Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería Mecánica Eléctrica

DISEÑO MEDIANTE MATLAB DE UN SISTEMA SOBRE RECONOCIMIENTO DE IMÁGENES BASADO EN EL APRENDIZAJE DE REDES NEURONALES ARTIFICIALES

Carlos Humberto Bucú Saz

Asesorado por la Inga. Ingrid Rodríguez de Loukota

Guatemala, agosto de 2012

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

DISEÑO MEDIANTE MATLAB DE UN SISTEMA SOBRE RECONOCIMIENTO DE IMÁGENES BASADO EN EL APRENDIZAJE DE REDES NEURONALES ARTIFICIALES

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

CARLOS HUMBERTO BUCÚ SAZ
ASESORADO POR LA INGA. INGRID RODRÍGUEZ DE LOUKOTA

AL CONFERÍRSELE EL TÍTULO DE

INGENIERO ELECTRÓNICO

GUATEMALA, AGOSTO DE 2012

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

NÓMINA DE JUNTA DIRECTIVA

DECANO	Ing. Murphy Olympo Paiz Recinos
VOCAL I	Ing. Alfredo Enrique Beber Aceituno
VOCAL II	Ing. Pedro Antonio Aguilar Polanco
VOCAL III	Ing. Miguel Ángel Dávila Calderón
VOCAL IV	Br. Juan Carlos Molina Jiménez
VOCAL V	Br. Mario Maldonado Muralles
SECRETARIO	Ing. Hugo Humberto Rivera Pérez

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

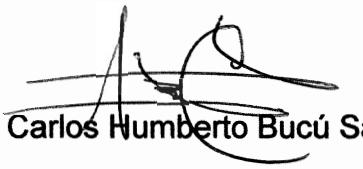
DECANO	Ing. Murphy Olympo Paiz Recinos
EXAMINADOR	Ing. Julio César Solares Peñate
EXAMINADOR	Ing. Byron Odilio Arrivillaga Méndez
EXAMINADOR	Ing. Marvin Marino Hernández Fernández
SECRETARIA	Inga. Marcia Ivónne Véliz Vargas

HONORABLE TRIBUNAL EXAMINADOR

En cumplimiento con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

DISEÑO MEDIANTE MATLAB DE UN SISTEMA SOBRE RECONOCIMIENTO DE IMÁGENES BASADO EN EL APRENDIZAJE DE REDES NEURONALES ARTIFICIALES

Tema que me fuera asignado por la Dirección de la Escuela de Ingeniería Mecánica Eléctrica, con fecha noviembre de 2010.



Carlos Humberto Bucú Saz

Guatemala 16 de marzo del 2012

Ingeniero
Carlos Eduardo Guzmán Salazar
Coordinador del Área de Electrónica
Escuela de Ingeniería Mecánica Eléctrica
Facultad de Ingeniería, USAC.

Estimado Ingeniero Guzmán.

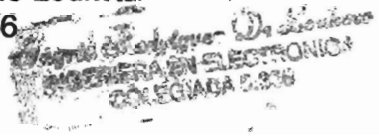
Me permito dar aprobación al trabajo de graduación titulado: **"DISEÑO MEDIANTE MATLAB DE UN SISTEMA SOBRE RECONOCIMIENTO DE IMÁGENES BASADO EN EL APRENDIZAJE DE REDES NEURONALES ARTIFICIALES"**, del señor **Carlos Humberto Bucú Saz**, por considerar que cumple con los requisitos establecidos.

Por tanto, el autor de este trabajo de graduación y, yo como asesora, nos hacemos responsables por el contenido y conclusiones del mismo.

Sin otro particular, me es grato saludarle.

Atentamente,

Inga. Ingrid Rodríguez de Loukota
Colegiada 5,356
Asesora



UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERIA

REF. EIME 20. 2012.

El Director de la Escuela de Ingeniería Mecánica Eléctrica, después de conocer el dictamen del Asesor, con el Visto Bueno del Coordinador de Área, al trabajo de Graduación del estudiante; Carlos Humberto Bucú Saz titulado: **"DISEÑO MEDIANTE MATLAB DE UN SISTEMA SOBRE RECONOCIMIENTO DE IMÁGENES BASADO EN EL APRENDIZAJE DE REDES NEURONALES ARTIFICIALES"**, procede a la autorización del mismo.

Ing. Guillermo Antonio Puente Romero

GUATEMALA, 25 DE ABRIL 2,012.

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERIA

Ref. EIME 13. 2012
Guatemala, 12 de abril 2012.

Señor Director
Ing. Guillermo Antonio Puente Romero
Escuela de Ingenieria Mecànica Elèctrica
Facultad de Ingenieria, USAC.

Señor Director:

Me permito dar aprobación al trabajo de Graduación titulado: "DISEÑO MEDIANTE MATLAB DE UN SISTEMA SOBRE RECONOCIMIENTO DE IMÁGENES BASADO EN EL APRENDIZAJE DE REDES NEURONALES ARTIFICIALES", del estudiante Carlos Humberto Bucú Saz, que cumple con los requisitos establecidos para tal fin.

Sin otro particular, aprovecho la oportunidad para saludarle.

Atentamente,
ID Y ENSEÑAD A TODOS

Ing. Carlos Eduardo Guzmán Salazar
Coordinador de Electrónica

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DIRECCION ESCUELA DE INGENIERIA MECANICA ELECTRICA
FACULTAD DE INGENIERIA

CEGS/sro

Universidad de San Carlos
De Guatemala

Facultad de Ingeniería
Decanato

Ref. DTG. 389-2012

El Decano de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer la aprobación por parte del Director de la Escuela de Ingeniería Mecánica Eléctrica, al trabajo de graduación titulado: **DISEÑO MEDIANTE MATLAB DE UN SISTEMA SOBRE RECONOCIMIENTO DE IMÁGENES BASADO EN EL APRENDIZAJE DE REDES NEURONALES ARTIFICIALES,** presentado por el estudiante universitario **Carlos Humberto Bucú Saz,** autoriza la impresión del mismo.

IMPRÍMASE.

Ing. Murphy Olympo Paiz Recinos
Decano

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DECANO
FACULTAD DE INGENIERIA

Guatemala, agosto de 2012

/cc

ACTO QUE DEDICO A:

Eliceo Bucú y Florinda Saz	Dedicado a mis padres, como muestra de agradecimiento a su amor, esfuerzo y dedicación.

AGRADECIMIENTOS A:

Mis padres	Eliceo Bucú y Florinda Saz, por su motivación y apoyo para poder alcanzar esta meta en mi vida.
Mis hermanos	Dr. Jose Manuel, Olga, y Josue Alexander, por el apoyo y cariño.
Mis amigos	Por su amistad sincera y los momentos que hemos compartido.
Mi asesora	Inga. Ingrid Rodríguez de Loukota, Por su colaboración en la realización de este trabajo de graduación.

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES

FIGURAS

TABLAS

LISTA DE SÍMBOLOS

Símbolo	Significado
A o B	Apertura de A por el elemento B
cov	Covarianza
A • B	Clausura de A por el elemento B
≠	Distinto de
A⊕B	Dilatación de A por B
AΘB	Erosión de A por B
TIFF	Formato de imagen, archivo de imagen con etiqueta
GIF	Formato de imagen, intercambios de gráficos
BMP	Formato de imagen, mapa de bits monocromo
PNG	Formato de imagen, gráficos de red portátiles
sing	Función signo
e	Función exponencial
μ	Imagen promedio
=	Igual
∩	Intersección entre dos conjuntos
ξ	La mediana de un conjunto
$\mathbf{X}^{\mathbf{T}}$	La transpuesta de X
>	Mayor que
≤	Menor igual que
≥	Mayor igual que
<	Menor que
i	Número de las capas en una red neuronal
ϵ	Pertenece a

%	Porcentaje
$\prod$	Producto de
Ø	Se dice de dos conjuntos que poseen elementos distintos
$\sum$	Sumatoria de
$\subseteq$	Subconjunto de o igual a
|	Tal que
λ	Valor propio
var	Varianza

GLOSARIO

Algoritmo	Conjunto de pasos que llevan a la solución de un problema.
Axón	Fibra larga que lleva la señal desde el cuerpo de la célula hacia otras neuronas.
BSS	Viene de la abreviatura *Blind Separation Source.*
CFN	Viene de la abreviatura *cascade forward neural network.*
Convolución	Operador matemático que transforma dos funciones *f* y *g* en una tercera función que en cierto sentido representa la magnitud en la que se superponen *f* y una versión trasladada e invertida de *g*.
Cuantización	Es la conversión de una señal discreta en el tiempo evaluada de forma continua a una señal discreta en el tiempo discretamente evaluada.
Dendritas	Son los árboles receptores de la red neuronal, son como fibras nerviosas que cargan de señales eléctricas el cuerpo de la célula.

Derivada	La derivada de una función es una medida de la rapidez con la que cambia el valor de dicha función según cambie el valor de su variable independiente.
Dilatar	Proceso morfológico que consiste en la unión de píxeles que se relacionan entre sí en una imagen digital.
Erosión	Proceso morfológico que consiste en la separación de píxeles que se relacionan entre sí en una imagen digital.
Emular	Proceso que trata de imitar las características de algún dispositivo o las funciones de algún proceso, procurando igualarlo o superarlo.
Filtro digital	Es un sistema que, dependiendo de las variaciones de la señal de entrada en el tiempo y amplitud, se realiza un procesamiento matemático sobre dicha señal obteniéndose en la salida el resultado del procesamiento matemático o la señal de salida.
Funciones de transferencia	Relación que existen entre la señal de salida con respecto a la señal de entrada.
Histograma	El histograma de una imagen es una representación del número de píxeles de cierto nivel de gris en función de los niveles de gris.

Inhibitorios	Potencial eléctrico que se produce en el interior de una dendrita que se opone a la formación de un potencial de acción.
JPEG	Viene de la abreviatura *Joint Photographic Experts Group.*
LDA	Viene de la abreviatura *Linear discrimination analysis*
ICA	Viene de la abreviatura *independent component analysis*.
Matriz	Arreglo de datos distribuidos en filas y columnas.
Morfología	Es un método no lineal de procesar imágenes digitales basándose en la forma. Su principal objetivo es la cuantificación de estructuras geométricas.
Muestreo	El muestreo consiste en el proceso de conversión de señales continuas a señales discretas en el tiempo. Este proceso es realizado midiendo la señal en momentos periódicos del tiempo.
Neurona	Célula nerviosa con características distintas a las demás células, consta de un cuerpo celular llamado soma del que parten diversas prolongaciones.
Ortogonal	Propiedad que poseen dos funciones *f* y *g* de un cierto espacio cuando su producto escalar es nulo.

Parámetros	Serie de características que pueden describir y permitir evaluar el efecto de algún dispositivo.
Patrón	Conjunto de características de una imagen utilizadas para identificar a otra.
PCA	Viene de la abreviatura de *principal component analysis.*
PE	Viene de la abreviatura de *processor element.*
Píxel	Es el menor de los elementos de una imagen al que se puede aplicar individualmente un color o una intensidad o que se puede diferenciar de los otros mediante un determinado procedimiento.
Procesamiento digital	Es el conjunto de operaciones que se aplican a las señales digitales con el objetivo de mejorar la calidad o facilitar la búsqueda de información.
RGB	Viene de la abreviatura de *Red, Green, Blue.*
RNA	Viene de la abreviatura de red neuronal artificial.
Ruido de imagen	Es cualquier evento que altera el valor real de una imagen digital.

Segmentación	Se trata de una operación en donde se divide una imagen en diferentes regiones, o dicho de otra forma, detectar automáticamente los bordes entre los elementos o regiones.
Sensores	Dispositivos que detectan cambios en su entorno, convirtiéndolos en algún tipo de señal.
Sinapsis	Punto de contacto entre el axón de una neurona y la dendrita de otra, el cual permite la transmisión de señales por medio de neurotransmisores.
Soma	Porción central de una neurona que contiene el núcleo y tiene unida una o más ramificaciones llamadas dendritas y axones.
Umbral de iluminación	Valor numérico que indica el nivel de iluminación en una imagen digital.
Vector	Parte de una matriz, que contiene datos ya sea una en forma de columna o fila.

RESUMEN

En el presente trabajo de graduación se explica cómo procesar digitalmente una imagen utilizando una herramienta tan versátil como Matlab; explicando también cómo elaborar un algoritmo para el reconocimiento de imágenes utilizando redes neuronales artificiales.

Inicialmente se establecen los fundamentos teóricos que respaldan el procesamiento digital de imagen, describiendo algunos formatos que utiliza Matlab para poder digitalizar imágenes; también se explican los pasos que se deben seguir para poder procesar una imagen digital en Matlab. El segundo capítulo se centra en la descripción y comparación entre una neurona biológica y una artificial, incluyendo una descripción de los tipos de redes neurales artificiales que podrían utilizarse y su comparación entre ellos.

El tercer capítulo describe algunos métodos estadísticos aplicados al reconocimiento de imágenes, ya que son herramientas esenciales para reducir el área de análisis de la imagen y poder crear bases de datos que ocupen el menor espacio en memoria del computador; también se describe las distintas formas utilizadas para entrenar una red neural. Finalmente, son explicados los recursos necesarios para la elaboración del sistema. Además, se incluye la explicación de las características o parámetros relevantes en el procesamiento digital de imágenes, y el por qué de la selección de los procedimientos utilizados en los algoritmos que constituyen el programa.

Terminando con un esquema gráfico general del programa y una presentación del funcionamiento del entorno gráfico del mismo.

OBJETIVOS

General

Diseñar mediante Matlab un sistema para el reconocimiento de imágenes entrenando una red neuronal artificial.

Específicos

1. Dar a conocer el manejo adecuado o apropiado del programa Matlab y las diferentes herramientas que posee para la digitalización y procesamiento de imágenes.

2. Dar a conocer el funcionamiento de las redes neuronales artificiales.

3. Aplicar métodos estadísticos en el reconocimiento de imagen.

4. Crear un prototipo flexible y eficaz para la identificación de imágenes.

INTRODUCCIÓN

Es indudable la importancia que tienen hoy en día los sistemas autónomos. De la misma forma, son bien conocidos los problemas de estos sistemas en el campo de la visión artificial, y más concretamente en el reconocimiento de formas y objetos.

El desarrollo de la tecnología y el aumento de la competencia en el sector industrial, han llevado a las empresas a reestructurarse y perfeccionarse optimizando sus procesos por medio de la automatización basada en el computador. La visión artificial es parte de la informática avanzada que intenta aproximar esta capacidad humana como lo es la visión.

Este proyecto se basa en la aplicación de las redes neuronales, una disciplina muy potente, ampliamente probada, que proporciona capacidades avanzadas de discriminación y aprendizaje de nuevos modelos, juntamente con el procesamiento de imágenes digitales se logra obtener un resultado satisfactorio en el campo de identificación de objetos y formas en una imagen digital.

1. PROCESAMIENTO DIGITAL DE IMÁGENES

El procesamiento digital de imágenes es el conjunto de técnicas que se aplican a las imágenes digitales con el objetivo de mejorar la calidad o facilitar la búsqueda de información, procesando las imágenes del mundo real de manera digital obtenidas mediante el proceso de muestreo y cuantificación de las señales de vídeo adquirida a través de sensores especializados (bien de cámaras u otro tipo de adquisición).

1.1. Introducción a Matlab

El nombre de Matlab proviene de la contracción de los términos *MATrix LABoratory*. Es un entorno de computación y desarrollo de aplicaciones que integra análisis numérico, cálculo matricial, proceso de señal y visualización gráfica en un entorno completo. En la actualidad goza de un alto nivel de implantación en centros de educación, así como en departamentos de investigación y desarrollo de muchas compañías industriales nacionales e internacionales.

Matlab fue originalmente desarrollado en lenguaje Fortran y al pasar de los años fue complementado y reimplementado en lenguaje C. Actualmente la licencia de Matlab es propiedad de *MathWorks Inc*. Está disponible para un amplio número de plataformas y opera bajo sistemas operativos como *Unix, Macintosh y Windows*.

Matlab dispone de varios programas de apoyo especializado llamadas herramientas (*toolbox*), para el presente trabajo se hizo uso de la herramienta de procesamiento de imagen.

1.1.1. Herramienta sobre el procesamiento de imágenes

Esta herramienta proporciona a Matlab, un conjunto de funciones que amplían las capacidades del producto para realizar desarrollo de aplicaciones y de nuevos algoritmos en el campo del procesamiento y análisis de imágenes. Algunas de las funciones más importantes son:

- Diseño de filtros y recuperación de imágenes
- Mejora de imágenes
- Operaciones morfológicas
- Definición de mapas de colores y modificación gráfica
- Operaciones geométricas
- Transformación de imágenes

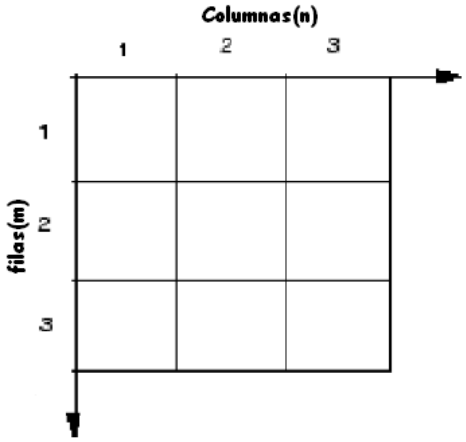
1.2. Imagen digital

Una imagen digital está compuesta de píxeles los cuales pueden definirse de alguna manera como pequeños puntos en la pantalla o imagen. Cada píxel es capaz de proporcionar información visual acerca de una pequeña región en particular de la imagen. A partir de esto, se puede considerar a una imagen digital, como un arreglo de datos de cómo se encuentra coloreado cada píxel.

Las imágenes digitales en Matlab se representan por medio de un arreglo de números reales o complejos. Dicho arreglo puede ser bidimensional o tridimensional, dependiendo del tipo de imagen de que se trate.

En general se puede decir que una imagen de m por n siendo m los píxeles de longitud (filas) y n de amplitud (columnas), se tendría una imagen almacenada de nxm píxeles de tamaño. Es decir que tener una imagen de 200x1 000 píxeles, significa que la imagen contiene información de 200 000 píxeles.

Figura 1. **Sistema de coordenadas en la ubicación de cada píxel**



Fuente: elaboración propia.

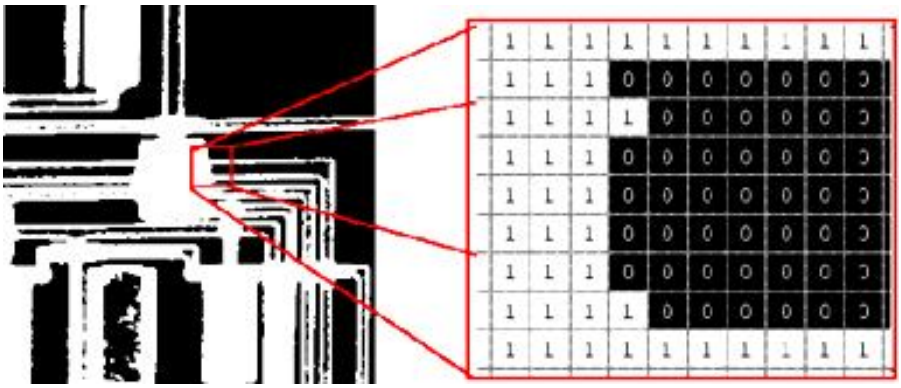
1.3. Tipos de imágenes utilizados en Matlab

En Matlab se trabaja cuatro tipos básicos de imágenes la cuales son, imágenes binarias, imágenes en escala de grises, imágenes indexadas, e imágenes RGB. Estos tipos determinan la forma de interpretación de los datos de la matriz de elementos como la intensidad de los valores de cada píxel entre otros.

1.3.1. Imagen binaria

En una imagen binaria, cada píxel toma uno de solo dos valores discretos 1 o 0. Una imagen binaria es almacenada también como un arreglo lógico. En la figura siguiente se muestra una imagen binaria con una visualización de algunos valores de los píxeles.

Figura 2. **Valores de píxel en una imagen binaria**



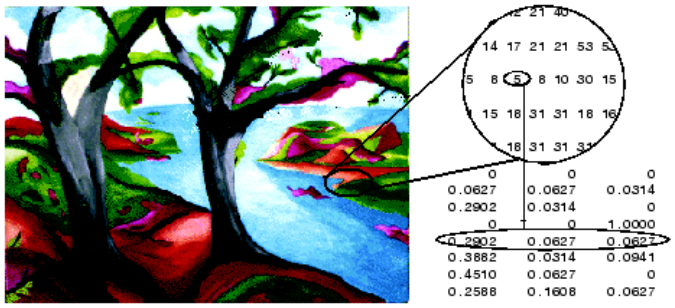
Fuente: Matlab Guía de usuario para el procesamiento de imágenes, p.58.

1.3.2. Imagen indexada

Es una forma práctica de representar las imágenes a color. Una imagen indexada guarda una imagen como dos matrices. La primera matriz tiene el mismo tamaño que la imagen y un número para cada píxel. La segunda matriz se conoce como mapa de color y su tamaño puede diferir del de la imagen.

Los números en la primera matriz indican el número a usar en la matriz mapa de color. En la figura siguiente se da un ejemplo de cómo es una imagen indexada.

Figura 3. **Imagen indexada**

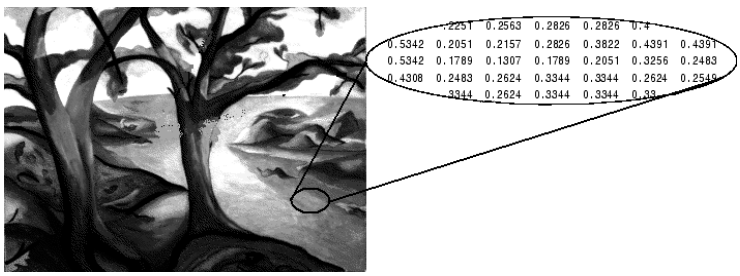


Fuente: Matlab Guía de usuario para el procesamiento de imágenes, p.60.

1.3.3. Imagen en escala de grises

Representa una imagen como una matriz, donde cada elemento tiene un valor que corresponde a cuan brillante u oscuro debería ser coloreado el píxel en la posición correspondiente. Existen dos formas de representar el número que a su vez representa la brillantez del píxel. La clase doublé, asigna un número flotante (número con decimales), entre 0 y 1 a cada píxel. El valor 0 corresponde al negro y el 1 al blanco. La otra clase es la uint8 la cual asigna un entero entre 0 y 255. El valor 0 corresponde al negro y 255 al blanco.

Figura 4. **Imagen a escala de grises de clase doublé**



Fuente: Matlab Guía de usuario para el procesamiento de imágenes, p.61.

1.3.4. Imagen RGB

Este es otro formato para imágenes a color. Se presenta en la forma de un arreglo de mxnx3 de clases uint8, uint16, single, o doublé cuyos valores de píxel especifican valores de intensidad. Este representa una imagen con tres matrices de tamaños que concuerdan con el de la imagen.

Cada matriz corresponde a uno de los colores básicos rojo, verde y azul, estableciendo cuanto de cada uno de estos colores debe ser usado en un determinado píxel .

Figura 5. **Imagen RGB de clase doublé**

Fuente: Matlab Guía de usuario para el procesamiento de imágenes, p.63.

1.4. Procesamiento

Cuando se adquiere una imagen mediante cualquier sistema de captura, por lo general ésta no es directamente utilizable por el sistema. La aparición de variaciones en intensidad debidas al ruido, por deficiencias en la iluminación, o la obtención de imágenes de bajo contraste, hace necesario un procesamiento de la imagen con el objetivo fundamental de corregir estos problemas, además de aplicar aquellas transformaciones a la imagen que acentúen las características que se deseen extraer de las mismas, de manera que se facilite las operaciones de las etapas posteriores.

1.4.1. Transformación del histograma

Las transformaciones del histograma pueden facilitar la segmentación de objetos de la imagen, aunque habitualmente sólo son utilizadas como técnicas de realce.

Las técnicas de realce pretenden aumentar el contraste de las imágenes. No en el sentido estricto de aumentar la calidad radiométrica, sino de mejorar algunas de sus características visuales para las siguientes etapas del análisis automático de las imágenes. Las causas de aplicar estos algoritmos se deben bien a una falta de iluminación uniforme en la escena o bien al deseo de aumentar el contraste entre los objetos presentes en la imagen, también se aplican cuando se pretende utilizar técnicas de segmentación. El objetivo de este procesado es facilitar las tareas de partición de la imagen. Con este fin se aumenta el contraste entre los objetos de la imagen.

1.4.2. Filtrado espacial

El filtrado espacial es la operación que se aplica a una imagen para resaltar o atenuar detalles espaciales con el fin de mejorar la interpretación visual o facilitar un procesamiento posterior, y constituye una de las técnicas comprendidas dentro del realce de imágenes. Ejemplos comunes incluyen aplicar filtros para mejorar los detalles de bordes en imágenes, o para reducir o eliminar patrones de ruido.

El filtrado espacial es una operación local en procesamiento de imagen, en el sentido de que modifica el valor de cada píxel de acuerdo con los valores de los píxeles que lo rodean, se trata entonces de transformar los niveles de gris originales de tal forma que se parezcan o diferencien más de los correspondientes a los píxeles cercanos.

Los filtros espaciales se pueden dividir en tres categorías, filtros pasa bajos, filtros pasa altos, filtros detectores de bordes, filtros de media. En este trabajo solo analizaremos algunos de estos filtros.

1.4.2.1. Filtro pasa bajos

Generalmente son utilizados para atenuar los detalles irrelevantes en una imagen, otra de las utilidades del filtro pasa bajos, aparte de la más obvia que es la atenuación del ruido, es el suavizado de los falsos contornos producidos por una cuantización con un número insuficiente de niveles de gris. El procedimiento básico del filtro pasa bajos es reemplazar el valor de cada píxel en una imagen por el promedio de los niveles de gris contenidos alrededor de la máscara utilizada, una máscara es semejante a un elemento estructural la cual se detallara más adelante.

1.4.2.2. Filtro de mediana

Los filtros estadísticos, son filtros espaciales no lineales cuya respuesta está basada en ordenar los píxeles abarcados por una máscara y luego reemplazar el valor del píxel central con el valor determinado por el resultado del ordenamiento. El más conocido de estos filtros es el filtro de mediana, el cual reemplaza el valor del píxel central por la mediana de los niveles de gris del vecindario de ese píxel (el valor original del píxel es incluido en el cálculo de la mediana). Los filtros de mediana son muy usados debido a que, para ciertos tipos de ruidos aleatorios, proveen una excelente reducción de ruido que los filtros lineales de suavizado del mismo tamaño.

Los filtros de mediana son particularmente efectivos cuando el ruido es del tipo impulso (también llamado ruido sal y pimienta) debido a que aparece como puntos negros o blancos sobrepuestos en la imagen.

La mediana, ξ, de un conjunto de valores es aquella en la que la mitad de los valores en el conjunto son menores o iguales que ξ, y la otra mitad es mayor o igual a ξ. Por ejemplo, si en una imagen tomamos un conjunto de píxeles de 3x3 con valores {1, 9, 5, 0, 8, 7, 1, 2, 4} la mediana para este caso será el valor 4, ya que la mitad de este conjunto es menor (o igual) y la otra mitad es mayor (o igual) a éste, ordenándolos se tendría {0, 1, 1, 2, 4, 5, 7, 8, 9}.

El procedimiento general para realizar el filtro de mediana en cualquier punto consiste en ordenar los valores de dicho píxel y los de su vecindario, determinar la mediana, y asignar éste último valor al píxel en cuestión.

1.4.3. Segmentación

Cuando ya se dispone de la imagen capturada y filtrada, es necesario aislar o separar los objetos de interés de la escena. Se pretende por tanto dividir una imagen en diferentes regiones, o dicho de otra forma, detectar automáticamente los bordes entre los elementos o regiones

Las técnicas básicas de segmentación se pueden dividir en tres grupos, aplicación de umbrales de niveles de gris, agrupación por rasgos comunes, extracción de bordes.

1.4.3.1. Aplicación de umbralización

La umbralización es una técnica de segmentación ampliamente utilizada en las aplicaciones industriales. Se emplea cuando hay una clara diferencia entre los objetos a extraer respecto del fondo de la escena. Los principios que rigen son la similitud entre los píxeles pertenecientes a un objeto y sus diferencias respecto al resto. Por tanto, la escena debe caracterizarse por un fondo uniforme y por objetos parecidos.

La mayoría de las técnicas de umbralización se basan en estadísticas sobre el histograma unidimensional de una imagen.

Al aplicar un umbral, la imagen de niveles de grises quedará binarizada etiquetando con 1 los píxeles correspondientes al objeto y con 0 aquellos que son del fondo. Por ejemplo, si los objetos son claros respecto del fondo, se aplicará.

Ecuación 1:

$$g(x,y)=\begin{cases}1 & f(x,y)>T\\ 0 & f(x,y)\leq T\end{cases}$$

En donde f(x,y) es la función que retorna el nivel de gris del píxel en la posición (x,y), g(x,y) será la imagen binarizada y T es el umbral. En el caso de que los objetos sean oscuros respecto del fondo, la asignación sería a la inversa.

Ecuación 2:

$$g(x,y)=\begin{cases}1 & f(x,y)<T\\ 0 & f(x,y)\geq T\end{cases}$$

El umbral puede depender de f(x,y), de alguna propiedad local del píxel, p(x,y) y hasta de su propia posición.

Ecuación 3:

$$T=T(f(x,y),p(x,y),x,y)$$

Si el umbral sólo depende de f(x,y) se dice que es un umbral global, en el caso de que además dependa de p(x,y), por ejemplo, el valor medio del entorno de vecindad, el umbral es denominado local y si depende también de la posición del píxel (x,y), se denominará dinámico. El problema es encontrar el umbral, operación nada sencilla ya que las imágenes están contaminadas con el ruido. Para acotar el estudio, sólo se centrará en las técnicas globales.

1.4.3.2. Extracción de bordes

Se entiende como borde aquella región donde aparece una fuerte variación del nivel de intensidad en los píxeles adyacentes. Su causa principal es originada por la intersección de varios objetos, con diferentes niveles de reflactancia, que al ser proyectados sobre la cámara generan discontinuidades de intensidad en los píxeles correspondidos.

Sin embargo, estas discontinuidades también aparecen de forma no deseada por la presencia del ruido, por el efecto de sombras sobre los propios objetos o por una iluminación no uniforme dentro de la escena.

El fundamento para la detección de los bordes está en la aplicación del operador derivada. Si se construye una imagen sintética con franjas de alto contraste y se adquiere una fila de la imagen, se observará una fuerte variación de la intensidad en los bordes de las franjas. Al aplicar la operación derivada, se observa que ésta toma un valor de máximo o mínimo justamente cuando en la transición se pasa de cóncavo a convexo o viceversa, esto se da en el punto de inflexión del borde.

Si en vez de emplear la primera derivada se realiza con la segunda, el punto de inflexión de la primera derivada coincidirá con un paso por cero. Ambos razonamiento son empleados para la detección de los bordes, estas dan lugar a varias técnicas las cuales son, técnica basada en el operador gradiente, técnica basada en el operador laplaciano, técnica basado en el operador canny entre otros.

Figura 6. **Detección de bordes utilizando operador canny**

Fuente: Matlab Guía de usuario para el procesamiento de imágenes, p.212.

1.4.4. Operaciones morfológicas

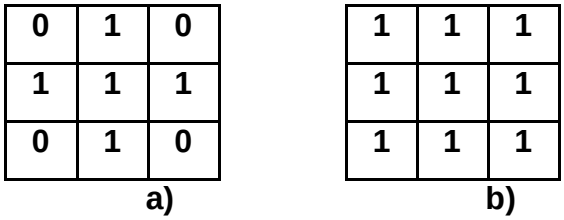
La morfología matemática es una herramienta muy utilizada en el procesamiento de imágenes. Las operaciones morfológicas pueden simplificar los datos de una imagen, preservar las características esenciales y eliminar aspectos irrelevantes. Teniendo en cuenta que la identificación, descomposición de objetos, la extracción de rasgos, la localización de defectos e incluso los defectos en líneas de ensamblaje están sumamente relacionados con las formas, es obvio el papel de la morfología matemática.

1.4.4.1. Elemento estructural

Dados los conjuntos A y B estos pueden ser considerados como una imagen u objeto, supongamos que A es la imagen y B es un elemento estructural. Para muchas operaciones la distinción de objetos depende de la convención utilizada de conectividad, que es la forma en la que se considera si dos píxel tienen relación como para considerar que forman parte del mismo objeto, por donde los píxeles del conjunto A están relacionados por el conjunto B.

Los elementos estructurales más comunes son los conjuntos que están 4 conectados, N4, y 8 conectados, N8, ilustrados en las siguientes figuras.

Figura 7. **Elementos estructurales (estándar) a) N4, b) N8**



Fuente: elaboración propia.

1.4.4.2. Dilatación

Sean A y B conjuntos en z^2. La dilatación de A por B, expresada por $A \oplus B$, se define como.

Ecuación 4:

$$A \oplus B = \{Z \mid (B)_Z \cap A \neq \emptyset\}$$

Esta ecuación consiste en obtener la reflexión de B sobre su origen y trasladar esta reflexión por z. La dilatación de A por B es entonces el conjunto de todos los desplazamientos, z, tal que la reflexión de B y A se solapan por al menos un elemento. Teniendo en cuenta lo anterior, la dilatación de A por B también se puede expresar como.

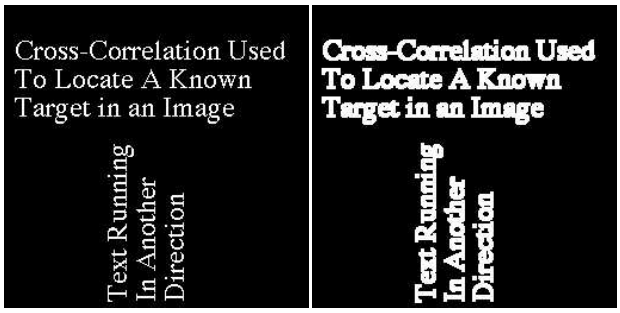
Ecuación 5:

$$A \oplus B = \{Z \mid ((B)_Z \cap A) \subseteq A\}$$

En general, la dilatación aumenta el tamaño de un objeto. La cantidad y la forma en que aumenta el tamaño dependen de la elección del elemento estructural.

Una de las aplicaciones más simples de la dilatación es la unión de píxeles relacionados, en la figura siguiente se muestra un ejemplo de una imagen dilatada con conectividad 8.

Figura 8. **Dilatación de imagen a) imagen original, b) dilatación de (a)**



a **b**

Fuente: Matlab Guía de usuario para el procesamiento de imágenes, p.168.

1.4.4.3. Erosión

Sean A y B conjuntos en z^2. La erosión de A por B, que se expresa como $A \Theta B$ y se define como.

Ecuación 6:

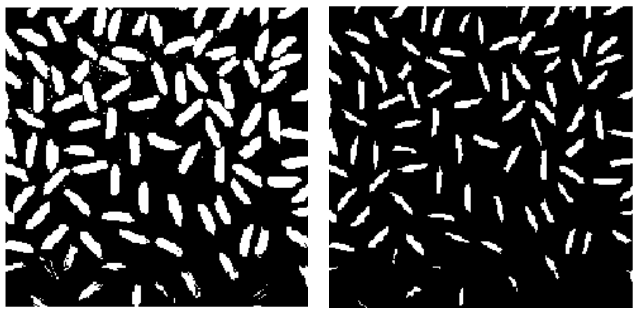
$$A \ominus B = \{Z \mid (B)_Z \subseteq A\}$$

Esta ecuación indica que la erosión de A por B es el conjunto de todos los puntos z tales que B, trasladado por z, está contenido en A.

Generalmente, la erosión disminuye el tamaño de los objetos. Lo contrario de la dilatación, la cantidad y la forma en que se produce esta disminución depende del elemento estructural elegido.

Uno de los usos más simples de la erosión es para la eliminación de detalles irrelevantes (en términos de tamaño) de una imagen binaria.

Figura 9. **Erosión de imagen a) imagen original, b) erosión de (a)**



a **b**

Fuente: Matlab Guía de usuario para el procesamiento de imágenes, p.172.

1.4.4.4 Apertura y clausura

Como vimos, la dilatación y la erosión están muy relacionadas con la forma, la primera operación expande la imagen mientras que la segunda la contrae. La dilatación y la erosión usualmente se usan de a pares, bien la dilatación seguida de la erosión o viceversa. En cualquier caso, el resultado de esta aplicación sucesiva de erosiones y dilataciones es una eliminación de detalles menores que no distorsiona la forma global del objeto.

La apertura de un conjunto A por el elemento estructural B, se define como.

Ecuación 7:

$$A \circ B = (A \ominus B) \oplus B$$

Es decir, la apertura de A por B es la erosión de A por B seguida por la dilatación del resultado por B.

De forma similar, la clausura de un conjunto A por el elemento estructural B, se define como.

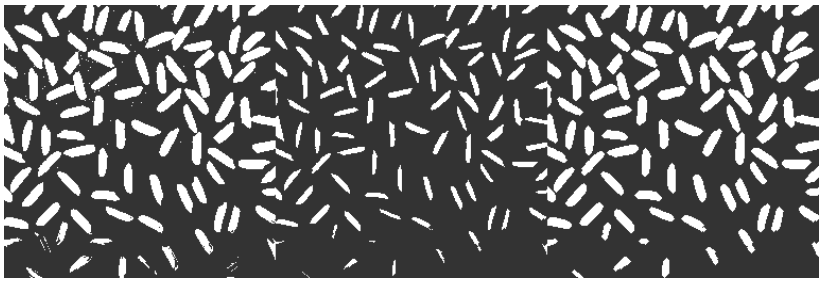
Ecuación 8:

$$A \blacksquare B = (A \oplus B) \ominus B$$

O sea, es la dilatación de A por B seguida por la erosión del resultado por B.

La apertura generalmente suaviza los contornos de un objeto y elimina protuberancias finas. La clausura también suaviza los contornos pero, contrariamente a la apertura, generalmente fusiona las hendiduras finas y largas presentes en los objetos, elimina agujeros pequeños y rellena brechas en el contorno .

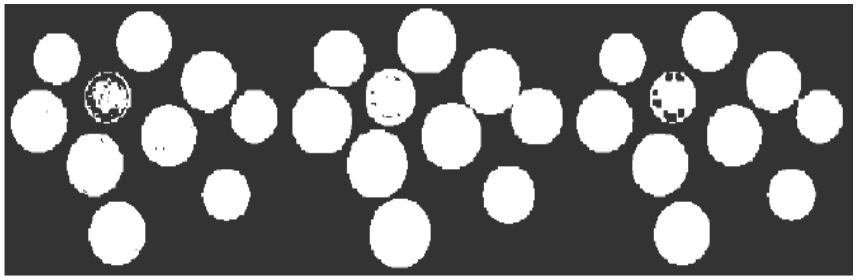
Figura 10. **Erosión y apertura a) imagen original, b) erosión de (a), c) apertura de (b)**



a **b** **c**

Fuente: Matlab Guía de usuario para el procesamiento de imágenes, p.179.

Figura 11. **Dilatación y clausura a) imagen original, b) dilatación de (a), c) cierre de (b)**



a **b** **c**

Fuente: Matlab Guía de usuario para el procesamiento de imágenes, p.180.

2. REDES NEURONALES ARTIFICIALES (RNA)

2.1. Introducción a la computación neuronal

Los sistemas de cómputo de hoy en día, son exitosos en la resolución de problemas matemáticos o científicos, pero definitivamente tienen una gran incapacidad para interpretar el mundo como tal, más específicamente como el cerebro humano lo hace.

Las RNA han surgido como un intento de desarrollar sistemas que emulen estas características del cerebro. De esta forma se define a las RNA como modelos matemáticos o computacionales inspirados en sistemas biológicos, adaptados y simulados en computadoras convencionales.

2.1.1. Historia de las redes neuronales artificiales

1936 - Alan Turing. Fue el primero en estudiar el cerebro como una forma de ver el mundo de la computación. Sin embargo, los primeros teóricos que concibieron los fundamentos de la computación neuronal fueron Warren McCulloch, un neurofisiólogo, y Walter Pitts, un matemático, quienes, en 1943, propusieron el clásico modelo de neurona en el que se basan las redes neuronales actuales. Seis años después, en 1949, Donald Hebb presentaba su conocida regla de aprendizaje. Aun hoy, este es el fundamento de la mayoría de las funciones de aprendizaje que pueden hallarse en una red neuronal.

Su idea fue que el aprendizaje ocurría cuando ciertos cambios en una neurona eran activados. También intentó encontrar semejanzas entre el aprendizaje y la actividad nerviosa.

1950 - Karl Lashley. En sus series de ensayos, encontró que la información no era almacenada en forma centralizada en el cerebro sino que era distribuida encima de él.

1956 - Congreso de Dartmouth. Este Congreso frecuentemente se menciona para indicar el nacimiento de la inteligencia artificial.

1957 - Frank Rosenblatt. Comenzó el desarrollo del perceptrón. Esta es la red neuronal más antigua, utilizándose hoy en día para aplicación como identificador de patrones. Este modelo era capaz de generalizar, es decir, después de haber aprendido una serie de patrones podía reconocer otros similares, aunque no se le hubiesen presentado en el entrenamiento. Sin embargo, tenía una serie de limitaciones, por ejemplo, su incapacidad para resolver el problema de la función or exclusiva y, en general, era incapaz de clasificar clases no separables linealmente.

1959 - Frank Rosenblatt, confirmó que, bajo ciertas condiciones, el aprendizaje del perceptrón convergía hacia un estado finito (teorema de convergencia del perceptrón).

1960 - Bernard Widroff y Marcian Hoff. Desarrollaron el modelo adaline. Esta fue la primera red neuronal aplicada a un problema real (filtros adaptativos para eliminar ecos en las líneas telefónicas) que se ha utilizado comercialmente durante varias décadas.

1969 - Marvin Minsky y Seymour Papert. En este año casi se produjo la muerte abrupta de las redes neuronales; ya que Minsky y Papert probaron (matemáticamente) que el perceptrón no era capaz de resolver problemas relativamente fáciles, tales como el aprendizaje de una función no lineal. Esto demostró que el perceptrón era muy débil, dado que las funciones no-lineales son extensamente empleadas en computación y en los problemas del mundo real.

1974 - Paul Werbos. Desarrolló la idea básica del algoritmo de aprendizaje de propagación hacia atrás (*backpropagation*), cuyo significado quedó definitivamente aclarado en 1985.

1977 - Stephen Grossberg, Teoría de Resonancia Adaptada (TRA). La Teoría de Resonancia Adaptada es una arquitectura de red que se diferencia de todas las demás previamente inventadas. La misma simula otras habilidades del cerebro.

1985 - John Hopfield. Provocó el renacimiento de las redes neuronales con su libro, Computación neuronal de decisiones en problemas de optimización.

1986 - David Rumelhart y Hinton. Redescubrieron el algoritmo de aprendizaje de propagación hacia atrás (*backpropagation*).

A partir de 1986, el panorama fue alentador con respecto a las investigaciones y el desarrollo de las redes neuronales. En la actualidad, son numerosos los trabajos que se realizan y publican cada año, las aplicaciones nuevas que surgen y las empresas que lanzan al mercado productos nuevos, tanto *hardware* como *software* (sobre todo para simulación) .

2.1.2. Característica de la red neuronal artificial

Las redes neuronales artificiales están inspiradas en las redes neuronales biológicas del cerebro humano.

Están constituidas por elementos que se comportan de forma similar a la neurona biológica en sus funciones más comunes. Estos elementos están organizados de una forma parecida a la que presenta el cerebro humano.

Las RNA al margen de parecerse al cerebro presentan una serie de características propias del cerebro. Por ejemplo aprenden de la experiencia, generalizan de ejemplos previos a ejemplos nuevos y abstraen las características principales de una serie de datos.

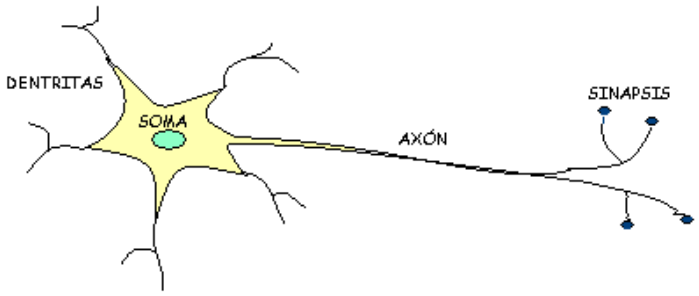
2.1.3. Estructura básica de una red neuronal

La neurona, es la unidad anatómica fundamental del sistema nervioso. La estructura de una neurona se observa en la figura 12, su fisiología se explica de acuerdo a los estudios realizados a lo largo de un valioso desarrollo, que viene desde muchos años atrás.

2.1.3.1 Analogía con el cerebro

La neurona es la unidad fundamental del sistema nervioso y en particular del cerebro. Cada neurona es una simple unidad procesadora que recibe y combina señales desde y hacia otras neuronas, si la combinación de entradas es suficientemente fuerte la salida de la neurona se activa. La figura siguiente muestra las partes que constituyen una neurona biológica.

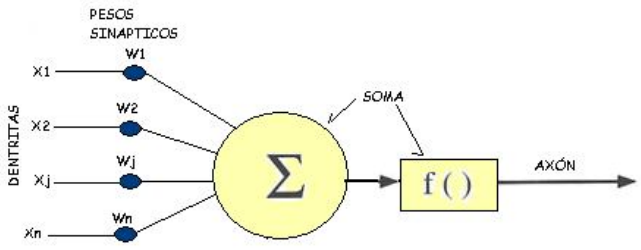
Figura 12. **Componentes de una neurona biológica**



Fuente: Tutorial de redes neuronales. http://ohm.utp.edu.co/neuronales.htm. Consulta 12 de diciembre de 2010.

La unidad análoga a la neurona biológica es el elemento procesador (PE). Un elemento procesador tiene varias entradas y las combina, normalmente con una suma básica. La suma de las entradas es modificada por una función de transferencia y el valor de la salida de esta función de transferencia se pasa directamente a la salida del elemento procesador, la salida del PE se puede conectar a las entradas de otras neuronas artificiales mediante conexiones ponderadas correspondientes a la eficacia de la sinapsis de las conexiones neuronales.

Figura 13. **Diagrama de una neurona artificial**

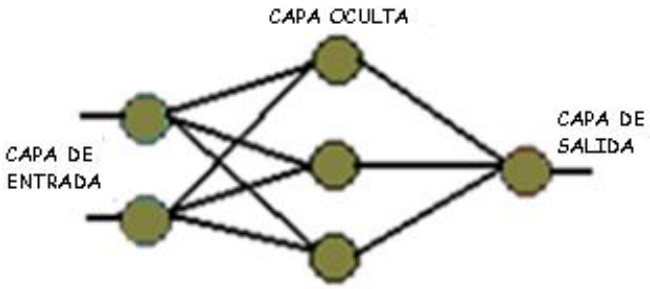


Fuente: elaboración propia.

Una red neuronal consiste en un conjunto de unidades elementales PE conectadas de una forma concreta. El interés de las RNA no reside solamente en el modelo del elemento PE, sino en las formas en que se conectan estos elementos procesadores. Generalmente los elementos PE están organizados en grupos llamados niveles o capas. Una red típica consiste en una secuencia de capas con conexiones entre capas adyacentes consecutivas.

Existen dos capas con conexiones con el mundo exterior. Una capa de entrada, donde se presentan los datos a la red, y una capa de salida que mantiene la respuesta de la red a una entrada. El resto de las capas reciben el nombre de capas ocultas.

Figura 14. **Arquitectura de una red neuronal**



Fuente: elaboración propia.

2.2. Fundamento de las redes neuronales

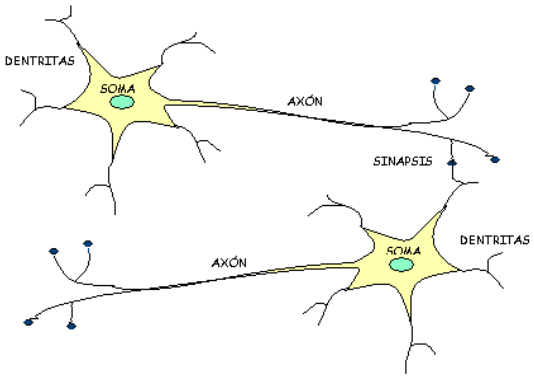
Los elementos procesadores más exitosos que se han diseñado están inspirados en los sistemas neuronales biológicos, así las redes neuronales artificiales son procesadores simples adaptativos organizados de manera ordenada que pueden procesar en paralelo señales complejas de tipo físico.

2.2.1. Neurona biológica

El cerebro consta de un gran número (aproximadamente 10^{11}) de elementos altamente interconectados (aproximadamente 10^4 conexiones por elemento), llamados neuronas, estas neuronas tienen tres componentes principales, las dendritas, el cuerpo de la célula o soma, y el axón. Las dendritas, son el árbol receptor de la red, son como fibras nerviosas que cargan de señales eléctricas el cuerpo de la célula.

El cuerpo de la célula, realiza la suma de esas señales de entrada. El axón es una fibra larga que lleva la señal desde el cuerpo de la célula hacia otras neuronas. El punto de contacto entre un axón de una célula y una dendrita de otra célula es llamado sinapsis, la longitud de la sinapsis es determinada por la complejidad del proceso químico que estabiliza la función de la red neuronal.

Figura 15. **Interconexión de dos neuronas biológicas**

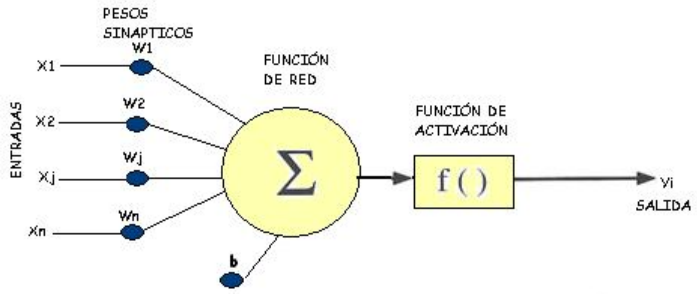


Fuente: Tutorial de redes neuronales. http://ohm.utp.edu.co/neuronales.htm. Consulta 12 de diciembre de 2010.

2.2.2. Neurona artificial

El modelo de una neurona artificial es una imitación del proceso de una neurona biológica.

Figura 16. **Neurona artificial**



Fuente: elaboración propia.

Las señales de entrada a una neurona artificial X_1, X_2,.., X_n son variables continuas en lugar de pulsos discretos, como se presentan en una neurona biológica. Cada señal de entrada pasa a través de una ganancia o peso, llamado peso sináptico o fortaleza de la conexión cuya función es análoga a la de la función sináptica de la neurona biológica. Los pesos pueden ser positivos (excitatorios), o negativos (inhibitorios), el nodo sumatorio (función de red) acumula todas las señales de entradas multiplicadas por los pesos. Al resultado se le suma un valor constante denominado polarización *bias* y es representado por b.

Ecuación 9:

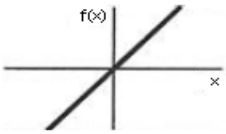
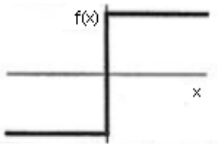
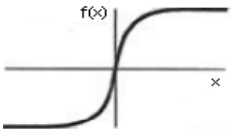
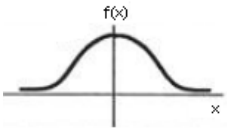
$$S=\sum_{i=1}^{n} X_i*W_i$$

Luego pasa a la salida a través de una función umbral o función de transferencia. Una vez que se ha calculado la activación del nodo, el valor de salida equivale a la ecuación siguiente.

Ecuación 10:

$$y_i = f\left[\sum_{i=1}^{n} X_i * W_i\right]$$

Tabla I. **Algunas funciones de activación**

Función	**Gráfica**
Lineal $y=kx$	
Escalón $y=sing(x)$	
Sigmoidea $y=\frac{1}{1+e^{-x}}$	
Gaussiana $y=Ae^{-Bx^2}$	

Fuente: elaboración propia.

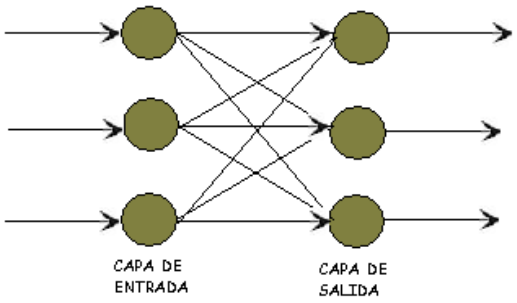
2.3. Clasificación de las RNA

Existen distintas clases de redes neuronales los cuales se diferencian por la forma en que los PE están organizados, por las formas en las que se entrenan, por la manera en la que asocia la información de entrada y de salida, y por la forma en la que dicha información es representada. Entre las formas más comunes están las siguientes.

2.3.1. Red neuronal monocapa

Se corresponde con la red neuronal más sencilla ya que se tiene una capa de neuronas que proyectan las entradas a una capa de neuronas de salida donde se realizan los diferentes cálculos.

Figura 17. **Red neuronal monocapa**

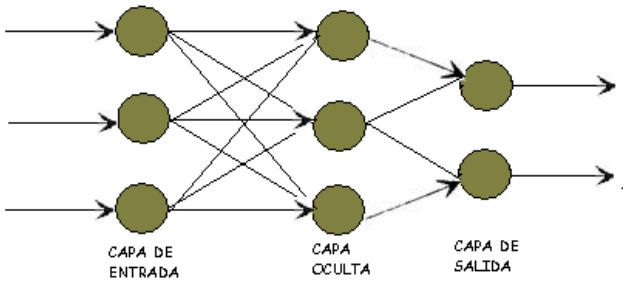


Fuente: elaboración propia.

2.3.2. Red neuronal multicapa

Es una generalización de la anterior, existiendo un conjunto de capas intermedias entre la capa de entrada y la de salida denominadas capas ocultas. Este tipo de red puede estar total o parcialmente conectada.

Figura 18. **Red neuronal multicapa**

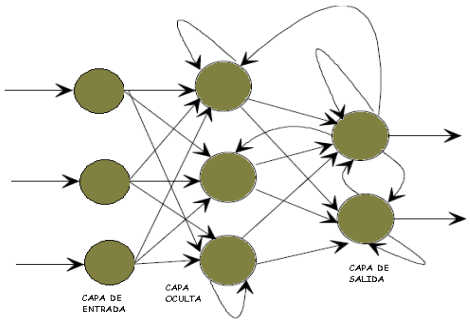


Fuente: elaboración propia.

2.3.3. Red neuronal recurrente

Esta clase de red se diferencia de las anteriores en la existencia de lazos de realimentación en la red. Estos lazos pueden ser entre neuronas de diferentes capas, neuronas de la misma capa o, entre una misma neurona. Esta estructura la hace especialmente adecuada para estudiar la dinámica de los sistemas no lineales.

Figura 19. **Red neuronal recurrente**



Fuente: elaboración propia.

2.4. Tipos de RNA

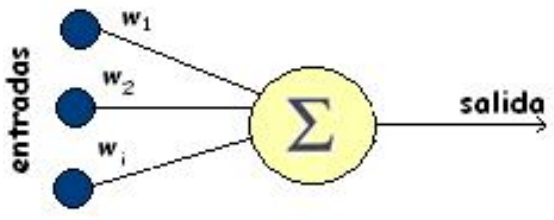
Existe una serie de modelos sobre redes neuronales que se diferencia entre sí por su forma de conexión, aprendizaje, etc. En este trabajo se explicaran las más utilizadas.

2.4.1. Perceptrón

La arquitectura del perceptrón, llamada mapeo de patrones, aprende a clasificar modelos mediante un aprendizaje supervisado. Los modelos que clasifica suelen ser generalmente vectores con valores binarios (0,1) y las categorías de la clasificación se expresan mediante vectores binarios, el perceptrón presenta dos capas de unidades procesadoras y sólo una de ellas presenta la capacidad de adaptar o modificar los pesos de las conexiones, la arquitectura del perceptrón admite capas adicionales pero éstas no disponen la capacidad de modificar sus propias conexiones.

La figura siguiente muestra la unidad procesadora básica del perceptrón. Las entradas llamadas x_i llegan por la parte izquierda, y cada conexión con la neurona tiene asignada un peso de valor w_i.

Figura 20. **Unidad procesadora del perceptrón**



Fuente: elaboración propia.

La unidad procesadora del perceptrón realiza la suma ponderada de las entradas según la ecuación 9.

Un aspecto común en muchas de las RNA es la entrada especial llamada *bias* siempre presenta un valor fijo, +1 y funciona como una masa en un circuito eléctrico donde no varía de valor (se puede utilizar como un valor constante de referencia).

El perceptrón comprueba si la suma de las entradas ponderadas es mayor o menor que un cierto valor umbral y genera la salida y_i según la ecuación siguiente.

Ecuación 11:

$$y_i = \begin{cases} 1 & \text{si} \quad S>0 \\ 0 & \text{si} \quad S\leq 0 \end{cases}$$

La salida y_i es transmitida a lo largo de la línea de salida y constituye uno de los componentes del vector de salida de la red.

2.4.2. Perceptrón multicapa

El perceptrón multicapa es una red neuronal formada por múltiples capas, esto le permite resolver problemas que no son linealmente separables, lo cual es la principal limitación del perceptrón**.**

El perceptrón multicapa puede ser totalmente o localmente conectado. En el primer caso cada salida de una neurona de la capa i es entrada de todas las neuronas de la capa i+1, mientras que en el segundo cada neurona de la capa i es entrada de una serie de neuronas de la capa i+1.

Las capas pueden clasificarse en tres tipos. Capa de entrada**,** constituida por aquellas neuronas que introducen los patrones de entrada en la red, en estas neuronas no se produce procesamiento. Capas ocultas**,** formada por aquellas neuronas cuyas entradas provienen de capas anteriores y cuyas salidas pasan a neuronas de capas posteriores. Capa de salida**,** neuronas cuyos valores de salida se corresponden con las salidas de toda la red.

2.4.3. Adaline-Madaline

La arquitectura de adaline fue creada por Bernard Widrow en 1959. Utiliza un dispositivo lógico que realiza una suma lineal de las entradas y genera una función umbral para el resultado de dicha suma, la arquitectura madaline creada también por Widrow presenta una configuración constituida por dos o más unidades adaline.

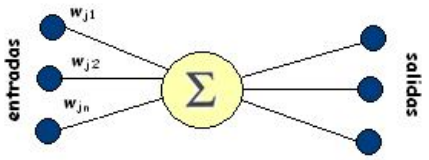
A lo largo del tiempo se han estudiado diferentes variaciones de los algoritmos de aprendizaje de la adaline y madaline, entre las aplicaciones investigadas destacan entre otras, filtros adaptativos de eliminación de ruido y reconocimiento de patrones de señales. No obstante, desde los primeros experimentos con la adaline y madaline se comprobó la capacidad de clasificar patrones linealmente separables, presentando la misma limitación que el perceptrón, la carencia de un método que ajuste más de una capa de pesos.

La diferencia entre el adaline y el perceptrón es que el perceptrón solo tiene capacidad para clasificar, ya que utiliza una función umbral sobre la suma ponderada de las entradas, a diferencia del adaline, que es capaz de estimar una salida real.

2.4.5. *Backpropagation*

La unidad procesadora básica de la red *backpropagation* se representa en la figura siguiente. Las entradas se muestran a la izquierda, y a la derecha se encuentran unidades que reciben la salida de la unidad procesadora situada en el centro de la figura.

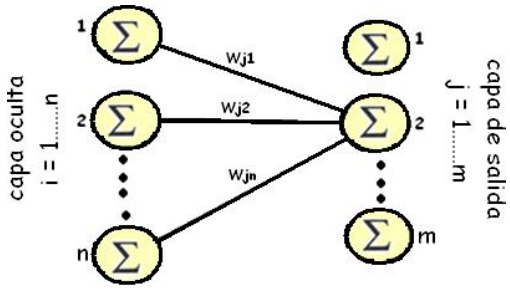
Figura 21. **Unidad procesadora de la red *backpropagation***



Fuente: elaboración propia.

La unidad procesadora se caracteriza por realizar una suma ponderada de las entradas llamadas x_j, presentar una salida y_j y tener un valor δ_j asociado que se utilizará en el proceso de ajuste de los pesos. El peso asociado a la conexión desde la unidad i a la unidad j se representa por w_{ji}, y es modificado durante el proceso de aprendizaje.

Figura 22. **Arquitectura básica de la red *backpropagation***



Fuente: elaboración propia.

Normalmente, *backpropagation* utiliza tres o más capas de unidades procesadoras. La figura 18 muestra la topología *backpropagation* típica de tres capas, la capa del lado izquierdo es la capa de entrada, y se caracteriza por ser la única capa cuyas unidades procesadoras reciben entradas desde el exterior. Sirven como puntos distribuidores, no realizan ninguna operación de cálculo.

Las unidades procesadoras de las demás capas procesan las señales como se indica en la figura 22. La capa ubicada en el centro es la capa oculta, y todas sus unidades procesadoras están interconectadas con la capa de la izquierda y con la capa de la derecha. La capa del lado derecho es la capa de salida que presenta la respuesta de la red.

Las redes *backpropagation* tienen un método de entrenamiento supervisado. A la red se le presenta una pareja de vectores, un vector de entrada emparejado con un vector de salida deseado. Por cada presentación los pesos son ajustados de forma que disminuya el error entre la salida deseada y la respuesta de la red.

El algoritmo de aprendizaje *backpropagation* se clasifica en dos fases, una fase de propagación hacia adelante y otra fase de propagación hacia atrás. Ambas fases se realizan por cada vector presentado en la sesión de entrenamiento.

La fase de propagación hacia adelante se inicia cuando se presenta un vector en la capa de entrada de la red. Cada unidad de la entrada se corresponde con un elemento del vector patrón de entrada. Las unidades de entrada toman el valor de su correspondiente elemento del vector patrón de entrada y se calcula el valor de activación o nivel de salida de la primera capa.

A continuación las demás capas realizarán la fase de propagación hacia adelante que determina el nivel de activación de las otras capas.

Como se dijo anterior, las unidades procesadoras de la capa de entrada no realizan ninguna operación de cálculo con sus entradas, ni operaciones con funciones umbrales, sólo asumen su salida como el valor del correspondiente elemento del vector de entrada.

Una vez se ha completado la fase de propagación hacia adelante se inicia la fase de corrección o fase de propagación hacia atrás. Los cálculos de las modificaciones de todos los pesos de las conexiones empiezan por la capa de salida y continua hacia atrás a través de todas las capas de la red hasta la capa de entrada.

Dentro de los tipos de ajuste de pesos se puede mencionar los, ajuste de unidades procesadoras de la capa de salida y ajuste de unidades procesadoras de las capas ocultas.

2.4.5. *Hopfield*

La *red hopfield* tiene una única capa de unidades procesadoras. Cada una de las unidades procesadoras tiene un valor o nivel de activación, también llamado estado, que es binario (la red presentada en 1982 se llama *Red Hopfield* Binaria).

Se considera que la *red hopfield* tiene un estado en cada momento, este estado se define por el vector de unos y ceros constituido por los estados de todas las unidades procesadoras.

El estado de una red con n unidades procesadoras, donde el elemento i tiene el estado u_i se representa según la ecuación siguiente.

Ecuación 12:

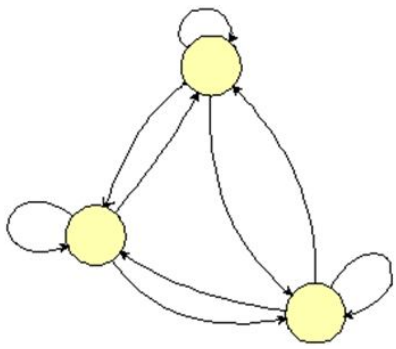
$$U=(u_1u_2,\ldots\ldots,u_n)=(++\text{--}_{\ldots\ldots}++)$$

En esta notación el signo + representa una unidad procesadora con el estado o valor binario 1 y el signo - representa una unidad procesadora con el estado o valor binario 0.

Las unidades procesadoras de la red *hopfield* están completamente interconectadas, cada unidad está conectada con todas las demás unidades. Esta topología convierte a la red *Hopfield* en una red recursiva ya que la salida de cada unidad está realimentada con las entradas de las demás unidades.

La figura siguiente muestra un diagrama de las unidades procesadoras de una red *Hopfield* y como se interconectan entre sí.

Figura 23. **Red *hopfield***



Fuente: elaboración propia.

Una característica de las redes *hopfield* es la doble conexión por cada pareja de unidades procesadoras, como se aprecia en la figura anterior. Además los pesos asignados a ambas conexiones tienen el mismo valor.

Inicialmente, la red tiene asignado un estado para cada unidad de proceso. El procedimiento de actualización se aplica a todas las unidades de una en una. Este procedimiento afecta al estado de cada unidad modificándolo o manteniéndolo constante. Este procedimiento de actualización permanece hasta que no se produzca ninguna modificación en la red.

El modo de operación de la red se puede visualizar geométricamente. Para un caso genérico de n neuronas, el número de estados posibles es 2^n y se le asocia un hipercubo de n dimensiones. Cuando se le presenta una nueva entrada, la red se mueve de un vértice a otro hasta que se estabiliza. El vértice estable está definido por los pesos de la red, las entradas actuales y el valor umbral de la función f de las neuronas. Si el vector entrada está parcialmente incompleto o es parcialmente incorrecto, la red se estabiliza en el vértice más próximo al vértice deseado .

3. RECONOCIMIENTO Y APRENDIZAJE

3.1. Reconocimiento

El campo de los métodos de reconocimiento de patrones es muy actual. Básicamente se trata de clasificar una serie de medidas a partir de la información captada por sensores (cámaras) para su posterior análisis. Dicha clasificación se realiza a partir de una serie de técnicas matemáticas y estadísticas. Evidentemente el uso de estas técnicas resulta más necesario e indicado cuando más sensores tengamos. Generalmente se trata de matrices o agrupaciones de sensores.

Estos métodos recurren a una serie de bases de datos para finalmente clasificar las medidas en uno y otro grupo. Dicha información facilitará en gran medida las decisiones a tomar por el sistema encargado de procesar la información.

Se agrupan los métodos de reconocimiento de patrones principalmente en 2 grupos, los estadísticos y los neuronales.

3.2. Métodos estadísticos

Los métodos estadísticos tratan de darle una solución al problema de la reducción de la dimensionalidad de una imagen, si es posible describir con precisión los valores de m variables por un pequeño subconjunto $p < m$ de ellas, se habrá reducido la dimensión del problema a costa de una pequeña pérdida de información.

3.2.1. Análisis de Componentes Principales (PCA)

El PCA es una técnica utilizada para reducir la dimensión de un conjunto de datos. Intuitivamente la técnica sirve para hallar las causas de la variabilidad de un conjunto de datos y ordenarlas por importancia.

Técnicamente, el PCA busca la proyección según la cual los datos queden mejor representados en términos de mínimos cuadrados. El PCA se emplea sobre todo en análisis exploratorio de datos y para construir modelos predictivos. El PCA se basa en el cálculo de la descomposición en auto valores de la matriz de covarianza**,** normalmente tras centrar los datos en la media de cada atributo.

El PCA construye una transformación lineal que escoge un nuevo sistema de coordenadas para el conjunto original de datos en el cual la varianza de mayor tamaño del conjunto de datos es capturada en el primer eje (llamado el primer componente principal), la segunda varianza más grande es el segundo eje, y así sucesivamente.

Para construir esta transformación lineal debe construirse primero la matriz de covarianza o matriz de coeficientes de correlación. Debido a la simetría de esta matriz existe una base completa de vectores propios de la misma. La transformación que lleva de las antiguas coordenadas a las coordenadas de la nueva base es precisamente la transformación lineal necesaria para reducir la dimensión de datos. Además las coordenadas en la nueva base dan la composición en factores subyacentes de los datos iníciales.

Una de las ventajas del PCA para reducir la dimensión de un grupo de datos, es que retiene aquellas características del conjunto de datos que contribuyen más a su varianza**,** manteniendo un orden de bajo nivel de los componentes principales e ignorando los de alto nivel. El objetivo es que esos componentes de bajo orden a veces contienen el más importante aspecto de esa información.

Supongamos que existe una muestra con n objetos para cada uno de los cuales se han medido m variables (aleatorias). El PCA permite encontrar un número de factores subyacentes p < m que explican aproximadamente el valor de las m variables para cada objeto. El hecho de que existan estos p factores subyacentes puede interpretarse como una reducción de la dimensión de los datos, donde antes necesitábamos m valores para caracterizar a cada objeto ahora nos bastan p valores. Cada uno de los p encontrados se llama componente principal, de ahí el nombre del método.

Existen dos formas básicas de aplicar el PCA, el método basado en la matriz de covarianzas, que se usa cuando los datos son dimensionalmente homogéneos y presentan valores medios similares y el método basado en la matriz de correlación, cuando los datos no son dimensionalmente homogéneos o el orden de magnitud de las variables aleatorias medidas no es el mismo.

3.2.1.1. Método basado en las covarianzas

El objetivo es transformar un conjunto dado de datos X de dimensión nxm a otro conjunto de datos Y de menor dimensión nxl con la menor pérdida de información útil posible utilizando para ello la matriz de covarianza.

Se parte de un conjunto de n muestras cada una de las cuales tiene m variables que las describen y el objetivo es que, cada una de esas muestras, se describa con solo l variables, donde l < m. Además, el número de componentes principales l tiene que ser inferior a la menor de las dimensiones de X.

Ecuación 12:

$$l \leq \min\{n,m\}$$

Los datos para el análisis tienen que estar centrados a media 0 o sea restándoles la media de cada columna también se tiene que dividir cada columna por su desviación estándar.

Ecuación 13:

$$X = \sum_{a=1}^{l} t_a p_a^T + E$$

Los vectores t_a son conocidos como *scores* y contienen la información de cómo las muestras están relacionadas unas con otras, además, tienen la propiedad de ser ortogonales. Los vectores p_a se llaman *loadings* e informan de la relación existente entre las variables y tienen la cualidad de ser ortonormales.

Al tomar menos componentes principales que variables y debido al error de ajuste del modelo con los datos, se produce un error que se acumula en la matriz E. El PCA se basa en la descomposición en vectores propios de la matriz de covarianza, la cual se calcula de la siguiente manera.

Ecuación 14:

$$\text{cov}(X)=\frac{X^T X}{n-1}$$

Ecuación 15:

$$\text{cov}(X)*p_a=\lambda_a*p_a$$

Ecuación 16:

$$\sum_{n=1}^{m} \lambda_a =1$$

Donde λ_a es el valor propio asociado al vector propio p_a. Por último.

Ecuación 17:

$$t_a=X\ p_a$$

Esta ecuación se puede entender como que t_a sean las proyecciones de X en p_a, donde los valores propios λ_a miden la cantidad de varianza capturada, es decir, la información que representan cada uno de los componentes principales. La cantidad de información que captura cada componente principal va disminuyendo según su número, es decir, el componente principal número uno representa más información que el dos y así sucesivamente.

3.2.1.2. Método basado en correlaciones

El método parte de la matriz de correlaciones, considerando el valor de cada una de las m variables aleatorias F_j. Para cada uno de los n individuos, se toma el valor de estas variables y se escribe el conjunto de datos.

Ecuación 18:

$$\left(F_j^{\beta}\right)_{j=1,\ldots\ldots,m}^{\beta=1,\ldots\ldots,n}$$

Obsérvese que cada conjunto

Ecuación 19:

$$M_j=\left\{F_j^{\beta}\middle|\beta=1,\ldots..,n\right\}$$

Puede considerarse una muestra aleatoria para la variable F_j. A partir de los mxn datos correspondientes a las m variables aleatorias, puede construirse la matriz de correlación muestral, que viene definida por la siguiente ecuación.

Ecuación 20:

$$R=\left[r_{ij}\right]\ \epsilon\ M_{mxn} \quad \text{donde} \quad r_{ij}=\frac{\operatorname{cov}\left(F_i,F_j\right)}{\sqrt{\operatorname{var}(F_i)\operatorname{var}\left(F_j\right)}}$$

Puesto que la matriz de correlaciones es simétrica entonces resulta diagonalmente separable y sus valores propios λ_i lo verifican.

Ecuación 21:

$$\sum_{i=1}^{m} \lambda_i = 1$$

Debido a la propiedad anterior estos m valores propios reciben el nombre de pesos de cada uno de los m componentes principales.

Los factores principales identificados matemáticamente se representan por la base de vectores propios de la matriz R. Está claro que cada una de las variables puede ser expresada como combinación lineal de los vectores propios o componentes principales .

3.2.2. Análisis de Componentes Independientes (ICA)

ICA es una herramienta de análisis cuyo objetivo es descomponer una señal observada en una combinación lineal de fuentes independientes. Surge de la técnica conocida por su sigla BSS (*Blind Separation Source)*, que intenta obtener las fuentes independientes a partir de combinaciones de las mismas.

Mientras que PCA decorrelaciona las señales de entrada utilizando estadísticos de segundo orden (minimizando el error cuadrático medio), ICA minimiza mayores órdenes de dependencia.

Se tiene una matriz de variables independientes llamadas fuentes $S=(S_1 \ldots S_n)$, y una matriz de observaciones X. En esta matriz de observaciones, cada columna es el resultado de un experimento aleatorio, y en cada fila se tiene el valor de una prueba de ese experimento. Como se ha dicho el método ICA trata de descomponer la señal observada en una combinación de fuentes independientes, la matriz de combinación (desconocida) se llamará A**.**

Ecuación 22:

$$X=A*S$$

Con el algoritmo ICA se busca la matriz de separación W, que cumple con la siguiente ecuación

Ecuación 23:

$$U=W*X=W*A*S$$

Donde U es la estimación de máxima probabilidad de las componentes independientes.

La esencia de los algoritmos que implementan ICA es la búsqueda de la matriz W según cierto método iterativo de optimización. Para una matriz U vista como arreglo de vectores, los vectores son estadísticamente independientes cuando.

Ecuación 24:

$$f_U(U) = \prod_i f_{U_i}(U_i)$$

Hay dos formas de implementar el método ICA para el reconocimiento de objetos. Se puede poner en cada fila de la matriz X una imagen diferente, así se tendrá que cada imagen es una variable aleatoria y los píxeles son pruebas. Otra opción es trasponer la matriz X y tener en cada columna una imagen, de manera que en este caso, los píxeles son variables aleatorias y cada imagen una prueba.

3.2.3. Análisis de Discriminación Lineal (LDA)

Este método es una técnica de aprendizaje supervisado para clasificar datos. La idea central de LDA es obtener una proyección de los datos en un espacio de menor (o incluso igual) dimensión que los datos entrantes, con el fin de que la separación de las clases sea la mayor posible. Es una técnica supervisada ya que para poder buscar esa proyección se debe entrenar el sistema con patrones etiquetados.

El escenario de trabajo necesario para aplicar este método al reconocimiento de objetos se basa en que se dispone de un conjunto de imágenes de entrenamiento X_i compuesto por un grupo de objetos en distintas posiciones y con diferentes vistas, etiquetados en C clases**.**

Cada clase cuenta con N_c patrones, y las imágenes de otros objetos diferentes pertenecen a distintas clases, teniendo así el espacio de entrenamiento separado en grupos.

Se trata de obtener un vector de proyección W, que haga que la razón entre la dispersión intraclase y la dispersión interclase sea máxima. Habrá que maximizar la siguiente función objetivo.

Ecuación 25:

$$J(w)=\frac{w^T*S_B*w}{w^T*S_w*w}$$

Donde S_B es la matriz de dispersión inter-clase y S_w es la matriz de dispersión intra-clase.

Ecuación 26:

$$S_B=\sum_c N_c(\mu_c-\mu)(\mu_c-\mu)^T$$

Ecuación 27:

$$S_W=\sum_c \sum_{i\epsilon C}(x_i-\mu_c)(x_i-\mu_c)^T$$

Siendo μ_c la media de cada clase, μ la media de todos los datos, N_c la cantidad de patrones de la clase C.

El vector W que maximiza esta función será aquel que cumpla la siguiente ecuación.

Ecuación 28:

$$S_B*w=\lambda*S_W*w$$

Si S_W es no singular se podria resolver el clásico problema de valores propios para la matriz $S_W^{-1}*S_B$ con la siguiente ecuación.

Ecuación 29:

$$S_W^{-1}*S_B*w=\lambda*w$$

Si ahora sustituimos la solución en la ecuación 25 obtenemos lo siguiente ecuación.

Ecuación 30:

$$J(W)=\frac{w^T*S_B*w}{w^T*S_W*w}=\lambda_k\frac{w_k^T*S_B*w_k}{w_k^T*S_W*w_k}=\lambda_k \quad \text{con } k=1\ldots d$$

Siendo w_k vector propio K de valor propio λ_k.

En consecuencia, para maximizar la solución se debe considerar el vector propio con mayor valor propio asociado.

3.3. Aprendizaje

Las redes neuronales manejan dos tipos de información. La primera, es la información volátil, que se refiere a los datos que se están usando y varían con la dinámica de la computación de la red, se encuentra almacenada en el estado dinámico de las neuronas. El segundo tipo de información que manejan las redes neuronales, es la información no volátil que se mantiene para recordar los patrones aprendidos y se encuentra almacenada en los pesos sinápticos.

El aprendizaje de las redes neuronales, es el proceso de presentar los vectores que se requieren aprender, a la red y el cambio de los pesos de las conexiones sinápticas usando una regla de aprendizaje.

La regla de aprendizaje consiste en algoritmos basados en fórmulas matemáticas, que usando técnicas como minimización del error o la optimización de alguna función, modifican el valor de los pesos sinápticos en función de las entradas disponibles y con ello optimizan la respuesta de la red a las salidas que se desea.

El aprendizaje se basa en el entrenamiento de la red con muestras, que usualmente son llamados vectores de entrenamiento. El proceso usual del algoritmo es que la red ejecuta los vectores iterativamente, cambiando los pesos de las sinapsis, hasta que convergen a un conjunto de pesos óptimos que representan a los vectores lo suficientemente bien, entonces mostrará una respuesta satisfactoria. Esto es, sus pesos sinápticos se ajustan para dar respuestas correctas al conjunto de vectores de entrenamiento que se le ha mostrado.

Sin embargo, hay que destacar que algunas redes no tienen un aprendizaje iterativo como el descrito en el párrafo anterior de presentar los vectores una y otra vez hasta que la red se establece para dar resultados correctos, si no que los pesos de las sinapsis son calculados previamente a partir de los vectores, como en la red de *hopfield*.

Se distingue tres tipos de aprendizaje, aprendizaje supervisado, que consiste en que la red dispone de los vectores de entrada y los vectores de salida que deseamos para esa entrada y en función de ellos se modifican los pesos de las sinapsis para ajustar la entrada a esa salida.

Otro modo de aprendizaje, es el aprendizaje no supervisado, consiste en no presentar vectores de salida, si no solo los vectores de entrada, y dejar a la red clasificar la salida en función de las características comunes de las entradas.

Existe otro tipo de aprendizaje, el aprendizaje reforzado, que usa una fórmula híbrida.

3.3.1. Aprendizaje supervisado

Estos algoritmos requieren el emparejamiento de cada vector de entrada con su correspondiente vector de salida. El entrenamiento consiste en presentar un vector de entrada a la red, calcular la salida de la red, compararla con la salida deseada, calcular el error o diferencia resultante que se utiliza para realimentar la red y cambiar los pesos de acuerdo con un algoritmo que tiende a minimizar el error.

Las parejas de vectores del conjunto de entrenamiento se aplican secuencialmente y de forma cíclica. Se calcula el error y el ajuste de los pesos por cada pareja hasta que el error para el conjunto de entrenamiento entero sea un valor pequeño y aceptable.

.

Las redes más significativas que usan este aprendizaje supervisado son el perceptrón, el perceptrón multicapa y la red de *hopfield*.

3.3.2. Aprendizaje no supervisado

Los sistemas neuronales con entrenamiento supervisado han tenido éxito en muchas aplicaciones y sin embargo tienen muchas críticas debido a que desde el punto de vista biológico no son muy lógicos. Resulta difícil creer que existe un mecanismo en el cerebro que compare las salidas deseadas con las salidas reales. En el caso de que exista.

Los sistemas no supervisados son modelos de aprendizaje más lógicos en los sistemas biológicos. Desarrollados por Kohonen en 1984 y otros investigadores, estos sistemas de aprendizaje no supervisado no requieren de un vector de salidas deseadas y por tanto no se realizan comparaciones entre las salidas reales y salidas esperadas. El conjunto de vectores de entrenamiento consiste únicamente en vectores de entrada. El algoritmo de entrenamiento modifica los pesos de la red de forma que produzca vectores de salida consistentes. El proceso de entrenamiento extrae las propiedades estadísticas del conjunto de vectores de entrenamiento y agrupa en clases los vectores similares.

Normalmente se usa el error cuadrático medio para determinar la similitud, aunque hay otras opciones. En general, los métodos de aprendizaje no supervisado usan representaciones modélicas de los objetos a reconocer y a clasificar. Entre los distintos tipos de aprendizaje no supervisado se distingue, el aprendizaje competitivo.

3.3.3. Aprendizaje competitivo

En el aprendizaje competitivo, las neuronas pugnan entre sí, para representar a una clase o vector de entrada.

La neurona seleccionada es aquella cuyos pesos incidentes se asemejan más al vector de entrada. El aprendizaje consiste en reforzar las conexiones de la unidad ganadora y debilitar las otras, para que los pesos de la unidad ganadora se asemejen cada vez más al vector de entrada.

La reconstrucción de un patrón de entrada a partir de una neurona ganadora consiste en tomar el peso de dicha neurona ya que son los valores que más se asemejan.

3.3.4. Aprendizaje reforzado

La base de este aprendizaje es muy parecida al aprendizaje supervisado pero la información que proporcionamos a la red es mínima se limita a indicar si la respuesta de la red es correcta o incorrecta.

Este tipo de aprendizaje se basa en la noción de condicionamiento por refuerzo, se aprenden las conductas reforzadas positivamente y las conductas reforzadas negativamente .

4. DISEÑO

4.1. Adquisición de la imagen

El *Toolbox* de adquisición de imágenes de Matlab es un conjunto de funciones que sirven, para adquirir imágenes, para visualizar videos, etc. En otras palabras, es la herramienta adecuada para realizar esté diseño.

4.1.1. Conexión del dispositivo

Para conectar un dispositivo de adquisición de imágenes a Matlab, se debe crear un objeto de entrada de video. Este objeto representa la conexión entre Matlab y el dispositivo. Se puede usar las propiedades de este objeto creado para controlar varios aspectos de la adquisición.

Para esto es necesaria alguna información acerca del dispositivo que se desea conectar tales como, el nombre del adaptador de la herramienta de Matlab para conectarse al dispositivo de adquisición de imágenes, el identificador del dispositivo al que se desea acceder, y el formato de video.

4.1.1.1. Nombre del adaptador

Un adaptador es el *software* de la herramienta de Matlab que usa para comunicarse con un dispositivo de adquisición de imágenes vía *driver*. La herramienta de adquisición de imágenes de Matlab incluye adaptadores para algunos vendedores de equipos de adquisición de imágenes y para particulares clases de dispositivos de adquisición de imágenes.

Para determinar cuáles adaptadores están disponibles en el sistema se debe llamar a la función *imaqwinfo.* Esta función retorna información acerca del *software* de adquisición de imágenes de Matlab y consta una lista de adaptadores disponibles en el sistema.

Para mayor información sobre cada uno de ellos indicar su identificador dentro del anterior comando. Por ejemplo, si se trata de video de *Windows* genérico, *imaqhwinfo('winvideo').*

El programa de adquisición fue realizado en Matlab R2009a, el cual incluye dos adaptadores que están cargados y disponibles que por lo general son *matrox* y *winvideo.* En el diseño se utilizo el adaptador *winvideo.*

4.1.1.2. Identificador del Dispositivo (ID)

El adaptador asigna un único número a cada dispositivo con el cual se puede comunicar. El adaptador asigna al primer dispositivo detectado con el ID de 1, al segundo dispositivo detectado con el ID de 2, y así sucesivamente.

Para encontrar el ID de un dispositivo en particular, se debe llamar a la función *imaqwinfo,* especificando el nombre del adaptador. Esta función retorna una estructura conteniendo información acerca de todos los dispositivos disponibles a través del adaptador *matrox* o *winvideo.* El comando debe ser escrito con la siguiente sintaxis *info = imaqwinfo ('winvideo').*

El este diseño solo se utilizo un dispositivo de entrada, por lo tanto el ID del dispositivo es el número 1.

4.1.1.3. Formato de video

El formato de video especifica las características de las imágenes en la secuencia de video, como la resolución de la imagen (largo y ancho), que son usados en la industria, y el tamaño del tipo de dato usado para almacenar la información del píxel. Los dispositivos de adquisición de imágenes comúnmente soportan múltiples formatos de video. Se puede especificar el formato de video cuando se crea el objeto de entrada de video. La especificación de formato puede ser opcional.

La herramienta de adquisición de imágenes de Matlab usa un formato que es por defecto el cual es RGB24 de un tamaño de 352x288 pero en el diseño se utilizo el formato RGB24 320x240.

4.1.2. Adquisición de la imagen

Como se había dicho anteriormente, hay que crear un objeto que controle el dispositivo de adquisición que se encuentra conectada al ordenador, la función *videoinput* crea el objeto, cuya sintaxis es el siguiente, *vidobj* = *videoinput(‘*adaptador*’*, ID, ‘formato*’)* .

En donde *vidobj* es una estructura que posee en sus campos, los parámetros de video necesarios para poder trabajar correctamente el dispositivo de adquisición y los parámetros de entrada son los explicados anterior mente, para el diseño se utiliza el adaptador *winvideo*, el ID es el 1, y el formato el RGB24 352x288.

Antes de empezar a tomar imágenes, es necesario ajustar la posición del dispositivo; el sistema de iluminación, etc. Para realizar estas tareas se requiere crear una pre visualización del entorno, esto es una ventana que muestre de forma continua la señal de vídeo, la función requerida es *preview*, su sintaxis es el siguiente, *preview(vidobj)*.

El único parámetro de entrada necesario es únicamente el objeto creado anteriormente el cual se nombro *vidobj*.

Después de haber creado el objeto de vídeo y tener una ventana de vídeo en línea, se puede cambiar algunas características de la adquisición.

Algunos pasos para una buena adquisición pueden ser, inicializar el objeto de vídeo con la función *start*, escrita de la siguiente forma, *start(vidobj).*

Otro paso al cual hay que prestarle atención es al tipo de disparo para adquirir las imágenes. Por defecto está definido como inmediato, sin embargo se puede hacer manual, que empiece a tomar imágenes después de un retardo, un sin fin de posibilidades que no son parte de este diseño.

Para configurar el disparo comúnmente se utiliza la función *getsnapshot,* escrita de la siguiente forma, *img = getsnapshot(vidobj).*

Esta función tomo una imagen del entorno, el único parámetro de entrada requerido es el objeto creado anteriormente, devolviendo la imagen tomada que es almacenada en la variable *img.*

El siguiente paso sería el de mostrar la imagen tomada, con la función *imshow* se logra desplegar la imagen, escrita de la siguiente forma, *imshow(img).*

En este caso el único parámetro de entrada es la variable que contiene la imagen tomada que es la variable *img*.

Para almacenar la imagen tomada en una ubicación del computador, utiliza la función *imwrite,* escrita de la siguiente forma, *imwrite(img,* 'fichas.jpg'*)*.

Los parámetros de entrada requeridos son; el nombre de la variable que contiene la imagen y el nombre que se le quiera poner a la imagen con su respectivo formato de acuerdo a la tabla II. En este caso la variable *img* contiene la imagen que nos interesa grabar y se almacena con el nombre fichas.jpg con formato JPEG.

Una vez terminado de adquirir la o las imágenes, habrá de liberar los recursos de la memoria utilizando las funciones *closepreview* y *delete*, escritas de la siguiente forma, *closepreview(vidobj), delete(vidobj)*.

La primera cierra la previsualización y la segunda borra todo lo relacionado al objeto creado al inicio.

El programa descrito anterior se reescribirá a continuación para tener una idea clara de los pasos seguidos en el diseño.

```
%% creación de objeto
vidobj = videoinput('winvideo',1,'RGB24 320x240');

%% previsualización
preview(vidobj);

%% inicializar el objeto de video
start(vidobj);

%% configuración, toma de imágenes
img = getsnapshot(vidobj);

%% muestra la imagen tomada
imshow(img);

%% escribe la imagen a una carpeta
imwrite(img,'fichas.jpg');

%% cierra la previsualización
closepreview(vidobj);

%% borra el objeto de video
delete(vidobj);
```

4.2. Procesamiento de la imagen

Cuando la imagen ya allá sido digitalizada y almacenada por el comando *imwrite* se puede leer. Para llamar a una imagen almacenada en el computador al ambiente de Matlab se utiliza la función *imread*, escrita de la siguiente forma, *img= imread('*fichas.jpg*').*

Donde fichas.jpg es el nombre del archivo con el que se encuentra guardado seguido de la extensión del formato, devolviendo tres matrices de la imagen (ya que es una imagen RGB) que se almacenan en la variable *img*, los formatos de imágenes que soporta Matlab son los mostrados en la tabla II.

Tabla II. **Formatos de imágenes soportados por Matlab**

Formato	Extensión
TIFF	.tiff
JPEG	.jpg
GIF	.gif
BMP	.bnp
PNG	.png
XWD	.xwd

Fuente: elaboración propia.

Figura 24. **La imagen fichas**

Fuente: elaboración propia.

Una vez que la imagen este almacenada en una variable, es posible utilizar las funciones de Matlab para procesar la imagen.

En el diseño, la imagen digitalizada almacenada y recuperada es de tipo RGB con resolución de 320x240 píxeles.

Cuando se tenga problemas de iluminación y se requiera ajustar los valores de contraste de la imagen, se utiliza la función *imadjust* especificando el rango de intensidad, existen otras funciones de realce pero para este diseño nos basta con esta función, escrita de la siguiente forma, *im=imadjust(img, stretchlim(img),[ ])*.

En donde los parámetros de entrada la imagen que se quiera ajustar *img,* y los niveles de contraste ya que es una imagen RGB posee 3 matrices, devolviendo 3 matrices con los nuevos contrastes en la variable *im.*

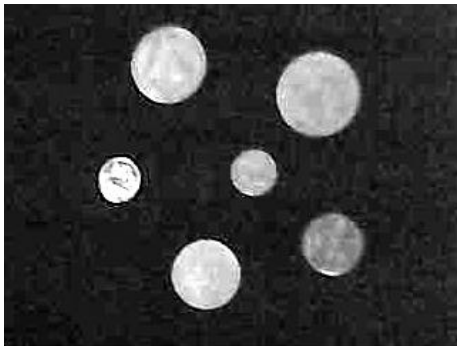
Figura 25. **Imagen ajustada**

Fuente: elaboración propia.

Para ciertas operaciones, puede ser útil convertir una imagen a un tipo diferente. Por ejemplo, se quiere pasar del formato que se tiene el RGB de tres matrices a un formato en escala de grises de solo una matriz, se utiliza la función *rgb2gray*, escrita de la siguiente forma, *imgray=rgb2gray(img)***.**

El único parámetro de entrada requerido es la variable que contiene a la imagen que se quiera cambiar el cual es *img,* devolviendo una matriz que es almacenada en la variable *imgray.*

Figura 26. **Imagen en escala de grises**

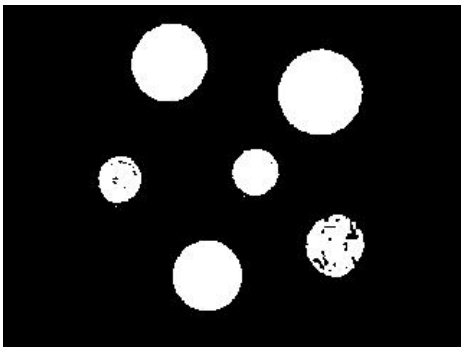


Fuente: elaboración propia.

Ya convertida a escala en grises suele realizarse otra conversión al formato binario para poder manipularla adecuadamente, haciendo a los píxeles obscuros 0 y a los píxeles claros 1, la función utilizada es *im2bw,* escrita de la siguiente forma, *imbw=im2bw(imbray,umbral)*.

Los parámetros requeridos son, el nombre de la variable que contenga la matriz de la imagen en escala de grises o en RGB que se desea convertir, en este caso *imgray* y el nivel umbral que es una constante que indica la división entre lo obscuro y lo claro de la imagen, devolviendo una matriz binaria que se almacena en la variable *imbw*.

Figura 27. **Imagen binarizada**



Fuente: elaboración propia.

Las transformaciones morfológicas son métodos que alteran una imagen binaria, tanto en su forma como en su aspecto, estos métodos nos ayudan a separar el objeto deseado del resto de la imagen.

Este conjunto de funciones inicia con la creación del elemento estructural, la función *strel,* crea una estructura de elementos que sirve como base para las operaciones morfológicas, escrita de la siguiente forma, *es = strel('disk',8)*.

Cuyos parámetros de entradas son, la forma del elemento para este diseño se utiliza el tipo disco especificado como *disk* el cual tiene un radio de 8 píxeles devolviendo una estructura radial almacenada en la variable *es*.

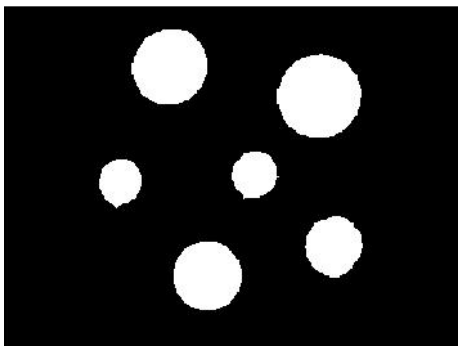
Seguido a la creación del elemento estructural, se prosigue a utilizar algunas operaciones morfológicas ya sea para que permitan la apertura o la clausura de la imagen, para este caso se utiliza la clausura con la función *imclose*, escrita de la siguiente forma, *imdila = imclose(imbw,se)*.

Los parámetros de entrada son, la imagen en binaria o escala en grises en este caso se utiliza la imagen binaria *Imbw*, y el elemento estructural creado anterior devolviendo una matriz que se almacena en la variable *imdila.*

Ya teniendo esta nueva imagen es necesario rellenar algunos píxeles no deseados, utilizando la función *imfill* se podrá rellenar esos agujeros, escrita de la siguiente forma, *imrell = imfill(imdila,'holes')*, rellenamos estos espacios para que la los objetos estén uniformes.

Los parámetros de entrada son, la imagen clausurada y la característica de los píxeles a rellenar en este caso son *holes*, devolviendo una matriz que se almacena en la variable *imrell*.

Figura 28. **Clausura y rellenado de la imagen**

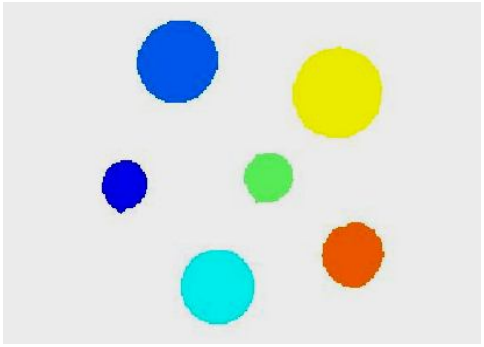


Fuente: elaboración propia.

Cuando se tienen varios elementos en una imagen binaria y se desea separarlas se emplea la función, *bwlabel* la cual realiza un etiquetado de los componentes existentes, este método puede ser considerada como una forma de averiguar cuántos elementos están presentes en la imagen.

La función *bwlabel* tiene el siguiente formato *[L Ne] = bwlabel(imrell, conectividad)*, devolviendo el número de elementos en la variable *Ne* y los niveles de conexión *L* entre objetos en la imagen *imrell*.

Figura 29. **Separación de objetos**

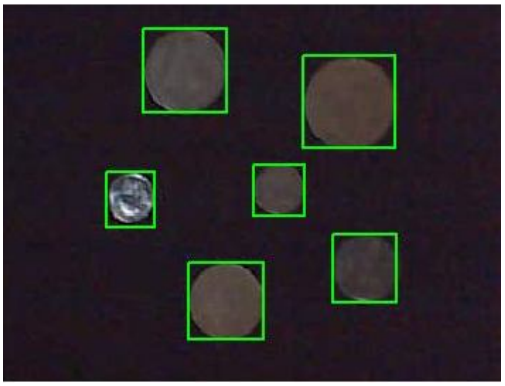


Fuente: elaboración propia.

Si se desea tener información de los objetos encontrados ya sea, su área, su centro, su perímetro, etc. Se utiliza la función *regionprops,* utilizando como único parámetro de entrada los niveles de conexión entre objetos obtenida con la función *bwlabel*, el formato seria *infor=regionprops(L)*, devolviendo una estructura almacenada en la variable *infor*, esta estructura contiene en sus campos las distintas características de cada objeto encontrado en la imagen binaria y sub campos con la información de cada uno de ellos.

En la figura siguiente se muestran los objetos encontrados y encerrados en un cuadro verde, que es una de las propiedades de la función *regionprops* llamada *BoundingBox,* para una mejor descripción vea el código al concluir este tema.

Figura 30. **Objetos encontrados en la imagen**

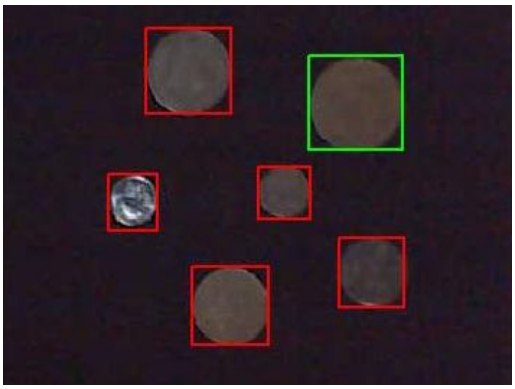


Fuente: elaboración propia.

Un ejemplo en la utilización de la función *reginprops* seria, si se deseara localizar la región de mayor área y cortarla en un cuadro separándola de la imagen original, esto se logra con las funciones que se detallaran a continuación.

Función *max* escrita de la siguiente forma, *[arax p]=max([infor.Area])*, esta función calcula el área máxima del campo *área* de la estructura *infor* obtenidas anteriormente, devolviendo dos parámetros, el valor del área máxima almacenada en la variable *arax,* y su posición en la estructura almacena en la variable *p*, con esto se encontraría el objeto con área máxima de la imagen binaria que sería el objeto con mayor tamaño en la imagen fichas.jpg, en la figura siguiente se muestra el objeto con área mayor encerrada en un cuadro verde mientras que los de menor área se encuentran encerrados en un cuadro rojo.

Figura 31. **Objeto de área mayor encontrado**



Fuente: elaboración propia.

Para cortar el objeto encontrado de mayor tamaño se utiliza la función *imcrop*, escrita de la siguiente forma, *cort = imcrop(img,infor(p).BoundingBox),* los parámetros de entradas son la imagen almacenada en la variable *img*, y el tamaño del objeto con área mayor. Este trozo de imagen posee las dimensiones especificado por el campo de la estructura *infor(p).BoundingBox* en donde *infor(p)* es el objeto y *BoundingBox* es un cuadro que encierra el objeto de mayor área, esta nueva imagen es almacenada en la variable *imcrop.*

Figura 32. **Objeto cortado**

Fuente: elaboración propia.

El programa que se acaba de describir es el principio en el que se baso este diseño para el procesamiento de la imagen y separación de objetos. Existen varias funciones para el procesamiento de imágenes no utilizadas en este diseño ya que Matlab es un potente *software* que permite al programador realizar tareas en distintas formas. El programa descrito anterior se reescribe a continuación para una mayor claridad de los pasos seguidos.

```
%% se lee la imagen almacenada
img = imread('image5.jpg');

%% se aclara la imagen ajustando los valores de intensidad
im = imadjust(img, stretchlim(img),[ ]);

%% convierte la imagen a escala en grises
imgray = rgb2gray(im);

%% binariza la imagen
imbw = im2bw(imgray);

%% Clausura de la imagen
se = strel('disk',8);
imdila = imclose(imbw,se);

%% relleno de la imagen
imrell = imfill(imdila,'holes');

%% separación de objetos
[L Ne] = bwlabel(imrell, 8);
imshow(label2rgb(L));

%% cuadros verdes en objetos
infor = regionprops(L);
imshow(img)
for n = 1:length(infor)
      rectangle('position',infor(n).BoundingBox,'EdgeColor','g','LineWidth',2);
end

%% localización de area mayor
[arax p] = max([infor.Area]);
s = find([infor.Area]< arax);
```

```
%% cuadros rojos en objetos menores
for n = 1:length(s)
      rectangle('position',infor(s(n)).BoundingBox,'EdgeColor','r','LineWidth',2);
end

%% objeto con área mayor, cortada
cort = imcrop(img,infor(p).BoundingBox);
imshow(cort)
```

4.3. Diseño de la red neuronal

La imagen adquirida por el dispositivo, tiene un tamaño de 320x240 en formato RGB, esto da como resultado tres matrices de 76 800 píxeles cada uno, esto implica mucho trabajo, incluso tiempo en el diseño y aprendizaje de la RNA. Es por esta razón que la imagen se procesa para trasladarla a las distintas escalas de conversión y separando los objetos deseados, esta etapa reduce el trabajo de diseño, ya que al procesar la imagen queda solo una matriz de 320x240 en escala de grises o binaria, con el objeto que se quiera reconocer.

En esta etapa del diseño se tomo en cuenta la cantidad de muestras que se deben tomar por cada objeto para una buena identificación, uno de los criterios más utilizados relaciona a un buen reconocimiento con la toma de una muestra de 3 imágenes por objeto en diferentes posiciones. Esto es otro inconveniente para el entrenamiento de una RNA ya que implicaría demasiada memoria del computador en el aprendizaje y demasiado tiempo en el entrenamiento. Para solucionar este inconveniente se utilizaron dos métodos, uno para la reducción de las dimensiones y otro para la reducción de las características.

4.3.1. Reducción de la imagen

Uno de los método para la reducción de la imagen, se realiza en la etapa de procesamiento, el cual consiste en recortar el área en donde se encuentra el objeto a reconocer y el resto es desechada, dando como resultado una disminución de la matriz a un promedio de 10 000 píxeles, dependiendo de la forma y el tamaño del objeto, comparando con la cantidad de píxeles que se tenía anterior de 76 800 píxeles serian unos 66 800 píxeles desechados.

Otro método serian los holísticos basados en estadística, en este trabajo se utilizo el método de análisis de componentes principales (PCA). En el que se tiene un conjunto de imágenes de entrenamiento, en este proyecto 3 imágenes de entrenamiento por cada objeto, seleccionadas aleatoriamente. Como primer paso se genera la base de datos que se va a utilizar para el reconocimiento, se tiene una base de datos para las imágenes recortadas en escala de gris almacenadas en la variable T.

Para crear esta base de datos, cada imagen (que tiene la forma de una matriz) es transformada en un vector columna. Utilizando la función *reshape* se puede transformar una matriz de n filas por m columnas en un vector columna de n*m filas, la sintaxis de la función es la siguiente *T=reshape(*nombre de la imagen,n*m,1*)*. Los vectores columna de cada una de las imágenes forman la matriz T. Si se tienen k imágenes, y cada imagen es de tamaño nxm, las dimensiones de esta matriz serán (n*m)xk.

A continuación se calcula la imagen promedio del conjunto de entrenamiento $\mu=\frac{1}{k}\sum_{i=1}^{k} t_i$, y se le resta a cada una de las imágenes de este conjunto $\varphi_i=t_i-\mu$, teniendo así la matriz $A=[\varphi_1 \ \varphi_2 \ldots\ldots \varphi_m]$.

Seguidamente habrá que calcular la matriz de covarianza de esta forma $C=A^T*A$ para después obtener los autovalores y autovectores, esto nos lo proporciona directamente la función *eig* cuya sintaxis se escribe de la siguiente forma, *[*autovectores autovalores*]=eig(C).*

A continuación se seleccionan los N autovectores de la matriz de covarianza que tienen mayores autovalores asociados. La elección del número de autovalores que se quiere conservar es crítica, un número muy alto supone un tiempo de procesamiento grande y una mayor necesidad de memoria en la etapa de clasificación. Un número muy bajo supone que se puede perder mucha información. Se trata por lo tanto de encontrar un equilibrio, para lo cual se observa el valor de los autovalores, para ver cómo evoluciona, y además se trata de mantener la mayor parte posible de la varianza inicial de la base de datos.

Como parte del diseño se toma el criterio de seleccionar los valores de autovectores que haga que se mantenga el 80% de la varianza, ya que se considera que este porcentaje es suficiente para el análisis, y se corresponde aproximadamente con el número de autovalores que tienen mayor valor.

Una vez que se han seleccionado los autovectores que van a ser utilizados, se proyectan todas las imágenes de la base en el nuevo espacio reducido, teniendo así los coeficientes de la representación de la imagen, que serán las características que se están buscando y que pasan al clasificador.

Cuando se tiene una nueva imagen de algún objeto que debe ser reconocida se sigue el mismo proceso. La imagen nueva es transformada en un vector columna, a continuación se le resta la media de las imágenes de entrenamiento y por último se calcula su proyección en el espacio reducido calculado con las imágenes de entrenamiento. Así se obtienen los coeficientes de la nueva imagen a reconocer.

4.3.2. Arquitectura de la red neuronal

Entre los distintos tipos de redes neuronales que pueden implementarse con Matlab se decide utilizar una red neuronal alimentada hacia delante en cascada (CFNN) que es un tipo de red recurrente, con aprendizaje *backpropagation*, este tipo de red incluye una conexión desde la entrada a cada una de las capas, y desde cada capa a la capa siguiente, es decir, si por ejemplo se tiene una red con tres capas habrá conexiones de la capa 1 a la capa 2, de la capa 2 a la capa 3 y de la capa 1 a la capa 3. Estas conexiones adicionales pueden mejorar la velocidad a la que la red aprende las relaciones deseadas.

La red que se va a crear tiene 3 capas, una capa de entrada, una capa oculta y una capa de salida, para los resultados que se van a presentar, el número de neuronas de la capa de entrada de cada red es igual al tamaño del vector de características que se obtiene en la etapa anterior, que como se dijo varía en función del número de autovectores que se utilizan para representar las imágenes. La capa de salida tiene un número de neuronas igual al número de imágenes que hay en la base de datos y que deben ser reconocidos, es decir, cada neurona de salida se corresponde con un objeto.

En cuanto a las neuronas de la capa oculta, no existe una regla fija que determine su número, pero si hay algunos criterios que han sido muy utilizados. Según uno de estos criterios, el número necesario de neuronas para la capa oculta es la mitad de la suma de las neuronas utilizadas en las capas de entrada y salida. También suele aplicarse el criterio según el cual el número de neuronas de la capa oculta no puede ser superior al doble del número de neuronas de entrada.

En este proyecto se ha tratado de utilizar un número de neuronas de la capa oculta que se aproxime a los dos criterios mencionados anteriormente. Tras realizar un estudio de las situaciones que se van a tener para realizar las pruebas se determina que el número de neuronas de la capa oculta es el siguiente.

Ecuación 31:

$$\text{neuronas ocultas} = \frac{\text{neuronas entrada}+\text{neuronas salida}}{2} - 2$$

Se tiene un número de neuronas que es la mitad de la suma de las neuronas de entrada y salida, pero se le resta 2 para adaptarse al segundo criterio que dice que el número de neuronas de la capa oculta no puede superar el doble del número de neuronas de la capa de entrada. El caso en el que se tienen 17 neuronas de entrada y 55 de salida. Según el primer criterio se tendrían 36 ((17+55)/2) neuronas, y según el segundo criterio el número de neuronas no puede ser mayor a 34 (17*2), por eso se le resta dos.

Para cada una de las capas de la red neuronal debe definirse una función de transferencia. Se decide que para la capa de entrada se utilice la función *tansig*, para la capa oculta la función *tansig* y para la capa de salida la función *purelin*.

Además se define como función de aprendizaje la función *learngdm* que realiza el aprendizaje mediante *backpropagation* por descenso de gradiente, y como función de actualización *trainlm*, que actualiza los pesos.

Para generar la arquitectura de la red neuronal que se va a utilizar para la fase de reconocimiento se emplea la siguiente función *newcf* la cual crea una red CFNN con *backpropagation*, cuyas entradas son, la matriz de entradas, una matriz de salidas deseadas, el tamaño de cada capa, la función de transferencia de cada capa, la función de entrenamiento y la función de aprendizaje descritas anterior mente, devolviendo la red creada con las características especificadas.

4.3.3. Entrenamiento de la red neuronal

Una vez que se tiene definida la arquitectura de la red neuronal que se va a utilizar, el siguiente paso es su entrenamiento. Para ello se entregan a la red los vectores de características de las imágenes de entrenamiento (3 imágenes por cada objeto de la base de datos), es decir, las imágenes representadas en el espacio reducido formado por los autovectores seleccionados. También es necesario proporcionar a la red las salidas correspondientes a cada una de las imágenes de entrenamiento, ya que se va a realizar un aprendizaje supervisado.

La salida para cada imagen de entrenamiento será un vector binario, formado por uno en la neurona correspondiente al objeto de la imagen, y cero en el resto de neuronas, para el entrenamiento se define arbitrariamente un error objetivo de 0,1 y un número máximo de épocas de 200.

Para realizar el entrenamiento de las redes neuronales se utiliza la función de Matlab, *train,* la cual realiza el entrenamiento de la red neuronal cuyas funciones de entradas son, la red neuronal a entrenar, entradas para el entrenamiento, salidas para el entrenamiento y devuelve, una red entrenada, el registro del proceso de entrenamiento, salida de la red, errores de la red.

Con las redes neuronales entrenadas ya es posible realizar el reconocimiento. Para ello se proporciona al sistema de la fase de reconocimiento una imagen de las del conjunto de prueba y se extraen sus características. El vector de características es introducido en la red correspondiente de manera que se obtiene un vector de salida de la red neuronal.

4.3.4. Simulación de la red neuronal

A continuación se obtiene la diferencia entre el vector de salida de la red neuronal y un vector formado por unos. De esta manera se tiene una idea del parecido de la imagen de prueba a cada una de las imágenes que forman la base de datos para el reconocimiento.

Se seleccionan una imagen de la base de datos a la que más se parece el objeto de la imagen de prueba, es decir, se seleccionan aquellos objetos de la base de datos para el que la diferencia calculada es menor.

Para la clasificación se utilizan funciones básicas de Matlab, además cabe destacar el uso de la siguiente función *sim* la cual simula una red neuronal, cuyas entradas son la respectiva red entrenada y la entrada de prueba a la red, devolviendo una salida de selección.

4.4. Diseño de la interfaz visual

Matlab posee una potente herramienta llamada Guide, la cual nos permite crear interfaces gráficas con facilidad, la figura siguiente muestra la interfaz gráfica principal.

Figura 33. **Interfaz principal**

Fuente: elaboración propia.

El botón pulsador analizar, pone la pantalla principal a escanear el entorno en busca de una imagen que reconocer, si se encuentra alguna imagen, se traslada una fotografía a la pantalla fotográfica que es la encargada de almacenar la imagen que se quiera analizar, después del análisis, la imagen con la que más se asemeje encontrada en la base de datos es mostrada en la pantalla de reconocimiento.

El botón pulsador exportar, sirve para analizar una imagen que se encuentra almacenada en algún archivo, al pulsarlo nos da la opción de buscar la imagen en formato JPEG la cual es trasladada directamente a la pantalla fotográfica luego es analizada y seguidamente se muestra una imagen a la que más se asemeje de la base de datos en la pantalla de reconocimiento.

El botón pulsador parar, para inmediatamente todo proceso que se esté llevando, solo dejando la pantalla principal en modo de escaneo. Todo lo anterior es válido si previamente se ha creado una base de datos con imágenes para poder entrenar a la red neuronal y así esta poder analizar las imágenes.

En la figura siguiente se muestra la interfaz que nos crea la base de datos.

Figura 34. **Interfaz de la adquisición**

Fuente: elaboración propia.

Botón pulsador tomar, tiene la función de tomar una fotografía de la pantalla principal que se encuentra en modo de escaneo de imágenes, esta fotografía es mostrada en la pantalla fotográfica.

Botón pulsador exportar, tiene la función de exportar alguna imagen que tengamos almacenado en algún archivo, nos da la opción de buscarla, luego es mostrada en la pantalla fotográfica.

Editor de texto ingresar nombre, en ella se escribe el nombre con el que se desea almacenar la imagen localizada en la pantalla fotográfica.

Botón pulsador guardar, tiene la función de guardar cualquier imagen que se encuentra colocada en la pantalla fotográfica y con el nombre escrito en el editor de texto desplegándonos un mensaje de imagen correctamente guardada.

Después de que se halla creado la base de datos se prosigue con el entrenamiento de la red neuronal. En la figura siguiente muestra la interfaz de entrenamiento.

Figura 35. **Interfaz de entrenamiento**

Fuente: elaboración propia.

Botón pulsador cargar, tiene la función de cargar la base de datos creada anteriormente y convertirlas en datos adecuados para proporcionárselos a los parámetros de red, en esta etapa también se construye la arquitectura de la red neuronal.

Botón pulsador entrenar, tiene la función de entrenar la red con los parámetros creados en paso anterior.

El programa posee un panel de menús en la parte superior, el cual se muestra en la figura siguiente.

Figura 36. **Panel de menús**

Fuente: elaboración propia.

Cada menú contiene otros submenús, el menú de archivo contiene los submenús, borrar base de datos y salir.

El submenú borrar base de datos, cuya función es la de eliminar tanto las imágenes que se encuentran en la base de datos así como la configuración de la red creada, en consecuencia se tendría que volver a crear una base de datos con nuevas imágenes y volver a entrenar la red con la nueva base de datos.

El submenú salir, tiene la función de cerrar el programa sin eliminar la base de datos ni la configuración de la red esto nos permite volver a utilizarla para análisis posteriores.

Figura 37. **Menú archivo**

Fuente: elaboración propia.

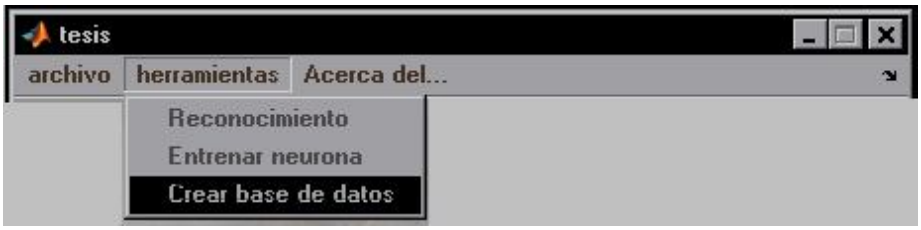
La opción de herramienta contiene los submenús, reconocimiento, entrenar neurona, crear base de datos.

El submenú reconocimiento, permite acceder a la interfaz principal la cual tiene la función de escanear su entorno y reconocer los objetos que encuentre.

El submenú entrenar neurona, permite acceder a la interfaz de entrenamiento esta tiene como función configurar y entrenar la red neuronal con la base de datos correspondiente.

El submenú crear base de datos, permite acceder a la interfaz de adquisición, esta tiene como función la de adquirir imágenes digitales mediante la cámara para poder crear la base de datos.

Figura 38. **Menú herramientas**



Fuente: elaboración propia.

CONCLUSIONES

1. Uno de los elementos que tiene gran influencia en la detección de imágenes es el escenario en donde se realizan las pruebas, ya que se llega a tener variaciones en la iluminación.

2. Con las distintas herramientas que posee Matlab se reduce la complejidad del procesamiento digital de imágenes.

3. La capacidad de procesamiento de las RNA es adquirida durante el proceso de aprendizaje, el cual consiste en variar los valores de los pesos sinápticos de las neuronas de la red, de acuerdo con un algoritmo de entrenamiento, de forma que se obtengan las salidas deseadas para todos los ejemplos de entrada.

4. La comprobación del tipo de imágenes que se utilizan en el sistema de reconocimiento influyen de manera determinante en los resultados que se obtienen, y en la elección de determinados parámetros.

5. Para la fase de reconocimiento se ha propuesto la extracción de las características que representan las imágenes mediante PCA. El número de autovectores que representan las imágenes viene dado por el número de autovalores que mantienen el 80% de la varianza. Este número no tiene porque ser el mismo para distintas pruebas con distintas base de datos.

6. Para poder reducir las dimensiones de las imágenes son necesarios los métodos estadísticos tales como PCA, ya que sin reducirlas llevaría demasiado tiempo el poder crear una RNA.

7. El reconocimiento de objetos es una de las aplicaciones del procesamiento digital de imágenes que permite interacción entre seres humanos y computadoras.

8. Los algoritmos de procesamiento de imágenes utilizados han permitido tener una adecuada descripción e identificación de características de los objetos.

RECOMENDACIONES

1. Las RNA, pueden ser utilizadas como alternativa en la solución de problemas en las que se requieren procesos de automatización.

2. Es necesario analizar detalladamente un problema para poder diseñar una RNA, ya que existen una gran variedad de RNA que podrían dar una solución, y unos pueden ser mejores que otros.

3. Investigar el potencial del sistema de reconocimiento de objetos como la base para distintas aplicaciones, modificando este diseño ya sea para el reconocimiento de rostros.

4. En la etapa de procesamiento podría utilizarse más filtros para eliminar el ruido y dejar la imagen deseada.

5. Realizar un estudio del número de neuronas que son necesarias en la capa oculta de las RNA que se utiliza en la etapa de clasificación.

6. En la etapa de procesamiento se podría implementar un algoritmo que permitiera compensar los cambios de iluminación y de esta manera no tener errores en el cambio de escenario.

7. En este diseño se utilizo el método estadístico PCA para reducir las dimensionalidad de la imagen, pero podrían hacerse pruebas con los otros métodos ya sea el ICA o el LDA.

8. Implementar los temas de procesamiento de imágenes y redes neuronales en lo que es el pensum de estudios en la carrera de ingeniería en electrónica ya que son temas muy importantes en la formación de un ingeniero.

BIBLIOGRAFÍA

1. BALLESTEROS, Alfonso. *Tutorial de redes neuronales artificiales* [en línea]. España: Universidad de Málaga, 2008 [ref. noviembre de 2010]. Disponible en Web: <http://www.redes-neuronales.netfirms.com.htm>.

2. BASOGAIN OLABE, Xabier. *Redes neuronales artificiales y sus aplicaciones*. Bilbao: Escuela Superior de Ingeniería, 79 p.

3. BEALE, Mark Hudson. *Neural Network Toolbox* [en línea]. 7a ed. Estados Unidos: The Mathworks, septiembre 2010 [ref. diciembre de 2010]. Disponible en Web: <www.mathworks.com>.

4. CUEVAS JIÉNEZ, Erik Valdemar. *Visión por computador utilizando Matlab y el toolbox de procesamiento digital de imágenes***.** 2006. 35p.

5. ESQUEDA ELIZONDO, José Jaime. *Fundamentos de Procesamiento de Imágenes*. México: Universidad Autónoma de Baja California, 2002. 51 p.

6. Facultad de Ingeniería Eléctrica. *Tutorial de redes neuronales* [en linea]. Pereira: Universidad Tecnológica, 2000 [ref. diciembre de 2010]. Disponible en Web: <http://ohm.utp.edu.co/neuronales.htm>.

7. GÁMEZ JIMÉNEZ, Carmen Virginia. *Diseño y desarrollo de un sistema de reconocimiento de caras***.** España: Universidad Carlos III de Madrid, Escuela Politécnica Superior, 2009. 160 p.

8. GESTAL POSE, Marcos. *Introducción a las redes neuronales* [en línea]. Universidad de Coruña: Depto. Tecnología de la información y comunicación, 2009 [ref. diciembre de 2010], Disponible en Web. <http://sabia.udc.es/~mgestal>.

9. GONZÁLES, Rafael; WOODS, Richard. *Digital image processing*. 2a ed. New Jersey: Pearson Prentice-Hall, 2004. 344 p. ISBN 0-13-008519-7.

10. Mathworks**.** *Creating graphical user interfaces* [en línea]. 8a ed. Estados Unidos: The Mathworks, marzo 2009 [ref. diciembre de 2010]. Disponible en Web: <www.mathworks.com>.

11. _____**.** *Image Processing Toolbox* [en línea]. 6a ed. Estados Unidos: The Mathworks, 2008 [ref. diciembre de 2010]. Disponible en Web: <www.mathworks.com>.

12. _____. *Librería de redes neuronales para Matlab* [en línea]. Estados Unidos: The Mathworks, 2010 [ref. diciembre de 2010]. Disponible en Web: <http://www.mathworks.com/access/helpdesk /help/pdf_doc/nnet /nnet.pdf.>.

13. QUIROA R, J. Vladimir. *Elementos procesadores neuronales* [en línea]. México: Universidad de Guadalajara, 2005 [ref. octubre de 2010]. Disponible en Web: <http://proton.ucting.udg.mx>.

14. ROJAS, Raúl. *Neural Networks*. Berlin: Springer-Verlag, 1995. 509 p.

15. SERRANO, Antonio. *Redes neuronales artificiales*. España: Universidad de Valencia, 2009. 145 p.

16. SMITH, Lindsey. *A tutorial on principal component analysis* [en linea]. 2002 [ref. octubre de 2010]. Disponible en Web: <http://www.cs.otago.ac.nz/cosc453/student_tutorials/principal_components.pdf>.

17. SOBRADO MALPARTIDA, Eddie Angel. *Sistema de visión artificial para el reconocimiento y manipulación de objetos utilizando un brazo robot*. Perú: Universidad Católica, Escuela de graduados, 2003. 138 p.

APÉNDICE

El código del programa utilizado se muestra a continuación.

```
function varargout = tesis(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @tesis_OpeningFcn, ...
                   'gui_OutputFcn',  @tesis_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end

% Función que se ejecuta cuando el programa es activado. Manda a cargar
% todo lo relacionado con la apariencia del mismo
function tesis_OpeningFcn(hObject, eventdata, handles, varargin)
global a
a = imread('Escudo.jpg');imshow(a,'Parent',handles.logo)
vidRes = get(handles.vid, 'VideoResolution');
imageRes = fliplr(vidRes);
handles.hImage = imshow(zeros(imageRes),'Parent',handles.toma);
preview(handles.vid,handles.hImage);
set(handles.nombre_muestra,'String','')
try
    mkdir('C:\fotos')
    mkdir('C:\variables')
end
set(handles.panel2,'visible','off');
set(handles.panel3,'visible','off');
set(handles.text2,'visible','off');
set(handles.panel1,'visible','on');
handles.output = hObject;
guidata(hObject, handles);

function varargout = tesis_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;
```

```
% Función que crea el objeto de video para la adquisición de
% imágenes y configura la previsualización
function tesis_CreateFcn(hObject, eventdata, handles)
a = imaqhwinfo;
b = char(a.InstalledAdaptors(2));
a = imaqhwinfo(b);
c = char(a.DeviceInfo.SupportedFormats(1,9));
try
   handles.vid = videoinput(b,1,c);
   preview(handles.vid);
   closepreview(handles.vid)
catch
   closepreview(handles.vid)
   questdlg('conecte cámara por favor ','cámara desconectada ',' aceptar ',' aceptar');
   close all
end
guidata(hObject, handles);

% función del botón pulsador tomar de la interfaz de adquisición
function foto_Callback(hObject, eventdata, handles)
   ima = getsnapshot(handles.vid);
   img = imadjust(ima,stretchlim(ima),[]);
   im_g = rangefilt(img);
   umb = graythresh(im_g);bw = im2bw(im_g,umb);
   se = strel('disk',8);
   bw = imclose(bw,se);
   bw = imfill(bw,'holes');
   [L Ne]=bwlabel(bw);
   BB= regionprops(L);
   [arax p]=max([BB.Area]);
   if Ne > 0
      cort = imcrop(ima,BB(p).BoundingBox);imshow(cort,'Parent',handles.extra);
   end
   drawnow

% Función del botón pulsador guardar de la interfaz de adquisición
function guardar_Callback(hObject, eventdata, handles)
img = getimage(handles.extra);
if isempty(img), return, end
if strcmp(get(handles.nombre_muestra,'string'),''),
   msgbox('escriba el nombre','error');return;
end
name=get(handles.nombre_muestra,'string');
```

```
name=strcat('C:\fotos\',name);
imwrite(img,strcat(name,'.jpg'));
set(handles.nombre_muestra,'String','')
msgbox('Imagen correctamente guardada ','save');
guidata(hObject, handles);

% Función del botón pulsador exportar de la interfaz de adquisición
function exportarB_Callback(hObject, eventdata, handles)
[filename path]=uigetfile('*.jpg',' escoge una imagen');
if path ~=0
   ima = imread([path filename]);
   img = imadjust(ima,stretchlim(ima),[]);
   im_g = rangefilt(img);
   umb = graythresh(im_g);bw = im2bw(im_g,umb);
   se = strel('disk',8);
   bw = imclose(bw,se);
   bw = imfill(bw,'holes');
   [L Ne]=bwlabel(bw);
   BB= regionprops(L);
   [arax p]=max([BB.Area]);
   if Ne > 0
      cort = imcrop(ima,BB(p).BoundingBox);
      imshow(cort,'Parent',handles.extra);
   end
   drawnow
end
guidata(hObject, handles);

% Función del botón pulsador entrenar de la interfaz de entrenamiento
% de la red
function inent_Callback(hObject, eventdata, handles)
try
   load c:\variables\p.mat;
catch
   set(handles.text2,'String','primero tiene que cargar los archivos');
   return;
end
set(handles.text2,'String','espere un momento red entrenándose');
pause(1e-6)
v=T;
O=uint8(ones(1,size(v,2)));
m=uint8(mean(v,2));
h = single(m)*single(O);
vzm=uint8(v)-uint8(h);
```

```
L=single(vzm)'*single(vzm);
[V,D]=eig(L);
V=single(vzm)*V;
N=11;
V=V(:,end:-1:end-(N-1));
cv=zeros(size(v,2),N);
for i=1:size(v,2);
   cv(i,:)=single(vzm(:,i))'*V;
end
Train_Number=size(cv,2);
num_personajes=size(v,2)/3;
num_train=3;
y_train=zeros(num_personajes,Train_Number);
j=1;
for i=1:num_personajes
   for k=j:j+(num_train-1)
      y_train(i,k)=1;
   end
   j=j+num_train;
end
x_train=cv';
m =size(x_train,1);
p =size(y_train,1);
neuronas_hidden=round(((m+p)/2))-2;
tamano_capas=[m,neuronas_hidden];
net.numInputs= m;
net=newcf(x_train,y_train,tamano_capas,{'tansig'                    'tansig'
'purelin'},'trainlm','learngdm','mse');
net.trainParam.epochs = 500;
net.trainParam.goal = 0.01;
net.trainParam.mu_max=1e18;
net.trainParam.max_fail=5;
red=train(net,x_train,y_train);
save c:\variables\red.mat red
set(handles.text2,'String','Proceso terminado puede iniciar el reconocimiento de
objetos');
guidata(hObject, handles);

% Función del botón pulsador cargar de la interfaz de entrenamiento
% de la red
function cargar_Callback(hObject, eventdata, handles)
set(handles.text2,'String','espere un momento archivos cargándose');
pause(1e-5)
if exist('c:\fotos\Thumbs.db')==2
```

```
    delete c:\fotos\Thumbs.db
end
Path='C:\fotos\';
Files = dir(Path);
Number = length(Files);
T=[];
for j=3:Number
    img = imread(strcat(Path,Files(j).name));
    img = imadjust(img,stretchlim(img),[]);
    img = rgb2gray(img);
    [y(j-2) x(j-2)]=size(img);
end
a(1)= max(y);
a(2)= max(x);
for j=3:Number
    img = imread(strcat(Path,Files(j).name));
    img= rgb2gray(img);
    img=imresize(img,[a(1) a(2)]);
    temp = reshape(img,a(1)*a(2),1);
    T = [T temp];
end
save c:\variables\p.mat T a
set(handles.text2,'String','archivos cargados con éxito puede comenzar a
entrenar la red');

% Función del botón pulsador exportar de la interfaz principal
function exportar_Callback(hObject, eventdata, handles)
 try
    [filename path]=uigetfile('*.jpg',' escoge una imagen');
    if path ==0;return;end
    ima = imread([path filename]);imshow(ima,'Parent',handles.extra);
    img = imadjust(ima,stretchlim(ima),[]);
    im_g = rangefilt(img);
    umb = graythresh(im_g);bw = im2bw(im_g,umb);
    se = strel('disk',8);
    bw = imclose(bw,se);
    bw = imfill(bw,'holes');
    [L Ne]=bwlabel(bw);
    BB= regionprops(L);
    [arax p]=max([BB.Area]);
    if Ne > 0
        cort = imcrop(ima,BB(p).BoundingBox);
        load c:\variables\p.mat;
        load c:\variables\red.mat;
```

```
        i = imadjust(cort,stretchlim(cort),[]);
        r= rgb2gray(i);
        r =imresize(r,[a(1) a(2)]);
        r = reshape(r,a(1)*a(2),1);
        O=uint8(ones(1,size(T,2)));
        m=uint8(mean(T,2));
        vzm=uint8(T)-uint8(single(m)*single(O));
        L=single(vzm)'*single(vzm);
        [V,D]=eig(L);
        V=single(vzm)*V;
        V=V(:,end:-1:end-10);
        p=r-m;
        s=single(p)'*V;
        test=double(s');
        y=sim(red,test);
        Path='C:\fotos\';
        Files=dir(Path);
        [v i]= max(y);
        str=Files((i*3)+2).name;
        i = imread(strcat(Path,str));imshow(i,'Parent',handles.camara)
    end
catch
    msgbox('cree una base de datos ','error');
end
guidata(hObject, handles);
% Función del botón pulsador analizar de la interfaz principal
function analizar_Callback(hObject, eventdata, handles)
try
    load C:\variables\p.mat;
    load C:\variables\red.mat;
catch
   msgbox('Necesita crear base de datos ','error');
   return;
end
    global bandera1
    bandera1 = 'true';
    while(strcmp(bandera1 , 'true')==1)
      img = getsnapshot(handles.vid);imshow(img,'Parent',handles.extra)
      im_g = imadjust(img,stretchlim(img),[]);
      im_g = rangefilt(im_g);
      umb = graythresh(im_g);bw = im2bw(im_g,umb);
      se = strel('disk',10);
      bw = imclose(bw,se);
      bw = imfill(bw,'holes');
```

```
    [L Ne]=bwlabel(bw);
    BB= regionprops(L);
    [arax p]=max([BB.Area]);
    if Ne > 0
        rectangle('position',BB(p).BoundingBox,'EdgeColor','g','LineWidth',2);
        drawnow
        cort = imcrop(img,BB(p).BoundingBox);
        cort = reconocimiento(cort);
        imshow(cort,'Parent',handles.camara);
        drawnow
    end
  end
  guidata(hObject, handles);

% Función del botón pulsador parar de la interfaz principal
function parar_Callback(hObject, eventdata, handles)
global bandera1
bandera1 = 'false';
guidata(hObject, handles);

% submenú borrar base de datos del menú archivo
function borrar_Callback(hObject, eventdata, handles)
try
  Path='C:\fotos\';
  Files = dir(Path);
  Number = length(Files);
  for j=3:Number
     delete(strcat(Path,Files(j).name));
  end
  delete c:\variables\p.mat
  delete c:\variables\red.mat
catch
  return;
end
guidata(hObject, handles);

 % submenú salir del menú archivo
function salir_Callback(hObject, eventdata, handles)
global bandera1
bandera1 = 'false';
imaqreset
close all

 % submenú reconocimiento del menú herramientas
```

```
function run_Callback(hObject, eventdata, handles)
    set(handles.panel2,'visible','off');
    set(handles.panel3,'visible','off');
    set(handles.text2,'visible','off');
    set(handles.panel1,'visible','on');
    set(handles.camara,'visible','on','color',[0    0    0],'Position',[59.6    0.769    27.2
7.69],'XTick',[],'YTick',[],'XTick',[]);
    guidata(hObject, handles);

 % submenú entrenar neurona del menú herramientas
function entrenar_Callback(hObject, eventdata, handles)
    set(handles.panel2,'visible','off');
    set(handles.panel3,'visible','on');
    set(handles.panel1,'visible','off');
    set(handles.text2,'visible','on');
    cla(handles.camara);
    set(handles.camara,'visible','off');
```

ANEXO

A continuación se muestra el código de la función PCA. El código se obtuvo de la tesis hecha por Carmen Virginia Gámez Jiménez, este código fue modificado para ajustarse al programa.

```
function [y_train, x_train]=pca(T)
v=single(T);
O=ones(1,size(v,2));        %vector fila de unos con la cantidad de imágenes
m=mean(v,2);                %vector columna con del número medio entre filas y
                            %con la misma cantidad de imágenes
h = single(m*O);            %matriz con las mismas dimensiones que T
                            %conteniendo los valores medios de cada fila
vzm=v-h;                    %matriz que contiene la diferencia entre la matriz
                            %de imagen y las medias
L=vzm'*vzm;                 %matriz de covarianzas
[V,D]=eig(L);               %eigen vectores y eigen valores
V=vzm*V;
N=11;
V=V(:,end:-1:end-(N-1));
cv=zeros(size(v,2),N);
for i=1:size(v,2);
   cv(i,:)=vzm(:,i)'*V;
end
Train_Number=size(cv,2);
num_personajes=size(v,2)/3;
num_train=3;
y_train=zeros(num_personajes,Train_Number);
j=1;
for i=1:num_personajes
   for k=j:j+(num_train-1)
      y_train(i,k)=1;
   end
   j=j+num_train;
end
x_train=cv';
```