbral ϑ recibe como entrada un vector de n números binarios o reales $x_i, \dots, x_i$; mientras que su salida está dada por:

$$\begin{cases} 1\,, si \sum_{j=1}^{n} \alpha_{i,j} \cdot x_j \geq \vartheta \\ 0\,, \quad de\ otro\ modo \end{cases}$$

Segunda generación: Como se mencionó anteriormente, la segunda generación comprende los modelos que utilizan una función de activación como la función sigmoidea y la función de saturación lineal, al utilizar funciones de activación no sólo se pueden realizar operaciones con salidas análogas, sino que también se incrementa el poder computacional para realizar operaciones con salidas booleanas. La forma en que se da la salida de los modelos de segunda generación, con una función continua está dada por:

$$g\left(\sum_{j=1}^{n} \alpha_{i,j} \cdot x_j - \vartheta\right)$$

Otra de las cualidades de esta generacion de redes neuronales, es que soportan algoritmos de aprendizaje que están basados en el desenso del gradiente. La figura 6 muestra ejemplos de funciones de activación.

Figura 6. Ejemplo de funciones de activación utilizadas en los modelos neuronales de segunda generación.

Tercera de generación: Esta generación de modelos neuronales, incrementan el nivel de realismo en una simulación neuronal, al incorporar el concepto de tiempo [9]. Por otro lado, la evidencia experimental que se ha acumulado durante los últimos años, indica que muchos sistemas neuronales biológicos utilizan la temporización de los potenciales de acción individuales ("pulsos") para codificar la información [7], con base en estos descubrimientos se han desarrollado distintos modelos neuronales pulsantes. La figura 7 muestra una respuesta común de 30 neuronas biológicas, en forma de patrones de pulsos en el tiempo.

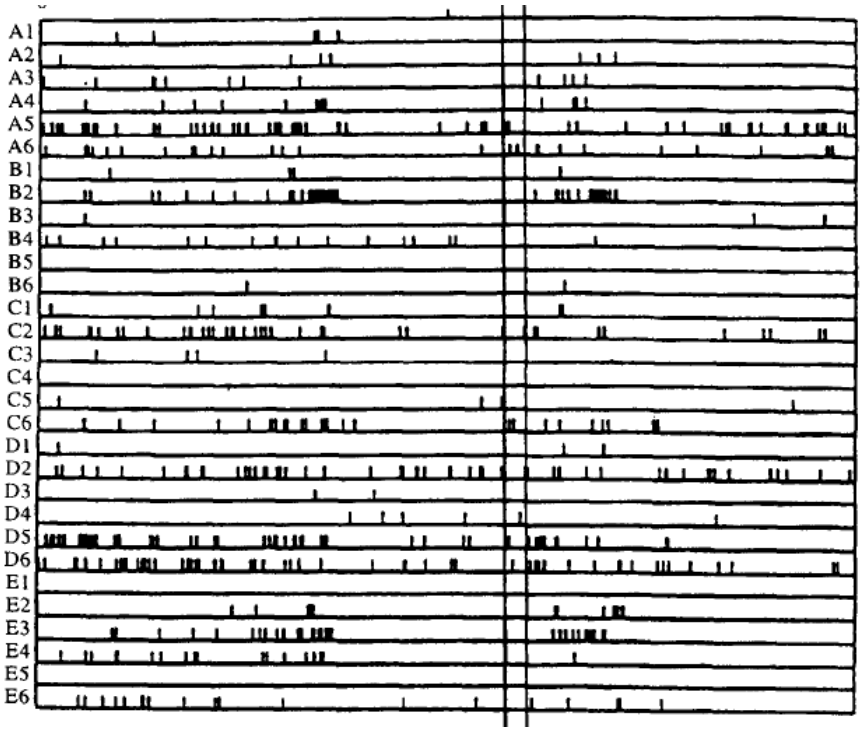


Figura 7. Respuesta en el tiempo de neuronas biológicas, simulación hecha durante 400 ms [57].

Desde un punto de vista matemático una red de neuronas pulsantes realiza un mapeo de una serie de tiempo $\langle \mathcal{F}_i \rangle_{i \in I_{input}}$ en otra $\langle \mathcal{F}_i \rangle_{i \in I_{output}}$. De esto se deduce que las Redes Neuronales Pulsantes trabajan tanto en su entrada como en su salida con series de tiempo, en lugar de vectores de números como en las redes convencionales [57].

3.3 Redes Neuronales Pulsantes.

Si de redes neuronales de tercera generación hablamos, los comienzos de éstas se remontan al año de 1952 cuando Hodgkin y Huxley desarrollaron un modelo matemático que describe la dinámica neuronal en términos de la activación e inactivación de conductividad de voltaje; uno de los resultados obtenidos por estos dos investigadores es que las neuronas son sistemas dinámicos, ya están constituidos por variables que describen su estado y cuentan con una ley que describe la evolución del estado de las variables en el tiempo [1].

Tomando como base los descubrimientos realizados por Hodgkin y Huxley han surgido numerosos modelos neuronales que conforman la tercera generación de redes neuronales [7] y que reproducen las dinámicas de los diferentes tipos de neuronas reales y su comportamiento en la codificación de información en forma de pulsos en el tiempo (patrones de disparo) [45], (para conocer mejor las características de los diferentes tipos de dinámicas revisar [3]).

Una de las principales características de estos modelos es que aumentan el nivel de realismo en la simulación al incorporar el concepto de tiempo durante el procesamiento de la información entrante [1] y [3]. En la figura 8 se muestran los patrones de disparo encontrados en las neuronas biológicas, en donde se pueden apreciar las diferentes formas en que se codifica la información de salida, tanto en frecuencia como en diferencia temporal de los potenciales de acción.

Dentro de los modelos más conocidos capaces de reproducir algunos de los patrones de disparo presentados en la figura anterior se encuentran: *Integrate-and-Fire, Integrate-and-Fire con adaptación, Integrate-and-Fire-or-Burst, Resonate-and-Fire, Quadratic-Integrate-and-Fire, Spiking Model by Izhikevich, FitzHugh-Nagumo, Hindmarsh-Rose, Morris-Lecar, Wilson Polinomial Neurons y Hodgkin-Huxley*.

Todos los modelos listados anteriormente pueden ser expresados en forma de ecuaciones diferenciales ordinarias [3]. La figura 9 muestra la capacidad de cada uno de los modelos para reproducir los patrones de disparo presentados en la figura 8.

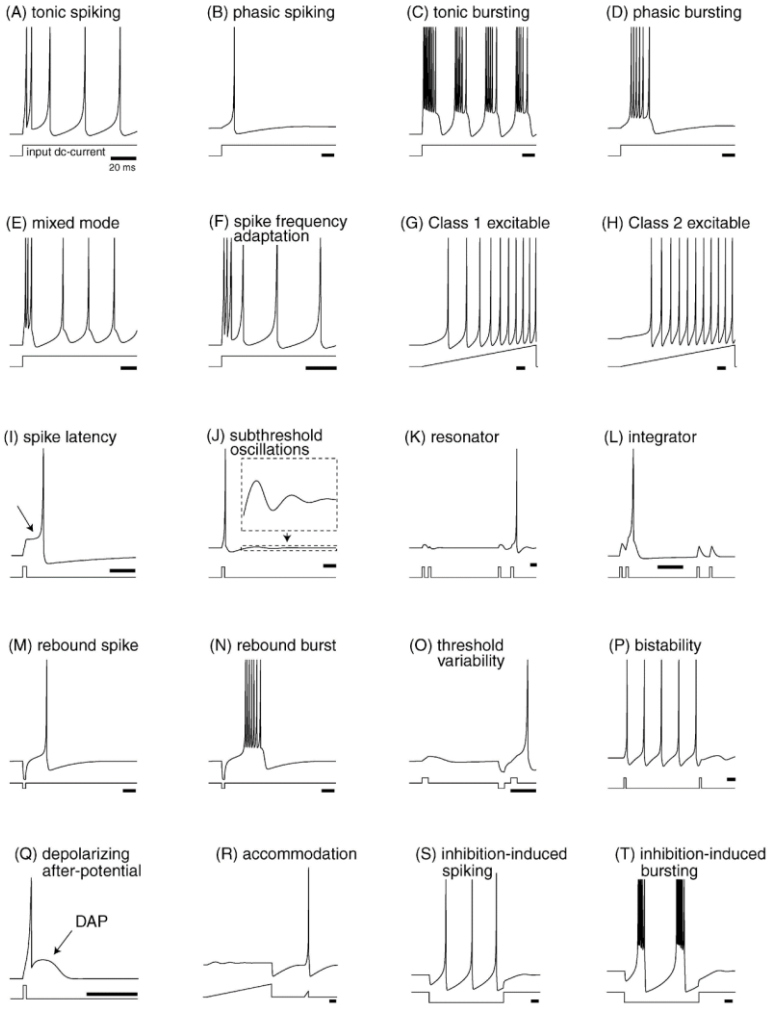


Figura 8. Principales dinámicas en la respuesta de neuronas biológicas [3] [41].

Models	biophysically meaningful	tonic spiking	phasic spiking	tonic bursting	phasic bursting	mixed mode	spike frequency adaptation	class 1 excitable	class 2 excitable	spike latency	subthreshold oscillations	resonator	integrator	rebound spike	rebound burst	threshold variability	bistability	DAP	accommodation	inhibition-indiced spiking	inhibition-induced bursting	chaos	# of FLOPS
integrate-and-fire	-	+	-	-	-	-	-	+	-	-	-	-	+	-	-	-	-	-	-	-	-	-	5
integrate-and-fire with adapt.	-	+	-	-	-	-	+	+	-	-	-	-	+	-	-	-	-	+	-	-	-	-	10
integrate-and-fire-or-burst	-	+	+		+	-	+	+	-	-	-	-	+	+	+	-	+	+	-	-	-		13
resonate-and-fire	-	+	+	-	-	-	-	+	+	-	+	+	+	+	-	-	+	+	+	-	-	+	10
quadratic integrate-and-fire	-	+	-	-	-	-	-	+	-	+	-	-	+	-	-	+	+	-	-	-	-	-	7
Izhikevich (2003)	-	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	13
FitzHugh-Nagumo	-	+	+	-		-	-	+	-	+	+	+	-	+	-	+	+	-	+	+	-	-	72
Hindmarsh-Rose	-	+	+	+			+	+	+	+	+	+	+	+	+	+	+	+	+	+		+	120
Morris-Lecar	+	+	+	-		-	-	+	+	+	+	+	+	+		+	+	-	+	+	-	-	600
Wilson	-	+	+	+			+	+	+	+	+	+	+	+	+	+		+	+				180
Hodgkin-Huxley	+	+	+	+			+	+	+	+	+	+	+	+	+	+	+	+	+	+		+	1200

Figura 9. Capacidad para reproducir patrones de disparo de los modelos de neuronas pulsantes [3].

La figura 10 muestra la gráfica costo-plausibilidad biológica de cada uno de los modelos mencionados tomando en cuenta un número aproximado de flops necesarios para la simulación del modelo en un periodo de un milisegundo.

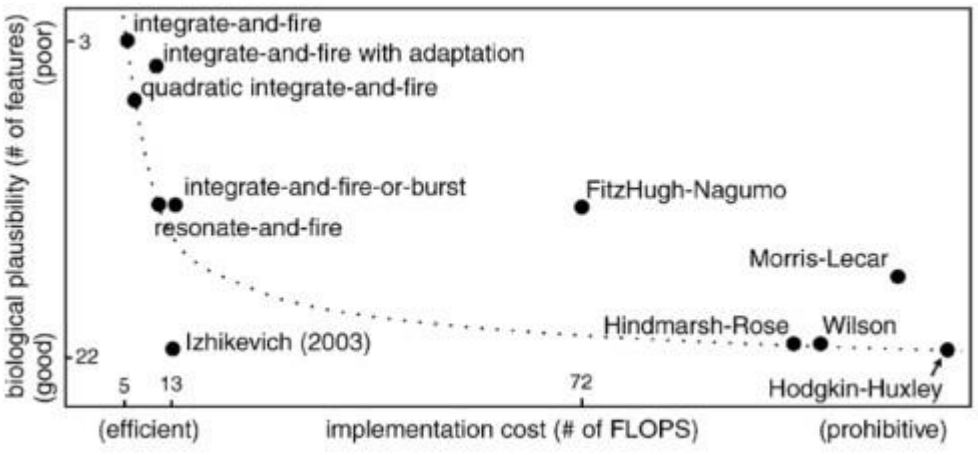


Figura 10. Comparación costo-plausibilidad de los diferentes modelos de neuronas pulsantes [3].

Como se puede apreciar en la figura anterior, mientras que el modelo *Integrate-and-Fire* presenta un bajo costo computacional, convirtiéndose en el modelo más eficiente, también presenta una baja plausibilidad biológica, en comparación, el modelo más cercano a la realidad es el propuesto por Hodgkin y Huxley, ya que puede reproducir la mayoría de los patrones de disparo, pero cuenta con la desventaja de ser muy caro computacionalmente hablando.

Una buena opción para realizar simulaciones con bajo costo y sin perder el comportamiento realista es el modelo propuesto por Izhikevich, que como se observa, tiene la capacidad de reproducir todos los patrones de disparo presentados en la figura 8, además de contar con una eficiencia comparable al modelo *Integrate-and-Fire* [3], [22], [38] y [41].

3.3.1 Modelo neuronal de Izhikevich.

En 2003 Eugene Izhikevich propone un modelo simplificado de neurona pulsante capaz de reproducir muchos de los patrones de disparo generado por las neuronas de los mamíferos. El modelo basa su comportamiento en un sistema de dos ecuaciones diferenciales ordinarias con una serie de parámetros que permiten ajustar el comportamiento del modelo y la forma de disparo, las ecuaciones que definen el modelo son las que se muestran a continuación [41] y [54]:

$$C\dot{v} = k(v - v_r)(v - v_t) - u + I$$
$$\dot{u} = a\{b(v - v_r) - u\}$$

Con la condición:

$$\text{si } v \geq v_{peak}, \text{entonces}$$
$$v \leftarrow c, u \leftarrow u + d$$

El modelo cuenta con nueve parámetros adimensionales que regulan el comportamiento y respuesta del modelo neuronal. A continuación se listan los parámetros utilizados en el modelo de Izhikevich.

$a = Constante\ de\ tiempo\ de\ recuperación.$

$u = Variable\ de\ recuperación, es\ amplificador\ si\ b < 0\ y\ restador\ si\ b > 0.$

$C = Capacitancia.$

$I = Corriente\ aplicada\ al\ modelo.$

$d = parametro\ utilizado\ para\ restablecer\ la\ variable\ \boldsymbol{u}\ depués\ de\ un\ disparo.$

$v_r = valor\ de\ reposo\ de\ la\ membrana.$

$v_t = Voltaje\ del\ umbral\ instantáneo.$

$c = Voltaje\ de\ reinicio.$

$v = potencial\ de\ membrana.$

$k = parámetro\ que\ determina\ la\ forma\ del\ pico.$

Con la selección de diferentes valores para los parámetros se pueden reproducir todas las dinámicas de disparo mostradas en la figura 8, además de los encontrados en neuronas talámicas y corticales [6] y [54]. Estas dinámicas de disparo encontradas en las neuronas del cerebro de los mamíferos pueden ser clasificadas en varios tipos de patrones de disparo, la figura 11 muestra la clasificación mencionada.

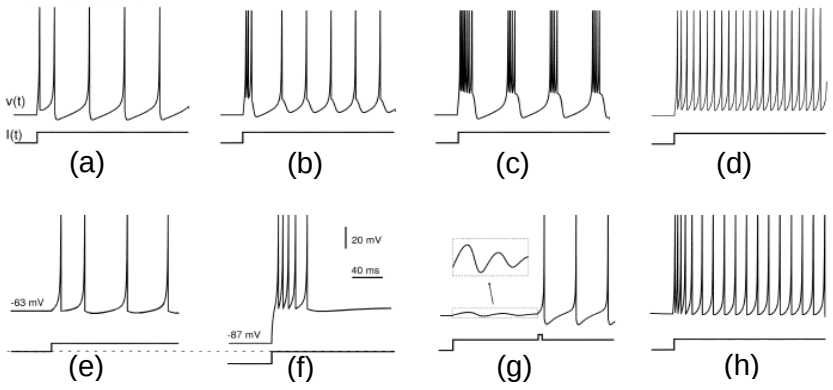


Figura 11. Patrones de disparo obtenidos con diferentes valores en los parámetros. (a) Pulsos Regulares RS, (b) Estallidos Intrínsecos IB, Vibratorio CH, (d) Pulsos Rápidos FS, (e) (f) Tálamo-Cortical TC, (g) Resonador, (h) Pulso de umbral bajo LTS.

De acuerdo al patrón de disparo que se requiera simular, se deben ajustar los valores en los parámetros, los valores sugeridos para poder generar las diferentes clases de respuesta del modelo, mostradas anteriormente, se presentan en la figura 12.

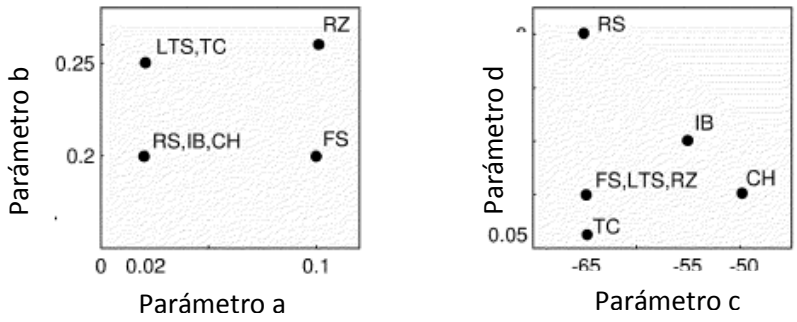


Figura 12. Posibles valores para los parámetros del modelo de Izhikevich según la dinámica de disparo deseada.

3.4 Entrenamiento de una Red Neuronal Artificial.

Teniendo como base las redes neuronales, el aprendizaje o entrenamiento puede ser visto como el proceso de ajuste de los parámetros libres de la red. Partiendo de un conjunto de pesos sinápticos aleatorio, el proceso de aprendizaje busca un conjunto de pesos que permitan a la red, desarrollar correctamente una determinada tarea [47]. El proceso de aprendizaje es un proceso iterativo, en el cual se va refinando la solución hasta alcanzar un nivel de operación suficientemente bueno.

El objetivo del método de entrenamiento es encontrar el conjunto de pesos sinápticos que minimizan (o maximizan) la función de rendimiento de la red. El método de optimización modifica iterativamente los pesos hasta alcanzar el punto óptimo de la red neuronal.

Todos los métodos de aprendizaje pueden ser englobados en dos categorías: aprendizaje supervisado y no supervisado, la selección del adecuado dependerá del problema [48].

3.4.1 Aprendizaje supervisado.

El método de aprendizaje supervisado consiste presentar dos vectores, vector de entrada y el vector de la salida deseada, la salida obtenida por la red neuronal es comparada con la salida deseada y los pesos son modificados con el fin de reducir el error cometido. El proceso se repite de forma iterativa hasta alcanzar un error aceptablemente pequeño entre los valores de entrada y salida. Al tener un n número de pares de este tipo se crea el conjunto de entrenamiento [48].

La mayoría de los algoritmos de aprendizaje supervisado para redes neuronales se basan en el ascenso o descenso del gradiente. Sin embargo para el entrenamiento de RNP no son tan factibles por la discontinuidad natural en el tiempo que presentan este tipos de neuronas [26], a pesar de esto se han desarrollado algoritmos de aprendizaje supervisado que utilizan esta función de descenso del gradiente, estos algoritmos están diseñados para mantener las ventajas de las neuronas pulsantes al mismo tiempo que brinda el poder de aprendizaje de una red neuronal con función sigmoidea [23], [31], [33] y [42].

Por otra parte, investigaciones actuales en el campo de la inteligencia artificial han dado una alternativa más a los mecanismos de entrenamiento supervisado, por medio de métodos evolutivos, aprovechando la habilidad de éstos para trabajar con números reales sin la necesidad de esquemas de codificación binaria compleja [26] y [32].

3.4.2 Aprendizaje no supervisado.

Es un modelo más cercano al sistema biológico, en este caso no se utiliza el vector de salida esperada y sólo hay vectores de entrada en el conjunto de entrenamiento. El algoritmo modifica los pesos de forma que las salidas sean consistentes, es decir, que a entradas muy parecidas, la red neuronal entregue

la misma salida [48]. Debido a la naturaleza del modelo, este enfoque no es adecuado para entrenar tareas que requieren de un objetivo definido [26].

3.5 Algoritmo de Evolución Diferencial.

El algoritmo de Evolución Diferencial pertenece al grupo de algoritmos evolutivos, es un método paralelo de búsqueda directa que utiliza NP vectores de parámetros en D-Dimensiones. Es un método para la solución de problemas de optimización principalmente en espacios de búsqueda continuo [20] [39] [49]. A continuación se describe la estrategia básica utilizada por el algoritmo de Evolución Diferencial, la descripción fue tomada directamente del artículo original [49].

La población del vector inicial se elige al azar y debe cubrir la totalidad del espacio de parámetros. Evolución Diferencial genera nuevos vectores de parámetros mediante la adición de la diferencia ponderada entre dos vectores de población a un tercer vector, esta operación es comúnmente llamada mutación. Los parámetros del vector mutado se mezclan entonces con los parámetros de otro vector llamado vector objetivo, seleccionado aleatoriamente, para producir el llamado vector de ensayo, la operación de mezcla se refiere a menudo como "cruza”. Si el vector de ensayo obtiene un valor de función de aptitud más bajo que el vector objetivo, sustituye a este último en la siguiente generación. Esta última operación se denomina selección.

Para dar una idea más clara de las operaciones realizadas por el algoritmo de evolución diferencial utilizado para el entrenamiento en este trabajo se dan las descripciones detalladas de cada uno de los procesos realizados a lo largo de la ejecución.

3.5.1 Mutación.

Para cada vector $x_i, G, i = 1,2,3, \dots, NP$, un vector mutante es generado de acuerdo con:

$$v_i, G + 1 = x_{r1}, G + F \cdot (x_{r2}, G - x_{r3}, G)$$

Con los índices aleatorios $x_{r1}, x_{r2}, x_{r3} \in \{1,2, \dots, NP\}$ como enteros diferentes, además de ser diferente de i, por tanto, NP debe ser mayor o igual a cuatro para que se cumpla la condición. La constante $F > 0$, es real y se encuentra en un rango de [0, 2], se encarga de la amplificación de la variación diferencial $(x_{r2}, G - x_{r3}, G)$.

3.5.2 Cruza.

Con el fin de aumentar la diversidad de los vectores de parámetros perturbados, se introduce la operación de cruza. Para este fin, el vector de ensayo

$$u_i, G + 1 = (u_{1i}, G + 1, u_{2i}, G + 1, \dots, u_{Di}, G + 1)$$

Es formado siguiendo la regla:

$$u_{ji}, G + 1 = \begin{cases} v_{ji}, G + 1 & \text{if } (randb(j) \leq CR) \; or \; j = rnbr(i) \\ x_{ji}, G & \text{if } (randb(j) > CR) \; or \; j \neq rnbr(i) \end{cases}$$
$$j = 1, 2, \dots, D$$

Donde $randb(j)$ es la j-esiva evaluación de un generador de números aleatorios uniformes con el resultado $\in [0, 1]$. CR es la constante de cruce $\in [0, 1]$, que tiene que ser determinado por el usuario. $rnbr(i)$ es un índice elegido al azar $\in 1, 2, \dots, D$, que asegura el vector de parámetros $u_i, G + 1$ recibe al menos un parámetro de $v_i, G + 1$.

3.5.3 Selección.

Para decidir si debe o no ser miembro de la generación de, $G+1$ el vector de ensayo $u_i, G+1$ se compara con el vector objetivo x_i, G. Si el vector $u_i, G+1$ obtiene un valor de la función costo menor que x_i, G; entonces $u_i, G+1$ se establece en $x_i, G+1$, de lo contrario, el valor antiguo x_i, G se conserva.

3.6 Umbralado de imágenes.

El umbralado es una técnica en el tratamiento digital de imágenes que permite, entre otras cosas, la separación de los objetos de interés con respecto al fondo. La selección de la estrategia más adecuada para cada caso se hace critica dependiendo del objeto principal del procesamiento siendo el tiempo de ejecución una de las mayores limitantes [11] y [52]. La figura 13 muestra el resultado de aplicar el umbralado a una imagen en escala de grises.

Figura 13. Objetivo principal del umbralado, izquierda: Imagen original, derecha: Imagen umbralada.

En el análisis de imágenes es de vital importancia la selección de un adecuado nivel de gris para realizar el umbralado. En la actualidad se han propuesto diversas técnicas, modelos y algoritmos de selección del valor de umbral optimo aun en condiciones de mal iluminación, hasta los modelos más complejos basados en el funcionamiento del sistema visual de los mamíferos [2], [51], [52] y [55].

Los métodos clásicos se pueden categorizar como se indica a continuación [27]:

- Métodos basados en basados en el histograma de la imagen. En donde a partir del histograma de una imagen es posible obtener un umbral de comparación para el agrupamiento de los píxeles.
- Métodos basados en la detección de discontinuidades. En donde la imagen es dividida a partir de cambios bruscos de los niveles de gris.
- Métodos basados en la propiedad de similitud de los valores de los niveles de gris. En donde se usan criterios de homogeneidad para la agrupación de pixeles.

De las técnicas encontradas en el primer grupo, uno de los métodos más utilizados en la literatura es el método de Otsu, llamado así en honor a Nobuyuki Otsu que lo inventó en 1979, utiliza técnicas estadísticas, para resolver el problema.

El método utiliza la variancia, que es una medida de la dispersión de valores, en este caso se trata de la dispersión de los niveles de gris, el histograma; calcula el valor umbral T, de forma que la dispersión dentro de cada segmento sea lo más pequeña posible, pero al mismo tiempo la dispersión sea lo más alta posible entre segmentos diferentes. Para ello se calcula el cociente entre ambas variancias y se busca un valor umbral para el que este cociente sea máximo [53].

Otros métodos para encontrar el umbral se detallan en el método de Kittler [58] y en el de Sahoo [59]. La figura 14 presenta un histograma en donde se observa que existe un valor umbral T que permite la separación de las dos zonas presentes en una imagen.

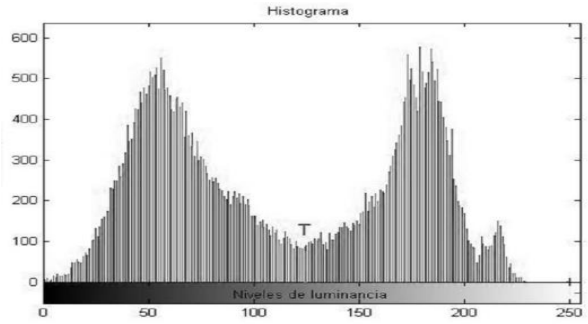


Figura 14. Histograma de una imagen en escala de grises.

3.7 Textura.

La textura es una característica utilizada en la identificación de objetos o regiones de interés en una imagen. En 1973 Haralick propuso un conjunto de 14 medidas de textura basadas en la dependencia espacial de los tonos de grises. Dicho de otra forma, la textura de una imagen es una cuantificación de la variación espacial de valores de tono que es imposible definirlo, precisamente por su carácter sensorial [21].

El uso de la textura de una imagen proviene de la habilidad de los humanos de reconocer diferencias texturales. Se han sugerido variables de textura basadas en estadísticas de 1er. orden (media, desviación estándar, varianza), estadísticas de 2do. orden, basadas en la matriz de Co-ocurrencia, entre las más usadas para medir la textura.

3.7.1 Matriz de Co-ocurrencia.

El método más comúnmente utilizado para medir matemáticamente la textura, es la matriz de Co-ocurrencia de niveles de grises, basada en estadísticas de 2do orden. Es un histograma de los niveles de grises de dos dimensiones para un par de pixeles. Esta matriz calcula la probabilidad de distribución conjunta de un par de píxeles. El anexo B proporciona una metodología detallada del cálculo de la matriz de Co-ocurrencia.

Capítulo 4
Metodología

En este capítulo se presenta la metodología utilizada para lograr los objetivos planteados en la tesis, en primer lugar se muestran los valores usados para el funcionamiento de la neurona pulsante, la forma de cálculo de la corriente de entrada hacia las neuronas y el método para determinar la clase perteneciente al vector de entrada por medio de la frecuencia de disparo. Posteriormente se detalla la metodología seguida para el caso de reconocimiento de patrones y finalmente el análisis de imágenes.

4.1 Parámetros, dinámica y clasificación usando la neurona de Izhikevich.

Como se mencionó en la sección 3.2.1 el modelo neuronal de Izhikevich está basado en un par de ecuaciones diferenciales ordinarias y nueve parámetros adimensionales que son los responsables del comportamiento de la neurona.

$$C\dot{v} = k(v - v_r)(v - v_t) - u + I \qquad \text{si } v \geq v_{peak} \text{ entonces}$$
$$\dot{u} = a\{b(v - v_r) - u\} \qquad v \leftarrow c, u \leftarrow u + d$$

Es importante recordar que en el modelo presentado anteriormente la variable I representa una corriente inyectada de manera directa a la neurona, por lo tanto es considerada como la entrada al modelo.

Por otra parte la variable v representa el potencial de membrana de la neurona, de esta forma, es considerada como la variable de salida del modelo ya que contiene la respuesta en el tiempo de la neurona al estímulo de entrada. Para entender mejor el funcionamiento del modelo dirigirse a la figura 15, en donde se muestra que la corriente de entrada I es calculada mediante una transformación aplicada a un vector de entrada $x_i \in \mathbb{R}^n$.

Después de realizar esta operación, se tiene un número real que representa la corriente de entrada, inyectada directamente en el modelo, y que provocará que el potencial de membrana v comience a cambiar su valor en función de los parámetros del modelo y de la variable de recuperación u, de tal forma que cuando v alcance el valor de corte de pico, dado por v_{peak}, se considera que la neurona ha generado un pulso.

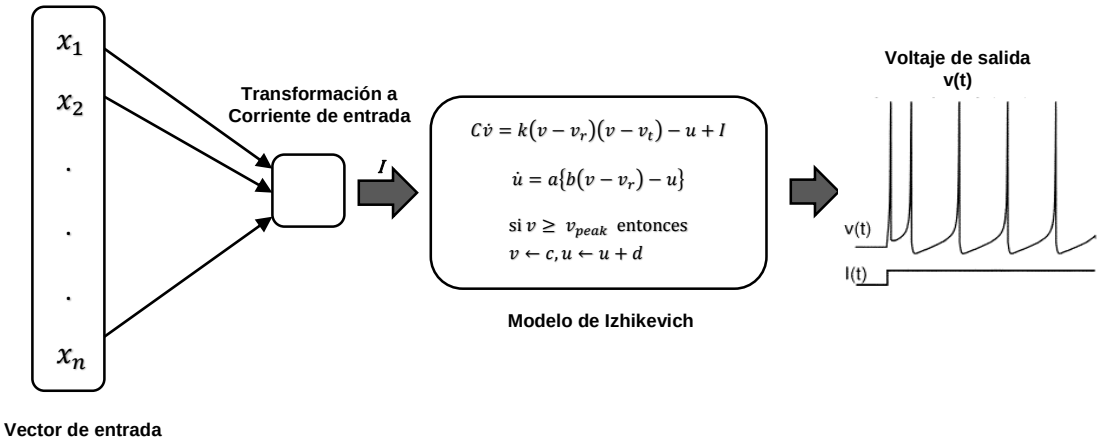


Figura 15. Funcionamiento del modelo de Izhikevich.

Cabe hacer mención que la forma del pulso generado no es importante para nuestra aplicación, sin embargo lo que nos interesa en realidad es el número de pulsos generados (o disparados) durante el intervalo de tiempo, frecuencia de disparo, ya que esta es una forma sencilla de trabajar con el modelo ya que sólo se utiliza parte de la información que está contenida en la respuesta en forma de tren de pulsos en el tiempo.

Para realizar el conteo de los pulsos generados por la inyección de cierta corriente de entrada, fue necesario agregar una nueva condición al modelo, dicha condición es la encargada incrementar una variable cada vez que el valor de v_{peak} sea rebasado o igualado, es decir, cada vez que se genere un pulso.

El modelo final después de la adición de la condición mencionada es el siguiente:

$$\begin{aligned} C\dot{v} &= k(v - v_r)(v - v_t) - u + I \\ \dot{u} &= a\{b(v - v_r) - u\} \end{aligned} \qquad \begin{aligned} &\text{si } v \geq v_{peak} \text{ entonces} \\ &v \leftarrow c, u \leftarrow u + d \\ &\text{si } v \geq v_{peak} \text{ entonces} \\ &cont \leftarrow cont + 1 \end{aligned}$$

Donde $cont = número\ de\ pulsos\ generados\ (frecuencia\ de\ disparo)$.

4.1.1 Dinámica de la neurona.

Como se mencionó anteriormente, el modelo de Izhikevich es capaz de reproducir la mayoría de las dinámicas mostradas por muchos de los tipos conocidos de neuronas biológicas, esto depende de los valores que tomen sus parámetros, ajustados para reproducir una dinámica de disparo específica.

Para el desarrollo de la presente tesis se optó por utilizar valores para los parámetros del modelo correspondientes al patrón de Picos Regulares (RS), que es el comportamiento presentado por un mayor número de neuronas encontradas en la corteza de los mamíferos [41].

La figura 16 muestra el comportamiento de las neuronas con el patrón de Picos Regulares, en ésta se presenta la respuesta de una neurona piramidal encontrada en la corteza visual, comparada con la respuesta del modelo de Izhikevich. Los valores para los parámetros son mostrados en la tabla 1, dichos valores son obtenidos de [41].

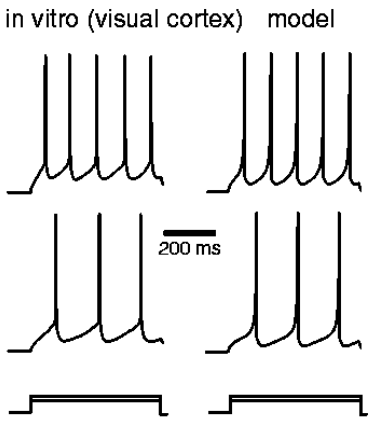


Figura 16. Comportamiento de la neurona en modo de Picos Regulares [54].

Tabla 1. Valores de los parámetros del modelo para la respuesta en forma de picos regulares.

Parámetro	Valor
C	100
v_r	-60
v_t	-40
v_{peak}	35
k	0.7
a	0.03
b	-2
c	-50
d	100
$cont$	0

Tomando en cuenta los valores seleccionados para el modelo, y sustituyéndolos en las ecuaciones se tiene que la forma final del modelo es la siguiente:

$$100\dot{v} = 0.7(v+60)(v+40) - u + I \qquad \text{si } v \geq 35, \text{entonces}$$
$$\dot{u} = 0.03\{-2(v+60) - u\} \qquad v \leftarrow -50, u \leftarrow u + 100$$

$$\text{si } v \geq 35, \text{entonces}$$
$$cont = cont + 1$$

4.1.2 Límites del modelo en corriente aplicada y en frecuencia de pulsos.

Haciendo un análisis para conocer las limitantes del modelo de Izhikevich, en cuanto a pulsos producidos y corriente de entrada, y así conocer la capacidad de respuesta ante una simulación hecha para un intervalo de tiempo de 1 segundo, con las características señaladas en secciones anteriores, se determinaron los valores mínimos y máximos de trabajo.

La figura 17 muestra la respuesta en número de pulsos generados ante una inyección de corriente, dicha corriente fue incrementada con un valor de 100pA iniciando con un valor de 0pA, hasta alcanzar un valor de máximo de 20,000pA (valor seleccionado de manera aleatoria).

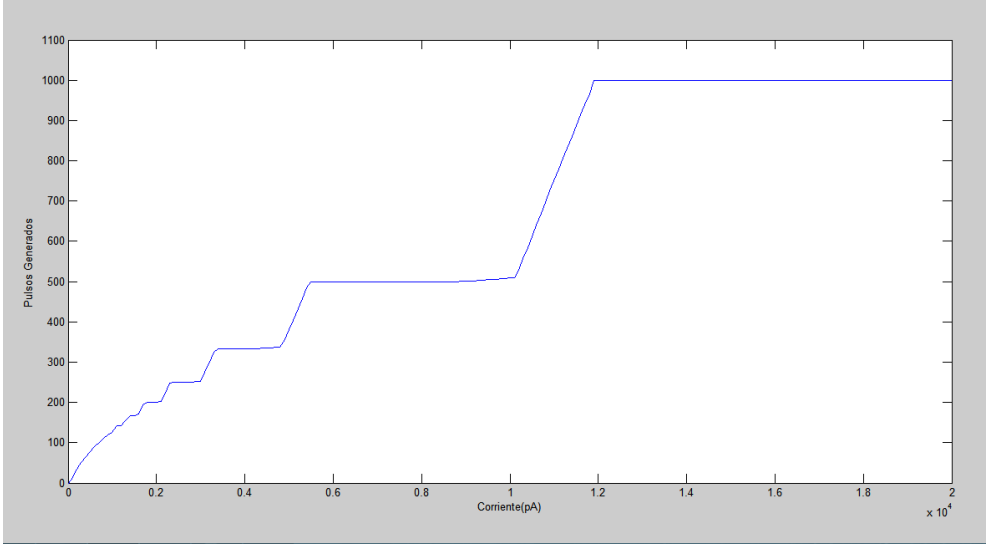


Figura 17. Respuesta en número de pulsos generados ante distintas corrientes de entrada.

De la gráfica anterior se puede deducir que existe un número máximo de pulsos que se pueden generar con el modelo de Izhikevich, con una dinámica de Pulsos Regulares y haciendo una simulación de 1 segundo, con una inyección constante de la corriente de entrada I. Dicho número máximo de pulsos en este caso es de 999, es decir, se puede obtener tan sólo una frecuencia de disparo de 999 pulsos por segundo. Esta primera deducción supone que también

existen límites en cuanto al valor de corriente de entrada que se puede inyectar al modelo; una capaz de activar el modelo (producir al menos un pulso) y otra que alcance el máximo número de pulsos (999 pulsos).

La figura 18 muestra la respuesta ante distintos valores en la corriente de entrada I.

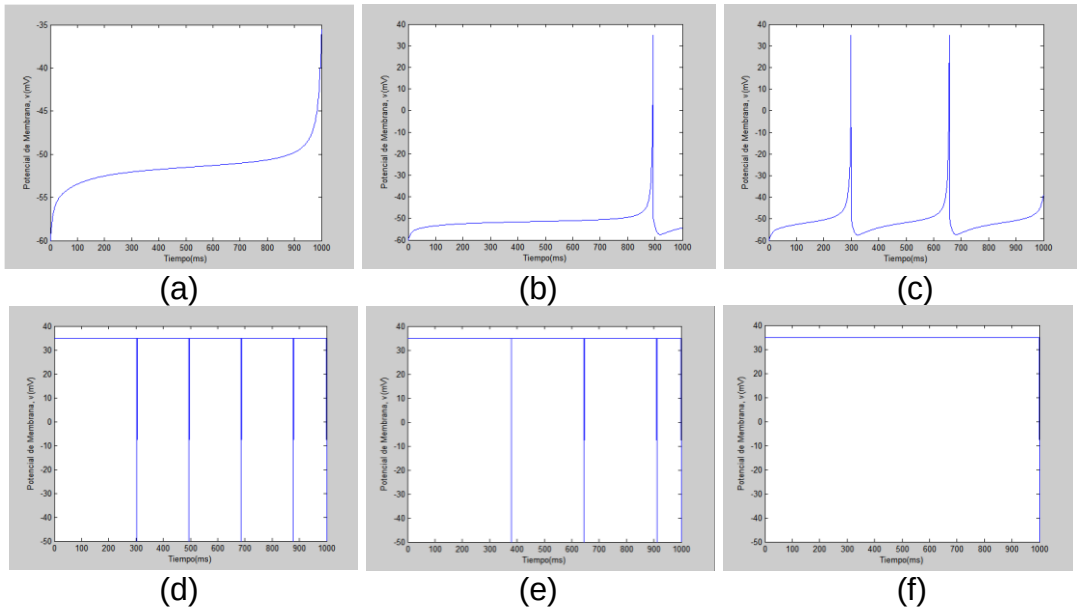


Figura 18. Frecuencia de picos generados a diferentes valores de I por el modelo de Izhikevich con una dinámica RS. (a) $I = 51.8 = 0\ pulsos$, (b) $I = 51.9 = 1\ pulso$, (c) $I = 55 = 2\ pulsos$, (d) $I = 11{,}883.0 = 995\ pulsos$, (e) $I = 11{,}883.3 = 996\ pulsos$, (f) $I = 11{,}883.4 = 999\ pulsos$.

De lo anterior se deduce que los límites para la corriente de entrada hacia una neurona basada en el modelo de Izhikevich, y con una dinámica de disparos de tipo RS están dados por:

$$51.9 \leq I \leq 11{,}883.4$$

Mientras que los límites, en cuanto a frecuencia de disparos en un intervalo de tiempo de 1 segundo se refiere, se encuentran entre:

$$0 \leq fd \leq 999$$

4.1.3 Cálculo de la corriente de entrada hacia el modelo de Izhikevich mediante funciones de transformación.

Como se mencionó en la sección anterior, la neurona pulsante recibe una corriente de entrada I; al tener como patrones de entrada un vector de rasgos, éstos no pueden ser introducidos directamente al modelo de Izhikevich, es necesario convertir este vector de entrada en una corriente única. Esta corriente es calculada a partir de un vector de rasgos de entrada $x_i \in \mathbb{R}^n$ en conjunto con un vector de valores $w_i \in \mathbb{R}^n$, que representa a los parámetros de una función de transformación; gracias a este último es posible obtener diferentes frecuencias de disparo ante distintos vectores de entrada.

Para convertir el vector de entrada $x_i \in \mathbb{R}^n$ a una corriente de entrada I, en el presente trabajo se utilizaron funciones de transformación para los valores de los vectores de entrada, de esta manera, los vectores $x_i \in \mathbb{R}^n$ por medio de la función de transformación y gracias a la sintonización de sus parámetros, son mapeados a un valor escalar que se corresponde con la corriente que activará a la neurona.

Estas funciones de transformación son similares, en cuanto a estructura, a las funciones Kernel utilizadas en las Máquinas de Vector Soporte (MVS) y que facilitan la separación lineal en un espacio de mayor dimensionalidad. Además para los experimentos se utilizó la función propuesta en [10] y [28] a la cual se ha asignado el nombre de función productos. El conjunto de funciones utilizadas en el trabajo es el siguiente:

Función lineal:

$$I = (x \cdot w') + \theta$$

Función polinomial:

$$I = (x \cdot w' + 1)^p + \theta$$

Función gaussiana RBF:

$$I = exp\left(\frac{-\|x - w'\|^2}{2\sigma^2}\right) + \theta$$

Función productos:

$$I = (x \cdot w' \cdot \gamma) + \theta$$

En muchos casos la corriente obtenida del cálculo de la función de transformación no es suficiente para hacer que la neurona pulsante se active, es por esta razón que el parámetro θ cuenta con un valor de 55pA. Al realizar esto, se garantiza que la neurona se activará en todos los casos con al menos dos picos, como se pudo observar en la figura 18, en donde se puede ver que con una corriente continua de 55pA se obtienen dos picos.

Visto de manera general, el proceso de conversión de un vector de rasgos en una frecuencia de disparos mediante la función de transformación, en conjunto con el modelo de Izhikevich, es como muestra la figura 19.

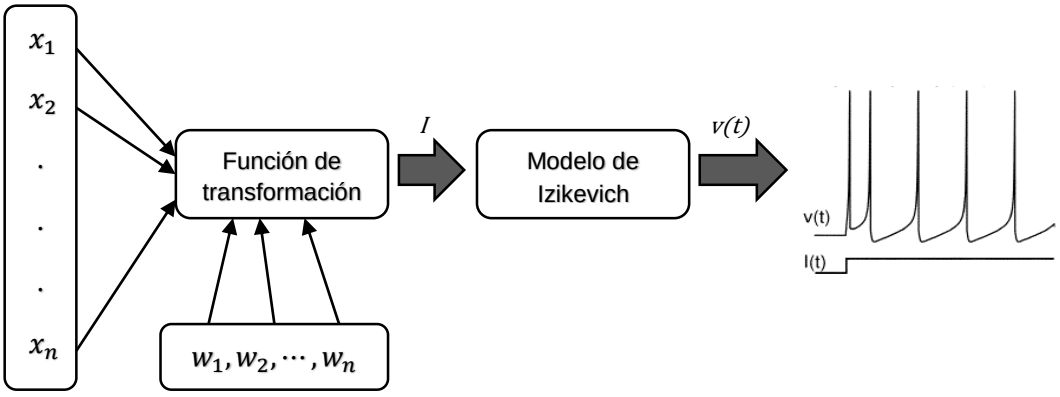


Figura 19. Conversión vector-frecuencia usando el modelo de Izhikevich.

4.1.4 Clasificación por medio de la frecuencia de disparo.

El cálculo de la frecuencia de disparos en la salida del modelo de Izhikevich, teniendo como base un vector de rasgos y un vector de valores para los

parámetros de una función de transformación, puede ser utilizado para resolver problemas de clasificación, como se ha demostrado en [9], [10], [17], [18], [23], [33] y [42].

Visto como un problema de Redes Neuronales Artificiales, se busca ajustar los parámetros de la red por medio de un conjunto de datos de entrenamiento, hasta encontrar un hiperplano capaz de realizar la clasificación exitosa para nuevos datos de entrada.

Para lograr una clasificación, en la presente tesis se buscó que después del ajuste de los parámetros de la transformación en la red neuronal, los vectores de rasgos pertenecientes a la misma clase generaran frecuencias de disparo parecidas o iguales, mientras que vectores pertenecientes a distintas clases generaran frecuencias lo suficientemente distintas, para poder discriminar entre ellas. En la tabla 2 se presenta el ejemplo de la generación de distintas frecuencias de disparo producida por dos clases.

Tabla 2. Frecuencia generada en el entrenamiento de dos clases.

Rasgo 1	**Rasgo 2**	**Clase**	**Frecuencia**
15	15	1	39
50	50	2	250

El cálculo de la corriente de entrada se realizó mediante la función I =rasgo 1 * rasgo 2

Como se observa en la tabla, el conjunto de entrenamiento está dado por dos vectores, uno por cada clase, de esta información se deduce que la frecuencia de disparo representativa de cada clase está dada por el resultado de estos vectores, en caso de tener un mayor número de vectores por clase, será el promedio de las frecuencias generadas por estos.

Posteriormente se hace la clasificación de vectores de rasgos de prueba, teniendo la frecuencia generada por éste, y calculando la distancia euclidiana entre la frecuencia del nuevo vector y las frecuencias generadas por el conjunto de entrenamiento. De esta manera la clase de un nuevo vector de rasgos de

entrada es determinada por medio de la frecuencia de disparo, utilizando la menor distancia euclidiana hacia los promedios de las clases como sigue:

$$clase = \arg \underset{k=1}{\overset{K}{min}} \left(|PF_k - fd|\right)$$

En donde PF_k es el promedio de la frecuencia generada por patrones de la misma clase en el entrenamiento (frecuencia de disparo representativa de la clase), y fd es la frecuencia de disparos generada por el vector de prueba. La tabla 3 presenta un ejemplo de clasificación de vectores de prueba pertenecientes a las clases mostradas en la tabla 2.

Tabla 3. Clasificación de vectores de prueba por medio de la frecuencia de disparo.

Rasgo 1	**Rasgo 2**	**Frecuencia de disparo**	**Distancia Euclidiana.**		**Clasificación**
			Clase 1	Clase 2	
16	15	41	**2**	209	1
17	10	28	**11**	222	1
56	48	251	212	**1**	2
50	52	250	211	**0**	2

Cl.1 = Clase 1, Cl. 2= Clase 2

4.2 Entrenamiento mediante ED.

El entrenamiento de la Red Neuronal Pulsante se decidió hacer mediante el algoritmo de Evolución Diferencial debido a las características que presenta en cuento a convergencia y valores usados. La principal característica por la que se eligió es el poder que tiene de trabajar con valores en los números reales, evitando así la utilización de esquemas binarios complejos.

Dada la naturaleza de los valores que se desean sintonizar en la Red Neuronal Pulsante, $w_i \in \mathbb{R}^n$, el algoritmo de Evolución Diferencial es un buen candidato para realizar la búsqueda de éstos. Además de que muestra buenas propiedades de convergencia y supera a otro tipo de algoritmos evolutivos [39],

por otro lado este algoritmo ha sido probado en el entrenamiento de RNP con buenos resultados.

En el presente trabajo se utilizaron diferentes valores para el algoritmo de ED, de acuerdo al problema que se pretende resolver, en este caso, para entrenar de manera no supervisada cada una de las neuronas de la RNP usada en el reconocimiento de patrones y el entrenamiento no supervisado de la neurona usada para el umbralado de imágenes.

Otra característica que se tomó en cuenta para elegir esté método de entrenamiento es su facilidad para implementarlo de manera paralela, ya que cada uno de los individuos de la población es evaluado de manera independiente y a que la única operación que requiere comunicación entre individuos es la reproducción [39].

4.3 Aplicación del modelo de Izhikevich en el reconocimiento de patrones.

Con el fin de probar que las Redes Neuronales Pulsantes, que utilizan el modelo neuronal de Izhikevich, son capaces de resolver problemas clasificación por medio de la frecuencia de disparo, se realizaron experimentos utilizando distintas bases de datos, siguiendo la metodología que se describe a continuación y que se muestra en la figura 20:

1. Se define la arquitectura para la RNP tomando como base la información de la base de datos; número de rasgos por vector, número de clases.

2. Selección de la función de transformación para el cálculo de la corriente de entrada I hacia las neuronas.

3. Se divide la base de datos en dos conjuntos, entrenamiento y prueba (conjunto de validación).

4. Se hace el entrenamiento de la RNP, usando el conjunto de entrenamiento, por medio del algoritmo de ED para así ajustar los parámetros de la función de transformación, vector $w_i \in \mathbb{R}^n$.

5. Se hace la validación de la Red Neuronal Pulsante usando los datos del conjunto de validación y el vector de valores $w_i \in \mathbb{R}^n$ obtenido en el entrenamiento, para calcular la tasa de clasificación correcta.

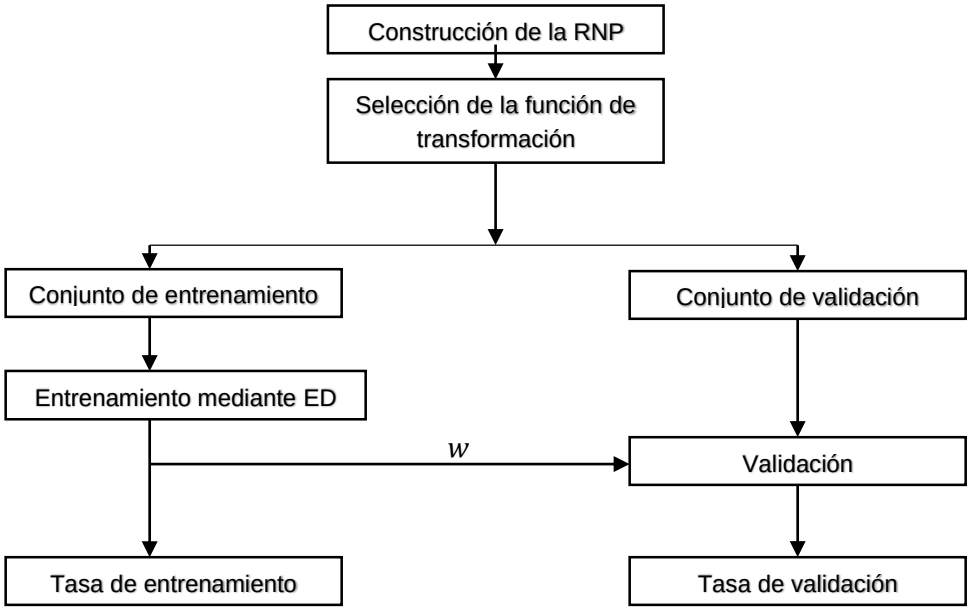


Figura 20. Metodología para los experimentos de reconocimiento de patrones.

Para hacer una validación del método propuesto se realizaron 10 experimentos sobre el conjunto de datos, usando como conjunto de entrenamiento el 50% del total de patrones y para la validación el 50% restante, esta separación se hizo de manera aleatoria para cada una de las repeticiones del experimento.

En las secciones siguientes se detalla la forma de que se construyó la Red Neuronal Pulsante utilizada para cada uno de los conjuntos de datos, basándose en la estructura de la base (número de rasgos por patrón y número de clases), posteriormente se dan los valores utilizados en el algoritmo de Evolución Diferencial para realizar el entrenamiento.

4.3.1 Arquitectura de la red neuronal.

La arquitectura utilizada para desarrollar esta sección es una red neuronal de tipo *feedforward* completamente conectada, en este tipo de redes se empieza

con un vector de entrada el cual es equivalente en magnitud al número de neuronas de la primera capa de la red, las cuales procesan dicho vector elemento por elemento en paralelo. La información, modificada en cada neurona, es transmitida hacia delante por la red pasando por las capas ocultas para finalmente ser procesada por la capa de salida.

La RNP utilizada cuenta con tres capas, a las cuales se les han dado los nombres de *E* (Capa de entrada), *O* (Capa intermedia) y *S* (Capa de salida). El número de neuronas correspondientes a cada capa, para todos los casos presentados en este trabajo, se obtienen siguiendo las tres reglas siguientes:

1. $neuronas\ en\ la\ capa\ E = n\acute{u}mero\ de\ rasgos\ en\ el\ vector\ x.$
2. $neuronas\ en\ la\ capa\ O = n\acute{u}mero\ de\ clases\ de\ la\ base\ de\ datos.$
3. $neuronas\ en\ la\ capa\ S = 1$

Mientras que la corriente de entrada a la neurona E_i es dada por el resultado de aplicar la función de transformación seleccionada al rasgo x_i, en conjunto con los parámetros w_i, el cálculo de las corrientes de entrada para las capas *O* y *S* se realiza utilizando como valores de entrada la frecuencia de disparos, dada por la variable $cont$, de las neuronas de la capa anterior inmediata, en forma de vector en conjunto con un vector de parámetros único para la función de transformación perteneciente cada neurona de la capa.

La capa de salida está formada por una sola neurona para que así, usando la frecuencia única de salida, se realice la clasificación de los patrones de entrada, logrando con esto la clasificación por medio de la frecuencia de disparo obtenida en ésta, además de eliminar la posibilidad de encontrar una activación de múltiples neuronas, como en los casos de RNA donde existen múltiples neuronas en la salida y en donde se espera que sólo una de ellas responda a patrones pertenecientes a cierta clase, como es el caso de las propuestas en [12], [13] y [19].

La función de transformación utilizada para el cálculo de la corriente es la misma para todas las neuronas de la red, y de ésta dependerá el número de parámetros a ajustar por cada neurona. La figura 21 muestra la arquitectura de la RNP vista de manera simplificada, en donde sólo se muestran las capas que conforman la red. La figura 22 muestra la forma en que se distribuye la información a través de las neuronas de distintas capas.

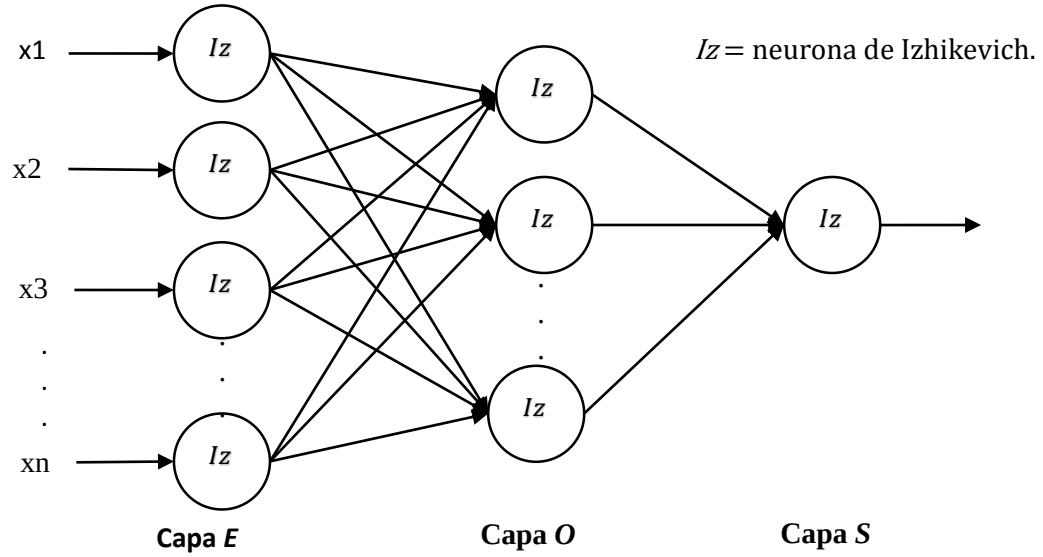


Figura 21. Arquitectura de la RNP utilizada en esta tesis para el reconocimiento de patrones.

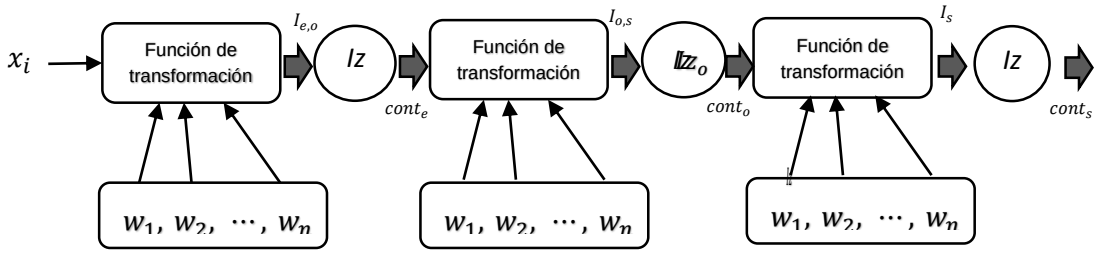


Figura 22. Paso de la información a través de las distintas capas de la RNP.

4.3.2 Valores del algoritmo de Evolución Diferencial.

El entrenamiento de la red neuronal se hace mediante el algoritmo de Evolución Diferencial, aprovechando el poder del algoritmo para realizar búsquedas sobre

espacios continuos y la baja tendencia que tiene de converger en un máximo local [10] y [28].

Los individuos de la población inicial fueron inicializados de manera aleatoria y representan los valores de los parámetros iniciales de la función de transformación de cada una de las neuronas que conforman la RNP. Cada individuo es representado por un vector cuyo tamaño es la sumatoria del número de valores necesarios para cada una de las neuronas que conforman la red neuronal, calculado de la siguiente manera para el caso de la función de transformación lineal y una sola neurona:

$$tamIndividuo = tamX$$

Y para los otros casos:

$$tamIndividuo = tamX + 1$$

Donde $tamX$ es el tamaño del vector de entrada hacia la neurona y el parámetro extra se debe a la forma de la función de cálculo de la corriente.

Como se mencionó anteriormente el número de parámetros totales correspondientes a cada capa de la red puede variar según la estructura del conjunto de entrenamiento (número de rasgos por vector de entrada, número de clases). Los valores de los parámetros utilizados en el algoritmo de ED son los que se presentan en la tabla 4.

Tabla 4. Parámetros utilizados en ED para el reconocimiento de patrones.

Parámetro	**Valor**
NP	60
MAXGEN	500
F	0.9
Cr	0.8
X_MAX	1
X_MIN	0

En donde los parámetros son: NP = Tamaño de población, MAXGEN = Número de generaciones, X_MAX y X_MIN = rangos para la generación de valores aleatorios.

Para obtener una frecuencia de disparo parecida entre patrones de la misma clase, y al mismo tiempo una separación entre clases que sea capaz de hacer una discriminación, se utilizó la siguiente función fitness:

$$f(x) = Pa/Pt$$

Donde Pa es el número de patrones del conjunto de entrenamiento correctamente aprendidos mediante el vector de valores $w_i \in \mathbb{R}^n$ para la función de transformación, propuesto por el individuo x_i, G del algoritmo de ED, y Pt es el número total de patrones contenidos en el conjunto de entrenamiento. El criterio de paro fue cumplir con el número máximo de generaciones

4.4 Umbralado de imágenes usando el modelo de Izhikevich.

La finalidad de una técnica de umbralado es lograr la óptima separación del objeto de interés, del fondo de la imagen. Viendo este problema como un problema de clasificación, la imagen representa un conjunto de datos en el que existen dos clases: el fondo de la imagen y el objeto, clase 0 y clase 1.

Teniendo en cuenta lo anterior se propone utilizar una neurona que usa el modelo neuronal de Izhikevich, capaz de realizar la separación de dos clases, para realizar el umbralado de algunas imágenes en color, utilizando como rasgos descriptores mediciones de color y textura. Para la realización de esta tarea se siguieron los pasos listados a continuación:

1. Teniendo la imagen original, son seleccionados dos segmentos representantes del fondo y del objeto de interés.

2. Usando estos segmentos se forma el conjunto de entrenamiento de la neurona pulsante al extraer rasgos de color y de textura.

3. Selección de la función de transformación usada para el cálculo de la corriente I.

4. Se hace el entrenamiento de la neurona por medio de ED para obtener el vector de valores $w_i \in \mathbb{R}^n$, para los parámetros de la función de transformación.

5. Una vez que se tiene el vector $w_i \in \mathbb{R}^n$, se hace la clasificación de los vectores de rasgos extraídos de la imagen original completa.

6. Se guarda la imagen en formato .bmp, con los valores de clasificación del vector de rasgos al pixel correspondiente.

La figura 23 presenta el diagrama de pasos seguido para realizar el umbralado de imágenes en color utilizando el método propuesto.

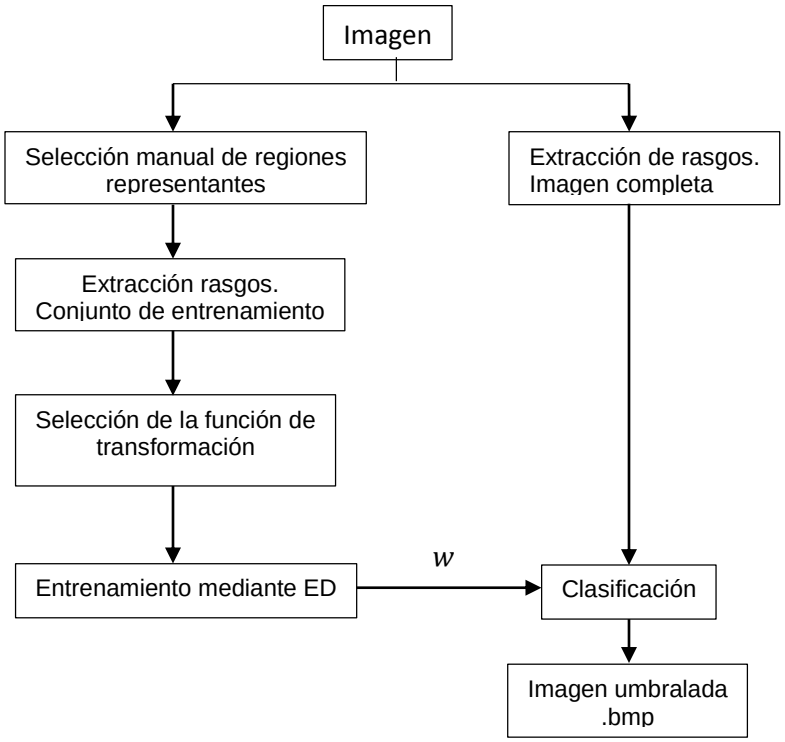


Figura 23. Metodología para realizar el umbralado de imágenes.

4.4.1 Arquitectura de la red neuronal para el umbralado de imágenes.

Para el desarrollo del caso propuesto se utilizará tan sólo una neurona basada en el modelo de Izhikevich. Dicha neurona deberá ser capaz de realizar el aprendizaje de un conjunto de vectores $x_i \in \mathbb{R}^n$ de entrenamiento, pertenecientes a las dos clases contenidas en la imagen, transformados en una corriente I, y generar en su salida una frecuencia de disparos diferentes entre sí para logar una correcta clasificación. La figura 24 muestra una idea grafica del método propuesto.

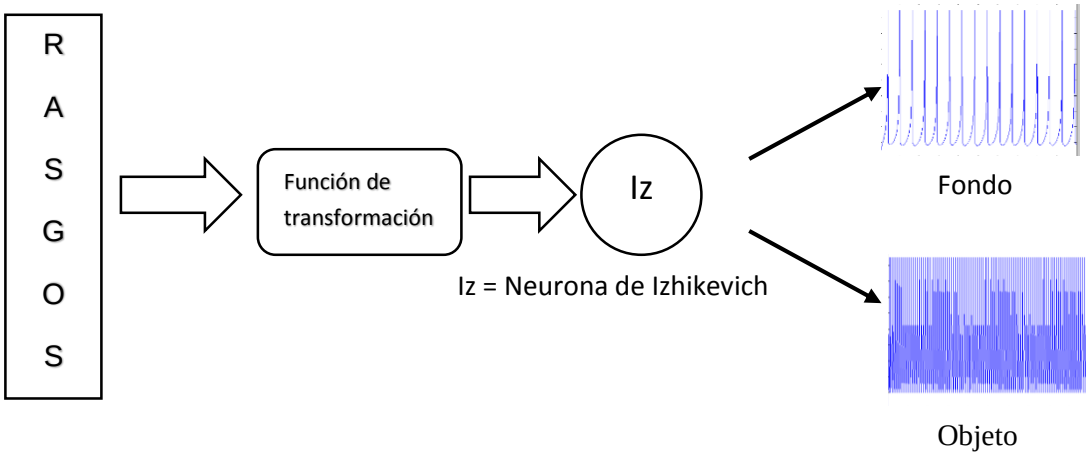


Figura 24. Proceso de umbralado de una imagen visto como un problema de clasificación.

4.4.2 Valores del algoritmo de Evolución Diferencial.

Los individuos de la población inicial fueron inicializados de manera aleatoria y representan los valores de los parámetros iniciales. Cada individuo es representado por un vector, cuyo tamaño es igual al tamaño del vector de rasgos $x_i \in \mathbb{R}^n$ que es presentado a la neurona, y dependiendo de la función de transformación seleccionada para el cálculo de la corriente, el número de valores del vector $w_i \in \mathbb{R}^n$ puede sufrir una adición, de esta forma para el caso de la transformación lineal:

$$tamIndividuo = tamX$$

Y para los otros casos:

$$tamIndividuo = tamX + 1$$

Donde $tamX$ es el tamaño del vector de entrada hacia la neurona y el peso extra se debe a la forma de la función de cálculo de la corriente. Los valores de los parámetros utilizados en el algoritmo de ED son los que se presentan en la tabla 4.

Tabla 5. Parámetros utilizados en ED para el umbralado de imágenes.

Parámetro	Valor
NP	40
MAXGEN	250
F	0.9
Cr	0.8
X_MAX	1
X_MIN	0

En donde los parámetros son: NP = Tamaño de población, MAXGEN = Número de generaciones, X_MAX y X_MIN = rangos para la generación de valores aleatorios.

Para obtener una frecuencia de disparo parecida entre patrones de la misma clase, y al mismo tiempo una separación entre clases que sea capaz de hacer una discriminación, y clasificación correcta de vectores de prueba, se utilizó la siguiente función fitness:

$$f(x) = SD(prom) * \frac{1}{\sum_{k=1}^{K} \frac{sumE_k}{prom_k}}$$

Donde: $SD(prom)$ es la desviación estándar de los promedios de pulsos por clase, $prom_k$ es el promedio de pulsos generados por patrones pertenecientes

a la misma clase y $sumE$ es la sumatoria de las distancias euclidianas entre patrones por clase.

$$sumE = \sum_{x=1}^{X-1} \left(\sum_{z=x+1}^{X} |pulsos_x - pulsos_z| \right) + 1$$

El criterio de paro del algoritmo fue llegar al máximo número de generaciones.

4.4.3 Formación del conjunto de datos de entrenamiento.

El conjunto de entrenamiento está dado por vectores formados por seis rasgos descriptores extraídos cada uno de los pixeles de la imagen, dichos pixeles pertenecen a pequeñas zonas representantes, tanto del fondo como del objeto.

El vector de rasgos descriptores se forma con tres rasgos de color, en este caso son los valores de los canales R (Red), G (Green) y B (Blue) de un pixel p en la imagen a color, y tres rasgos de textura, extraídos de un pixel p_g de la versión en escala de grises de la misma imagen por medio de la matriz de Co-ocurrencia [21], estos rasgos son la homogeneidad (H), el contraste (C) y la disimilaridad (D), en el anexo B se presenta la metodología utilizada para el cálculo de estás tres medidas de textura. La figura 25 muestra cómo se forma un vector de rasgos para un pixel en una imagen.

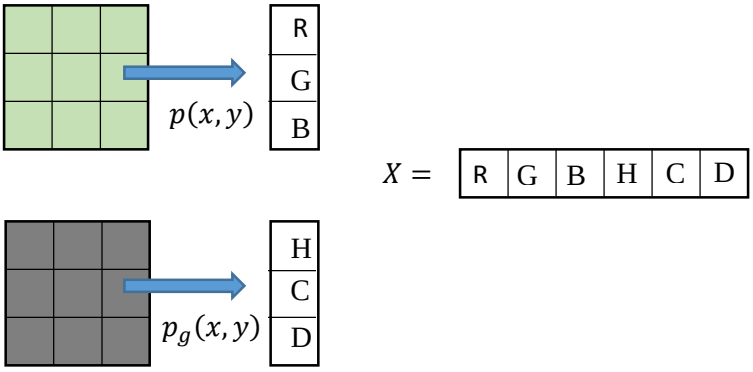


Figura 25. Formación de un vector de rasgos para un píxel.

De esta forma, teniendo dos secciones de la imagen original, una sección representante del objeto de interés y una sección representante del fondo, se forma un conjunto de vectores que se utiliza para el entrenamiento de la neurona pulsante. Posteriormente se extraen los mismos rasgos pero de la imagen completa para así tener una clasificación de la totalidad de los pixeles y obtener como resultado una imagen binaria.

Capítulo 5
Resultados experimentales y discusión

En el presente capítulo se muestran los resultados obtenidos al aplicar la metodología descrita en el capítulo anterior. Se presentan resultados obtenidos para los casos de reconocimiento de patrones y el umbralado de imágenes.

5.1 Implementación en software del método propuesto.

En la implementación de los pasos descritos para resolver los problemas de reconocimiento de patrones y umbralado de imágenes, fue desarrollada en dos partes: el entrenamiento y la validación (o prueba) de la RNP, utilizando los lenguajes C++ y MATLAB respectivamente.

En el anexo C se presentan los códigos de las funciones principales en las aplicaciones para los dos casos, dando detalle de la forma del algoritmo de Evolución Diferencial y de las funciones de aptitud.

5.1.1 Entrenamiento de la RNP para el reconocimiento de patrones.

La programación del algoritmo de ED fue codificado utilizando una aplicación de consola escrita en lenguaje C++, esta aplicación se encarga de realizar la

lectura de un archivo con extensión .dat, en el que se encuentra el conjunto de entrenamiento, y del cual extrae la información para crear la estructura de la RNP, concretamente el número de clases y el número de rasgos que contiene cada vector. Posteriormente es obtenida información utilizada para el cálculo de aptitud, esta información es el número de patrones por clase, la clase a la que pertenece cada patrón y el número de patrones totales.

Después de iniciar el entrenamiento mediante la aplicación, el número de pulsos generados por cada uno de los patrones es mostrado por medio de la consola del sistema. Después de terminado, los valores encontrados por el algoritmo de ED son almacenados en un archivo .dat, que será utilizado posteriormente para la validación. La figura 26 muestra la consola del sistema mostrando la información durante el proceso de entrenamiento de la RNP. Mientras que en el anexo C se muestra el código utilizado para la aplicación de entrenamiento.

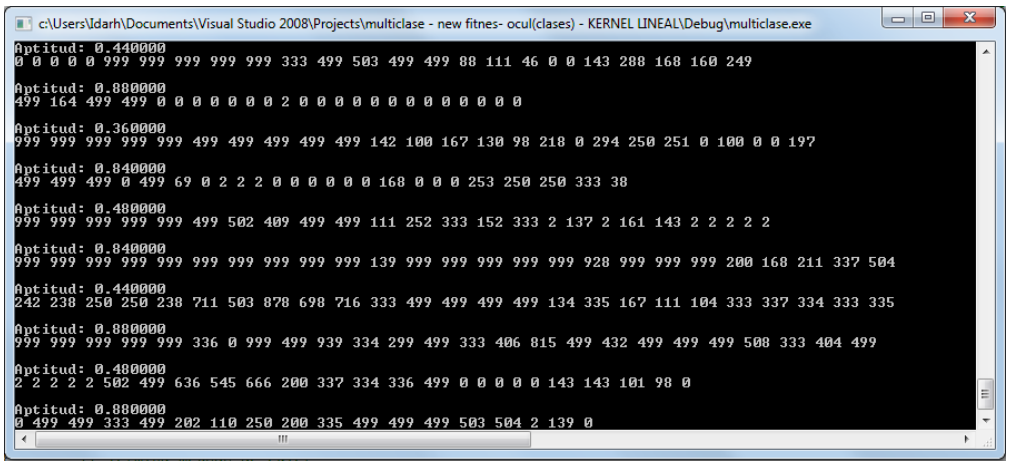


Figura 26. Consola de la aplicación en el entrenamiento de RNP.

5.1.2 Validación de la RNP para el reconocimiento de patrones.

Para realizar la validación por medio del conjunto de prueba, se desarrolló una aplicación en lenguaje de MATLAB, la figura 27 muestra la GUIDE desarrollada para la aplicación.

Figura 27. Interfaz creada para realizar la validación de la RNP.

Como se observa la interfaz cuenta con tres botones en la parte superior, con los cuales se seleccionarán el conjunto de entrenamiento usado, el conjunto de validación y los valores de los parámetros almacenados al finalizar el entrenamiento. Todos estos archivos tienen una extensión .dat.

Después de seleccionar los valores, en cuanto a número de clases y número de rasgos, y la función a utilizar se procede a realizar la clasificación de los dos conjuntos con los valores para los parámetros seleccionados, el resultado es mostrado en la interfaz.

5.1.3 Implementación para el umbralado de imágenes.

Para resolver el problema del umbralado de imágenes mediante el método propuesto, se desarrolló una aplicación utilizando el lenguaje de programación MATLAB, interfaz mediante la cual es posible seleccionar la imagen a tratar, los segmentos representantes y realizar los procesos tanto de entrenamiento de la neurona, como la clasificación de la totalidad de la imagen para obtener una imagen binaria. La figura 28 muestra la interfaz de usuario desarrollada para realizar la tarea mencionada.

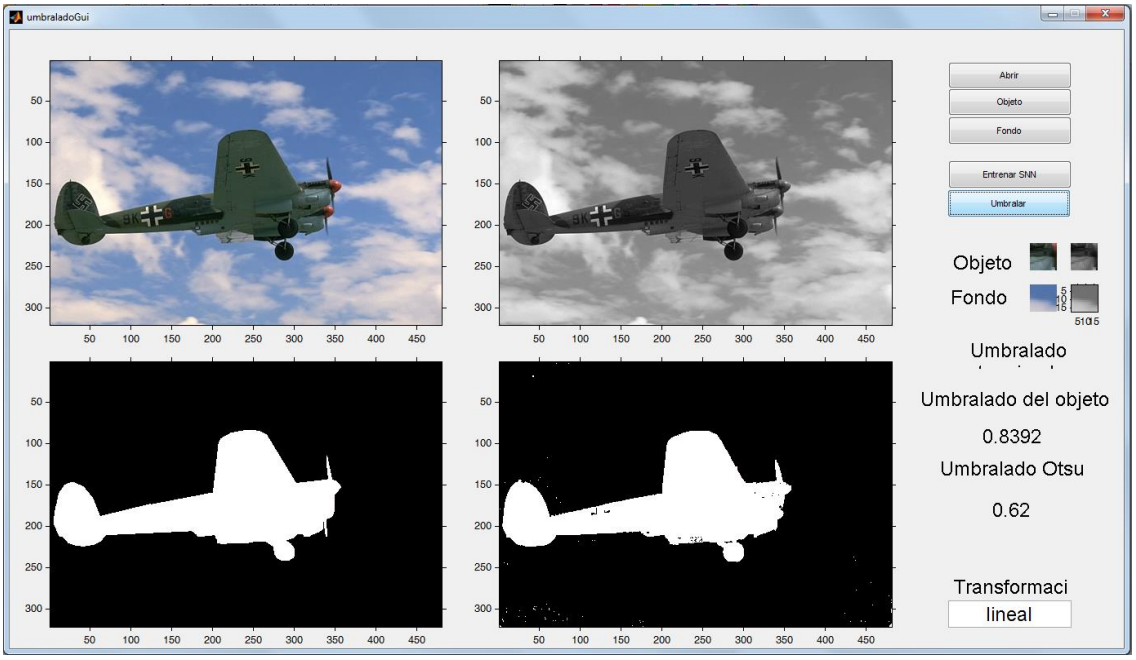


Figura 28. Interfaz para el umbralado de imágenes mediante RNP.

5.2 Resultados para el problema XOR.

El problema XOR (O exclusivo) es el clásico ejemplo de problema de clasificación no lineal en dos dimensiones, el problema cuenta con dos clases, 0 y 1, la tabla 6 muestra los vectores de rasgos del problema XOR junto con su clasificación.

Tabla 6. Rasgos y clasificación del problema XOR

x1	x2	Clase
0	0	0
1	0	1
0	1	1
1	1	0

Se utilizó la transformación polinomial con un valor de $p = 2$ para el cálculo de la corriente I de entrada hacia una neurona con el modelo de Izhikevich, después del ajuste de los pesos mediante ED se logra la clasificación correcta de los vectores de entrada mediante la frecuencia de disparo. La tabla 7

muestra los vectores de entrada así como la frecuencia de disparo de cada uno de ellos.

Tabla 7. Clasificación mediante la frecuencia de disparo del problema XOR

x1	x2	Frecuencia	Clase
0	0	414	0
1	0	999	1
0	1	999	1
1	1	414	0

Como se puede observar en la tabla anterior las frecuencias de disparo muestran una amplia separación entre clases, mientras que los vectores de rasgos pertenecientes a las mismas clases muestran una frecuencia parecida.

Gracias a la trasformación del vector de rasgos de entrada a una corriente por medio de la función polinomial, es posible encontrar un hiperplano capaz de separar las clases de manera correcta. La figura 29 muestra de manera gráfica el problema XOR en el espacio original de los vectores de entrada y en el espacio de mayor dimensión, tomando como una dimensión más el valor de la corriente obtenida.

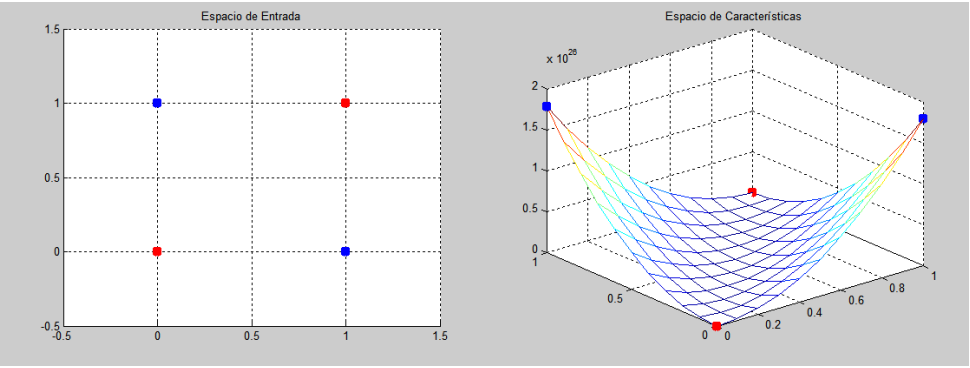


Figura 29. Visualización del problema XOR con la dimensión aumentada gracias a la transformación polinomial.

En el anexo A se presentan casos en los que se utilizó una neurona pulsante en conjunto con transformaciones polinomial y gaussiana RBF para la solución de otros dos problemas de clasificación.

5.3 Resultados en el reconocimiento de voz mediante la arquitectura propuesta.

Se utilizó un conjunto de rasgos extraídos de grabaciones de voz con el fin de observar el comportamiento de la Red Neuronal Pulsante de Izhikevich con la arquitectura propuesta. El conjunto de datos está conformado por 50 vectores de rasgos, en donde cada vector contiene 11 coeficientes MFCC, que describen la grabación de una de las vocales (a, e, i, o, u), es decir, se tienen 10 repeticiones de cada vocal.

Los coeficientes cepstrales en la frecuencia de Mel (MFCC) son obtenidos utilizando dos herramientas del procesamiento digital de señales: la transformada de Fourier y los bancos de filtros. Comúnmente el número de coeficientes a extraer en una señal de voz, está dado por la frecuencia de muestreo a la que está grabada la señal. En este caso las muestras están grabadas a una frecuencia de 8,000 muestras por segundo. El número de coeficientes a calcular está dado por:

$$m = \frac{fs}{1{,}000} + 3$$

Donde m es el número de coeficientes a calcular, fs es la frecuencia de muestreo a la que está grabada la señal.

La arquitectura tomada para la Red Neuronal Pulsante, siguiendo las reglas anteriormente expuestas, quedó definida de la siguiente forma: 11 neuronas en la capa de entrada (11 rasgos), cinco neuronas en la capa intermedia (5 clases), una neurona en la capa de salida para realizar la clasificación por

medio de la frecuencia de disparo. La función utilizada para el cálculo de la corriente de entrada fue la transformación productos. Los resultados obtenidos se presentan en la figura 30.

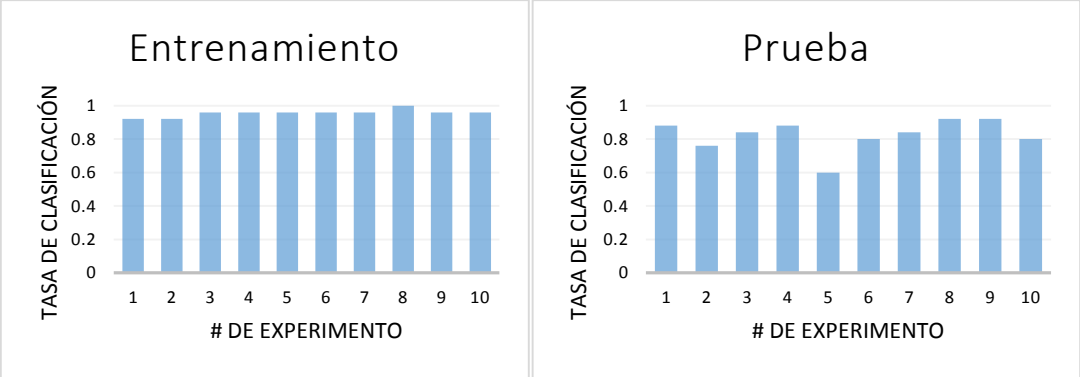


Figura 30. Resultados obtenidos del conjunto de datos extraídos de grabaciones de voz.

Como se observa en la figura anterior, la Red Neuronal Pulsante es capaz de aprender un promedio de 0.956 del total de los patrones en el entrenamiento y de clasificar un 0.824 en la validación, esto utilizando la arquitectura propuesta y el entrenamiento por medio del algoritmo de ED.

5.4 Resultados obtenidos en problemas de reconocimiento de patrones.

Después de observar los resultados obtenidos en el problema de clasificación anterior y con el fin de probar el desempeño del método propuesto en el caso del reconocimiento de patrones por medio de una Red Neuronal Pulsante, cuya unidad principal esté basada en el modelo neuronal de Izhikevich, en conjunto con las funciones de trasformación, se realizaron experimentos donde se utilizaron cuatro bases de datos de la *UCI machine learning repository* [56].

Para validar la precisión del método propuesto, se realizaron 10 experimentos sobre cada base de datos. Se hace la comparación de los resultados obtenidos con la RNP propuesta, contra los obtenidos mediante una red MLP y con una

Máquina de Vector Soporte implementada sobrer el software WEKA, además de otro trabajo con RNP.

Los resultados en los experimentos son mostrados de manera separada en las figuras siguientes, las gráficas muestran los resultados obtenidos durante la fase de entrenamiento y durante la fase de validación. Posteriormente en la tabla 8 se presentarán los promedios obtenidos en los experimentos.

5.4.1 Resultados obtenidos de la base de datos de la planta de Iris.

La base de datos de la planta de Iris consiste en 50 ejemplos de cada una de las tres clases que la conforman (*iris setosa, Iris virginica e Iris versicolor*). Cada patrón está compuesto por cuatro rasgos que corresponden a las medidas de: largo y ancho, de sépalo y pétalo, dadas en centímetros. La base cuenta con tres clases, dos linealmente separables y una no linealmente separable.

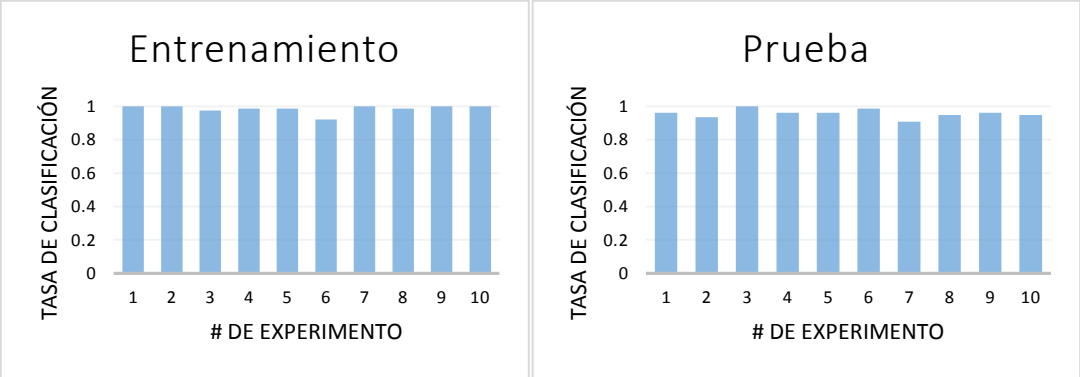


Figura 31. Resultados obtenidos con la base de datos de la planta de Iris y la transformación lineal.

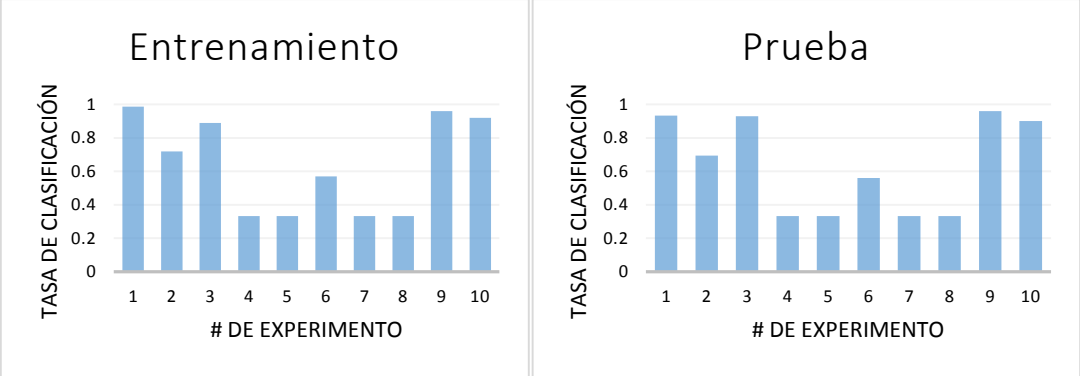


Figura 32. Resultados obtenidos con la base de datos de la planta de Iris y la transformación polinomial.

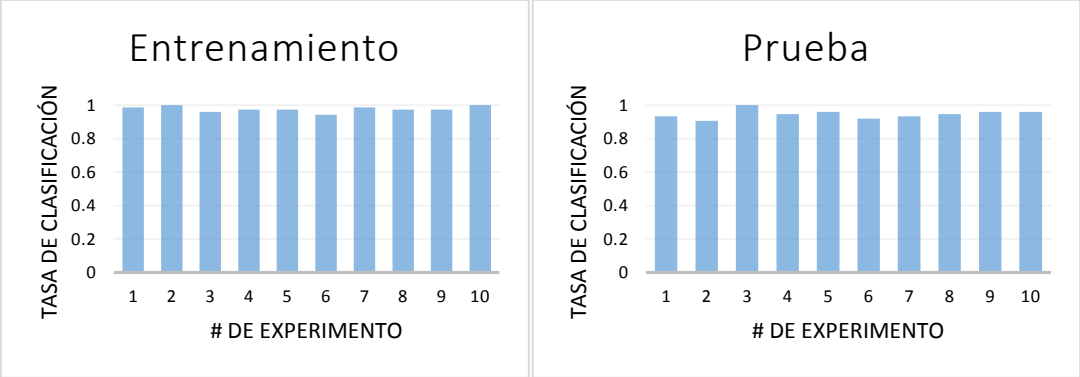


Figura 33. Resultados obtenidos con la base de datos de la planta de Iris y la transformación productos.

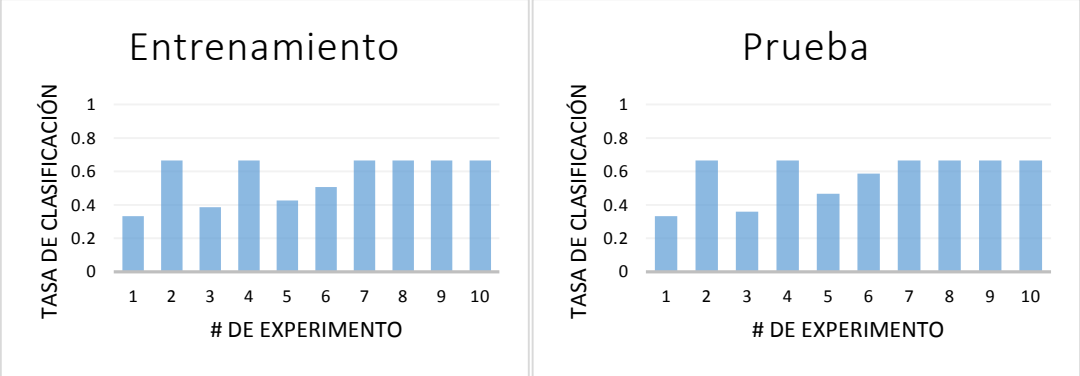


Figura 34. Resultados obtenidos con la base de datos de la planta de Iris y la transformación RBF.

5.4.2 Resultados obtenidos de la base de datos del vino.

La base de datos del vino está compuesta por un total de 178 patrones, cada patrón consta de 14 rasgos que son resultados de análisis químicos realizados a tres tipos de vinos. Las figuras siguientes presentan los resultados obtenidos al aplicarse el método propuesto para su clasificación.

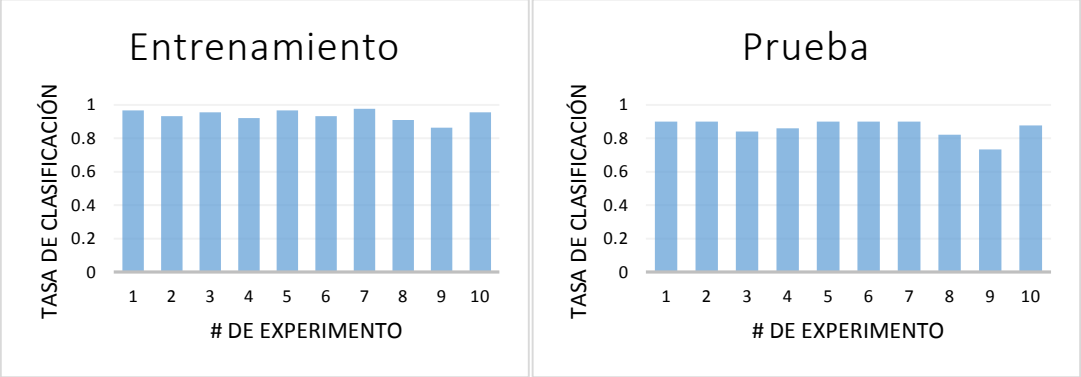


Figura 35. Resultados obtenidos con la base de datos del vino y la transformación lineal.

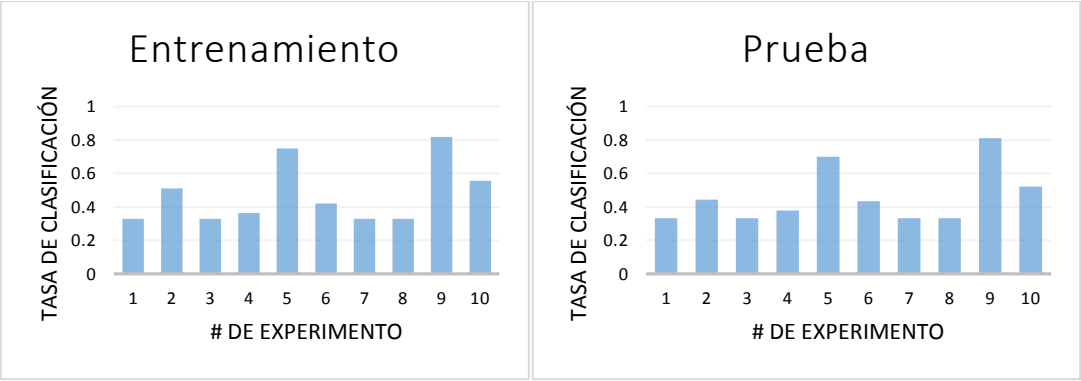


Figura 36. Resultados obtenidos con la base de datos del vino y la transformación polinomial.

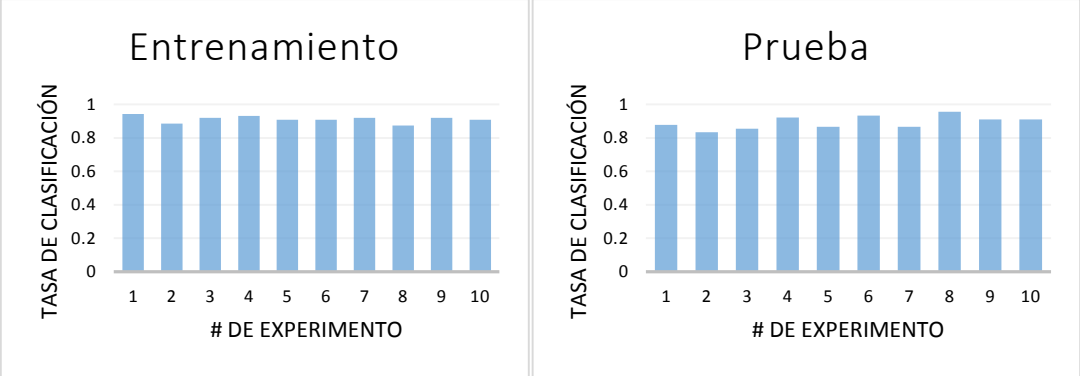


Figura 37. Resultados obtenidos con la base de datos del vino y la transformación productos.

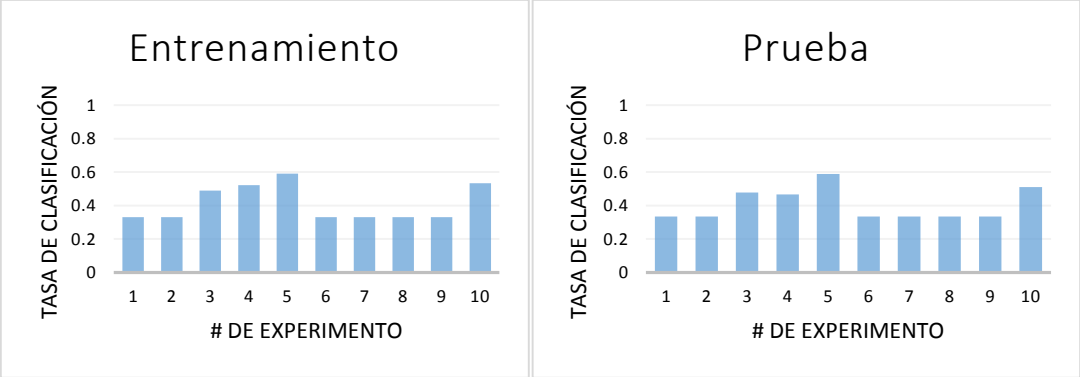


Figura 38. Resultados obtenidos con la base de datos del vino y la transformación RBF.

5.4.3 Resultados obtenidos de la base de datos del vidrio.

La base de datos del vidrio es un conjunto de datos que se generó para ayudar en la investigación criminológica. El conjunto de datos contiene 214 patrones, nueve atributos numéricos y el nombre de clase. Cada instancia pertenece a una de las 7 clases posibles. Las figuras siguientes muestran los resultados de aplicar el método propuesto en la base de datos del vidrio.

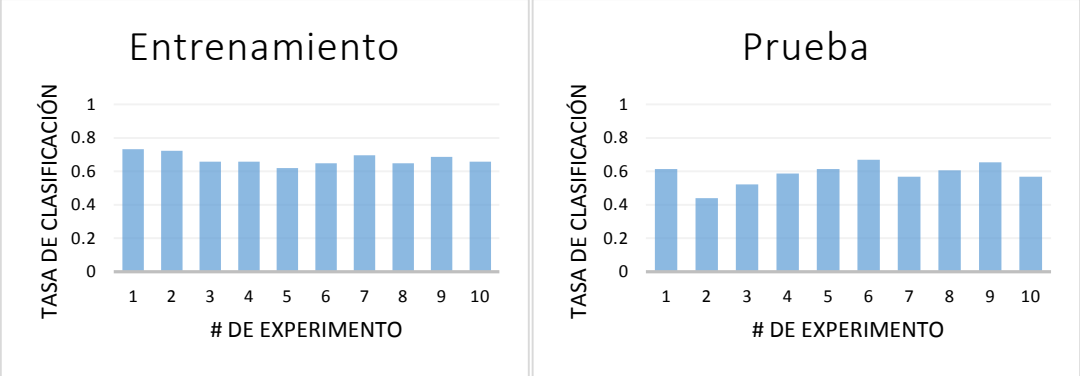


Figura 39. Resultados obtenidos con la base de datos del vidrio y la transformación lineal.

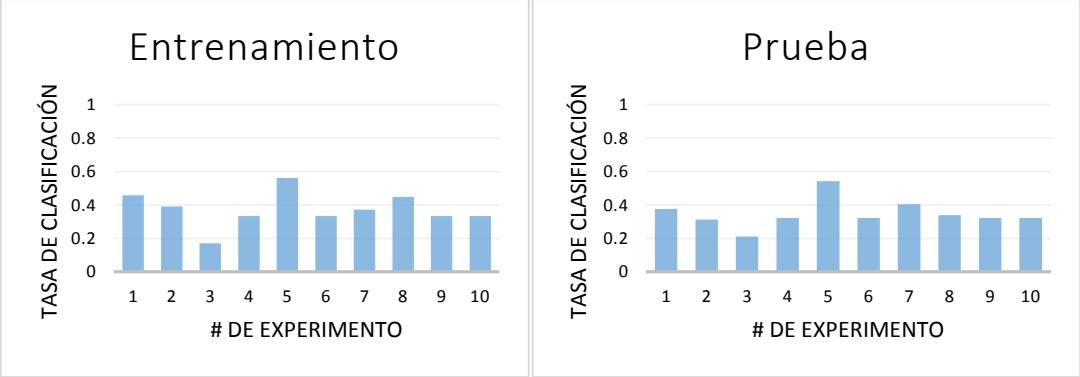


Figura 40. Resultados obtenidos con la base de datos del vidrio y la transformación polinomial.

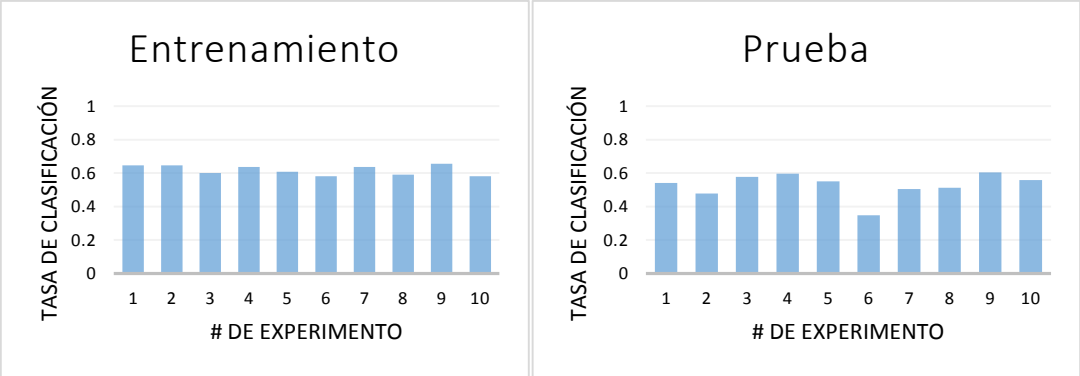


Figura 41. Resultados obtenidos con la base de datos del vidrio y la transformación productos.

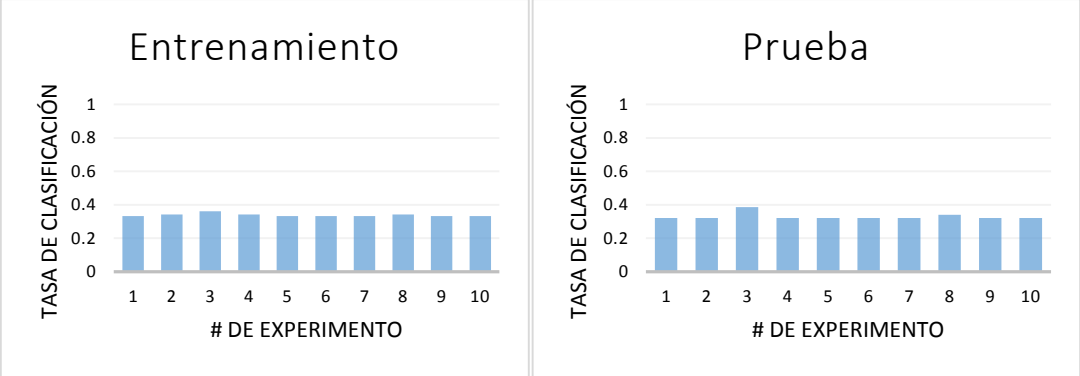


Figura 42. Resultados obtenidos con la base de datos del vidrio y la transformación RBF.

5.4.4 Resultados obtenidos con la base de datos de los desórdenes del hígado.

La base de datos de los desórdenes del hígado cuenta con dos clases, 345 patrones compuestos de nueve rasgos. Los resultados de clasificación por medio del método propuesto son mostrados en las figuras siguientes.

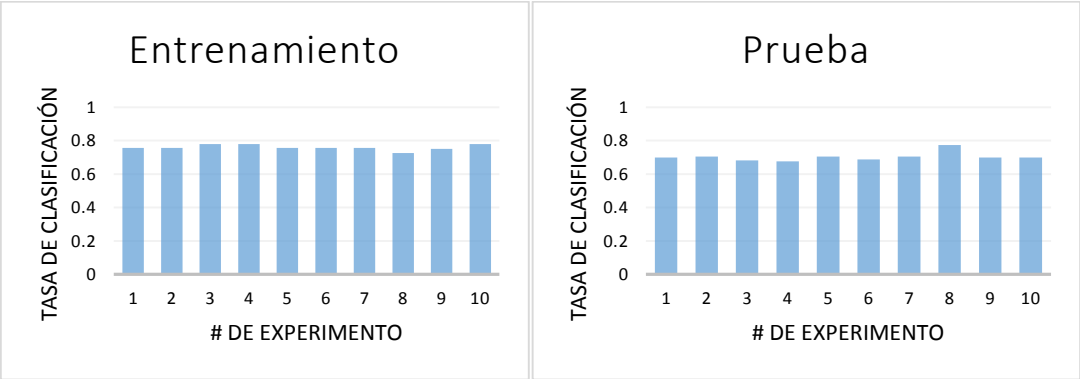


Figura 43. Resultados obtenidos con la base de datos de los desórdenes del hígado y la transformación lineal.

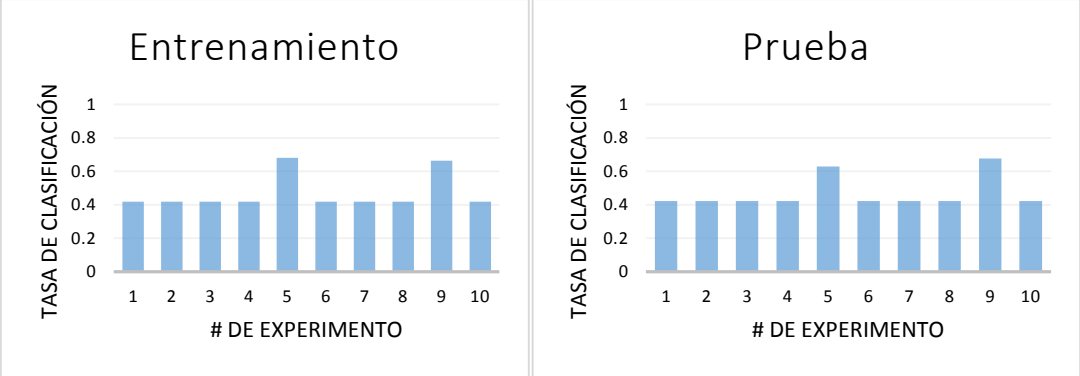


Figura 44. Resultados obtenidos con la base de datos de los desórdenes del hígado y la transformación polinomial.

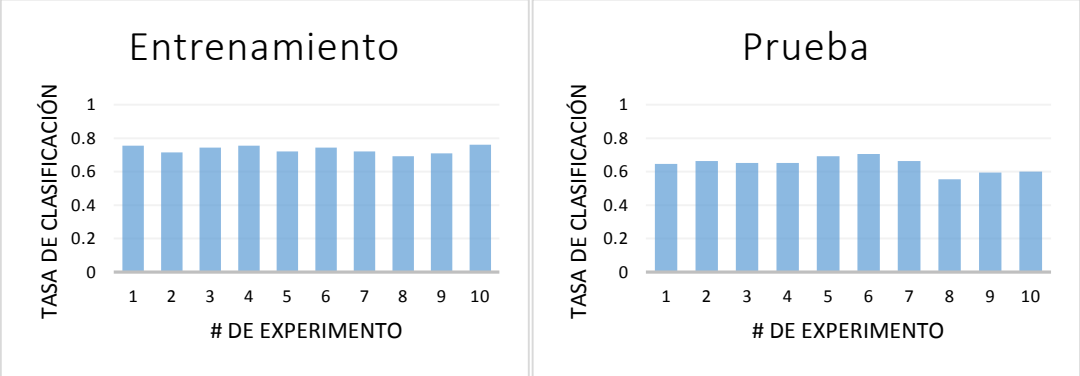


Figura 45. Resultados obtenidos con la base de datos de los desórdenes del hígado y la transformación productos.

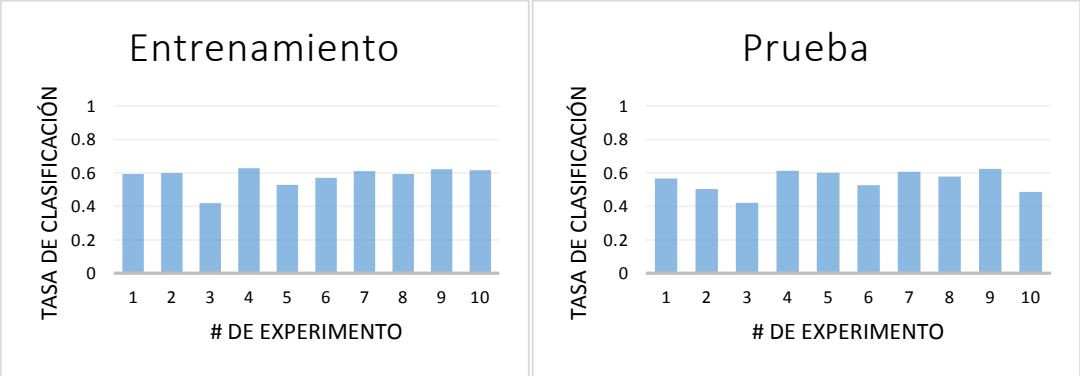


Figura 46. Resultados obtenidos con la base de datos de los desórdenes del hígado y la transformación RBF.

En la tabla 8 se presenta el promedio de clasificación de todos los experimentos en su entrenamiento y en su validación, y en la tabla 9 se muestran los resultados obtenidos mediante MVS implementada mediante WEKA.

Tabla 8. Media de clasificación de los experimentos de reconocimiento de patrones usando funciones de transformación.

Base	**Lineal**		**Polinomial**		**Producto**		**RBF**		**MLP**	**RNP[10]**
	Ent.	Val.	Ent.	Val.	Ent.	Val.	Ent.	Val.	Val.	Val.
Planta de Iris	0.9853	0.9559	0.6366	0.6296	0.9799	0.9466	0.5652	0.5745	**0.96**	0.9308
Vino	0.9374	0.8633	0.4738	0.4621	0.9124	0.8932	0.4113	0.4044	**0.9775**	0.8319
Vidrio	0.6723	0.5847	0.3733	0.3467	0.6190	0.5275	0.3390	0.3293	0.6074	**0.7411**
Hígado	0.7592	**0.7034**	0.4691	0.4681	0.7319	0.6433	0.5778	0.5525	0.6279	0.6870

Tabla 9. Resultados de clasificación mediante Máquina de Vector Soporte (MVS).

	Máquina de Vector Soporte		
Base	**Lineal**	**Polinomial**	**RBF**
Planta de Iris	0.9866	0.96	1
Vino	0.9775	0.9775	0.4157
Vidrio	0.5420	0.6168	0.55
Hígado	0.6279	0.5755	0.5755

Los resultados obtenidos en los experimentos muestran que el método propuesto tiene un buen desempeño realizando la clasificación de bases de datos multi-clase. Y que la arquitectura propuesta obtiene resultados muy similares a los obtenidos con MLP y con MVS en algunos casos.

Las funciones de trasformación del vector de entrada a corriente que mejores resultados mostraron, fueron la lineal y productos, mientras que la que peores resultados mostró fue la función gaussiana RBF.

5.5 Resultados obtenidos en el umbralado de imágenes.

Se seleccionaron diez imágenes de la base de datos tomada de [24] para aplicar el método propuesto para realizar el umbralado de imágenes. Estas imágenes son las que se presentan en la figura 47.

Figura 47. Imágenes seleccionadas para aplicar el método de umbralado.

El conjunto de imágenes cuenta con una imagen binaria realizada a mano en la que se puede observar un umbralado cercano al ideal. Estas imágenes binarias se presentan en la figura 48.

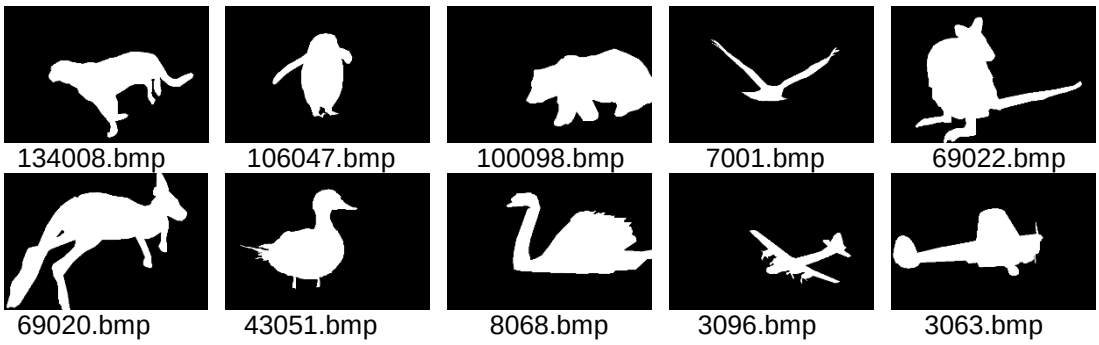


Figura 48. Umbralado manual de las imágenes seleccionadas.

Siguiendo el método descrito anteriormente para realizar el umbralado como un problema de clasificación, se seleccionaron dos segmentos correspondientes al objeto de interés y al fondo de la imagen. Las dimensiones de estos segmentos fueron de 17x17 pixeles y su selección se hizo de manera manual utilizando la interfaz gráfica desarrollada en MATLAB.

Después de obtener los rasgos descriptores de color y textura mencionados en secciones anteriores, se hizo el entrenamiento de la neurona utilizando las cuatro funciones de transformación, los resultados del conjunto de prueba obtenidos son mostrados en la figura 49, 50, 51 y 52.

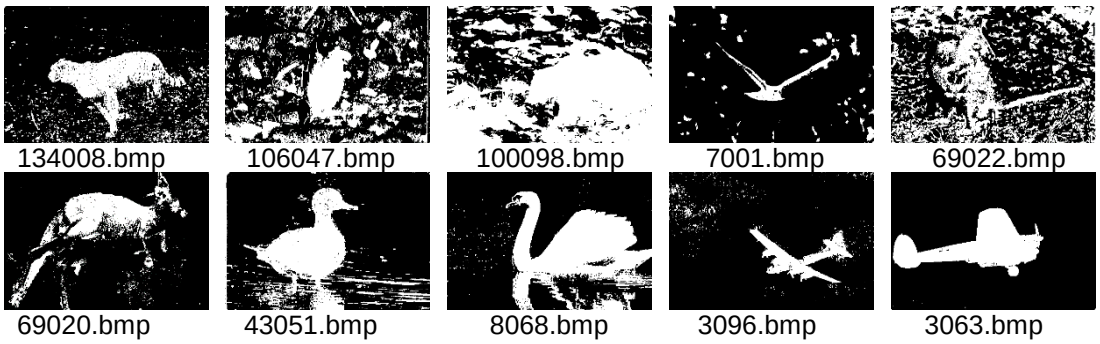


Figura 49. Umbralado de las imágenes mediante una transformación lineal.

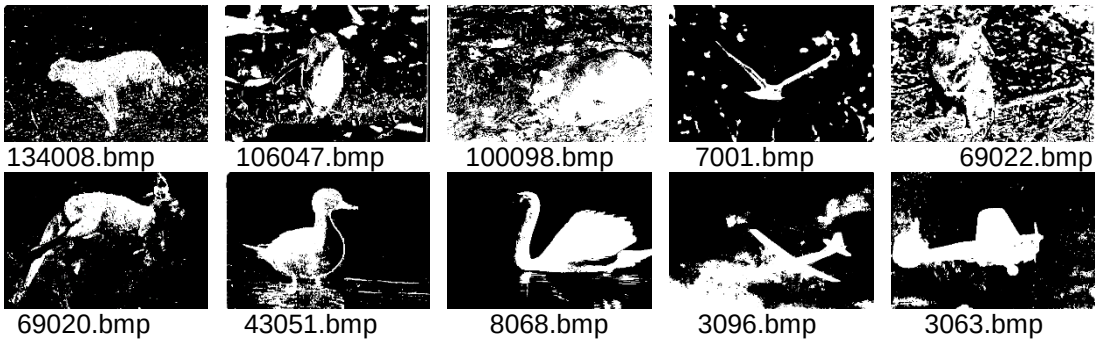


Figura 50. Umbralado de las imágenes mediante una transformación polinomial.

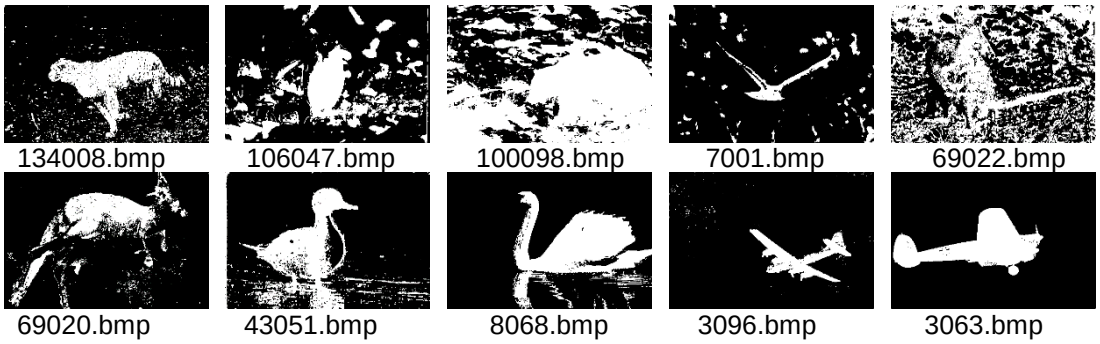


Figura 51. Umbralado de las imágenes mediante una transformación productos.

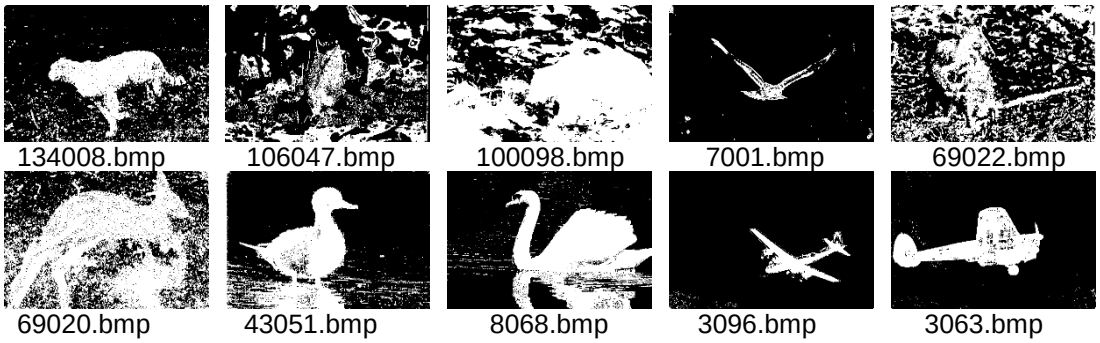


Figura 52. Umbralado de las imágenes mediante una transformación RBF.

Para tener un punto de comparación se realizó el umbralado por el método clásico de Otsu [53] obteniendo los resultados mostrados en la figura 53.

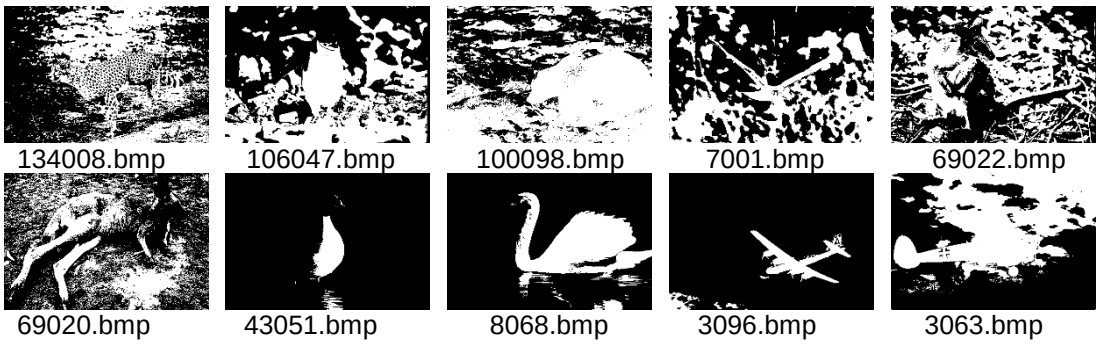


Figura 53. Umbralado de las imágenes mediante el método de Otsu.

La figura 54 muestra las diferencias entre las imágenes con umbral manual tomadas de la base de datos y los resultados presentados anteriormente, esto para apreciar mejor el desempeño de cada una de las funciones de transformación, estas imágenes fueron obtenidas al aplicar la función XOR a las imágenes binarias obtenidas y a la imagen umbralada de manera manual (los pixeles en blanco son los errores).

Posteriormente en la tabla 10 se muestran las tasas de error obtenidas de cada una de las imágenes para tener una manera numérica de medir los resultados y hacer una comparación con una de las técnicas más utilizadas para encontrar el umbral de manera automática, el método de Otsu [27].

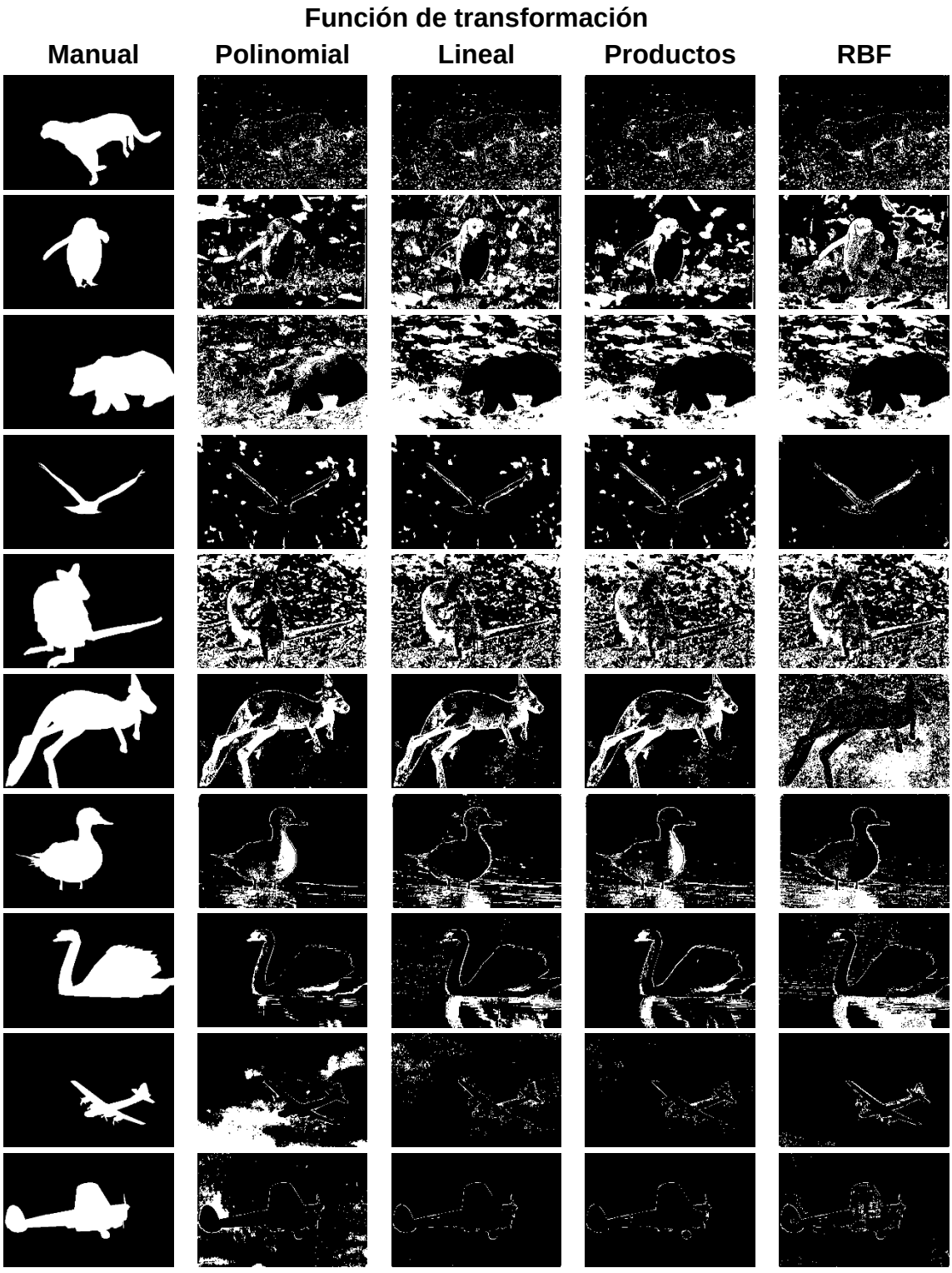


Figura 54. Error en el umbralado mediante el método propuesto.

Tabla 10. Tasa de error en el umbralado mediante el método propuesto.

Imagen	Método de Otsu	Función Lineal	Función Polinomial	Función Productos	Función RBF
134008.bmp	0.3985	0.0830	0.0757	**0.0720**	0.0918
106047.bmp	0.4308	0.2975	0.2041	**0.2040**	0.2792
100098.bmp	0.4232	0.4254	**0.3393**	0.4138	0.4270
7001.bmp	0.3676	0.0442	0.0572	0.0523	**0.0220**
69022.bmp	0.4981	**0.3512**	0.3768	0.4030	0.3521
69020.bmp	0.4128	**0.1532**	0.1658	0.1561	0.3547
43051.bmp	0.1148	0.1042	**0.0974**	0.1166	0.1370
8068.bmp	0.0513	0.0945	**0.0455**	0.0636	0.1331
3096.bmp	0.0235	0.0257	0.1938	**0.0109**	0.0168
3063.bmp	0.3003	0.0072	0.0596	**0.0071**	0.0171

Como se puede observar en el umbralado realizado mediante una neurona pulsante con el modelo de Izhikevich es comparable, y mejor en algunos casos, que uno de los métodos más utilizados en la literatura, como lo es el método de Otsu.

En los experimentos realizados se observó que la función productos es la que mejor se comportaba al poder realizar la clasificación de una manera más acertada, aunque como se demostró las cuatro funciones utilizadas pueden resolver el problema de una manera comparable.

5.6 Discusión de resultados.

Después de analizar los resultados obtenidos se pueden observar dos ventajas del método propuesto, comparado con los encontrados en la literatura, por una parte la utilización de una función no lineal para el cálculo de la corriente de entrada hacia la neurona facilita la solución de problemas como el XOR en el cual se necesitó de sólo una neurona, en comparación con las nueve necesarias en [32] y [42], o las ocho y diez utilizadas en [23]; esto por significa una disminución del número de pesos o parámetros a ajustar.

Por otra parte, la utilización de una neurona a salida en una RNP de tipo *feedforward*, garantiza la eliminación del posible problema de “empate” entre neuronas que puede ocurrir en [12], [13] y [19], y que según se reportó, fue uno de los problemas que se presentaron al realizar la clasificación por medio de la activación de una neurona específica, en una capa de salida de múltiples neuronas, además de brindar la oportunidad de realizar la clasificación por medio de la frecuencia de disparo, que es la principal característica de las RNP.

La aplicación de funciones de transformación, que fueron utilizadas es este trabajo, desempeñaron bien la función de ser el mecanismo para el cálculo de la corriente de entrada hacia el modelo neuronal, en algunos casos mejor que la función que usa el producto punto, con lo cual se puede observar que distintas funciones pueden servir para este fin. Además de que, analizando los conjuntos de datos, se puede deducir que la función para el cálculo de la corriente puede ser diferente dependiendo de la forma en la dispersión de los patrones.

Si bien, no se mejoraron los resultados reportados en [10], si se logró tener resultados cercanos, lo cual indica que nuestro método puede ser mejorado para lograr un desempeño más alto.

Capítulo 6

Conclusiones, trabajos a futuro y recomendaciones

En el presente capítulo se detallan las conclusiones a las que se llegaron después de desarrollar el trabajo y analizar los resultados obtenidos, además se dan propuestas para trabajos futuros y las recomendaciones sobre el tema.

6.1 Conclusiones.

Es posible conjuntar una Red Neuronal Pulsante con un algoritmo evolutivo, como Evolución Diferencial, para realizar el ajuste de sus parámetros, pudiendo así, realizar tareas de reconocimiento de patrones y análisis de imágenes.

Una RNP con arquitectura *feedforward* puede dar resultados de clasificación similares en algunos casos a los métodos más utilizados para este problema, como lo es el perceptrón multicapa.

El algoritmo de Evolución Diferencial es una buena alternativa para realizar el entrenamiento no supervisado de RNP, ya que permite encontrar los valores en los parámetros para el cálculo de la corriente de entrada hacia las neuronas, que resuelven el problema de clasificación.

El umbralado de imágenes en donde sólo existen dos clases (objeto y fondo) es posible realizarlo a través de una RNP, utilizando rasgos descriptores de textura y color, teniendo en algunos casos, mejores resultados el método de Otsu.

Problemas de clasificación complejos, dada la dimensión de su vector de descriptores, es simplificado gracias a la transformación que se hace a una sola dimensión (frecuencia de disparos) por medio de la neurona pulsante.

El uso de funciones de trasformación para el cálculo de la corriente de entrada a una RNP puede facilitar el problema de clasificación de patrones no linealmente separables como el caso del XOR, además de disminuir el número de valores a ajustar y neuronas necesarias para realizar la tarea.

El uso de una sola neurona en la capa de salida de una Red Neuronal Pulsante con tres capas, da la capacidad de hacer la clasificación de patrones por medio de la frecuencia de disparo.

6.2 Trabajos a futuro.

Programar el método de entrenamiento en un lenguaje de programación paralelo como por ejemplo CUDA, para agilizar así el proceso.

Implementar el método de entrenamiento y clasificación en un lenguaje de descripción de Hardware y en una arquitectura de tipo FPGA.

Programar la Red Neuronal Pulsante en un lenguaje de programación paralelo para realizar experimentos de clasificación con un mayor número de clases.

Probar el desempeño de Redes Neuronales Pulsantes para el análisis de información dinámica, aprovechando la capacidad de estas para responder a cambios en la corriente de entrada y reflejarlas en la frecuencia de salida.

6.3 Recomendaciones.

Probar con las diferentes tipos de respuesta que puede generar la el modelo de Izhikevich para comprender un mejor como es que responde a ciertos fenómenos como el cambio en la corriente de entrada.

Utilizar programación paralela para realizar la simulación y así agilizarla ya que esto en muchas ocasiones representa un problema al momento de hacer los experimentos.

En caso de realizar trabajos sobre reconocimiento de patrones multi-clase usar lo mostrado en este trabajo como base, ya que se ha demostrado que la arquitectura utilizada funciona en algunos casos.

Referencias

[1] F.A. Segovia, S.C. Corrales, "Acoplamiento Excitatorio e Inhibitorio de Neuronas Pulsantes", Redes de Ingeniería, Vol. 2, pp. 99-104, Agosto-Diciembre 2011.

[2] J.J. Báez Rojas, M.L. Guerrero, J. Conde, A. Padilla, and G. Urcid, “Segmentación de imágenes en color”, Revista Mexicana de Física, Vol. 50(6), pp. 579-587, Diciembre 2004.

[3] E.M. Izhikevich, “Which Model to Use for Cortical Spiking Neurons?”, IEEE Transactions On Neural Networks, Vol. 15(5), pp. 1063-1069, Septiembre 2004.

[4] J. Misra, I. Saha, “Artificial Neural Networks in Hardware: A Survey of Two Decades of Progress”, Neurocompiting, Vol. 74, pp. 239-255, 2010.

[5] A. Arista, R.A. Vázquez, “Implementación Configurable y Multipropósito de Redes Neuronales de Tercera Generación en GPUs”, Memorias del Primer Concurso de Investigación, Desarrollo e Innovación Tecnológica, pp. 35-38, 2012.

[6] E.M. Izhikevich, “Simple Model of Spiking Neurons”, IEEE Transactions On Neural Networks, Vol. 14(6), pp. 1569-1572, Noviembre 2003.

[7] W. Maass, “Networks of Spiking Neurons: The Third Generation of Neural Network Models”, Neural Networks, Vol. 10(9), pp. 1659-1671, 1997.

[8] J.A. Ramírez, M.I. Chacón, “Redes neuronales artificiales para el procesamiento de imágenes, una revisión de la última década”, Revista de Ingeniería Eléctrica, Electrónica y Computación, Vol. 9(1), pp. 7-16, Julio 2011.

[9] R.A. Vázquez, “Pattern Recognition Using Spiking Neurons and Firing Rates”, *Proceedings of the 12th Ibero-American conference on Advances in artificial intelligence* (IBERAMIA'10), Angel Kuri-Morales and Guillermo R. Simari (Eds.). Springer-Verlag, Berlin, Heidelberg, pp. 423-432, 2010.

[10] R.A. Vázquez, “Izhikevich Neuron Model and its Application in Pattern Recognition”, *Australian Journal of Intelligent Information Processing Systems*, Vol. 11(1), pp. 35-40, 2010.

[11] J.A. Cortes, A.M. Muriel y J.A. Mendoza, “Comparación Cualitativa y Cuantitativa de las Técnicas Básicas de Umbralización Global Basadas en Histogramas Para el Procesamiento Digital de Imágenes”, *Scientia et Technica Año XVI,* No. 49, Universidad Tecnológica de Pereira, ISSN 0122-1701, pp. 266-272, 2011.

[12] K.L. Rice, M.A. Bhuiyan, T.M. Taha, C.N. Vutsinas y M.C. Smith, “FPGA Implementation of Izhikevich Spiking Neural Networks for Character Recognition”, *International Conference on Reconfigurable Computing and FPGAs*, pp. 451-456, 2009.

[13] M.A. Bhuiyan, R. Jalasutram y T.M. Taha, “Character Recognition with Two Spiking Neural Network Models on Multicore Architectures”, *Computational Intelligence for Multimedia Signal and Vision Processing*, pp. 29-34, 2009.

[14] G. Kuntimad, H.S. Ranganath, “Perfect Image Segmentation Using Pulse Coupled Neural Networks”, *IEEE Transactions on Neural Networks*, Vol 10(3), pp. 591-598, Mayo 1999.

[15] E. Yee, J. Teo, “Evolutionary Spiking Neural Networks as Racing Car Controllers”, *Hybrid Intelligent Systems (HIS), 2011 11th International Conference on,* pp. 411-416, Diciembre 2011.

[16] A. Bouganis, M. Shanahan, “Training a Spiking Neural Network to Control a 4-DoF Robotic Arm Based on Spike Timing-Dependent Plasticity”, *Neural Networks (IJCNN), The 2010 International Joint Conference on*, pp. 4104-4111, Barcelona España, Julio 2010.

[17] C. Elías, ”Control de un Par de Robots Humanoides Mediante Redes Neuronales Pulsantes para Realizar Formaciones”, *Tesis de Maestría, Centro de Investigación en Computación CIC-IPN*, Agosto 2013.

[18] M.A. Brauer, ”Control de un robot humanoide mediante Redes Neuronales Pulsantes para la manipulación de objetos”, *Tesis de Maestría, Centro de Investigación en Computación CIC-IPN*, Agosto 2013.

[19] A. Gupta, L.N. Long, “Character Recognition Using Spiking Neural Networks”, *Neural Networks, 2007. IJCNN 2007. International Joint Conference on*, pp.53-58, Orlando FL., Agosto 2007.

[20] E. Menzuram J. Velázquez, C. Coello, “A comparative Study of Differential Evolution Variants for Global Optimization”, *In Proceedings of the 8th annual conference on Genetic and evolutionary computation (GECCO '06). ACM*, pp. 485-492, New York Julio 2006.

[21] M. Presutti, “La Matriz de Co-Ocurrencia en la Clasificación Multiespectral: Tutorial para la Enseñanza de Medidas Texturales en Cursos de Grado Universitario”, *4° Jornada de Educação em Sensoriamento Remoto no Âmbito do Mercosul*, Brazil, Agosto 2004.

[22] J. Fox, “Massively Parallel Neural Computation”, *Technical Report, University of Cambridge*, Marzo 2013.

[23] S.M. Bohte, J.N. Kok y H. La Pautre, “SpikeProp: Backpropagation from Networks of Spiking Neurons”, *Neurocomputing*, Vol. 48, pp.17–37, 2002.

[24] H. Li, J. Cai, T. Nhat y J. Zheng, ”A Benchmark for Semantic Image Segmentation”, *Multimedia and Expo (ICME), 2013 IEEE International Conference*, pp. 1-6, Julio 2013.

[25] Q. Wu, T.M. McGinnity, L. Maguire, G.D. Valderrama y P. Dempster, “Colour Image Segmentation Based on a Spiking Neural Networks Model Inspired by the Visual System”, *Advanced Intelligent Computing Theories and Applications*, Vol. 6215, pp 49-57, 2010.

[26] A. Kasinski, F. Ponulak, “Comparison of Supervised Learning Methods for Spike Time Coding in Spiking Neural Networks”, International Journal of Applied Mathematics and Computer Science, Vol. 16(1), pp.101-113, 2006.

[27] R. Rodriguez, J.H. Sossa, *Procesamiento y Análisis Digital de Imágenes*, Primera Edición, Grupo Editorial Alfaomega, 2012.

[28] R.A. Vázquez, A. Cachón, “Integrate and Fire Neurons and their Application in Pattern Recognition”, *2010 7th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE 2010),* pp. 424-428, 2010.

[29] W. Swiercz, K.J. Cios, K. Staley, L. Kurgan, F. Accurso, y S. Sagel, “A Nwe Synaptic Platicity Rule for Networks of Spiking Neurons”, *IEEE Transactions on Neural Networks,* Vol. 17(1), pp. 94-105, Junio 2006.

[30] S.M. Bohte, J.N. Kok, “Applications of Spiking Neural Networks”, *Information Processing Letters,* Vol. 95, pp. 519--520, 2005,

[31] J. Xin, M.J. Embrechts, “Supervised Learning with Spiking Neural Networks”, *Proceedings. IJCNN '01, International Joint Conference on Neural Networks,* Vol. 3, pp. 1772–1777, 2001.

[32] S. Kamoi, R. Iwai, H. Kinjo y T. Yamamoto, “Pulse Pattern Training of Spiking Neural Networks Using Improved Genetic Algorithm”, Proceedings 2003, IEEE International Symposium on Computational Intelligence in Robotics and Automation, pp. 977-981, 2003.

[33] B. Schrauwen, J.V. Campenhout, “Extending SpikeProp”, Proceedings. 2004 IEEE International Joint Conference on Neural Networks, Vol. 1, pp. 471-475, 2004.

[34] Q. Wu, T.M. McGinnity, L. Maguire y J. Cai, “Detection of Stright Lines using a Spiking Neural Netwron Model”, *2009 Fifth International Conference on Natural Computation*, pp. 385-389, 2009.

[35] E. Xie, T.M. McGinnity, Q. Wu, J. Cai y R. Cai, “GPU Implementation of Spiking Neural Networks for Color Image Segmentation”, *2011 4th International Congress on Image and Signal Processing*, pp. 1246-1250, 2011.

[36] A. Herrera, S. Quintana, J.L. Pérez y G. Hernández, “Electronic Implementation and Analysis of a Small Neural Network*”, Instrumentation and Development,* Vol. 3(7), pp. 25-33, 1997.

[37] T.J. Strain, L.J. MacDaid, L.P.Maguire y T.M. McGuinnity, “A Supervised STDP Based Training Algorithm with Dynamic Threshold Neurons”, *International Joint Conference on Neural Networks*, pp. 3409-3414, 2006.

[38] H. Wang, H. Wang, “Improvement of Izhikevich’s Neuronal and Neural Network Model”, *International Conference on Information Engineering and Computer Science,* pp. 1-4, 2009.

[39] N.G Pavlidis, D.K. Tasoulis, V.P. Plagianakos, G. Nikiforidis y M.N. Vrahatis, “Spiking Neural Network Training Using Evolutionary Algorithms”, *Proceedings of International Joint Conference on Neural Networks*, pp. 2190-2194, Agosto 2004.

[40] S. Ratnasingam, T.M. MacGuinnity, “A Spiking Neural Network for Tactile Form Based Object Recognition”, *Proceedings of International Joint Conference on Neural Networks*, pp. 880-885, 2011.

[41] E. M. Izhikevich, “Dynamical Systems in Neuroscience: The Geometry of Excitability and Bursting”, *The MIT Press, Cambridge, Massachusetts, London, England*, 2007.

[42] S.M. Bohte, J.N. Kok y H. La Poutre, “Error-backpropagation in Temporally Encoded Networks of Spiking Neurons”, *Neurocomputing*, Vol. 48, pp.17-37, 2002.

[43] Q. Wu, M. McGuinnity, L. Maguire, A. Belatreche y B. Glackin, “Edge

Detection Based on Spiking Neural Network Model", *Advanced Intelligent Computing Theories and Applications. With Aspects of Artificial Intelligence*, pp. 26-34, 2007.

[44] Q. Wu, M. McGuinnity, L. Maguire, B. Glackin y A. Belatreche, "Information Processing Functionality of Spiking Neurons for Image Feature Extracction", *Seventh International Workshop on Information Processing in Cells and Tissue,* pp. 1-12 August 2007.

[45] I. Peralta, J.T. Molas, C.E. Martínez y H.L. Rufiner, "Implementación de una Re Neuronal pulsante Parametrizable en FPGA", *Anales de la XIV reunión de procesamiento de la información Control*, 2011.

[46] L. Szymanski, B. McCane, "Visualising Kernel Spaces", *Image and Vision Computing New Zealand (IVCNZ),* pp. 449-452, 2011.

[47] L.F. Bertona, "Entrenamiento de Redes Neuronales Basdo en Algoritmos Evolutivos", *Tesis de Grado en Ingeniería Informatica, Laboratorio de Sistemas Inteligentes, Facultad de Ingeniería, Universidad de Buenos Aires*, Noviembre 2005.

[48] F.J. Gómez, M.A. Fernández, M.T. López y M. Alonso, "Aprendizaje con Redes Neuronales Artificiales", *Revista Ensayos*, No. 9, pp. 169-180, Diciembre 1994.

[49] R. Storn, K. Price, "Differential Evolution-A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces", *Journal of Global Optimization*, Vol. 11, pp. 341-359, 1997.

[50] G. Jabbour, R. Márquez, Renny y L. Ruiz, "Reconocimiento de Firmas Off-Line Mediante Máquinas de Vectores de Soporte*", Revista Ciencia e Ingeniería*, Vol. 31(1), pp. 43-52, 2010.

[51] C.J.C. Burges, "A Tutorial on Support Vector Machines for Pattern recognition", *Data Mining and Knowledge Discovery*, Vol. 2, pp. 121-167, 1998.

[52] J.R. Parker, "Gray Level Thresholding in Badly Illuminated Images", *IEEE Transactions on Pattern Analysis and Machine Intelligence,* Vol. 13(8), pp. 813-819, Agosto 1991.

[53] N. Otsu, "A Thresholding Selection method from Gray-Level Hisgrams", *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-9(1), pp. 62-66, Junio 1979.

[54] E. Izhikevich, G. M. Edelman, "Large-Scale model of Mammalian Thalamocortical Systems", *PNAS*, Vol. 105(9), pp. 3596-3598, Marzo 2008.

[55] B. Meftah, O. Lezoray y A. Benyettou, “Segmentation and Edge Detection Based on Spiking Neural Network Model”, *Neural Process. Lett.* Vol. 32(2), pp. 131-146, Octubre 2010.

[56] P.M. Murphy, D.W. Aha, “UCI repository of machine learning database”, Dep. Inf. Comput. Sci., Univ. Califormia. Irvine, CA, 1994.

[57] W. Maass y C.M. Bishop, “Pulsed Neural Networks”, *Massachusetts Institute of Technology*, 2da. Edición, 1999.

[58] J. Kittler y J. Illingworth, “Minimun Error Thresholding”, *Pattern Recognition*, Vol. 19(1), pp.41-47, 1986.

[59] P. Sahoo, D. Slaaf y T. Albert, “Threshold Selection Using a Minimal Histogram Entropy Difference”, Optical engineering, Vol. 36(7), pp. 2, 1998.

Anexo A

Problemas de clasificación mediante funciones de transformación

Para probar el desempeño de las funciones de transformación en los problemas de reconocimiento de patrones, en donde se tienen diferentes dispersiones, se tomaron dos conjuntos sencillos que presentan diferentes dispersiones y que cuentan con dos clases. Se les asignaron los nombres de “Problema de clasificación mediante función polinomial” y “Problema de clasificación mediante función RBF”. Los resultados obtenidos en la respuesta de la neurona en forma de frecuencia de pulsos se presentan a continuación.

A.1 Problema de clasificación mediante función polinomial.

Es un problema en donde los patrones muestran una dispersión de separabilidad no lineal, figura 55 (izquierda), para su solución se utilizó una función de transformación polinomial, cuyo efecto sobre el espacio de los rasgos originales, figura 55 (derecha) permitió la correcta clasificación de los patrones a través de su frecuencia de disparo.

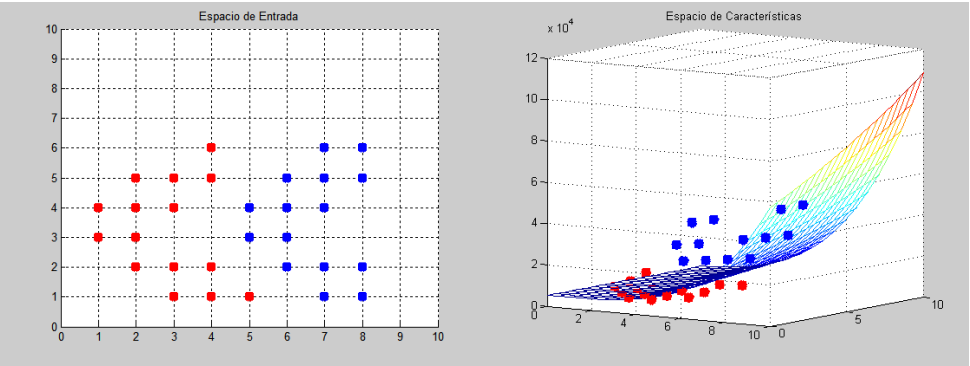


Figura 55. Dispersión en el espacio del problema de clasificación 1.

Después de entrenar una neurona pulsante de Izhikevich se realizó la clasificación de los patrones de acuerdo a la frecuencia de disparo a la salida de ésta. La tabla 11 muestra las frecuencias obtenidas por los patrones.

Tabla 11. Frecuencias de disparo y promedios por clase del problema de clasificación1.

Clase 1			**Clase 2**		
X1	**X2**	**Pulsos**	**X1**	**X2**	**Pulsos**
5	1	510	7	4	999
4	1	499	8	1	999
3	1	499	7	1	999
4	2	499	8	2	999
3	2	499	7	2	999
2	2	499	6	2	999
2	3	499	6	3	999
1	3	499	5	3	914
3	4	499	6	4	999
2	4	499	5	4	999
1	4	499	8	5	999
4	5	505	7	5	999
3	5	499	6	5	999
2	5	499	8	6	999
4	6	510	7	6	999
Promedio		500.8667	**Promedio**		993.3333

Después de obtener los promedios por clase se puede hacer una clasificación de todos los patrones de acuerdo a la menor distancia euclidiana.

A.2 Problema de clasificación mediante función RBF.

El problema de clasificación 2 es un conjunto de patrones de separabilidad no lineal en el espacio original de los rasgos, la figura 56 presenta la dispersión en el espacio original de los datos (izquierda), y la transformación de los rasgos de entrada hacia una corriente por medio de una función de tipo RBF (derecha).

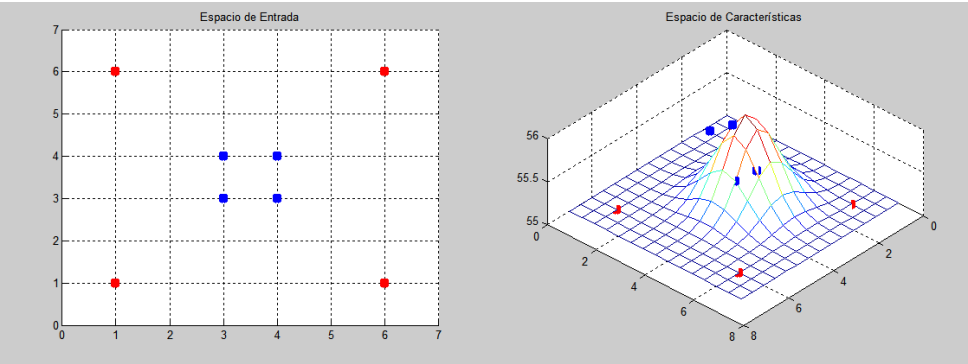


Figura 56. Dispersión en el espacio del problema de clasificación 2.

Después de entrenar una neurona pulsante de Izhikevich se realizó la clasificación de los patrones de acuerdo a la frecuencia de disparo a la salida de ésta. La tabla 12 muestra las frecuencias obtenidas por cada uno de los patrones.

Tabla 12. Frecuencias de disparo del problema de clasificación 2.

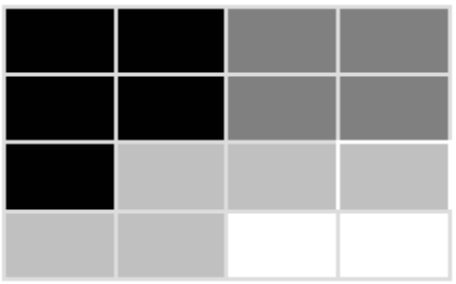
X1	X2	Pulsos
1	1	2
6	1	2
1	6	2
6	6	2
3	3	3
4	3	3
3	4	3
4	4	3

En este caso no fue necesario calcular el promedio por clase, ni la distancia euclidiana para determinar la clase correspondiente a cada patrón, ya que como se observa, los patrones pertenecientes a la misma clase presentan un número idéntico de pulsos generados.

Anexo B

Metodología para calcular medidas de textura por medio de la matriz de Co-ocurrencia

En el presente anexo se muestra la metodología para calcular los tres rasgos de textura utilizados para hacer el umbralado de las imágenes. Para dar un ejemplo del cálculo de la matriz de Co-ocurrencia se utilizará una imagen muestra de 4x4 píxeles de tamaño, en donde los valores corresponden a niveles de grises. La figura 57 presenta la imagen de muestra [21].



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

Figura 57. Imagen de muestra sobre la que se harán los cálculos de medidas texturales.

B.1 Cálculo de la matriz de Co-ocurrencia.

La Matriz de Co-ocurrencia considera la relación espacial entre dos píxeles, llamados pixel de referencia y pixel vecino. Por ejemplo, si se escoge el pixel vecino, que está situado un pixel a la derecha de cada pixel de referencia, esto se expresa como (1,0): 1 pixel en la dirección x, 0 pixel en la dirección y. Cada pixel en la ventana se va convirtiendo sucesivamente en el pixel de referencia, empezando por el ubicado arriba a la izquierda y finalizando abajo a la derecha. En este trabajo se utilizó un pixel de separación (un pixel de referencia y su inmediato vecino) (1,0).

Tomando en cuenta esto, se tiene que conocer las posibles combinaciones de niveles de grises que pudieran ocurrir en la imagen, para el caso de nuestra imagen de prueba todas las posibles combinaciones se en la tabla 13.

Tabla 13. Todas las posibles combinaciones de niveles de gris de la imagen de prueba.

Píxel vecino **Píxel de referencia**	**0**	**1**	**2**	**3**
0	(0,0)	(1,0)	(2,0)	(3,0)
1	(0,1)	(1,1)	(2,1)	(3,1)
2	(0,2)	(1,2)	(2,2)	(3,2)
3	(0,3)	(1,3)	(2,3)	(3,3)

De la tabla anterior se deduce que cada una de las celdas debe ser llenada con el número de veces que aparece la combinación que representa en la imagen. En el primer caso, (0,0), la celda contendrá el número de veces que un píxel con valor de 0 (píxel vecino) aparece a la derecha de un píxel con valor 0 (píxel de referencia). Siguiendo esta regla, la matriz Co-ocurrencia quedará de la siguiente manera, tabla 14.

Tabla 14. Matriz de Co-ocurrencia (1,0) para la imagen de prueba.

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

Una vez que se tiene el cálculo de la matriz de Co-ocurrencia es necesario transformarla en una matriz simétrica a su diagonal, ya que para los cálculos siguientes es necesario que se cuente con esta característica. Para lograr esto es necesario hacer la suma de su matriz transpuesta (intercambiar filas por columnas). La tabla 15 muestra la matriz después de la transformación.

Tabla 15. Matriz simétrica de la imagen de prueba.

4	2	1	0
2	**4**	0	0
1	0	**6**	1
0	0	1	**2**

Posteriormente se realiza el cálculo de la probabilidad, es decir, el número de veces que una combinación se presenta, sobre el número de posibles combinaciones en la imagen. En este caso el número total de posibles combinaciones es de 24, tomando en cuenta la relación de (1,0) (12 combinaciones) y su transpuesta (12 combinaciones), figura 58.

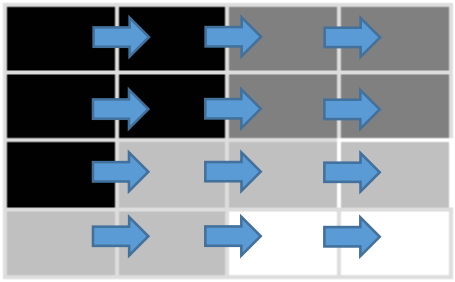


Figura 58. Pares de píxeles para una relación de (1,0).

Al conocer el número total de posibles combinaciones es necesario saber que probabilidad hay de que suceda una combinación en la imagen original, como ejemplo tomaremos la Combinación (2,2) ocurre 6 veces sobre las 24 posibles, por lo que la probabilidad es de 0.250. A esta operación se le llama normalización de la matriz. En la tabla 16 se detallan una aproximación a las probabilidades para las combinaciones que pueden en la imagen de prueba.

Tabla 16. Matriz normalizada de la imagen original.

0.166	0.083	0.042	0
0.083	0.166	0	0
0.042	0	0.250	0.042
0	0	0.042	0.083

B.2 Cálculo de las medidas de textura.

A continuación se describe la forma de obtener las tres medidas de textura utilizadas en el presente trabajo, siendo éstas la Homogeneidad, el Contraste y la Disimilaridad. Los cálculos se hacen con base en la matriz de Co-ocurrencia calculada en la sección anterior para la imagen de muestra.

B.2.1 Homogeneidad.

La Homogeneidad es alta cuando la matriz de coocurrencia se concentra a lo largo de la diagonal. Esto ocurre cuando la imagen es localmente homogénea de acuerdo al tamaño de la ventana. La medida de homogeneidad es calculada a partir de la siguiente ecuación:

$$\sum_{i,j=0}^{N-1} \frac{P_{i,j}}{1} + (i-j)^2$$

Donde $P_{i,j}$ es la probabilidad de Co-ocurrencia de los valores de gris i y j para una distancia dada.

De esta forma al aplicar la ecuación anterior a los valores de la matriz normalizada, se tiene la matriz resultado mostrada en la tabla 17.

Tabla 17. Matriz resultado de aplicar la ecuación de homogeneidad a la matriz normalizada.

0.166	0.042	0.008	0
0.042	0.166	0	0
0.008	0	0.250	0.021
0	0	0.021	0.083

Y al hacer la sumatoria de todos los resultados se obtiene el valor de la homogeneidad el cual es de **0.807**.

B.2.2 Contraste.

Es lo opuesto a la homogeneidad, es decir es una medida de la variación local en una imagen. Tiene un valor alto cuando la región dentro de la escala de la ventana tiene un alto contraste. La ecuación para el cálculo del contraste es la siguiente:

$$\sum_{i,j=0}^{N-1} P_{i,j}(i-j)^2$$

B.2.3 Disimilaridad.

Es una medida de textura parecida al contraste, es alta cuando la región tiene un alto contraste. La ecuación para determinar la medida de disimilaridad es la siguiente:

$$\sum_{i,j=0}^{N-1} P_{i,j}\,|i,j|$$

Anexo C.
Código fuente

C.1 Código del entrenamiento para la RNP en el problema de reconocimiento de patrones.

Las siguientes líneas de código representan las funciones principales de la aplicación de consola desarrollada sobre lenguaje C++ para el entrenamiento de la RNP con el objetivo de resolver el problema de reconocimiento de patrones mediante.

C.1.1 Función principal.

En primer lugar se presenta la función principal, que es la encargada de seleccionar y extraer la información del archivo que contiene el conjunto de entrenamiento, además de que contiene el algoritmo de Evolución Diferencial; una vez que se terminan las generaciones del ED, la función escribe los valores obtenidos de la maximización de la función fitness, en un archivo que será seleccionado posteriormente para realizar la validación.

```
// multiclase.cpp : Defines the entry point for the console
application.
//

#include "stdafx.h"
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <malloc.h>
#include <windows.h>

// Caracteristicas de la base de datos
//iris 3 clases
//wine 3 clases
//glass 7 clases
//liver 2 clases
#define clases 5
```

```
#define GENERACIONES 500 //Número de generaciones del ED
/*****************************************/


double aptitud(double w[], int tam, double pat[], int ras, int
cls, int tamClases[], int filas, int pesos);
int calcOculta(int a);

int main()
{

FILE *fp;
int PESOS_EXTRA = 1;
int *tamClases; // vector en donde se almacenará el número de
patrones por clase
tamClases = (int*)malloc(clases * sizeof(int));
double *patrones;

for (int i= 1; i < 11; i++)/// quitar
     {    /// quitar
     FILE *arch;     /// quitar
     char *indice1;/// quitar
     indice1 = (char*)malloc(sizeof(char));/// quitar
     char *indice2;/// quitar
     indice2 = (char*)malloc(sizeof(char));/// quitar
     char pesos[] = "pesos.dat";/// quitar
     char archivo[] = "Entrena.dat";/// quitar

          itoa(i, indice1, 10);/// quitar
          itoa(i, indice2, 10);/// quitar
//FILE *arch;//apuntador para el archivo que contiene los patrones
clock_t t_ini, t_fin;
srand (time(NULL)); // Se obtiene la semilla para la generación de
números aleatorios

/**********************se indica el
archivo*****************************/
/******/char ruta[] = "./MFCC/";
strcat(indice2, archivo);
strcat(ruta, indice2);
/*********/
/*************************************************/
char c;// caracter que se extrae del archivo
int rasgos = 0; // Número de rasgos
int flag = 0; // Bandera para leer sólo una vez el número de
columnas
int col = 0; // Número de columnas
int fil = 1; // número de filas = número de patrones totales

//se inicializa el vector de tamaños de clase
for (int i = 0; i < clases; i++)
          tamClases[i] = 0;
```

```
//Se cuentan los rasgos y patrones
    if ((fp = fopen(ruta,"r+"))==NULL)
          return(printf("Error: No se encuentra el archivo\n\nFin
del programa\n\n"));
     else
          {
               c = getc(fp);
               while (c!= EOF)
               {
                    if ((c == '\t' || c == ' ')&& flag == 0)
                         col++;     // cuenta el número de comas
lo que significa que obtiene el número de rasgos.
                                        // NOTA: falta agregar el
número que corresponde a la columna de clase.
                    else if (c == '\n')
                    {
                         fil++;
                         flag = 1;
                    }
                    //putchar(c);
                    c = getc(fp);   // siguiente caracter del
archivo
               }
               rasgos = col;   // se suma uno más por la columna
de clases
               col = col + 1;
          }

fclose(fp);

int n = rasgos + rasgos * PESOS_EXTRA + rasgos * clases + clases *
PESOS_EXTRA + clases + PESOS_EXTRA;/*PESOS PARA CADA UNO DE LOS
RASGOS ADEMÁS DE LOS FACTORES DE la funcion de transformacion*/
int g = 0; //Contador para el número de epocas del algoritmo
genético
int jrand;// Número aleatorio para hacer o no la cruza.
double F = 0.9; // Factor para el algoritmo genético, tasa de
mutación
double Cr = 0.8; //Factor para el algoritmo genético, tasa de
recombinación
int np = 60; // Tamaño de la población inicial
double X_MAX = 1; //valor máximo para la generación de números
aleatorios
double X_MIN = 0; //Valor minimo para la generación de números
aleatorios
int r1; int r2; int r3; //Selección de los padres


/* Declaración de las matrices y vectores para la población
inicial y aptitudes*/

double *poblacion; //población inicial del algoritmo de ED
```

```
double *prueba; //población de prueba
double *hijos; //matriz de hijos generados por la población
inicial
/* Vectorres para almacenar los valores de aptitud de las
poblaciones inicial, de prueba e hijos*/
double *aptPob;
double *aptPrueba;
double *aptHijos;
double *vector;
// array = (int *)malloc (N*sizeof(int));
//Reserva de memoria parala matrices
poblacion = (double*)malloc ((np * n) * sizeof(double));
prueba = (double*)malloc ((np * n) * sizeof(double));
hijos = (double*)malloc ((np * n) * sizeof(double));

//Reserva de memoria para los vectores de aptitud
aptPob    = (double*)malloc (np * sizeof(double));
aptPrueba = (double*)malloc (np * sizeof(double));
aptHijos  = (double*)malloc (np * sizeof(double));
vector    = (double*)malloc ( n * sizeof(double));// vector donde
se almacenan los pesos a evaluar en la aptitud

/* SE HACE LA LECTURA DEL FICHERO PARA TENER LOS PATRONES EN UN
VECTOR*/

patrones = (double*)malloc((fil * col) * sizeof(double));// Se
crea el vector para los patrones

fp = fopen(ruta,"r"); //Se abre el fichero

    if(fp==NULL)
        printf("error al abrir el archivo *.dat");

    for(int i = 0; i < fil; i++)
    {
        for(int j = 0; j < col; j++)// Se suma 1 porque al final
de cada patrón se tiene la clase a la que pertenece
        {
            fscanf(fp, "%lg", &patrones[i * col + j]); //se guarda
en un array el número leido del archivo

                if (j == rasgos)
                     tamClases[(int)patrones[i * col + rasgos]-
1]++;

        }
     //printf("\n ");
    }
fclose(fp);//Se cierra el archivo
/*TERMINA LA LECTURA DEL ARCHIVO Y YA SE TIENEN LOS PATRONES EN UN
VECTOR*/
```

```
/* INICIA EL ALGORITMO DE EVOLUCION DIFERENCIAL */
/* El algoritmo inicia con la creación de la población inicial de
manera aleatoria*/

t_ini = clock();//Inicia el conteo del tiempo

for (int i = 0; i < np; i++ )
{
      // Se aprovecha el loop para la inicializacion de los
vectores de aptitud
      aptPob[i] = 0;
      aptPrueba[i] = 0;
      aptHijos[i] = 0;

      for (int j = 0; j < n; j++)
      {
            poblacion[i * n + j] = (double)(rand()%101)/100;//Se
inicializa la población inicial del ED con valores entre 0 y 1
            vector[j] = poblacion[i * n + j];
      }
      aptPob[i] = aptitud(vector/*pesos a evaluar*/, n/*núm
pesos*/, patrones/*vector de patrones*/, rasgos, clases,
tamClases, fil, PESOS_EXTRA);


}// fin del for de llenado de poblacion

/* lanzamiento de las epocas del ED*/
while (g < GENERACIONES)
{
      for (int i = 0; i < np; i++)
      {
            r1 = rand()% (np);
            r2 = r1;
            r3 = r1;
            while (r1 == r2)
            {
                  r2 = rand()% (np);
            }

            while (r3 == r1 || r3 == r2)
            {
                  r3 = rand()% (np);
            }

/* Cruza de los padres*/
            for (int j = 0; j < n; j++)
            {
                  jrand = rand()% (np);
                  // Generación de la población de prueba
                  if ((double)(rand()%101)/100 < Cr ||  j == jrand)
```

```
                    prueba[i * n + j] = poblacion[r3 * n + j] +
F*(poblacion[r1 * n + j] - poblacion[r2 * n + j]);

               else
                    prueba[i * n + j] = poblacion[i * n + j];

               vector[j] = prueba[i * n + j];

          }// Fin de la cruza

          /*Enviar a calcular la aptitud de este vector de pessos
i*/
     aptPrueba[i] = aptitud(vector/*pesos a evaluar*/, n/*núm
pesos*/, patrones/*vector de patrones*/, rasgos, clases,
tamClases, fil, PESOS_EXTRA);

// Se crean a los hijos de la generación, mutación

          if (aptPob[i] <= aptPrueba[i])
                    for (int j = 0; j < n; j++)
                    {
                         hijos[i * n + j] = prueba[i * n + j];
                         vector[j] = hijos[i * n + j];

                    }
          else
               for (int j=0; j < n; j++)
               {
                    hijos[i * n + j] = poblacion[i * n + j];
                    vector[j] = hijos[i * n + j];
               }


          /*#Envuar a aclacular la aptitud de los hijos*/
     aptHijos[i] = aptitud(vector/*pesos a evaluar*/, n/*núm
pesos*/, patrones/*vector de patrones*/, rasgos, clases,
tamClases, fil, PESOS_EXTRA);
     }//Fin del for (i)

for (int i = 0; i < np; i++)
{
     for (int j = 0; j< n; j++)
          poblacion[i * n + j] = hijos[i * n + j];

     aptPob[i] = aptHijos[i];
}



g++;
}// fin del while de generaciones
int indice = 0;
```

```
double mayor = 0;
double menor = 100000000;

for (int i=0; i < np; i++)
{
      if (aptPob[i] > mayor)
      {
      mayor = aptPob[i];
      indice = i;
      }
      if (aptPob[i] < menor)
            menor = aptPob[i];
}
t_fin = clock();
printf("\n\n\nEl valor de aptitud mayor es= %.16g", mayor);
printf("\nEl valor de aptitud menor es= %.16g\n\nValor de los
pesos: \n", menor);

if ((arch = fopen(strcat(indice1,pesos), "wb+")) == NULL)
           { printf ( "Error en apertura del fichero para escritura
\n " );
           }
else
           {
      for (int i=0; i <n;i++)
            fprintf(arch,"%lf\t",poblacion[indice * n + i]);

fprintf (arch,"\n");
      }
fclose(arch);

for (int i=0; i < n; i++)
printf("%e  ", poblacion[indice * n + i]);
printf("\nTiempo de ejecucion = %.16g seg",(double)(t_fin - t_ini)
/ CLOCKS_PER_SEC);
printf("\n\nNumero de pesos calculados= %d",n);
//printf("\nEl indice del menor es: %d", indice);

// se libera la memoria reservada
free(poblacion);
free(prueba);
free(hijos);
free(aptPob);
free(aptPrueba);
free(aptHijos);
free(vector);
}//// Quitar porque es el del for de las 10 c0rridas
scanf("%d",0);
return 0;
}
```

C.1.2 Neurona de Izhikevich.

El código siguiente, escrito en lenguaje C++, es el encargado de resolver el modelo de Izhikevich para realizar la simulación de la neurona durante el periodo de tiempo establecido. Es en esta función donde debe ser seleccionada la función de transformación para calcular la corriente de entrada hacia la neurona.

```
#include "stdafx.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define T  1000
#define TAU  1
#define N T/TAU


double izhikevichE(double w[], double x, int tam)
{               // Inicio de la funcion neurona


double contador = 0;
int C = 100;
int vt = -40;
double k = 0.7;
double a = 0.03;
int b = -2;
int c = -50;
int  d = 100;
double vpeak = 35; //Altura máxima del pico generado por la
neurona, umbral de disparo



double v[N];
double u[N];
double vr = -60;

double I = 55; // corriente de entrada a la neurona

double p1 = w[0]; // peso para el producto punto y la norma
double peso= w[1];
double norma = 0;

/*++++++++++++++++++++++++++++++++++RBF++++++++++++++++++++++++++++
++++++++++++++++++++++++*/
//norma = sqrt(pow(x-peso,2));
```

```
//I += exp(-(pow(norma,2)/ (2 * pow(p1,2))));
/*++++++++++++++++++++++++++++++++++RBF++++++++++++++++++++++++++
++++++++++++++++++++++++*/

/*++++++++++++++++++++++++++++++++++Polinoial++++++++++++++++++++
++++++++++++++++++++++++++++++*/
//I += pow((peso * x) + 1,p1);
/*++++++++++++++++++++++++++++++++++Polinomial++++++++++++++++++++
++++++++++++++++++++++++++++++*/


/*++++++++++++++++++++++++++++++++++Productos++++++++++++++++++++
++++++++++++++++++++++++++++++*/
//I += x * peso * p1;// Claculo de la corriente de entrada hacia
la neurona funcion productos
/*++++++++++++++++++++++++++++++++++Productos++++++++++++++++++++
++++++++++++++++++++++++++++++*/
for (int i=0; i < N; i++)
{
v[i] = vr; //Inicialización del vector v con los valores de vr
u[i] = 0;  // Inicialización del vector u
}

 for (int i = 0; i < N-1; i++)
 { // Inicia el metodo de euler
    v[i+1] = v[i] + TAU *(k *( v[i] - vr) * (v[i] - vt) - u[i] +
I) / C;
    u[i+1] = u[i] + TAU * a * (b * (v[i] - vr) - u[i]);
    if (v[i+1] >= vpeak)
     {
        v[i] = vpeak;
        v[i + 1] = c;
        u[i + 1] = u[i + 1] + d;
     }

 }        // Termina metodo de euler


for (int i = 0; i < N; i++) /* Conteo de los picos generados por
la neurona*/
{
    if ( v[i] == vpeak )
        contador ++;
}

return contador;

}//Fin de la funcion neurona
```

C.1.3 Función de aptitud.

La función de aptitud es la encargada de determinar qué tan viable es la solución propuesta por el individuo que se está evaluando, en este caso, es la función que determina que tan bien se está haciendo el entrenamiento del conjunto. El código utilizado para calcular esta función de aptitud es el siguiente y está escrito en lenguaje C++.

```
/*Funcion de ptitud utilizada para encontrar los pesos mediante el
algoritmogenetico*/

#include "stdafx.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

double izhikevich(double w[], double x[], int tam, int
pesosExtra);
double izhikevichS(double w[], double x[], int tam);
double izhikevichE(double w[], double x, int tam);
double promedio(double a[], int tam);
double desviacion(double a[], int tam);
int calcOculta(int a);
double clasificacion(double picos[], int clases,int fil, double
promedios[], int tamClases[]);
double aptitud(double w[], int tam, double patrones[], int rasgos,
int clases, int tamClases[], int fil, int pesosExtra)
{
int neuronasOcultas = clases;
int pesosEntrada = rasgos + rasgos * pesosExtra; // pesos por
neurona (una por rasgo)
int pesosIntermedia = rasgos * neuronasOcultas + neuronasOcultas *
pesosExtra;// pesos de la capa intermedia
int pesosSalida = neuronasOcultas + pesosExtra; // pesos de la
capa de salida
double *capaEntrada;
double *capaOculta;
double *capaSalida;
capaEntrada = (double*)malloc(pesosEntrada * sizeof(double));
//Vector para almacenar los pesos que seran enviados a la neurona
capaOculta = (double*)malloc(pesosIntermedia * sizeof(double));
//Vector para almacenar los pesos que seran enviados a la neurona
capaSalida = (double*)malloc(pesosSalida * sizeof(double));
//Vector para almacenar los pesos que seran enviados a la neurona
double apt = 0; // valor que será retornado de la funcion
double *prom; // promedio de picos por clase
```

```
double *desviaciones;
prom = (double*)malloc(clases * sizeof(double));//reserva de
memoria dinamica
desviaciones = (double*)malloc(clases * sizeof(double));//reserva
de memoria dinamica para las desviaciones estandar de los picos
por las clases
double *euclidiana; // distancia euclidiana por cada clase
euclidiana = (double*)malloc(clases * sizeof(double));// reserva
de memoria dinamica
double desv = 0;
double sumE = 0;
double E = 1;
double *picos;// vector de los picos generados por la capa de
salida de todos los patrones
picos = (double*)malloc(fil * sizeof(double));
double *picosEntrada;// picos generados por las neuronas de la
capa de entrada
picosEntrada = (double*)malloc(rasgos * sizeof(double));
double *picosOculta;// picos generados por la capa oculta
picosOculta = (double*)malloc(neuronasOcultas * sizeof(double));
int suma = 0;
double *x;// vector de pesos por neurona de entrada
double *xx;// vector de picos para los promedios
double *xxx;//vector de pesos por neurona intermedia
xxx = (double*)malloc((rasgos + pesosExtra) * sizeof(double));
x = (double*)malloc((1 + pesosExtra) * sizeof(double));

/*SEPARACIÓN DE LOS PESOS EN LAS DIFERENTES CAPAS*/
for (int i = 0; i < tam; i++)
{
     if ( i < pesosEntrada){
          capaEntrada[i] = w[i];// Se asigna los pesos de la
primera capa
          //printf("Entrada: %lf \n",capaEntrada[i]);
     }
     else if ( i < pesosEntrada + pesosIntermedia){
               capaOculta[i - pesosEntrada] = w[i];// Se asignan
los pesos de la segunda capa
               //printf("Intermedia: %lf \n",capaOculta[i -
pesosEntrada]);
     }
     else if ( i >= pesosEntrada + pesosIntermedia){
               capaSalida[i - (pesosEntrada + pesosIntermedia)] =
w[i];
               //printf("Salida: %lf \n",capaSalida[i -
(pesosEntrada + pesosIntermedia)]);
     }
}
for (int i=0; i < fil; i++) //se pasan todos los patrones por la
neurona pulsante para obtener la frecuencia generada por cada uno
{
```

```
      for (int j = 0; j < rasgos; j++)//se´pasan por las neuronas
de entrada
      {
            for (int k = 0; k < pesosExtra + 1; k++){
                        x[k] = capaEntrada[j * (pesosExtra + 1) + k];
            //printf("%lf ", x[k]);
            }
                  picosEntrada[j] = izhikevichE(x, patrones[i *
(rasgos + 1) + j], pesosExtra + 1);// Capa de neuronas de entrada

      }

      for (int j = 0; j < neuronasOcultas; j++)
      {
            for (int k = 0; k < rasgos + pesosExtra; k++){
                  xxx[k] = capaOculta[j * (rasgos + pesosExtra) +
k];// Pesos correspondientes a la neurona

            }
      picosOculta[j] = izhikevich(xxx, picosEntrada, rasgos +
pesosExtra, pesosExtra);// capa de neuronas ocultas
      }

picos[i] = izhikevich(capaSalida, picosOculta, pesosSalida,
pesosExtra);// los picos del patron después de pasar por toda la
red
printf("%d ", (int)picos[i]);
} // En este punto ya se tiene todos los picos de los patrones,
ahora inicia el calculo del valor de aptitud

suma = 0;
//Calculo del promedio por clase
for (int i = 0; i < clases; i++)
{
      xx = (double*)malloc(tamClases[i] * sizeof(double));

      for (int j = suma; j < suma + tamClases[i]; j++)
      {
            xx[j - suma] = picos[suma + (j - suma)];
            //printf("%d ", picos[suma + (j - suma)]);
      }
suma = suma + tamClases[i];
prom[i] = promedio(xx,tamClases[i]);
//printf("\n");
free(xx);
}
//printf("\n");
//desv = desviacion(prom, clases);
apt = clasificacion(picos,clases, fil, prom, tamClases);
//printf("SumaE %lf\n", sumaE);
printf("\nAptitud: %lf \n\n",apt);
return apt;}
```

C.1.4 Clasificación.

Esta es la función encargada de hacer el cálculo de la distancia euclidiana, y por consiguiente, hacer la clasificación de los patrones de acuerdo a su frecuencia de disparo; como en los códigos anteriores, éste está escrito en lenguaje C++.

```
#include "stdafx.h"
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <math.h>


double clasificacion(double picos[], int clases, int fil, double
promedios[], int tamClases[])
{
int *clasCorr; //vector para la clasificacion correcta de los
patrones, sirve para la comparación
clasCorr = (int*)malloc((fil) * sizeof(int));
int *claseProm; //vector para la clasificacion de los patrones de
acuerdo a sus picos
claseProm = (int*)malloc((fil) * sizeof(int));
double distEuc = 0; //variable para la distancia euclidiana con
respecto de todos los promedios de las clases
double euc = 0;// variable para almacenar la menor distancia
euclidiana del patron
//distEuc = (int*)malloc((clases) * sizeof(int));
double iguales = 0;//variable para contar el número de patrones
correctamente clasificados
double porcentaje = 0;//porcentaje de clasificación correcta

     for (int i = 0; i < fil; i++) // se llena el vector de la
clasificación de acuerdo al promedio de los picos generados por
las clases
     {
          for (int j = 0; j < clases; j++)
          {
           distEuc = sqrt(pow(picos[i] - promedios[j],2));// se
calcula la distancia con respecto a cada uno de los promedios
                if (j == 0)
                {
                     claseProm[i] = j;
                     euc = distEuc;
                }
                     else if(distEuc < euc)
                     {
                               claseProm[i] = j;
                               euc = distEuc;
```

```
                        }
                }
                //printf("%d ",claseProm[i]);
        }
//printf("\n");
// se hace la clasificación de los picos por medio de la distancia
euclidiana
int suma = 0;
for (int i = 0; i < clases; i++) // se llena el vector de la
clasificación correcta
        {
                for (int j = suma; j < suma + tamClases[i]; j++)
                {
                        clasCorr[suma + (j - suma)] = i;
                        //printf("%d ",i);
                }
        suma = suma + tamClases[i];
        }
//printf("\n");
//inicia la comparación
        for (int i = 0; i < fil; i++) // hace la comparación para
        {
                if (clasCorr[i] == claseProm[i])
                        iguales++;
        }
porcentaje = iguales / fil;
return porcentaje;
}
```

C.2 Código utilizado en el problema de umbralado de imágenes.

En este caso, como se mencionó anteriormente el entrenamiento se hace mediante el mismo algoritmo de evolución diferencial, esta vez programado en lenguaje MATLAB, por lo tanto no se expondrá en este apartado ya que maneja la misma estructura; sin embargo, sí se presentará el código utilizado para calcular la aptitud del vector evaluado.

C.2.1 Función de aptitud para la neurona encargada del umbralado.

```
function apt=aptitud(patrones,pesos,kernel)

[numPatrones columnas]=size(patrones);

C=100; vr=-60; vt=-40; k=0.7;
a=0.03; b=-2; c=-50; d=100;
vpeak=35;
T=1000; tau=1;
n=round(T/tau);
v=vr*ones(1,n); u=0*v;
clases=2;
% pClases=25;%%%%%%%Hay que cambiar a 2 para el problema XOR
E=1;
%%[numPatrones colPatr]=size(patrones);%%Numero de filas de los
patrones y numero de columnas de cada patrón
prom=zeros(1,clases);%%%Arreglo para guardar el promedio de los
picos generados por clase
desviacion=0;%%Desviacion estandar
euclidiana=zeros(1,clases);%%%%Arreglo para almacenar la suma de
las distancias euclidianas entre los patrones de cada clase
apt=0;%%Valor de la aptitud
contApt=zeros(1,numPatrones);%%Vector para almacenar el numero de
picos por patron


for z=1:numPatrones %%%Todos los patrones
cont=0;

x = patrones(z,:);
x = x(1:columnas-1);

if strcmp(kernel,'RBF')
    w=pesos(1:columnas-1);
    p1=pesos(columnas);
```

```
I=exp(-(norm(x-w))^2/(2*p1^2))+55;%%%funcion gaussiano RBF
%I=exp((-norm(x-w))^2/p1^2);%
else if strcmp(kernel,'poly')
        w=pesos(1:columnas-1);
        p1=pesos(columnas);
         I=((x*w'+1)^p1) + 55;%%funcion polinomico

    else if strcmp(kernel, 'productos')
        w=pesos(1:columnas-1);
        p1=pesos(columnas);
        I=(x*w'* p1) + 55;%%%funcion producto
        else
        w=pesos(1:columnas-1);
        I=(x*w') + 55;%%%funcion producto
        end
    end
end

for i=1:n-1
    v(i+1)=v(i)+tau*(k*(v(i)-vr)*(v(i)-vt)-u(i)+I)/C;
    u(i+1)=u(i)+tau*a*(b*(v(i)-vr)-u(i));
    if v(i+1) >=vpeak
        v(i) = vpeak;
        v(i+1) = c;
        u(i+1) = u(i+1)+d;

    end

end

for i=1:n
    if v(i)==35
        cont=cont+1;%%Numero de picos
    end
end

contApt(z)=cont;%% Se guarda el resultado del patrón
end%%Fin del recorrido a todos los patrones
contApt;
%%%INICIA LA FUNCION DE APTITUD
  pClases = numPatrones/clases;
%%Se cálcula el promedio de las dos clases
prom(1)=mean(contApt(1:pClases));
prom(2)=mean(contApt(pClases + 1:numPatrones));


% for i=0:clases-1
    suma=0;
    for j=1 : pClases-1%%desde el patron 1 hasta n-1 de la primera
clase
        for k= j+1 : pClases%%desde el patron siguiente al indice
j hasta n
```

```
            suma=suma + sqrt((contApt(j) - contApt(k))^2);%%suma
de la Distancia euclidiana por clase 1
        end
    end
    euclidiana(1)=suma;
    %%%%%%Se hace lo mismo pero para la clase 2
    suma=0;
    for j = pClases + 1 : numPatrones - 1
        for k = j + 1 : numPatrones
            suma=suma + sqrt((contApt(j) - contApt(k))^2);%%suma
de la Distancia euclidiana por clase
        end
    end
    euclidiana(2)=suma;

desviacion = std(prom);%%Desviacion estandar de los promedios de
las clases

sumE = sum(euclidiana) + E;%%Evita la división por cero
suma=0;

for i = 1:clases
    suma=suma + (sumE/prom(i));
end
apt=desviacion * (1/suma);%%aptitud de la solución
end
```

C.2.2 Matriz de Co-ocurrencia.

Se detalla el código utilizado para calcular la matriz de Co-ocurrencia, utilizada para extraer rasgos de textura de la imagen a umbralar, en este caso la función que realiza los cálculos recibe la imagen en escala de grises; esta función está escrita en lenguaje de MATLAB.

```
%% Cálculo de la Homogeneidad el contraste y la disimilaridad por
medio de la matriz de co-ocurrencia
function descriptores = coOcurrencia(img)

[N M] = size(img);
% valores = depura(img);
valores = 0:1:255;
tamV=length(valores);
matrizCO = zeros(tamV,tamV);%%Matriz en la que se almacenarán los
valores de ocurrencia con la relación espacial (1,0)
matrizCOT = zeros(tamV,tamV);
for i=1:tamV
    for j=1:tamV %%Recorre todas las posibilidades de combinacion
de los valores en escala de gris
        for k=1:N
            for l=1:M-1
                if img(k,l)==valores(j) && img(k,l+1)==valores(i);
                    matrizCO(j,i) = matrizCO(j,i)+1;
                end
            end
        end

    end
end
%% Se hace la transpuesta
for i=1:tamV
    for j=1:tamV
        matrizCOT(j,i) = matrizCO(i,j);
    end
end
%% Creacion de la matriz simetrica
matrizCO = matrizCO + matrizCOT;

%% Normalización de la  matriz, esta matriz es la que se utiliza
para el cálculo de los descriptores de la  textura

pares = (N * M - N) * 2;

for i=1:tamV
    for j=1:tamV
```

```
        matrizN(i,j) = matrizCO(i,j)/pares;%%Llenado de la matriz
normalizada
    end
end

%%% Cálculo de las medidas de textura%%%

%%Homegeneidad de la textura
h = 0;
for i = 1 : tamV
    for j = 1 : tamV
        matrizH(i,j) = matrizN(i,j)/(1+(valores(j)-
valores(i))^2);%%Cálculo de la homogeneidad
        h = h + matrizH(i,j);
    end
end

%%Cálculo del contraste de la textura
contraste = 0;
for i = 1 : tamV
    for j = 1 : tamV
        matrizCon(i,j) = matrizN(i,j)*(valores(j)-
valores(i))^2;%%Cálculo de la homogeneidad
        contraste = contraste + matrizCon(i,j);
    end
end

%%Cálculo de la disimilaridad de la textura
dis = 0;
for i = 1 : tamV
    for j = 1 : tamV
        matrizDis(i,j) = matrizN(i,j) * sqrt((valores(j)-
valores(i))^2);%%Cálculo de la homogeneidad
        dis = dis + matrizDis(i,j);
    end
end
% descriptores = [h contraste dis];
descriptores =[h contraste dis];
end
```

C.2.3 Clasificación de los píxeles de la imagen.

El siguiente código es el encargado de asignar uno de los dos valores posibles para crear la imagen binaria (1 ó 0), a través de la clasificación del píxel, mediante la frecuencia de disparo obtenida al pasar el vector de rasgos extraídos de éste, por la neurona entrenada. La clasificación como en el caso anterior se hace mediante la distancia euclidiana.

```
function class=clasifica(pulsos, prom)%%vector de picos de todos
los patrones y el número de patrones que corresponde a la clase1
clase=[1 0];%%Etiquetas para las clases
distancia=zeros(1,length(clase));%%Vector de distancias euclidiana
del patron con respecto al promedio de picos de cada clase

    for j = 1 : 2
        distancia(j)=sqrt((pulsos- prom(j))^2);%%Se calcula la
distancia euclidiana con los promedios de las dos clases
    end

    if prom(1) == 0 || prom(2)==0 || prom(1)== prom(2) ||
distancia(1) == distancia(2);%% Si las distancias son iguales, se
manda a la calse 0
        class = 0;
    else
        class = clase(find(distancia == min(distancia)));%% Se
busca la etiqueta con la distancia menor
    end

end
```