RECONOCIMIENTO GESTUAL PARA INTERFAZ HOMBRE MÁQUINA BAJO ENTORNO CONTROLADO

DIEGO FELIPE SORA VARGAS

FUNDACIÓN UNIVERSITARIA LOS LIBERTADORES
FACULTAD DE INGENIERÍA
PROGRAMA DE INGENIERÍA ELECTRÓNICA
BOGOTÁ, D.C.
2014

RECONOCIMIENTO GESTUAL PARA INTERFAZ HOMBRE MÁQUINA BAJO ENTORNO CONTROLADO

DIEGO FELIPE SORA VARGAS

TRABAJO DE GRADO PARA OPTAR POR EL TITULO DE INGENIERO ELECTRÓNICO

Director Científico:
Ing. MSc I&E IVÁN DARÍO LADINO VEGA

FUNDACIÓN UNIVERSITARIA LOS LIBERTADORES
FACULTAD DE INGENIERÍA
PROGRAMA DE INGENIERÍA ELECTRÓNICA
BOGOTÁ, D.C.
2014

Nota de aceptación

_____
_____
_____
_____
_____
_____
_____

_____
Firma presidente del Jurado

_____
Firma Jurado

_____
Firma Jurado

Bogotá, D.C. 31 de Enero de 2014

Las directivas de la Fundación Universitaria Los Libertadores, los jurados calificadores y el cuerpo docente no son responsables por los criterios e ideas expuestas en el presente documento. Estos corresponden únicamente al autor.

AGRADECIMIENTOS

A Dios, por haberme brindado situaciones de crecimiento a lo largo de mi vida, por haberme dado entendimiento, sabiduría y paciencia para afrontar las dificultades que se presentaron a lo largo del desarrollo de este constante proceso de formación.

A mis padres, que con su esfuerzo, acompañamiento, consejos y principios ayudaron a formarme como persona y profesional.

A mis hermanos, que con su apoyo y confianza ayudaron al cumplimiento de los logros trazados.

A la Fundación Universitaria Los Libertadores, por proveerme de un espacio académico y conocimientos suficientes para lograr una de mis metas académicas.

Al ingeniero Iván Ladino Vega, director del proyecto de grado, por haberme hecho participe de este proyecto, y que con su asesoría, apoyo conceptual, concejo y tiempo ayudaron de manera significativa en el desarrollo y culminación del mismo.

Y finalmente a mis compañeros de carrera, con quienes compartí, aprendí y disfrute en este largo periodo de aprendizaje.

CONTENIDO

pág.

INTRODUCCIÓN 16

OBJETIVOS 17

1. VISIÓN ARTIFICIAL 18
1.1 DEFINICIÓN DE INTELIGENCIA ARTIFICIAL 18
1.1.1 Áreas de aplicación de la inteligencia artificial 18
1.2 BASES DE LA INTELIGENCIA ARTIFICIAL MODERNA 18
1.3 SISTEMAS DE VISIÓN ARTIFICIAL 19
1.3.1 Definición técnica de visión artificial 19
1.3.2 Objetivos de los sistemas visión artificial 19
1.3.3 Aplicaciones de los sistemas visión artificial 20
1.3.4 Diagrama de bloques de un sistema de visión artificial 21

2. PROCESOS DE UN SISTEMA DE VISIÓN ARTIFICIAL 23
2.1 ADQUISICIÓN DE IMÁGENES 23
2.1.1 Cámara de adquisición de imágenes 23
2.2 PREPROCESAMIENTO DE IMÁGENES 24
2.2.1 Modelos de color 24
2.2.1.1 Modelo RGB 24
2.2.1.2 Modelo HLS 25
2.2.1.3 Modelo HSB o HSV 26
2.2.1.4 Modelo CMYK 26
2.3 SEGMENTACIÓN DE IMÁGENES 27
2.3.1 Conversión del modelo de color RGB al HSV 27
2.3.2 Operaciones morfológicas en imágenes 29
2.3.2.1 Erosión 29
2.3.2.2 Dilatación 30
2.4 CARACTERIZACIÓN DE IMÁGENES 30
2.5 CLASIFICACIÓN DE IMÁGENES 30
2.6 DISEÑO Y ENTRENAMIENTO DE UNA RED NEURONAL ARTIFICIAL 31
2.7 RECONOCIMIENTO DE PATRONES 31

3. EXTRACCIÓN DE CARACTERISTICAS EN IMÁGENES 32
3.1 MOMENTOS GENERALES DE IMAGEN 32
3.2 MOMENTOS INVARIANTES DE IMAGEN 34
3.2.1 Momentos invariantes a Traslaciones 34
3.2.2 Momentos invariantes a Escalas 34
3.2.3 Momentos invariantes a Traslaciones, Rotaciones y Escalas 34

4. REDES NEURONALES 38
4.1 LA NEURONA 38
4.2 NEURONAS ARTIFICIALES 39
4.2.1 Neuronas de dos estados 39
4.2.2 El perceptrón 41
4.2.3 Aprendizaje del perceptrón 43
4.3 REDES NEURONALES ARTIFICIALES 45
4.3.1 Arquitecturas de red 45
4.3.2 Entrenamiento de una red neuronal artificial 50
4.3.2.1 Algoritmo de propagación hacia atrás (Backpropagation) 50

5. IMPLEMENTACIÓN 64
5.1 HARDWARE UTILIZADO EN EL PROYECTO 65
5.1.1 Cámara de adquisición de imágenes 65
5.2 SOFTWARE UTILIZADO EN EL PROYECTO 65
5.2.1 IDE Visual C++ 65
5.2.2 Biblioteca de visión artificial OpenCV 65
5.3 DESARROLLO OBJETIVO ESPECÍFICO 1 66
5.3.1 Adquisición y procesamiento de imágenes 66
5.3.2 Adecuación del ambiente de trabajo 67
5.3.3 Modelo de color para manipular las imágenes 68
5.4 DESARROLLO OBJETIVO ESPECÍFICO 2 70
5.4.1 Segmentación de imágenes 70
5.4.2 Mejora de las imágenes por medio de operaciones morfológicas 70
5.4.3 Creación de la base de datos de imágenes 72
5.5 DESARROLLO OBJETIVO ESPECÍFICO 3 73
5.5.1 Extracción de características de imágenes 73
5.5.2 Creación de la base de datos de patrones 76
5.5.3 Clasificación de imágenes 76
5.5.4 Diseño de la Red Neuronal Artificial 79
5.5.5 Entrenamiento de la Red Neuronal Artificial 83

6. RESULTADOS 89

7. CONCLUSIONES 96

8. RECOMENDACIONES 97

BIBLIOGRAFIA 98

ANEXOS 100

LISTA DE CUADROS

pág.

Cuadro 1.1 Aplicaciones de la visión artificial 20

Cuadro 3.1 Momentos invariantes de HU para las versiones de la letra “L” 36
Cuadro 3.2 Momentos invariantes de HU para las versiones del símbolo “petasti” 37

Cuadro 4.1 Estados de entrada y salida para neurona de McCulloch-Pitts con umbral de 1 40
Cuadro 4.2 Estados de entrada y salida para neurona de McCulloch-Pitts con umbral de 2 41
Cuadro 4.3 Modelo, aprendizaje y tarea de las diferentes redes neuronales 49

LISTA DE FIGURAS

pág.

Figura 1.1 Esquema de un sistema de Visión Artificial 22

Figura 2.1 Funcionamiento interno de una cámara 23
Figura 2.2 Modelo de color RGB 24
Figura 2.3 Modelo de color HSL 25
Figura 2.4 Modelo de color HSB 26
Figura 2.5 Modelo de color CMYK 27
Figura 2.6 Ejemplo de erosión de una imagen 29
Figura 2.7 Ejemplo de erosión de una imagen 30

Figura 3.1 Letra “L” expresada en el Lenguaje de Señas Americano 35
Figura 3.2 Símbolo bizantino “petasti” en varias escalas y versiones giradas 36

Figura 4.1 Estructura de una neurona biológica 39
Figura 4.2 Neurona sencilla McCulloch-Pitts OR inclusivo 40
Figura 4.3 Flujo de señales del perceptrón 42
Figura 4.4 Clasificación del perceptrón ilustrada en un plano 43
Figura 4.5 Esquema de conexiones de un Perceptrón sencillo 45
Figura 4.6 Red Feedforward o acíclica con una sola capa de neuronas 46
Figura 4.7 Conexión de una red feedforward o acíclica con una capa oculta y una capa de salida 47
Figura 4.8 Red recurrente sin lazo de auto realimentación y sin neuronas ocultas 48
Figura 4.9 Red recurrente con neuronas ocultas 48
Figura 4.10 Clasificación de los tipos de redes neuronales artificiales 50
Figura 4.11 Función sigmoidea con diferentes valores de pendiente 51
Figura 4.12 Red neuronal artificial multicapa 52
Figura 4.13 Composición de una neurona artificial 53
Figura 4.14 Flujo hacia adelante de una RNA multicapa (primera capa) 54
Figura 4.15 Flujo hacia adelante de una RNA multicapa (segunda capa) 55
Figura 4.16 Flujo hacia adelante de una RNA multicapa (respuesta de la red) 56
Figura 4.17 Flujo hacia atrás de una RNA multicapa (propagación de error) 56
Figura 4.18 Flujo hacia atrás de una RNA multicapa (propagación de error en segunda capa) 57
Figura 4.19 Flujo hacia atrás de una RNA multicapa (propagación de error en primera capa) 58
Figura 4.20 Actualización de pesos sinápticos de una RNA multicapa (primera capa) 60
Figura 4.21 Actualización de pesos sinápticos de una RNA multicapa (segunda capa) 62
Figura 4.22 Actualización de pesos sinápticos de una RNA multicapa (capa de salida) 63

Figura 5.1 Imagen adquirida en espacio de color RGB con recuadro 64
Figura 5.2 Cámara web Startec HC-328 65
Figura 5.3 Imagen adquirida en espacio de color RGB con recuadro 67
Figura 5.4 Sistema de iluminación posterior (Backlight) 68
Figura 5.5 Imagen adquirida en espacio de color RGB 69
Figura 5.6 Imagen adquirida en espacio de color HSV 69
Figura 5.7 Objeto en modelo HSV y binario 70
Figura 5.8 Elemento estructurante para las operaciones morfológicas 71
Figura 5.9 Primera operación morfológica a la imagen segmentada (Erosión) 71
Figura 5.10 Segunda operación morfológica a la imagen segmentada (Dilatación) 72
Figura 5.11 Tercera operación morfológica a la imagen segmentada (Dilatación) 72
Figura 5.12 Creación de la base de datos de imágenes 73
Figura 5.13 Imagen segmentada y contorno de imagen 75
Figura 5.14 Siete momentos invariantes de HU 75
Figura 5.15 Momentos de HU obtenidos de la letra A 76
Figura 5.16 Alfabeto de señas Americano (ASL) 77
Figura 5.17 Dispersión de los patrones del alfabeto ASL 78
Figura 5.18 Dispersión de los patrones numéricos del alfabeto ASL 78
Figura 5.19 Dispersión de los patrones gestuales ASL a reconocer 79
Figura 5.20 Interpretación geométrica del rol de las capas ocultas en un espacio bidimensional 80
Figura 5.21 RNA implementada 81
Figura 5.22 Entrenamiento RNA (Aciertos) 86
Figura 5.23 Entrenamiento RNA (Fracasos) 87
Figura 5.23 Entrenamiento RNA (Fracasos) 88

Figura 6.1 Dispersión de los patrones gestuales ASL 89
Figura 6.2 Reconocimiento gestual letra A 90
Figura 6.3 Reconocimiento gestual letra C 90
Figura 6.4 Reconocimiento gestual letra E 91
Figura 6.5 Reconocimiento gestual letra L 91
Figura 6.6 Reconocimiento gestual letra S 92
Figura 6.7 Reconocimiento gestual letra T 92
Figura 6.8 Reconocimiento gestual letra W 93
Figura 6.9 Reconocimiento gestual número 3 93
Figura 6.10 Reconocimiento gestual número 5 94
Figura 6.11 Reconocimiento gestual número 7 94
Figura 6.12 Reconocimiento gestual número 8 95

LISTA DE ANEXOS

pág.

Anexo A. CÓDIGO EN LENGUAJE C++ PARA CAPTURAR Y GUARDAR IMÁGENES 101

Anexo B. CÓDIGO EN LENGUAJE C++ PARA EXTRAER LOS MOMENTOS DE HU DE LA BASE DE DATOS 104

Anexo C. CÓDIGO EN LENGUAJE C++ PARA ENTRENAR LA RED NEURONAL ARTIFICIAL 110

Anexo D. CÓDIGO EN LENGUAJE C++ PARA EL RECONOCIMIENTO DE GESTOS ASL 125

GLOSARIO

ALFABETO: conjunto de los símbolos empleados en un sistema de comunicación.

CLASE DE PATRONES: una clase de patrones es un conjunto de patrones que comparten algunas propiedades.

GESTO: movimiento del rostro, de las manos o de otras partes del cuerpo con los que se expresan diversos afectos del ánimo.

IMAGEN: arreglo bidimensional de píxeles con diferente intensidad luminosa.

INTELIGENCIA: capacidad de entender o comprender.

INTELIGENCIA ARTIFICIAL: desarrollo y utilización de ordenes con los que se intenta reproducir los procesos de la inteligencia humana.

LENGUA DE SEÑAS: la lengua de señas, o lengua de signos, es una lengua natural de expresión y configuración gesto-espacial y percepción visual (o incluso táctil por ciertas personas con sordo ceguera), gracias a la cual las personas sordas pueden establecer un canal de comunicación con su entorno social, ya sea conformado por otros individuos sordos o por cualquier persona que conozca la lengua de señas empleada. Mientras que con el lenguaje oral la comunicación se establece en un canal vocal-auditivo, el lenguaje de señas lo hace por un canal gesto-viso-espacial.

LENGUAJE: un lenguaje (del provenzal *lenguatge* y este del latín *lingua*) es un sistema de comunicación estructurado para el que existe un contexto de uso y ciertos principios combinatorios formales. Existen contextos tanto naturales como artificiales.

Desde un punto de vista más amplio, el lenguaje indica una característica común al hombre y a los animales, para expresar sus experiencias y comunicarlas a otros mediante el uso de símbolos, señales y sonidos registrados por los órganos de los sentidos. El ser humano emplea un lenguaje complejo que expresa con secuencias sonoras y signos gráficos. Los animales, por su parte, se comunican a través de signos sonoros y corporales, que aún el hombre no ha podido descifrar, que en muchos casos distan de ser sencillos.

NEURONA: unidad morfológica y funcional del tejido nervioso altamente especializada y sin capacidad de reproducirse. Cada neurona está formada por un cuerpo celular o soma, unas prolongaciones citoplasmáticas denominadas dendritas y una prolongación axial, el axón. Las funciones del soma se relacionan con el mantenimiento metabólico y el crecimiento de la propia célula. Las dendritas constituyen la parte neuronal especializada en la recepción de la excitación: reciben una señal, la transforman en un impulso nervioso y la conducen hasta el axón; éste es el responsable de la conducción de la excitación en dirección opuesta a la zona

dendrítica, es decir, hasta el extremo terminal del axón. Las neuronas se pueden clasificar en motoras o sensitivas; las motoras o eferentes conducen el impulso nervioso desde el encéfalo hasta los músculos, lugar donde la corriente nerviosa se traduce en movimiento, y las sensitivas o aferentes que conducen el impulso desde la periferia al sistema nervioso central y permiten la relación del organismo con el ambiente exterior.

PATRÓN: un patrón es una descripción cuantitativa o estructural de un objeto o alguna entidad de interés.

PERCEPTRÓN: fue la primera red neural que causó un impacto significativo sobre una gran comunidad, particularmente entre los ingenieros, el perceptrón fue propuesto por Frank Rosenblatt en 1958 en un reporte técnico ampliamente distribuido por el Laboratorio Aeronáutico de Cornell.

PÍXEL: elemento básico de una imagen (picture element).

RED NEURONAL ARTIFICIAL: las redes neuronales artificiales son sistemas ideados como abstracciones de las estructuras neurobiológicas (cerebros) encontradas en la naturaleza y tienen la característica de ser sistemas desordenados capaces de guardar información. Se trata de un sistema de interconexión de neuronas que colaboran entre sí para producir un estímulo de salida.

SEÑAL: una señal es un elemento al que se le ha asignado un significado arbitrario; es decir, significa lo que hemos decidido o acordado que signifiquen, pero igualmente podría significar otra cosa. Es una especie de signo como un gesto u otro tipo que nos informa o nos avisa de algo. Sustituye por lo tanto a la palabra escrita y obedece, como todo signo, a una convención, de manera que resulta fácilmente interpretada.

SIGNO: es una unidad capaz de transmitir contenidos representativos, como el signo lingüístico. Un signo, en términos generales, es todo aquello que sirve para transmitir una información. Pensemos en una balanza, que lo asociamos a una información específica (instrumento que sirve para pesar) llamada significado, y el objeto material, de la realidad que percibimos por medio de los sentidos, denominado referente.

Pero hay también signos no lingüísticos que se caracterizan por emitir significados no precisamente con la voz o la escritura, sino con otros medios como: el empleado por los sordomudos valiéndose de las manos; el código Morse, a partir de puntos y rayas, utilizado en la comunicación telegráfica; el sistema Braille, manejado por los ciegos, cuyos signos se dibujan en relieve para que puedan percibirse a través del tacto.

SÍMBOLO: cuando un signo no sólo informa de un significado, sino que además evoca valores y sentimientos, representando ideas abstractas de una manera metafórica o alegórica, se conoce como símbolo.

La balanza, se ha utilizado desde la antigüedad como símbolo de la justicia y del derecho, dado que representaba la medición a través de la cual se podía dar a cada uno lo que es justo y necesario.

Podemos decir, entonces, que un símbolo es la representación perceptible de una idea, con rasgos asociados por una convención socialmente aceptada.

SINTAXIS: parte de la gramática que enseña a coordinar y unir las palabras para formar las oraciones y expresar conceptos.

VISIÓN POR COMPUTADORA: consiste en la adquisición, procesamiento, clasificación y reconocimiento de imágenes digitales.

RESUMEN

En este proyecto se logra hacer el reconocimiento gestual de una serie de señas que forman parte del alfabeto del Lenguaje de Señas Americano (ASL), usando la teoría de momentos de imagen y el aprendizaje de las Redes Neuronales Artificiales (RNA).

El proyecto se realiza teniendo en cuenta un esquema de visión artificial, el cual consiste en siete etapas: adquisición de la imagen, pre-procesamiento de la imagen, segmentación, extracción de características, clasificación, diseño y entrenamiento de la red neuronal artificial y reconocimiento del gesto.

Para el desarrollo de este proyecto se tomó como apoyo la biblioteca de visión artificial OpenCV versión 2.4.4 para Windows, suministrada de manera gratuita por la Corporación Intel.

Palabras clave:

- Reconocimiento de imágenes.
- Visión Artificial.
- Momentos de imagen.
- Redes Neuronales Artificiales.

INTRODUCCIÓN

La curiosidad del ser humano y su esfuerzo por crear mecanismos con capacidades similares a las suyas, han llevado al desarrollo constante de todo tipo de tecnologías enfocadas a mejorar la calidad de vida de las personas, en este caso esta tecnología es la inteligencia artificial, y en específico uno de sus campos, el de la visión artificial.

Uno de los objetivos fundamentales en el que se centra el desarrollo del proyecto, y que también forma parte de los propósitos de la visión artificial es el reconocimiento de imágenes, con lo que se incluye la adquisición, segmentación y posteriormente su reconocimiento, pero la interpretación de los datos adquiridos por medio de los sensores (cámara), es uno de los problemas más significativos que se presentan en el desarrollo del proyecto, y es por esto que se hace uso de operaciones y métodos matemáticos para solucionar este inconveniente.

Este proyecto busca por medio de la visión por computadora reconocer patrones gestuales del Lenguaje de Señas Americano, por eso el primer paso es adquirir la imagen de la cual se va a hacer el reconocimiento, debido a que las imágenes no siempre se van a presentar de manera óptima para el análisis, se debe hace un pre-procesamiento para eliminar ruido de la misma, después de esto se realiza la segmentación de las partes que contienen información relevante para su posterior uso, en este proceso de adecuación y segmentación de la imagen se hace uso de filtros digitales, transformación de espacios de color y operaciones morfológicas. Una vez se tiene la imagen con la que se va a trabajar (imagen segmentada), se procede a extraer las características propias que la diferencian de otras, por medio del método de momentos de imagen, y así de este modo poderla identificar.

La clasificación de los diferentes tipos de gestos presentados en las imágenes será resuelta, por medio de la implementación de una red neuronal artificial, cuyo entrenamiento se dirigirá a reconocer patrones del alfabeto ASL, que sean linealmente separables y suficientemente separados entre sí.

OBJETIVOS

Objetivo General

Desarrollar e implementar una aplicación en software para reconocer las señas estáticas del alfabeto del Lenguaje de Señas Americano (ASL).

Objetivos Específicos

- Diseñar el algoritmo de pre procesamiento de las imágenes para reducción de ruido.

- Construir el algoritmo de segmentación para extraer el gesto de la imagen.

- Implementar por medio de Redes Neuronales Artificiales la máquina de reconocimiento.

1. VISIÓN ARTIFICIAL

1.1 DEFINICIÓN DE INTELIGENCIA ARTIFICIAL

Para entender de una manera adecuada el término de inteligencia artificial, hay que definir primero, el término inteligencia. La Real Academia de la Lengua Española define la inteligencia como “Potencia intelectual: facultad de conocer, de entender o comprender”. Una vez definido esto, la inteligencia artificial se podría definir de la misma forma pero aplicado a las máquinas. Otra definición válida y más cercana a la realidad la propuso Marvin Minsky, uno de los pioneros de la Inteligencia Artificial (IA), quién dijo: “La Inteligencia Artificial es la ciencia de construir máquinas para que hagan cosas que, si las hicieran los humanos, requerirían inteligencia”.

1.1.1 Áreas de aplicación de la inteligencia artificial

La IA es aplicada en diversas áreas, de manera que pueda ayudar a solucionar problemas en la cotidianidad de los seres humano. A continuación se presentan algunas de ellas:

- **Tratamiento de lenguajes naturales** En este campo se puede englobar aplicaciones que realicen traducciones entre idiomas, interfaces hombre-máquina que permita interrogar una base de datos o dar órdenes a un sistema operativo, etc…, de manera que la comunicación sea más amigable al usuario.
- **Sistemas expertos** En esta área están englobados aquellos sistemas donde la experiencia de personal cualificado se incorpora a dichos sistemas para conseguir deducciones más cercanas a la realidad.
- **Robótica** Navegación de robots móviles, control de brazos de robots, ensamblaje de piezas, etc.
- **Problemas de percepción: visión y habla** Reconocimiento de objetos y del habla, detección de defectos en piezas por medio de visión, apoyo en diagnósticos médicos, etc.
- **Aprendizaje** Modelización de conductas para su posterior implantación en computadoras.

1.2 BASES DE LA INTELIGENCIA ARTIFICIAL MODERNA

Los primeros esfuerzos en el estudio de la simulación automática de la inteligencia tuvieron lugar en el período de 1945 a 1956. Durante este período las diferencias fundamentales entre el funcionamiento del cerebro humano, los sistemas de retro-alimentación (*feedback*) y los computadores digitales no estaban aún bien definidas. En 1943 Wiener, en colaboración con Rosenblueth y Bigelow, establece las bases de la Cibernética (control, autorregulación) integrando aproximaciones mecánicas

(automatismos), biológicas (regulación natural), fisiológicas (neuronas), formales (Lógica) y de procesamiento de información. McCulloch y Pitts explican formalmente la realización de operaciones lógicas mediante el interconexionado neural y su control por realimentación, demostrando que cualquier ley de entrada/salida podría implementarse con una red neuronal. Seis años más tarde, en 1949, Hebb sugerirá mecanismos de aprendizaje. Acababa de nacer el paradigma conexionista. En 1950 Turing aborda desde otra perspectiva la cuestión: "¿Puede una máquina pensar?". Turing propone un test (bautizado con su nombre) de caracterización basado en la capacidad de emular el comportamiento humano en el llamado juego de imitación. El test de Turing se define de la siguiente manera:

> "Disponemos a un humano y a una máquina en habitaciones diferentes. Un observador les hace una serie de preguntas a uno y a otro a través de la puerta. Si pasado un cierto tiempo el observador no es capaz de determinar quién es el humano y quién la máquina, podemos concluir diciendo que la máquina posee inteligencia."

1.3 SISTEMAS DE VISIÓN ARTIFICIAL

La visión artificial (VA) es un campo de la inteligencia artificial, que con la ayuda de una serie de elementos y técnicas permite la obtención de imágenes, las cuales pueden ser manipuladas y analizadas con el fin de extraer información relevante para la toma de decisiones.

1.3.1 Definición técnica de visión artificial

La visión artificial consiste en la captación de imágenes en línea mediante cámaras y su posterior tratamiento a través de técnicas de procesamiento avanzadas, permitiendo así poder intervenir sobre un proceso (modificación de variables del mismo) o producto (detección de unidades defectuosas), para el control de calidad y seguridad de toda la producción.

1.3.2 Objetivos de los sistemas visión artificial

Los sistemas de VA tienen generalmente como objetivos los siguientes:

Automatizar tareas repetitivas de inspección realizadas por operadores.

- Realizar controles de calidad de productos que no era posible verificar por métodos tradicionales.
- Realizar inspecciones de objetos sin contacto físico.
- Realizar la inspección del 100% de la producción (calidad total) a gran velocidad.

- Reducir el tiempo de ciclo en procesos automatizados.
- Realizar inspecciones en procesos donde existe diversidad de piezas con cambios frecuentes de producción.
- Reconocimiento de ciertos objetos en imágenes (por ejemplo, caras humanas).
- Seguimiento de un objeto en una secuencia de imágenes.

1.3.3 Aplicaciones de los sistemas visión artificial

Las aplicaciones de Visión Artificial se dividen en tres grandes categorías:

- Control de procesos
- Control de calidad
- Aplicaciones no industriales (por ejemplo, control del tráfico)

Algunas aplicaciones se muestran a continuación:

Cuadro 1.1 Aplicaciones de la visión artificial

Área de producción	Aplicación
Control de calidad	Inspección de productos (papel, aluminio, acero,...)
	Identificación de piezas
	Etiquetados (fechas de caducidad,...)
	Inspección de circuitos impresos
	Control de calidad de los alimentos (naranjas,...)
Robótica	Control de soldaduras
	Guiado de robots (vehículos no tripulados)
Biomédicas	Análisis de imágenes de microscopía (virus, células, proteínas)
	Resonancias magnéticas, tomografías, genoma humano
Astronomía	Exploración del espacio
Reconocimiento de caracteres	Control de cheques, inspección de textos, ...

Cuadro 1.1 (Continuación)

Área de producción	Aplicación
Control de tráfico	Matrículas de coches
	Tráfico viario
Meteorología	Predicción del tiempo
Agricultura	Interpretación de fotografías aéreas
	Control de plantaciones
Militares	Seguimiento de objetivos
	Vigilancia por satélites

Fuente: PLATERO, Carlos. Apuntes de Visión Artificial. Madrid (España). Disponible en internet: http://www.elai.upm.es/webantigua/spain/Asignaturas/MIP_VisionArtificial/ApuntesVA/cap1IntroVA.pdf.

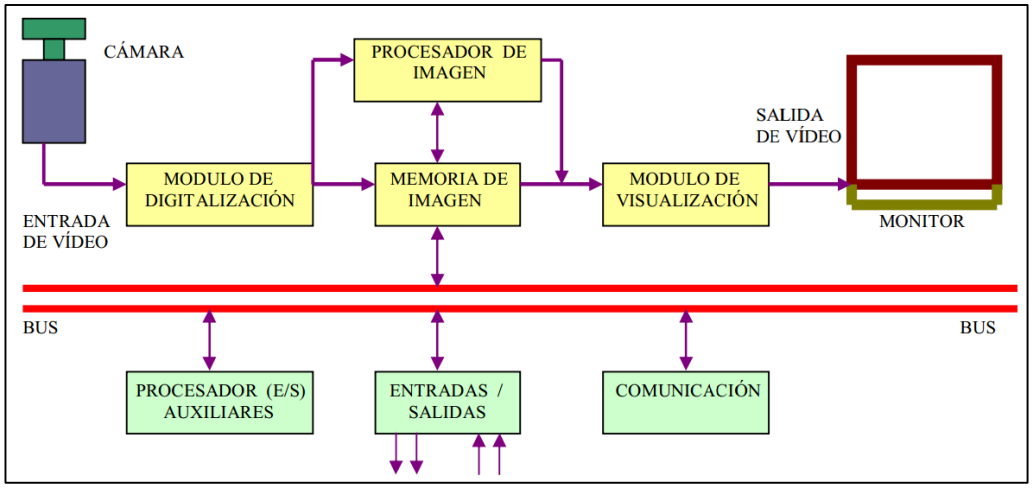
1.3.4 Diagrama de bloques de un sistema de visión artificial

A continuación se muestra un diagrama de bloques de un sistema general de visión artificial, donde se identificarán las fases que intervienen en los procesos de presentación de las imágenes adquiridas.

Un sistema de visión artificial:

- Capta una imagen de un objeto real
- La convierte en formato digital
- La procesa mediante un ordenador
- Obtiene unos resultados del proceso

Figura 1.1 Esquema de un sistema de Visión Artificial



Fuente: C.I.P. ETI de Tudela. Visión Artificial. Tudela (España). Disponible en internet: http://www.etitudela.com/celula/downloads/visionartificial.pdf.

Módulo de digitalización. Convierte la señal analógica proporcionada por la cámara a una señal digital (para su posterior procesamiento).

Memoria de imagen. Almacena la señal procedente del módulo de digitalización.

Módulo de visualización. Convierte la señal digital residente en memoria, en señal de vídeo analógica para poder ser visualizada en el monitor de TV.

Procesador de imagen. Procesa e interpreta las imágenes captadas por la cámara.

Módulo de entradas/salidas. Gestiona la entrada de sincronismo de captación de imagen y las salidas de control que actúan sobre dispositivos externos en función del resultado de la inspección.

Comunicaciones. Vía I/O, Ethernet, RS232 (la más estándar).

2. PROCESOS DE UN SISTEMA DE VISIÓN ARTIFICIAL

Un sistema de visión artificial está compuesto de fases o etapas, las cuales al llevarse a cabo cumplen el objetivo de reconocer o resaltar las características de la imagen tratada, en este proyecto, se trabajan siete etapas las cuales serán descritas a continuación:

2.1 ADQUISICIÓN DE IMÁGENES

En esta primera fase, se tendrán en cuenta una serie de objetos y cualidades de la escena para realizar una muy buena adquisición de las imágenes, esto con el fin de disminuir la cantidad de procesos a la hora de manipular la imagen digitalmente.

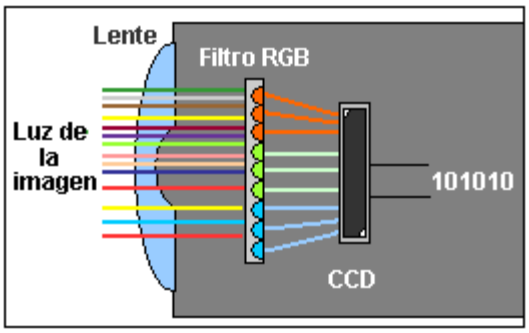
2.1.1 Cámara de adquisición de imágenes

Es el dispositivo de entrada al sistema de visión artificial, y es el encargado de capturar la escena y transmitirla por medio de un cable de datos al computador.

Este dispositivo está compuesto general y básicamente por tres elementos, un lente, un filtro RGB (Red-Green-Blue) y un sensor CCD (charge-coupled device o dispositivo de carga acoplada) o un sensor CMOS (Active Pixel Sensor APS).

El proceso es el siguiente, la luz de la imagen pasa por la lente, esta se refleja en un filtro RGB, el cuál descompone la luz en tres colores básicos: rojo, verde y azul. Esta división de rayos se concentra en un chip sensible a la luz CCD o CMOS, el cuál asigna valores binarios a cada píxel y envía los datos digitales para su codificación y posterior almacenamiento.

Figura 2.1 Funcionamiento interno de una cámara



Fuente: La cámara Web. Disponible en internet: http://www.informaticamoderna.com/Camara_web.htm

2.2 PREPROCESAMIENTO DE IMÁGENES

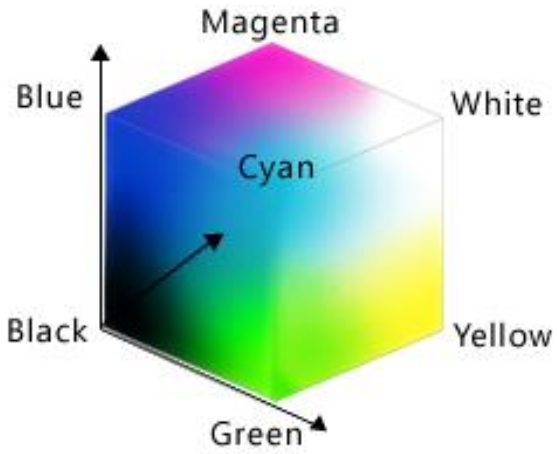
En esta etapa se manipula la imagen digital previamente adquirida, de manera que se eliminen los defectos o problemas que nos impidan una buena segmentación, estos problemas pueden ser ruido granular, iluminación, entre otros.

2.2.1 Modelos de color

Los colores puede representarse de diversas maneras, y cada una de ellas se denomina un modelo de color, a continuación se describirán de manera breve los más comunes.

2.2.1.1 Modelo RGB. Este es un modelo de color aditivo, es decir, cuanto más rojo, verde y azul se agregue, más se parecerá el color al blanco. Cuando se mezcla la misma cantidad de rojo, verde y azul, siempre se obtiene un gris neutro. Para oscurecer un color, debe quitar la misma cantidad de los tres colores. Los escáneres y los monitores se basan en el modelo de color RGB, por lo que se trata de un modelo natural para describir colores en un equipo, sobre todo cuando se trabaja con imágenes digitalizadas. El valor de cada "canal" (rojo, verde o azul) puede ir desde 0 (sin color) hasta 255 (color con la máxima saturación). El color RGB en ocasiones también se denomina color de 24 bits o millones de colores.

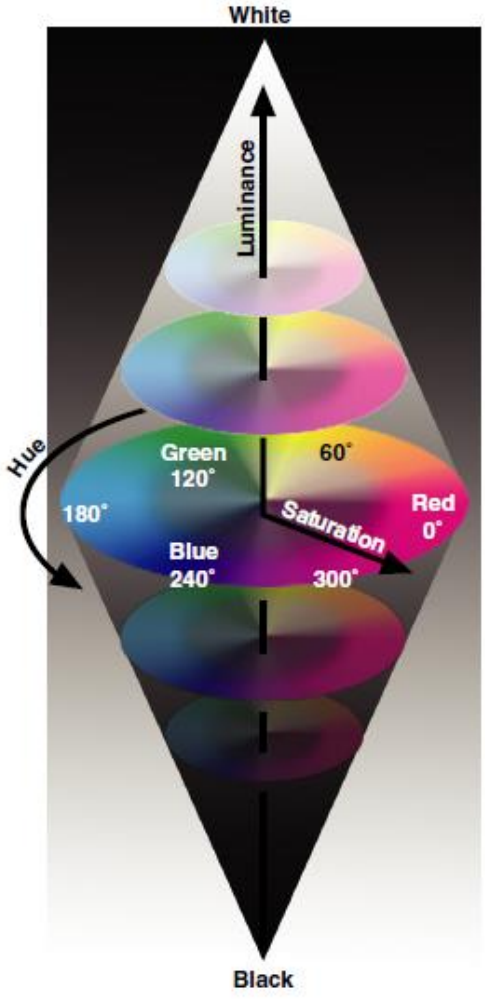
Figura 2.2 Modelo de color RGB



Fuente: MICROSOFT CORPORATION. Disponible en internet: http://msdn.microsoft.com/en-us/library/windows/desktop/aa511283.aspx.

2.2.1.2 Modelo HLS. Muchas personas consideran el modelo de color HLS más intuitivo porque define los colores en función del matiz, la claridad (o luminosidad) y la saturación. Para especificar un color, puede seleccionar su matiz en un espectro de arco iris, seleccionar su saturación (la pureza del color) y establecer su luminosidad (de claro a oscuro). El rojo vivo es un color brillante muy saturado. Los tonos pastel, como el rosa claro, son menos saturados. El matiz se especifica en grados (de 0 a 360 grados) y la saturación y la claridad se especifican en porcentajes de 0 hasta 100 por ciento. Todo color HLS con cero saturación es un gris neutro.

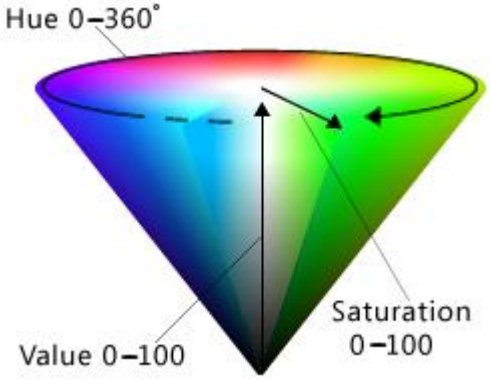
Figura 2.3 Modelo de color HSL



Fuente: NATIONAL INSTRUMENTS. Disponible en internet: http://zone.ni.com/reference/en-XX/help/372916P-01/nivisionconcepts/color_spectrum/.

2.2.1.3 Modelo HSB o HSV. El modelo de color HSB o también llamado HSV, es muy similar al HLS, excepto en que se usa el valor B (luminosidad) del color en vez de la claridad.

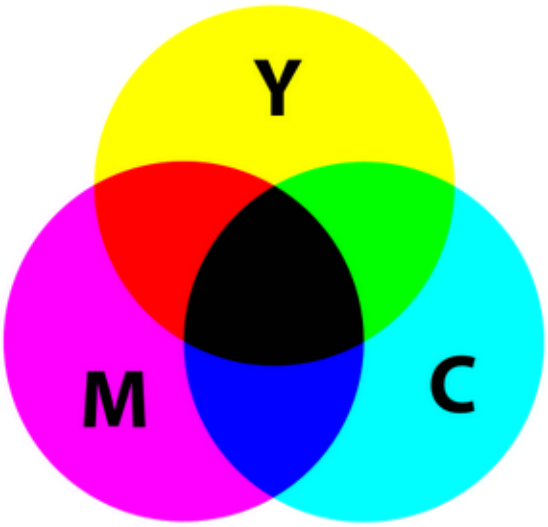
Figura 2.4 Modelo de color HSB



Fuente: MICROSOFT CORPORATION. Disponible en internet: http://msdn.microsoft.com/en-us/library/windows/desktop/aa511283.aspx.

2.2.1.4 Modelo CMYK. La impresión realizada por cada impresora y máquina de imprenta se basa en los tres colores primarios: cian, magenta y amarillo. Éstos se denominan colores sustractivos porque cuanto más color se agrega, más se acerca al negro. Desafortunadamente, las tintas en el mundo real no son nunca perfectas y es necesario agregar negro (K) para lograr una gama más completa de colores impresos. El modelo CMYK es adecuado principalmente para la creación de imágenes que se van a imprimir. Observe que muchas combinaciones distintas de cian, magenta, amarillo y negro pueden definir el mismo color. La definición de un gris neutro en CMYK puede resultar complicada (por ejemplo, siempre se suele necesitar más cian que los otros colores) y variará en función del dispositivo de salida.

Figura 2.5 Modelo de color CMYK



Fuente: DESKSHARE. Disponible en internet:
http://www.deskshare.com/lang/sp/help/fp/ColorBasics.aspx.

2.3 SEGMENTACIÓN DE IMÁGENES

La segmentación es el proceso en el cual una imagen es separada o dividida en grupos de pixeles, esto se hace con el fin de separar los aspectos más significativos de la imagen para poder ser analizada de forma más sencilla.

Hay diversas maneras de segmentar imágenes, entre ellas está la segmentación por características, las cuales incluye la segmentación por niveles de gris, por color o texturas. En la segmentación basada en transiciones se encuentran la detección de puntos, líneas y bordes. En la segmentación basada en modelos podemos destacar la transformada de Hough, entre otros tipos de segmentación.

A continuación, se describirá la conversión del modelo de color RGB a HSV, la cual fue usada en el proyecto para la segmentación por color.

2.3.1 Conversión del modelo de color RGB al HSV

Los modelos de color actuales están diseñados para ser usados en ciertas aplicaciones, dado a su mejor desenvolvimiento en ciertas áreas, como en el caso del modelo RGB, el cual está orientado al hardware y se ve aplicado con regularidad en televisores, monitores, etc. Por otro lado, el modelo de color HSV está orientado al usuario, es una representación cercana a la forma en que los humanos percibimos el color y tiene la característica de agrupar las tonalidades de color, por esta razón es usado en este proyecto.

Para transformar las coordenadas RGB a HSV primero se dividen cada uno de los valores de Rojo, Verde y Azul en 255, esto con el fin de cambiar el rango de 0 a 255 a uno nuevo de 0 a 1.

$$R' = R/255$$

$$G' = G/255$$

$$B' = \frac{B}{255}$$

$$C_{max} = \max(R', G', B')$$

$$C_{min} = \min(R', G', B')$$

$$\Delta = C_{max} - C_{min}$$

- Matiz (Hue):

  Si C_{max} = R'

$$H = 60° * \left(\frac{G' - B'}{\Delta}\right) mod\ 6$$

  Si C_{max} = G'

$$H = 60° * \left(\frac{B' - R'}{\Delta}\right) + 2$$

  Si C_{max} = B'

$$H = 60° * \left(\frac{R' - G}{\Delta}\right) + 4$$

- Saturación (Saturation):

  Si Δ = 0

$$S = 0$$

  Si Δ <> 0

$$S = \frac{\Delta}{C_{max}}$$

- Valor (Value):

$$V = C_{max}$$

2.3.2 Operaciones morfológicas en imágenes

Las operaciones morfológicas son métodos usados en el procesamiento de las imágenes binarias, su operación es hecha de pixel a pixel, y es basado sobre formas. Su función es la de mejorar la imagen segmentada, eliminando posibles puntos o zonas no deseadas, definiendo de mejor manera los contornos contenientes de la información.

Existen dos operaciones morfológicas básicas: erosión y dilatación (usadas en el proyecto); combinándolas se obtienen nuevas operaciones más complejas, como la apertura y el cierre.

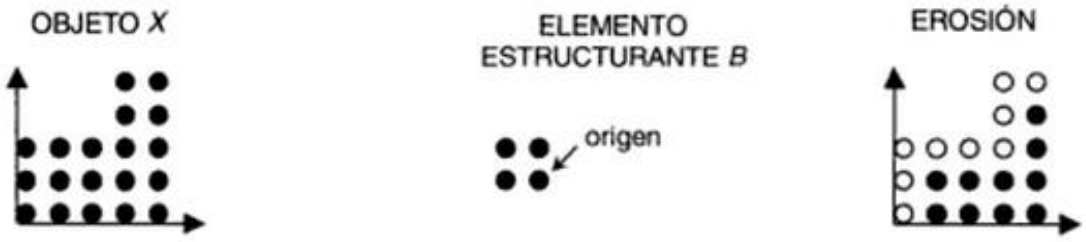
Llamando al objeto a analizar *X*, al elemento estructurarte *B*, y B_x la traslación de *B* de forma que su origen esté ubicado en *x*, se pueden definir las siguientes operaciones básicas:

2.3.2.1 Erosión.

La erosión de *X* realizada con *B* se define como el conjunto de todos los puntos *x* tales que B_x está incluido en *X*:

$$\varepsilon_B(X) = \{x: B_x \subset X\}$$

Figura 2.6 Ejemplo de erosión de una imagen



Fuente: FAÚNDEZ ZANUY, Marcos. TRATAMIENTO DIGITAL DE VOZ E IMAGEN Y APLICACIÓN A LA MULTIMEDIA, 2000, p.173

2.3.2.2 Dilatación.

La dilatación de *X* realizada con *B* se define como el conjunto de todos los puntos *x* tales que B_x intersecciona con *X* por lo menos en un punto:

$$\delta_B(X) = \{x: B_x \cap X \neq 0\}$$

Figura 2.7 Ejemplo de erosión de una imagen

Fuente: FAÚNDEZ ZANUY, Marcos. TRATAMIENTO DIGITAL DE VOZ E IMAGEN Y APLICACIÓN A LA MULTIMEDIA, 2000, p.173

2.4 CARACTERIZACIÓN DE IMÁGENES

La caracterización de las imágenes es un paso muy importante el sistema de visión artificial, y solo puede hacerse después de un buen procesamiento y segmentación de ésta. El objetivo de éste es el de individualizar cada una de las imágenes, dándole características representativas, con el fin de poderla diferenciar de otras. Hay diferentes modelos de caracterización de imágenes, y el usado en el proyecto es el de momentos invariantes, el cual será descrito más adelante.

2.5 CLASIFICACIÓN DE IMÁGENES

En este proceso de identificación de patrones, es necesario tener una perspectiva de cuáles serán los patrones a reconocer, por tanto, la organización de estos es expuesta en un sistema de coordenadas (coordenadas cartesianas de dos dimensiones), para poder identificar los patrones con posibilidad de ser reconocidos. Esta información podrá verse en el capítulo de implementación.

2.6 DISEÑO Y ENTRENAMIENTO DE UNA RED NEURONAL ARTIFICIAL

Después de haber cumplido todas las etapas descritas anteriormente, es hora de entrar en la parte del diseño y entrenamiento de la red neuronal artificial, ya que es ésta la que reconocerá los gestos presentes en las imágenes adquiridas.

Para poder diseñar una RNA, hay que tener en cuenta los diversos modelos existentes y sus características más importantes.

Después de haber tomado una decisión se procede a montar la estructura de la red. Hay que tener en cuenta, que no existen ecuaciones que ayuden a determinar el número de neuronas a usar, y es por esto que el proceso se hará de manera experimental, partiendo de que el número de neuronas en la capa de entrada corresponderá a la cantidad de características extraídas de la imagen.

Dado a la importancia de este tema en el proyecto, se dedicará un capitulo para su explicación.

2.7 RECONOCIMIENTO DE PATRONES

Para finalizar las etapas del sistema de visión artificial, se encuentra el reconocimiento del gesto. Esta parte va de la mano con la del entrenamiento de la RNA, aquí el sistema se encargará de identificar el gesto presentado, de acuerdo a los gestos usados en el entrenamiento de la red.

3. EXTRACCIÓN DE CARACTERISTICAS EN IMÁGENES

Para poder hacer el reconocimiento de una imagen, hay que primero haberla descrito, esto consiste en extraer sus características, con el fin de poderla diferenciar de otras imágenes.

Los descriptores deben de ser independientes del tamaño, localización u orientación del objeto, y deben ser suficientes para discriminar objetos entre sí.

Los descriptores se basan en la evaluación de alguna característica del objeto, por ejemplo:

a. Descriptores unidimensionales: códigos de cadena, perímetro, forma del perímetro.
b. Descriptores bidimensionales: área, momentos de inercia, etc.
c. Descriptores específicos: número de agujeros, área de agujeros, posición relativa de agujeros, rasgos diferenciadores de un objeto, etc.

En el desarrollo del proyecto se usaron descriptores bidimensionales, los cuales serán descritos a continuación.

3.1 MOMENTOS GENERALES DE IMAGEN

La geometría de una región plana se basa en el tamaño, la posición, la orientación y la forma. Todas estas medidas están relacionadas con una familia de parámetros llamadas momentos.

La teoría de los momentos proporciona una interesante y útil alternativa para la representación de formas de objetos. Si se tiene un objeto en una región que viene dado por los puntos en los que $f(x,y) > 0$, se define el momento de orden p,q como:

$$m_{pq} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} x^p y^q \, f(x,y) \, dx \, dy$$

El interés de estos momentos generales desde el punto de vista de la caracterización discriminante de los contornos de los objetos, es que, estos contornos pueden modelarse como un tipo especial de funciones $f(x,y)$ acotadas y por ende se pueden calcular los momentos generales.

Aquí radica la aplicabilidad de los momentos en el reconocimiento de formas, dada una función acotada $f(x,y)$ existe un conjunto de momentos generales y viceversa.

Es decir, dado un conjunto de momentos generales se puede reconstruir una función $f(x,y)$ única, simbólicamente:

$$f(x,y) \leftrightarrow \{m_{pq}\} \qquad p,q = 0{,}1 \ldots\ldots \infty$$

Ahora, dado a que se trabajan sobre imágenes digitales (discretas), es necesario pasar de las integrales dobles a las sumatorias dobles.

Los momentos generales discretos de la función $I_o(x,y)$ serán:

$$m_{pq} = \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} x^p y^q \, I_o(x,y) \quad p,q = 0{,}1 \ldots\ldots \infty$$

Donde los píxeles que forman parte del objeto tendrán un peso de uno y los demás será de cero.

$$I_o(x,y) = 1 \qquad Si\ forma\ parte\ del\ objeto$$

$$I_o(x,y) = 0 \qquad Si\ no\ forma\ parte\ del\ objeto$$

Un momento de gran interés es el de orden cero (el orden de un momento viene dado por la suma de los índices p y q):

$$m_{00} = \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} x^p y^q \, I_o(x,y)$$

Este momento nos da como resultado el área del objeto.

Los momentos de orden uno, m_{01} y m_{10} junto con m_{00} determinan el llamado centro de gravedad del objeto:

$$\bar{x} = \frac{m_{10}}{m_{00}} = \frac{\sum\sum x \, I(x,y)}{\sum\sum I(x,y)}$$

$$\bar{y} = \frac{m_{01}}{m_{00}} = \frac{\sum\sum y \, I(x,y)}{\sum\sum I(x,y)}$$

3.2 MOMENTOS INVARIANTES DE IMAGEN

El principal inconveniente que tienen los momentos generales es que sus valores dependerán de la posición, escala y rotación del objeto.

Es por esto, que se han desarrollado modelos matemáticos que permiten corregir estas variaciones, entre los cuales tenemos los momentos invariantes a traslaciones, momentos invariantes a su escala (u homotecias) y los momentos invariantes a traslaciones, rotaciones y escalas, los cuales se describirán a continuación.

3.2.1 Momentos invariantes a Traslaciones

Estos momentos son llamados momentos centrales, ya que referencian los momentos generales al centro de gravedad del objeto para poder hacer una descripción invariante a la posición.

$$u_{pq} = \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} (x - \bar{x})^p (y - \bar{y})^q \, I_o(x, y)$$

3.2.2 Momentos invariantes a Escalas

Debido a que u_{00} es el área del objeto, se pueden normalizar los momentos centrales para obtener una descripción invariante al tamaño.

$$\eta_{pq} = \frac{u_{pq}}{u_{00}^{\propto}}$$

Donde:

$$\propto = \frac{p+q}{2} + 1$$

3.2.3 Momentos invariantes a Traslaciones, Rotaciones y Escalas

MING KUEI HU describió el método de momentos invariantes absolutos. Estas se componen de grupos de expresiones de momentos centralizados no lineales. El resultado es un conjunto de momentos invariantes ortogonales absolutos, que pueden ser utilizados para la identificación de patrones independientemente de la escala, posición, y la rotación. Se calculan a partir de los momentos centrales normalizados hasta el orden tres.

Momentos invariantes de HU:

$$I_1 = \eta_{20} + \eta_{02}$$

$$I_2 = (\eta_{20} - \eta_{02})^2 + 4\eta^2{}_{11}$$

$$I_3 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2$$

$$I_4 = (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2$$

$$I_5 = (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]$$

$$I_6 = (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2 + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03})]$$

Por último, un sesgo invariante, para ayudar a distinguir imágenes espejo, es:

$$I_7 = (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] + (\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]$$

A continuación, se presentarán dos ejemplos de imágenes en diferentes posiciones y ángulos, junto con sus respectivos momentos de HU que comprobaran su efectividad.

Figura 3.1 Letra “L” expresada en el Lenguaje de Señas Americano

Fuente: Autor.

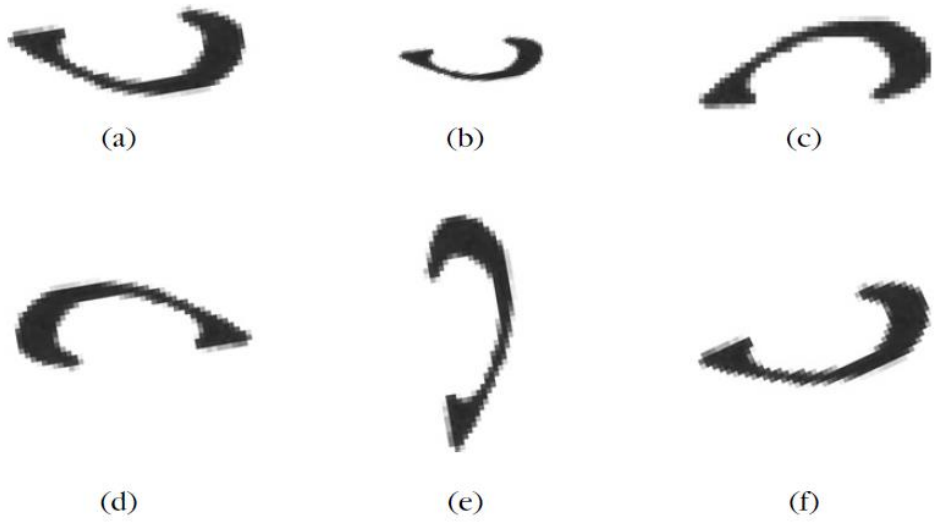
$$I_1 = \eta_{20} + \eta_{02} = 0.06735928345249488100 + 0.12790642912660383000 = 0.19526571257$$

Cuadro 3.1 Momentos invariantes de HU para las versiones de la letra “L”

Momentos	0°	Escalado	180°	15°	Espejo	90°
I_1	0,1952657	0,1951134	0,1952657	0,1954117	0,1948309	0,1952657
I_2	59,23x10^{-4}	59,12x10^{-4}	59,23x10^{-4}	60,03x10^{-4}	58,202x10^{-4}	59,23x10^{-4}
I_3	26,81x10^{-5}	26,68x10^{-5}	26,81x10^{-5}	26,13x10^{-5}	26,81x10^{-5}	26,81x10^{-5}
I_4	45,23x10^{-6}	42,65x10^{-6}	45,23 x10^{-6}	43,89x10^{-6}	42,91x10^{-6}	45,23x10^{-6}
I_5	-4,21x10^{-9}	-3,95x10^{-9}	-4,21x10^{-9}	-4,073x10^{-9}	-3,992x10^{-9}	-4,21x10^{-9}
I_6	18,25x10^{-7}	17,14x10^{-7}	18,25 x10^{-7}	17,81x10^{-7}	16,808x10^{-7}	18,25x10^{-7}
I_7	-2,64x10^{-9}	-2,25x10^{-9}	-2,64 x10^{-9}	-2,34x10^{-9}	-2,29x10^{-9}	-2,64x10^{-9}

Fuente: Autor.

Figura 3.2 Símbolo bizantino “petasti” en varias escalas y versiones giradas



Fuente: THEODORIDIS, Sergios. Pattern Recognition 4 ed. 2009, p. 426

Cuadro 3.2 Momentos invariantes de HU para las versiones del símbolo “petasti”

Momentos	0°	Escalado	180°	15°	Espejo	90°
I_1	93,13	91,76	93,13	94,28	93,13	93,13
I_2	58,13	56,6	58,13	58,59	58,13	58,13
I_3	26,7	25,06	26,7	27	26,7	26,7
I_4	15,92	14,78	15,92	15,83	15,92	15,92
I_5	3,24	2,8	3,24	3,22	3,24	3,24
I_6	10,7	9,71	10,7	10,57	10,7	10,7
I_7	0,53	0,46	0,53	0,56	-0,53	0,53

Fuente: THEODORIDIS, Sergios. Pattern Recognition 4 ed. 2009, p. 426

4. REDES NEURONALES

El uso de las redes neuronales artificiales en el reconocimiento de patrones, es uno de los métodos más usados gracias a su efectividad, ya que por medio de variados modelos de entrenamiento se puede lograr que una red aprenda valores específicos, correspondientes a los elementos que se desean reconocer.

Las Redes Neuronales Artificiales, RNA (o en inglés Artificial Neural Networks ANN) fueron originalmente una simulación abstracta de los sistemas nerviosos biológicos, formados por un conjunto de unidades llamadas "neuronas" o "nodos" conectadas unas con otras. Estas conexiones tienen una gran semejanza con las dendritas y los axones en los sistemas nerviosos biológicos.

Las RNA son herramientas usadas en la resolución de problemas de forma individual o combinadas con otros métodos, para aquellas tareas de clasificación, identificación, diagnóstico, optimización o predicción en las que el balance datos/conocimiento se inclina hacia los datos, y donde adicionalmente, puede haber la necesidad de aprendizaje en tiempo de ejecución y de cierta tolerancia a fallos. En estos casos las RNA se adaptan dinámicamente reajustando constantemente los “pesos” de sus interconexiones.

4.1 LA NEURONA

La neurona genérica esta modelada con base en las neuronas motoras o moto neuronas espinales, una de las neuronas mejor caracterizadas en los mamíferos. Las neuronas son células, y tienen un núcleo y el aparato metabólico celular relacionado.

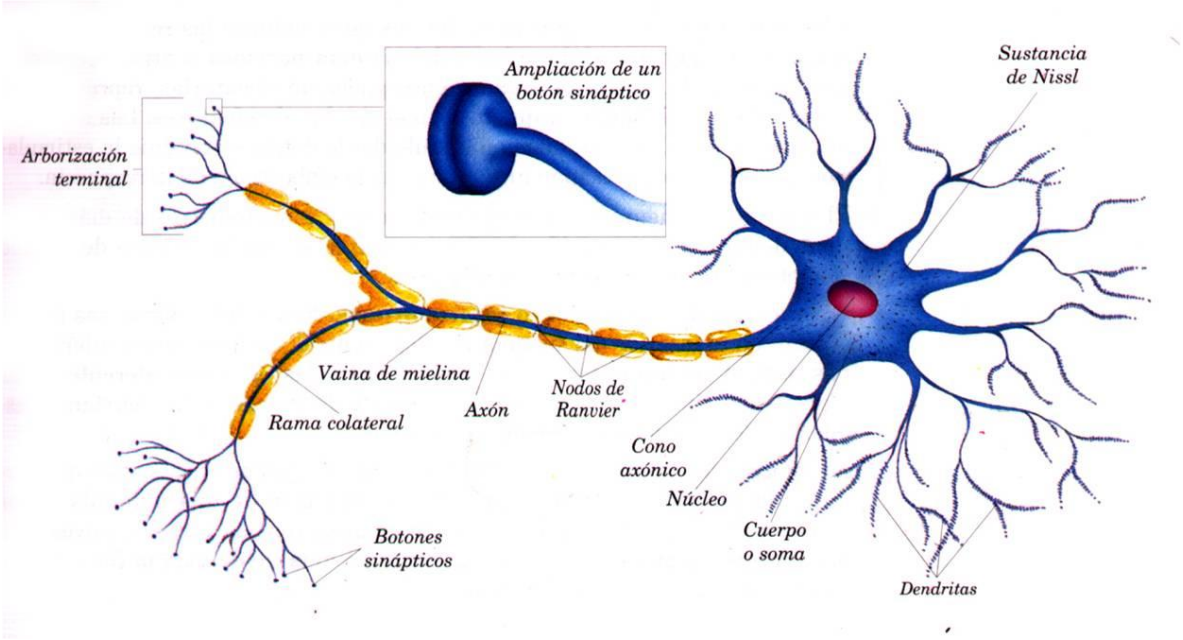
Un extremo de la célula, el de entrada, tiene un número de finas ramificaciones, llamadas dendritas, el cuerpo celular se le conoce como soma.

La mayoría de las neuronas tienen una larga y delgada ramificación, el axón, que se aparta del cuerpo celular y puede extenderse por metros. El axón es la línea de transmisión de la neurona. Cuando los axones llegan a su destino final, se ramifican nuevamente en lo que se conoce como arborización terminal. En los extremos de las ramas axonales, hay unas complejas estructuras especializadas llamadas sinapsis.

En la imagen normal de la neurona, las dendritas reciben entradas desde otras células, el soma y las dendritas procesan e integran las entradas, y la información se transmite por el axón hacia la sinapsis, cuyas salidas proporcionan entradas a

otras neuronas u órganos efectores. Las sinapsis permiten que una célula influya en la actividad de otras.

Figura 4.1 Estructura de una neurona biológica



Fuente: OdontoAyuda. Neurona. Disponible en internet: http://odontoayuda.com/etiqueta/histologia/

4.2 NEURONAS ARTIFICIALES

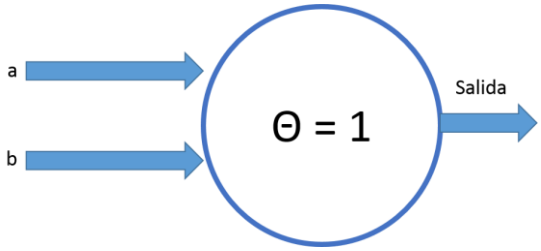
4.2.1 Neuronas de dos estados

En 1943, Warren McCulloch y Walter Pitts escribieron un famoso e influyente trabajo que estaba basado en los cómputos que podrían ser realizados por neuronas de dos estados, estos investigadores realizaron un número de suposiciones gobernando la operación de las neuronas que definen lo que se ha venido a conocer como como la neurona de McCulloch-Pitts. Este es un dispositivo binario, lo que quiere decir que solo puede estar en uno de dos estados posibles. Cada neurona tiene un umbral fijo, y recibe entradas de sinapsis excitadoras, todas las cuales tienen pesos idénticos. Las activaciones sinápticas excitadoras se suman linealmente. La neurona también puede recibir entradas de sinapsis inhibidoras,

cuya acción es absoluta; esto es, si la sinapsis inhibidora está activa, la neurona no se puede activar.

El modo de operación para el modelo de la neurona McCulloch-Pitts es simple. Durante el cuanto de tiempo (tiempo para la integración de las entradas sinápticas), la neurona responde a la actividad de sus sinapsis, que reflejan el estado de las células presinápticas. Si no hay alguna sinapsis inhibidora activa, la neurona añade sus entradas sinápticas y verifica si la suma alcanza o excede su umbral. Si lo hace, la neurona se activa. Si no lo hace, la neurona no se activa.

Figura 4.2 Neurona sencilla McCulloch-Pitts OR inclusivo



Fuente: ANDERSON, James A. Redes neurales. 2007, p.

Cuadro 4.1 Estados de entrada y salida para neurona de McCulloch-Pitts con umbral de 1

Entrada (tiempo = t)		Salida (tiempo = t+1)
a	b	
0	0	0
1	0	1
0	1	1
1	1	1

Fuente: ANDERSON, James A. Redes neurales. 2007, p.

Si se le diera a la unidad el umbral de dos, podría computar la función lógica de conjunción o AND, ya que únicamente se activaría si *a* y *b* están activas.

Cuadro 4.2 Estados de entrada y salida para neurona de McCulloch-Pitts con umbral de 2

Entrada (tiempo = t)		Salida (tiempo = t+1)
a	b	
0	0	0
1	0	0
0	1	0
1	1	1

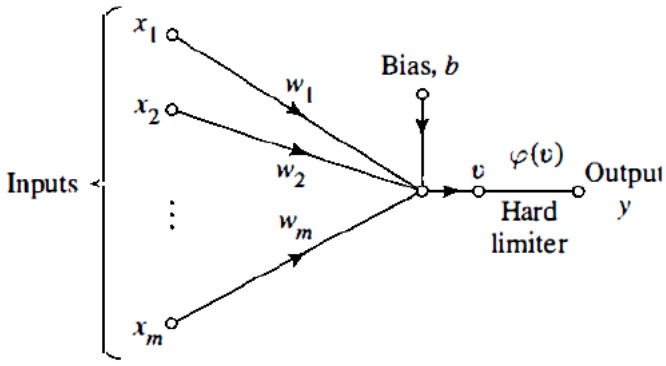
Fuente: ANDERSON, James A. Redes neurales. 2007, p.

4.2.2 El perceptrón

El perceptrón fue la primera red neural que causó un impacto significativo sobre una gran comunidad, particularmente entre los ingenieros, quienes se excitaron e intrigaron con las aplicaciones prácticas posibles de una máquina aprendiz real. El perceptrón fue propuesto por Frank Rosenblatt en 1958 en un reporte técnico ampliamente distribuido por el Laboratorio Aeronáutico de Cornell.

Este modelo neuronal consiste en una combinación lineal seguida por un umbral, descrita por la figura 4.3.

Figura 4.3 Flujo de señales del perceptrón



Fuente: HAYKIN, Simon S. Neural networks 2 ed. 1999, p. 136

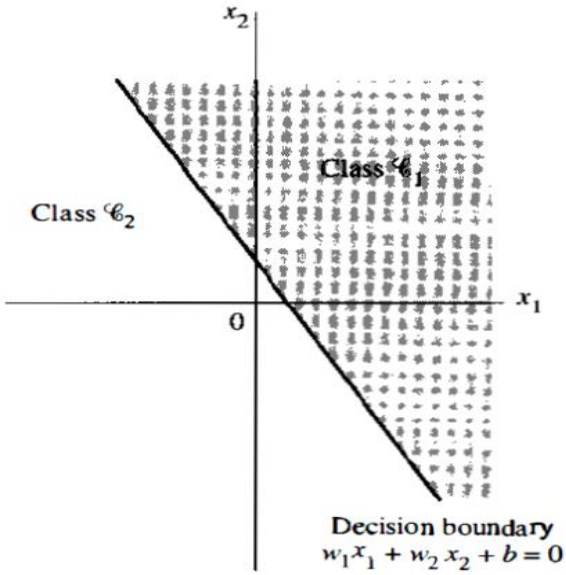
El nodo sumador del modelo neuronal calcula la combinación lineal de las entradas, incorporando también un sesgo (bias) de manera externa. El resultado de la suma es comparado con el umbral, siendo la respuesta de la neurona igual a +1 si el umbral de entrada es superado, y -1 si no lo logra.

En el modelo gráfico de la neurona, los pesos sinápticos del perceptrón son expresados por ω_1, $\omega_{2, \ldots,}\omega_m$. Las entradas del perceptrón son expresadas por x_1, $x_{2, \ldots,}$ x_m. El sesgo aplicado externamente es expresado por b. En el modelo neuronal se encuentra que el hard limiter o el campo local inducido de la neurona es.

$$v = \sum_{i=1}^{m} w_i x_i + b$$

El objetivo del perceptrón es clasificar correctamente el conjunto de estímulos aplicados a la entrada (x_1, $x_{2, \ldots,}$ x_m) en una de dos clases C_1 o C_2. Las entradas se clasificaran como C_1 si la salida y del perceptrón es +1 y C_2 si la salida es -1.

Figura 4.4 Clasificación del perceptrón ilustrada en un plano



Fuente: HAYKIN, Simon S. Neural networks 2 ed. 1999, p. 137

4.2.3 Aprendizaje del perceptrón

El elemento básico de computo en el perceptrón es un dispositivo conocido en la literatura temprana como la *unidad de lógica de umbral* ULU (Nilson, 1965).

En una red neuronal, las únicas capas que son modificables son las capas que van desde la capa A hasta la capa R; aquellas de la retina a la capa A están fijas.

Por ejemplo, al tener dos clases de patrones linealmente separables, llamados {**a**} y {**b**}, se tienen también cierto número de ejemplos de cada clase. Se le presentan al perceptrón estos ejemplos con cierta clase de secuencia (*secuencia de entrenamiento*), {**y**}. El *k*-enésimo miembro de la secuencia de entrenamiento se la conoce como {$\mathbf{y_k}$}. El supervisor conoce la clasificación correcta para cada patrón y puede ajustar todos los pesos.

Al presentarle al sistema un patrón este nos dará una respuesta, si es correcta, no se hace ningún cambio, pero si la respuesta que produce es incorrecta, se procederá a hacer los respectivos ajustes.

Las ULUs de este perceptrón solamente señalan -1 o +1. Si la salida de la capa R es -1, lo que es incorrecto para este ejemplo, entonces, se suma una unidad positiva, *c*, a las fuerzas de conexión desde las células ULUs de la capa A con una actividad de +1 hacia la unidad incorrecta de la capa R. esto incrementará la entrada

positiva hacia la unidad de la capa R, por lo que tendrá mayores posibilidades de exceder el umbral la siguiente vez. Las conexiones de la unidad responden incorrectamente en la capa R hacia las células de la capa A con una actividad de -1, tienen una cantidad de *–c* restada de sus pesos. Nuevamente esto hará que la unidad incorrecta de la capa R tenga mayores posibilidades de exceder el umbral la siguiente vez, ya que recibirá una entrada menos negativa. Si la unidad de la capa R dio como salida un +1 y debió haber respondido con un -1, se aplica la misma regla con el signo opuesto.

La regla corresponde a tomar el patrón presentado, multiplicarlo por *c*, y sumar o restar el resultado de los pesos, dependiendo de si la capa de la unidad R tenía o no una actividad de +1 o de -1.

Considerando un vector de pesos, **w**, y que $\mathbf{w_k}$ es el conjunto de pesos en la unidad de la capa R que se está estudiando en la *k* prueba de entrenamiento.

Entonces, si la clasificación es correcta:

$$w_{k+1} = w_k$$

Si la clasificación es incorrecta:

$$w_{k+1} = w_k + cy_k$$

Si la actividad de la unidad de la capa R debió haber sido +1 y,

$$w_{k+1} = w_k - cy_k$$

Si la actividad de la unidad de la capa R debió haber sido -1.

Considerando los pesos luego de que se hizo la corrección, y siendo la respuesta de la capa R -1, lo que resulto equivocado. Eso significa que el producto interno,

$$[w_k, y_k] \leq 0.$$

Se suman *c* veces $\mathbf{y_k}$ a los pesos. Si el patrón $\mathbf{y_k}$ ocurriera otra vez, el perceptrón tendrá una mayor probabilidad de dar la clasificación correcta, porque el nuevo producto interno en respuesta a $\mathbf{y_k}$ es:

$$\begin{aligned}[w_{k+1}, y_k] &= [(w_k + cy_k), y_k] \\ &= [w_k + y_k] + c[y_k\ , y_k] \\ &> [w_k, y_k].\end{aligned}$$

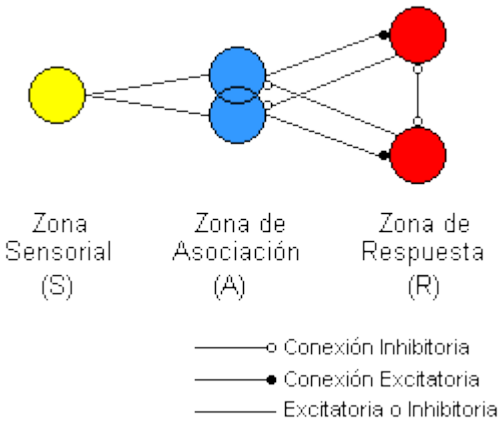
Esta cantidad es más positiva que el resultado antes del aprendizaje, ya que el producto interno de $\mathbf{y_k}$ por sí mismo siempre es positivo, la longitud al cuadrado. Por

tanto, lo pesos están siendo cambiados en la dirección correcta a fin de dar una respuesta correcta la siguiente vez. La segunda regla tenderá a *decrementar* la salida llegando a unidad de la capa R mediante el mismo mecanismo.

Ambas regla harán que sea más probable estar en lo correcto la siguiente vez que sea presentado este patrón.

Esta regla es una forma modificada del aprendizaje hebbiano, en el que el aprendizaje solamente ocurre cuando se comete un error.

Figura 4.5 Esquema de conexiones de un Perceptrón sencillo



Fuente: Universidad de Guadalajara. Perceptrón. Guadalajara (México). Disponible en internet: http://proton.ucting.udg.mx/posgrado/cursos/idc/neuronales2/AntecedentesP.htm

4.3 REDES NEURONALES ARTIFICIALES

4.3.1 Arquitecturas de red

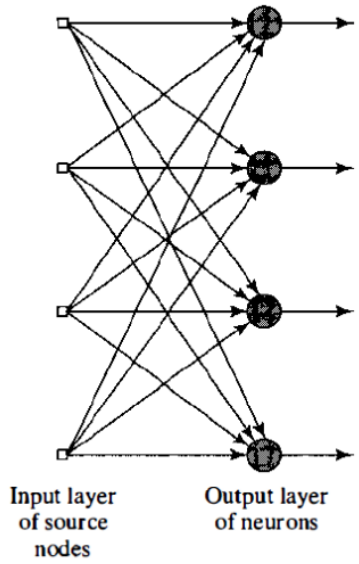
La forma en que las neuronas o las redes neuronales son construidas, van ligadas íntimamente con el algoritmo de entrenamiento usado en dicha red. Generalmente, pueden identificarse tres clases diferentes de arquitecturas de red:

1. Redes Feedforward de una sola capa

En una red neuronal, donde las neuronas son organizadas en forma de capas. En la forma más simple de una red de capas, se tiene una capa de entrada de

fuente de nodos que se une a las neuronas de la capa de salida, pero no viceversa. En la figura 4.6 se puede apreciar una red de una sola capa.

Figura 4.6 Red Feedforward o acíclica con una sola capa de neuronas



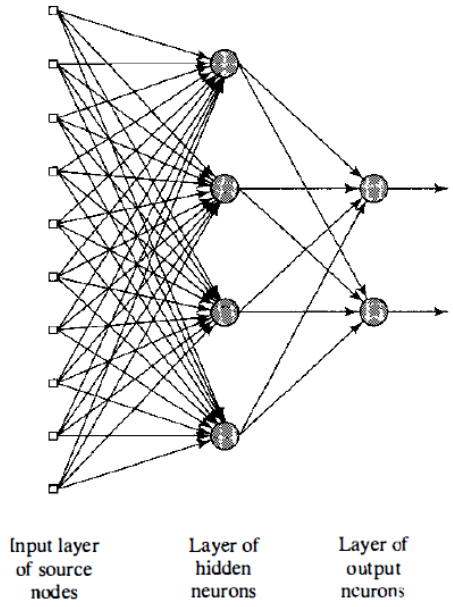
Fuente: HAYKIN, Simon S. Neural networks 2 ed. 1999, p. 21

2. Redes Feedforward multicapa

La segunda clase de redes neuronales tipo feedforward son fácilmente reconocidas por tener una o más capas ocultas, cuyos nodos de cálculo son llamados neuronas ocultas o unidades ocultas. La función de las neuronas ocultas es la de intervenir entre los nodos de entrada y las neuronas de salida de una manera útil.

Los nodos fuente de la capa de entrada de la red suministran los respectivos elementos de los patrones de activación (vector de entrada), que son las señales de entrada a las neuronas de la segunda capa. Las señales de salida de esta segunda capa son usadas como entradas de la capa siguiente, y así sucesivamente para el resto de la red.

Figura 4.7 Conexión de una red feedforward o acíclica con una capa oculta y una capa de salida

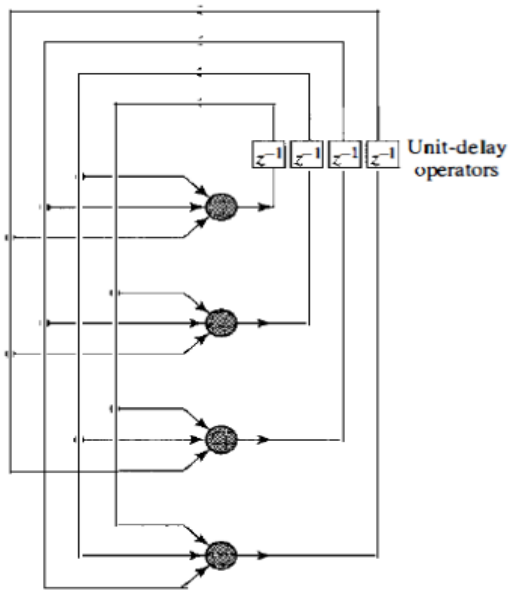


Fuente: HAYKIN, Simon S. Neural networks 2 ed. 1999, p. 22

3. Redes recurrentes

Una red neuronal recurrente se distingue de una red feedforward en que ésta tiene por lo menos un lazo de realimentación (feedback). Por ejemplo, una red recurrente puede estar formada por una sola capa, y con la señal de salida de cada neurona alimentando las entradas de todas las demás neuronas, como se muestra en la figura 4.8. En este tipo de red no hay auto realimentación; la auto-realimentación se refiere a la situación donde la salida de una neurona es realimentada en su propia entrada.

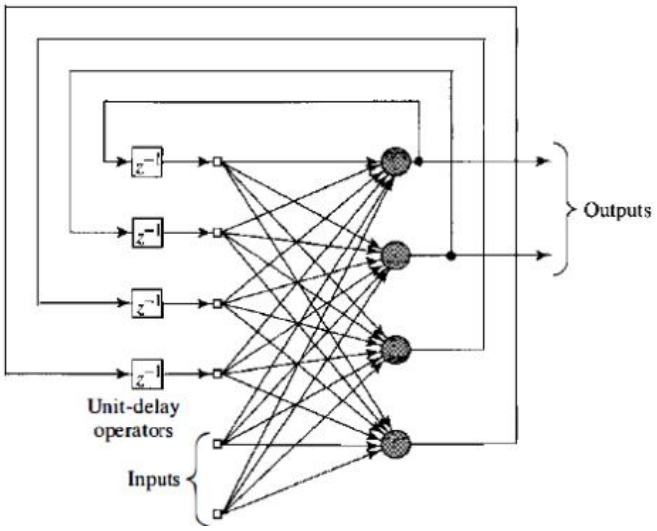
Figura 4.8 Red recurrente sin lazo de auto realimentación y sin neuronas ocultas



Fuente: HAYKIN, Simon S. Neural networks 2 ed. 1999, p. 23

En la figura 4.9 se ilustra otra clase de redes recurrentes con neuronas ocultas. Las conexiones de retroalimentación se originan a partir de las neuronas ocultas, así como de las neuronas de salida.

Figura 4.9 Red recurrente con neuronas ocultas



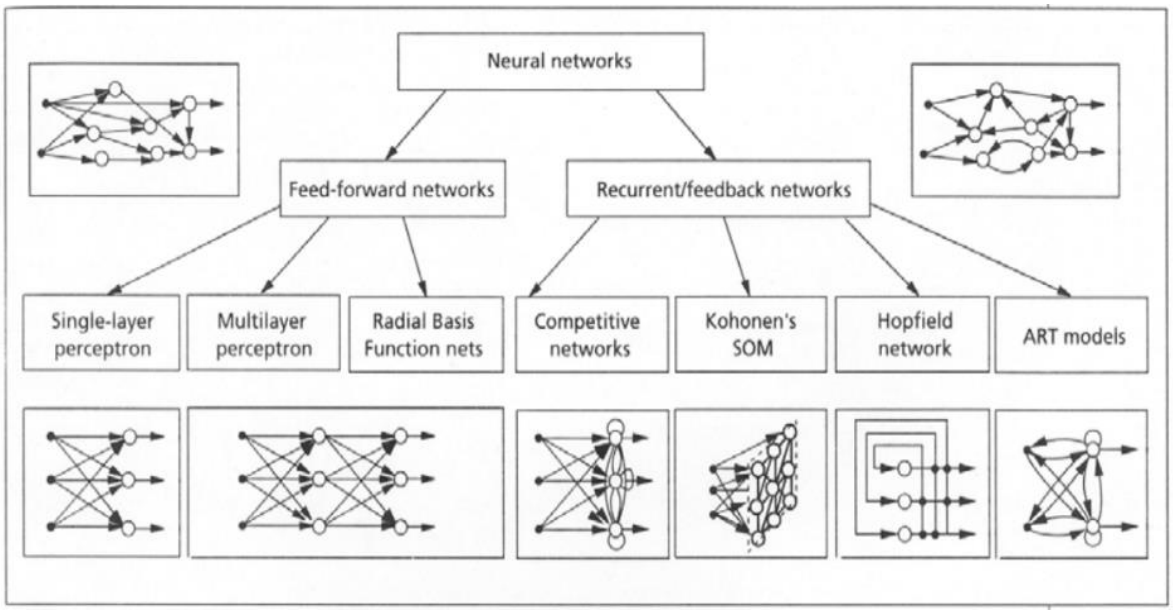
Fuente: HAYKIN, Simon S. Neural networks 2 ed. 1999, p. 24

Cuadro 4.3 Modelo, aprendizaje y tarea de las diferentes redes neuronales

Paradigma	Regla	Arquitectura	Algoritmo de aprendizaje	Tarea
Supervisado	Corrección de errores	Perceptrón simple o multicapa	Aprendizaje del perceptrón. Propagación hacia atrás. Adaline y Madaline	Clasificación de patrones. Predicción de la aproximación de funciones y control.
	Boltzmann	Recurrente	Aprendizaje de Boltzmann	Clasificación de patrones.
	Hebbian	Multicapa de propagación hacia delante	Análisis lineal discriminante	Análisis de datos. Clasificación de patrones.
	Competitivo	Competitivo	Cuantización del vector de aprendizaje	Categorización en clases. Compresión de datos.
		Red ART	ARTMap	Clasificación de patrones. Categorización en clases.
No supervisado	Corrección de errores	Multicapa de propagación hacia delante	Proyección de Sammon	Análisis de datos.
	Hebbian	De propagación hacia adelante o competitivo	Análisis de componente principal.	Análisis de datos. Compresión de datos.
		Red de Hopfield	Aprendizaje de memoria asociativa.	Memoria asociativa.
	Competitivo	Competitivo	Cuantización de vector.	Categorización. Compresión de datos.
		SOM de Kohonen	SOM de Kohonen	Categorización. Análisis de datos.
		Red ART	ART1, ART2	Categorización.
Híbrido	Corrección de errores y competitivo.	Red RBF	Algoritmo de aprendizaje de RBF	Clasificación de patrones. Predicción de la aproximación de funciones y control.

Fuente: MUÑOZ GUTIERREZ, Carlos. Redes neurales. Madrid (España): Universidad Complutense. Disponible en internet: http://pendientedemigracion.ucm.es/info/pslogica/redesdoc.pdf

Figura 4.10 Clasificación de los tipos de redes neuronales artificiales



Fuente: MUÑOZ GUTIERREZ, Carlos. Redes neurales. Madrid (España): universidad Complutense. Disponible en internet: http://pendientedemigracion.ucm.es/info/pslogica/redesdoc.pdf

4.3.2 Entrenamiento de una red neuronal artificial

Debido a los diferentes tipos de redes y sus respectivos métodos de entrenamiento, era necesario tomar la decisión de qué tipo de RNA se implementaría, junto con su algoritmo de entrenamiento, y se decidió por trabajar con una red neuronal de arquitectura perceptrón multicapa con un entrenamiento supervisado y con el algoritmo de propagación hacia atrás (Backporpagation).

A continuación se describirá el algoritmo de entrenamiento usado.

4.3.2.1 Algoritmo de propagación hacia atrás (Backpropagation).

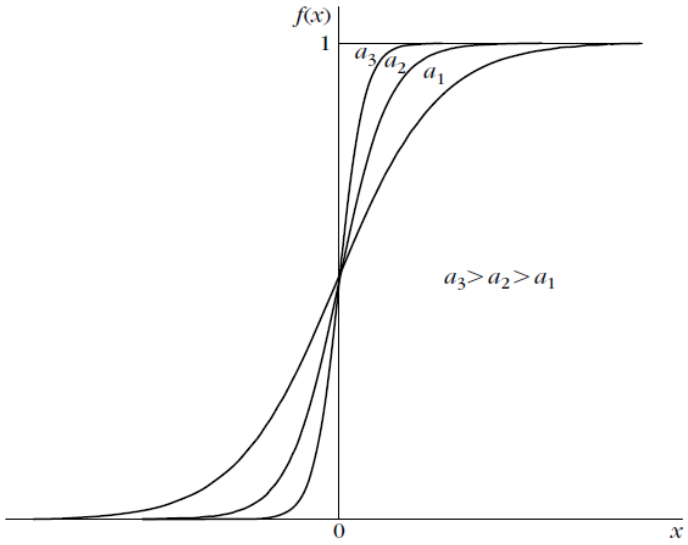
Para el correcto funcionamiento de una RNA, hay que tener una buena función de activación en cada uno de los perceptrones que la componen, uno de los más recomendados es la función sigmoidea.

$$f(x) = \frac{1}{1 + \exp(-ax)}$$

Donde a es el parámetro de pendiente de la función.

En la siguiente figura se muestra la función sigmoidea con diferentes valores de a junto a la función paso unitario.

Figura 4.11 Función sigmoidea con diferentes valores de pendiente



Fuente: THEODORIDIS, Sergios. Pattern Recognition 4 ed. 2009, p. 163

La función sigmoidea es de gran relevancia en el comportamiento de las RNA, debido a su semejanza con los comportamientos excitatorios de la neurona biológica.

A menudo, se realiza una variación a la función sigmoidea original con el fin de ampliar su rango de trabajo y mejorar la respuesta de la red. En este caso se usará una función sigmoidea bipolar de modo que:

$$f(x) = \frac{2}{1 + \exp(-ax)} - 1$$

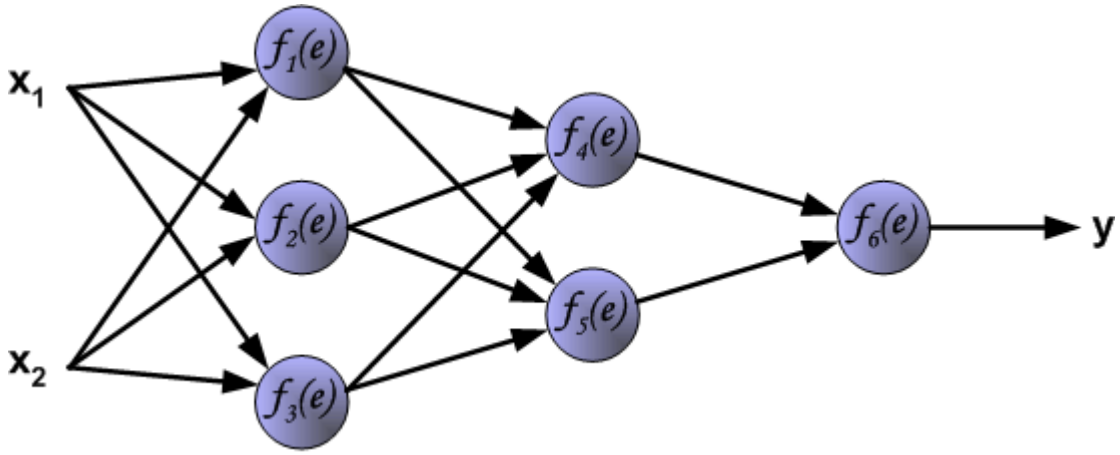
Donde la función variará entre los valores de 1 y -1.

El algoritmo de entrenamiento de propagación hacia atrás o backpropagation está compuesto de los siguientes pasos:

- Inicialización: se inicia poniendo valores pequeños (entre -1 y 1) a todos pesos de la red de manera aleatoria.
- Cálculo hacia adelante: teniendo los patrones de entrada a la red y todos los valores de pesos asignados a las conexiones entre neuronas, se procede a calcular cada una de las salidas de estas, haciendo la ponderación de los valores de entrada.
- Cálculo hacia atrás: después de obtener el resultado final de la red, se compara con la salida deseada, y el error producto de esta comparación es propagado hacia atrás, con un valor directamente proporcional a los pesos de las conexiones neuronales hasta llegar a la primera capa de neuronas.
- Actualización de pesos: en este paso se actualizan los valores de los pesos sinápticos, teniendo en cuenta los valores de peso actuales, el valor de error y las respuestas neuronales.

Las siguientes ilustraciones mostrarán paso a paso el desarrollo del algoritmo backpropagation en una RNA, el ejemplo usado consta de tres capas con dos entradas y una salida, como se muestra a continuación:

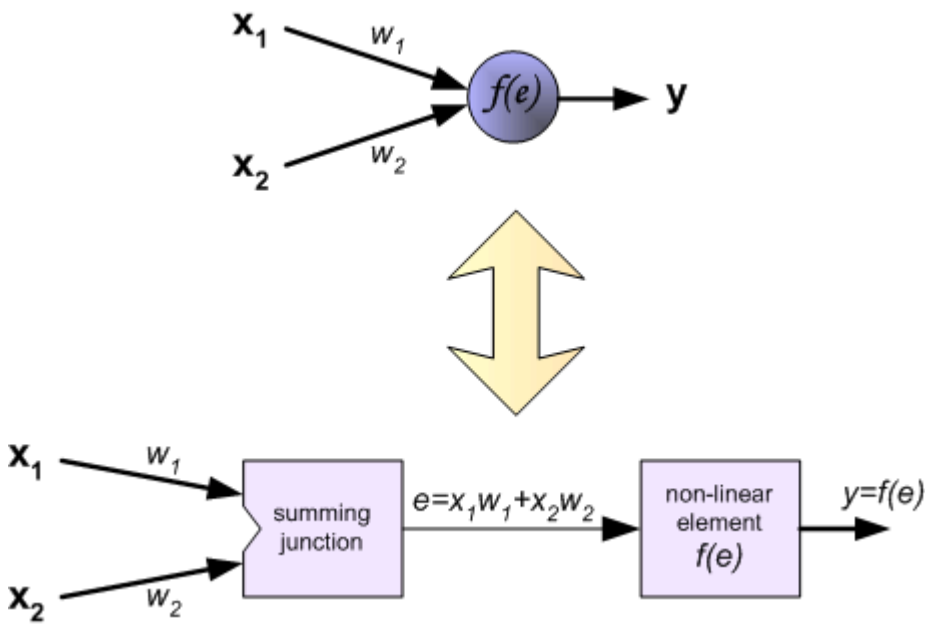
Figura 4.12 Red neuronal artificial multicapa



Fuente: BERNACKI, Mariusz y WŁODARCZYK, Przemysław. Principles of training multi-layer neural network using backpropagation. Cracovia: Universidad AGH. Departamento de electrónica, 2005. Disponible en internet: http://home.agh.edu.pl/~vlsi/AI/backp_t_en/backprop.html

Cada neurona está compuesta por dos unidades. La primera se encarga de hacer la ponderación de los pesos sinápticos y las señales de entrada. La segunda unidad se encarga de operar la respuesta de la unidad número uno en una función no lineal, llamada función de activación. Esta última respuesta, es también la respuesta de la neurona.

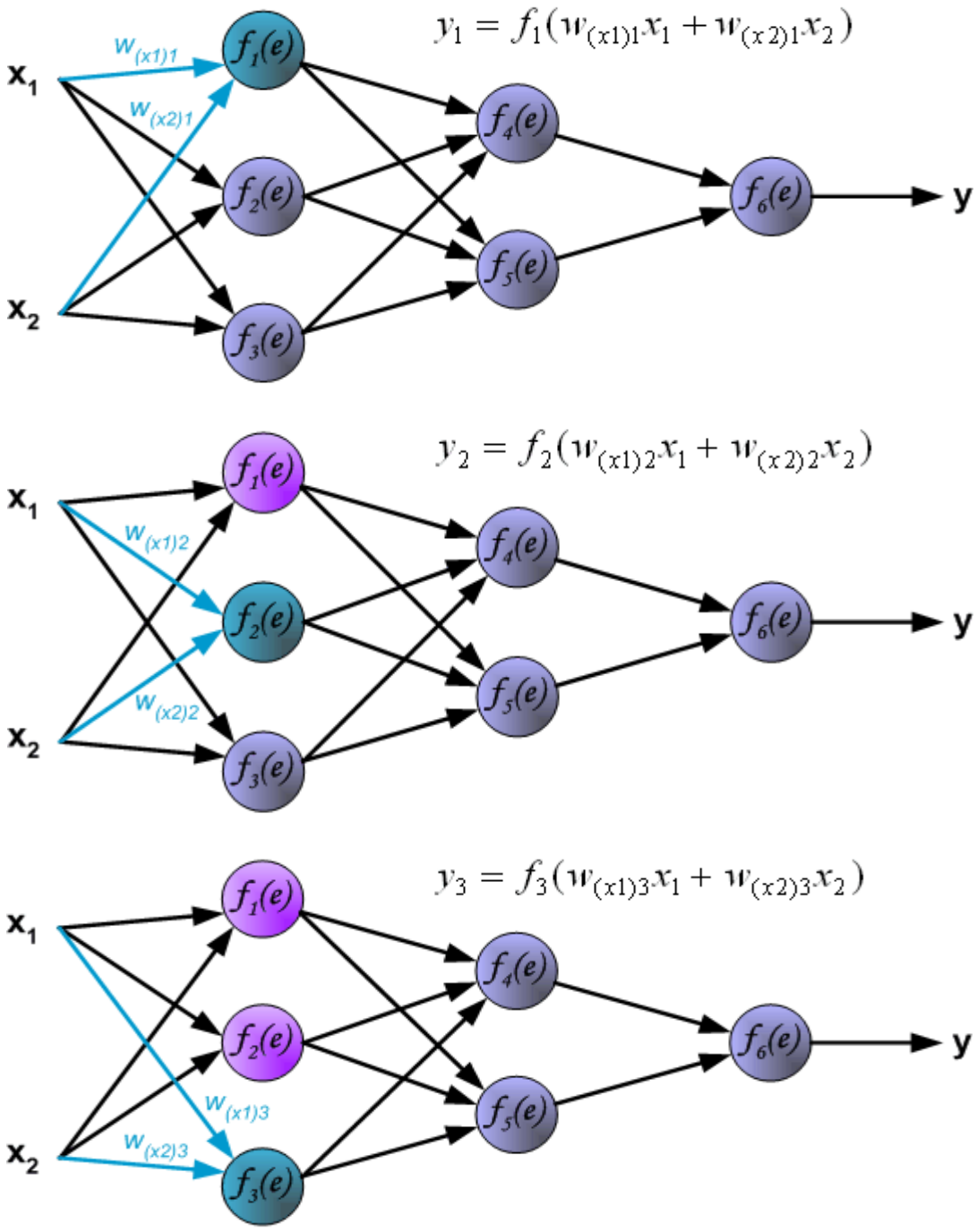
Figura 4.13 Composición de una neurona artificial



Fuente: BERNACKI, Mariusz y WŁODARCZYK, Przemysław. Principles of training multi-layer neural network using backpropagation. Cracovia: Universidad AGH. Departamento de electrónica, 2005. Disponible en internet: http://home.agh.edu.pl/~vlsi/AI/backp_t_en/backprop.html

Después de darle valores aleatorios a las conexiones sinápticas y tener los valores de entrada para el entrenamiento de la RNA, se procede a calcular la respuesta de la red, procesando capa por capa de neuronas.

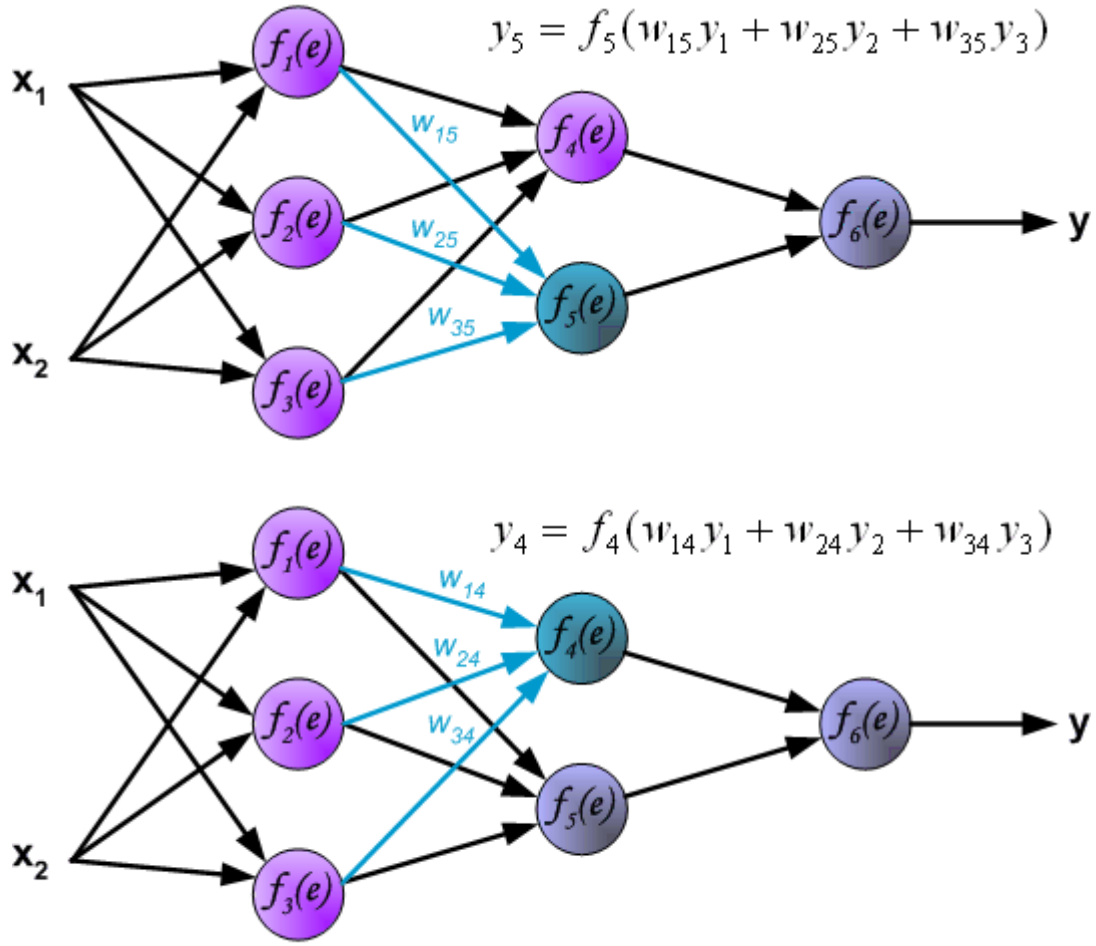
Figura 4.14 Flujo hacia adelante de una RNA multicapa (primera capa)



Fuente: BERNACKI, Mariusz y WŁODARCZYK, Przemysław. Principles of training multi-layer neural network using backpropagation. Cracovia: Universidad AGH. Departamento de electrónica, 2005. Disponible en internet: http://home.agh.edu.pl/~vlsi/AI/backp_t_en/backprop.html

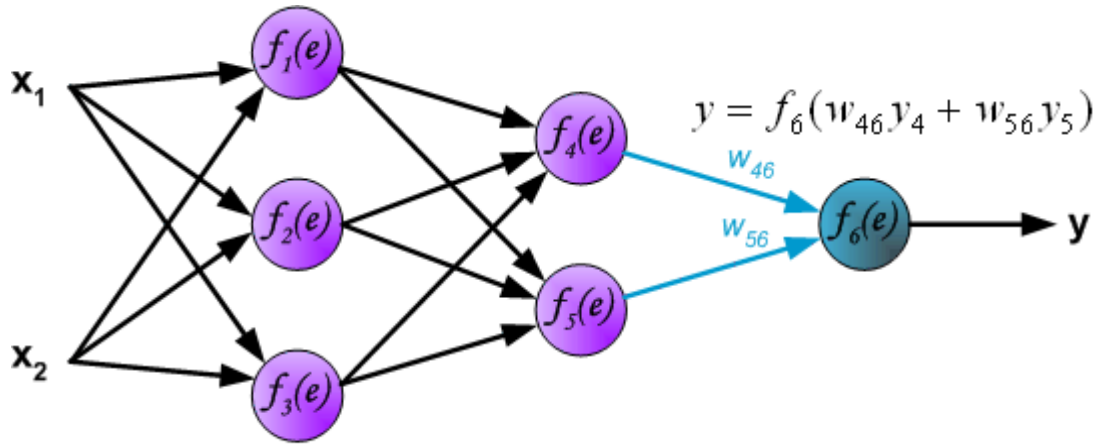
Las salidas de la primera capa de neuronas se convierten en entradas de la segunda capa, y de esta manera se repite hasta llegar a la capa de salida.

Figura 4.15 Flujo hacia adelante de una RNA multicapa (segunda capa)



Fuente: BERNACKI, Mariusz y WŁODARCZYK, Przemysław. Principles of training multi-layer neural network using backpropagation. Cracovia: Universidad AGH. Departamento de electrónica, 2005. Disponible en internet: http://home.agh.edu.pl/~vlsi/AI/backp_t_en/backprop.html

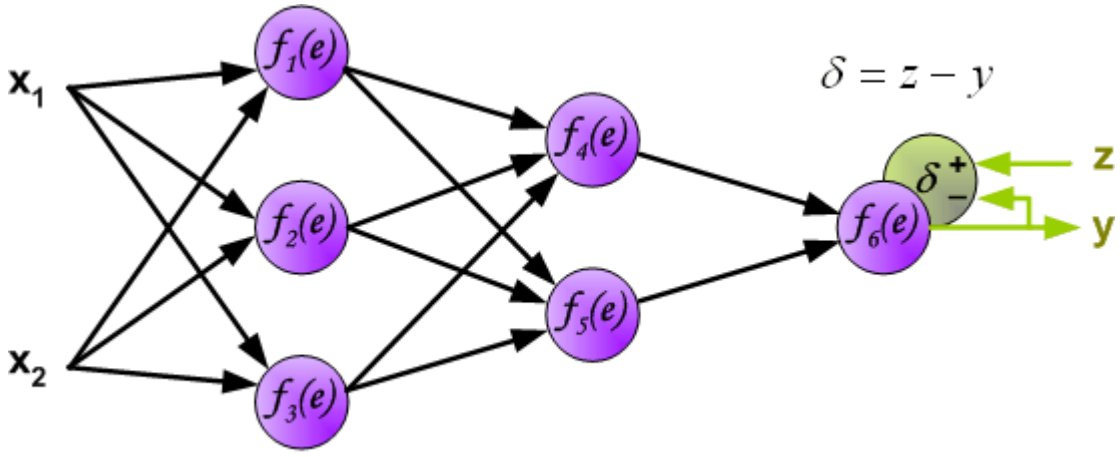
Figura 4.16 Flujo hacia adelante de una RNA multicapa (respuesta de la red)



Fuente: BERNACKI, Mariusz y WŁODARCZYK, Przemysław. Principles of training multi-layer neural network using backpropagation. Cracovia: Universidad AGH. Departamento de electrónica, 2005. Disponible en internet: http://home.agh.edu.pl/~vlsi/AI/backp_t_en/backprop.html

El siguiente paso es comparar la respuesta de la RNA con la salida esperada del conjunto de valores de entrenamiento. La diferencia se llama señal de error δ.

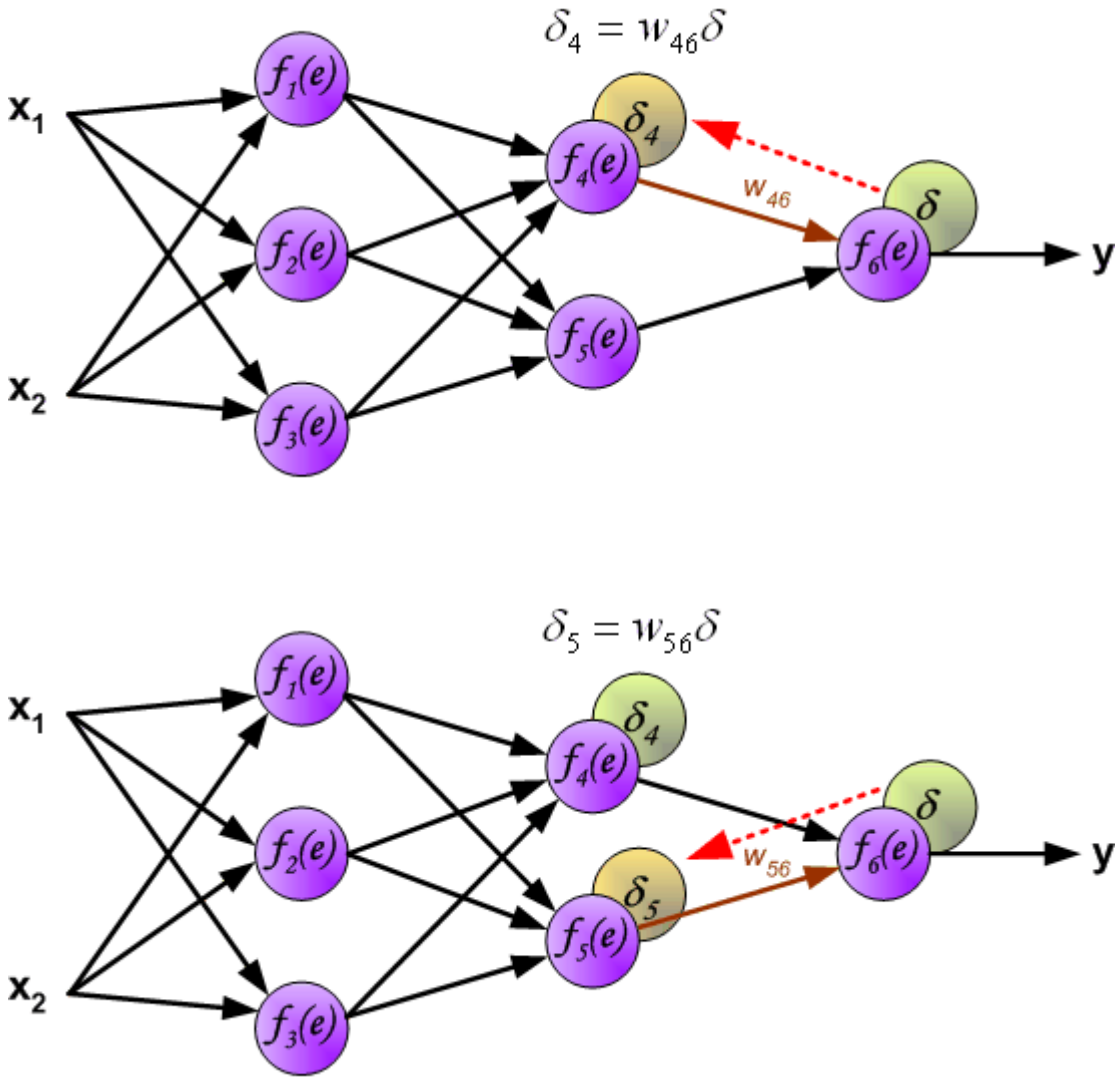
Figura 4.17 Flujo hacia atrás de una RNA multicapa (propagación de error)



Fuente: BERNACKI, Mariusz y WŁODARCZYK, Przemysław. Principles of training multi-layer neural network using backpropagation. Cracovia: Universidad AGH. Departamento de electrónica, 2005. Disponible en internet: http://home.agh.edu.pl/~vlsi/AI/backp_t_en/backprop.html

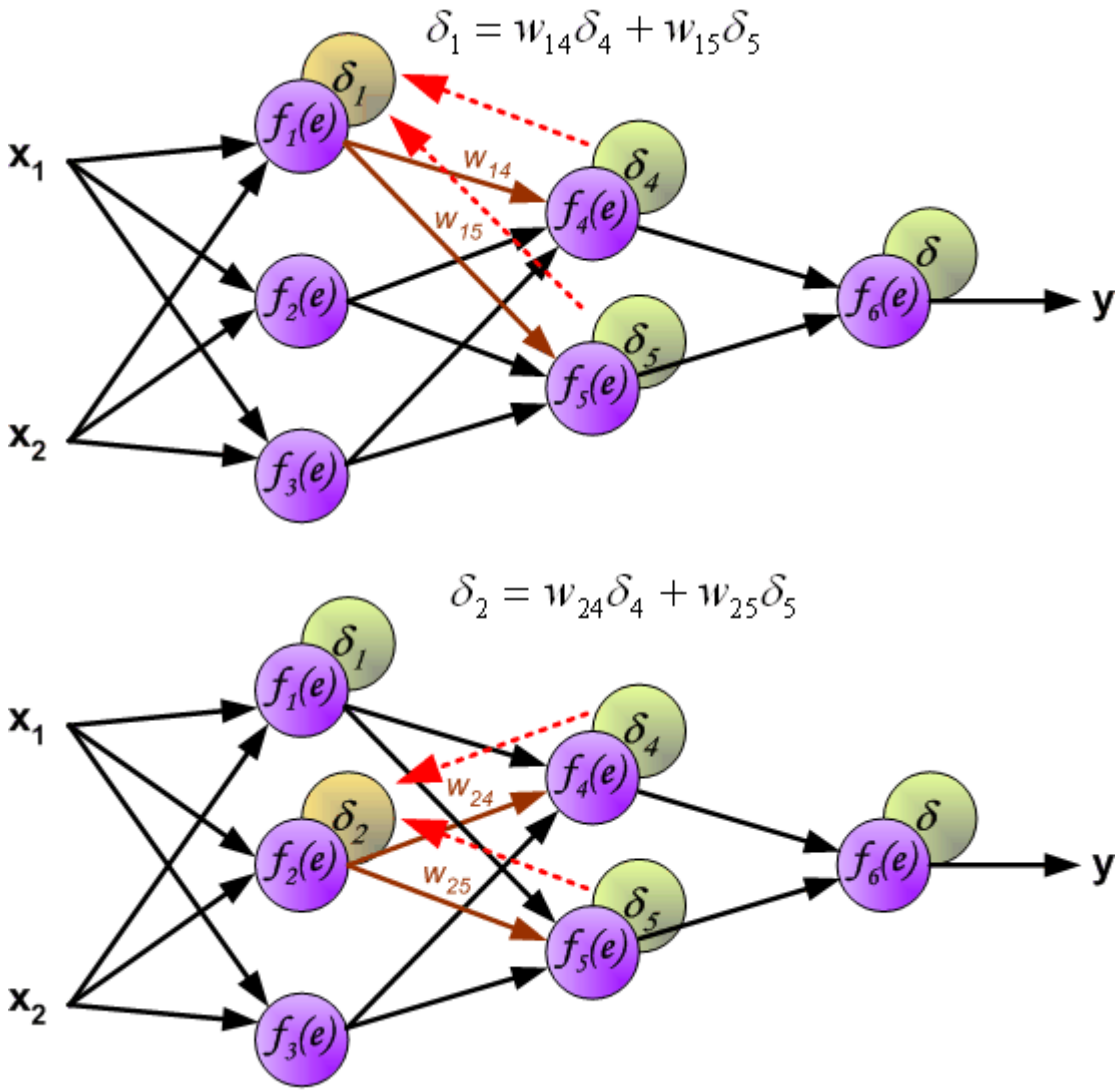
El cálculo del error de las neuronas internas no es posible determinarlo de manera directa, por lo que el algoritmo de backpropagation propone propagar la señal de error δ hacia atrás, dirigiéndose por cada una de las conexiones sinápticas hasta llegar a la capa de neuronas de entrada.

Figura 4.18 Flujo hacia atrás de una RNA multicapa (propagación de error en segunda capa)



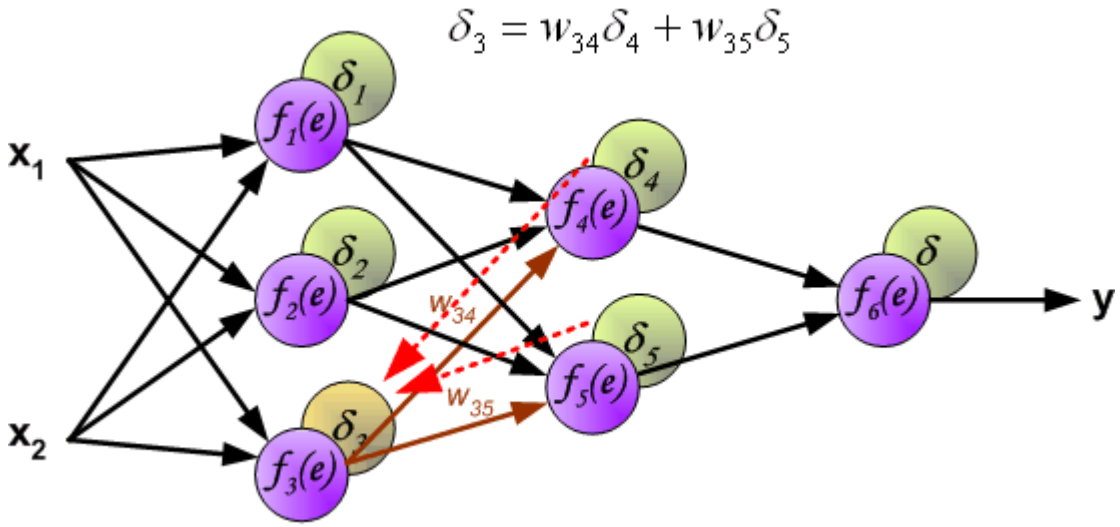
Fuente: BERNACKI, Mariusz y WŁODARCZYK, Przemysław. Principles of training multi-layer neural network using backpropagation. Cracovia: Universidad AGH. Departamento de electrónica, 2005. Disponible en internet: http://home.agh.edu.pl/~vlsi/AI/backp_t_en/backprop.html

Figura 4.19 Flujo hacia atrás de una RNA multicapa (propagación de error en primera capa)



Fuente: BERNACKI, Mariusz y WŁODARCZYK, Przemysław. Principles of training multi-layer neural network using backpropagation. Cracovia: Universidad AGH. Departamento de electrónica, 2005. Disponible en internet: http://home.agh.edu.pl/~vlsi/AI/backp_t_en/backprop.html

Figura 4.19 (Continuación)

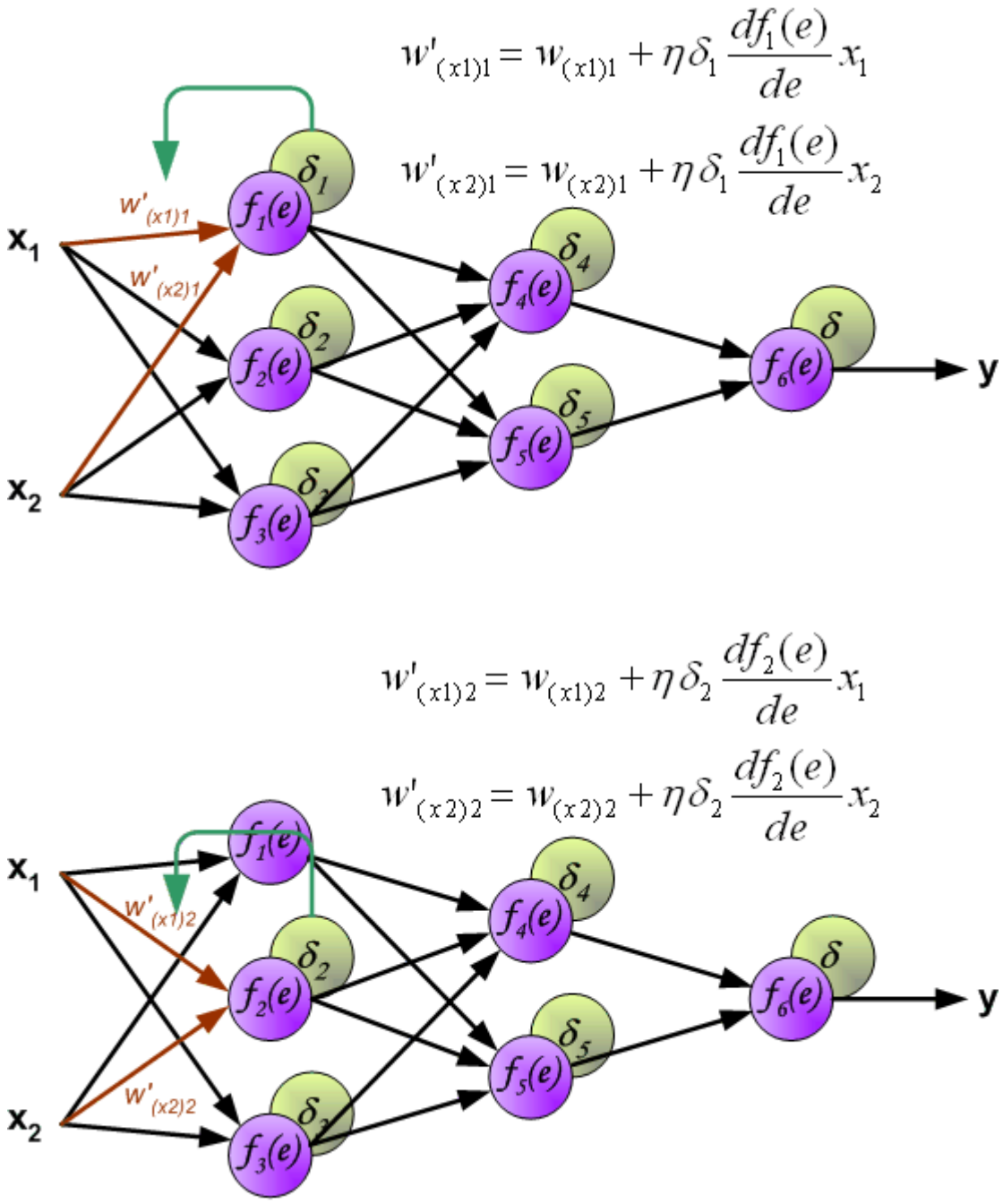


Fuente: BERNACKI, Mariusz y WŁODARCZYK, Przemysław. Principles of training multi-layer neural network using backpropagation. Cracovia: Universidad AGH. Departamento de electrónica, 2005. Disponible en internet: http://home.agh.edu.pl/~vlsi/AI/backp_t_en/backprop.html

Cuando las señales de error de cada una de las neuronas son calculadas, los pesos sinápticos deben ser modificados, con el fin de minimizar el error en la siguiente iteración.

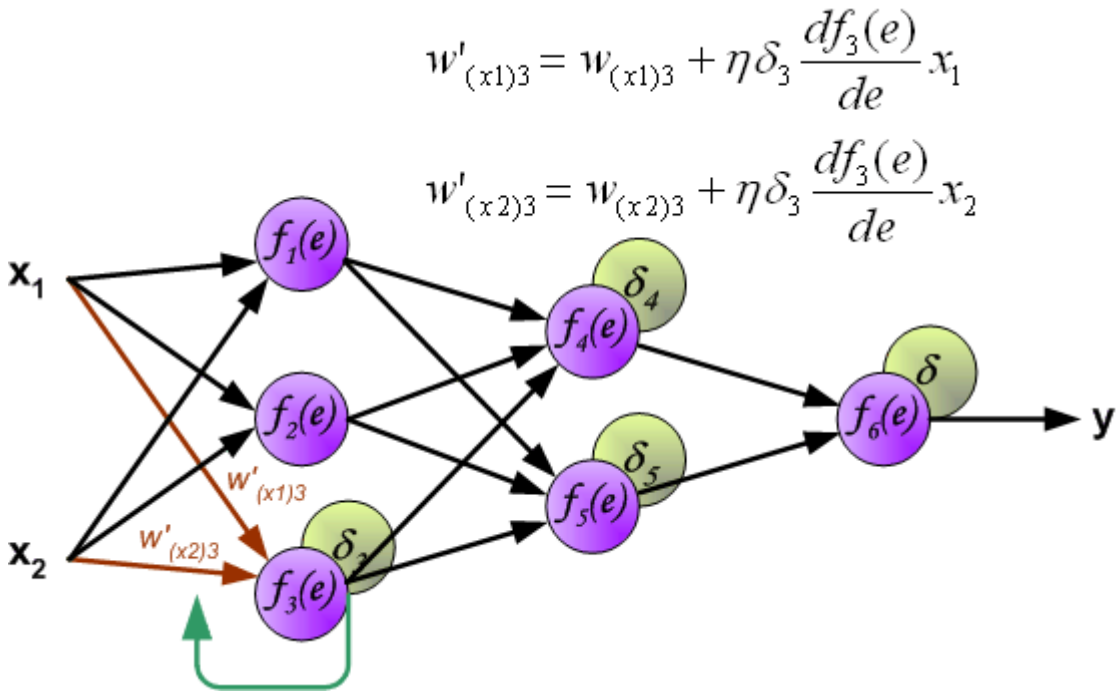
En las ecuaciones η es la razón de aprendizaje de la red, $\frac{df(e)}{de}$ es la derivada de la función de activación de la neurona, w es el peso actual de la conexión sináptica y w' es el peso actualizado para dicha conexión.

Figura 4.20 Actualización de pesos sinápticos de una RNA multicapa (primera capa)



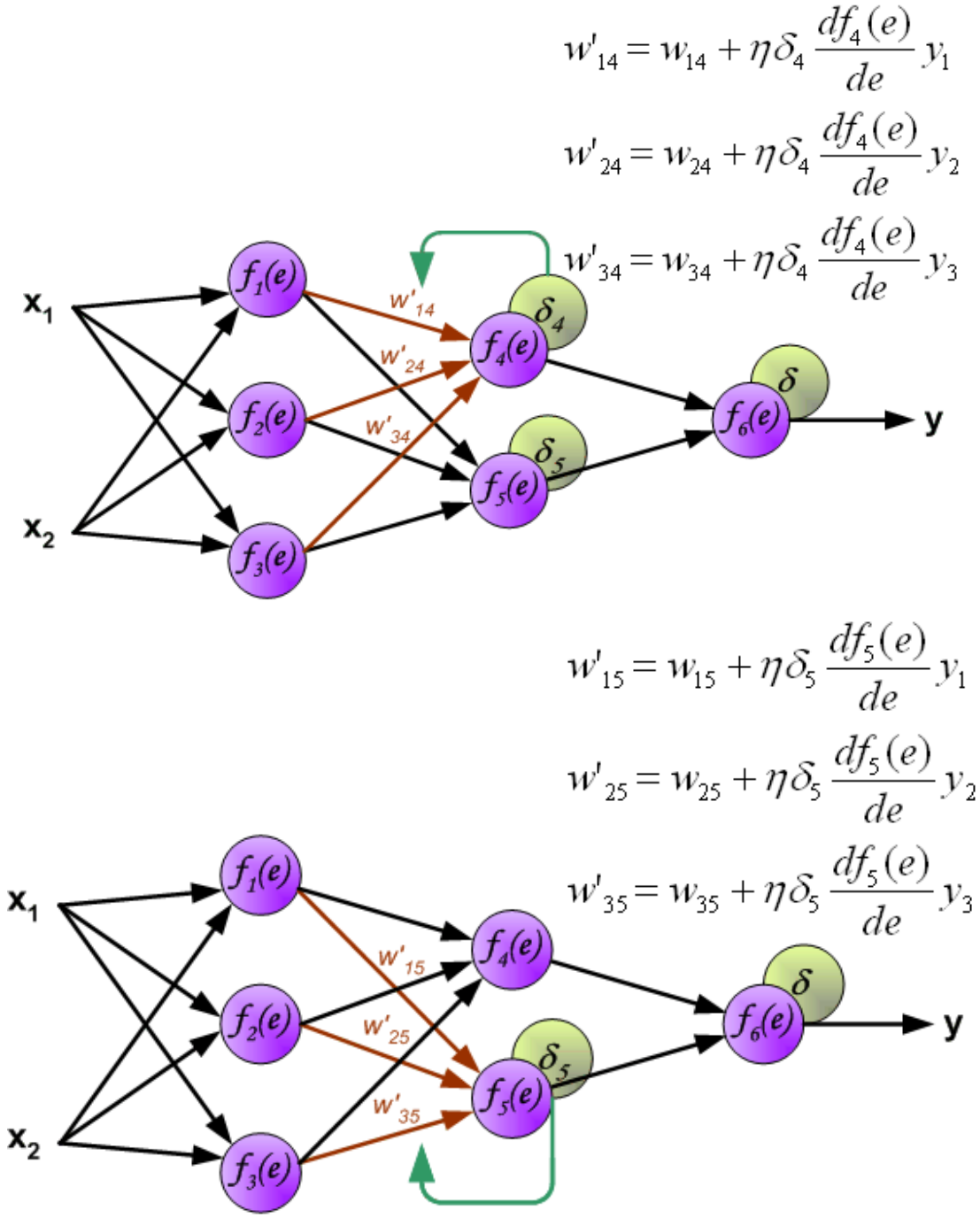
Fuente: BERNACKI, Mariusz y WŁODARCZYK, Przemysław. Principles of training multi-layer neural network using backpropagation. Cracovia: Universidad AGH. Departamento de electrónica, 2005. Disponible en internet: http://home.agh.edu.pl/~vlsi/AI/backp_t_en/backprop.html

Figura 4.20 (Continuación)



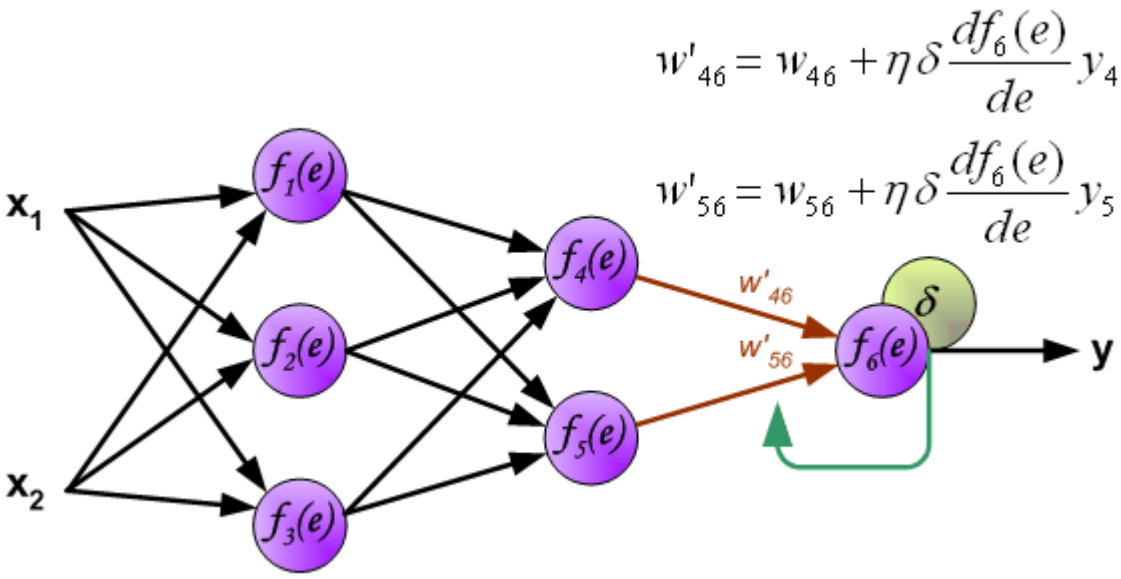
Fuente: BERNACKI, Mariusz y WŁODARCZYK, Przemysław. Principles of training multi-layer neural network using backpropagation. Cracovia: Universidad AGH. Departamento de electrónica, 2005. Disponible en internet: http://home.agh.edu.pl/~vlsi/AI/backp_t_en/backprop.html

Figura 4.21 Actualización de pesos sinápticos de una RNA multicapa (segunda capa)



Fuente: BERNACKI, Mariusz y WŁODARCZYK, Przemysław. Principles of training multi-layer neural network using backpropagation. Cracovia: Universidad AGH. Departamento de electrónica, 2005. Disponible en internet: http://home.agh.edu.pl/~vlsi/AI/backp_t_en/backprop.html

Figura 4.22 Actualización de pesos sinápticos de una RNA multicapa (capa de salida)



Fuente: BERNACKI, Mariusz y WŁODARCZYK, Przemysław. Principles of training multi-layer neural network using backpropagation. Cracovia: Universidad AGH. Departamento de electrónica, 2005. Disponible en internet: http://home.agh.edu.pl/~vlsi/AI/backp_t_en/backprop.html

Este proceso se repite hasta que todo el conjunto de patrones de entrenamiento se cumpla.

5. IMPLEMENTACIÓN

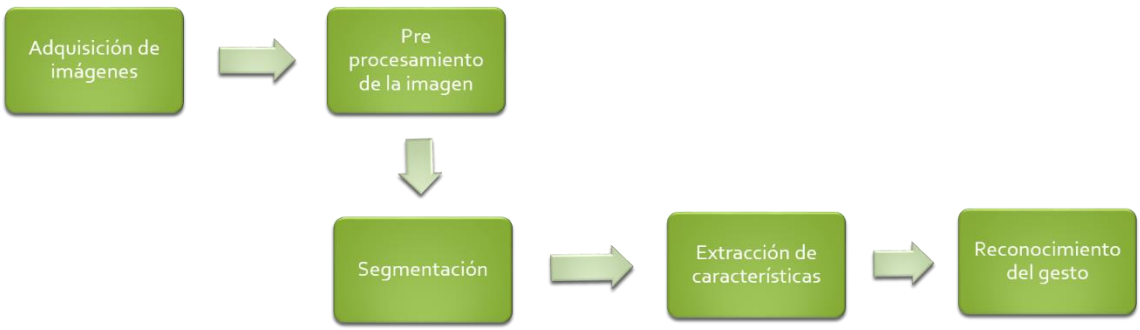
En este capítulo se describirán los elementos usados en el desarrollo del proyecto, como también se mostrarán las fases de desarrollo del mismo.

El proyecto se desarrolló teniendo en cuenta la siguiente estructura de trabajo:

- Adquisición de la imagen en el espacio de color RGB con C++.
- Adecuación del ambiente de trabajo (Sistema de iluminación).
- Transformación del modelo de color RGB a HSV para manipular las imágenes en C++.
- Extracción del objeto usando el modelo de color HSV con sus respectivas componentes en C++.
- Eliminación del ruido en la imagen por medio de operaciones morfológicas.
- Creación del banco de imágenes que componen el alfabeto de señas del ASL.
- Extracción de las características de las imágenes usando el modelo de momento de imágenes.
- Clasificación de los patrones extraídos en grupos según el gesto.
- Diseño de la Red Neuronal Artificial (Creación de las neuronas en C++).
- Entrenamiento de la Red Neuronal Artificial (Implementación del algoritmo Backpropagation en C++).

Este proyecto se realizó teniendo en cuenta el siguiente esquema de visión artificial.

Figura 5.1 Imagen adquirida en espacio de color RGB con recuadro



Fuente: Autor

5.1 HARDWARE UTILIZADO EN EL PROYECTO

Dentro de los elementos físicos de tipo electrónico usados en el proyecto, se hizo uso de una cámara web.

5.1.1 Cámara de adquisición de imágenes

Se empleó una cámara web HC-328 Black, con un lente de ¼'', de imagen a color con sensor CMOS y una resolución de 1.3 megapíxeles.

Figura 5.2 Cámara web Startec HC-328

Fuente: Startec. Cámara Web HC-328 Black. Disponible en internet: http://www.mystartec.com/productos/camaras/camara_web.html#nogo

5.2 SOFTWARE UTILIZADO EN EL PROYECTO

En la parte de software, se hizo uso del entorno de desarrollo integrado (IDE) Visual C++ y la biblioteca de visión artificial OpenCV.

5.2.1 IDE Visual C++

Este IDE nos proporciona la facilidad de poder desarrollar aplicaciones con el lenguaje C++, además de ser compatible con la biblioteca OpenCV versión 2.4.4 para Windows.

5.2.2 Biblioteca de visión artificial OpenCV

Para el procesamiento de imágenes se hace uso de la biblioteca gratuita de visión artificial OpenCV versión 2.4.4 para Windows, desarrollada por la corporación Intel y distribuida de manera gratuita. Esta biblioteca también está disponible para otras plataformas como Linux, Mac, Android e iOS.

5.3 DESARROLLO OBJETIVO ESPECÍFICO 1

- Diseñar el algoritmo de pre procesamiento de las imágenes para reducción de ruido.

Actividades

- Adquisición de la imagen.
- Adecuación del ambiente de trabajo.
- Establecer un modelo de color para manipular las imágenes.

5.3.1 Adquisición y procesamiento de imágenes

En esta parte se adquiere la imagen con la que se va a trabajar, y se cambian los espacios de color del RGB (original), al HSV para segmentar por color (en el caso del proyecto el color negro).

La captura de la imagen se hace con la función VideoCapture, como se muestra a continuación:

```
//Iniciamos la captura
	VideoCapture cap;
	cap.open(1);

	while(1){
		//(Lo que captura la WebCam lo pasa a la Mat image)
		cap>>image;
		//ubicamos un rectangulo en la pantalla

	rectangle(image,Point(0,0),Point(2*w/3,2*w/3),Scalar(0,138,21),2,4);
```

Donde cap va a ser la matriz que guardará la información que envía la cámara, y cap.open(1) habilitará la cámara número 1 que tengamos conectada al computador.

Para limitar el rango de captura de la imagen se dibuja un cuadro, el cual servirá para colocar la mano y extraer una imagen más controlada. Esto se hace con la función rectangle de OpenCV.

Figura 5.3 Imagen adquirida en espacio de color RGB con recuadro

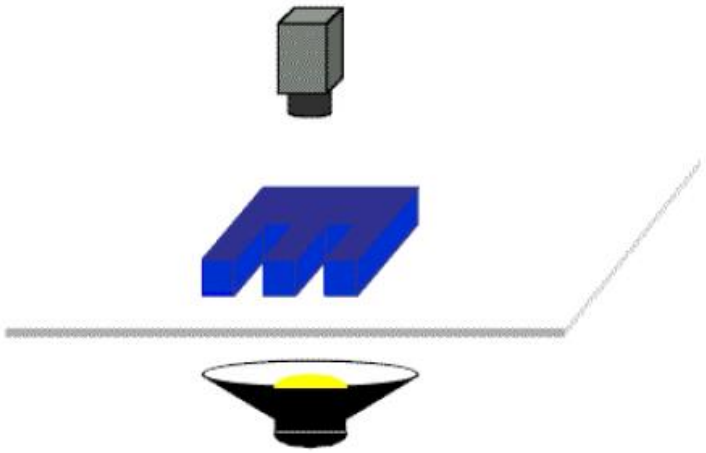
Fuente: Autor

5.3.2 Adecuación del ambiente de trabajo

El sistema de iluminación en esta parte del proyecto ayudará a disminuir la cantidad y complejidad de las líneas de código de pre procesamiento de las imágenes adquiridas. Por eso un buen sistema de iluminación hará la diferencia, a la hora de adquirir y segmentar las imágenes.

El sistema de iluminación usado es el de iluminación posterior, en el cual la iluminación se dirige hacia la cámara y el objeto se sitúa en la mitad de ambos.

Figura 5.4 Sistema de iluminación posterior (Backlight)



Fuente: SOBRADO MALPARTIDA, Eddie Ángel. Sistema de Visión Artificial para el Reconocimiento y Manipulación de Objetos utilizando un Brazo Robot.

Ventajas

- Alto contraste entre el objeto y el background.
- Permite delinear el contorno del objeto.
- Permite visualizar perforaciones pasantes a través del objeto.

5.3.3 Modelo de color para manipular las imágenes

Ahora la transformación de espacios de color se hace necesaria para poder extraer el objeto que se va a manipular, es por eso que se usa el espacio de color HSV, y se usa el componente valor o brillo de este modelo igual a cero, para extraer el color negro de la imagen.

Figura 5.5 Imagen adquirida en espacio de color RGB

Fuente: Autor

Figura 5.6 Imagen adquirida en espacio de color HSV

Fuente: Autor

```
//Convierte al modelo HSV la imagen capturada
              cvtColor(image,HSV,COLOR_BGR2HSV);

              //Mostramos las imagenes de la WebCam y la HSV
              imshow(windowName,image);
              imshow(windowName1,HSV);

              //El rango del filtro HSV va a estar desde _MIN hasta _MAX de los
valores de
           //Hue, Saturation y Value
              inRange(HSV,Scalar(H_MIN,S_MIN,V_MIN),
                                 Scalar(H_MAX,S_MAX,V_MAX),Filtro);
```

La sección del código anterior muestra la conversión del modelo RGB al HSV, cuyos valores de Matiz, Saturación y Valor son iguales a cero.

5.4 DESARROLLO OBJETIVO ESPECÍFICO 2

- Diseñar el algoritmo de pre procesamiento de las imágenes para reducción de ruido.

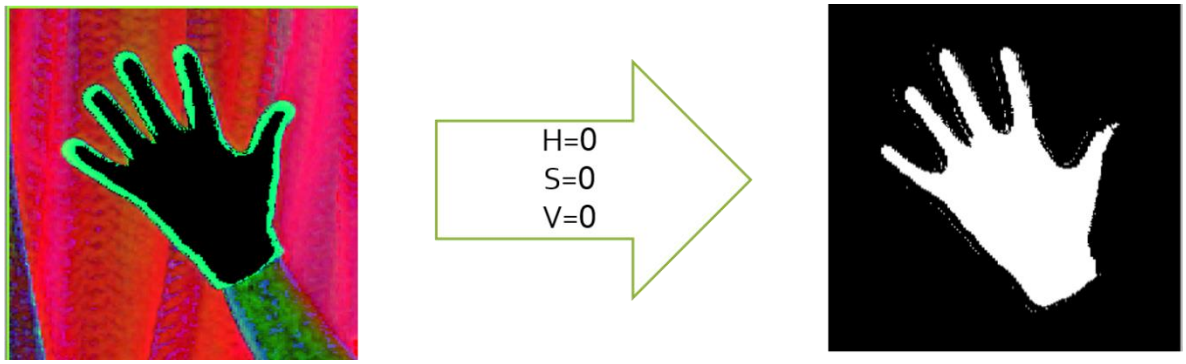
Actividades

- Extracción del objeto usando el modelo de color HSV.
- Mejorar la imagen por medio de operaciones morfológicas.

5.4.1 Segmentación de imágenes

En la segmentación, se separa de la imagen adquirida el objeto relevante de información, en este caso será el color negro del guante. Esta separación no será del todo exacta, y habrá ciertos puntos de color negro que se separarán junto con la figura de la mano, para lo cual se usarán las operaciones morfológicas de erosión y dilatación para suprimir estas áreas.

Figura 5.7 Objeto en modelo HSV y binario



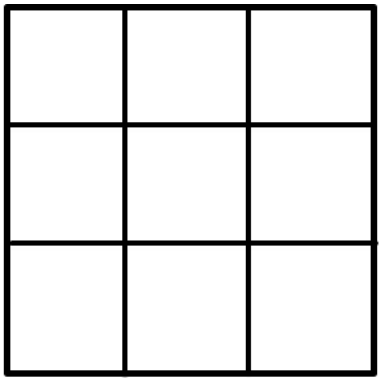
Fuente: Autor

5.4.2 Mejora de las imágenes por medio de operaciones morfológicas

A partir de la imagen segmentada, se procede a manipularla con el fin de mejorar su apariencia disminuyendo el ruido y definiendo su contorno.

Las operaciones morfológicas de erosión y dilatación se aplican a objeto segmentado con un elemento estructurante cuadrado de dimensiones 3x3.

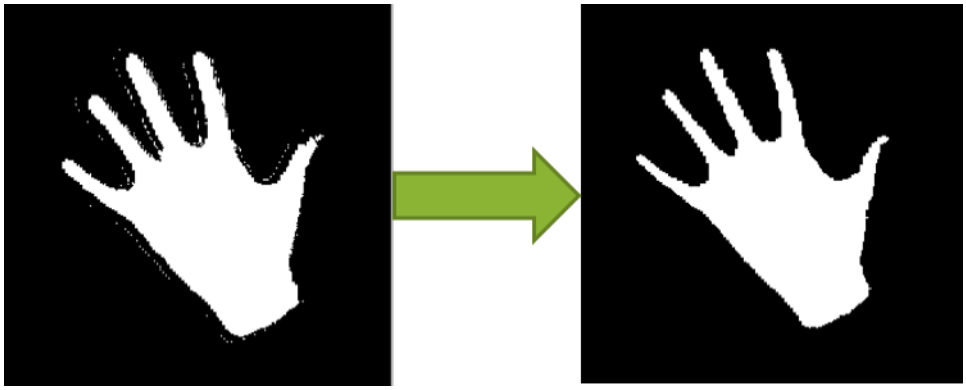
Figura 5.8 Elemento estructurante para las operaciones morfológicas



Fuente: Autor

La primera operación realizada es la de erosión, con esta eliminamos los pequeños puntos blancos (ruido) alrededor del objeto. Aunque obtenemos una nueva imagen sin ruido está muy delgada, entonces se aplicará la dilatación hasta obtener una imagen limpia lo suficientemente robusta a la original.

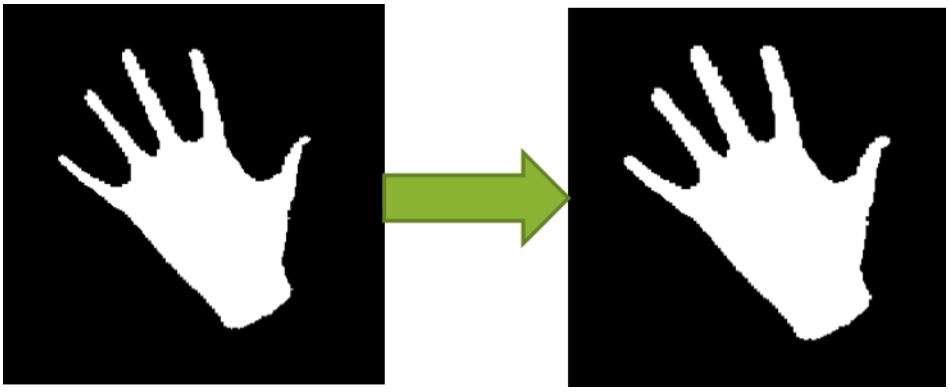
Figura 5.9 Primera operación morfológica a la imagen segmentada (Erosión)



Fuente: Autor

La segunda operación aplicada es la dilatacion, con el mismo elemento estructurador, generando una imagen más robusta y sin ruido.

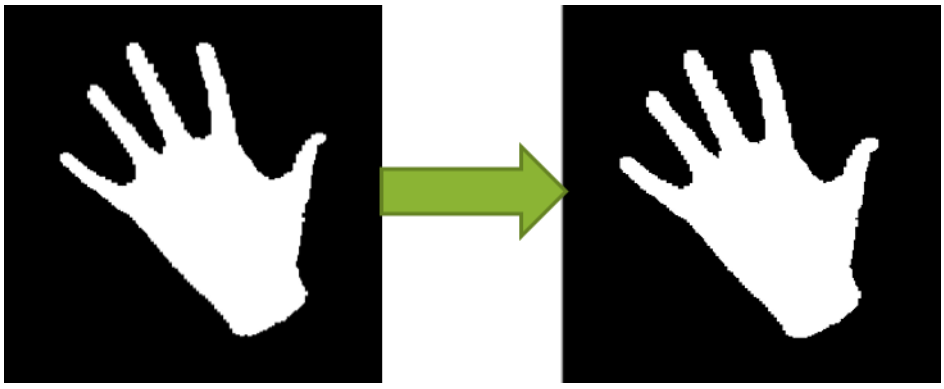
Figura 5.10 Segunda operación morfológica a la imagen segmentada (Dilatación)



Fuente: Autor

Para finalizar, se realiza una nueva dilatación de la imagen, y como resultado se obtiene una imagen totalmente limpia, lista para extraer características de ella.

Figura 5.11 Tercera operación morfológica a la imagen segmentada (Dilatación)



Fuente: Autor

5.4.3 Creación de la base de datos de imágenes

Teniendo las imágenes procesadas es necesario crear una base de datos, donde se guardarán 25 imágenes por gesto del alfabeto ASL, incluyendo números.

Figura 5.12 Creación de la base de datos de imágenes

Fuente: Autor

Estas imágenes se toman en diferentes posiciones, ángulos y escalas, con el fin de tener una base de datos diversa, para que a la hora del entrenamiento de la RNA pueda reconocer las variaciones dentro de un mismo patrón de imágenes.

5.5 DESARROLLO OBJETIVO ESPECÍFICO 3

- Implementar por medio de Redes Neuronales Artificiales la máquina de reconocimiento.

Actividades

- Extracción de las características de las imágenes.
- Diseño de la Red Neuronal Artificial.
- Entrenamiento de la Red Neuronal Artificial.

5.5.1 Extracción de características de imágenes

Para la extracción de características (momentos invariantes de HU), es necesario poseer el contorno de la imagen y procesarlo.

```
void Momentos(string imagen){
      Mat image = imread(imagen);              //Crea una matriz image, donde
                                               leerá la imagen guardada
      src = image;

  cvtColor( src, src_gray, CV_BGR2GRAY );      //Filtra la imagen
  blur( src_gray, src_gray, Size(3,3) );       //Suaviza la imagen

  Mat canny_output;
  vector<vector<Point> > contours;
  vector<Vec4i> hierarchy;

  /// Detecta bordes
  Canny( src_gray, canny_output, thresh, thresh*2, 3 );

  /// Busca Contornos
  findContours( canny_output, contours, hierarchy, CV_RETR_TREE,
CV_CHAIN_APPROX_SIMPLE, Point(0, 0) );
```

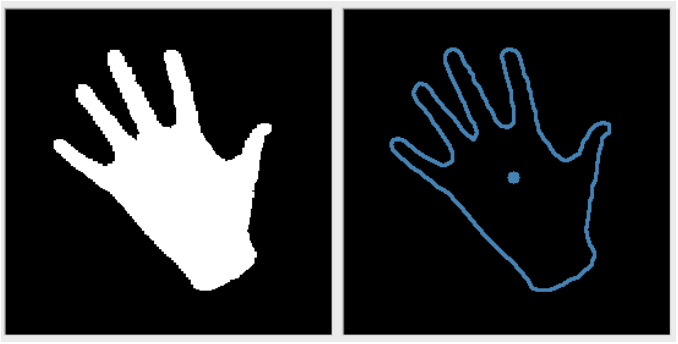
El segmento anterior de código muestra la extracción del contorno, necesario para la extraer los momentos de HU con la ayuda de OpenCV.

En esta parte se lee la imagen y se hallan los contornos existentes, los cuales serán guardados en un vector llamado contours. Diríjase al Anexo B.

```
  ///IF de control para reconocer si hay contornos o no los hay
  if (contours.size()>0){

  //CONTORNO MAYOR
  int Mayor, Cont1, n=0;
  Cont1=contourArea(contours[0]);
  for( int i = 0; i < contours.size(); i++ )
     {
               Mayor=contourArea(contours[i]);
               if (Mayor > Cont1){
                      n=i;
                      Cont1=contourArea(contours[i]);
               }
     }
  //printf("\n El Contorno Mayor Es El Numero %d  \n",n);
```

El código anterior evalúa los contornos existentes y escoge el mayor, para poder extraer de este los momentos.

Figura 5.13 Imagen segmentada y contorno de imagen



Fuente: Autor

Los momentos de HU son almacenados en un vector de siete posiciones, los cuales se imprimen, se clasifican y se guardan para usarlos en el entrenamiento.

```
  ////////////////////MOMENTOS//////////////////////
  vector<Moments> mu(contours.size() );  //Vector mu par los Momentos

  mu[n] = moments( contours[n], true ); //Extrae los Momentos de la imagen

  double h[7];       //Arreglo de 7 posiciones para almacenar los 7 Momentos de HU
  HuMoments(mu[n],h);

  //printf("\n \t MOMENTOS DE HU \n {%.20f, %.20f, %.20f, %.20f, %.20f, %.20f,
%.20f},",h[0],h[1],h[2],h[3],h[4],h[5],h[6]);  //Siete Momentos de HU
  printf("\n{%.20f, %.20f, %.20f, %.20f},",h[0],h[1],h[2],h[3]);  //Momentos
1,2,3,4
```

En respuesta al código anterior se presenta en la pantalla los valores de los momentos de HU, estos valores serán transcritos y almacenados para usarlos en el proceso de entrenamiento.

Figura 5.14 Siete momentos invariantes de HU

```
        MOMENTOS DE HU
[1] 0.22295144611261952000
[2] 0.01026180634947908100
[3] 0.00022062275720190262
[4] 0.00020571700029011329
[5] -0.00000001873267897545
[6] 0.00002059445838828256
[7] -0.00000003962060798682
```

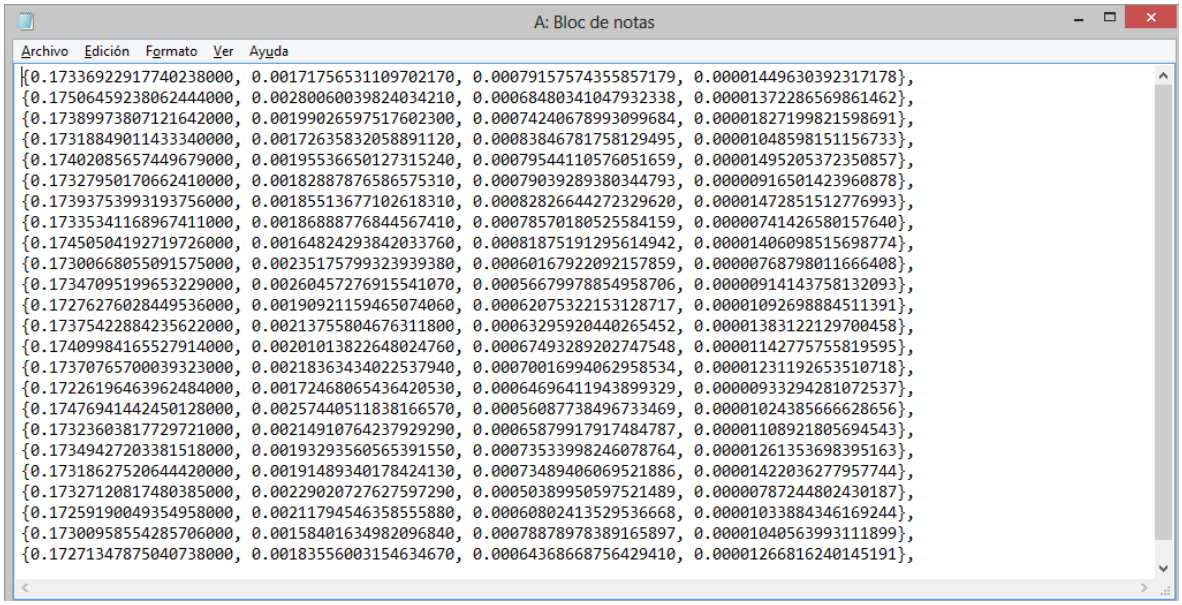
Fuente: Autor

5.5.2 Creación de la base de datos de patrones

Los momentos de menor orden muestran mayor información, es por esto que se usan los primeros cuatro momentos. Los momentos cinco, seis y siete a pesar de ser invariantes son más sensible a pequeños cambios de la imagen, es por esto que no se usan en el entrenamiento.

Los primeros cuatro momentos invariantes de HU son organizados en archivos .txt, con estructura de vectores de cuatro posiciones, para ser agregados al código de entrenamiento de la RNA.

Figura 5.15 Momentos de HU obtenidos de la letra A



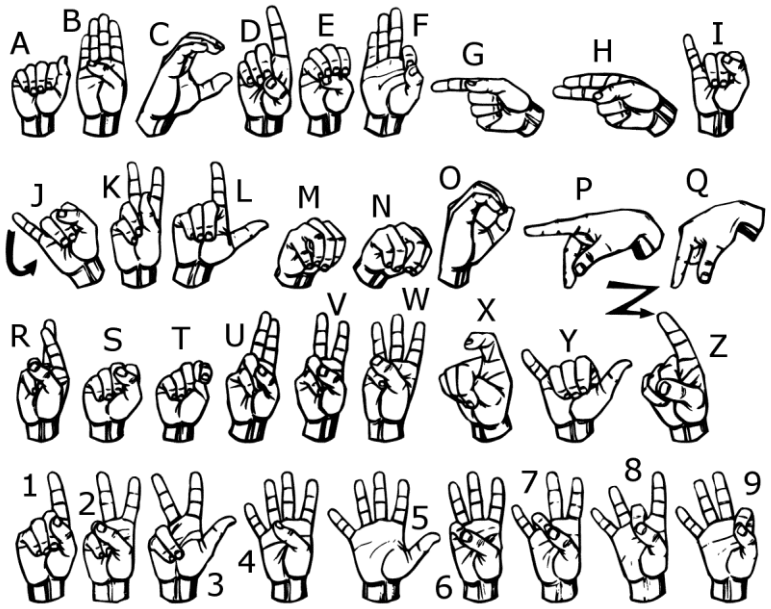
Fuente: Autor

5.5.3 Clasificación de imágenes

Como los patrones a reconocer pertenecen al alfabeto de señas Americano, estos se han agrupado en carpetas independientes, y se han extraído todos sus momentos.

A continuación se mostrarán las señas del alfabeto ASL y la distribución de sus patrones en una gráfica de dispersión.

Figura 5.16 Alfabeto de señas Americano (ASL)



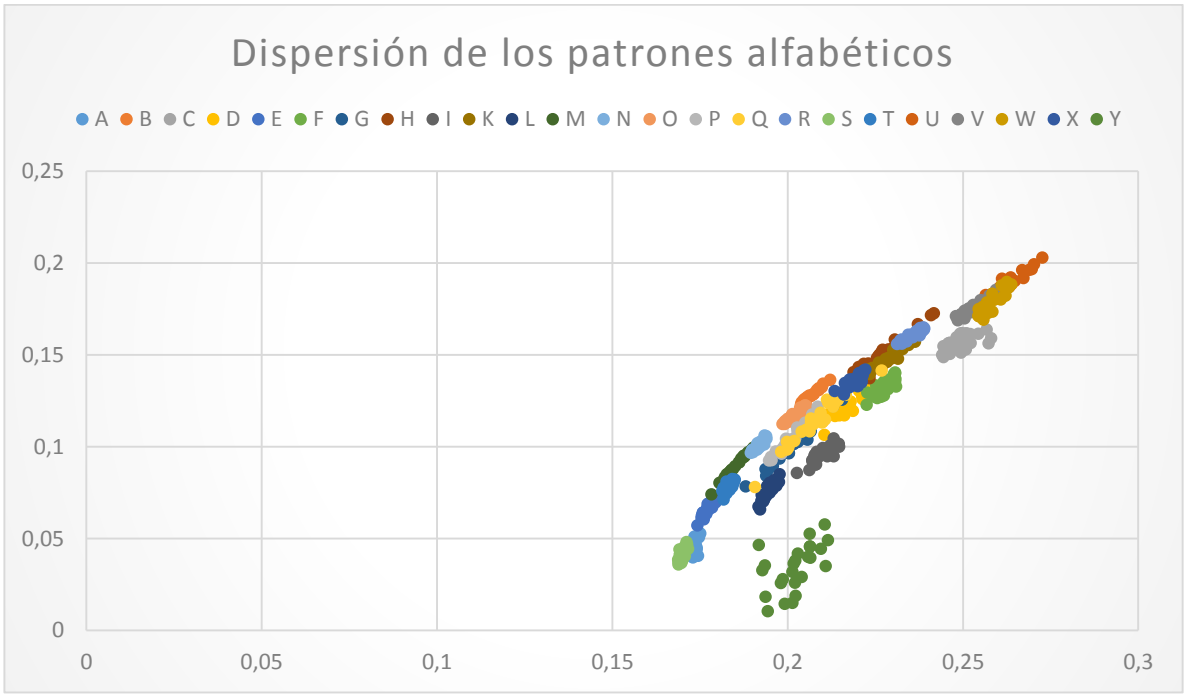
Fuente: Lifeprint.com. Sign Language. Disponible en internet: http://lifeprint.com/asl101/topics/wallpaper1.htm

Para poder evidenciar la convergencia de los patrones extraídos, se grafican en un plano bidimensional donde los puntos X y Y son:

$$X = \eta_{20} + \eta_{02}$$

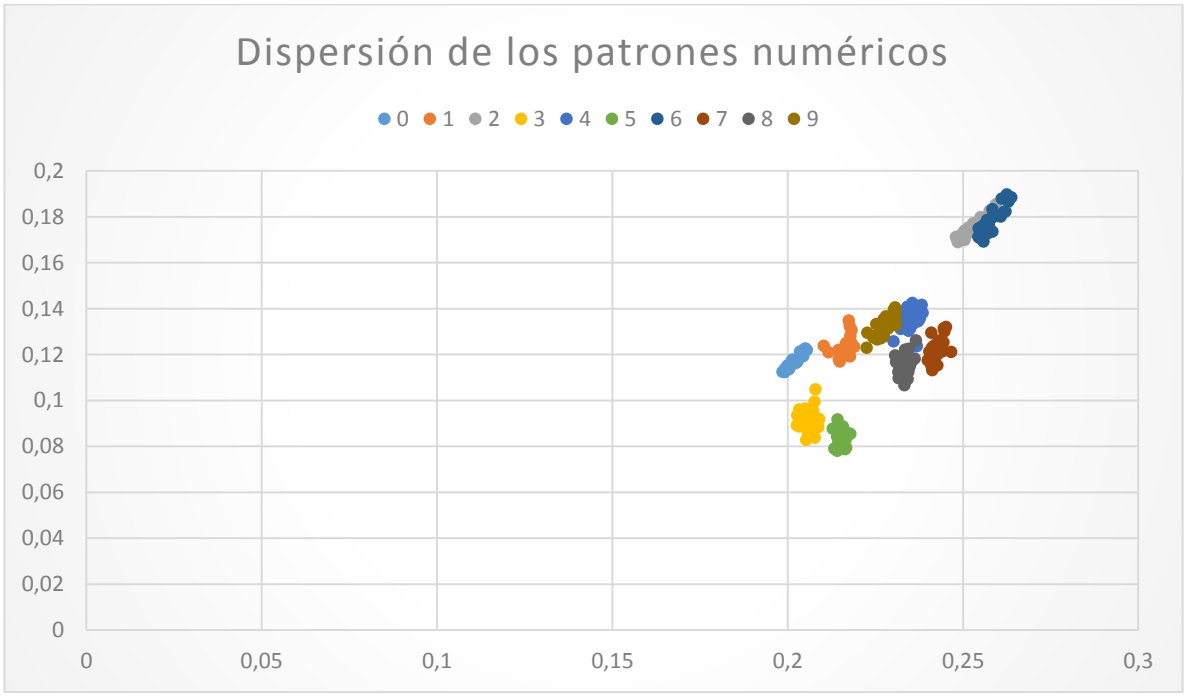
$$Y = \sqrt{(\eta_{20} - \eta_{02})^2 + 4\eta^2{}_{11}}$$

Figura 5.17 Dispersión de los patrones del alfabeto ASL



Fuente: Autor

Figura 5.18 Dispersión de los patrones numéricos del alfabeto ASL

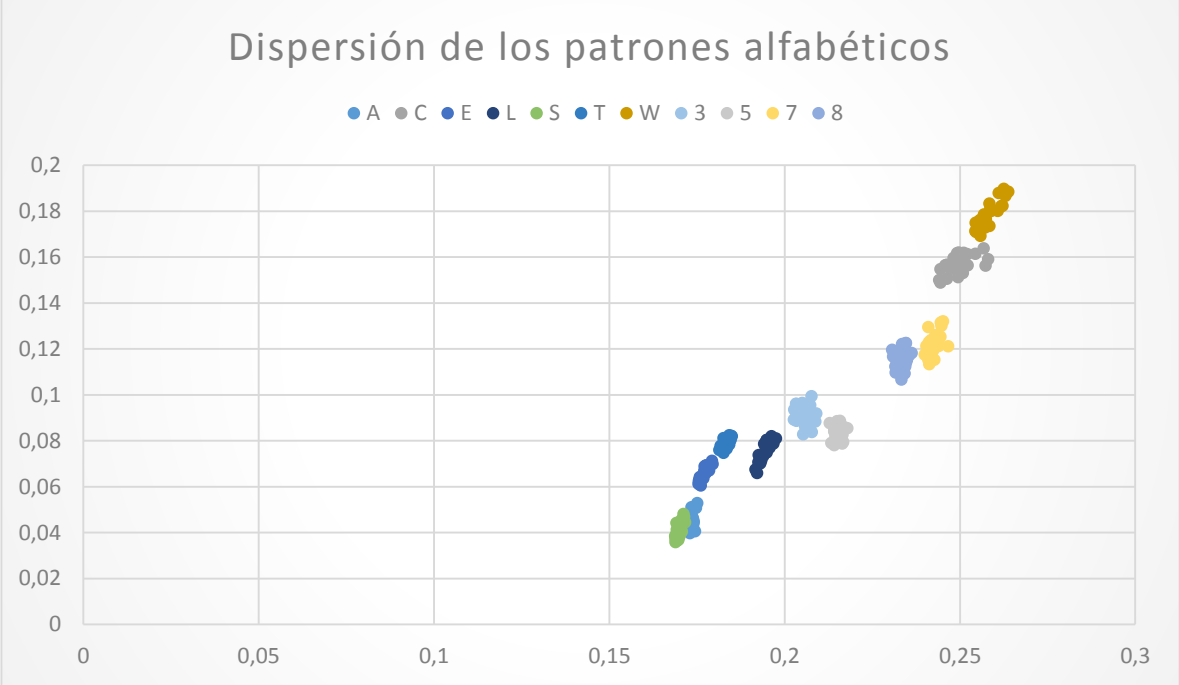


Fuente: Autor

Debido a que muchos de patrones que describen el alfabeto ASL se solapan, se decide reconocer los que son linealmente separables y estén más separados entre sí.

A continuación se muestran los patrones escogidos para el reconocimiento.

Figura 5.19 Dispersión de los patrones gestuales ASL a reconocer



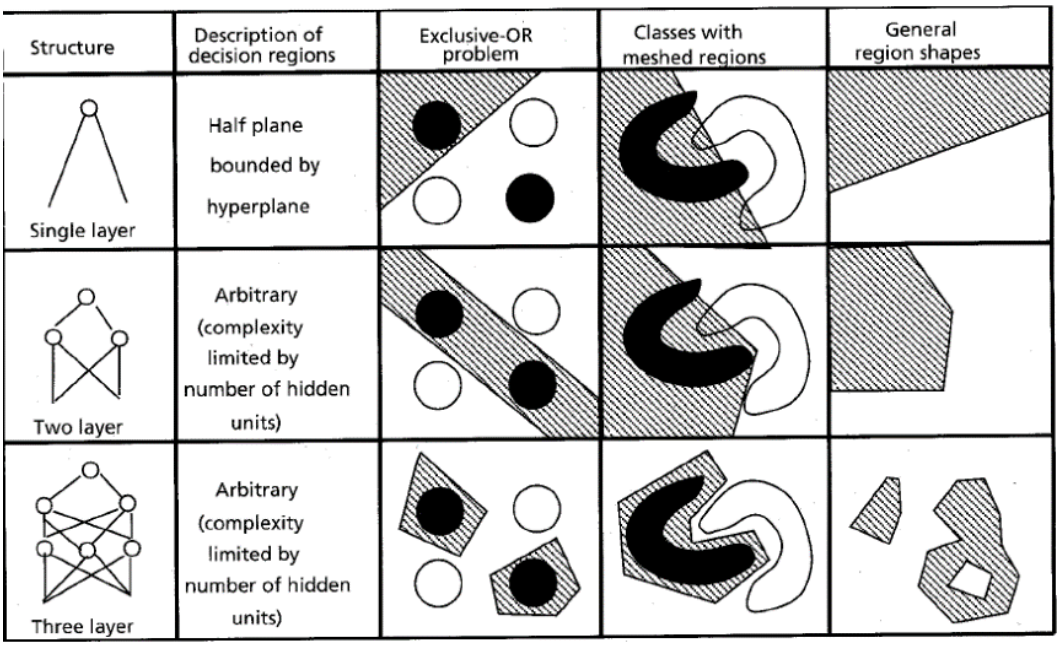
Fuente: Autor

5.5.4 Diseño de la Red Neuronal Artificial

El diseño de una RNA se hace de manera experimental, pero estudios en esta área han demostrado que hay relación entre el número de capas ocultas y su cantidad de neuronas con el ajuste de regiones en un plano bidimensional.

La figura 5.19 muestra esta relación, donde al aplicar una mayor cantidad de neuronas junto con capas ocultas delimitará el área de clasificación, dando mejores resultados al reconocimiento de patrones.

Figura 5.20 Interpretación geométrica del rol de las capas ocultas en un espacio bidimensional

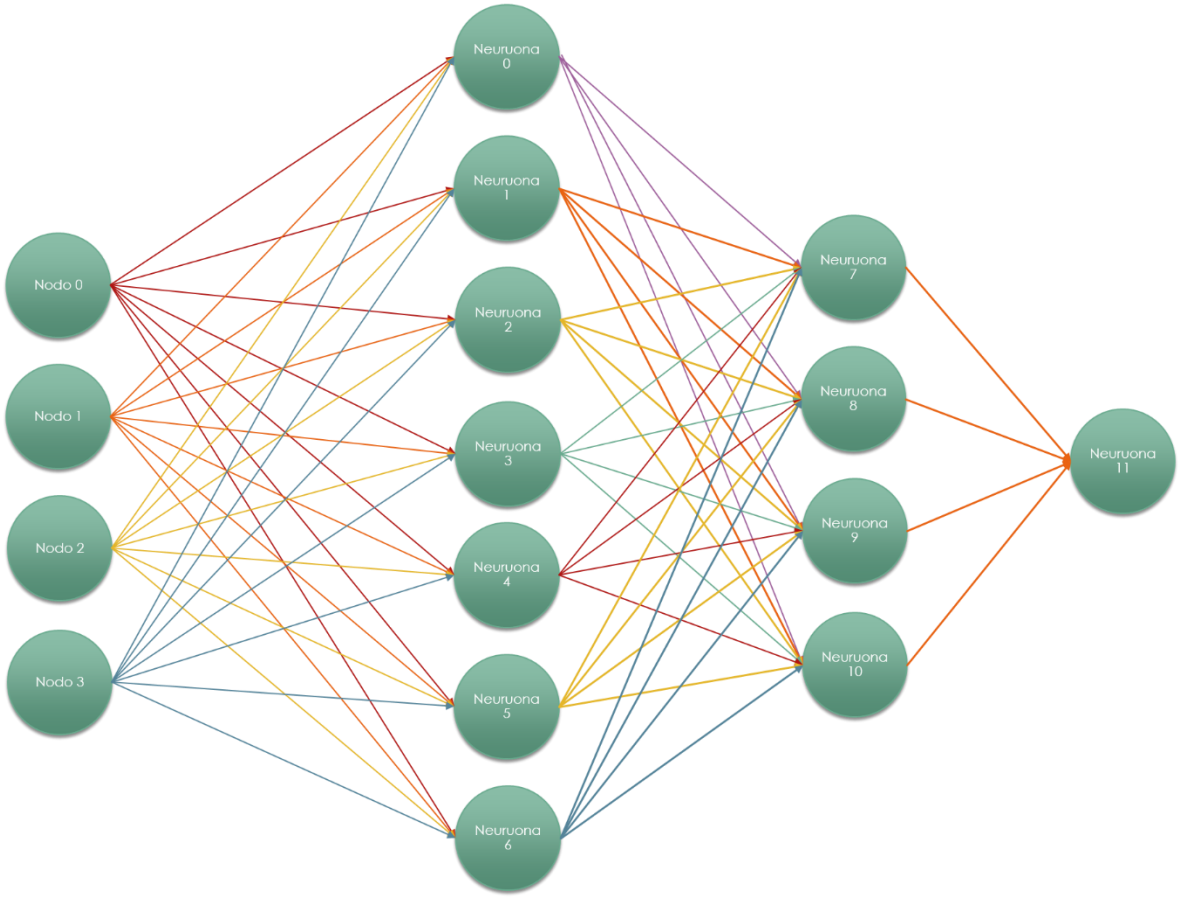


Fuente: ANIL K., Jain; JIANCHANG, Mao y K.M., Mohiudin. Artificial Neural Networks: A Tutorial, 1996 [base de datos en línea]. [Citado en 20 enero de 2014]. Disponible en biblioteca digital IEEE Xplore®.

Mediante este método experimental, se concluyó que una red de cuatro capas era suficiente para el reconocimiento de los patrones anteriormente expuestos, con cuatro nodos en la capa de entrada, siete neuronas en la primera capa oculta, cuatro neuronas en la segunda capa oculta y una neurona en la capa de salida, como se muestra en la figura 5.21.

Se usaron solo cuatro nodos de entrada, porque son los primeros cuatro momentos de HU los que poseen la información más relevante de los gestos.

Figura 5.21 RNA implementada



Fuente: Autor

Al ver que muchas neuronas repiten sus métodos de operación, se crea una función para las neuronas de la primera capa oculta con ocho entradas, y otra función para las neuronas de la segunda capa oculta de 14 entradas.

A continuación se muestran los apartes de las funciones de la primera y segunda capa oculta de neuronas.

```
//Función del modelo de Perceptrón de la primera capa (Neuronas)----w(pesos de la
conexiones sinápticas),--pat(patrones de entrada)
double Perceptron1C(double w0,double w1,double w2,double w3,double pat0,double
                    pat1,double pat2,double pat3){
       double outPer1C;           //Variable para la operación de "Salida del
                                  Perceptrón" outPer
       x = (w0*pat0)+(w1*pat1)+(w2*pat2)+(w3*pat3);    //Operación Neuronal
                                                 (Comportamiento de la Neurona)
       outPer1C=sigm(x);
       //Respuesta Neuronal, con la función de excitación tipo Sigmoidea Bipolar
       return outPer1C;
       //Retorna respuesta
}

//Función del modelo de Perceptrón de la segunda capa (Neuronas)----w(pesos de la
conexiones sinápticas),--sal(patrones de entrada)
double Perceptron2C(double w0,double w1,double w2,double w3,double w4,double
w5,double w6,double sal0,double sal1,double sal2,double sal3,double sal4,double
sal5,double sal6){
       double outPer2C;           //Variable para la operación de "Salida del
                                  Perceptrón" outPer
       x = (w0*sal0)+(w1*sal1)+(w2*sal2)+(w3*sal3)+(w4*sal4)+(w5*sal5)+(w6*sal6);
       //Operación Neuronal (Comportamiento de la Neurona)
       outPer2C=sigm(x);                          //Respuesta Neuronal, con
                                  la función de excitación tipo Sigmoidea Bipolar
       return outPer2C;                           //Retorna respuesta
}
```

La función sigmoidea bipolar corresponde a la función de excitación de cada una de las neuronas de la red, con un valor máximo de 1 y mínimo de -1.

A continuación se describe esta función en lenguaje C++.

```
////////////////Función de activación de las neuronas (Sigmoidea)/////////////////
double sigm(double s){
        double sig;                    //Variable de operación función Sigmoide
        sig=-1+(2/(1+(exp(-1*s))));    //Función de activación de la Neurona
                                       (Sigmoide bipolar Max=1; Min=-1)
        return sig;
}
```

Para obtener mayor detalle de los códigos dirigirse al anexo C.

5.5.5 Entrenamiento de la Red Neuronal Artificial

El desarrollo del algoritmo de entrenamiento opera con una serie de funciones adicionales, como la Derivsigm, que describe la derivada de la función sigmoidea, y el RecalPesos, que calcula el error con respecto a la salida deseada y calcula los nuevos valores de pesos sinápticos.

A continuación se describirán los códigos.

El código de la derivada de la función de activación de las neuronas, es fundamental para la actualización de los pesos sinápticos en el proceso de aprendizaje, como se vio en la sección anterior.

```
////Función que opera con la Derivada de la función Sigmoidea////////
double Dervsigm(double der){
	double deri;		//Variable de operación Derivada de Sigmoidea
	deri=(2*(exp(-1*der)))/(pow((1+(exp(-1*der))),2));//Función Derivada de
                                                          Sigmoidea
	return deri;
}
```

Ahora, para calcular los nuevos valores de pesos sinápticos, hay que propagar el error desde la capa de salida hasta la primera capa oculta de neuronas, como se muestra a continuación.

```
void RecalPesos(){

	////DIFERENCIAS O ERROR DE LA SEÑAL////

	d[11]=SalEsp[y][0]-Salida[11];	//Error de la Neurona 11. Se comparó con
                                        la Salida Esperada
	//Propagación del error hacia atrás
	d[7]=d[11]*Pesos[56];		//Error de las Neuronas de la Segunda Capa
                                        Oculta.
	d[8]=d[11]*Pesos[57];
	d[9]=d[11]*Pesos[58];
	d[10]=d[11]*Pesos[59];
	//Propagación del error hacia atrás
	d[0]=d[7]*Pesos[28] + d[8]*Pesos[29] + d[9]*Pesos[30] + d[10]*Pesos[31];
	//Error de las Neuronas de la Primera Capa Oculta.
	d[1]=d[7]*Pesos[32] + d[8]*Pesos[33] + d[9]*Pesos[34] + d[10]*Pesos[35];
	d[2]=d[7]*Pesos[36] + d[8]*Pesos[37] + d[9]*Pesos[38] + d[10]*Pesos[39];
	d[3]=d[7]*Pesos[40] + d[8]*Pesos[41] + d[9]*Pesos[42] + d[10]*Pesos[43];
	d[4]=d[7]*Pesos[44] + d[8]*Pesos[45] + d[9]*Pesos[46] + d[10]*Pesos[47];
	d[5]=d[7]*Pesos[48] + d[8]*Pesos[49] + d[9]*Pesos[50] + d[10]*Pesos[51];
	d[6]=d[7]*Pesos[52] + d[8]*Pesos[53] + d[9]*Pesos[54] + d[10]*Pesos[55];
```

A partir del error propagado hacia todas las demás neuronas, se proceden a actualizar los pesos sinápticos de la primera capa oculta como se mostró en el capítulo anterior.

```
////////PRIMERA CAPA OCULTA//////////CORRECCION DE PESOS SINAPTICOS/////////

      Pesos[0]=Pesos[0]+(e*d[0]*Dervsigm(Entradas [i][0])*Entradas [i][0]);
      Pesos[7]=Pesos[7]+(e*d[0]*Dervsigm(Entradas [i][1])*Entradas [i][1]);
      Pesos[14]=Pesos[14]+(e*d[0]*Dervsigm(Entradas [i][2])*Entradas [i][2]);
      Pesos[21]=Pesos[21]+(e*d[0]*Dervsigm(Entradas [i][3])*Entradas [i][3]);

      Pesos[1]=Pesos[1]+(e*d[1]*Dervsigm(Entradas [i][0])*Entradas [i][0]);
      Pesos[8]=Pesos[8]+(e*d[1]*Dervsigm(Entradas [i][1])*Entradas [i][1]);
      Pesos[15]=Pesos[15]+(e*d[1]*Dervsigm(Entradas [i][2])*Entradas [i][2]);
      Pesos[22]=Pesos[22]+(e*d[1]*Dervsigm(Entradas [i][3])*Entradas [i][3]);

      Pesos[2]=Pesos[2]+(e*d[2]*Dervsigm(Entradas [i][0])*Entradas [i][0]);
      Pesos[9]=Pesos[9]+(e*d[2]*Dervsigm(Entradas [i][1])*Entradas [i][1]);
      Pesos[16]=Pesos[16]+(e*d[2]*Dervsigm(Entradas [i][2])*Entradas [i][2]);
      Pesos[23]=Pesos[23]+(e*d[2]*Dervsigm(Entradas [i][3])*Entradas [i][3]);

      Pesos[3]=Pesos[3]+(e*d[3]*Dervsigm(Entradas [i][0])*Entradas [i][0]);
      Pesos[10]=Pesos[10]+(e*d[3]*Dervsigm(Entradas [i][1])*Entradas [i][1]);
      Pesos[17]=Pesos[17]+(e*d[3]*Dervsigm(Entradas [i][2])*Entradas [i][2]);
      Pesos[24]=Pesos[24]+(e*d[3]*Dervsigm(Entradas [i][3])*Entradas [i][3]);

      Pesos[4]=Pesos[4]+(e*d[4]*Dervsigm(Entradas [i][0])*Entradas [i][0]);
      Pesos[11]=Pesos[11]+(e*d[4]*Dervsigm(Entradas [i][1])*Entradas [i][1]);
      Pesos[18]=Pesos[18]+(e*d[4]*Dervsigm(Entradas [i][2])*Entradas [i][2]);
      Pesos[25]=Pesos[25]+(e*d[4]*Dervsigm(Entradas [i][3])*Entradas [i][3]);

      Pesos[5]=Pesos[5]+(e*d[5]*Dervsigm(Entradas [i][0])*Entradas [i][0]);
      Pesos[12]=Pesos[12]+(e*d[5]*Dervsigm(Entradas [i][1])*Entradas [i][1]);
      Pesos[19]=Pesos[19]+(e*d[5]*Dervsigm(Entradas [i][2])*Entradas [i][2]);
      Pesos[26]=Pesos[26]+(e*d[5]*Dervsigm(Entradas [i][3])*Entradas [i][3]);

      Pesos[6]=Pesos[6]+(e*d[6]*Dervsigm(Entradas [i][0])*Entradas [i][0]);
      Pesos[13]=Pesos[13]+(e*d[6]*Dervsigm(Entradas [i][1])*Entradas [i][1]);
      Pesos[20]=Pesos[20]+(e*d[6]*Dervsigm(Entradas [i][2])*Entradas [i][2]);
      Pesos[27]=Pesos[27]+(e*d[6]*Dervsigm(Entradas [i][3])*Entradas [i][3]);
```

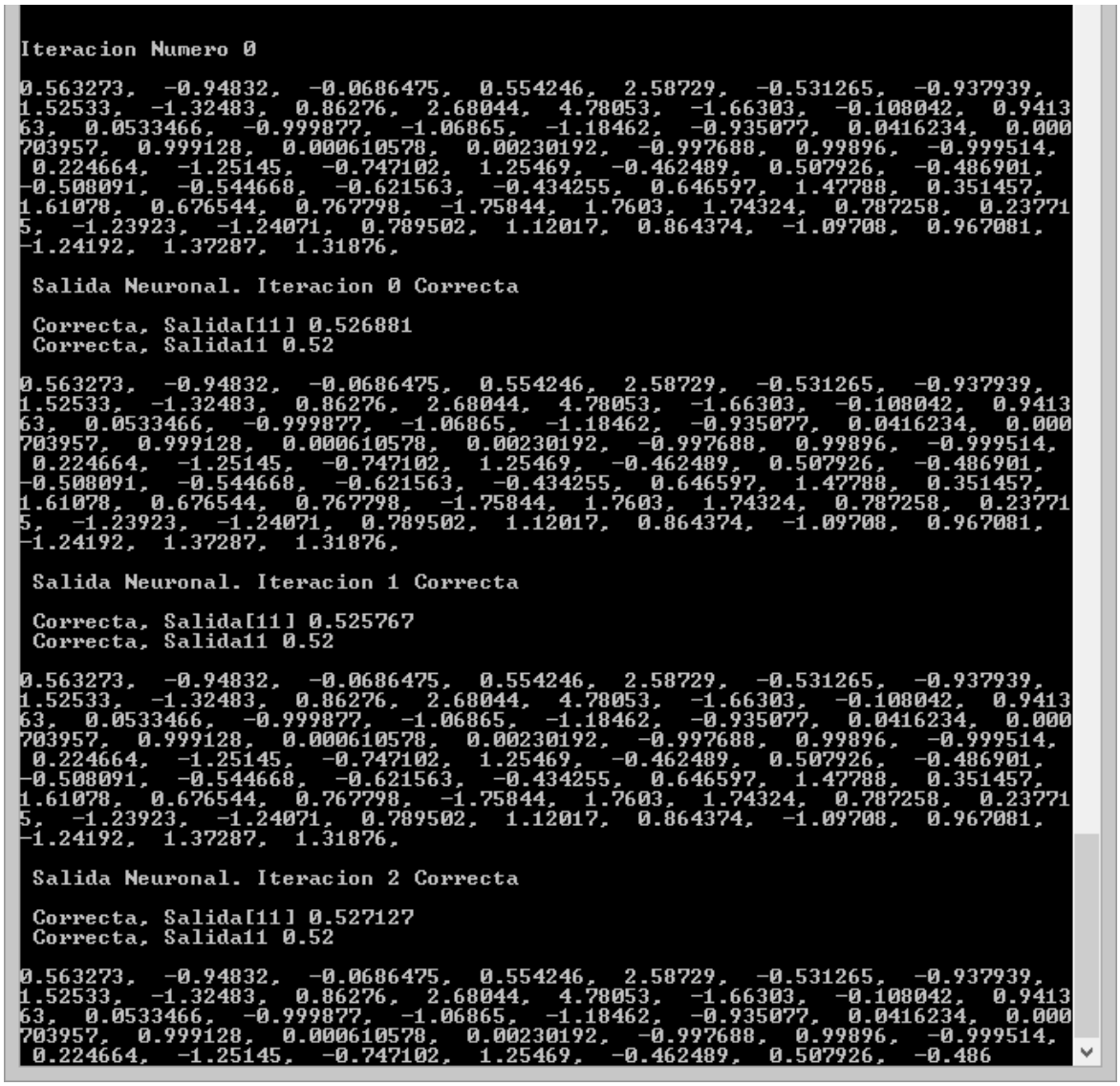
Y seguido se actualizarán los pesos sinápticos de las neuronas de la capa oculta número dos y la neurona de salida.

```
///////SEGUNDA CAPA OCULTA//////////CORRECCION DE PESOS SINAPTICOS/////////

      Pesos[28]=Pesos[28]+(e*d[7]*Dervsigm(Salida[0])*Salida[0]);
      Pesos[32]=Pesos[32]+(e*d[7]*Dervsigm(Salida[1])*Salida[1]);
      Pesos[36]=Pesos[36]+(e*d[7]*Dervsigm(Salida[2])*Salida[2]);
      Pesos[40]=Pesos[40]+(e*d[7]*Dervsigm(Salida[3])*Salida[3]);
      Pesos[44]=Pesos[44]+(e*d[7]*Dervsigm(Salida[4])*Salida[4]);
      Pesos[48]=Pesos[48]+(e*d[7]*Dervsigm(Salida[5])*Salida[5]);
      Pesos[52]=Pesos[52]+(e*d[7]*Dervsigm(Salida[6])*Salida[6]);

      Pesos[29]=Pesos[29]+(e*d[8]*Dervsigm(Salida[0])*Salida[0]);
      Pesos[33]=Pesos[33]+(e*d[8]*Dervsigm(Salida[1])*Salida[1]);
      Pesos[37]=Pesos[37]+(e*d[8]*Dervsigm(Salida[2])*Salida[2]);
      Pesos[41]=Pesos[41]+(e*d[8]*Dervsigm(Salida[3])*Salida[3]);
      Pesos[45]=Pesos[45]+(e*d[8]*Dervsigm(Salida[4])*Salida[4]);
      Pesos[49]=Pesos[49]+(e*d[8]*Dervsigm(Salida[5])*Salida[5]);
      Pesos[53]=Pesos[53]+(e*d[8]*Dervsigm(Salida[6])*Salida[6]);

      Pesos[30]=Pesos[30]+(e*d[9]*Dervsigm(Salida[0])*Salida[0]);
      Pesos[34]=Pesos[34]+(e*d[9]*Dervsigm(Salida[1])*Salida[1]);
      Pesos[38]=Pesos[38]+(e*d[9]*Dervsigm(Salida[2])*Salida[2]);
      Pesos[42]=Pesos[42]+(e*d[9]*Dervsigm(Salida[3])*Salida[3]);
      Pesos[46]=Pesos[46]+(e*d[9]*Dervsigm(Salida[4])*Salida[4]);
      Pesos[50]=Pesos[50]+(e*d[9]*Dervsigm(Salida[5])*Salida[5]);
      Pesos[54]=Pesos[54]+(e*d[9]*Dervsigm(Salida[6])*Salida[6]);

      Pesos[31]=Pesos[31]+(e*d[10]*Dervsigm(Salida[0])*Salida[0]);
      Pesos[35]=Pesos[35]+(e*d[10]*Dervsigm(Salida[1])*Salida[1]);
      Pesos[39]=Pesos[39]+(e*d[10]*Dervsigm(Salida[2])*Salida[2]);
      Pesos[43]=Pesos[43]+(e*d[10]*Dervsigm(Salida[3])*Salida[3]);
      Pesos[47]=Pesos[47]+(e*d[10]*Dervsigm(Salida[4])*Salida[4]);
      Pesos[51]=Pesos[51]+(e*d[10]*Dervsigm(Salida[5])*Salida[5]);
      Pesos[55]=Pesos[55]+(e*d[10]*Dervsigm(Salida[6])*Salida[6]);


////////CAPA DE SALIDA //////////////CORRECCION DE PESOS SINAPTICOS/////////

      Pesos[56]=Pesos[56]+(e*d[11]*Dervsigm(Salida[7])*Salida[7]);
      Pesos[57]=Pesos[57]+(e*d[11]*Dervsigm(Salida[8])*Salida[8]);
      Pesos[58]=Pesos[58]+(e*d[11]*Dervsigm(Salida[9])*Salida[9]);
      Pesos[59]=Pesos[59]+(e*d[11]*Dervsigm(Salida[10])*Salida[10]);


}
```

El proceso de entrenamiento es sencillo:

1. Se agregan valores aleatorios entre -1 y 1 a todos los pesos sinápticos.
2. Se agregan los patrones a aprender a los cuatro nodos de entrada.
3. Se ordena que todas las neuronas realicen sus operaciones correspondientes, teniendo en cuenta sus valores de entrada y sus respectivos pesos sinápticos.
4. Se compara el valor obtenido por la red con el valor esperado.
5. Si coincide se compara el siguiente patrón.

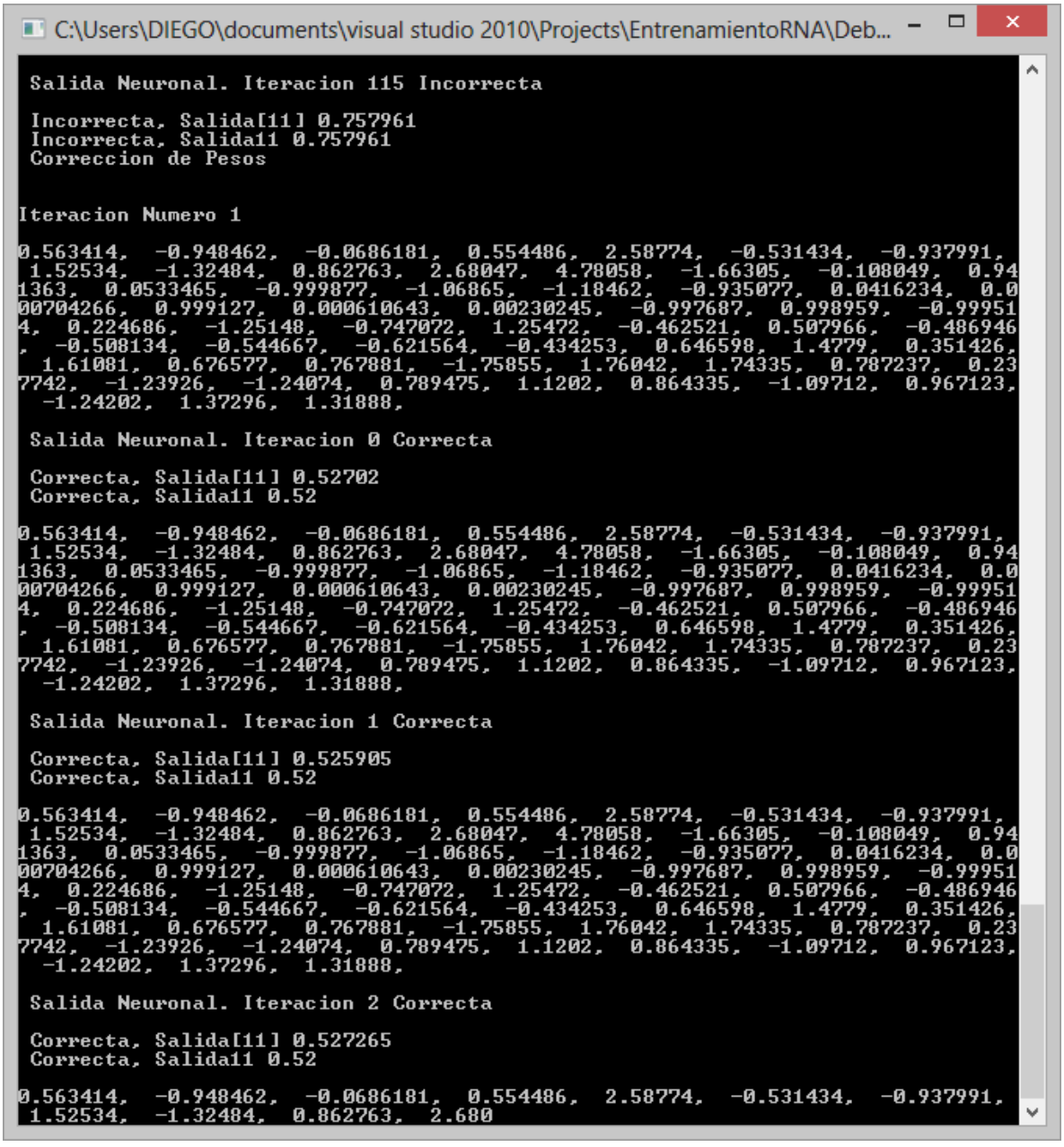
Figura 5.22 Entrenamiento RNA (Aciertos)



Fuente: Autor

6. Si la respuesta de la red no coincide con la esperada se propaga el error hasta la primera capa oculta de neuronas.
7. Se actualizan pesos sinápticos.
8. Inicia nuevamente el proceso de entrenamiento desde el patrón número 1 hasta que todas las respuestas coincidan con las esperadas.

Figura 5.23 Entrenamiento RNA (Fracasos)



Fuente: Autor

Figura 5.24 Entrenamiento RNA (Concluido)

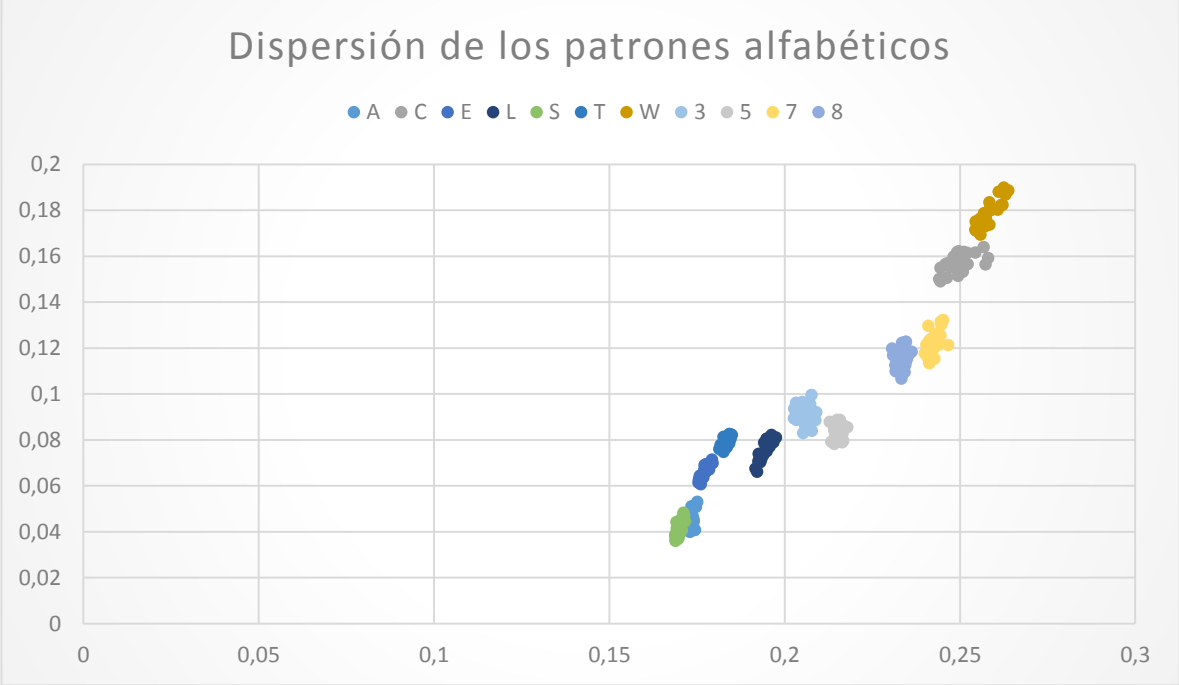
Fuente: Autor

6. RESULTADOS

A partir de la red neuronal artificial empleada se empieza el entrenamiento patrón a patrón, siendo estos los más separados unos de otros para evitar contradicciones en el entrenamiento.

A continuación se muestra en la figura 6.1 la distribución de los patrones a reconocer en este proyecto.

Figura 6.1 Dispersión de los patrones gestuales ASL



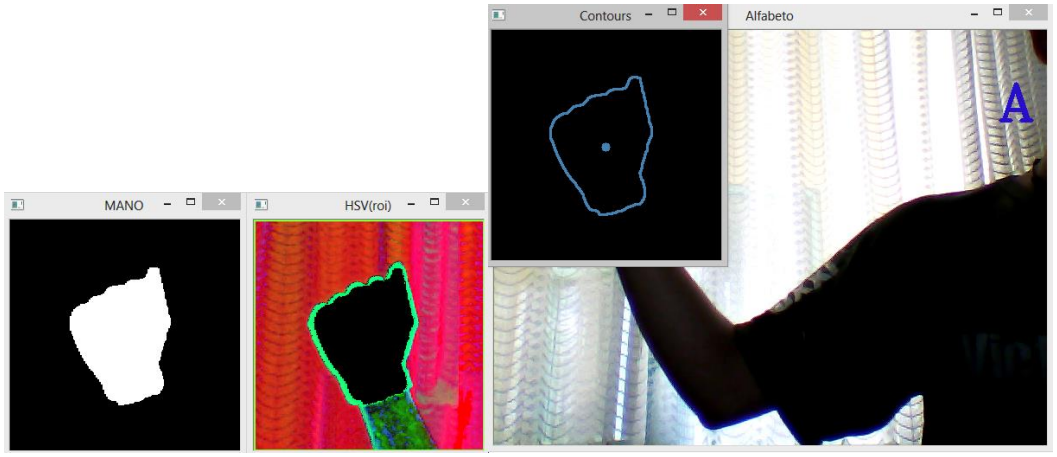
Fuente: Autor

Posterior al entrenamiento se lograron identificar los patrones anteriores, y los valores de los pesos sinápticos definitivos son los siguientes:

{0.563414, -0.948462, -0.0686181, 0.554486, 2.58774, -0.531434, -0.937991, 1.52534, -1.32484, 0.862763, 2.68047, 4.78058, -1.66305, -0.108049, 0.941363, 0.0533465, -0.999877, -1.06865, -1.18462,-0.935077, 0.0416234, 0.000704266, 0.999127, 0.000610643, 0.00230245, -0.997687, 0.998959, -0.999514, 0.224686, -1.25148, -0.747072, 1.25472, -0.462521, 0.507966, -0.486946, -0.508134, -0.544667, -0.621564, -0.434253, 0.646598, 1.4779, 0.351426, 1.61081, 0.676577, 0.767881, -1.75855, 1.76042, 1.74335, 0.787237, 0.237742, -1.23926, -1.24074, 0.789475, 1.1202, 0.864335, -1.09712, 0.967123, -1.24202, 1.37296, 1.31888}

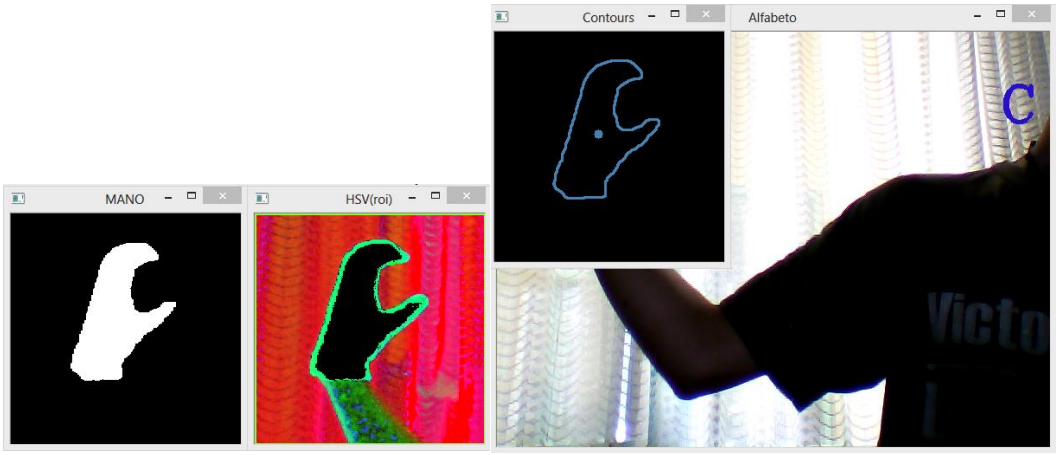
A continuación se muestran los resultados del entrenamiento.

Figura 6.2 Reconocimiento gestual letra A



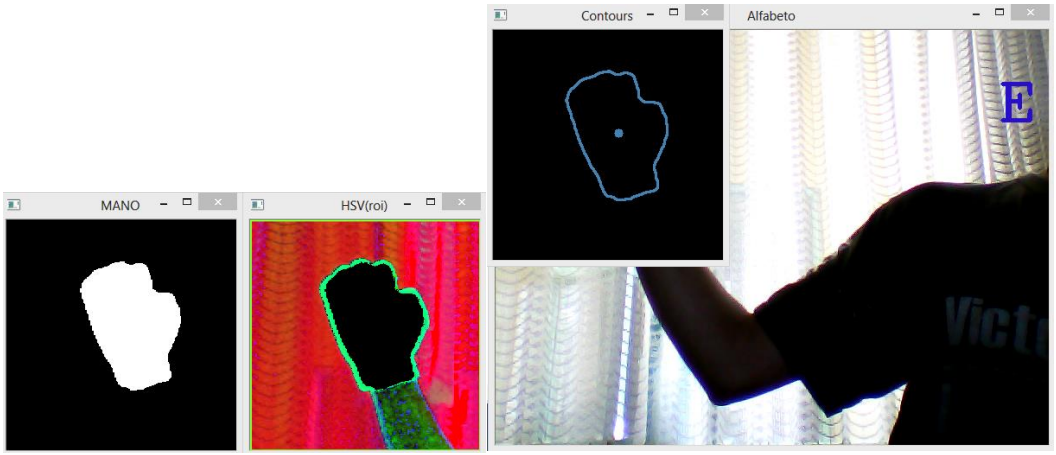
Fuente: Autor

Figura 6.3 Reconocimiento gestual letra C



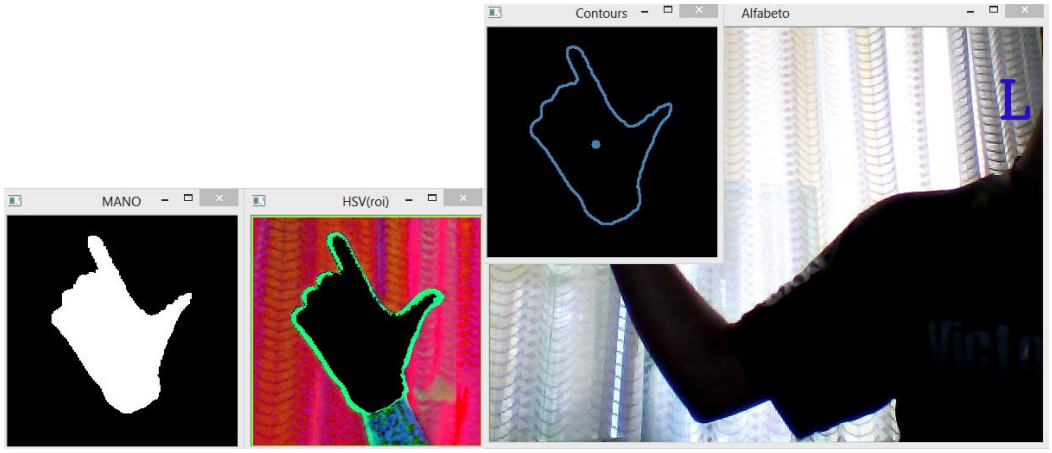
Fuente: Autor

Figura 6.4 Reconocimiento gestual letra E



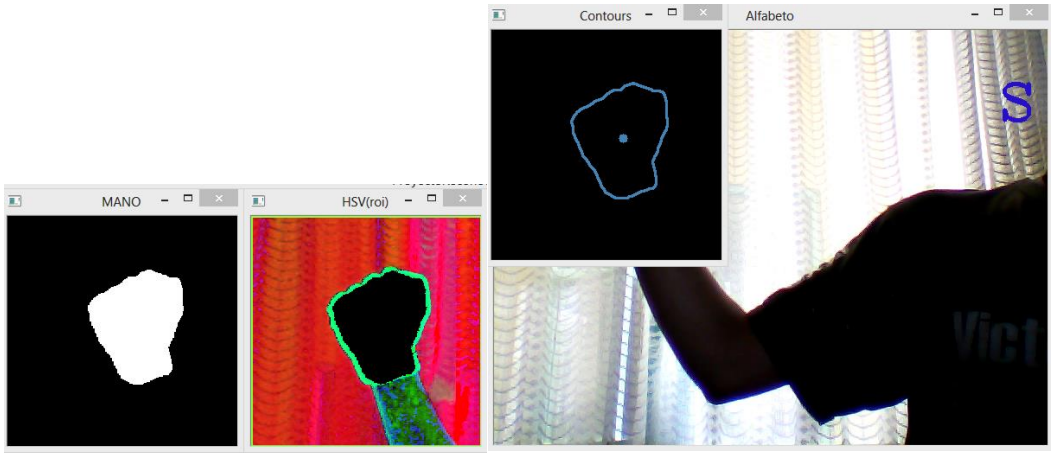
Fuente: Autor

Figura 6.5 Reconocimiento gestual letra L



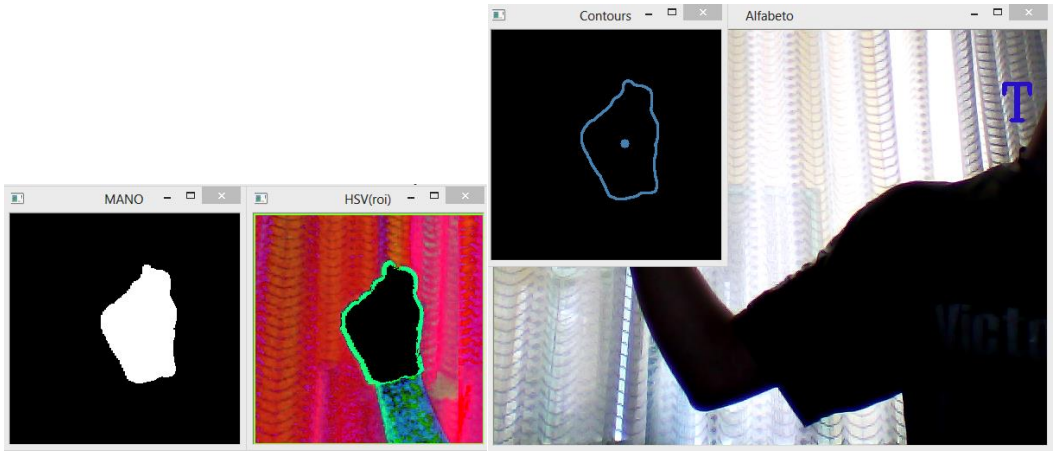
Fuente: Autor

Figura 6.6 Reconocimiento gestual letra S



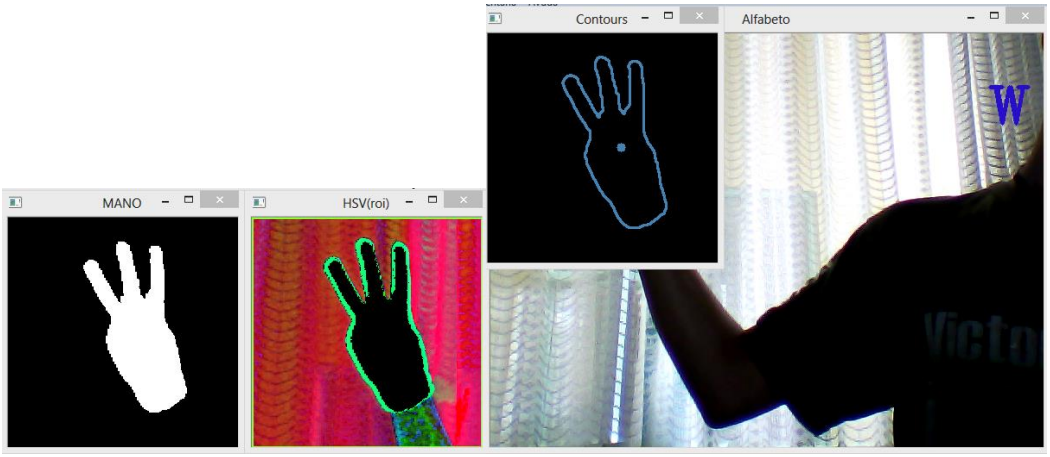
Fuente: Autor

Figura 6.7 Reconocimiento gestual letra T



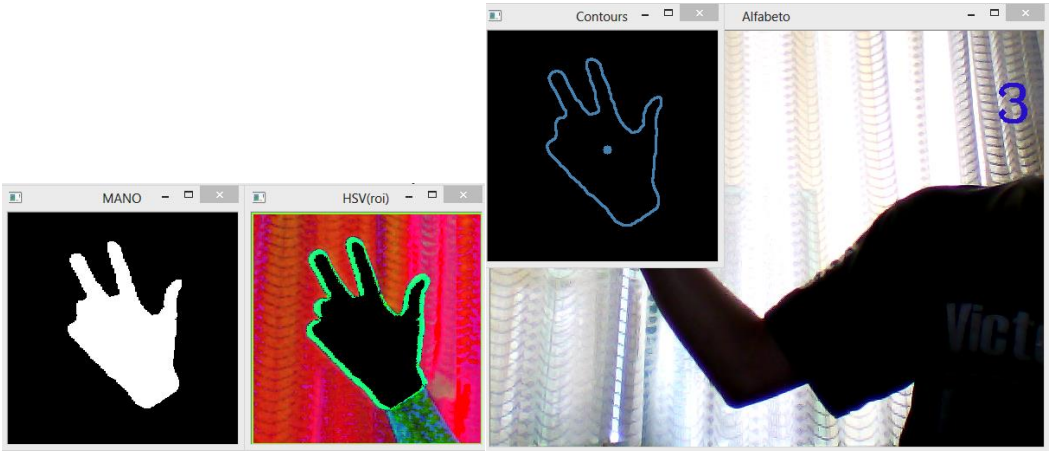
Fuente: Autor

Figura 6.8 Reconocimiento gestual letra W



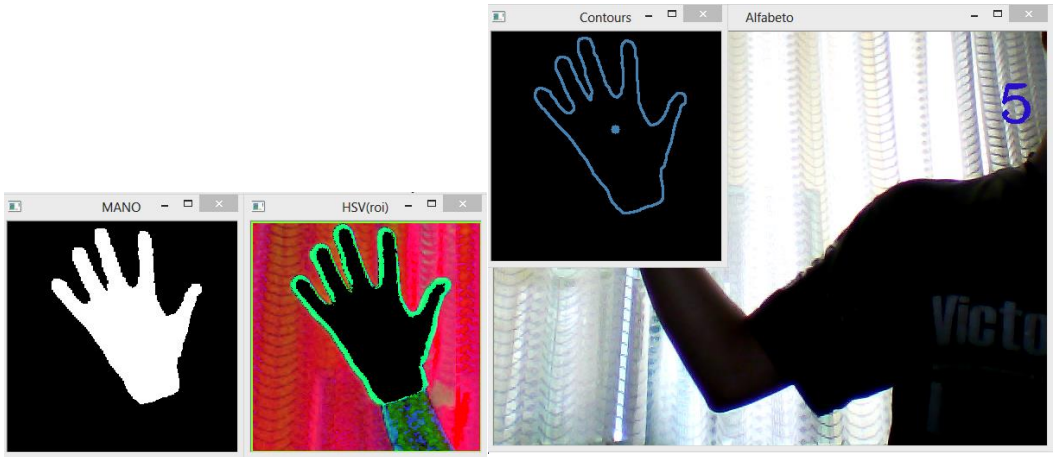
Fuente: Autor

Figura 6.9 Reconocimiento gestual número 3



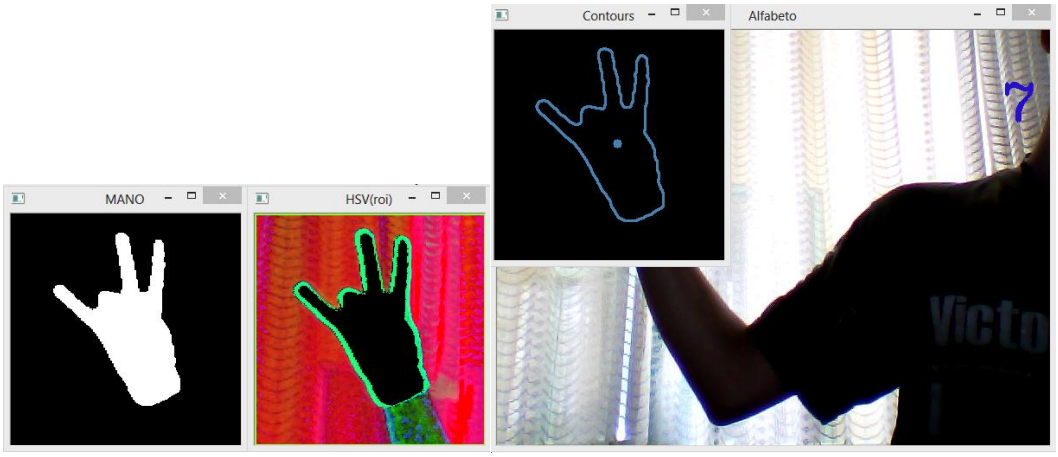
Fuente: Autor

Figura 6.10 Reconocimiento gestual número 5



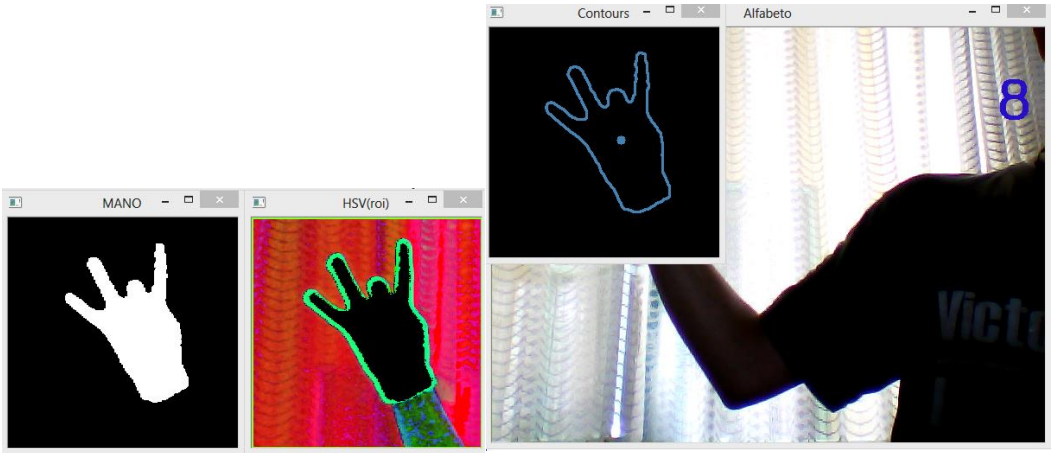
Fuente: Autor

Figura 6.11 Reconocimiento gestual número 7



Fuente: Autor

Figura 6.12 Reconocimiento gestual número 8



Fuente: Autor

7. CONCLUSIONES

La segmentación por color es un punto fundamental del proyecto, y aunque existen diversos métodos de extracción del color de una imagen, es recomendado el espacio el color HSV con sus componentes de matiz, saturación y valor ya que una buena combinación de estos parámetros resuelve inconvenientes de sombras y ayuda a la definición del contorno de la imagen a tratar.

En el reconocimiento de patrones, las características que los describen tienen un papel primordial, en el proyecto se usan los momentos invariantes de HU para obtener las características de las imágenes a reconocer, y aunque algunas de ellas sean diferentes a simple vista hay que tener siempre un gráfico de dispersión de las características extraídas para tener una visión del alcance de los algoritmos, ya que muchos de estos patrones pueden estar sobrepuestos.

El aprendizaje de patrones por una red neuronal artificial es lento y complejo, pero tiene grades ventajas al ser finalizado, ya que puede ser muy preciso en el reconocimiento. El tiempo que tardará la RNA en aprender cierta cantidad de patrones es proporcional a la complejidad de la red, y mientras más compleja sea más tiempo tardará, esto puede llevar desde días hasta semanas de entrenamiento.

La complejidad de la estructura de una RNA depende de la cantidad de patrones que debe aprender, y de que tan dispersos están los patrones entre sí. La cantidad de neuronas y capas ocultas de éstas ayudaran a reconocer más patrones, pero hay que tener especial cuidado en su entrenamiento, ya que es mucho más lento y puede ser que no aprenda correctamente, debido a que la red puede ser o demasiado compleja o muy básica para los patrones que se quieran reconocer. Este proceso es experimental y hay que tener mucha paciencia en su desarrollo.

8. RECOMENDACIONES

En el proceso de diseño de una RNA, se recomienda usar pocas neuronas de salida en la red. Si se van a reconocer pocos patrones, por ejemplo cuatro, se pueden implementar cuatro neuronas de salida con una función de activación tipo escalón. Si por el contrario el número de patrones es mayor, es mejor usar pocas neuronas de salida, como por ejemplo una o dos, con funciones de activación tipo sigmoidea, esto se debe a que la red debe arrojar tantos resultados como patrones tiene que aprender.

Las capas ocultas de neuronas ayudan a separar los diferentes patrones a aprender, con dos capas ocultas es más que suficiente para un proyecto de este tipo, con una cantidad moderada de patrones.

En el proceso de aprendizaje de una RNA, es recomendable entrenar la red patrón por patrón, para conocer sus propios alcances y posibles errores de diseño o de entrenamiento.

Se recomienda usar sus propios códigos de entrenamiento, para poder hacer los ajustes respectivos de acuerdo a la RNA empleada, además de ser una buena práctica para entender el modo en que una RNA opera.

Para proyectos de reconocimiento de patrones puede hacerse uso de diversos tipos de descriptores de imagen, según sea el caso de reconocimiento. Algunos descriptores de imágenes son los diámetros de Feret, aproximaciones poligonales, descriptores de Fourier, momentos de Zernike, descriptores SIFT (Scale Invariant Feature Transform), entre otros.

BIBLIOGRAFIA

ANDERSON, James A. Redes neurales. México: Alfaomega, 2007.

ANDINA, Diego y VEGA CORONA, Antonio. Redes Neurales Artificiales [en línea]. Madrid (España), 3 septiembre 2009 [citado 27 agosto, 2013]. Disponible en internet: http://www.gc.ssr.upm.es/inves/neural/ann2/anntutor.htm.

BERNACKI, Mariusz y WŁODARCZYK, Przemysław. Principles of training multi-layer neural network using backpropagation [en línea]. Cracovia: Universidad AGH. Departamento de electrónica, 2005 [citado 17 diciembre, 2013]. Disponible en internet: http://home.agh.edu.pl/~vlsi/AI/backp_t_en/backprop.html

CASTELLANOS MORENO, Arnulfo. ¿Qué son las redes neuronales? [en línea]. Sonora (México), mayo 2009 [citado 30 agosto, 2013]. Disponible en internet: http://info.fisica.uson.mx/arnulfo.castellanos/archivos_html/quesonredneu.htm.

C.I.P. ETI de Tudela. Visión Artificial [en línea]. Tudela (España) [citado 10 diciembre, 2013]. Disponible en internet: http://www.etitudela.com/celula/downloads/visionartificial.pdf.

ESCOLANO RUIZ, Francisco, *et al*. Inteligencia artificial Modelos, Técnicas y Áreas de Aplicación. Madrid (España): Thomson, 2003.

ESQUEDA ELIZONDO, José Jaime y PALAFOX MAESTRE, Luis Enrique. Fundamento de procesamiento de imágenes. México: Universidad Autónoma de Baja California, 2005.

FAÚNDEZ ZANUY, Marcos. Tratamiento Digital de Voz e Imagen y Aplicación a la Multimedia. Barcelona: Marcombo, S.A., 2000.

GOBIERNO DE ESPAÑA. MINISTERIO DE EDUCACIÓN. Visión Artificial Aplicación práctica de la visión artificial en el control de procesos industriales [en línea]. Madrid (España), febrero 2012 [citado 10 diciembre, 2013]. Disponible en internet: http://visionartificial.fpcat.cat/wp-content/uploads/UD_1_didac_Conceptos_previos.pdf.

HAYKIN, Simon S. Neural networks. A comprehensive Foundation. 2ed. Upper Saddle River (New Jersey): Prentice Hall International, Inc. 1999.

MARTÍN, Marcos. Descriptores de Imagen [en línea]. España, mayo 2002 [citado 15 diciembre, 2013]. Disponible en internet: http://poseidon.tel.uva.es/~carlos/ltif10001/descriptores.pdf

MICROSOFT CORPORATION. Modelos de color [en línea]. [Citado 11 diciembre, 2013]. Disponible en internet: http://msdn.microsoft.com/es-es/library/dd129721.aspx.

MING KUEI, Hu. Visual pattern recognition by moment invariants. 1962 Journal IEEE Transactions on information theory vol.8 [base de datos en línea]. [Citado en 15 noviembre de 2013]. Disponible en biblioteca digital IEEE Xplore®.

PLATERO, Carlos. Apuntes de Visión Artificial [en línea]. Madrid (España) [citado 10 diciembre, 2013]. Disponible en internet: http://www.elai.upm.es/webantigua/spain/Asignaturas/MIP_VisionArtificial/ApuntesVA/cap1IntroVA.pdf.

Real Academia Española. Diccionario de la lengua española [en línea]. 22ª ed. Madrid (España), 2001 [citado 2 diciembre, 2013]. Disponible en internet: http://lema.rae.es/drae/.

SALAS, Rodrigo. Redes Neuronales Artificiales [en línea]. Valparaíso (Chile), 4 noviembre 2004 [citado 27 agosto, 2013]. Disponible en internet: http://www.inf.utfsm.cl/~rsalas/Pagina_Investigacion/docs/Apuntes/Redes%20Neuronales%20Artificiales.pdf.

SHUTLER, Jaime. Statistical Moments [en línea]. Universidad de Southampton (Inglaterra) [Citado en 15 noviembre de 2013]. Disponible en internet: http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/SHUTLER3/CVonline_moments.html.

Signos, señales y símbolos. El Nuevo Diario, publicado 7 enero 2011 [citado 30 agosto, 2013]. Disponible en internet: http://www.elnuevodiario.com.ni/opinion/91902.

SOBRADO MALPARTIDA, Eddie Ángel. Sistema de Visión Artificial para el Reconocimiento y Manipulación de Objetos utilizando un Brazo Robot. Tesis Magister en Ingeniería de Control y Automatización. Lima (Perú). Pontificia Universidad Católica del Perú, 2003. 39,42-44 p.

THEODORIDIS, Sergios y KONSTANTINOS, Koutroumbas. Pattern Recognition. 4 ed. EE.UU: Academic Press, 2009.

ANEXOS

ANEXO A.
CÓDIGO EN LENGUAJE C++ PARA CAPTURAR Y GUARDAR IMÁGENES

```
//Incluir Librerías
#include <opencv\cv.h>
#include <opencv\highgui.h>
#include <opencv2\imgproc\imgproc.hpp>
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <opencv2\opencv.hpp>
#include <opencv2\core\core.hpp>
#include <opencv2\highgui\highgui.hpp>

//Espacio de Nombres
#define w 400
using namespace cv;
using namespace std;

//Valores iniciales de los filtros HSV
int H_MIN = 0;
int H_MAX = 0;
int S_MIN = 0;
int S_MAX = 0;
int V_MIN = 0;
int V_MAX = 0;

//Constantes string (Nombramos las ventanas)
string   windowName = "Imagen Original";
string  windowName1 = "Imagen HSV";
string         MANO = "MANO";

//Creamos las matrices de trabajo
Mat image, HSV, Filtro;
Mat src; Mat src_gray;
int d = 1;
int thresh = 100;
```

```
//Función Principal
int main(){
	//Crear las ventanas para mostrar las imágenes
	namedWindow( windowName, CV_WINDOW_AUTOSIZE);
	namedWindow(windowName1, CV_WINDOW_AUTOSIZE);
	namedWindow(MANO, CV_WINDOW_AUTOSIZE);

	//Iniciamos la captura
	VideoCapture cap;
	cap.open(1);

	while(1){
		//(Lo que captura la Webcam lo pasa a la Mat image)
		cap>>image;
		//ubicamos un rectangulo en la pantalla
		rectangle( image,Point( 0, 0 ),Point( 2*w/3, 2*w/3),Scalar( 0, 138, 21 )
		,2,4);

		//Convierte al modelo HSV la imagen capturada
		cvtColor(image,HSV,COLOR_BGR2HSV);

		//Mostramos las imágenes de la Webcam y la HSV
		imshow(windowName,image);
		imshow(windowName1,HSV);

		//El rango del filtro HSV va a estar desde _MIN hasta _MAX de los
		 valores de   //Hue, Saturation y Value
		inRange(HSV,Scalar(H_MIN,S_MIN,V_MIN),
					Scalar(H_MAX,S_MAX,V_MAX),Filtro);

		//Crear elemento de estructuración para realizar dilatación y erosión
		de la imagen. Matriz 3x3
		Mat element = getStructuringElement( MORPH_RECT, Size(3,3));

		//Creamos una matriz temporal para almacenar la erosión/dilatación
		Mat temp;

		//Mediante la dilatación y erosión iremos eliminando el ruido
		erode(Filtro,temp,element);
		//Dilatamos un par de veces
		dilate(temp,temp,element);
		dilate(temp,temp,element);

		//Hacemos un rectángulo de la sección que nos interesa (ROI)
		Rect roi(0, 0 ,(2*w/3), 2*w/3);
```

```
                //Ubicamos la ROI en una matriz - Es la matriz de donde extraeremos
                 la ROI
                //(en este caso temp)
                Mat image_roi = temp(roi);

                //Mostramos la ROI
                imshow(MANO,image_roi);

                //Guardar imagen
                string imageOut="C:\\A\\A1.jpg";  //Dirección de destino y nombre.jpg
                imwrite(imageOut, image_roi);

                waitKey(33); //Refresca cada 33ms
        }
        return 0;
}
```

ANEXO B.
CÓDIGO EN LENGUAJE C++ PARA EXTRAER LOS MOMENTOS DE HU DE LA BASE DE DATOS

```
//Incluir librerías requeridas
#include <opencv2\opencv.hpp>
#include <opencv2\core\core.hpp>
#include <opencv2\highgui\highgui.hpp>
#include <opencv2\imgproc\imgproc.hpp>
#include <iostream>
#include <stdio.h>
#include <stdlib.h>

//Utilizar espacio de nombres
using namespace cv;
using namespace std;

//Matrices de trabajo
Mat src; Mat src_gray;
int d = 1;                    //Variable contadora de Imágenes
int thresh = 100;      //Umbral para segmentar

//Dirección de las imagenes a las que se les desea extraer los Momentos
string imagen1="C:\\PROYECTO\\ESE\\S1.jpg";
string imagen2="C:\\PROYECTO\\ESE\\S2.jpg";
string imagen3="C:\\PROYECTO\\ESE\\S3.jpg";
string imagen4="C:\\PROYECTO\\ESE\\S4.jpg";
string imagen5="C:\\PROYECTO\\ESE\\S5.jpg";
string imagen6="C:\\PROYECTO\\ESE\\S6.jpg";
string imagen7="C:\\PROYECTO\\ESE\\S7.jpg";
string imagen8="C:\\PROYECTO\\ESE\\S8.jpg";
string imagen9="C:\\PROYECTO\\ESE\\S9.jpg";
string imagen10="C:\\PROYECTO\\ESE\\S10.jpg";
string imagen11="C:\\PROYECTO\\ESE\\S11.jpg";
string imagen12="C:\\PROYECTO\\ESE\\S12.jpg";
string imagen13="C:\\PROYECTO\\ESE\\S13.jpg";
string imagen14="C:\\PROYECTO\\ESE\\S14.jpg";
string imagen15="C:\\PROYECTO\\ESE\\S15.jpg";
string imagen16="C:\\PROYECTO\\ESE\\S16.jpg";
string imagen17="C:\\PROYECTO\\ESE\\S17.jpg";
string imagen18="C:\\PROYECTO\\ESE\\S18.jpg";
string imagen19="C:\\PROYECTO\\ESE\\S19.jpg";
string imagen20="C:\\PROYECTO\\ESE\\S20.jpg";
string imagen21="C:\\PROYECTO\\ESE\\S21.jpg";
```

```
string imagen22="C:\\PROYECTO\\ESE\\S22.jpg";
string imagen23="C:\\PROYECTO\\ESE\\S23.jpg";
string imagen24="C:\\PROYECTO\\ESE\\S24.jpg";
string imagen25="C:\\PROYECTO\\ESE\\S25.jpg";

////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//Funcion de extraccion de Momentos
void Momentos(string imagen);

//////////////////////FUNCION PRINCIPAL//////////////////////////
int main( int argc, char** argv ){

        //cout<<"\n MOMENTOS DE HU IMAGEN "; cout<< d<<endl;
        Momentos(imagen1);        //Llama la Funcion Momentos para Extraer los
        Momentos de la Imanen1
        d++;

        //cout<<"\n MOMENTOS DE HU IMAGEN "; cout<< d<<endl;
        Momentos(imagen2);
        d++;

        //cout<<"\n MOMENTOS DE HU IMAGEN "; cout<< d<<endl;
        Momentos(imagen3);
        d++;

        //cout<<"\n MOMENTOS DE HU IMAGEN "; cout<< d<<endl;
        Momentos(imagen4);
        d++;

        //cout<<"\n MOMENTOS DE HU IMAGEN "; cout<< d<<endl;
        Momentos(imagen5);
        d++;

        //cout<<"\n MOMENTOS DE HU IMAGEN "; cout<< d<<endl;
        Momentos(imagen6);
        d++;

        //cout<<"\n MOMENTOS DE HU IMAGEN "; cout<< d<<endl;
        Momentos(imagen7);
        d++;

        //cout<<"\n MOMENTOS DE HU IMAGEN "; cout<< d<<endl;
        Momentos(imagen8);
        d++;
```

```
//cout<<"\n MOMENTOS DE HU IMAGEN "; cout<< d<<endl;
Momentos(imagen9);
d++;

//cout<<"\n MOMENTOS DE HU IMAGEN "; cout<< d<<endl;
Momentos(imagen10);
d++;

//cout<<"\n MOMENTOS DE HU IMAGEN "; cout<< d<<endl;
Momentos(imagen11);
d++;

//cout<<"\n MOMENTOS DE HU IMAGEN "; cout<< d<<endl;
Momentos(imagen12);
d++;
system("pause");

//cout<<"\n MOMENTOS DE HU IMAGEN "; cout<< d<<endl;
Momentos(imagen13);
d++;

//cout<<"\n MOMENTOS DE HU IMAGEN "; cout<< d<<endl;
Momentos(imagen14);
d++;

//cout<<"\n MOMENTOS DE HU IMAGEN "; cout<< d<<endl;
Momentos(imagen15);
d++;

//cout<<"\n MOMENTOS DE HU IMAGEN "; cout<< d<<endl;
Momentos(imagen16);
d++;

//cout<<"\n MOMENTOS DE HU IMAGEN "; cout<< d<<endl;
Momentos(imagen17);
d++;

//cout<<"\n MOMENTOS DE HU IMAGEN "; cout<< d<<endl;
Momentos(imagen18);
d++;

//cout<<"\n MOMENTOS DE HU IMAGEN "; cout<< d<<endl;
Momentos(imagen19);
d++;
```

```
        //cout<<"\n MOMENTOS DE HU IMAGEN "; cout<< d<<endl;
        Momentos(imagen20);
        d++;

        //cout<<"\n MOMENTOS DE HU IMAGEN "; cout<< d<<endl;
        Momentos(imagen21);
        d++;

        //cout<<"\n MOMENTOS DE HU IMAGEN "; cout<< d<<endl;
        Momentos(imagen22);
        d++;

        //cout<<"\n MOMENTOS DE HU IMAGEN "; cout<< d<<endl;
        Momentos(imagen23);
        d++;

        //cout<<"\n MOMENTOS DE HU IMAGEN "; cout<< d<<endl;
        Momentos(imagen24);
        d++;

        //cout<<"\n MOMENTOS DE HU IMAGEN "; cout<< d<<endl;
        Momentos(imagen25);
        d++;

  waitKey(0);
  return(0);
}

////////////////////FUNCION MOMENTOS//////////////////////////
void Momentos(string imagen){
        Mat image = imread(imagen);             //Crea una matriz image, donde
        leerá la imagen guardada
        src = image;

  cvtColor( src, src_gray, CV_BGR2GRAY );    //Filtra la imagen
  blur( src_gray, src_gray, Size(3,3) );    //Suaviza la imagen

  Mat canny_output;
  vector<vector<Point> > contours;
  vector<Vec4i> hierarchy;

  /// Detecta bordes
  Canny( src_gray, canny_output, thresh, thresh*2, 3 );
```

```
/// Busca Contornos
findContours( canny_output, contours, hierarchy, CV_RETR_TREE,
CV_CHAIN_APPROX_SIMPLE, Point(0, 0) );

//printf("\n NUMERO DE CONTORNOS %d \n",contours.size());
///IF de control para reconocer si hay contornos o no los hay
if (contours.size()>0){
//CONTORNO MAYOR
int Mayor, Cont1, n=0;
Cont1=contourArea(contours[0]);
for( int i = 0; i < contours.size(); i++ )
  {
                Mayor=contourArea(contours[i]);
                if (Mayor > Cont1){
                        n=i;
                        Cont1=contourArea(contours[i]);
                }
  }
//printf("\n El Contorno Mayor Es El Numero %d  \n",n);

//////////////////MOMENTOS//////////////////////
vector<Moments> mu(contours.size() );        //Vector mu par los Momentos

mu[n] = moments( contours[n], true ); //Extrae los Momentos de la imagen

double h[7];//Arreglo de 7 posiciones para almacenar los 7 Momentos de HU
HuMoments(mu[n],h);
//printf("\n Contorno Numero %d  \n",n);
//printf("\n \t MOMENTOS DE HU \n [1] %.20f \n [2] %.20f \n [3] %.20f \n [4] %.20f
\n [5] %.20f \n [6] %.20f \n [7] %.20f \n",h[0],h[1],h[2],h[3],h[4],h[5],h[6]);  //Siete
Momentos de HU
//printf("\n \t MOMENTOS DE HU \n {%.20f, %.20f, %.20f, %.20f, %.20f, %.20f,
%.20f},",h[0],h[1],h[2],h[3],h[4],h[5],h[6]);  //Siete Momentos de HU
printf("\n{%.20f, %.20f, %.20f, %.20f},",h[0],h[1],h[2],h[3]);  //Momentos 1,2,3,4
//printf("\n%.20f      %.20f %.20f",h[4],h[5],h[6]);                //Momentos 5,6,7
//printf("\n%.20f  %.20f",h[0],h[1]);  //Momentos 1 y 2 para uso de Gráficas

/// Dibuja Contorno
Mat drawing = Mat::zeros( canny_output.size(), CV_8UC3 );        //Crea matriz
para dibujar el contorno

Scalar color = Scalar(200, 120, 50 );    //Color del Contorno BGR
drawContours( drawing, contours, n, color, 2, 8, hierarchy, 0, Point() );    //Dibuja
el contorno
```

```
/// Muestra el Contorno en la pantalla
imshow( "Contours", drawing );
}
else {            //Si no hay contorno regresa a main
          waitKey(2000);
          main(0,0);
                  }
        }
```

ANEXO C.
CÓDIGO EN LENGUAJE C++ PARA ENTRENAR LA RED NEURONAL ARTIFICIAL

```
//Incluir Librerías necesarias
#include<iostream>
#include<stdlib.h>
#include<vector>
#include <time.h>
#include <math.h>

//Espacio de nombres
using namespace std;
#define length(SalEsp) (sizeof(SalEsp)/sizeof(SalEsp[0]))    //Define
length(SalEsp) como una función para obtener la longitud de SalEsp (Salida
Esperada)
#define length(Pesos) (sizeof(Pesos)/sizeof(Pesos[0]))              //Define
length(Pesos) como una función para obtener la longitud de Pesos (Pesos de las
conexiones Sinápticas)
#define row(Entradas) (sizeof(Entradas)/sizeof(Entradas)[0])        //Define
row(Entrada) como una funcion para obtener la cantidad de filas de las entradas

////Funcion que opera con la funcion Sigmoide///////
double sigm(double s){
        double sig;                                  //Variable de operación función
Sigmoide
        sig=-1+(2/(1+(exp(-1*s))));        //Función Sigmoide Bipolar (Max 1; Min -1)
        return sig;
}

////Función que opera con la Derivada de la función Sigmoide///////
double Dervsigm(double der){
       double deri;                                  //Variable de operación Derivada de
Sigmoide
       deri=(2*(exp(-1*der)))/(pow((1+(exp(-1*der))),2));      //Función Derivada de
Sigmoide
       return deri;
}

//Función del modelo de Perceptrón de la primera capa (Neuronas)----w(pesos de
la conexiones sinápticas),--pat(patrones de entrada)
double Perceptron1C(double w0,double w1,double w2,double w3,double
pat0,double pat1,double pat2,double pat3){
       double x;      //Variable para las operaciones de los Perceptrones
```

```
        double outPer1C;            //Variable para la operación de "Salida del
Perceptrón" outPer
        x = (w0*pat0)+(w1*pat1)+(w2*pat2)+(w3*pat3);        //Operación Neuronal
(Comportamiento de la Neurona)
        outPer1C=sigm(x);
        //Respuesta Neuronal, con la función de excitación tipo Sigmoide Bipolar
        return outPer1C;                                                    //Retorna
respuesta
}

//Función del modelo de Perceptrón de la segunda capa (Neuronas)----w(pesos de
la conexiones sinápticas),--sal(patrones de entrada)
double Perceptron2C(double w0,double w1,double w2,double w3,double
w4,double w5,double w6,double sal0,double sal1,double sal2,double sal3,double
sal4,double sal5,double sal6){
        double x;      //Variable para las operaciones de los Perceptrones
        double outPer2C;            //Variable para la operación de "Salida del
Perceptrón" outPer
        x =
(w0*sal0)+(w1*sal1)+(w2*sal2)+(w3*sal3)+(w4*sal4)+(w5*sal5)+(w6*sal6);
        //Operación Neuronal (Comportamiento de la Neurona)
        outPer2C=sigm(x);                                   //Respuesta Neuronal, con la
función de excitación tipo Sigmoide Bipolar
        return outPer2C;                                    //Retorna respuesta
}


double Salida [12];  //Arreglo de 12 posiciones para las salidas de cada una de las
12 Neuronas de la RNA
double d[12];        //Arreglo de 12 posiciones para guardar el error de cada una
de las Neuronas
double Pesos [60]={0.563124,  -0.94817,  -0.0686787,  0.553991,  2.58682,  -
0.531087,  -0.937884,1.52531,  -1.32481,
        0.862756,  2.68041,  4.78047,  -1.66301,  -0.108036,  0.941363,
0.0533466,  -0.999877,  -1.06865,  -1.18462,
        -0.935077,  0.0416234,  0.00070363,  0.999128,  0.00061051,  0.00230136,
-0.997689,  0.99896,  -0.999514, 0.224642,
        -1.25142,  -0.747134,  1.25466,  -0.462456,  0.507883,  -0.486854,-
0.508046,  -0.544669,  -0.621562,  -0.434256,
        0.646596,  1.47785,  0.351491,  1.61074,  0.676508,  0.76771,  -1.75833,
1.76018,  1.74312,  0.78728,  0.237687,
        -1.2392,  -1.24068,  0.789531,  1.12013,  0.864415,  -1.09704,  0.967036,  -
1.24181,  1.37277,  1.31864};
//  PESOS RESULTADOS DEL ULTIMO ENTRENAMIENTO DE LA RED
NEURONAL ARTIFICIAL
```

```
double SalEsp [6][1]= {{0.52},{0.55},{0.58},{0.62},{0.76},{0.74}};      //Salida esperada para cada uno de los grupos de Patrones

double Salida11;                                    //Variable Salida11, se usa para comparar la respuesta de la RNA con las salidas esperadas

double Entradas [150][4] = {                //Arreglo del grupo de Patrones de Entrada. 150 Patrones

{0.17046594464485831000, 0.00162754084216391890,
0.00057538944246830206, 0.00000604401620370542},
{0.16918141860625635000, 0.00195387420465927210,
0.00030522649353073334, 0.00000486871631249871},
{0.17020175908011764000, 0.00178776386567365120,
0.00048683220545606837, 0.00000769447280623983},
{0.17134747048867022000, 0.00220234697826169560,
0.00036580659395702060, 0.00000428928086186904},
{0.16995842626543572000, 0.00146378344131573890,
0.00053458523645011379, 0.00000613790843311602},
{0.17065894326029249000, 0.00208387443905507040,
0.00039045425033247712, 0.00000599048148606499},
{0.16930372520407710000, 0.00171178547521442640,
0.00041451758998927113, 0.00000567432894858869},
{0.16885849609488796000, 0.00138678206539449350,
0.00046758346467012951, 0.00000512207558970276},
{0.16879726337757067000, 0.00148449222853467620,
0.00043241485973942456, 0.00000514416886254030},
{0.16997318323501079000, 0.00152368595140568420,
0.00056261493986502144, 0.00000662476888380587},
{0.16963364405587483000, 0.00148812567174820360,
0.00052056882576309734, 0.00000615042596580837},
{0.17117674275230882000, 0.00231570439622744580,
0.00036440462015072242, 0.00000637892072463770},
{0.17054523755453699000, 0.00171112768535027090,
0.00056214413452234031, 0.00000664561051166404},
{0.17070441236595438000, 0.00164314201137230670,
0.00055332039186689597, 0.00000672161262150209},
{0.16885658316284266000, 0.00128720297735181890,
0.00049455240352638135, 0.00000584797162979301},
{0.17162767638220611000, 0.00197592329443200880,
0.00055813061846603565, 0.00000903924770794263},
{0.16969156875251207000, 0.00136197968866880660,
0.00056224711947255694, 0.00000553641250683114},
```

```
{0.17029721108955148000, 0.00156154163735791430,
0.00056213128161298337, 0.00000692768891932834},
{0.16943989471221030000, 0.00135074771719373930,
0.00054259093891611805, 0.00000473636886415387},
{0.16962645042517427000, 0.00156196180302438910,
0.00050001325078191032, 0.00000589121373506336},
{0.17129193597276240000, 0.00200197163026369100,
0.00051679636621046975, 0.00000625422286125095},
{0.16898291051661612000, 0.00150108679442165450,
0.00044785595000498108, 0.00000571722043225539},
{0.17112863302172221000, 0.00214877955044817230,
0.00046158031368597201, 0.00000859328436939879},
{0.17081876543419522000, 0.00193368989563554200,
0.00046869905607880918, 0.00000707303045799991},    //Patrones
{0.16921770047802248000, 0.00157654582532877140,
0.00045328666586838285, 0.00000745730483829965},    //S

{0.17688071716482540000, 0.00403911415072314390,
0.00039356653452520798, 0.00001553933073451443},
{0.17844918052399095000, 0.00449411908748124920,
0.00038952067506055907, 0.00001313901732110813},
{0.17548084254842380000, 0.00375645153455896100,
0.00038324849150068156, 0.00000982790090970636},
{0.17933958602080860000, 0.00508050672461521880,
0.00032545925490305196, 0.00001548528858982761},
{0.17750607566515691000, 0.00433341350109325230,
0.00035276013321773207, 0.00001323285007367201},
{0.17720015932239425000, 0.00474695677525374240,
0.00027556262491386688, 0.00001099923828434270},
{0.17896594904943250000, 0.00483917890934555150,
0.00039708265255288504, 0.00001808417380083195},
{0.17806599287973096000, 0.00455291679641532670,
0.00042082330040435939, 0.00001474342469195987},
{0.17811485621417511000, 0.00477381909506375230,
0.00032987974925697004, 0.00001287206787956397},
{0.17714650520027886000, 0.00448252537025838580,
0.00028263621379380385, 0.00000881305128932073},
{0.17606105140470163000, 0.00400044592221810490,
0.00034869784869348769, 0.00000976366140627021},
{0.17569597098868006000, 0.00390366653893133450,
0.00035228008534242389, 0.00001136728874045617},
{0.17872850503051901000, 0.00481567587705582830,
0.00048567147542530917, 0.00001819325934982056},
{0.17585949951281077000, 0.00402635940122210690,
0.00031232480866075229, 0.00000835109831001255},
```

```
{0.17762591286834511000, 0.00479598042142504940,
0.00032701169318005263, 0.00001276918925454236},
{0.17552886301165249000, 0.00390303178247256500,
0.00030728115862283553, 0.00000882628631946902},
{0.17592569256996449000, 0.00411918636949198170,
0.00032047192765371948, 0.00000973976923483261},
{0.17579346011997954000, 0.00397390169801586750,
0.00037859146008638870, 0.00001197572726721963},
{0.17579346011997954000, 0.00397390169801586750,
0.00037859146008638870, 0.00001197572726721963},
{0.17753707687932735000, 0.00453338071772768460,
0.00043447177970793534, 0.00001740813631410056},
{0.17756971276038233000, 0.00478413829898831360,
0.00027690154161584616, 0.00001069483414278186},
{0.17659200705434164000, 0.00418695254634543060,
0.00037058673971435976, 0.00001302449959272800},
{0.17613441936478297000, 0.00367103179182305590,
0.00044916365458554130, 0.00001251257604267640},
{0.17944065668493223000, 0.00487356210599207760,
0.00042156408403189654, 0.00001661966501438933},    //Patrones
{0.17582896279105945000, 0.00411719993550896040,
0.00031253746214773587, 0.00001065209373796162},    //E

{0.19500893943709419000, 0.00586514909501798570,
0.00021420761896677072, 0.00004354100016136504},
{0.19495081602082270000, 0.00561250902906979330,
0.00024315041765719903, 0.00004304088493881302},
{0.19463442815304416000, 0.00592381058998057120,
0.00024427473271996023, 0.00003953289894944714},
{0.19419944231881708000, 0.00618687357367424770,
0.00022984857445084380, 0.00003172011275382461},
{0.19334514590022306000, 0.00510413573318953000,
0.00021735740712072889, 0.00005103253296095380},
{0.19555608318500489000, 0.00648867618466719610,
0.00020504575252987094, 0.00003175544246213607},
{0.19526571257909869000, 0.00592300987968585880,
0.00026816072123004837, 0.00004523073875655055},
{0.19684096489241415000, 0.00621777708338295490,
0.00020880098451653311, 0.00005082793632101434},
{0.19168441959038870000, 0.00454923884375207920,
0.00024370152957525545, 0.00003443933960804428},
{0.19575925238726061000, 0.00592237347614260260,
0.00032630761784657551, 0.00004453277868479990},
{0.19549722255599516000, 0.00626267865724821050,
0.00027775258970206570, 0.00004897952578296474},
```

```
{0.19383006333434766000, 0.00537985920605349960,
0.00026722833043402606, 0.00003933177172594089},
{0.19756544319557798000, 0.00655717883937687820,
0.00026293583343600852, 0.00004178663893284925},
{0.19627451796126213000, 0.00673012170512803270,
0.00032181016730907448, 0.00004045852496657803},
{0.19212396909102991000, 0.00435687155197588860,
0.00023802660606440462, 0.00002672815227975255},
{0.19641067736539197000, 0.00641686040824202690,
0.00032165069899825461, 0.00003332878163822730},
{0.19589586448361329000, 0.00622284002121926020,
0.00021397892300193059, 0.00003973042587089962},
{0.19494592472878441000, 0.00645888184939940460,
0.00028589502093351809, 0.00003151736034337391},
{0.19494592472878441000, 0.00645888184939940460,
0.00028589502093351809, 0.00003151736034337391},
{0.19328467805872468000, 0.00534848482703393320,
0.00026864433489850304, 0.00004096959699118598},
{0.19386763096857690000, 0.00537508916719706480,
0.00027381399606724255, 0.00004050167431329795},
{0.19260439183044903000, 0.00544397710845875590,
0.00017347014017029738, 0.00003574274681456033},
{0.19309574095400056000, 0.00492798341783000010,
0.00025278082765069481, 0.00002947528617125305},
{0.19253156828860651000, 0.00499434848441141640,
0.00015025511506284230, 0.00004074725732099686},    //Patrones
{0.19296416993319082000, 0.00523552608640760930,
0.00019821376759900067, 0.00004384911697336941},    //L

{0.20901529837286734000, 0.00844203469874542890,
0.00009131681499996354, 0.00016122725239737059},
{0.20371217921740042000, 0.00815291998978420580,
0.00006518445055443468, 0.00012552148065729936},
{0.20273706040388284000, 0.00874681367814754420,
0.00004877635752354237, 0.00008321078201018115},
{0.20587664050348647000, 0.00742377632141247650,
0.00008253507720639829, 0.00013997808081725658},
{0.20880639750637994000, 0.00781510244631470130,
0.00018589945734685982, 0.00025636524210547297},
{0.20664797152776282000, 0.00876610357566270060,
0.00003255751443335211, 0.00013656687305843495},
{0.20625454029413381000, 0.00833198912866709930,
0.00002659494836239074, 0.00014780537805820997},
{0.20812360203055555000, 0.00798740281148096860,
0.00006056637989169764, 0.00016087394299240686},
```

```
{0.20505119118194101000, 0.00803670311964454350,
0.00002503230970639782, 0.00013340924625987867},
{0.20409763997867855000, 0.00879905258903339320,
0.00001085880176464159, 0.00013009302385871997},
{0.20339956715613983000, 0.00823622574118355880,
0.00001782958261734240, 0.00013770912202955998},
{0.20725688507577200000, 0.00749461581127158680,
0.00023460990186194674, 0.00024289288485051185},
{0.20262199552213572000, 0.00797695450593622660,
0.00002117321751296543, 0.00012178944520050481},
{0.20538079400506143000, 0.00827358916915360390,
0.00003176948502110759, 0.00015421361841492129},
{0.20605523140124032000, 0.00856253427025958570,
0.00002883265405859986, 0.00014819020246543659},
{0.20732928301035369000, 0.00911158275138623800,
0.00006550829912802664, 0.00016297680590864305},
{0.20766145398026475000, 0.00989834684108833840,
0.00001696598315890990, 0.00015119815315134727},
{0.20325954781015915000, 0.00922638537897501120,
0.00002072694009310239, 0.00009946536431763475},
{0.20325954781015915000, 0.00922638537897501120,
0.00002072694009310239, 0.00009946536431763475},
{0.20502948174824545000, 0.00930557319960825450,
0.00000724033661221765, 0.00013350197587960166},
{0.20530121598891055000, 0.00686285306905395210,
0.00027964743530390544, 0.00021089374725413525},
{0.20502074604446574000, 0.00923097566119265860,
0.00007068815595782315, 0.00008372440517080754},
{0.20780033381055146000, 0.00702985113577668170,
0.00027045015917009918, 0.00025339205050593506},
{0.20341352259723744000, 0.00785749285631255100,
0.00008797875480334814, 0.00013630241426316314},    //Patrones
{0.20328121656028286000, 0.00801526476304722550,
0.00009248324810477543, 0.00012802309147557679},    //3

{0.25741015390387084000, 0.03117514545963467200,
0.00004767568647162701, 0.00021001594251755828},
{0.26253568140011330000, 0.03600508221140013400,
0.00007012138091417173, 0.00032332042783083065},
{0.25686884541508559000, 0.03191518628046603100,
0.00003005034814177902, 0.00040589024619661432},
{0.26084674448533346000, 0.03245438071846377900,
0.00003866864148330108, 0.00025168527497508857},
{0.26379760728420365000, 0.03552011774569237300,
0.00000668169230183425, 0.00025488931385766100},
```

{0.25907479102526465000, 0.03248423376083826200,
0.00004086576499726145, 0.00045324779042188760},
{0.25843458604168718000, 0.03360535105423176300,
0.00001669496708361923, 0.00019448614965231778},
{0.26298454163604496000, 0.03484935772908519200,
0.00005829279988198965, 0.00016971599811209083},
{0.25613170927208029000, 0.02996914065988330600,
0.00002935254635371315, 0.00051471848298257586},
{0.25660605976776740000, 0.03138415303027041600,
0.00014429971259551474, 0.00056271534827162821},
{0.26107471230134688000, 0.03530828219183474600,
0.00006275713929401237, 0.00024240780019523720},
{0.25870351219303667000, 0.03286311248622852300,
0.00007421866307223536, 0.00047599446326807411},
{0.25729546707484613000, 0.03168283722981083500,
0.00005832591552615587, 0.00021369100942996710},
{0.25712759301887828000, 0.02988328856048793500,
0.00022757766823837712, 0.00016184983565332815},
{0.25685907649155659000, 0.03006808518817967500,
0.00011744792147424939, 0.00022781421208704867},
{0.25445367578110856000, 0.02936015067714728400,
0.00008018520510451977, 0.00054913763863634010},
{0.25839430859450657000, 0.03013764003807766300,
0.00016317234487101434, 0.00022529573754185997},
{0.25852956489623929000, 0.03233086352334000500,
0.00005674239758073107, 0.00055322078379228454},
{0.26214169073620169000, 0.03323282489916819200,
0.00006309683925906030, 0.00056876696225418070},
{0.25588211142348272000, 0.02865189410436518000,
0.00016547265388978791, 0.00022390256476980037},
{0.25560215118052887000, 0.02897961730737929500,
0.00009849735312833632, 0.00022175100283365685},
{0.25472861796947177000, 0.02923231594655742100,
0.00001751632065340115, 0.00041993151290995769},
{0.25567896363939596000, 0.03102785733090214800,
0.00010582198271520414, 0.00021798873019963029},
{0.25450840492451399000, 0.03063441878780727400,
0.00012345923115258348, 0.00047795116273556862}, //Patrones
{0.26177404431712331000, 0.03321364782864447100,
0.00006812986511875844, 0.00023909200506935660}, //W

{0.25675682151895224000, 0.02684037096284262400,
0.00046009373590201366, 0.00049194420410362584},
{0.24733483533272932000, 0.02401103003674097400,
0.00036201947144326636, 0.00029404245023619669},

{0.24406413634197385000, 0.02249730756625464500,
0.00036445126705967847, 0.00025505562339115279},
{0.24861613872727342000, 0.02490132072585548800,
0.00040539090550307491, 0.00027666858030777789},
{0.25024592699935017000, 0.02600185759827421500,
0.00024157370107139655, 0.00028575971135566303},
{0.25729408993878788000, 0.02441921941061393800,
0.00044238515771817473, 0.00059921715477956696},
{0.25226620903835928000, 0.02449122668589234100,
0.00037983316059649562, 0.00029950174862515593},
{0.24631240996549597000, 0.02265525410051512000,
0.00021500440226506515, 0.00036123779691980482},
{0.24447364446768372000, 0.02218240616644185500,
0.00038264935786174630, 0.00033571977447975588},
{0.25808078980196669000, 0.02531657479535417300,
0.00053465261601063512, 0.00052605714567886721},
{0.25439694547624353000, 0.02608601372046818000,
0.00020807711482470268, 0.00026915264509205371},
{0.24822984782034041000, 0.02549989596853993000,
0.00013886584094431918, 0.00010107478137552478},
{0.25084451406709501000, 0.02344237629325434900,
0.00004258636747531078, 0.00027675346917323279},
{0.25112681973767287000, 0.02618603751052788600,
0.00006750413580755965, 0.00018280398996155570},
{0.24585877313662372000, 0.02445631608681007200,
0.00009879353394126199, 0.00013389679491121714},
{0.24891301741653316000, 0.02433197683552675700,
0.00037789537472891061, 0.00020349735595160754},
{0.24962698181269899000, 0.02624301107624835100,
0.00007164902647473578, 0.00018075516511359586},
{0.24949873664522051000, 0.02289474786953827100,
0.00061533564687157069, 0.00048961675644098993},
{0.24626113788279219000, 0.02456073554565129500,
0.00018648764568564101, 0.00008024759660565397},
{0.25107358644081990000, 0.02513618879688487400,
0.00030827994359665281, 0.00021587025192455669},
{0.25078909769614260000, 0.02614068194745686500,
0.00003447337671346906, 0.00024811038258641752},
{0.25215413198339731000, 0.02603806211948396900,
0.00005347026243900502, 0.00032146383018537462},
{0.24921707359305251000, 0.02613449748052235100,
0.00007186850982967301, 0.00008944308067733935},
{0.24929159548532970000, 0.02371140434824439000,
0.00058082625051173115, 0.00034131746889698396}, //Patrones

```
{0.24446916139297462000, 0.02395926731414437700,
0.00005253773711350976, 0.00012434524103171050},     //C
};

float e=0.23; //Factor de Aprendizaje de la RNA
int i=0;                //Variable de control para ciclos for
int ite=0;              //Variable que lleva el conteo de las Iteraciones del
Entrenamiento
int y=0;                //"y" Va a ser el valor de fila que van a tomar como valores de
salida esperados para el conjunto
                              //de datos de aprendizaje

//Funcion Red Neuronal, se encuentra la conexión Neuronal
void RedN();
//Funcion Recalcula Pesos, Se encarga de adaptar los pesos Sinápticos
void RecalPesos();

///////////////FUNCION PRINCIPAL/////////////////
int main(){
        //Llenamos el vector de pesos con valores aleatorios
        /*srand (time(NULL));
          for (int k=0;k<length(Pesos);k++ ){              //Funciona con tabla
unidimensional
                Pesos[k]=-1+rand()%3;
                cout<< Pesos[k] <<"  ";
        }
        //cout<<endl;
        //cout<<row(Entradas)<<endl<<endl;
        //cout<<length(SalEsp)<<endl<< length(Pesos)<<endl;
        system("PAUSE");*/

        system("PAUSE");

        i=0;    //Reinicia la variable i.

        while (i<row(Entradas)){    //Ciclo while para control. Se ejecuta hasta que el
entrenamiento se complete

                RedN();       //Llama la función RedN() para que todas las Neuronas
Operen

                //IF para separar los patrones correctos de los erróneos

                if (i<25){y=0;}
                else if (i>24 && i<50){y=1;}
```

```
                else if (i>49 && i<75){y=2;}
                else if (i>74 && i<100){y=3;}
                else if (i>99 && i<125){y=4;}
                else {y=5;}

                        //Si la salida del Perceptrón de la capa de Salida es igual a la
Salida Esperada el cód seguirá, de no ser así Corregirá Pesos
                        if (Salida11==SalEsp[y][0] )
                        {cout<<"\n \n Salida Neuronal. Iteración "<<i<<" Correcta \n";
                        cout<<"\n Correcta, Salida[11] "<<Salida[11]<<endl;
        //Salida Neurona 11
                        cout<<" Correcta, Salida11 "<<Salida11<<endl<<endl;i++;}
        //Salida Correspondiente según clasificación; Continua con el siguiente
Patrón
                        else {cout<<"\n \n Salida Neuronal. Iteración "<<i<<" Incorrecta
\n \n";
                        cout<<" Incorrecta, Salida[11] "<<Salida[11]<<endl;
        //Salida Neurona 11
                        cout<<" Incorrecta, Salida11 "<<Salida11<<endl;
                //Salida Correspondiente según clasificación
                        cout<<" Corrección de Pesos \n"<<endl;RecalPesos();i=0;}
        //Recalcula Pesos; Reinicia i=0


                for (int j=0;j<length(Pesos);j++ ){          //Ciclo for que imprime los
pesos Calculados
                cout<< Pesos[j] <<",  ";
                }
        }       //end while

        cout<<endl;
        system("PAUSE");
}

//////////////////FUNCION RECALCULA PESOS//////////////////////
void RecalPesos(){

        ////DIFERENCIAS O ERROR DE LA SEÑAL////

        d[11]=SalEsp[y][0]-Salida[11];     //Error de la Neurona 11. Se comparó con
la Salida Esperada
        //Propagación del error hacia atrás
        d[7]=d[11]*Pesos[56];       //Error de las Neuronas de la Segunda Capa
Oculta.
        d[8]=d[11]*Pesos[57];
```

```
d[9]=d[11]*Pesos[58];
d[10]=d[11]*Pesos[59];
//Propagación del error hacia atrás
d[0]=d[7]*Pesos[28] + d[8]*Pesos[29] + d[9]*Pesos[30] + d[10]*Pesos[31];
//Error de las Neuronas de la Primera Capa Oculta.
d[1]=d[7]*Pesos[32] + d[8]*Pesos[33] + d[9]*Pesos[34] + d[10]*Pesos[35];
d[2]=d[7]*Pesos[36] + d[8]*Pesos[37] + d[9]*Pesos[38] + d[10]*Pesos[39];
d[3]=d[7]*Pesos[40] + d[8]*Pesos[41] + d[9]*Pesos[42] + d[10]*Pesos[43];
d[4]=d[7]*Pesos[44] + d[8]*Pesos[45] + d[9]*Pesos[46] + d[10]*Pesos[47];
d[5]=d[7]*Pesos[48] + d[8]*Pesos[49] + d[9]*Pesos[50] + d[10]*Pesos[51];
d[6]=d[7]*Pesos[52] + d[8]*Pesos[53] + d[9]*Pesos[54] + d[10]*Pesos[55];


////////PRIMERA CAPA OCULTA//////////CORRECCION DE PESOS
SINAPTICOS/////////

Pesos[0]=Pesos[0]+(e*d[0]*Dervsigm(Entradas [i][0])*Entradas [i][0]);
Pesos[7]=Pesos[7]+(e*d[0]*Dervsigm(Entradas [i][1])*Entradas [i][1]);
Pesos[14]=Pesos[14]+(e*d[0]*Dervsigm(Entradas [i][2])*Entradas [i][2]);
Pesos[21]=Pesos[21]+(e*d[0]*Dervsigm(Entradas [i][3])*Entradas [i][3]);

Pesos[1]=Pesos[1]+(e*d[1]*Dervsigm(Entradas [i][0])*Entradas [i][0]);
Pesos[8]=Pesos[8]+(e*d[1]*Dervsigm(Entradas [i][1])*Entradas [i][1]);
Pesos[15]=Pesos[15]+(e*d[1]*Dervsigm(Entradas [i][2])*Entradas [i][2]);
Pesos[22]=Pesos[22]+(e*d[1]*Dervsigm(Entradas [i][3])*Entradas [i][3]);

Pesos[2]=Pesos[2]+(e*d[2]*Dervsigm(Entradas [i][0])*Entradas [i][0]);
Pesos[9]=Pesos[9]+(e*d[2]*Dervsigm(Entradas [i][1])*Entradas [i][1]);
Pesos[16]=Pesos[16]+(e*d[2]*Dervsigm(Entradas [i][2])*Entradas [i][2]);
Pesos[23]=Pesos[23]+(e*d[2]*Dervsigm(Entradas [i][3])*Entradas [i][3]);

Pesos[3]=Pesos[3]+(e*d[3]*Dervsigm(Entradas [i][0])*Entradas [i][0]);
Pesos[10]=Pesos[10]+(e*d[3]*Dervsigm(Entradas [i][1])*Entradas [i][1]);
Pesos[17]=Pesos[17]+(e*d[3]*Dervsigm(Entradas [i][2])*Entradas [i][2]);
Pesos[24]=Pesos[24]+(e*d[3]*Dervsigm(Entradas [i][3])*Entradas [i][3]);

Pesos[4]=Pesos[4]+(e*d[4]*Dervsigm(Entradas [i][0])*Entradas [i][0]);
Pesos[11]=Pesos[11]+(e*d[4]*Dervsigm(Entradas [i][1])*Entradas [i][1]);
Pesos[18]=Pesos[18]+(e*d[4]*Dervsigm(Entradas [i][2])*Entradas [i][2]);
Pesos[25]=Pesos[25]+(e*d[4]*Dervsigm(Entradas [i][3])*Entradas [i][3]);

Pesos[5]=Pesos[5]+(e*d[5]*Dervsigm(Entradas [i][0])*Entradas [i][0]);
Pesos[12]=Pesos[12]+(e*d[5]*Dervsigm(Entradas [i][1])*Entradas [i][1]);
Pesos[19]=Pesos[19]+(e*d[5]*Dervsigm(Entradas [i][2])*Entradas [i][2]);
Pesos[26]=Pesos[26]+(e*d[5]*Dervsigm(Entradas [i][3])*Entradas [i][3]);
```

```
Pesos[6]=Pesos[6]+(e*d[6]*Dervsigm(Entradas [i][0])*Entradas [i][0]);
Pesos[13]=Pesos[13]+(e*d[6]*Dervsigm(Entradas [i][1])*Entradas [i][1]);
Pesos[20]=Pesos[20]+(e*d[6]*Dervsigm(Entradas [i][2])*Entradas [i][2]);
Pesos[27]=Pesos[27]+(e*d[6]*Dervsigm(Entradas [i][3])*Entradas [i][3]);


        ////////SEGUNDA CAPA OCULTA//////////CORRECCION DE PESOS
SINAPTICOS/////////

Pesos[28]=Pesos[28]+(e*d[7]*Dervsigm(Salida[0])*Salida[0]);
Pesos[32]=Pesos[32]+(e*d[7]*Dervsigm(Salida[1])*Salida[1]);
Pesos[36]=Pesos[36]+(e*d[7]*Dervsigm(Salida[2])*Salida[2]);
Pesos[40]=Pesos[40]+(e*d[7]*Dervsigm(Salida[3])*Salida[3]);
Pesos[44]=Pesos[44]+(e*d[7]*Dervsigm(Salida[4])*Salida[4]);
Pesos[48]=Pesos[48]+(e*d[7]*Dervsigm(Salida[5])*Salida[5]);
Pesos[52]=Pesos[52]+(e*d[7]*Dervsigm(Salida[6])*Salida[6]);

Pesos[29]=Pesos[29]+(e*d[8]*Dervsigm(Salida[0])*Salida[0]);
Pesos[33]=Pesos[33]+(e*d[8]*Dervsigm(Salida[1])*Salida[1]);
Pesos[37]=Pesos[37]+(e*d[8]*Dervsigm(Salida[2])*Salida[2]);
Pesos[41]=Pesos[41]+(e*d[8]*Dervsigm(Salida[3])*Salida[3]);
Pesos[45]=Pesos[45]+(e*d[8]*Dervsigm(Salida[4])*Salida[4]);
Pesos[49]=Pesos[49]+(e*d[8]*Dervsigm(Salida[5])*Salida[5]);
Pesos[53]=Pesos[53]+(e*d[8]*Dervsigm(Salida[6])*Salida[6]);

Pesos[30]=Pesos[30]+(e*d[9]*Dervsigm(Salida[0])*Salida[0]);
Pesos[34]=Pesos[34]+(e*d[9]*Dervsigm(Salida[1])*Salida[1]);
Pesos[38]=Pesos[38]+(e*d[9]*Dervsigm(Salida[2])*Salida[2]);
Pesos[42]=Pesos[42]+(e*d[9]*Dervsigm(Salida[3])*Salida[3]);
Pesos[46]=Pesos[46]+(e*d[9]*Dervsigm(Salida[4])*Salida[4]);
Pesos[50]=Pesos[50]+(e*d[9]*Dervsigm(Salida[5])*Salida[5]);
Pesos[54]=Pesos[54]+(e*d[9]*Dervsigm(Salida[6])*Salida[6]);

Pesos[31]=Pesos[31]+(e*d[10]*Dervsigm(Salida[0])*Salida[0]);
Pesos[35]=Pesos[35]+(e*d[10]*Dervsigm(Salida[1])*Salida[1]);
Pesos[39]=Pesos[39]+(e*d[10]*Dervsigm(Salida[2])*Salida[2]);
Pesos[43]=Pesos[43]+(e*d[10]*Dervsigm(Salida[3])*Salida[3]);
Pesos[47]=Pesos[47]+(e*d[10]*Dervsigm(Salida[4])*Salida[4]);
Pesos[51]=Pesos[51]+(e*d[10]*Dervsigm(Salida[5])*Salida[5]);
Pesos[55]=Pesos[55]+(e*d[10]*Dervsigm(Salida[6])*Salida[6]);


        ////////CAPA DE SALIDA //////////////CORRECCION DE PESOS
SINAPTICOS/////////
```

```
        Pesos[56]=Pesos[56]+(e*d[11]*Dervsigm(Salida[7])*Salida[7]);
        Pesos[57]=Pesos[57]+(e*d[11]*Dervsigm(Salida[8])*Salida[8]);
        Pesos[58]=Pesos[58]+(e*d[11]*Dervsigm(Salida[9])*Salida[9]);
        Pesos[59]=Pesos[59]+(e*d[11]*Dervsigm(Salida[10])*Salida[10]);


                cout<<endl<<"Iteración Numero "<<ite++<<endl;
                cout<<endl;
}

////////////////////////////////////FUNCION RED NEURONAL
ARTIFICIAL//////////////////////////////////////////////////////
void RedN(){          //Operan cada una de las Neuronas

        //Salidas de los perceptrones de la Prinera Capa Oculta
        Salida[0]=Perceptron1C(Pesos[0],Pesos[7],Pesos[14],Pesos[21],Entradas
[i][0],Entradas [i][1],Entradas [i][2],Entradas [i][3]);
        Salida[1]=Perceptron1C(Pesos[1],Pesos[8],Pesos[15],Pesos[22],Entradas
[i][0],Entradas [i][1],Entradas [i][2],Entradas [i][3]);
        Salida[2]=Perceptron1C(Pesos[2],Pesos[9],Pesos[16],Pesos[23],Entradas
[i][0],Entradas [i][1],Entradas [i][2],Entradas [i][3]);
        Salida[3]=Perceptron1C(Pesos[3],Pesos[10],Pesos[17],Pesos[24],Entradas
[i][0],Entradas [i][1],Entradas [i][2],Entradas [i][3]);
        Salida[4]=Perceptron1C(Pesos[4],Pesos[11],Pesos[18],Pesos[25],Entradas
[i][0],Entradas [i][1],Entradas [i][2],Entradas [i][3]);
        Salida[5]=Perceptron1C(Pesos[5],Pesos[12],Pesos[19],Pesos[26],Entradas
[i][0],Entradas [i][1],Entradas [i][2],Entradas [i][3]);
        Salida[6]=Perceptron1C(Pesos[6],Pesos[13],Pesos[20],Pesos[27],Entradas
[i][0],Entradas [i][1],Entradas [i][2],Entradas [i][3]);

        //Salidas de los perceptrones de la Segunda Capa Oculta
        Salida[7]=Perceptron2C(Pesos[28],Pesos[32],Pesos[36],Pesos[40],Pesos[4
4],Pesos[48],Pesos[52],Salida[0],Salida[1],Salida[2],Salida[3],Salida[4],Salida[5],Sa
lida[6]);
        Salida[8]=Perceptron2C(Pesos[29],Pesos[33],Pesos[37],Pesos[41],Pesos[4
5],Pesos[49],Pesos[53],Salida[0],Salida[1],Salida[2],Salida[3],Salida[4],Salida[5],Sa
lida[6]);
        Salida[9]=Perceptron2C(Pesos[30],Pesos[34],Pesos[38],Pesos[42],Pesos[4
6],Pesos[50],Pesos[54],Salida[0],Salida[1],Salida[2],Salida[3],Salida[4],Salida[5],Sa
lida[6]);
        Salida[10]=Perceptron2C(Pesos[31],Pesos[35],Pesos[39],Pesos[43],Pesos[
47],Pesos[51],Pesos[55],Salida[0],Salida[1],Salida[2],Salida[3],Salida[4],Salida[5],S
alida[6]);
```

```
        //Salida del perceptron de la Ultima Capa (Se usa el modelo de Primera
Capa)
        Salida[11]=Perceptron1C(Pesos[56],Pesos[57],Pesos[58],Pesos[59],Salida[
7],Salida[8],Salida[9],Salida[10]);

        //Asignación del Valor Promedio de Salida, según los valores Deseados
        if (Salida[11]>=0.5199 && Salida[11]<0.5338){          //%    "S"
        Salida11=0.52;}
        else if (Salida[11]>0.548 && Salida[11]<0.564){                //%    "E"
        Salida11=0.55;}
        else if (Salida[11]>0.585 && Salida[11]<0.607){                //%    "L"
        Salida11=0.58;}
        else if (Salida[11]>0.62 && Salida[11]<0.637){         //%    "3"
        Salida11=0.62;}
        else if (Salida[11]>0.758 && Salida[11]<0.7809){       //%    "W"
        Salida11=0.76;}
        else if (Salida[11]>0.7286 && Salida[11]<0.756){       //%    "C"
        Salida11=0.74;}

        else {Salida11=Salida[11];}
}
```

ANEXO D.
CÓDIGO EN LENGUAJE C++ PARA EL RECONOCIMIENTO DE GESTOS ASL

```
//Incluir librerías requeridas
#include <opencv2\opencv.hpp>
#include <opencv2\core\core.hpp>
#include <opencv2\highgui\highgui.hpp>
#include <opencv2\imgproc\imgproc.hpp>
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <vector>
#include <time.h>
#include <math.h>

////////////////////////////////////////////////  PESOS RED NEURONAL//////////////////////////////////

double Pesos [60]={0.563414,  -0.948462,  -0.0686181,  0.554486,  2.58774,  -
0.531434,  -0.937991, 1.52534,  -1.32484,
        0.862763,  2.68047,  4.78058,  -1.66305,  -0.108049,  0.941363,
0.0533465,  -0.999877,  -1.06865,  -1.18462,
        -0.935077,  0.0416234,  0.000704266,  0.999127,  0.000610643,
0.00230245,  -0.997687,  0.998959,  -0.999514,
        0.224686,  -1.25148,  -0.747072,  1.25472,  -0.462521,  0.507966,  -
0.486946,  -0.508134,  -0.544667,  -0.621564,
        -0.434253,  0.646598,  1.4779,  0.351426,  1.61081,  0.676577,  0.767881,
-1.75855,  1.76042,  1.74335,  0.787237,
        0.237742,  -1.23926,  -1.24074,  0.789475,  1.1202,  0.864335,  -1.09712,
0.967123,  -1.24202,  1.37296,  1.31888};
//  PESOS RESULTADOS DEL ENTRENAMIENTO DE LA RED NEURONAL
ARTIFICIAL

double Salida [12];  //Arreglo de 12 posiciones para las salidas de cada una de las
12 Neuronas de la RNA
double Salida11;     //Variable Salida11, se usa para comparar la respuesta de la
RNA con las salidas esperadas
double h[7];         //Arreglo de 7 posiciones para guardar los 7 Momentos
Invariantes de Imagen
double x=0;  //"x" Variable para las operaciones de los Perceptrones o Neuronas

int i=0;             //Variable de control para ciclos for

////////////////Función de activación de las neuronas (Sigmoide)////////////////
double sigm(double s){
```

```
        double sig;                                    //Variable de operación función
Sigmoide
        sig=-1+(2/(1+(exp(-1*s))));        //Función de activación de la Neurona
(Sigmoide bipolar Max=1; Min=-1)
        return sig;
}

//Función del modelo de Perceptrón de la primera capa (Neuronas)----w(pesos de
la conexiones sinápticas),--pat(patrones de entrada)
double Perceptron1C(double w0,double w1,double w2,double w3,double
pat0,double pat1,double pat2,double pat3){
        double outPer1C;            //Variable para la operación de "Salida del
Perceptrón" outPer
        x = (w0*pat0)+(w1*pat1)+(w2*pat2)+(w3*pat3);        //Operación Neuronal
(Comportamiento de la Neurona)
        outPer1C=sigm(x);
        //Respuesta Neuronal, con la función de excitación tipo Sigmoide Bipolar
        return outPer1C;                                                    //Retorna
respuesta
}

//Función del modelo de Perceptrón de la segunda capa (Neuronas)----w(pesos de
la conexiones sinápticas),--sal(patrones de entrada)
double Perceptron2C(double w0,double w1,double w2,double w3,double
w4,double w5,double w6,double sal0,double sal1,double sal2,double sal3,double
sal4,double sal5,double sal6){
        double outPer2C;            //Variable para la operación de "Salida del
Perceptrón" outPer
        x =
(w0*sal0)+(w1*sal1)+(w2*sal2)+(w3*sal3)+(w4*sal4)+(w5*sal5)+(w6*sal6);
        //Operación Neuronal (Comportamiento de la Neurona)
        outPer2C=sigm(x);                                   //Respuesta Neuronal, con la
función de excitación tipo Sigmoide Bipolar
        return outPer2C;                                    //Retorna respuesta
}

//Espacio de nombres
#define w 400                       //Constante w = 400; para dimension del
recuadro de la pantalla
using namespace cv;
using namespace std;

///Valores iniciales de los filtros HSV
int H_MIN = 0;
int H_MAX = 0;
```

```
int S_MIN = 0;
int S_MAX = 0;
int V_MIN = 0;
int V_MAX = 0;

//Creamos las matrices de trabajo para video
Mat imagen, HSV, Filtro, imagenbl, image_roi;
int umbral = 150;     //Umbral para filtro Canny

//Constantes string (Nombramos las ventanas)
string  windowName1 = "Imagen HSV";
string        MANO = "MANO";

//Hacemos un rectangulo de la seccion que nos interesa (ROI)
Rect roi(0, 0 ,(2*w/3), 2*w/3);                 //Dimensiones del rectángulo

/// FUNCIONES ADICIONALES -
//Función de Contorno y Momentos de HU
void contorno(int, void* );
//Función de Salida con las opciones de respuesta del gesto
void alfabeto(int, void*);
//Función que muestra letras en la pantalla
void Texto(string texto);
//Función RedN, esta contiene la conexión Neuronal
void RedN();
//Función Casos, esta identifica a cual gesto corresponde el patrón
void Casos();

///FUNCION PRINCIPAL
int main(){

        //Iniciamos la captura
        VideoCapture cap;  //Crea matriz cap para video
        cap.open(1);          //Captura la imagen en la matriz cap

        while(1){                       //Ciclo while

                //Copia la imagen de la cámara a la matriz imagen
                cap>>imagen;

                //Se ubica un rectángulo en la pantalla (Región de interés ROI)
                rectangle( imagen,Point( 0, 0 ),Point( 2*w/3, 2*w/3),Scalar( 0, 138, 21
),2,4); //Dibuja el rectángulo (Scalar son los colores BGR)
```

```
                //Convierte del espacio de color RGB al modelo HSV la imagen
capturada
                cvtColor(imagen,HSV,COLOR_BGR2HSV);

                //Se muestran las imágenes capturadas -Cámara  -HSV
                imshow("Alfabeto", imagen);
                imshow("HSV(roi)",HSV(roi));

                //Rango de segmentación de la imagen para el color negro, el
resultado se plasmara en la matriz Filtro
                //El rango del filtro HSV va a estar desde _MIN hasta _MAX de los
valores de
          //Hue, Saturation y Value
                inRange(HSV,Scalar(H_MIN,S_MIN,V_MIN),
                                        Scalar(H_MAX,S_MAX,V_MAX),Filtro);

                //Crear elemento de estructuración para realizar dilatación y erosión
de la imagen (morfología rectangular)
                Mat element = getStructuringElement(MORPH_RECT, Size(3,3));
        //Tamaño 3x3

                //Se crea una matriz temporal para almacenar la erosión/dilatación
                Mat temp;
                //imshow("Extracción", Filtro(roi));
                //Mediante la dilatación y erosión se irá eliminando el ruido
                erode(Filtro,temp,element);
                //imshow("Erosión",temp(roi));
                //Dilatamos un par de veces
                dilate(temp,temp,element);
                //imshow("Dilatación1",temp(roi));
                dilate(temp,temp,element);
                //imshow("Dilatación2",temp(roi));

                //Ubicamos la ROI en una matriz - Es la matriz de donde extraeremos
                la ROI
                //(en este caso temp)
                image_roi = temp(roi);

                //Mostramos la ROI
                imshow(MANO,temp(roi));

                //Filtro "blur" o difuminacion, suaviza la imagen
                blur( image_roi, imagenbl, Size(3,3) );
                //imshow("Imagen suavizada", imagenbl);
```

```
                //Llamamos a las demás funciones
                contorno(0,0);          //Función Contorno
                RedN();                 //Función RNA
                cout<<"\n Salida Red "<<Salida[11]<<endl;      //Imprimimos la
                respuesta de la RNA
                alfabeto(0,0);//Función Alfabeto

                waitKey(33); //Refrescamos cada 33 ms
        }
        return 0;
}

//////////////////////FUNCION CONTORNOS Y OBTENCION DE MOMENTOS DE HU
DE LA IMAGEN//////////////////////////////////////
void contorno(int,void*)
{
  Mat canny_output;          //Matriz de salida del filtro Canny
  vector<vector<Point>> contours;          //Vector de contornos
  vector<Vec4i> hierarchy;                 //Vector de jerarquía para la función de
contornos

  /// Detectar bordes usando filtro "Canny"
  Canny( imagenbl, canny_output, umbral, umbral*2, 3 );
  //imshow("Salida Canny", canny_output);

  /// Encontrar contornos usando "findContours"
  findContours( canny_output, contours, hierarchy, CV_RETR_TREE,
CV_CHAIN_APPROX_SIMPLE, Point(0, 0) );
  printf("\n NUMERO DE CONTORNOS: %d \n",contours.size());              //Imprime
la cantidad de contornos que encuentra en la imagen

  /// IF de control para reconocer la existencia de contornos
   if (contours.size()>0){

  //CONTORNO MAYOR
  int Mayor, Cont1, n=0;     //Variables de control ciclo for
  Cont1=contourArea(contours[0]);          //Se asigna a Cont1 el área del contorno 0
  for( int i = 0; i < contours.size(); i++ )    //Ciclo for
    {
                Mayor=contourArea(contours[i]);
                if (Mayor > Cont1){
                        n=i;
                        Cont1=contourArea(contours[i]);
                }
    }
```

```
  //printf("\n El Contorno Mayor Es El Numero %d  \n",n);

  /// Obtener los MOMENTOS de la imagen con "moments"
  vector<Moments> mm(contours.size());                  //Vector de momentos mm
igual a contours

  mm[n] = moments( contours[n], true );   //Se buscan los momentos del contorno
mayor
  //printf(" * Contour[%d] - Momentos = %.2f %.2f %.2f %.2f %.2f %.2f %.2f
\n", i,
mm[i].m00,mm[i].m01,mm[i].m20,mm[i].m11,mm[i].m02,mm[i].m30,mm[i].m21,mm[i
].m03);          //Momentos Generales
  //printf(" * Contour[%d] - Momentos = %.2f %.2f %.2f %.2f %.2f %.2f %.2f
\n", i,
mm[i].m00,mm[i].mu02,mm[i].mu03,mm[i].mu11,mm[i].mu12,mm[i].mu20,mm[i].mu
21,mm[i].mu30);      //Momentos Cetrales
  //printf(" * Contour[%d] - Momentos = %.2f %.2f %.2f %.2f %.2f %.2f %.2f
\n", i,
mm[i].m00,mm[i].nu02,mm[i].nu03,mm[i].nu11,mm[i].nu12,mm[i].nu20,mm[i].nu21,
mm[i].nu30);  //Momentos Centrales Normalizados
  HuMoments(mm[n],h);
  printf("\n \t MOMENTOS DE HU \n [1] %.20f \n [2] %.20f \n [3] %.20f \n [4] %.20f
\n [5] %.20f \n [6] %.20f \n [7] %.20f \n",h[0],h[1],h[2],h[3],h[4],h[5],h[6]);
//Momentos Invariantes de HU

  /// Encontrar el Centro de Masa del Contorno
  vector<Point2f> mc (contours.size());             //Puntos Centro de Masa (mc)

  mc[n] = Point2f( mm[n].m10/mm[n].m00 , mm[n].m01/mm[n].m00 ); //Centro de
Masa a partir de los Momentos Generales de primer orden

  /// Dibujar contornos con "drawContours"
  Mat drawing = Mat::zeros( canny_output.size(), CV_8UC3 );

  Scalar color = Scalar( 181,131,70 );     //Color Contorno BGR
  Scalar colormc = Scalar( 18,11,250 );  //Color Centro de Masa BGR
  drawContours( drawing, contours, n, color, 2, 8, hierarchy, 0, Point() );    //Dibuja
contorno n
  circle( drawing, mc[n], 5, color, -1, 8, 0 );                 //Dibuja Centros de Masa
  circle( imagen, mc[n], 5, colormc, -1, 8, 0 );    //Dibuja Centros de Masa

  /// Mostramos en una ventana
  imshow( "Contorno", drawing );
  }
```

```
  else {                              //Si no encuentra contornos regresa las variables
a 0
            waitKey(300);
            //main();
            h[0]=0;
            h[1]=0;
            h[2]=0;
            h[3]=0;
            h[4]=0;
            h[5]=0;
            h[6]=0;
  }
}

////////////////////////////////////////////RED NEURONAL////////////////////////////////////////////////
void RedN(){          //Conexiones Neuronales

          //Salidas de los perceptrones de la Primera Capa Oculta
          Salida[0]=Perceptron1C(Pesos[0],Pesos[7],Pesos[14],Pesos[21],h[0],h[1],h[
2],h[3]);
          Salida[1]=Perceptron1C(Pesos[1],Pesos[8],Pesos[15],Pesos[22],h[0],h[1],h[
2],h[3]);
          Salida[2]=Perceptron1C(Pesos[2],Pesos[9],Pesos[16],Pesos[23],h[0],h[1],h[
2],h[3]);
          Salida[3]=Perceptron1C(Pesos[3],Pesos[10],Pesos[17],Pesos[24],h[0],h[1],h
[2],h[3]);
          Salida[4]=Perceptron1C(Pesos[4],Pesos[11],Pesos[18],Pesos[25],h[0],h[1],h
[2],h[3]);
          Salida[5]=Perceptron1C(Pesos[5],Pesos[12],Pesos[19],Pesos[26],h[0],h[1],h
[2],h[3]);
          Salida[6]=Perceptron1C(Pesos[6],Pesos[13],Pesos[20],Pesos[27],h[0],h[1],h
[2],h[3]);

          //Salidas de los perceptrones de la Segunda Capa Oculta
          Salida[7]=Perceptron2C(Pesos[28],Pesos[32],Pesos[36],Pesos[40],Pesos[4
4],Pesos[48],Pesos[52],Salida[0],Salida[1],Salida[2],Salida[3],Salida[4],Salida[5],Sa
lida[6]);
          Salida[8]=Perceptron2C(Pesos[29],Pesos[33],Pesos[37],Pesos[41],Pesos[4
5],Pesos[49],Pesos[53],Salida[0],Salida[1],Salida[2],Salida[3],Salida[4],Salida[5],Sa
lida[6]);
          Salida[9]=Perceptron2C(Pesos[30],Pesos[34],Pesos[38],Pesos[42],Pesos[4
6],Pesos[50],Pesos[54],Salida[0],Salida[1],Salida[2],Salida[3],Salida[4],Salida[5],Sa
lida[6]);
```

```
	Salida[10]=Perceptron2C(Pesos[31],Pesos[35],Pesos[39],Pesos[43],Pesos[
47],Pesos[51],Pesos[55],Salida[0],Salida[1],Salida[2],Salida[3],Salida[4],Salida[5],S
alida[6]);

	//Salida del perceptrón de la Ultima Capa (Se usa el modelo de Primera
Capa)
	Salida[11]=Perceptron1C(Pesos[56],Pesos[57],Pesos[58],Pesos[59],Salida[
7],Salida[8],Salida[9],Salida[10]);

	Casos();	//Llama a la Función Casos()
}

////////////////////////////FUNCION CASOS/////////////////////////////////////////
void Casos(){				//Determina a que Clase corresponde el Patrón
de entrada

	if (Salida[11]>0.62 && Salida[11]<0.644){			//%	"3"
	Salida11=0.62;}
	else if (Salida[11]>0.645 && Salida[11]<0.664){		//%	"5"
	Salida11=0.63;}
	else if (Salida[11]>0.697 && Salida[11]<0.7163){	//%	"7"
	Salida11=0.7;}
	else if (Salida[11]>0.679 && Salida[11]<0.696){		//%	"8"
	Salida11=0.69;}
	else if (Salida[11]>0.7286 && Salida[11]<0.756){	//%	"C"
	Salida11=0.74;}
	else if (Salida[11]>0.548 && Salida[11]<0.566){		//%	"E"
	Salida11=0.55;}
	else if (Salida[11]>0.585 && Salida[11]<0.607){		//%	"L"
	Salida11=0.58;}
	else if (Salida[11]>=0.5199 && Salida[11]<0.5338){	//%	"S"
	Salida11=0.52;}
	else if (Salida[11]>0.758 && Salida[11]<0.7849){	//%	"W"
	Salida11=0.76;}
	else if (Salida[11]>0.536 && Salida[11]<0.545){		//%	"A"
	Salida11=0.53;}
	else if (Salida[11]>0.575 && Salida[11]<0.585){		//%	"T"
	Salida11=0.57;}

	else {Salida11=Salida[11];}
}
```

```
//////////////////////FUNCION ALFABETO/////////////////////////////////
void alfabeto(int, void*)
{
	//TEXTO EN PANTALLA

	if (Salida11 == 0.62){
		string texto = "3";
		Texto(texto);		//Llama a la Función Texto()
		}
	else if (Salida11 == 0.63){
			string texto = "5";
			Texto(texto);
		}
	else if (Salida11 == 0.7){
			string texto = "7";
			Texto(texto);
		}
	else if (Salida11 == 0.69){
			string texto = "8";
			Texto(texto);
		}
	else if (Salida11 == 0.74){
			string texto = "C";
			Texto(texto);
		}
	else if (Salida11 == 0.55){
			string texto = "E";
			Texto(texto);
		}
	else if (Salida11 == 0.58){
			string texto = "L";
			Texto(texto);
		}
	else if (Salida11 == 0.52){
			string texto = "S";
			Texto(texto);
		}
	else if (Salida11 == 0.76){
			string texto = "W";
			Texto(texto);
		}
	else if (Salida11 == 0.53){
			string texto = "A";
			Texto(texto);
		}
```

```
        else if (Salida11 == 0.57){
                        string texto = "T";
                        Texto(texto);
                }
        else {
        ///SI NO HAY MANO
                string texto = "??";
                Texto(texto);
                }
}

//////////////////////////FUNCION TEXTO/////////////////////////////////////////////////////
void Texto(string texto){
                int fontFace = FONT_HERSHEY_COMPLEX;  //Fuente de la imagen
                /*The font type, one of FONT_HERSHEY_SIMPLEX ,
                FONT_HERSHEY_PLAIN , FONT_HERSHEY_DUPLEX ,
FONT_HERSHEY_COMPLEX , FONT_HERSHEY_TRIPLEX ,
                FONT_HERSHEY_COMPLEX_SMALL ,
FONT_HERSHEY_SCRIPT_SIMPLEX or FONT_HERSHEY_SCRIPT_COMPLEX
                Where each of the font id's can be combined with
FONT_HERSHEY_ITALIC to get the slanted letters.*/
                double fontScale = 2;        //Escala de la letra
                Scalar ColorText = Scalar( 202, 15, 40 );         //Color de la letra
                int thickness = 4;
                int baseline=0;

                Size textSize = getTextSize(texto, fontFace,fontScale, thickness,
&baseline);

                //Posicion de la letra
                Point textOrg((imagen.cols - textSize.width)/1.02,
                                        (imagen.rows + textSize.height)/5);

                //Colocamos el texto en la pantalla
                putText(imagen, texto, textOrg, fontFace, fontScale,ColorText,
thickness, 8,false);

                //Mostrar en ventana
                imshow("Alfabeto", imagen);
}
```