UNIVERSIDAD DE ORIENTE
NÚCLEO DE ANZOÁTEGUI
ESCUELA DE INGENIERÍA Y CIENCIAS APLICADAS
DEPARTAMENTO DE COMPUTACIÓN Y SISTEMAS

"DESARROLLO DE UNA APLICACIÓN PARA EL RECONOCIMIENTO AUTOMÁTICO DE PLACAS A PARTIR DE LA IMÁGEN DIGITAL DE UN VEHÍCULO"

REALIZADO POR:

Martínez. M, Luis Edmundo

Zapata. B, Ronel José

Trabajo de grado presentado como requisito parcial para optar al Título de

INGENIERO EN COMPUTACIÓN

Barcelona, Marzo de 2009

UNIVERSIDAD DE ORIENTE
NÚCLEO DE ANZOÁTEGUI
ESCUELA DE INGENIERÍA Y CIENCIAS APLICADAS
DEPARTAMENTO DE COMPUTACIÓN Y SISTEMAS

"DESARROLLO DE UNA APLICACIÓN PARA EL RECONOCIMIENTO AUTOMÁTICO DE PLACAS A PARTIR DE LA IMÁGEN DIGITAL DE UN VEHÍCULO"

ASESOR:

_____
Lic. José Luis Bastardo. Msc.
Asesor Académico

Barcelona, Marzo de 2009

UNIVERSIDAD DE ORIENTE

NÚCLEO DE ANZOÁTEGUI

ESCUELA DE INGENIERÍA Y CIENCIAS APLICADAS

DEPARTAMENTO DE COMPUTACIÓN Y SISTEMAS

"DESARROLLO DE UNA APLICACIÓN PARA EL RECONOCIMIENTO AUTOMÁTICO DE PLACAS A PARTIR DE LA IMÁGEN DIGITAL DE UN VEHÍCULO"

JURADO CALIFICADOR:

_____

Lic. José Luis Bastardo. Msc.
Asesor Académico

_____

Profesor Claudio Cortínez
Jurado Principal

_____

Profesor Pedro Dorta
Jurado Principal

Barcelona, Marzo de 2009

RESOLUCION

ARTÍCULO Nº 44 Del Reglamento de Trabajo de Grado

"Los trabajos de grado son de exclusiva propiedad de la Universidad y sólo podrán ser utilizados para otros fines con el conocimiento del Consejo de Núcleo respectivo, quién lo participará al Consejo Universitario".

RESUMEN

En el presente trabajo se desarrolla una Aplicación que permite el reconocimiento automático de placas a partir de la imagen digital de un vehículo. Se emplean diversas técnicas de visión artificial como lo son las operaciones morfológicas, el método de segmentación para determinar la ubicación de la placa y Redes Neuronales artificiales entrenadas con el algoritmo Backpropagation para el reconocimiento de los caracteres que se encuentran dentro de la misma.

AGRADECIMIENTOS

A la universidad por darme tantas experiencias en mi vida.

A mis padres Ramon Martinez y Dighla Mendoza por ser mi apoyo y mi ejemplo a seguir.

A mis hermanos Dira y Ramon por impulsar mi locura y ser mis consejeros.

A mi chica cósmica (desy) por estar y ser ella.

A mi hermana marianna (ñaña) graziani por regañarme y compartir tantas cosas conmigo.

A mi amiga enoes (noes) medina por no ser mezquina con su conocimiento y ayudarme a finiquitar la tesis.

A mi hermano luis (chino) cordero, le debo hasta el alma.

A mis amigos Israel, Luifer, Ronel, HP, Jose (enano) A gonsalez, Lili, Mafe, vane por darme todo el animo del mundo.

A los profesores José Luis Bastardo y Claudio Cortínez por ser mis asesores y guías durante el desarrollo de este proyecto.

A mis secretarias preferidas, Maria, Ysabel, Dayana y Carmen hacerme mas favores hubiera sido imposible.

A todos mis tios, familiares y amigos que compartieron conmigo y me brindaron apoyo en algún momento de esta etapa universitaria.

Luis Edmundo Martínez Mendoza.

AGRADECIMIENTOS

A Dios por haberme dado la confianza, perseverancia y tenacidad para sobrellevar todas las adversidades y culminar este capítulo de mi vida. A mis padres Zulme Beltrán y Ronel Zapata por su apoyo y amor incondicional, ellos que siempre están a mi lado dando su apoyo y ayudándome de la mejor manera, a mi hermano Luis Mauricio por haberme ayudado.

A mis tíos Luis Francisco, Francisco Javier, Mirna, Lubia, Nidia, Leudys Beltrán que siempre están pendiente de mi recordándome el camino que debo seguir en la vida y estar conmigo siempre.

A mi prima Andreina y mi tío Naiffer por ayudarme desde pequeño con los estudios y su asesoría.

A Oscar Saavedra, Sair, Omar Marín y José Miguel por ser excelentes compañeros de estudios y excelentes personas desde el inicio de nuestra amistad.

A mis amigos Luis Matrínez, Luis Cordero y Enoes por estar a mi lado en el desarrollo del proyecto de tesis, fueron de gran ayuda en todo momento y un gran apoyo incondicional.

A los profesores José Luis Bastardo y Claudio Cortínez por ser mis asesores y guías durante el desarrollo de este proyecto.

A todos mis familiares y compañeros que de una u otra forma me han brindado su apoyo incondicional a lo largo de mi vida.

Ronel José Zapata Beltrán.

LISTA DE FIGURAS

LISTA DE TABLAS

CAPITULO 1 PLANTEAMIENTO DEL PROBLEMA

1.1 Introducción

La Inteligencia Artificial en los últimos años ha tenido un gran auge, principalmente en el área de Visión Artificial, esto es debido a que se aplica en diversos procesos científicos y militares, extendiéndose su uso además, en un amplio rango de sectores industriales para la automatización de tareas anteriormente reservadas para la inspección visual humana.

La visión artificial es una técnica basada en la adquisición de imágenes, generalmente en dos dimensiones, para luego procesarlas digitalmente mediante computadora, microcontrolador, DSP(*Digital Signal Processor*), etc., con el fin de extraer y medir determinadas propiedades de las imágenes adquiridas.

Dentro de la Visión Artificial se aplican diversos algoritmos y métodos estadísticos para realizar las técnicas de adquisición de imágenes, dentro de los cuales se encuentran:

- Operaciones Morfológicas, éstas son técnicas para el análisis y procesamiento de estructuras geométricas,
- Clustering se encarga de clasificar los objetos en diferentes grupos según sus características,

- Reconocimiento de patrones.

El reconocimiento de patrones y muy particularmente el reconocimiento de caracteres puede ser abordado usando el concepto de redes neuronales.

Los sistemas de Redes Neuronales Artificiales aún cuando están basados en los sistemas neurales biológicos del cerebro, resultan incompletos ya que los sistemas neurales biológicos son infinitamente complejos.

Las Redes Neuronales Artificiales son un paradigma de aprendizaje y procesamiento automático inspirado en la forma en que funciona el sistema nervioso de los animales.

Las Redes Neurales al experimentar cambios entre sus conexiones pueden aprender, estos cambios son conocidos como proceso de entrenamiento, el cual se guía por una regla de aprendizaje, donde la experiencia es la que impera en las modificaciones que se realicen. Estas son aplicadas con éxito en problemas donde la imprecisión no puede ser ignorada. Entre los problemas que se benefician de esta característica están los de reconocimiento, los de clasificación y los de predicción.

Los modelos de Redes Neuronales Artificiales, combinan modelos matemáticos de las células nerviosas y modelos de arquitecturas que describen las interconexiones que existen entre estas células. Dentro de estos modelos existen redes que por su topología se pueden clasificar como monocapas o multicapas.

El algoritmo Backpropagation es una regla de aprendizaje que se puede aplicar a modelos con más de dos capas. Una característica importante de este algoritmo es la capacidad de organizar en la capa intermedia la información para conseguir cualquier correspondencia entre la entrada y salida de la red.

El funcionamiento de la red BackPropagation consiste en un aprendizaje de un conjunto de pares de entradas salidas dados como ejemplo, mediante un ciclo propagación adaptación de dos fases. Primero se aplica un patrón de entrada, que se va propagando a través de todas las capas superiores hasta generar una salida, se compara este resultado con la salida que se desea obtener y se calcula un valor del error para cada neurona de salida. Luego el error se transmite hacia atrás partiendo de la salida hacia todas las neuronas de la capa intermedia, este proceso se repite capa por capa. De ahí todas las neuronas reciben un error que describe el aporte relativo de la neurona al error total. Con esté valor de error se ajustan los pesos de cada neurona de manera que la siguiente vez que se presente el mismo patrón el error disminuya.

1.2 El problema

Se tiene una imagen estandarizada de un vehículo de la cual se desea extraer el número de placa del mismo, utilizando operaciones morfológicas (Dilatación, Erosión, Resta de imágenes, Apertura, Cierre) para la fase de localización, luego se utilizarán los métodos de segmentación para dividir la imagen en regiones u objetos cuyos pixeles posean atributos similares y por último se utilizará el algoritmo Backpropagation para el reconocimiento de cada uno de los caracteres de la placa.

1.3 El propósito

El propósito de este proyecto es desarrollar una aplicación capaz de encontrar una solución al problema planteado.

La morfología matemática es una herramienta muy utilizada en el procesamiento de imágenes. Las operaciones morfológicas pueden simplificar los datos, preservar las características esenciales y eliminar aspectos irrelevantes de una imagen, esto hace que la implementación de esta metodología sea la idónea para este proyecto.

La segmentación es uno de los procesos más importantes de un sistema automatizado de visión ya que permite extraer los objetos de la imagen para su posterior descripción y reconocimiento, este es un paso indispensable para la extracción de los dígitos de la placa del carro.

1.4 Importancia

Académicamente, éste proyecto servirá como base a otros proyectos de desarrollo de software, orientados a la utilización de herramientas de Redes Neuronales Artificiales y Visión Artificial, siguiendo un proceso sistematizado y organizado para su análisis y desarrollo.

1.5 Alcance

En el desarrollo de este estudio, se especificarán los requisitos del sistema y sus riesgos; se realizarán los diagramas pertinentes, al igual que la arquitectura del sistema propuesto, se trabajará con los formatos de imagen del tipo .JPEG, .TIFF, .bmp, restringiendo su tamaño a una resolución de 1024 x 768 pixeles para su posterior evaluación, se diseñará y construirá una interfaz accesible, interactiva, amigable y de fácil manejo para los usuarios.

1.6 Originalidad

Es esencial mencionar que este proyecto es el primero en el Departamento de Computación y Sistemas en utilizar computación emergente específicamente el área de redes neuronales para la identificación de caracteres en conjunto con la aplicación de la visión artificial para el reconocimiento automático de placas a partir de la imagen digital de un vehículo.

1.7 Objetivos

1.7.1 Objetivo general

- Desarrollar una aplicación para el reconocimiento automático de placas a partir de la imagen digital de un vehículo.

1.7.2 Objetivos específicos

- Caracterizar la data de entrada.
- Diseñar los algoritmos de preprocesamiento de la data.
- Diseñar los algoritmos de reconocimiento de patrones para la ubicación de la placa.
- Diseñar los algoritmos de reconocimiento de patrones para el reconocimiento de los caracteres.
- Codificar los algoritmos.
- Realizar las pruebas de unidad e integración del sistema.

CAPITULO 2 MARCO TEÓRICO

2.1 Antecedentes

Los proyectos usados como antecedentes están basados en visión artificial.

- Para el año 2004, los estudiantes Juan Díaz y Lila Fermín, de la Universidad de Oriente Núcleo Anzoátegui, desarrollaron el proyecto titulado **"Desarrollo de un software para el reconocimiento de caracteres alfabéticos manuscritos mediante Algoritmos de Visión Artificial"**, para optar al título de Ingeniero en Computación. Este proyecto se realizó con la finalidad de reconocer la escritura de las personas para agilizar los trámites en los que no se utiliza un computador, el usuario introduce al software una imagen donde se encuentran los caracteres que se desean reconocer y el sistema analiza cada uno de los caracteres contenidos en la imagen y genera por cada carácter su equivalente en código ASCII. Para la codificación se utilizó el Lenguaje de Programación JAVA.[7]

- En el año 2005, los estudiantes Charlie Sulbarán y Sioli Yánez, de la Universidad de Oriente Núcleo Anzoátegui, desarrollaron el proyecto **"Desarrollo de un Agente Inteligente que intercepta la trayectoria de un proyectil que se desplaza en un ambiente limitado de 2 dimensiones, utilizando técnicas de Visión Artificial"**, para optar al título de Ingeniero en Computación. Este proyecto consiste en un sistema que se encarga de monitorear una zona de estudio llamada campo de juego a través de una

cámara (sensor óptico del agente) que permite la entrada de información al sistema, la finalidad es detectar una pelota que será lanzada por un usuario, se evaluarán una secuencia de imágenes para que el agente le haga un seguimiento a la pelota donde se determina su trayectoria y se logra interceptar mediante una barra deslizante (efector). La herramienta utilizada para codificar los módulos fue el Lenguaje de Programación JAVA, junto con un entorno de trabajo llamado JBuilderX que funciona sobre plataforma JAVA.[5]

- Para el año 2007, los estudiantes Desiree Sucre y Abraham Verde, de la Universidad de Oriente Núcleo Anzoátegui, desarrollaron el proyecto titulado **"Desarrollo de un Agente Inteligente que identifique objetos geométricos específicos según su forma o color dentro de un ambiente definido aplicando técnicas de Visión Artificial"**, para optar al título de Ingeniero en Computación. El sistema de monitorear una zona de estudio llamada Campo de Juego a través de una cámara Web que le proporciona la entrada de información al sistema en forma de imágenes, con la finalidad de identificar los objetos geométricos que se encuentren en el ambiente para determinar su forma, color, y ubicación. Las interfaces de usuario fueron desarrolladas en una extensión del Lenguaje PHP llamado PHP-GTK, que utilizan las librería GTK, y el software de control fue codificado en Lenguaje C++.[2]

- Para el año 2007, los estudiantes Luna Raquel y Henry Parada, de la Universidad de Oriente Núcleo Anzoátegui, desarrollaron el proyecto titulado **"Desarrollo de un sistema reconocedor de carácter alfanuméricos, que transforma una imagen escaneada de un manuscrito en un archivo de texto mediante algoritmos de Visión Artificial"**, para optar al título de Ingeniero en Computación. Este proyecto se realizó con la finalidad de

reconocer caracteres alfanuméricos, el sistema compara patrones predefinidos y de acuerdo a su similitud se identifica el carácter, el texto reconocido es almacenado en formato RTF. Se utilizó el Proceso Unificado de Desarrollo de Software para la realización del proyecto, y el Lenguaje de Programación JAVA para su codificación.[3]

2.2 Redes neuronales

Las redes de neuronas artificiales son un paradigma de aprendizaje y procesamiento automático inspirado en la forma en que funciona el sistema nervioso de los animales. Se trata de un sistema de interconexión de neuronas artificiales en una red que colabora para producir un estímulo de salida como se puede apreciar en la figura 2.1.

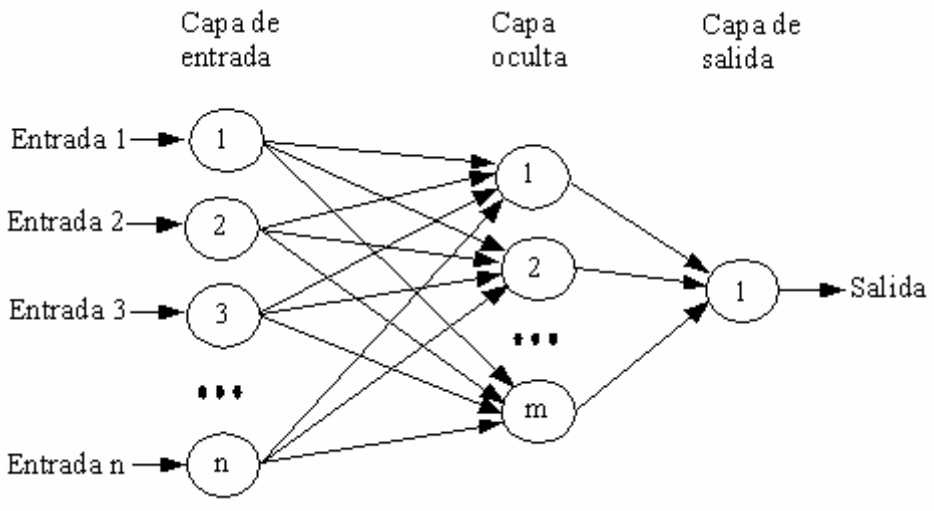


Figura 2.1. Red Neuronal Artificial Perceptrón multicapa con n neuronas de entrada, m neuronas en su capa oculta y una neurona de salida.

La neurona artificial es un elemento que posee un estado interno, llamado nivel de activación, y recibe señales que le permiten cambiar de estado. Las neuronas poseen una función que les permite cambiar de nivel de activación a partir de señales que reciben provenientes del exterior o de las neuronas a las cuales están conectadas; a dicha función se le denomina función de transición de estado o función de activación.

2.2.1 Funcionamiento

Una de las misiones en una Red Neuronal consiste en simular las propiedades observadas en los sistemas neuronales biológicos a través de modelos matemáticos recreados mediante mecanismos artificiales (como un circuito integrado, un ordenador o un conjunto de válvulas). El objetivo es conseguir que las máquinas den respuestas similares a las que es capaz de dar el cerebro, estas respuestas se caracterizan por su generalización y su robustez. **[17]**

Una Red Neuronal se compone de unidades llamadas neuronas. Cada neurona recibe una serie de entradas a través de interconexiones y emite una salida. Esta salida viene dada por tres funciones:

1. Una función de propagación (también conocida como función de excitación), que por lo general consiste en el sumatorio de cada entrada multiplicada por el peso de su interconexión (valor neto). Si el peso es positivo, la conexión se denomina excitatoria; si es negativo, se denomina inhibitoria.

2. Una función de activación, que modifica a la anterior. Puede no existir, siendo en este caso la salida la misma función de propagación.

3. Una función de transferencia, que se aplica al valor devuelto por la función de activación. Se utiliza para acotar la salida de la neurona y generalmente viene dada por la interpretación que queramos darle a dichas salidas. Algunas de las más utilizadas son la función sigmoide (para obtener valores en el intervalo [0,1]) y la tangente hiperbólica (para obtener valores en el intervalo [-1,1]).

2.3 Visión artificial

2.3.1 Historia

En el año 1826 el químico francés Niepce (1765-1833) llevó a cabo la primera fotografía, colocando una superficie fotosensible dentro de una cámara oscura para fijar la imagen.

Posteriormente, en 1838 el químico francés Daguerre (1787-1851) hizo el primer proceso fotográfico práctico. Daguerre utilizó una placa fotográfica que era revelada con vapor de mercurio y fijada con trisulfato de sodio.

Desde que se inventó la fotografía se ha intentado extraer características físicas de las imágenes. La fotogrametría dio sus primeros pasos desde imágenes capturadas en globos. La astronomía avanzó enormemente con el análisis de imágenes recibidas por los telescopios. El análisis de radiografías transformó la medicina. Se podrían citar muchos más ejemplos que durante décadas han

transformado la percepción de la ciencia con el procesamiento de las imágenes, algunas veces por separado y otras de forma multidisciplinar.

Sin embargo, el momento histórico que hace que estas técnicas confluyan y den un cuerpo de conocimiento propio, surge en la década de los 80. La revolución de la Electrónica, con las cámaras de vídeo CCD y los microprocesadores, junto con la evolución de las Ciencias de la Computación hace que sea factible la Visión Artificial. Ésta pretende capturar la información visual del entorno físico para extraer características relevantes, utilizando procedimientos automáticos. Según Marr, "Visión es un proceso que produce a partir de imágenes del mundo exterior una descripción útil para el observador y no tiene información irrelevante". **[11]**

Para algunos autores, como González y Woods, los primeros atisbos de este proceder se remontan a la década de los años 20 del siglo XX, cuando se transmitían imágenes transoceánicas, a través de cable submarino. Las fotografías periodísticas entre Europa y América tardaban una semana en llegar a través de los barcos. Al emplear las primeras técnicas de procesamiento de las imágenes se pasó sólo a tres horas, las imágenes se codificaban a cinco niveles de grises y se transmitían por teléfono. No obstante, éste podría ser el principio de las técnicas de procesamiento de las imágenes, pero no el de la Visión Artificial, tal cual se ha definido. El concepto de Visión Artificial es más amplio y recupera para sí, todos los conocimientos de análisis de las imágenes desempeñado por otras disciplinas desde los albores de la fotografía.

2.3.2 Definición

La visión por computador, también denominada Visión Artificial, puede definirse como el proceso de extracción de información del mundo físico a partir de imágenes,

utilizando para ello un computador. También podría definirse como un sistema autónomo que realiza alguna de las tareas del sistema de visión humano (o animal). La información o tarea que este sistema pueda llegar a extraer o realizar, puede ir desde la simple detección de objetos sencillos en la imagen hasta la interpretación tridimensional de complicadas escenas.

La visión por computador es una disciplina de creciente y continuo interés en el campo científico-técnico como consecuencia e importancia y número de las posibles aplicaciones, entre las que se encuentran: robótica, procesos de inspección automática, navegación autónoma de vehículos, análisis de imágenes, etc.

2.3.3 Funcionamiento

Inicialmente, la información visual consiste en energía luminosa procedente del entorno. Para poder utilizar esta información se hace preciso transformarla a un formato susceptible de ser procesado. En la visión humana esta tarea la realiza el ojo humano (ver figura 2.2) y en concreto la retina la cual contiene dos tipos de receptores: los conos y los bastones. Los primeros, situados sobre la fóvea; son los responsables de la percepción del color, mientras que los bastones sólo son sensibles a intensidades luminosas.

En la visión por computador esta tarea se realiza con una cámara fotográfica (o algún otro dispositivo similar), que convierte la energía luminosa en corriente eléctrica, que puede ser muestreada y digitalizada para su procesamiento en un computador.

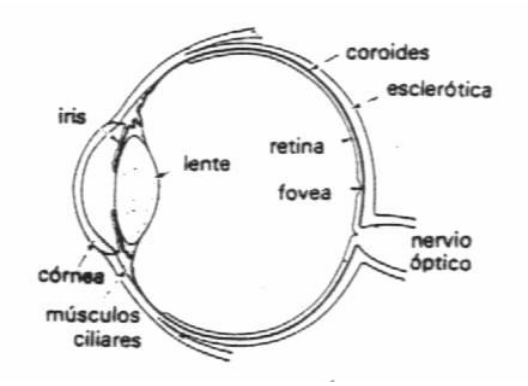


Figura 2.2. Ojo Humano

El muestreo se realiza, normalmente, a lo largo de una rejilla rectangular, produciendo así una matriz bidimensional de valores que se corresponden con la energía luminosa. Cada elemento de esa matriz se conoce como "Píxel". Si el dispositivo fotosensitivo fuera sensible a diferentes longitudes de ondas, entonces el valor de intensidad se corresponde con un determinado color. En cualquier caso, esta matriz de niveles de grises es la información que tiene que manejar el computador, y a partir de la cual debe extraer la información.

2.3.4 Terminología y campos relacionados

Existen varios términos que hacen referencia al campo de la Visión Artificial: visión por computador, visión de máquina, visión de robot, visión computacional, análisis de imágenes, interpretación de escena. Cada uno de estos términos, aunque abordando el mismo problema, posee connotaciones distintas y enfatizan aspectos distintos involucrados en el proceso de la Visión Artificial.

Por ejemplo el término “visión de máquina” es más utilizado en ingeniería y en aplicaciones de ésta, haciendo referencia a la construcción de máquinas capaces de percibir e interpretar el entorno.

El término de “visión computacional” surgió de la investigación interdisciplinaria de psicofísicos, técnicos en computación, neurólogos, etc. El objetivo de la visión computacional es expresar el proceso de la visión en términos de computación, entendiendo ésta, no solo como computación numérica sino en un sentido más amplio como sistema de procesamiento algorítmico abstracto.[13]

El término "visión de robot" hace referencia a aquellos métodos y algoritmos particularmente diseñados para dotar a un robot (normalmente un manipulador) de percepción visual de su entorno. Una característica de estos sistemas proviene, precisamente, de las restricciones que el entorno le impone. No se contemplan, por ejemplo, escenarios naturales, y si es habitual el control de la iluminación (Horn, 1988).

Existen varios campos muy estrechamente relacionados con la visión por computador:

- **El procesamiento de imágenes**, que involucra la transformación de una imagen para obtener otra de más calidad o mejor acondicionada para la posterior extracci6n de información.

- **Los gráficos por computado**r, donde se aborda el problema de plasmar en un formato bidimensional el mundo real. Este proceso es el inverso al que se realiza en 1a visión artificial.

- **El reconocimiento de patrones**, que aborda la clasificaci6n de objetos en clases representadas por prototipos o patrones.

- **La inteligencia artificial**, y más concretamente los problemas de interpretación, aprendizaje y razonamiento cognitivo.

Si se considera el espacio definido por éstas cuatro campos se puede decir que la visión artificial se sitúa en algún lugar de este espacio. Dependiendo del problema concreto, una componente puede ser más importante que otra, así por ejemplo, en un problema estéreo normalmente se involucra menos Inteligencia Artificial que en un problema de interpretación de una escena, en un problema de inspección automática no es relevante, normalmente, la relación geométrica entre el mundo 3D y la imagen, mientras que son de vital importancia el procesamiento de imagen y el reconocimiento de las regiones y formas que en esta aparecen.

2.3.5 Etapas

Las etapas a considerar en un proceso de visión artificial dependen del objetivo perseguido: reconocer, localizar, estimar forma, etc. En la figura 2.3 se muestran, en un sentido global, las distintas etapas que suelen considerarse.

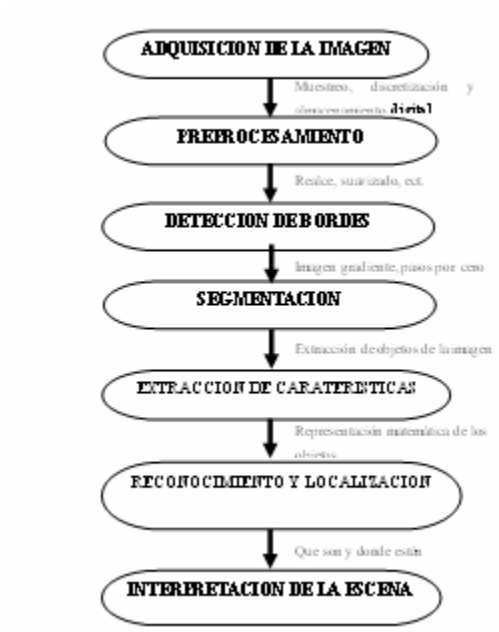


Figura 2.3. Etapas de un proceso de visión por computador.

Esta estructuración no significa, sin embargo, que cualquier proceso de visión por computador tenga necesariamente que pasar por cada una de ellas. Por ejemplo, en un sistema estéreo o de seguimiento visual, las etapas de segmentación, extracción de características y reconocimiento no suelen ser necesarias, mientras que si se requiere de la implantación de otras técnicas específicas. En cierta forma, se puede decir que ésta división se corresponde más con lo que podría denominarse un problema de reconocimiento e interpretación de la escena.

Aunque las etapas que aparecen en la figura 2.3, resulta conveniente en este momento definir el cometido de cada una de ellas.

- **Adquisición de la imagen**: tiene por objeto plasmar en una imagen digital el mundo real tridimensional.

- **Preprocesamiento**: incluye aquellas operaciones encaminadas a preparar la imagen para posteriores etapas, como son la eliminación de ruido y el realce.

- **Detección de bordes**: su importancia es vital en muchos de los procesos de visión ya que permite extraer de la imagen los bordes de los objetos.

- **Segmentación:** tiene por objeto determinar en la imagen regiones cuyos píxeles comparten algún tipo de atributo. Estas regiones, previsiblemente, van a corresponder a objetos de interés de la escena.

- **Extracción de características**: obtiene una representación matemática de los objetos previamente segmentados.

- **Reconocimiento**: se c1asifican los objetos como pertenecientes a aquella clase o prototipo cuyas características más se asemejen a la del objeto.

- **Localización:** se procede a localizar al objeto en el espacio 3D. Para ello es necesario recurrir a técnicas de triangulación (activa o pasiva) o a restringir el espacio de acuerdo con el conocimiento que se tenga de la escena. Un ejemplo de este último caso sería la localización de un objeto situado sobre una cinta transportadora de la que se conoce su posición espacial, extraída de una única imagen.

- **Interpretación:** con la información obtenida en las etapas anteriores se procede a interpretar la escena, considerando para ello la relación entre los objetos simples previamente reconocidos y localizados, así

como un cierto conocimiento sobre restricciones y reglas que rigen el mundo real. Ésta etapa está estrechamente ligada a la Inteligencia Artificial, estando aún en sus primeros pasos y sin resultados definitivos.

En un sentido más amplio, estas etapas suelen agruparse en dos niveles: visión de bajo nivel y análisis de la escena.

La visión de bajo nivel incluye las primeras etapas de procesamiento encaminadas a obtener características más o menos básicas de la imagen, como bordes, regiones, atributos de éstas como el color y la textura, movimiento, etc.

El segundo nivel de procesamiento, el análisis de la escena o visión de alto nivel, toman las características extraídas en el nivel anterior y construye una descripción de la escena a un nivel superior. Algunas de las componentes de esta tarea son el análisis de formas, y el reconocimiento y localización de objetos. Este nivel a menudo requiere de mecanismos de interpretación, irresolubles hoy en día.

2.3.6 Formación de la imagen

Para analizar el proceso de formación de la imagen o, lo que es lo mismo, el proceso mediante el cual los objetos del mundo tridimensional se proyectan en un plano imagen (Bidimensional) hay que abordar dos cuestiones fundamentales:

- ¿Dónde se proyecta en la imagen cada punto de la escena?
- ¿Con qué brillo aparecerá cada punto proyectado?

La primera de estas cuestiones tiene que ver con el problema geométrico de la formación de imágenes, mientras que la segunda está relacionada con el problema radiométrico.

2.3.7 Óptica

La función de la óptica de una cámara es la de captar los rayos luminosos para concentrarlos sobre el sensor de imagen, idealmente, la imagen obtenida debería ser una fiel reproducción de los objetos de la escena, aunque invertida y con diferente tamaño.

Exteriormente, la óptica u objetivo adopta la forma de un cilindro metálico cuya cara anterior es una lente; en su parte posterior (montura) posee una rosca o un sistema de bayoneta que permite su fijación a la cámara. En su interior se dispone de una agrupación de lentes de características diversas, junto con los dispositivos que posibilitan el desplazamiento de las mismas.

En la parte anterior de la óptica aparecen impresas una serie de anotaciones que definen algunas de las características de ésta. Una de ellas es la distancia focal, que se expresa en milímetros, pudiendo ser fija o variable (en el caso de que la óptica posea "zoom"). Aparece también una escala graduada de distancias que se extiende

desde fracciones de metro hasta el infinito y sirve para regular el enfoque. Finalmente, se inscribe un valor, denominado F-número, que hace referencia a la luminosidad máxima que deja pasar la óptica, es decir, indica la mayor apertura posible del diafragma.

Las ópticas comerciales están compuestas por agrupaciones de lentes positivas o convergentes y negativas o divergentes. Los lentes son dispositivos ópticos transparentes que, merced a fenómenos de refracción de la luz al pasar a través suya, desvían los haces de luz que reciben. Las lentes positivas hacen converger en un punto todos los rayos paralelos que las atraviesan. Las lentes negativas, por el contrario, se caracterizan porque los rayos de luz paralelos que las atraviesan se apartan de este paralelismo, divergen entre sí.

2.3.7.1 Modelo pin-hole

Es el modelo más simple de lente. En éste cada punto de un objeto del espacio se proyecta en un punto de un plano denominado plano imagen, donde se forma la imagen de la escena 3D esta se ilustra en la figura 2.4. Puesto que el agujero es de diámetro infinitesimal, de los rayos de luz que proceden de cada punto 3D sólo uno pasa por él, por lo que sobre cada elemento del sensor imagen incide un único rayo de luz. De acuerdo con este proceso, todos los puntos del espacio estarán siempre perfectamente enfocados.

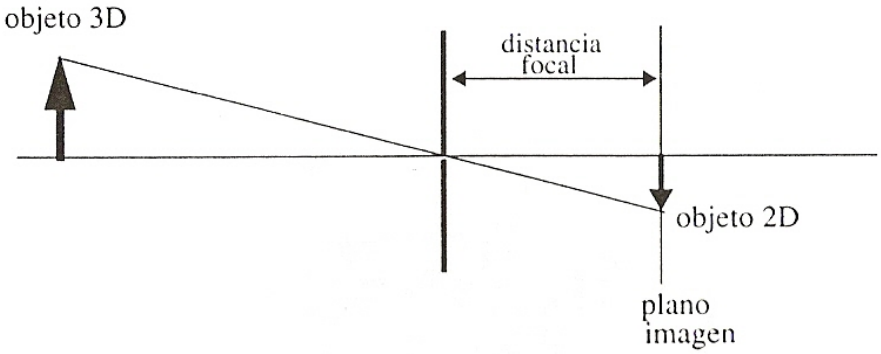


Figura 2.4.- Proyección de un punto de un objeto 3D en el plano imagen utilizando el modelo del agujero ("pin-hole").

Por su simplicidad, este modelo es muy utilizado en un gran número de aplicaciones y procesos de visión por computador. Sin embargo, desde un punto de vista radiométrico es incapaz de explicar la formación de imágenes, ya que, un único rayo procedente de un punto del espacio no posee energía suficiente para excitar el elemento del sensor. Si el agujero se hace mayor permitiría una cantidad de rayos suficientes para ello, pero la imagen estaría desenfocada.

Además, el inconveniente fundamental de este modelo es que no contempla la mayoría de las características y parámetros típicos de una óptica como el desenfoque de objetos, el control de la cantidad de luz incidente en el sensor, etc.

2.3.7.2 Modelo de lente delgada

Es un sistema más completo que el anterior que permite modelar la mayoría de los parámetros de las ópticas reales. Este sistema modela la óptica como una única lente que concentra en un punto los infinitos rayos luminosos procedentes de un determinado punto del espacio. De acuerdo con este modelo, todos los rayos paralelos al eje óptico de la lente convergen en un punto, que es el foco F como se observa en la figura 2.5.

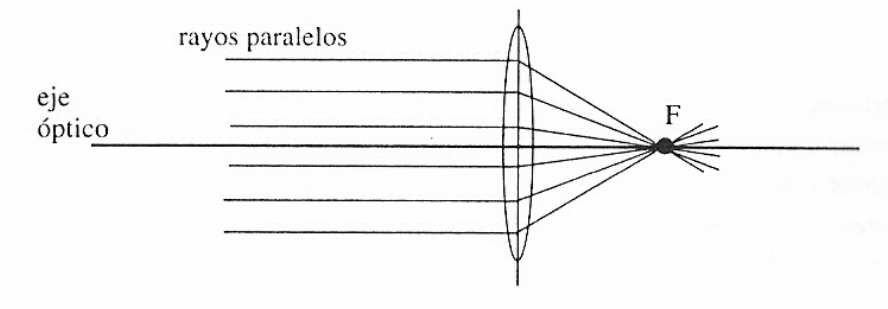


Figura 2.5 todos los rayos paralelos al eje óptico convergen en el foco F**.**

En la figura 2.6 se muestra el proceso de formación de la imagen con este modelo. La proyección de un punto dado A_0 se determina tomando como base a dos rayos principales: uno paralelo al eje óptico de la lente (r_1) y otro que pasa por el centro óptico (r_2). El punto de intersección de ambos determina el punto Ai donde convergerán los infinitos rayos que provenientes de A_0 pasan a través de la lente. Asimismo, el foco F de la lente viene dado por el punto donde el rayo r_1, refractado de r_1 corta al eje óptico.

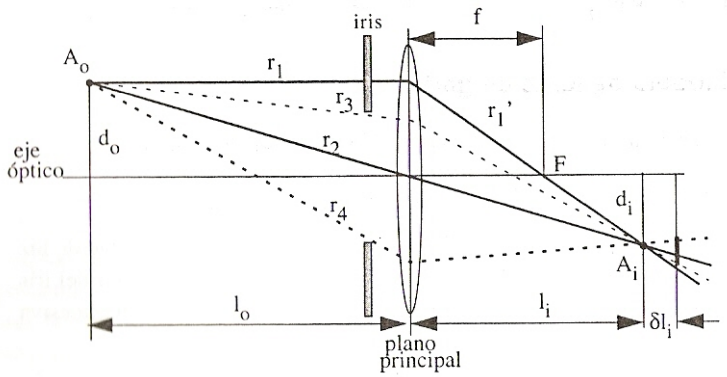


Figura 2.6 Proceso de formación de la imagen en el modelo de lente delgada.

Los parámetros que intervienen en este modelo son:

- Distancia focal ("zoom")
- Distancia de enfoque
- Profundidad de campo
- Apertura (iris o diafragma**)**

2.3.7.3 Distancia focal

La distancia focal es la distancia "f" del centro de la lente al foco, y determina el tamaño de la imagen formada en el sensor (efecto "zoom"). En concreto, una distancia focal pequeña reduce el tamaño de los objetos en la imagen, mientras que una grande lo aumenta. Puesto que el tamaño del sensor es fijo, la distancia focal controla también el ángulo de visión: a mayor distancia focal menor ángulo de visión.

Para una óptica dada, la distancia focal depende de la capacidad de hacer converger los rayos de luz, y es combinación de toda una serie de factores que intervienen en el fenómeno de la refracción de ésta. Las ópticas normalmente contienen hasta una docena de lentes asociadas en grupos. La variación de la distancia focal y, por tanto, del aumento, se produce por el movimiento relativo entre estos grupos de lentes.

En una óptica sin aumentos la distancia focal es fija (teóricamente). Sin embargo, si ésta posee "zoom" la distancia focal se puede modificar en un rango dado. Por ejemplo, el término (expresado en la óptica) "x6" o también "1:6" indica que la distancia focal varia en un factor de 6. Los valores mínimos y máximos de f, en este caso, suelen ser 12'5 mm y 75 mm, respectivamente.

La distancia focal del ojo humano, aproximadamente 20 mm, establece la frontera entre el denominado teleobjetivo, con distancia focal mayor que 25 mm, y el gran-angular cuya distancia focal es menor de 15 mm.

Es la distancia, medida desde el plano de la lente, a la que se encuentra el plano del espacio que permanece enfocado en el plano imagen ("Io" en la figura 2.6).

Con referencia a la figura 2.6, si el plano imagen se desplaza ligeramente un $\sigma 1_i$, el punto del espacio A_o ya no se proyecta en un único punto A_i, sino en un área de tamaño σA_i denominada círculo de confusión. Esto origina el desenfoque del

punto A_o' El mismo efecto se produce si es el propio punto A_o' manteniendo fijo el plano imagen, el que se aleja o acerca del plano principal.

La distancia de enfoque Io y la distancia focal están relacionados mediante la ecuación (2.1):

$$1/f = 1/i_0 + 1/i_1 \quad (2.1)$$

De acuerdo con esta fórmula, asumiendo que f es fijo, la distancia de enfoque se modifica variando la distancia entre la lente y el plano imagen.

El desenfoque de una imagen da lugar a una imagen borrosa (poco perfilada) y con poco contraste. Típicamente, para ópticas comerciales, la distancia de enfoque puede ajustarse desde un metro hasta el infinito.

2.3.8 Operaciones morfológicas

La morfología matemática es una herramienta muy utilizada en el procesamiento de imágenes.

Las operaciones morfológicas pueden simplificar los datos de una imagen, preservar las características esenciales y eliminar aspectos irrelevantes. Teniendo en cuenta que la identificación y descomposición de objetos, la extracción de rasgos, la localización de defectos e incluso los defectos en líneas de ensamblaje están

sumamente relacionados con las formas, es obvio el papel de la morfología matemática.

La morfología matemática se puede usar, entre otros, con los siguientes objetivos:

- **Preprocesamiento de imágenes**: supresión de ruido, simplificación de formas.
- **Destacar la estructura de objetos:** extraer el esqueleto, marcado de objetos, envolvente convexa, ampliación, reducción.
- **Descripción cualitativa de objetos**: área, perímetro, diámetro, etc.

2.3.8.1 Representación de imágenes binarias

Definiremos una imagen binaria como una función de dos variables discretas a[m,n] que puede tomar dos valores, '0' o '1', dependiendo del nivel de gris de la imagen (una imagen binaria tiene dos niveles: blanco y negro), un ejemplo de esto se puede observar en la figura 2.7.

Se puede proponer una definición alternativa si consideramos que una imagen consiste en un conjunto de coordenadas discretas (también pueden ser reales pero no es el objetivo de este estudio). En este sentido, el conjunto corresponde a todos aquellos puntos o píxeles que pertenecen a la imagen. Por lo tanto, se puede decir que

en morfología matemática los conjuntos representan objetos en una imagen. Por ejemplo, el conjunto de todos los píxeles negros en una imagen binaria constituye una descripción completa de la misma.

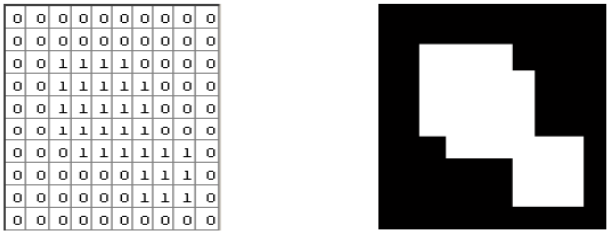


Figura 2.7: Representación de una imagen binaria

En las imágenes binarias, los conjuntos en cuestión pertenecen al espacio **Z2**, donde cada elemento del conjunto es una 2-upla (vector 2-D) cuyas coordenadas son las coordenadas **[m, n]** de un píxel blanco (o negro, según la convención) de la imagen. En la figura 2.8 se pueden ver dos conjuntos, **A** y **B**. Observemos que se ha colocado un sistema de coordenadas. El conjunto (u objeto) **A** consiste en los puntos {[2,3]; [2,4]; [2,5]; [1,3]; [1,4]; [1,5]; [0,5]} mientras que el B contiene los puntos {[0,0]; [0,1]; [1,0]}.

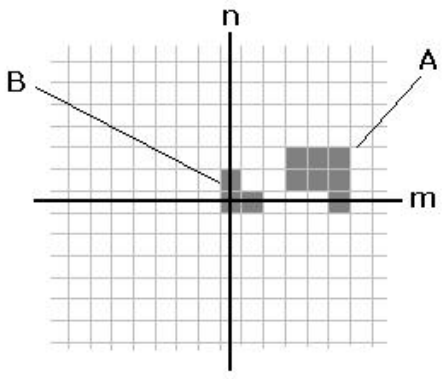


Figura 2.8: Imagen binaria con dos conjuntos

En este punto se debe acotar que en la mayoría de los lenguajes de programación los arreglos de elementos que serán los encargados de contener la imagen no admiten índices negativos y en general menores a 1. Por lo tanto, será necesario realizar una pequeña modificación al sistema de coordenadas que consiste en un simple desplazamiento para poder operar sobre una imagen (contenida en un arreglo).

2.3.8.2 Dilatación y Erosión

Estas operaciones son fundamentales en el procesamiento morfológico. De hecho, la mayoría de los algoritmos morfológicos están basados en estas dos operaciones.

2.3.8.2.1 Elemento Estructural

Si bien los conjuntos A y B, de la figura 2.8, pueden ser considerados como una imagen (u objeto), generalmente se considera que A es la imagen y B es el elemento estructural. El elemento estructural es en morfología matemática lo que la máscara (o núcleo) de convolución es en los filtros lineales. Los elementos estructurales más comunes son los conjuntos que están 4-conectados, N4, y 8-conectados, N8, ilustrados en la figura 2.9

a b

Figura 2.9: Elementos estructurales estándar. a) N4. b) N8.

2.3.8.2.2 Dilatación

Sean A y B conjuntos en Z2. La dilatación de A por B, expresada por A $\oplus$ B, se define como:

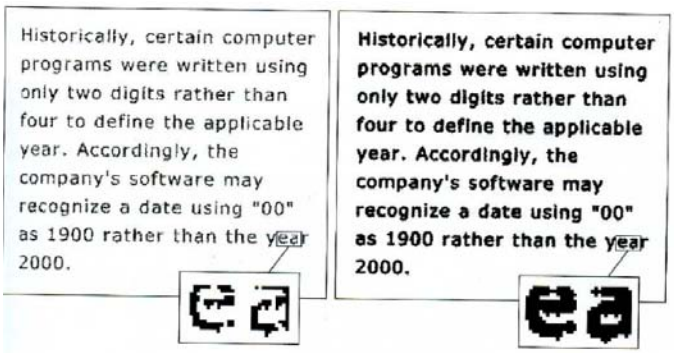
$$A \oplus B = \{z \mid (\hat{B})_z \cap A \neq \varnothing\} \quad (2.2)$$

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

$$A \oplus B = \{z \mid ((\hat{B})_z \cap A) \subseteq A\} \quad (2.3)$$

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

Una de las aplicaciones más simples de la dilatación es la unión de píxeles relacionados. La figura 2.10 (a) muestra un texto cuyos caracteres han perdido píxeles debido a que previamente ha sido filtrado. Se sabe que la longitud máxima de las rupturas es de dos píxeles. Si se dilata la imagen con un elemento estructurante de conectividad 4 (N4) como el que se muestra en la figura 2.9 (a) se pueden unir los caracteres partidos y obtener como resultado la figura 2.10 b).



a b

Figura 2.10: a) Texto de poca resolución con caracteres partidos. b) Dilatación de la imagen de (a) con el elemento estructurante N4.

2.3.8.2.3 Erosión

Sean A y B conjuntos en Z2. La erosión de A por B, se define Como

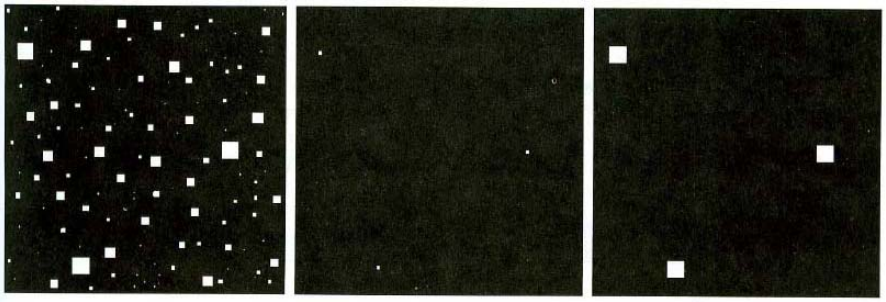
$$A\Theta B = \{z \mid (B)_z \subseteq A\} \quad (2.4)$$

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

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

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

La figura 2.11 a) muestra una imagen compuesta por cuadrados cuyos lados tienen 1, 3, 5, 9, y 15 píxeles. Supongamos que queremos eliminar todos los cuadrados excepto los más grandes. Esto los podemos hacer erosionando la imagen con un elemento estructural cuyo tamaño sea un poco menor que el de los cuadrados que deseamos conservar. Por ejemplo, si elegimos un elemento estructural de 13x13 podemos obtener la imagen de la figura 2.11 b). Como se observa, sólo se han mantenido las porciones de los cuadrados más grandes. Luego, podemos restituir el tamaño de estos 3 cuadrados a su tamaño original de 15x15 dilatando la imagen erosionada (figura 2.11 b) con el mismo elemento estructural utilizado para la erosión. El resultado puede verse en la figura 2.11 c).



a b c

Figura 2.11: a) Imagen con cuadrados de 1, 2,3, 5, 7, 9 y 15 píxeles de lado. b) Erosión de (a) con un elemento estructural de 15x15. c) Dilatación de (b) con el mismo elemento estructural.

2.3.8.2.4 Algoritmos Simples de Dilatación y Erosión

Las expresiones que se dieron anteriormente para la dilatación y la erosión de imágenes binarias son las definiciones formales y pueden resultar difíciles de comprender a la hora de construir algoritmos para tales fines.

A continuación se enuncia la manera de armar algoritmos de dilatación y erosión teniendo en cuenta la idea básica e intuitiva de estas dos operaciones:

- **Dilatación**: tomar cada píxel del objeto (con valor "1") y colocarle en "1" todos aquellos píxeles pertenecientes al fondo (background) que tienen una conectividad C (C=4, C=8, etc) con el píxel del objeto. En pocas palabras, poner a "1" los píxeles del fondo vecinos a los píxeles del objeto.

- **Erosión**: tomar cada píxel del objeto que tiene una conectividad C con los píxeles del fondo y colocarle en “0”. En otras palabras, poner a “0” los píxeles del objeto vecinos a los píxeles del fondo.

2.3.8.2.5 Apertura y Clausura (Opening and Closing)

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

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

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

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

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

$$A \bullet B = (A \oplus B) \Theta B$$

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

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

2.3.9 Procesamiento de la imagen

En un proceso de Visión Artificial, éste tiene como finalidad producir otra imagen de mayor calidad que simplifique y facilite posteriores etapas. El objetivo, por tanto, no es extraer información de la imagen sino actuar convenientemente sobre los niveles de grises de ella para compensar defectos de iluminación (poco contraste, no-uniformidad, etc.) así como eliminar ruido y efectos espurios.

En un ámbito más global, el procesamiento de imágenes no se limita a estas operaciones sino que incluye además la compresión y descompresión de imágenes, restauración, etc. El campo de aplicaciones es muy extenso incluyendo aplicaciones médicas, comunicación de vídeo, meteorología, codificación, etc.

Históricamente, el procesamiento de imágenes se ha desarrollado desde dos enfoques diferentes. Uno ha sido la extensión natural del procesamiento digital de señales, desde donde se ha abordado como un problema de teoría de filtrado de señales, involucrando de manera importante operaciones en el dominio de frecuencias. El segundo, más heurístico, considera la imagen digital corno una matriz

de muestras discretas, realizando operaciones aritméticas sobre los elementos de esta matriz.

2.3.9.1 Histograma

El histograma de una imagen es una representación gráfica de la frecuencia con la que los niveles de grises aparecen en dicha imagen. El eje de abscisas indica los distintos niveles (discretos) de grises y en ordenadas se representa la frecuencia o, también a veces, el número de píxeles que poseen el nivel de gris (ver figura 2.12).

Éste se construye, simplemente, rastreando toda la imagen y contabilizando el número de píxeles que poseen cada nivel de gris.

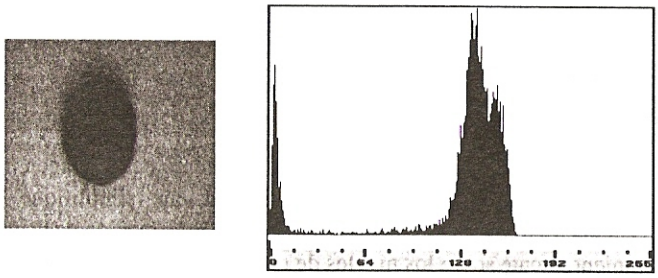


Figura 2.12 una imagen con su correspondiente histograma.

El histograma proporciona información estadística sobre cómo están distribuidos los distintos niveles de grises de la imagen. Esta información puede ser bastante útil para tareas tales como conocer si la digitalización se ha efectuado

correctamente, decidir el valor de umbralización de una imagen, tener una estimación del brillo medio y contraste, etc. En la figura 2.13 se muestran algunos histogramas típicos donde es posible extraer cierto tipo de información.

Cuando el histograma se expresa en términos de la frecuencia de aparición de los distintos niveles de grises es posible establecer una cierta analogía entre éste y una función de densidad de probabilidad (F.D.P.). Lógicamente, esta analogía será más sólida cuanto mayor sea el número de niveles de intensidad de la imagen.

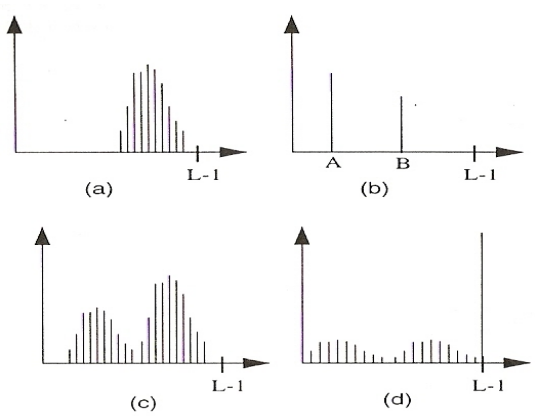


Figura 2.13 Algunos histogramas típicos. (a) Imagen con mucho brillo y poco contraste (mala digitalización). (b) Imagen binaria con dos niveles Ay B. (c) Histograma bimodal típico de objeto sobre fondo. (d) Saturación del blanco (mala digitalización).

2.3.9.2 Tablas de consulta

Una tabla de consulta, normalmente conocida como LUT (look-up table), es una función discreta definida sobre los números naturales comprendidos en el intervalo

[0,L-1], siendo L el número de niveles de grises empleados en la digitalización. Esta función define una transformación píxel a píxel entre los niveles de grises de la imagen a procesar y de la imagen resultante. Normalmente, el "0" corresponde al negro o nivel más oscuro, y el L-1 al blanco o nivel de gris más claro.

En la figura 2.14a se ilustra el principio de funcionamiento de una LUT. En la práctica, puesto que los niveles son discretos, la transformada se almacena en una tabla (vector) cuyos elementos son los nuevos niveles de grises correspondientes al intervalo [Q,L-I]. La transformación, por tanto, consiste en asignar como nuevo nivel de gris el correspondiente al elemento indexado con el nivel de entrada (ver figura 2.14b).

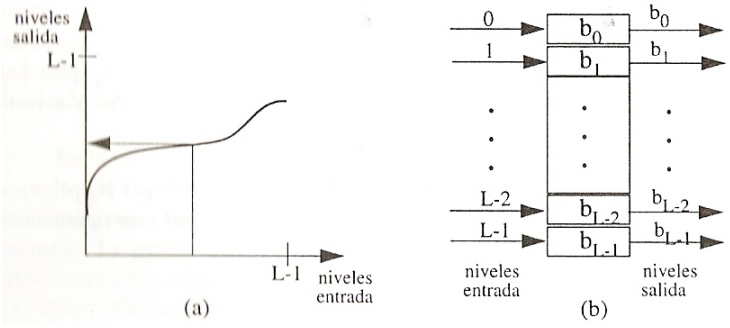


Figura 2.14 Funcionamiento de una tabla de consulta (LUT). (a) Transformación contínua. (b) transformación discreta.

Este tipo de transformación, aunque muy simple, tiene una gran utilidad en el procesamiento de imágenes, ya que su implementación es fácil y se puede aplicar en tiempo real (a la frecuencia de digitalización), en la figura 2.15 se muestran algunos ejemplos.

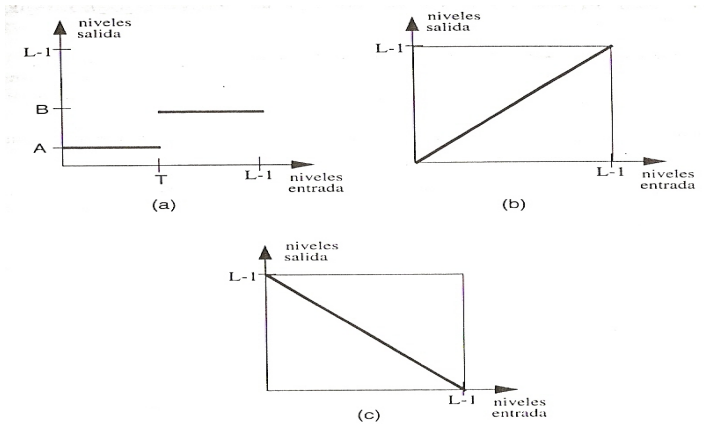


Figura 2.15 (a) Operación de umbralización. (b) No se modifica la imagen (operación identidad). (c) inversión de la paleta de grises.

2.3.9.3 Relaciones básicas entre pixeles

- **Medidas de distancia**

Se dice que D es una función distancia si verifica:

$$D(p_1,p_2) \geq 0 \; [D(p_1,p_2) = 0 \; si \; p_1{=}p_2] \quad (2.5)$$

$$D(p_1,p_2) = \; D(p_2,p_1) \quad (2.6)$$

$$D(p_1,p_3) \leq \; D(p_1,p_3) + \; D(p_2,p_3) \quad (2.7)$$

Donde p_1, p_2 y p_3 son tres píxeles de coordenadas (x_1,y_1), (x_2,y_2) y (x_3,y_3) respectivamente.

$$D_e(p_1,p_2)= [(x_1-x_2)^2+(y_1-y_2)]^{1/2} \quad (2.8)$$

El lugar geométrico de los píxeles cuya distancia euclídea a "p_1" es menor o igual que un valor dado r es un círculo de radio r centrado sobre "p_1".

Además de la función distancia euclídea suelen utilizarse también otras métricas, más adecuadas al formato discreto de las imágenes digitales, entre las que se encuentran la distancia DR, denominada rectangular o Manhattan, definida como:

$$D_R(p_1,p_2) = |x_1-x_2| + |y_1-y_2| \quad (2.9)$$

y la distancia D_T, denominada Tchebichev, y definida como:

$$D_T(p_1,p_2) = \max(|x_1-x_2|,|y_1-y_2|) \quad (2.10)$$

El lugar geométrico de los pixeles a distancias D_R y D_T de "p_1" menos que un valor dado r son un rombo y un cuadrado, respectivamente, centrados sobre el píxel "p_1" ver figura 2.16

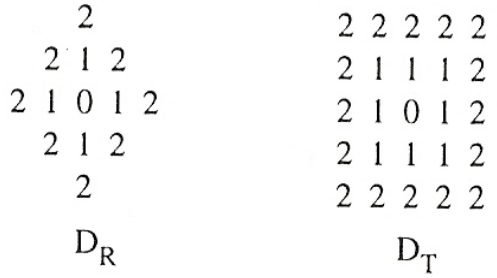


Figura 2.16 El lugar geométrico de los pixeles

- **Vecinos de un Píxel**

Se define el conjunto de los 4-vecinos de un píxel dado “p”, denotado por $N_4(p)$, como el conjunto formado por los pixeles de arriba, abajo, a la derecha y a la izquierda de “p”, es decir, por aquellos pixeles “q” que verifican $D_R(p,q)=1$, figura 2.16.

Se define el conjunto de los vecinos diagonales de "p", denotado por $N_8(p)$, como el conjunto formado por los píxeles situados en las diagonales de "p" y a distancia D_T igual a uno (ver figura 2.17)

Finalmente, se define el conjunto de 8-vecinos de "p", denotado por $N_8(p)$, como el conjunto formado aquellos píxeles "q" que verifican DT(p,q)=l (ver figura 2.17).

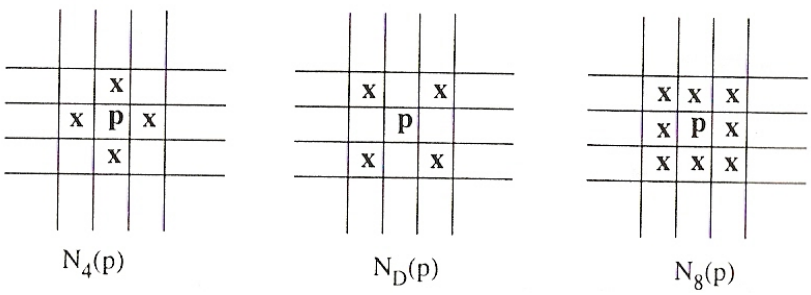


Figura 2.17. Diferentes conjuntos de vecinos del píxel “p”.

2.3.9.4 Operación de convolución

La convolución es una operación que modifica el nivel de gris de los píxeles de la imagen teniendo en cuenta los píxeles de su entorno de vecindad. Este tipo de

operación, a menudo denominada filtrado de la imagen, es fundamental como herramienta para el procesado de la imagen, y en particular, para la eliminación de ruido y detección de bordes.

2.3.9.5 Realce

El principal objetivo de las técnicas de realce es aumentar el contraste mediante la redistribución de los niveles de grises de la imagen. En visión por computador, la finalidad de este aumento de contraste no es mejorar la calidad de la imagen para una mejor visualización, sino obtener una imagen resultante que sea más adecuada que la original para una determinada aplicación.

Una forma simple pero eficaz de aumentar el contraste, bien de una imagen completa o bien de determinadas regiones de interés dentro de ella, consiste en la modificación de la escala de grises de dicha imagen mediante una tabla de consulta o LUT.

- **Transformación de escala de grises**

La idea que subyace en este tipo de procesamiento no es más que la de aprovechar todos los niveles de la escala de grises.

Considérese, por ejemplo, la figura 2.18 donde se muestra un histograma de intensidades. Obsérvese cómo existe una amplia banda de niveles de grises que no

son utilizados en la imagen. Aprovechando estos niveles, la transformación de la escala de grises permite dilatar el rango de intensidades con el consiguiente aumento de contraste de la imagen.

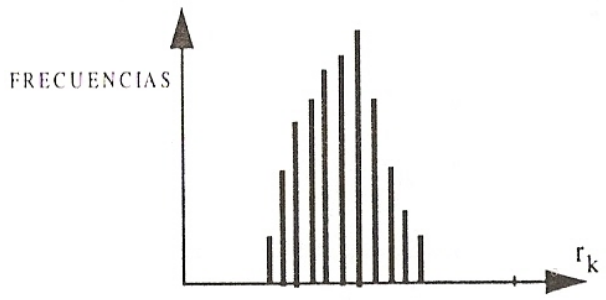


Figura 2.18 Histograma de una imagen en la cual no todos los niveles de grises son utilizados.

A diferencia de las técnicas anteriormente descritas en las cuales la función de transformación viene dada explícitamente por el tipo de histograma deseado, en este caso la información proporcionada por el histograma de la imagen a procesar no interviene directamente en la obtención de la función de transformación, aunque sí es tenida en cuenta.

La modificación de la escala de grises de una imagen f con niveles de intensidades discretos r_k viene dada por una transformación:

$$S_k = T(r_k)\ k = 0,1, \ldots, L-1 \quad (2.11)$$

Donde s_k son los niveles de grises (también discretos) de la imagen resultante, y L es el número de estos niveles.

Se supone que ambas escalas disponen del mismo rango de niveles, es decir, si r_k toma valores entre 0 y 255, también los S_k se encuentran comprendidos en dicho rango. Esta suposición es usual para la gran mayoría de las aplicaciones en procesamiento de imágenes:

Considérese el caso de aumento de contraste de una imagen con histograma como el mostrado en la figura 2.18, en la cual sus intensidades no ocupan por completo el rango disponible. Supóngase que, para una imagen dada f se tiene:

$$a \leq rk = f(i,j) \leq b \; \forall \; pixel \; (i,j) \qquad (2.12)$$

Donde [a, b] es un intervalo del rango de intensidades $[r_0, r_{L-1}]$.

La transformación lineal:

$$s_k = \frac{r_{L-1} - r_0}{b - a} \cdot (r_k - a) + r_0 \qquad a \leq r_k \leq b \qquad (2.13)$$

"estira" y traslada la escala de grises para ocupar el rango completo $[r_o' \; r_{L-1},]$ (ver figura 2.19).

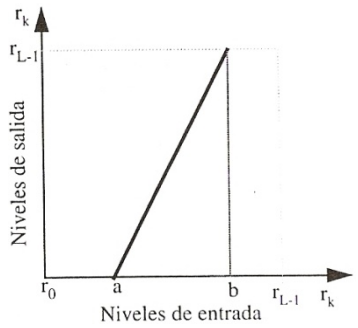


Figura 2.19 Dilatación del intervalo [a,b] al rango completo de intensidades $[r_0, r_{L-1}]$.

Esta transformación también puede utilizarse cuando la mayoría de los niveles de grises de la imagen dada quedan dentro del subrango [a,b]. En este caso puede utilizarse la función:

$$s_k = \begin{cases} \dfrac{r_{L-1} - r_0}{b - a} \cdot (r_k - a) + r_0 & a \le r_k \le b \\ r_0 & r_k < a \\ r_{L-1} & r_k > b \end{cases}$$

Figura 2.20. Función de transformación

Esta transformación dilata el intervalo [a, b] de la escala original a la vez que comprime los intervalos $[r_0, a]$ y $[b, r_{L-1}]$. Esta compresión puede tolerarse si son pocos los puntos en estos dos últimos intervalos, esto es, si es poca la información perdida.

De forma general, se pueden dilatar determinadas regiones a costa de la compresión de otras, siempre y cuando, el objetivo prioritario sea el de conseguir el aumento de contraste en bandas concretas, y no importe demasiado la pérdida de información en las regiones comprimidas.

2.3.10 Segmentación de la imagen

La segmentación es el proceso que divide una imagen en regiones u objetos cuyos píxeles poseen atributos similares (por ejemplo, niveles de grises, textura, etc.). Cada región segmentada suele tener un significado físico dentro de la imagen.

La segmentación es uno de los procesos más importantes de un sistema automatizado de visión ya que permite extraer los objetos de la imagen para su posterior descripción y reconocimiento. En la figura 2.21 se sitúa la etapa de segmentación dentro de lo que sería un proceso típico de reconocimiento de formas. Los flujos de realimentación que se muestran hacen referencia a la necesidad de replantear algunas de las etapas previas, en aquellos casos en los que la interpretación de la escena es "inconsistente" o contradice el conocimiento que a priori se tiene sobre ella. Un ejemplo de tal tipo de interpretación podría ser la de un bolígrafo suspendido en el aire. Tal escena es, obviamente, "inconsistente", y por tanto obliga a pensar que se ha errado en algunas de las etapas anteriores.

Las distintas técnicas de segmentación pueden encuadrarse en tres grupos fundamentales: técnicas basadas en la detección de la frontera, técnicas de umbralización y técnicas basadas en el agrupamiento de píxeles. Las dos últimas

enfocan la segmentación como un problema de clasificación de píxeles o grupos de píxeles, donde:

- píxeles de una misma región deben ser similares;
- píxeles de regiones distintas deben ser no-similares;
- las regiones resultantes deben tener cierto significado para el procesamiento posterior.

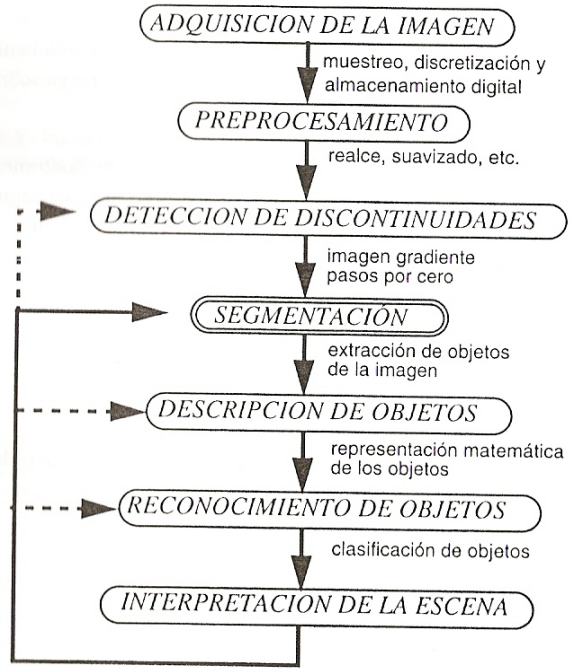


Figura 2.21 Etapa de la segmentación dentro del proceso de Visión Artificial

Esta similitud entre píxeles puede residir en el nivel de gris, textura, proximidad, posición en la imagen, etc.

Por otro lado, la segmentación basada en la frontera persigue el aislamiento de los objetos del resto de la imagen previa identificación de los píxeles que configuran la frontera de éstos. De esta manera queda definida la forma del objeto y, por consiguiente, los píxeles que lo integran.

2.3.10.1 Técnicas basadas en la frontera

La segmentación de una imagen puede llevarse a cabo mediante la detección de los límites de cada región, es decir, localizando los lugares donde se produce un cambio significativo de los niveles de intensidad de los píxeles (detección de bordes).

Esta operación puede efectuarse utilizando operadores basados en la primera y/o segunda derivada. En el primer caso, como consecuencia del ruido, iluminación no uniforme y otros, los operadores gradiente rara vez definen por completo la frontera de los objetos, necesitándose, en tal caso, algoritmos que a continuación se encarguen de realizar la unión de los píxeles detectados mediante el seguimiento del contorno.

En el segundo caso, con la obtención de los cruces por cero tras la aplicación del operador LoG, se obtienen contornos cerrados que definen por completo la frontera de los objetos y, por consiguiente, posibilitan la segmentación de objetos de la imagen. En la figura 6.2 se muestra un ejemplo de segmentación de la cabeza sobre un fondo claro no-uniforme (Rodríguez, 1995).

2.3.10.2 Segmentación mediante umbralización

Considérese el histograma de la figura 2.22a correspondiente a una imagen, f(x,y), compuesta de un objeto claro sobre fondo oscuro (o al contrario). Una forma de extraer el objeto de la imagen es seleccionar un umbral de intensidad T por encima del cual se encuentran todos los píxeles pertenecientes al objeto. De este modo, cualquier punto (x,y) para el cual f(x,y)>T es un punto del objeto, mientras que si f(x,y)<= T será un punto del fondo. En la práctica pueden ser necesarios más de un umbral para particionar la imagen (figura 2.22b).

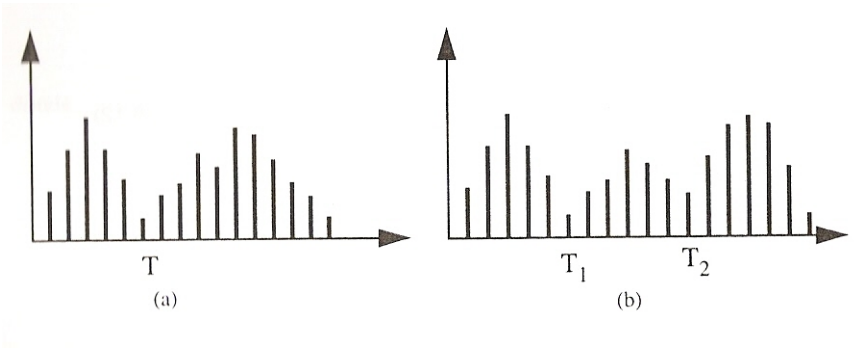


Figura 2.22 (a) umbral 1 (b) umbral 2

En un sentido más general, una operación de umbralización puede verse como una comparación con una función de la forma:

$$T = T [x, y, p(x,y), f(x,y)] \quad (2.14)$$

Donde f(x,y) es el nivel de gris del punto (x,y), y p(x,y) es alguna propiedad local de ese punto (por ejemplo, el nivel de gris medio en un entorno centrado en (x,y)).

Así, la imagen umbralizada g(x,y) se obtiene como:

$$g(x, y) = \begin{cases} 1 & \text{si } f(x,y) > T \\ 0 & \text{si } f(x,y) \leq T \end{cases} \quad (2.15)$$

Un píxel de g(x,y) cuyo valor sea "1" corresponderá al objeto, mientras que los de valor "0" corresponderán al fondo.

Cuando T depende solamente de f(x,y), el umbral se denomina global. Si T depende de f(x,y) y p(x,y), el umbral se llama local. Si además T depende de las coordenadas x e y se llama umbral dinámico.

2.3.10.3 Umbralización global

Los umbrales globales se utilizan cuando existe una clara definición entre los objetos y el fondo, y cuando la iluminación es relativamente uniforme. Las técnicas que usan retroiluminación o luz estructurada dan como resultado imágenes que pueden ser segmentadas mediante umbrales globales.

Supóngase que una imagen dada f(x,y) tiene el histograma mostrado en la figura 2.22a, donde una gran cantidad de píxeles de f(x,y) son oscuros, estando el

resto de píxeles distribuidos en la restante banda de niveles de grises. Este tipo de histogramas es característico de objetos claros sobre fondo oscuro (o al contrario).

Para detectar los límites de los objetos podemos dividir la escala de grises en dos bandas, B1 y B2, separadas por un umbral T. El objetivo es seleccionar el umbral T, a fin de que B1 contenga, de la forma más fiable posible, el conjunto de píxeles que forman el fondo y, B2 el conjunto de píxeles que forman el objeto.

A fin de detectar la frontera del objeto se realizan dos pasadas sobre f(x,y). El procedimiento es el siguiente:

- **Paso l**. Para cada fila de f(x,y) crear una fila en una imagen g1 (x,y) usando la relación siguiente:

$$g1(x,y) = \begin{cases} L_E & \text{si } f(x, y) \text{ y } f(x-1, y) \text{ están en diferentes bandas} \\ L_B & \text{en otro caso} \end{cases} \qquad (2.16)$$

Donde L_E y L_B son niveles especificados para el borde y el fondo, respectivamente.

- **Paso 2**. Para cada columna de f(x,y), crear una columna en una imagen intermedia g2(x,y) usando la siguiente relación:

$$g^2(x,y) = \begin{cases} L_E \text{ si } f(x, y) \text{ y } f(x-1, y) \text{ están en diferentes bandas} \\ L_B \text{ en otro caso} \end{cases} \quad (2.17)$$

La imagen deseada donde se distinguen claramente los puntos del borde y los del fondo, se obtiene mediante la siguiente relación:

$$g(x,y) = \begin{cases} L_E \text{ si } gl(x, y) \text{ o } g^2(x, y) \text{ es igual a } L_E \\ L_B \text{ en otro caso} \end{cases} \quad (2.18)$$

- **Selección del umbrál óptimo**

La selección de un umbral óptimo es crucial para poder realizar una buena segmentación.

En algunos casos es posible considerar el histograma como formado por la suma de funciones de densidad de probabilidad (figura 2.23). Supóngase que tenemos un histograma bimodal (dos picos y un valle). En este caso, la función continua que aproxima el histograma viene dada por:

$$p(z) = P_1(z)*p_1(z) + P_2(z)*p_2(z) \quad (2.19)$$

Donde z es una variable aleatoria que representa la intensidad, $P_1(z)$ y $P_2(z)$ son las funciones de densidad de probabilidad y P_1 Y P_2 son las probabilidades a priori de ocurrencia de los dos tipos de ni veles de intensidad de la imagen. P_1 Y P_2 verificarán, por tanto, la ecuación:

$$P_1 + P_2 = 1 \quad (2.20)$$

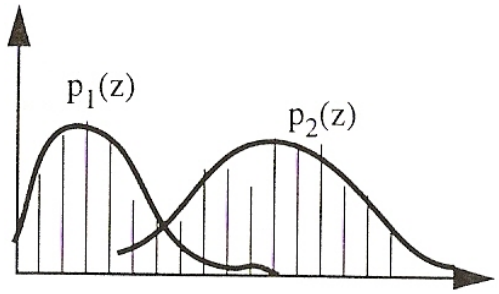


Figura 2.23 Histograma de intensidad de una imagen con su aproximación como suma de dos funciones de densidad de probabilidad.

Así se pueden considerar dos funciones de decisión:

$$d_1(z) = P_1(z)*p_1(z) \quad (2.21)$$

que caracteriza los niveles de intensidad de los pixeles del objeto, y

$$d_2(z) = P_2(z)*p_2(z) \quad (2.22)$$

que caracteriza los niveles de intensidad de los pixeles del fondo.

De acuerdo con la teoría de decisiones, el error medio de clasificar erróneamente un píxel de intensidad z se minimiza cuando éste se asigna a la clase cuya función de decisión es mayor, es decir: "un píxel con intensidad z se clasifica como píxel de objeto si $d_1(z) > d_2(z)$ o como píxel de fondo si $d_2(z) > d_1(z)$".

El umbral óptimo viene entonces determinado por el valor de z para el cual $d_1(z) = d_2(z)$, es decir, haciendo z=T en las ecuaciones anteriores se tiene:

$$P_1 * p_1(T) = P_1 * p_2(T) \quad (2.23)$$

Por tanto, si se conoce la forma de las funciones $p_1(z)$ y p2(z), se puede usar la ecuación anterior para calcular el umbral óptimo que separe al objeto del fondo.

2.3.10.4 Segmentación basada en regiones

Sea R una región que representa a la imagen completa. La segmentación de la imagen se puede ver como un proceso que divide a R en n subregiones, $R_1, R_2, \ldots Rn$, tal que:

$$\begin{aligned}
&\text{(a)} \bigcup_{i=1} R_i = R \\
&\text{(b) } R_i \text{ con i = 1, 2,..., n es una región conectada} \\
&\text{(c) } R_i \cap R_j = \varnothing \quad \forall i, j \quad i \neq j \\
&\text{(d) } P(R_i) = \text{TRUE para i = 1, 2,..., n} \\
&\text{(e) } P(R_i \cup R_j) = \text{FALSE para } i \neq j
\end{aligned} \quad (2.24)$$

Donde P (R_i) es predicado lógico definido por los puntos del conjunto R_i, basado en alguna medida de similitud, y Ø es el conjunto vacío.

La condición (a) indica que la unión de todas las regiones obtenidas tras la segmentación debe ser la imagen completa. La condición (b) requiere que los puntos de una región estén conectados. La condición (c) indica que las regiones han de ser disjuntas. La condición (d) indica que los píxeles de una región segmentada deben satisfacer unas determinadas propiedades (por ejemplo: $P(R_i)$ = TRUE si todos los pixeles de R_i tienen la misma intensidad). Finalmente, la condición (e) indica que las regiones R_i y R_j son distintas según el criterio del predicado P.

2.3.10.4.1 Crecimiento de regiones mediante adición de pixeles

El crecimiento de regiones es un procedimiento que agrupa píxeles o subregiones en regiones más grandes. El método más simple es la adición de píxeles, donde se parte de un conjunto de puntos semillas a los que se les van añadiendo pixeles vecinos que poseen propiedades similares (por ejemplo el nivel de gris, textura, color, etc). Si se usan *n* semillas, al final se podrá obtener una segmentación con un máximo de *n* regiones, además del fondo.

Ejemplo 1:

Como ilustración a esta técnica considérese la figura 2.24a, donde los números dentro de cada celda representan valores de intensidad. Supóngase que se

eligen como semillas las celdas (3,2) y (3,4). Al usar dos semillas se obtienen dos regiones:

R_1, asociada con la semilla (3,2), y R2, asociada a la semilla (3,4). La propiedad o predicado P empleado para añadir un píxel a una región es que la diferencia en valor absoluto entre la intensidad de dicho píxel y la semilla sea menor que un umbral T. Si un píxel satisface esa propiedad para las dos semillas se asignará a la región R_1. El resultado obtenido para T = 3 se muestra en la figura 2.24b, donde los píxeles que pertenecen a R_1 se han marcado con una "a", mientras que los pertenecientes a R_2 se marcan con una "b'. Obsérvese que, si se hubiese tomado T = 8, el resultado dependería del orden de crecimiento de las regiones. Si, por ejemplo, se deja crecer primero la región R_1, el resultado es una única región, como la que se muestra en la figura 2.24c.

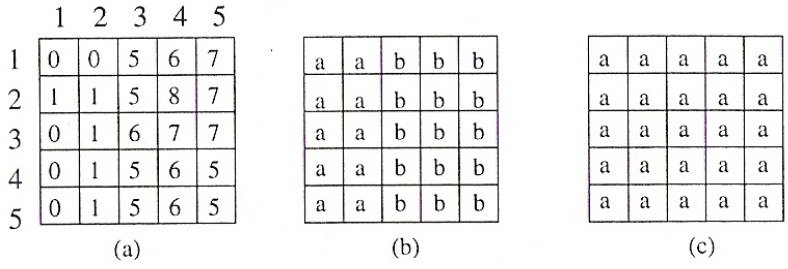


Figura 2.24. Ilustración de la técnica de crecimiento de regiones.

Esta técnica plantea dos problemas fundamentales:

1) la colocación de semillas.

2) la elección de las propiedades para la inclusión de píxeles en cada región.

La colocación de las semillas se suele basar en la naturaleza del problema, y requiere algún tipo de información previa de las regiones a segmentar.

Las propiedades que controlen la incorporación de nuevos píxeles pueden tener en cuenta diferentes características de la imagen, por ejemplo:

- **Textura:** (variaciones de los niveles de grises que siguen un determinado patrón).

Para su utilización se requieren imágenes con elevada resolución.

- **Niveles de gris**: La condición de agregación de nuevos píxeles suele tener en cuenta la intensidad media actualizada de la región (media dinámica), así como la dispersión de ésta (desviación típica de las intensidades).Suelen aparecer problemas con las sombras, reflejos, etc.

- **División y fusión de regiones**

Esta técnica se basa en la subdivisión inicial de la imagen en un conjunto de regiones arbitrarias y disjuntas. A partir de este conjunto, se comienza un proceso iterativo de

fusión y/o división de dichas regiones, sujeto a las condiciones establecidas en la introducción de la sección 2.3.10.4**.** El método viene dado por el siguiente algoritmo:

1) Sea R_0 la región inicial, constituida por la imagen completa;
2) Seleccionar un predicado P;
3) Para toda región R_i, tal que, $P(R_i)$ = FALSE: Subdividir R_i en cuatro cuadrantes disjuntos;
4) Fusionar cualquier par de regiones adyacentes R_j y R_k, para las que se verifique $P(R_j \, UR_k)$ = TRUE;
5) Si existen más regiones para fusionar o dividir entonces ir a 3, si no, parar;

Para la representación de las sucesivas subdivisiones se suele usar en un árbol cuaternario (ver figura 2.25).

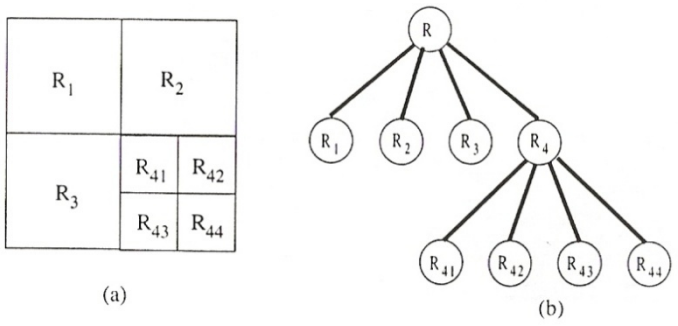


Figura2.25 a) imagen particionada según el método de división y fusión. b) Árbol cuaternario correspondiente.

Como se observa el nodo raíz corresponde a la imagen completa, mientras que cada nodo corresponde a una subdivisión. En este caso solo R_4 ha sido subdividida.

Basándose en el procedimiento básico anterior se han propuesto un gran número de variaciones [14]

Como ilustración de éste método considérese la imagen de la figura 2.26, consistente en un único objeto sobre un fondo, ambos con intensidades constantes. Se considerará que $P(R_i)$ = TRUE si todos los píxeles de R_i tienen la misma intensidad. Las figuras 2.26a, 2.26b y 2.26c muestran las distintas etapas resultantes al aplicar el algoritmo anterior.

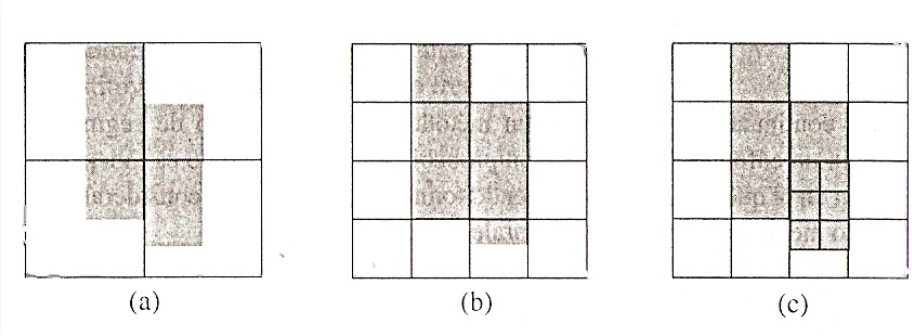


Figura 2.26 Etapas Resultantes

2.3.11 Extracción de características

Para intentar reconocer y/o localizar un objeto de la imagen es necesario extraer características del mismo que permitan representarlo y describirlo matemáticamente. En este tema se abordan distintos procedimientos para realizar esta representación y descripción.

Se entiende por representación de un objeto previamente segmentado cualquier proceso que conlleve la transformación de los píxeles que lo integran en un formato más compacto y de un nivel superior. Por ejemplo, en primera instancia el contorno de un objeto viene dado por los píxeles que integran dicha frontera.

Una forma más eficaz de representar el contorno del objeto puede ser mediante un ajuste poligonal del mismo, que simplifica tu información a manejar. A veces, los píxeles no son transformados sino que ellos mismos constituyen la representación que define el objeto.

A partir de una determinada representación es necesario describir matemáticamente el objeto, esto es, extraer sus características (color, tamaño, posición, etc.) mediante un vector, denominado vector de características, que cuantifica el valor de las mismas.

Los descriptores pueden hacer referencia a características externas de la región (su contorno), o bien a los píxe1es que forman la región en sí, recibiendo el nombre de descriptores de contorno y de región, respectivamente.

En un gran número de ocasiones es conveniente que los descriptores, además de simples, precisos y con gran poder de discriminación, sean invariantes, si es posible, ante escalado, posición y rotación del objeto en la imagen. Cuando se pretende, no sólo el reconocimiento sino también la localización, las invariancias ante posición y orientación no son, lógicamente, deseables.

2.3.12 Reconocimiento mediante redes neuronales artificiales

Las Redes Neuronales están formadas por una serie de unidades de procesamiento muy simples que están conectadas entre sí y, cuyas salidas dependen, además de la entrada, de las conexiones existentes entre las unidades de procesamiento.

Las características fundamentales que hacen de las redes neuronales una potente herramienta para el diseño de sistemas de reconocimiento de formas son la capacidad de aprendizaje, la facilidad de diseño, y la capacidad de manejo de problemas no-lineales.

2.3.12.1 Funcionamiento de una Red Neuronal

Las Redes Neuronales Artificiales se construyen a partir del modelado formal (normalmente matemático) del funcionamiento de las redes neuronales existentes en los seres vivos.

Los cuerpos celulares se representan por los nodos, y los caminos entre neuronas, consistentes en axones conectados a dendritas a través de sinapsis, se modelan mediante enlaces.

El funcionamiento de estas redes tiene un alto grado de paralelismo, cada elemento de la red ejecuta sus funciones independientemente de los demás elementos de la red. La comunicación dentro de la red es local, en el sentido de que la información fluye unidireccionalmente a través de los enlaces entre nodos. La forma

natural de comunicación en Redes Neuronales Artificiales es asíncrona aunque existen modelos que funcionan de forma síncrona.

En la figura 2.27 se muestra la arquitectura básica de una Red Neuronal. Cada nodo de la red tiene asociado un nivel de activación, normalmente dado por una función sigmoidal de la forma:

$$h_j(I_j) = \frac{1}{1 + e^{-\frac{I_j + \theta_j}{\theta_0}}} \qquad (2.25)$$

Donde I_j j= 1.2 ...N_j representa la entrada al elemento de activación de cada nodo en la capa j. Θ_j es un valor de referencia. y Θ_0 controla la forma de la función sigmoidal. Como se muestra en la figura 2.27. Esta función crece lentamente a partir de un valor mínimo (0) pasa a una región de rápido crecimiento y finalmente se aproxima asintóticamente a un valor máximo (1).

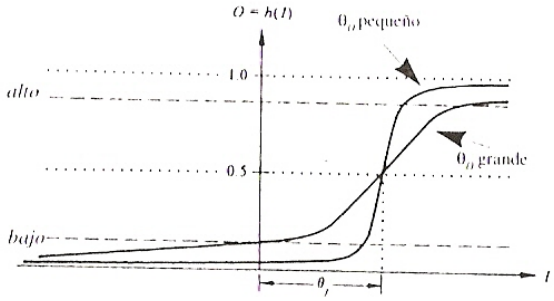


Figura 2.27.- Función sigmoidal frecuentemente utilizada como función de activación de una nodo de la red.

Cada nivel de activación de un nodo fluye a través de los enlaces con los que se conecta con otros nodos receptores. Esto supone que el nivel de activación del nodo fuente se multiplica por el peso del enlace. De este modo, la actividad de un

nodo puede afectar a los nodos receptores en diferentes grados. Esta transmisión internodo de niveles de activación se llama flujo de activación, y permite la transferencia de información en una red neuronal artificial. [9]

Como se observa en el modelo de la figura 2.28, los nodos suman algebraicamente los niveles de activación ponderados que reciben a través de los enlaces de otros nodos. Así pues, si K es la capa que precede a la J, la entrada al elemento de activación de cada nodo en la capa J viene dado por:

$$I_j = \sum_{k=1}^{N_K} w_{jk} O_k \qquad (2.26)$$

Para j=I,2, …, N_J, siendo N_J y N_K el número de nodos de las capas J y K, respectivamente, y w_{jk} son los pesos que modifican la salidas 0K de los nodos de la capa K. Las salidas de la capa K son

$$O_k = h_k(I_k) \qquad (2.27)$$

para k=1,2, …, N_K

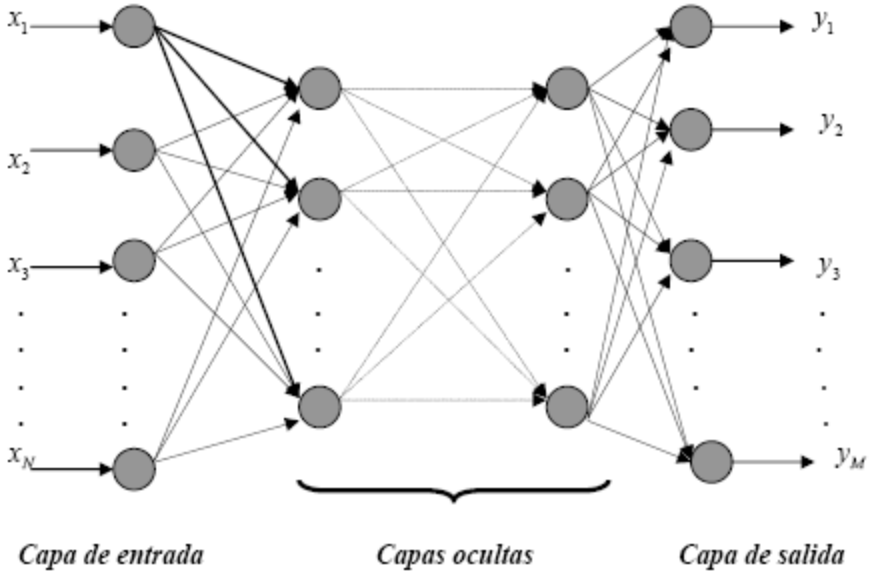


Figura 2.28.- modelo multicapas de una red neuronal.

2.3.12.2 Clasificación de modelos

Las Redes Neuronales Artificiales pueden ser caracterizadas por la topología, el modelo de neurona y el algoritmo de entrenamiento. Hay dos topologías principales: los modelos hacia adelante *(feedforward).* y los modelos con realimentación *(feedback}* . [15]

En los modelos feedback la matriz de pesos se crea a partir del producto escalar de todo patrón de entrada consigo mismo o con una entrada asociada y sumando todos los productos escalares. Los modelos más conocidos son: el modelo Hopfield y el modelo BAM.

Los primeros modelos de redes fueron lineales hacia adelante. En 1972 dos autores. *l.A*. Anderson (neurofisiólogo) y Teuvo Kohonen (ingeniero eléctrico). Propusieron simultáneamente y de forma independiente el modelo de *asociado lineal.* El asociador lineal usa una regla simple, a saber, la regla de Hebb (1949). Esto impone un límite al número de patrones que pueden ser almacenados. El sistema trabaja bien si el número de patrones aleatorios no excede del 20 por ciento del número de neuronas. Si los patrones de entrada no son ortogonales, entonces habrá interferencia entre ellos, y sólo unos pocos patrones podrán ser almacenados y recuperados.

2.3.12.3 Algoritmos de entrenamiento

Hay dos tipos principales de algoritmos de entrenamiento: supervisados y no supervisados.

- **El aprendizaje supervisado:** es la forma más elemental de adaptación. Requiere un conocimiento a priori de las respuestas deseadas. Durante el entrenamiento, la salida de la red se compara con la respuesta ideal y el error que exista se utiliza para corregir la red. El aprendizaje se produce como resultado de cambiar los pesos para reducir el error. Desde el punto de vista cognoscitivo, es como si la red ganara experiencia. Para redes monocapas ésta tarea es fácil de llevar a cabo observando y dirigiendo cada neurona individualmente. En redes multicapas la dificultad aumenta debido a la corrección de las capas ocultas (intermedias).

- **El aprendizaje no-supervisado:** no requiere influencia externa a la Red Neuronal Artificial para ajustar los pesos de las conexiones entre neuronas. La red no recibe ninguna información por parte del entorno que le indique si la

salida generada es o no correcta, así que existen varias posibilidades en cuanto a la interpretación de la salida de estas redes.

- **El modelo supervisado Backpropagation:** es un tipo de red que emplea un ciclo propagación – adaptación de dos fases. Una vez que se ha aplicado un patrón a la entrada de la red como estímulo, este se propaga desde la primera capa a través de las capas superiores de la red, hasta generar una salida. La señal de salida se compara con la salida deseada y se calcula una señal de error para cada una de las salidas.

Las salidas de error se propagan hacia atrás, partiendo de la capa de salida, hacia todas las neuronas de la capa oculta que contribuyen directamente a la salida. Sin embargo las neuronas de la capa oculta solo reciben una fracción de la señal total del error, basándose aproximadamente en la contribución relativa que haya aportado cada neurona a la salida original. Este proceso se repite, capa por capa, hasta que todas las neuronas de la red hayan recibido una señal de error que describa su contribución relativa al error total.

Basándose en la señal de error percibida, se actualizan los pesos de conexión de cada neurona, para hacer que la red converja hacia un estado que permita clasificar correctamente todos los patrones de entrenamiento.

La importancia de este proceso consiste en que, a medida que se entrena la red, las neuronas de las capas intermedias se organizan a sí mismas de tal modo que las distintas neuronas aprenden a reconocer distintas características del espacio total de entrada.

Después del entrenamiento, cuando se les presente un patrón arbitrario de entrada que contenga ruido o que esté incompleto, las neuronas de la capa oculta de la red responderán con una salida activa si la nueva entrada contiene un patrón que se asemeje a aquella característica que las neuronas individuales hayan aprendido a reconocer durante su entrenamiento. Y a la inversa, las unidades de las capas ocultas tienen una tendencia a inhibir su salida si el patrón de entrada no contiene la característica para reconocer, para la cual han sido entrenadas.

Varias investigaciones han demostrado que, durante el proceso de entrenamiento, el algoritmo Backpropagation tiende a desarrollar relaciones internas entre neuronas con el fin de organizar los datos de entrenamiento en clases.

Esta tendencia se puede extrapolar, para llegar a la hipótesis consistente en que todas las unidades de la capa oculta de una red neuronal artificial son asociadas de alguna manera a características específicas del patrón de entrada como consecuencia del entrenamiento. Lo que sea o no exactamente la asociación puede no resultar evidente para el observador humano, lo importante es que la red ha encontrado una representación interna que le permite generar las salidas deseadas cuando se le dan las entradas, en el proceso de entrenamiento. Esta misma representación interna se puede aplicar a entradas que la red no haya visto antes, y la red clasificará estas entradas según las características que compartan con los ejemplos de entrenamiento.

Uno de los paradigmas más implementado actualmente es el de Backpropagation porque es el mejor modelo de propósito general y, probablemente el que mejor generaliza las respuestas.

2.3.12.4 Estructura del algoritmo Backpropagation

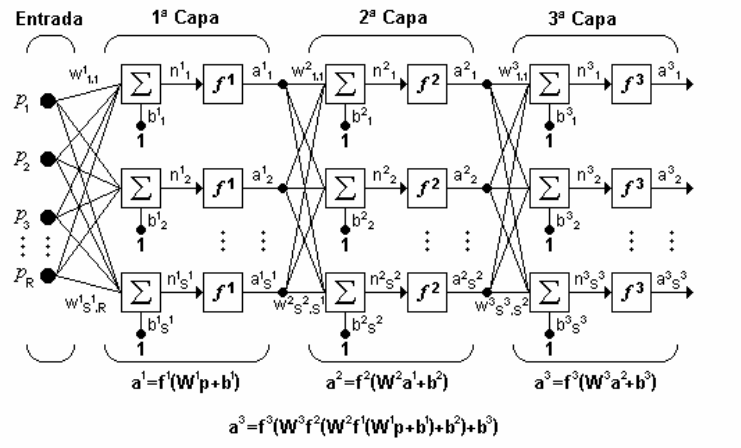


Figura 2.29 Red de tres capas

La estructura típica de una red multicapa se observa en la figura 2.29

Puede notarse que esta red de tres capas equivale a tener tres redes tipo Perceptrón en cascada; la salida de la primera red, es la entrada a la segunda y la salida de la segunda red es la entrada a la tercera. Cada capa puede tener diferente número de neuronas, e incluso distinta función de transferencia.

En la figura 2.29 $\boldsymbol{W}^1$ representa la matriz de pesos para la primera capa, $\boldsymbol{W}^2$ los pesos de la segunda y así similarmente para todas las capas que incluya una red. Para identificar la estructura de una red multicapa, se empleará una notación abreviada, donde el número de entradas va seguido del número de neuronas en cada capa: $R : S^1 : S^2 : S^3$.

Donde S representa el número de neuronas y el exponente representa la capa a la cual la neurona corresponde.

La notación de la figura 2.29 es bastante clara cuando se desea conocer la estructura detallada de la red, e identificar cada una de las conexiones, pero cuando la red es muy grande, el proceso de conexión se torna muy complejo y es bastante útil utilizar el esquema de la figura 2.30. [13]

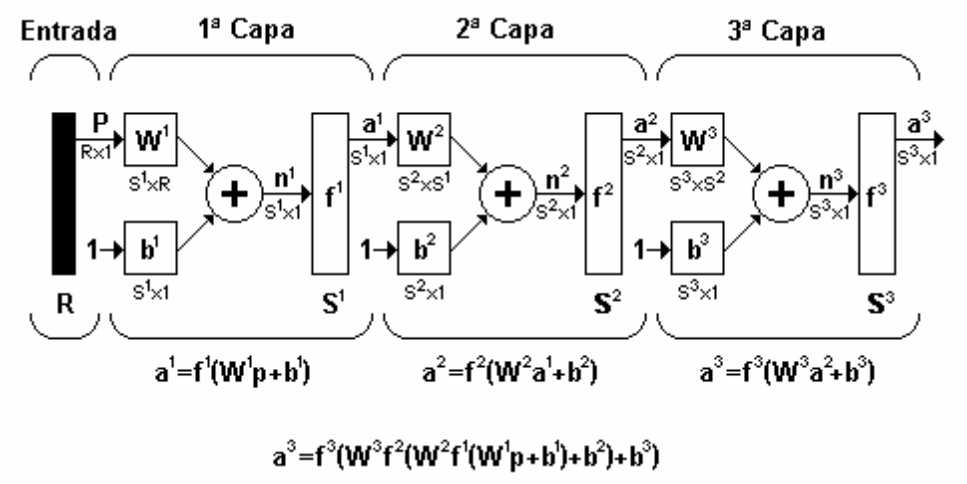


Figura 2.30 Notación compacta de una red de tres capas

- **Funcionamiento red del algoritmo Backpropagation**

El sistema de entrenamiento mediante Backpropagation consiste en:

- Empezar con unos pesos sinápticos cualquiera (generalmente elegidos al azar).
- Introducir unos datos de entrada (en la capa de entradas) elegidos al azar entre los datos de entrada que se van a usar para el entrenamiento.
- Dejar que la red genere un vector de datos de salida (propagación hacia delante).
- Comparar la salida generada por la red con la salida deseada.
- La diferencia obtenida entre la salida generada y la deseada (denominada *error*) se usa para ajustar los pesos sinápticos de las neuronas de la capa de salidas.
- El error se *propaga hacia atrás* (Backpropagation), hacia la capa de neuronas anterior, y se usa para ajustar los pesos sinápticos en esta capa.
- Se continua propagando el error hacia atrás y ajustando los pesos hasta que se alcance la capa de entradas.

Este proceso se repetirá con los diferentes datos de entrenamiento. [16]

- **Regla de Aprendizaje**

El algoritmo LMS (del inglés, *Least-Mean-Square algorithm*) se usa para encontrar los coeficientes que permiten obtener el valor esperado mínimo del cuadrado de la señal de error, definida como la diferencia entre la señal deseada y la señal producida a la salida del filtro.

El algoritmo Backpropagation para redes multicapa es una generalización del algoritmo LMS, ambos algoritmos realizan su labor de actualización de pesos y ganancias con base en el error medio cuadrático. Éste algoritmo trabaja bajo aprendizaje supervisado y por tanto necesita un set de entrenamiento que le describa cada salida y su valor de salida esperado de la siguiente forma:

$$T= (X_1, Y_1), (X_2, Y_2), \ldots, (X_Q, Y_Q)\} \quad (2.28)$$

Donde X_Q es una entrada a la red e Y_Q es la correspondiente salida deseada para el patrón q-ésimo. El algoritmo debe ajustar los parámetros de la red para minimizar el error medio cuadrático.

El entrenamiento de una Red Neuronal multicapa se realiza mediante un proceso de aprendizaje, para realizar este proceso se debe inicialmente tener definida la topología de la red, esto es: número de neuronas en la capa de entrada el cual depende del número de componentes del vector de entrada, cantidad de capas ocultas y número de neuronas de cada una de ellas, número de neuronas en la capa de la salida el cual depende del número de componentes del vector de salida o patrones objetivo y funciones de transferencia requeridas en cada capa, con base en la topología escogida se asignan valores iníciales a cada uno de los parámetros que conforma la red.

Es importante recalcar que no existe una técnica para determinar el número de capas ocultas, ni el número de neuronas que debe contener cada una de ellas para un

problema específico, esta elección es determinada por la experiencia del diseñador, el cual debe cumplir con las limitaciones de tipo computacional.

Cada patrón de entrenamiento se propaga a través de la red y sus parámetros para producir una respuesta en la capa de salida, la cual se compara con los patrones objetivo o salidas deseadas para calcular el error en el aprendizaje, este error marca el camino más adecuado para la actualización de los pesos y ganancias que al final del entrenamiento producirán una respuesta satisfactoria a todos los patrones de entrenamiento, esto se logra minimizando el error medio cuadrático en cada iteración del proceso de aprendizaje. [16]

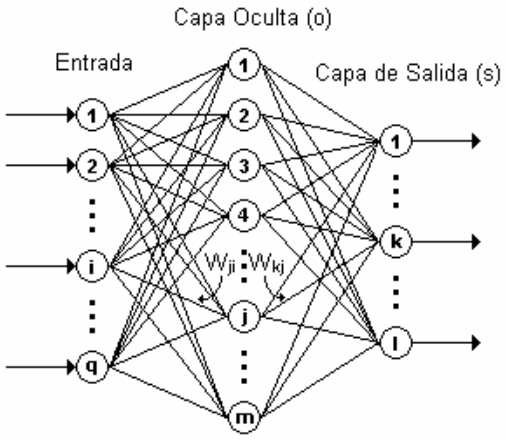


Figura 2.31 Disposición de una red sencilla de 3 capas

Es importante aclarar que en la figura 2.31:

q: Equivale al número de componentes el vector de entrada.

m: Número de neuronas de la capa oculta

l: Número de neuronas de la capa de salida

2.3.12.5 Calculo del Error

El error es una función que evalúa la diferencia entre las salidas de la red y las salidas deseadas. En la mayor parte de los casos, la función error se define como: [12]

$$E = \frac{1}{2} \left(\sum_v Y_i^v - S_i^v\right)^2 \quad (2.29)$$

Donde:

Yi = denotan las salidas deseadas de las neuronas de la capa de salida.

v = es el número de patrones o muestras del conjunto de entrenamiento.

Si = denotan las salidas obtenidas en la capa de salida.

Las salidas Yi vienen dadas en el conjunto de entrenamiento, para el cálculo de las salidas Si se utiliza la siguiente ecuación:

$$S_i^v = \left(\sum_j W_{ij} \left(\sum_k w_{jk} X_k^v\right)\right) \quad (2.30)$$

Donde:

F = es la función de transferencia.

Wij = denota los pesos sinápticos de las neuronas de salida;

wjk = denota los pesos sinápticos de las neuronas de la capa oculta;

Xk = denota los componentes de los patrones de entrenamiento.

La función de transferencia se denota por la siguiente ecuación:

$$(x) = \frac{1}{1+e(-x)} \quad (2.31)$$

2.3.12.6 Modificación de los Pesos

La forma explícita para la modificación de los pesos es de gran importancia práctica.

Para los pesos sinápticos W_{ij} de las neuronas de la capa de salida se tienen los siguientes cálculos: [12]

$$\delta W_{ij} = -\lambda \frac{\partial E}{\partial Wij} = \lambda \sum_v \Delta_i^v V_j^v \quad (2.32)$$

Donde:

E = representa la ecuación de error.

Y Δi se calcula de la siguiente forma:

$$\Delta_i^v = '(\sum_h W_{ih} V_h^v)(Y_i^v - S_i^v) \quad (2.33)$$

Donde

h= representa el número de neuronas de la capa oculta.

En este caso V juega el papel de patrón de entrada.

En el caso de los pesos sinápticos w_{jk} de las neuronas de la capa oculta, al calcular el gradiente, se deriva con respecto a w_{jk} variable implícita en la expresión de la función de error (2.27), de tal modo que empleando la regla de la cadena tenemos:

$$\delta w_{jk} = -\lambda \frac{\partial E}{\partial w_{jk}} = \lambda \sum_{v} \delta_j^v X_k^v \quad (2.34)$$

Donde tenemos que:

$$\delta_j^v = \mathrm{F}'\left(\sum_h w_{jh} X_h^v\right) \sum_i \Delta_i^v W_{ij} \quad (2.35)$$

La derivada de la función de transferencia F' para las ecuaciones 2.32 y 2.35 se denota por:

$$\mathrm{F}'(x) = \frac{1}{1+e(-x)}(-e(-x) = \frac{1}{1+e(-x)} \frac{e(-x)}{1+e(-x)} \quad (2.36)$$

Simplificando tenemos:

$$\mathrm{F}'(x) = \mathrm{F}(x)(1-\mathrm{F}(x)) \quad (2.37)$$

Donde F(x) está representada en la ecuación 2.29.

La importancia del algoritmo Backpropagation consiste en su capacidad de autoadaptar los pesos de las neuronas de las capas intermedias para aprender la relación que existe entre un conjunto de patrones dados y sus salidas correspondientes, esto da un porcentaje muy elevado de reconocimiento al momento de la búsqueda de estos, es por esto que es la red más utilizada para el reconocimiento de los mismos.

2.4 LENGUAJE UNIFICADO DE MODELADO (UML)

2.4.1 Definición

El lenguaje de modelado unificado contiene una notación robusta para el modelado y desarrollo de sistemas orientados a objeto. Proporciona la tecnología necesaria para apoyar la práctica de la ingeniería del software orientada a objetos.

Como resultado de la aplicación de UML se puede producir un arreglo de modelos y documentos de trabajo. Sin embargo, éstos los reducen los ingenieros de software para lograr que el desarrollo sea más ágil y reactivo ante el cambio.

Otros métodos de modelaje como OMT (Object Modeling Technique) o Booch sí definen procesos concretos. En UML los procesos de desarrollo son diferentes según los distintos dominios de trabajo; no puede ser el mismo el proceso para crear una aplicación en tiempo real, que el proceso de desarrollo de una aplicación orientada a gestión, por poner un ejemplo. [10]

2.4.2 Elementos de UML

- **Actor**: es una entidad externa (de fuera del sistema) que interacciona con el sistema participando (y normalmente iniciando) en un caso de uso. Los actores pueden ser gente real (por ejemplo, usuarios del sistema), otros ordenadores o eventos externos.

Los actores no representan a personas físicas o a sistemas, sino su papel. Esto significa que cuando una persona interacciona con el sistema de diferentes maneras, estará representado por varios actores. [6]

- **Clases**: En una clase se agrupan todos los objetos que comparten los mismos atributos, métodos y relaciones. Los atributos son características y propiedades comunes en todos los objetos de la clase. Los métodos son operaciones que deben cumplir las instancias de la clase. Las clases se representan como un rectángulo donde figuran el nombre de la clase, sus atributos y sus métodos. [6]

- **Artefacto**: es una información que es utilizada o producida mediante un proceso de desarrollo de software. Pueden ser artefactos un modelo, una descripción o un software. Los artefactos de UML se especifican en forma de diagramas, éstos, junto con la documentación sobre el sistema constituyen los artefactos principales que el modelador puede observar. [6]

- **Estado:** Los estados son los ladrillos de los diagramas de estado. Un estado pertenece a exactamente una clase y representa un resumen de los valores y atributos que puede tener la clase. Un estado UML describe el estado interno de un objeto de una clase particular. [8]

Tenga en cuenta que no todos los cambios en los atributos de un objeto deben estar representados por estados, sino únicamente aquellos cambios que pueden afectar significativamente a la forma de funcionamiento del objeto. [4]

- **Actividad**: es un único paso de un proceso. [8]

2.4.3 Diagramas de UML

Un diagrama es una representación gráfica de una colección de elementos del modelo, que habitualmente toma forma de grafo donde los arcos que conectan sus vértices son las relaciones entre los objetos y los vértices se corresponden con los elementos del modelo.

Los distintos puntos de vista de un sistema real que se quieren representar para obtener el modelo se dibuja de forma que se resaltan los detalles necesarios para entender el sistema. [4]

- **Diagramas de Casos de Uso**: Un diagrama de casos de uso es un diagrama que muestra un conjunto de casos de uso con sus relaciones y los actores implicados. Es un diagrama que sirve para modelar la vista estática de un programa. La vista estática nos permite visualizar el comportamiento externo del programa; de esta forma se consigue conocer qué es lo que debe hacer el programa independientemente de cómo lo haga y sabremos los elementos que interactúan con el sistema. Los elementos implicados en un diagrama de casos

de uso son los casos de uso, las relaciones y los actores. Las relaciones y los casos de uso ya han sido explicados anteriormente y el papel del actor también ha sido comentado pero merece la pena detallarlo más: Un actor es un rol que interactúa con el sistema. Se define como rol porque un actor puede ser tanto un usuario de la aplicación como otro sistema o dispositivos externos.

Los diagramas de casos de uso se utilizan para ilustrar los requerimientos del sistema al mostrar cómo reacciona una respuesta a eventos que se producen en el mismo. En este tipo de diagrama intervienen algunos conceptos nuevos: un actor es una entidad externa al sistema que se modela y que puede interactuar con él; un ejemplo de actor podría ser un usuario o cualquier otro sistema. Las relaciones entre casos de uso y actores pueden ser las siguientes: [4]

- Un actor se comunica con un caso de uso.
- Un caso de uso extiende otro caso de uso.
- Un caso de uso usa otro caso de uso.

- **Diagramas de Secuencia**: Un diagrama de secuencia es un diagrama de interacción UML. Estos diagramas muestran la secuencia de mensajes que se van lanzando los objetos implicados en una determinada operación del programa. Dentro del diagrama los objetos se alinean en el eje X respetando su orden de aparición. En el eje Y se van mostrando los mensajes que se envían, también respetando su orden temporal. Cada objeto tiene una línea de vida donde se sitúa su foco de control. El foco de control es un rectángulo que representa el tiempo durante el que un objeto está activo ejecutando una acción. Con este sencillo esquema podemos visualizar la comunicación y

sincronización bajo un estricto orden temporal de los objetos implicados en las distintas funcionalidades de un sistema. [8]

- **Diagrama de Clases de Análisis:** Es utilizado por los desarrolladores de software para determinar los requerimientos funcionales, considerando una o varias clases, o sub-sistemas del sistema a desarrollar. Los casos de uso se describen mediante clases de análisis y sus objetos. El diagrama de clases de análisis se construye examinando los casos de usuarios, cerrando sus reacciones e identificando los roles de los clasificadores. [4]

- **Diagrama de Clases de Diseño:** Se emplean para modelar la estructura estática de las clases en el sistema, sus tipos, sus contenidos y las relaciones que se establecen entre ellos. A través de este diagrama se definen las características de cada una de las clases, interfaces, colaboraciones y relaciones de dependencia y generalización**.** [8]

- **Diagramas de actividad**: Son similares a los diagramas de flujo de otras metodologías Orientadas a Objetos. En realidad se corresponden con un caso especial de los diagramas de estado donde los estados son estados de acción (estados con una acción interna y una o más transiciones que suceden al finalizar esta acción, o lo que es lo mismo, un paso en la ejecución de lo que será un procedimiento) y las transiciones vienen provocadas por la finalización de las acciones que tienen lugar en los estados de origen. Siempre van unidos a una clase o a la implementación de un caso de uso o de un método (que tiene el mismo significado que en cualquier otra metodología OO). Los diagramas de actividad se utilizan para mostrar el flujo de operaciones que se desencadenan en un procedimiento interno del sistema.[6]

- **Diagramas de Implementación**: Se derivan de los diagramas de proceso y módulos de la metodología de Booch, aunque presentan algunas

modificaciones. Los diagramas de implementación muestran los aspectos físicos del sistema. Incluyen la estructura del código fuente y la implementación, en tiempo de implementación. Existen dos tipos:

- **Diagramas de componentes:** Muestra la dependencia entre los distintos componentes de software, incluyendo componentes de código fuente, binario y ejecutable. Un componente es un fragmento de código software (un fuente, binario o ejecutable) que se utiliza para mostrar dependencias en tiempo de compilación.

- **Diagrama de plataformas o despliegue:** Muestra la configuración de los componentes hardware, los procesos, los elementos de procesamiento en tiempo de ejecución y los objetos que existen en tiempo de ejecución. En este tipo de diagramas intervienen nodos, asociaciones de comunicación, componentes dentro de los nodos y objetos que se encuentran a su vez dentro de los componentes. Un nodo es un objeto físico en tiempo de ejecución, es decir una máquina que se compone habitualmente de, por lo menos, memoria y capacidad de procesamiento, a su vez puede estar formado por otros componentes.[6]

- **Modelo de Dominio:** es un artefacto de la disciplina de análisis, construido con las reglas de UML durante la fase de concepción, en la tarea construcción del modelo de dominio, presentado como uno o más diagramas de clases y que contiene, no conceptos propios de un sistema de software sino de la propia realidad física.

Los modelos de dominio pueden utilizarse para capturar y expresar el entendimiento ganado en un área bajo análisis como paso previo al diseño de un sistema, ya sea de software o de otro tipo. Similares a los mapas mentales utilizados en el aprendizaje, el modelo de dominio es utilizado por el analísta como un medio para comprender el sector industrial o de negocios al cual el sistema va a servir [8]

- **Diagrama de Paquetes**: Los diagramas de paquetes se usan para reflejar la organización de paquetes y sus elementos. Cuando se emplean para representaciones, los diagramas de paquete de los elementos de clase se usan para proveer una visualización de los espacios de nombres. Los elementos contenidos en un paquete comparten el mismo espacio de nombre, el hecho de compartir espacios de nombres requiere que los elementos contenidos en un espacio de nombre específico tengan nombres únicos. Los paquetes se pueden construir para representar relaciones tanto físicas como lógicas. [8]

2.4.4 Metodología OMT

La metodología OMT (Object Modeling Technique) fue creada por James Rumbaugh y Michael Blaha en 1991, mientras James dirigía un equipo de investigación de los laboratorios General Electric. [4]

OMT es una de las metodologías de *análisis y diseño orientadas a objetos, más maduras* y eficientes que existen en la actualidad. [4]

Las fases que conforman a la metodología OMT son:

- **Análisis**. El analista construye un modelo del dominio del problema, mostrando sus propiedades más importantes. El modelo de análisis es una abstracción resumida y precisa de lo que debe de hacer el sistema deseado y no de la forma en que se hará. Los elementos del modelo deben ser conceptos del dominio de aplicación y no conceptos informáticos tales como estructuras de datos. Un buen modelo debe poder ser entendido y criticado por expertos en el dominio del problema que no tengan conocimientos informáticos. [4]

- **Diseño del sistema**. El diseñador del sistema toma decisiones de alto nivel sobre la arquitectura del mismo. Durante esta fase el sistema se organiza en subsistemas basándose tanto en la estructura del análisis como en la arquitectura propuesta. Se selecciona una estrategia para afrontar el problema. [4]

- **Diseño de objetos**. El diseñador de objetos construye un modelo de diseño basándose en el modelo de análisis, pero incorporando detalles de implementación. El diseño de objetos se centra en las estructuras de datos y algoritmos que son necesarios para implementar cada clase. OMT describe la forma en que el diseño puede ser implementado en distintos lenguajes (orientados y no orientados a objetos, bases de datos, etc.). [4]

- **Implementación**. Las clases de objetos y relaciones desarrolladas durante el análisis de objetos se traducen finalmente a una implementación concreta. Durante la fase de implementación es importante tener en cuenta los

principios de la ingeniería del software de forma que la correspondencia con el diseño sea directa y el sistema implementado sea flexible y extensible. [4]

La metodología OMT emplea tres clases de modelos para describir el sistema:

- **Modelo de objetos.** Describe la estructura estática de los objetos del sistema (identidad, relaciones con otros objetos, atributos y operaciones). El modelo de objetos proporciona el entorno esencial en el cual se pueden situar el modelo dinámico y el modelo funcional. El objetivo es capturar aquellos conceptos del mundo real que sean importantes para la aplicación. Se representa mediante diagramas de objetos.

- **Modelo dinámico.** Describe los aspectos de un sistema que tratan de la temporización y secuencia de operaciones (sucesos que marcan los cambios, secuencias de sucesos, estados que definen el contexto para los sucesos) y la organización de sucesos y estados. Captura el control, aquel aspecto de un sistema que describe las secuencias de operaciones que se producen sin tener en cuenta lo que hagan las operaciones, aquello a lo que afecten o la forma en que están implementadas. Se representa gráficamente mediante diagramas de estado.

- **Modelo funcional.** Describe las transformaciones de valores de datos (funciones, correspondencias, restricciones y dependencias funcionales) que ocurren dentro del sistema. Captura lo que hace el sistema,

independientemente de cuándo se haga o de la forma en que se haga. Se representa mediante diagramas de flujo de datos. [4]

La OMT, por ejemplo, intenta abstraer la realidad utilizando tres clases de modelos OO: el modelo de objetos, que describe la estructura estática; el modelo dinámico, con el que describe las relaciones temporales entre objetos; y el modelo funcional que describe las relaciones funcionales entre valores. Mediante estas tres fases de construcción de modelos, se consigue una abstracción de la realidad que tiene en sí misma información sobre las principales características de ésta. [4]

Los modelos además, al no ser una representación que incluya todos los detalles de los originales, permiten probar más fácilmente los sistemas que modelan y determinar los errores.

Según se indica en la Metodología OMT (Rumbaugh), los modelos permiten una mejor comunicación con el cliente por distintas razones:

- Es posible enseñar al cliente una posible aproximación de lo que será el producto final.
- Proporcionan una primera aproximación al problema que permite visualizar cómo quedará el resultado.
- Reducen la complejidad del original en subconjuntos que son fácilmente tratables por separado. [4]

Se consigue un modelo completo de la realidad cuando el modelo captura los aspectos importantes del problema y omite el resto. Los lenguajes de programación que estamos acostumbrados a utilizar no son adecuados para realizar modelos completos de sistemas reales porque necesitan una especificación total con detalles que no son importantes para el algoritmo que están implementando.

En OMT se modela un sistema desde tres puntos de vista diferentes donde cada uno representa una parte del sistema y una unión lo describe de forma completa. En esta técnica de modelado se utilizó una aproximación al proceso de implementación de software habitual donde se utilizan estructuras de datos (modelo de objetos), las operaciones que se realizan con ellos tienen una secuencia en el tiempo (modelo dinámico) y se realiza una transformación sobre sus valores (modelo funcional). [4]

CAPITULO 3: FASE DE ANÁLISIS

3.1 Introducción

En este capítulo se procederá a describir por medio de diagramas UML la fase de inicio del Sistema Reconocedor de Placa (SRP).

El propósito fundamental de este capítulo es guiar el desarrollo hacia el sistema correcto. Esto se consigue mediante una descripción de los requisitos del sistema, es decir, que las condiciones o capacidades que el sistema debe cumplir, sean suficientemente buenas como para que pueda llegarse a un acuerdo entre los usuarios y los desarrolladores sobre qué debe y qué no debe hacer el sistema.

Existen diferentes puntos de partida para la captura de los requisitos correctos y para la construcción del sistema deseado. Para ello, es necesario que los desarrolladores, posean un firme conocimiento del contexto en el que se emplaza el sistema, pudiendo tomar en consideración para lograrlo.

3.2 Requisitos

En esta parte del capítulo presentaremos los requisitos necesarios para el buen funcionamiento del sistema SRP.

3.2.1 Requisitos Funcionales

Los requisitos funcionales, son aquellos que muestran la forma en la que el sistema interactúa con él o los usuarios finales, es decir, se toman en cuenta los procesos y métodos que se necesitan para ejecutar la interacción. A continuación se muestran los requisitos funcionales identificados en el sistema.

A continuación se enumeran los requisitos funcionales del sistema SRP:

- El sistema debe ser capaz de cargar una imagen con los siguientes formatos: JPG, TIFF, Bitmap y .PNG.
- El sistema debe permitir que el usuario visualice la placa obtenida por la aplicación al terminar la búsqueda en ésta en la imagen.
- El sistema debe guardar en un arreglo cada uno de los caracteres de la placa obtenida.
- El sistema debe ser capaz de dar una solución óptima al problema planteado por el usuario.

3.2.2 Requisitos No Funcionales

Los requisitos no funcionales especifican propiedades del sistema, como restricciones del entorno o de la implementación, rendimiento, dependencias de la plataforma, facilidad de mantenimiento, extensibilidad, y fiabilidad.

Los requisitos que no son vitales para el funcionamiento del sistema SRP:

- El sistema debe contar con una interfaz gráfica agradable de tal forma que, permita interactuar, de forma fácil, segura y cómoda; entre los usuarios y el sistema.
- El sistema debe ser realizado bajo plataforma de software libre basándose en el decreto presidencial 3.390, con el objeto de permitir que el código fuente pueda ser modificado sin restricciones legales además de reducir el costo de desarrollo.
- La estructura y diseño del sistema deben adaptarse fácilmente a cualquier cambio o mejora en los mismos.

3.2.3 Requisitos de Software

- Sistema Operativo Windows XP service pack 2 o superior.
- Sistema Operativo Linux.
- Java Runtime Environment (JRE) 1.7.
- Java Advanced Imaging 1.3 o superior.

3.2.4 Requisitos de Hardware

- Pentium 4 3.0 GHz o Superior
- 512 MB de memória RAM o superior.

3.3 Riesgos del Sistema

En todo desarrollo de sistema es importante considerar los riesgos; entiéndase por riesgos la variable del proyecto que pone en peligro o impide el éxito del mismo. Los riegos constituyen la probabilidad de que un proyecto sufra sucesos no deseables.

- La aplicación no consiga la placa dentro de la imagen.
  La prioridad de este riesgo es crítica ya que de darse esto el sistema no estaría cumpliendo con los requisitos del usuario. La solución sería validar que la fotografía contenga la placa.

- La imagen de entrada no tiene el formato correcto.
  La prioridad es secundaria, la solución sería indicarle al usuario que visite la ayuda del sistema donde se le indicaría el formato correcto de dicha imagen.

- La placa se encuentre muy inclinada.
  La prioridad es crítica ya que no se podría encontrar los caracteres de la placa debido a la distorsión (cambio de morfología) de los mismos. La solución sería indicarle al usuario que visite la ayuda del sistema donde se le indicaría el formato correcto de dicha imagen.

- La foto contenga sombra sobre la placa.
  La prioridad es crítica ya que no se podría encontrar los caracteres de la placa debido a la distorsión (cambio de morfología) de los mismos. La solución sería indicarle al usuario que visite la ayuda del sistema donde se le indicaría el formato correcto de dicha imagen.

- La distancia a la que se tomó la foto sea muy grande o muy pequeña.
  La prioridad es crítica ya que no se podría encontrar los caracteres de la placa debido a la distorsión (cambio de morfología) de los mismos. La solución sería indicarle al usuario que visite la ayuda del sistema donde se le indicaría el formato correcto de dicha imagen.

3.4 Diagrama de Dominio

Un modelo de dominio es una representación visual de las clases conceptuales u objetos del mundo real en un dominio de interés. También se les denomina modelos conceptuales, modelo de objetos del dominio y modelos de objetos de análisis.

Para la descripción del modelo de dominio debe realizarse diagramas de clases basado en Lenguaje de Modelado UML.

En la figura 3.1 se representan las clases más importantes del modelo de dominio.

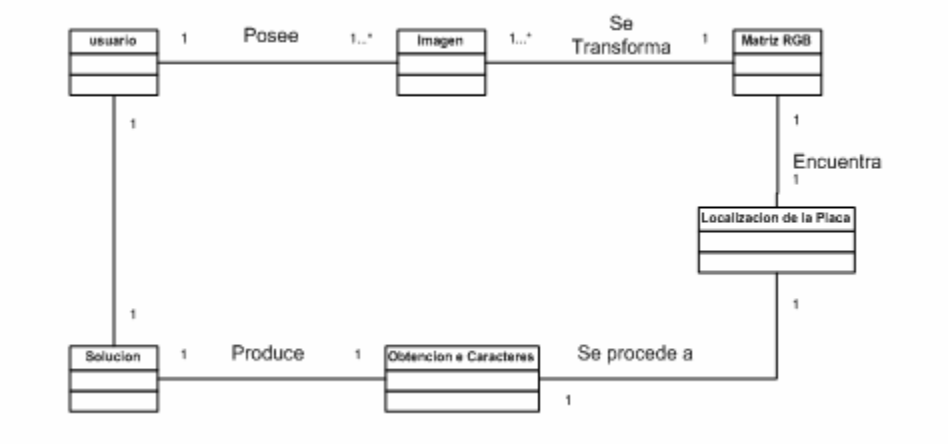


Figura 3.1 Diagrama de Dominio

3.4.1 Glosario de términos del diagrama de dominio

- **Usuario:** es quien introduce la imagen.
- **Imagen:** contiene la placa a buscar.
- **Matriz RBG:** la imagen se divide en pixeles y se carga una matriz con sus valores en RBG.
- **Localización de la placa:** se realizan todas las operaciones morfológicas y matemáticas para lograr encontrar el lugar donde se encuentra la placa.
- **Obtención de caracteres:** se aplica el algoritmo BackPropagation para buscar los patrones y reconocer los caracteres obtenidos en dicha placa.
- **Solución:** aquí el usuario visualiza los números de la placa que se encuentra en la foto.

3.5 Diagrama de Caso de Uso

El modelo de casos de uso incluye los casos de usos y actores. Los casos de uso son fragmentos de funcionalidad que el sistema ofrece para aportar un resultado de valor para sus actores.

En la figura 3.2 se muestra el caso de uso del sistema SRP.

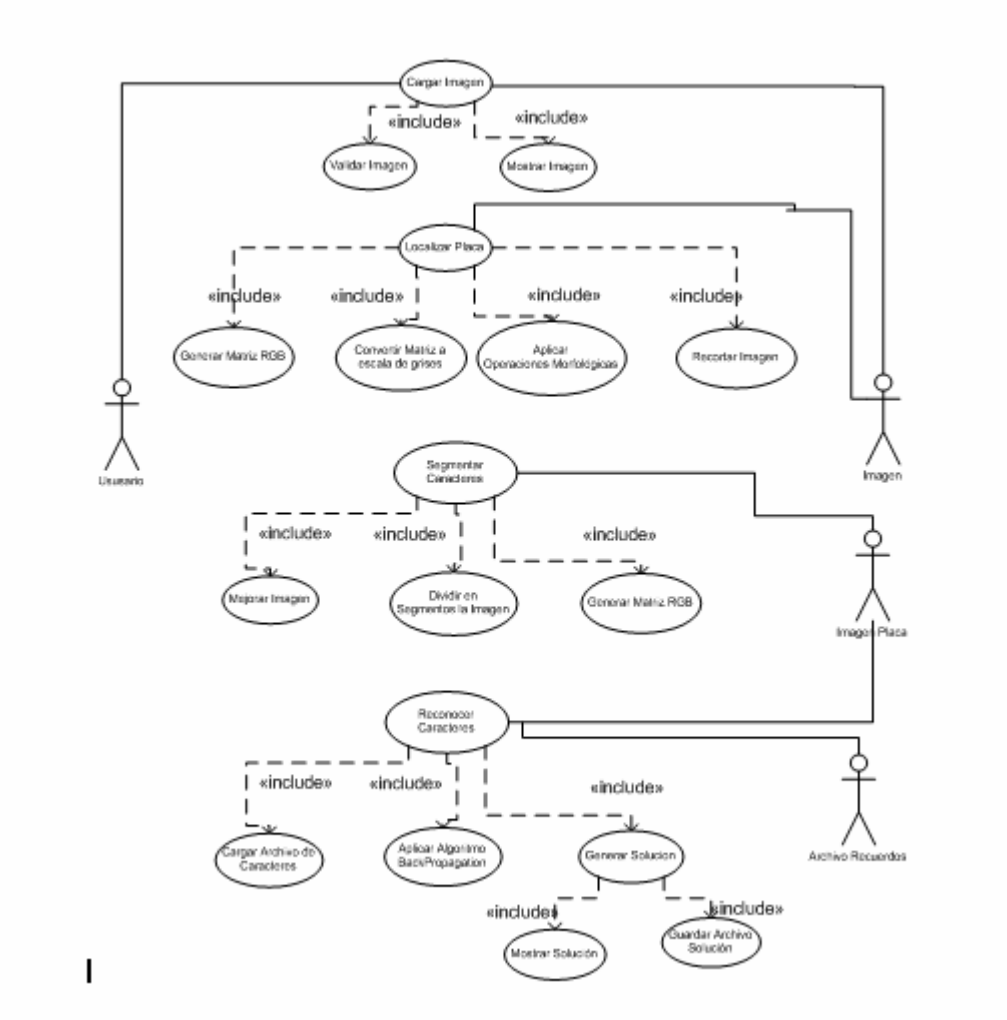


Figura 3.2 Caso de Uso del sistema SRP

3.5.1 Identificación de Actores

Los actores representan a todos aquellos que de una u otra manera interactúan con el sistema. Estos pueden ser los usuarios (personas que hacen uso del sistema) y

también pueden estar representados por sistemas o dispositivos externos que se comunican con el sistema. Es importante realizar la identificación de los actores ya que ello permite obtener una visión del entorno externo del sistema.

A continuación se describe cada uno de los actores que forman parte del ambiente en que se desempeña el sistema.

- **Usuario:** como su nombre lo indica es quien va a usar el sistema, este se encarga de suministrar la imagen donde se encuentra el vehículo con la placa a buscar.
- **Imagen:** contiene la información con la placa a buscar.
- **Imagen Placa:** contiene la imagen de la placa ya recortada.
- **Archivo Recuerdos:** los caracteres de muestra contenidos en la Red Neuronal BackPropagation.

3.5.2 Identificación de Casos de Uso

- **Caso de Uso 1 : Cargar Archivo**
  Este caso de uso se encarga de leer la imagen que contiene la placa a buscar, ésta es suministrada por el usuario y consta de 2 casos de usos los cuales son:

  - **Caso de Uso 1.1: Validar Imagen**

Este caso de uso se encarga de comprobar que la imagen contenga el formato que requiere el sistema, en caso contrario se le emite un mensaje de advertencia al usuario para que verifique el formato permitido a utilizar en el sistema.

- **Caso de Uso 1.2: Mostrar Imagen**

  En este caso de uso una vez que se verifica la imagen se muestra en la aplicación.

- **Caso de Uso 2: Localizar Placa**

  Este caso de uso, como lo indica su nombre, se encarga de localizar la ubicación de la placa dentro de la imagen. Este caso de uso está formado por:

  - **Caso de Uso 2.1: Generar Matriz RGB**

    En este caso de uso la imagen a tratar es representada mediante una matriz donde cada casilla va a estar cargada con el valor de un píxel en RGB.

  - **Caso de Uso 2.2: Convertir Matriz a escala de grises**

    En este caso de uso se procede a convertir la matriz RGB en escala de grises, es decir pasarla a una sola banda mediante el uso de una LUT (look up table) o tabla de consulta.

  - **Caso de Uso 2.3: Aplicar Operaciones Morfológicas**

    En esta caso de uso se aplican las operaciones morfológicas (dilatación, erosión, opening y closing) con la finalidad de eliminar los posibles ruidos (quitar los pixeles que no sean necesarios) en la imagen para poder localizar la placa.

- **Caso de Uso 2.4: Recortar Imagen**

  En este caso de uso, luego de encontrar la ubicación de la placa se procede a recortar ese espacio para su posterior procesamiento.

- **Caso de Uso 3: Segmentar Caracteres**

  En este caso de uso se procede a separar a cada uno de los caracteres que se encuentran dentro de la placa encontrada para su posterior reconocimiento, para lograr esto se hacen necesarios los siguientes casos de uso:

  - **Caso de Uso 3.1: Mejorar Imagen**

    Se escala la imagen para facilitar la búsqueda de los caracteres de la placa.

  - **Caso de Uso 3.2: Dividir en Segmentos la Imagen**

    Se aplica crecimiento de una región y se dividen los distintos objetos que se encuentren en la imagen para colocarlos en clases distintas.

  - **Caso de uso 3.3: Filtrar Segmentos**

    Se eliminan las clases que contengan Pixeles blancos para solo dejar los caracteres de la placa.

- **Caso de Uso 4: Reconocer Caracteres**

  En este caso de uso se resuelve el problema por medio del algoritmo Backpropagation.

  - **Caso de Uso 4.1: Cargar Archivo Caracter**

Se carga una matriz con todos los recuerdos, es decir tanto las veintiséis (26) letras como los nueve (9) números a reconocer.

- **Caso de Uso 4.2: Aplicar Red Neuronal Backpropagation**
  Se aplica el algoritmo Backpropagation a cada uno de los segmentos de la placa para así determinar cual carácter se está analizando y reconocerlo mediante los recuerdos.

- **Caso de Uso 4.3: Generar solución**

  Aquí el sistema le envía al usuario la solución obtenida por la Red Neuronal Backpropagation.

  - **Caso de Uso 4.3.1: Mostrar Solución**
    Se muestra en pantalla los caracteres reconocidos por dicha Red
  - **Caso de Uso 4.3.2: Guardar Archivo Solución**
    Se guarda en un arreglo los caracteres que se reconocieron por el algoritmo Backpropagation.

3.6 Diagrama de Actividades

Son similares a los diagramas de flujo de otras metodologías Orientadas a Objetos. En realidad se corresponden con un caso especial de los diagramas de estado donde los estados son estados de acción y las transiciones vienen provocadas por la finalización de las acciones que tienen lugar en los estados de origen.

3.6.1 Diagrama de Actividades: Cargar Imagen

El usuario proporciona la imagen a la aplicación, luego se verifica si el formato en el que fue entregada dicha Imagen es el permitido si es permitido se muestra la imagen y de no ser así muestra un mensaje de error (***ver figura 3.3***).

3.6.2 Diagrama de Actividades: Localizar Placa

La imagen a tratar es representada mediante una matriz donde cada casilla va a estar cargada con el valor de un píxel en RGB, se procede a convertir la matriz RGB en escala de grises, es decir pasarla a una sola banda mediante el uso de una LUT (look up table) o tabla de consulta. Se aplican las operaciones morfológicas (dilatación, erosión, opening y closing) con la finalidad de limpiar los posibles ruidos (quitar los pixeles que no sean necesarios) en la imagen para poder localizar la placa y luego de encontrar la ubicación de la placa se procede a recortar ese espacio para su posterior procesamiento (ver figura 3.4).

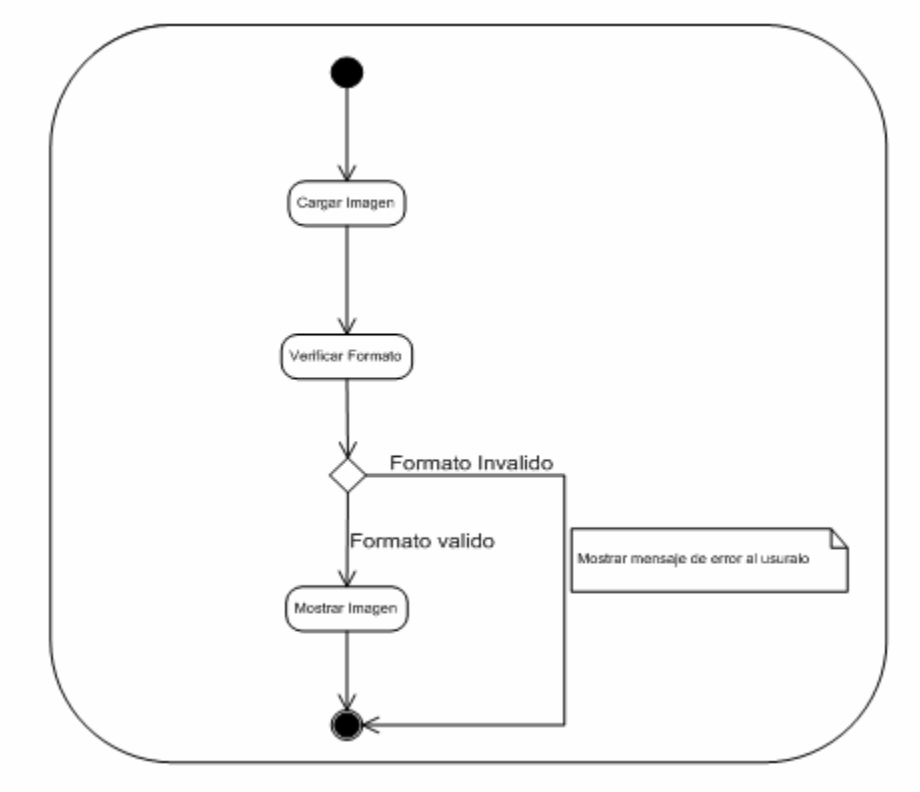


Figura 3.3 Diagrama de Actividades Cargar Imagen

3.6.3 Diagrama de Actividades Segmentar Caracteres

Se escala la imagen para facilitar la búsqueda de los caracteres de la placa, se aplica crecimiento de una región y se dividen los distintos objetos que se encuentren en la imagen para colocarlos en clases distintas para luego eliminar las clases que contengan Pixeles blancos y así obtener sólo los caracteres de la placa (***ver figura 3.5***).

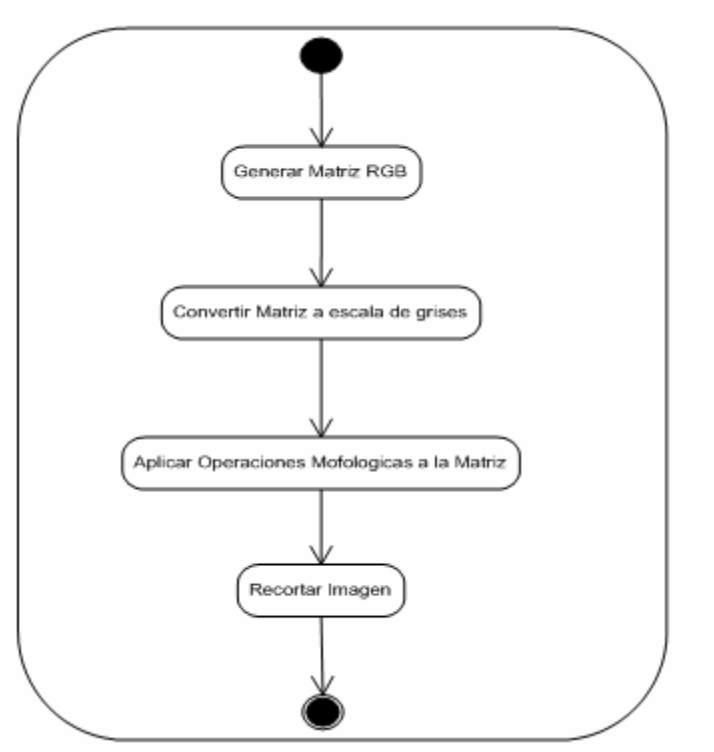


Figura 3.4 Diagrama de Actividades Localizar Placa

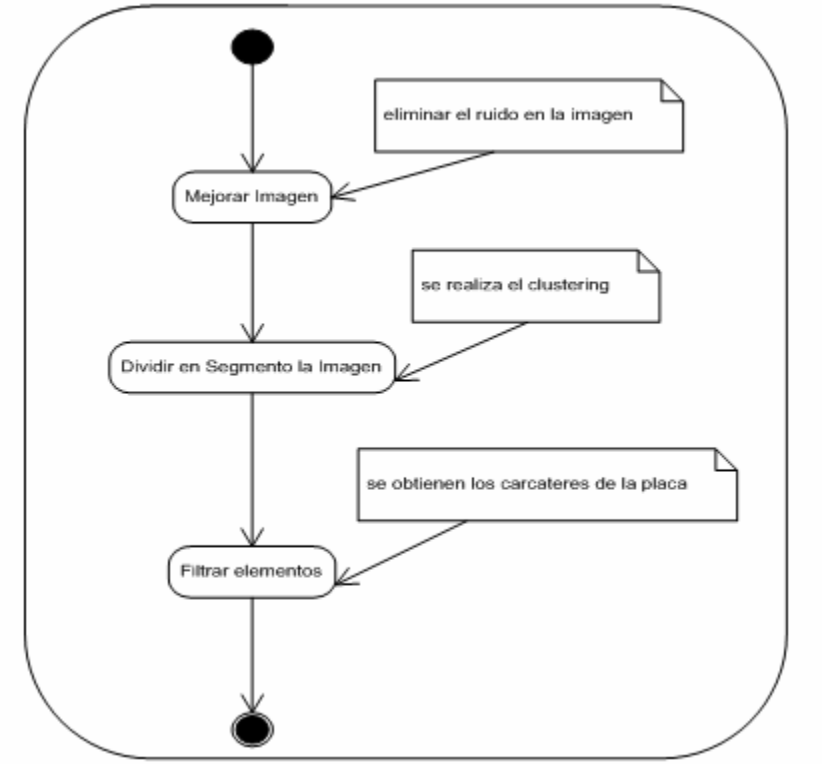


Figura 3.5 Diagrama de Actividades Segmentar Caracteres

3.6.4 Diagrama de Actividades Reconocer Caracteres

Se carga una matriz con todos los recuerdos, es decir tanto las veintiséis (26) letras como los nueve (9) números a reconocer luego se aplica la Red Neuronal de Hopfield con los caracteres que se obtuvieron de la placa uno por uno para así determinar cual carácter se está analizando y reconocerlo gracias a los recuerdos previamente cargados, al terminar de analizar cada una de los caracteres la Red envía al usuario la solución obtenida y se muestra en pantalla los caracteres reconocidos por ésta y por último se guardan estos en un arreglo.

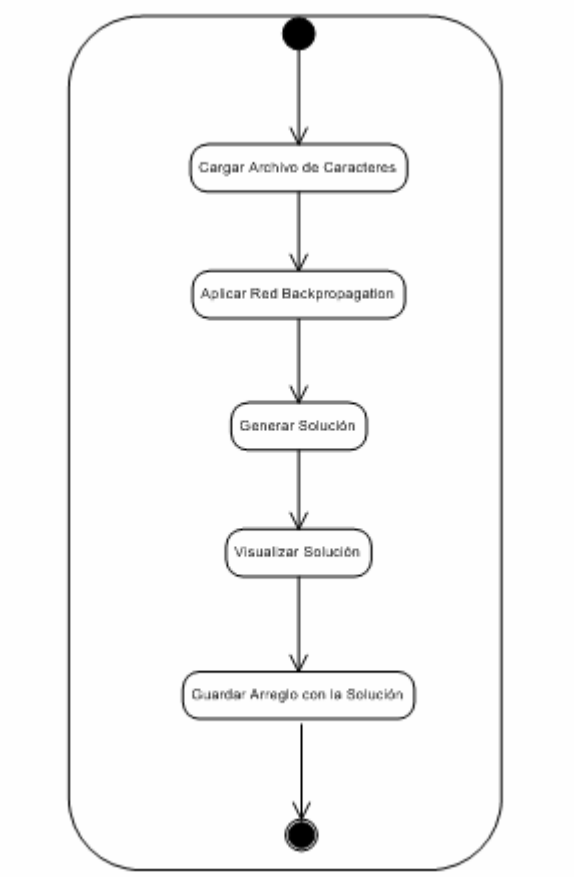


Figura 3.5 Diagrama de Actividades Reconocer Caracteres

CAPITULO 4: FASE DE DISEÑO

4.1 Diagrama de Clase de Diseño

En este diagrama se describe la estructura del sistema SRP, mostrando sus clases, atributos y las relaciones entre ellos. Los diagramas de clases de diseño son utilizados durante el proceso de análisis y diseño de los sistemas, donde se crea el diseño conceptual de la información que se manejará en el sistema, y los componentes que se encargaran del funcionamiento y la relación entre uno y otro.

En este diagrama se especifican el nombre de la clase tal como se encuentra en la aplicación, el nombre y tipo de los atributos, donde se indicará con signo más (+) si es público y con un signo menos (-) si es privado, luego se nombran los métodos de la clase, indicando también si son públicos o privados. Así como también se debe indicar la relación entre las clases.

En la figura 4.1 se muestra un ejemplo del formato, y en la figura 4.2 se muestra el diagrama de clase de diseño del sistema SRP.

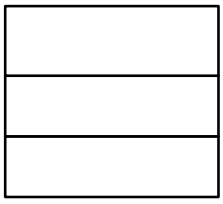


Figura 4.1 Estructura del Diagrama clase de diseño

4.2 Diagrama de Secuencia

Éste es un diagrama de interacción UML. Estos diagramas muestran la secuencia de mensajes que se van lanzando los objetos implicados en una determinada operación del programa.

Dentro del diagrama los objetos se alinean en el eje X respetando su orden de aparición. En el eje Y se van mostrando los mensajes que se envían, también respetando su orden temporal. Cada objeto tiene una línea de vida donde se sitúa su foco de control.

Con este sencillo esquema podemos visualizar la comunicación y sincronización bajo un estricto orden temporal de los objetos implicados en las distintas funcionalidades de un sistema.

A continuación se muestran los diagramas de secuencia correspondientes a los casos de uso fundamentales para el sistema SRP.

4.2.1 Diagrama de Secuencia para el Caso de Uso Cargar Imagen

En el diagrama de secuencia del caso de uso Cargar Imagen, el proceso de comunicación de los objetos que interactúan, se establece a través de los mensajes que se envían entre ellos. Dicho proceso se describe de la siguiente manera: (Ver figura 4.3).

1. El actor Usuario realiza un llamado al objeto **Interfaz** por medio del mensaje *Solicitud Cargar Imagen.*

2. El objeto **Interfaz** activa al objeto **Gestor** por medio del mensaje *Leer archivo imagen*.

3. El actor Archivo Imagen le envía al objeto **Gestor** los Datos de la Imagen.

4. El objeto **Gestor** activa al objeto **FileFillter** por medio del mensaje *Validar Imagen*.

5. El objeto **FileFillter** se hace un llamado a sí mismo con el mensaje *Verificar Imagen*.

6. El objeto **FileFillter** activa a **Gestor** por medio del mensaje *Enviar Imagen*.

7. El objeto **Gestor** activa al objeto **Interfaz** por medio del mensaje *Imagen*.

8. El objeto **Interfaz** le envía el mensaje *Mostrar Imagen* al Usuario.

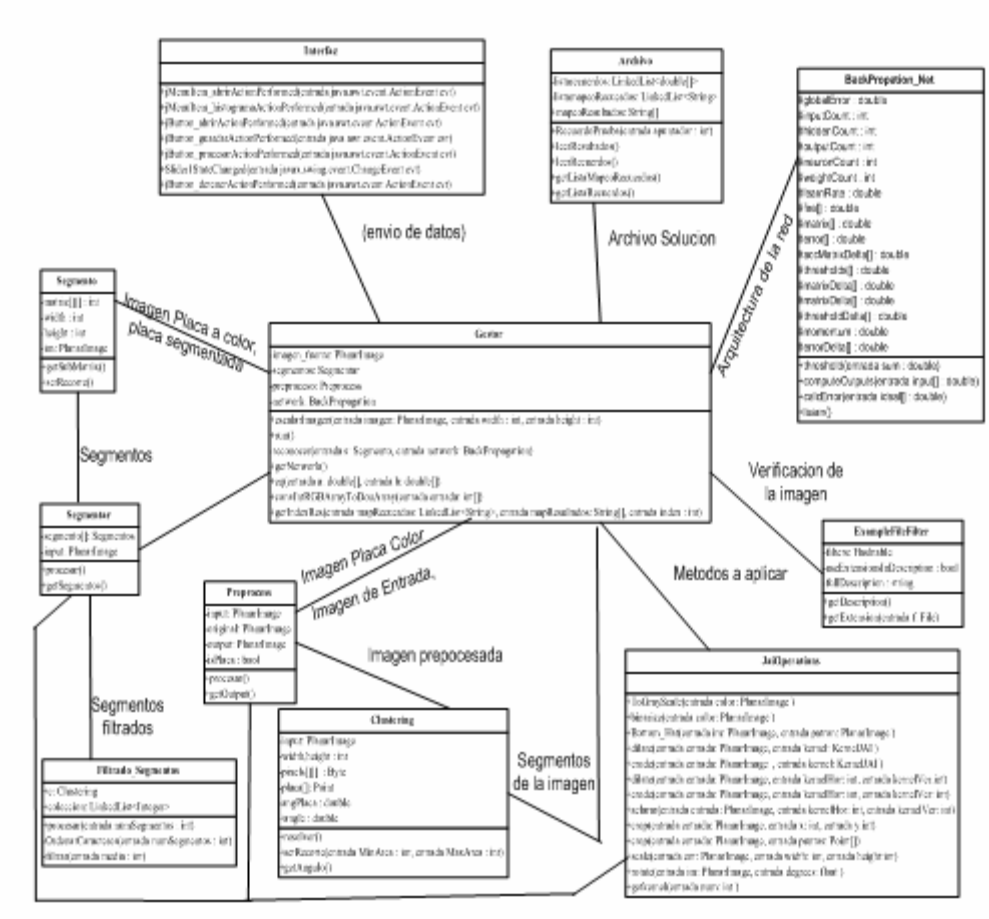


Figura 4.2 Diagrama de Clase de Diseño del sistema SRP

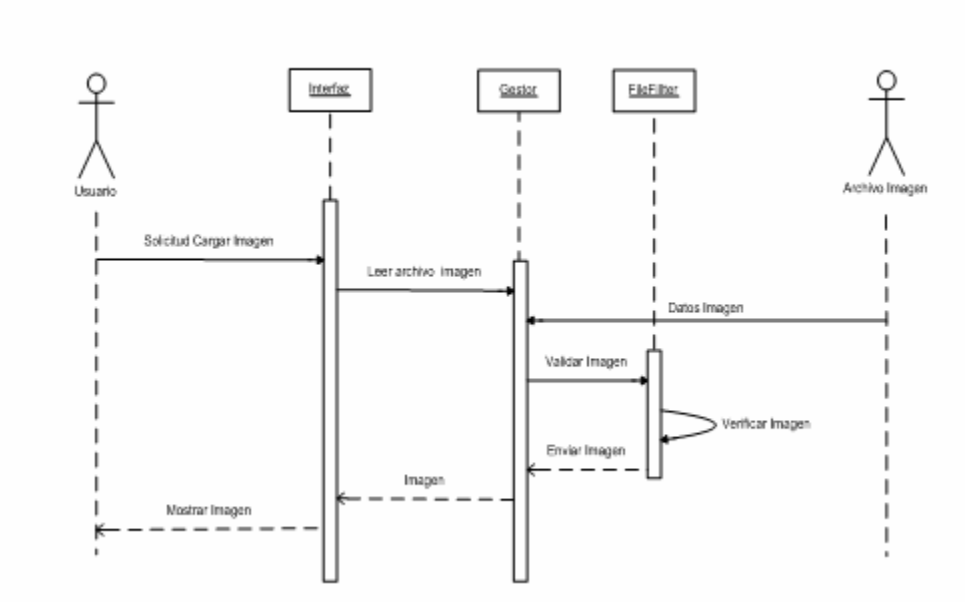


Figura 4.3 Diagrama de Secuencia para el Caso de Uso Cargar Imagen

4.2.2 Diagrama de Secuencia para el Caso de Uso Localizar Placa

En el diagrama de secuencia del caso de uso Localizar Placa, el proceso de comunicación de los objetos que interactúan, se establece a través de los mensajes que se envían entre ellos. Dicho proceso se describe de la siguiente manera: (Ver figura 4.4.).

1. El actor Usuario realiza un llamado al objeto **Interfaz** por medio del mensaje *Solicitar Localizar Placa*.

2. El objeto **Interfaz** activa al objeto **Gestor** por medio del mensaje *Iniciar Proceso de Localización*.

3. El actor **Imagen** realiza un llamado al objeto **Gestor** por medio del mensaje *Imagen*.

4. El objeto **Gestor** activa al objeto **Preproceso** por medio del mensaje *Enviar Imagen*.

5. El objeto **Preproceso** se hace un llamado a sí mismo con el mensaje *Procesar Imagen*.

6. El objeto **Preproceso** activa al objeto **Clustering** por medio del mensaje *Enviar Imagen Procesada*.

7. El objeto **Clustering** le envía el mensaje *Enviar Subdivision de la Imagen* al objeto **Preproceso**.

8. El objeto **Preproceso** se hace un llamado a sí mismo con el mensaje *Recortar Placa Localizada*.

9. El objeto **Preproceso** le envía al Objeto **Gestor** la placa localizada.

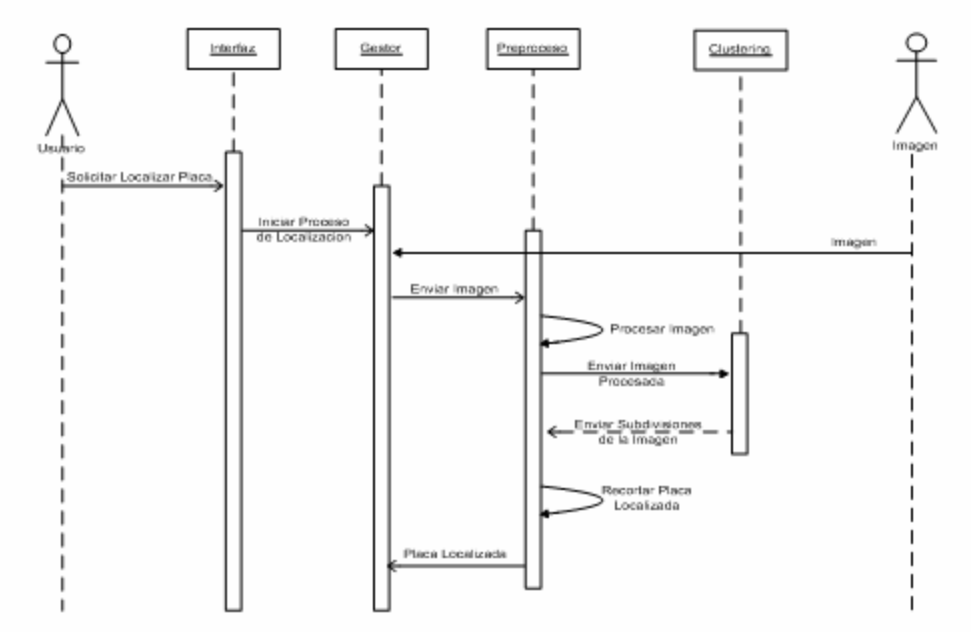


Figura 4.4 Diagrama de Secuencia para el Caso de Localizar Placa

4.2.3 Diagrama de Secuencia para el Caso de Uso Segmentar Caracteres

En el diagrama de secuencia del caso de uso Segmentar Carcateres, el proceso de comunicación de los objetos que interactúan, se establece a través de los mensajes que se envían entre ellos. Dicho proceso se describe de la siguiente manera: (Ver figura 4.5.).

1. El actor Imagen Placa activa al **Gestor** mediante el mensaje *Datos Imagen*.

2. El objeto **Gestor** activa al objeto **Segmentar** enviándole la placa localizada.

3. El objeto **Segmentar** se hace un llamado a sí mismo con el mensaje *Mejorar Imagen de la Placa*.

4. El objeto **Segmentar** activa al objeto **Clustering** enviándole la imagen mejorada de la placa.

5. El objeto **Clustering** se hace un llamado a sí mismo con el mensaje *dividir en Segmentos la Imagen de la Placa*.

6. El objeto **Clustering** activa al objeto **Filtrado_segmentos** enviándole la placa localizada.

7. El objeto **Filtrado_segmentos** se hace un llamado a sí mismo con el mensaje *Filtrar Segmentos*.

8. El objeto **Filtrado_segmentos** envía al objeto **Segmentos** los segmentos filtrados.

9. El objeto **Segmentar** envía uno a uno los segmentos filtrados al objeto **Segmentos**.

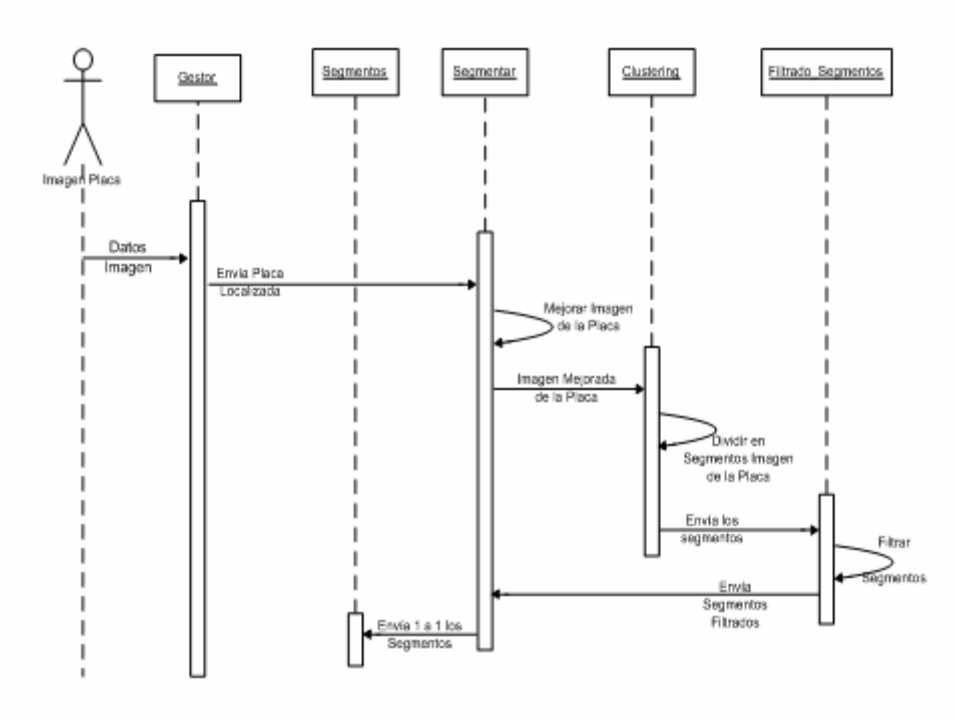


Figura 4.5 Diagrama de Secuencia para el Caso Segmentar Caracteres

4.2.4 Diagrama de Secuencia para el Caso de Uso Reconocer Caracteres

En el diagrama de secuencia del caso de uso Generar Población Inicial, el proceso de comunicación de los objetos que interactúan, se establece a través de los mensajes que se envían entre ellos. Dicho proceso se describe de la siguiente manera (*Ver* figura 4.6):

1. El actor **Archivo Recuerdos** activa al objeto **BackPropation** a través dell mensaje *Enviar recuerdos*.

2. El objeto **BackPropation** se hace un llamado a sí mismo para entrenar los pesos de la red BackPropation.

3. El objeto **Gestor** envía segmento a segmento los caracteres de la placa al objeto **BackPropagation**.

4. El objeto **BackPropagation** se hace un llamado a si mismo para reconocer caracter por caracter.

5. El objeto **BackPropagation** envía los caracteres reconocidos al objeto al objteto **Gestor**.

6. El objeto **Gestor** activa al objeto **Interfaz** mediante el mensaje *Mostrar Solución*.

7. El objeto **Interfaz** muestra la placa reconocida al actor Usuario.

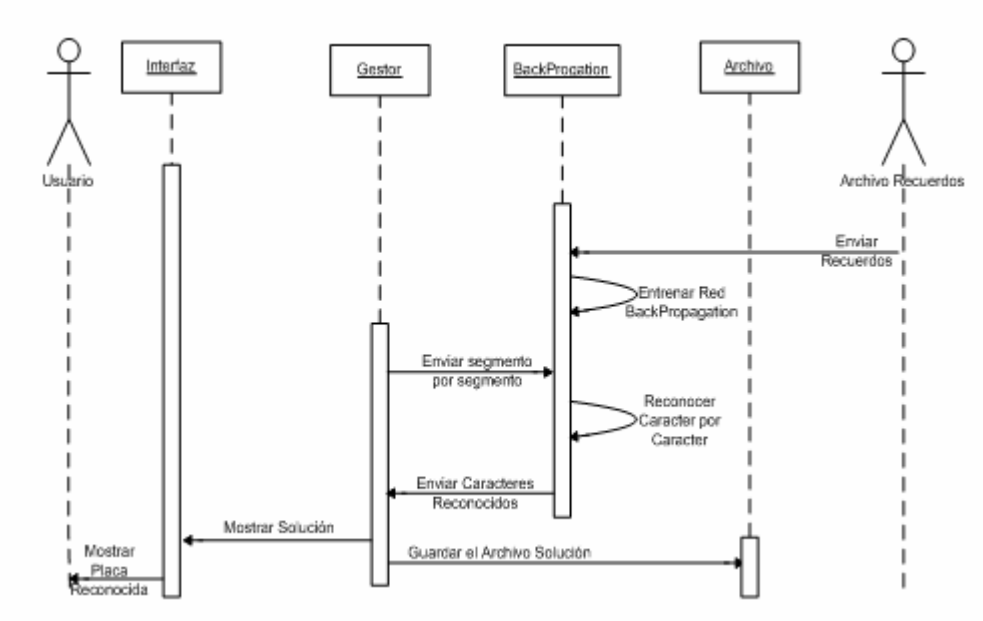


Figura 4.6 Diagrama de Secuencia para el Caso Reconocer Caracteres

4.3 Diagrama de Paquetes

Los diagramas de paquetes se usan para reflejar la organización de paquetes y sus elementos. Los elementos contenidos en un paquete comparten el mismo espacio de nombre, el hecho de compartir espacios de nombres requiere que los elementos contenidos en un espacio de nombre específico tengan nombres únicos. Los paquetes se pueden construir para representar relaciones tanto físicas como lógicas.

En la figura 4.7 se muestra el paquete FileFilter está relacionado con el caso de uso Cargar Imagen y contiene el archivo .java ExampleFileFilter.

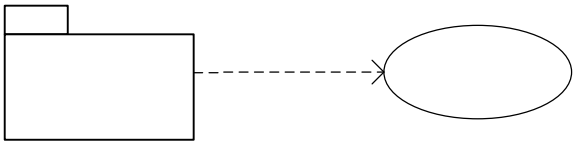


Figura 4.7 Diagrama de Paquete FileFilter

En la figura 4.8 se muestra el paquete Preprocesar el cual contiene los archivos .java Clustering, JaiOperations, Preprocess y está relacionado con el caso de uso Localizar Placa.

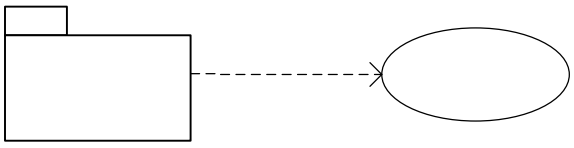


Figura 4.8 Diagrama de Paquete Preprocesar

En la figura 4.9 se muestra el paquete Segmentacion_Caracteres el cual contiene los archivos .java Filtrado_Segmentos, ImageThiner, Segmentar, Segmento y está relacionado con el caso de uso Segmentar caracteres.

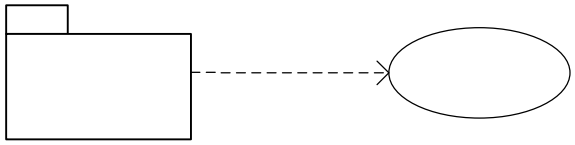


Figura 4.9 Diagrama de Paquete Segmentacion_Caracteres

En la figura 4.10 se muestra el paquete BackPropagation_Net el cual contiene los archivos .java Archivo, BackPropagation, Recuerdos, Resultados y está relacionado con el caso de uso Reconocer Caracteres.

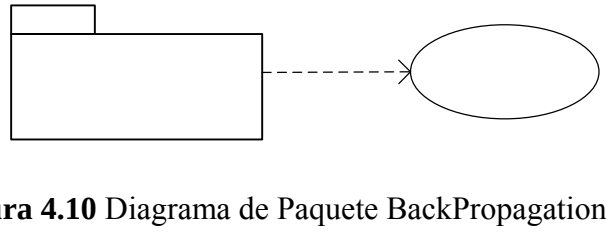


Figura 4.10 Diagrama de Paquete BackPropagation_Net

En la figura 4.11 se muestra el paquete Reconocedor el cual contiene los archivos .java Gestor, Interfaz, Output y está relacionado con todos los casos de uso.

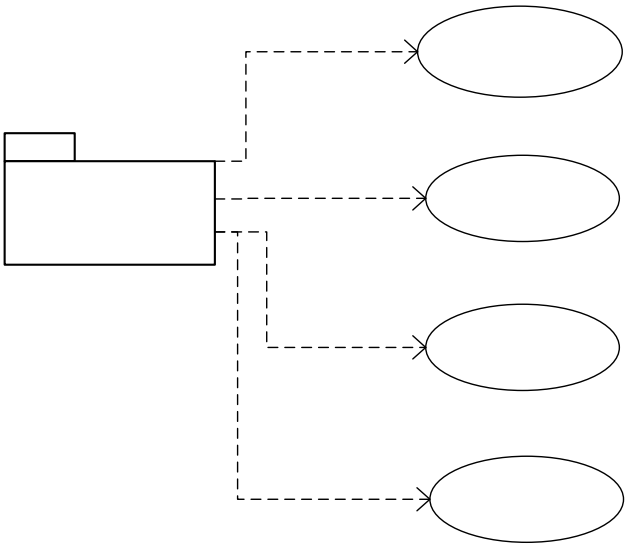


Figura 4.11 paquete Reconocedor el cual contiene los archivos .java

En la figura 4.12 se muestra el paquete Imágenes, donde se encuentran todas las imágenes utilizadas en la aplicación.

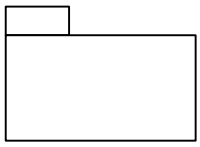


Figura 4.12 Diagrama de Paquete Imágenes

En la figura 4.13 se muestra la relación entre los paquetes del sistema.

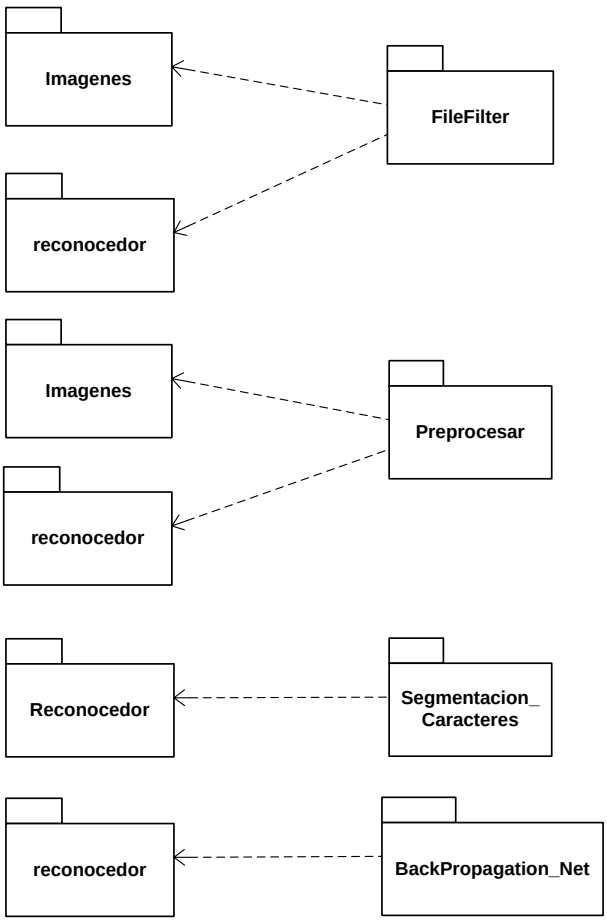


Figura 4.13 Relación entre los paquetes

4.4 Diagrama de Capas

En este diagrama se muestra la distribución en cuatro capas de los paquetes que permiten el funcionamiento del sistema SRP, la cual se distribuye de la siguiente manera:

- **Capa Específica de la Aplicación**: aquí se encuentran los paquetes internos del sistema como son el de Imágenes, FileFilter, Preprocesar, Segmentacion_Caracteres, BackPropagation_Net, Reconocedor.

- **Capa General de la Aplicación**: aquí es donde se muestra el paquete de la aplicación como tal.

- **Capa Intermedia**: en esta capa se encuentra el JRE (Java Runtime Environment) y la JAI (Java Advanced Image) la cual contiene un set de interfaces orientadas a objetos q facilitan la manipulación de imágenes para la programación a un alto nivel.

- **Capa de Software:** aquí se encuentra el sistema operativo donde puede funcionar la aplicación. El cual puede ser Windows o Linux.

En la figura 4.14 se muestra el diagrama de capas del sistema SRP.

Figura 4.14 Diagrama de Capas del sistema SRP

4.5 Técnica de Desarrollo de Sistemas de Objetos (TDSO)

A continuación se mostrará la especificación formal de las clases, los métodos y el universo de las clases del sistema SRP:

4.5.1 Descripción del Universo de Clases

Tabla 1 Definición del Universo de Clases del sistema SRP

10/10/2008		Versión 1.0
Universo de Clases del Sistema Reconocedor de Placa (SRP)		
{ Colección de clases que son requisitos para implantar el SRP }		
1	Preprocess(PlanarImage entrada)	Se procede a localizar y recortar la placa de la imagen original, para su posterior procesamiento.
2	JaiOperations()	Contiene todas las Operaciones Morfológicas necesitadas en el sistema y sus aplicaciones.
3	Clustering(PlanarImage entrada)	Realiza un algoritmo de "region growing" para segmentar una imagen en blanco y negro en regiones o clases, además de reconocer si la placa se encuentra en la imagen.
4	ExampleFileFilter()	Filtra las extensiones de los archivos que se especiquen, para permitir que sólo los archivos de imágenes sean tomados en cuenta.

Tabla 2 Definición del Universo de Clases del sistema SRP

10/10/2008		Versión 1.0
Universo de Clases del Sistema Reconocedor de Placa (SRP)		
{ Colección de clases que son requisitos para implantar el SRP }		
5	Filtrado_Segmentos(Clustering c)	Filtra los segmentos irrelevantes de la imagen de la placa, y ordena los caracteres

		de la placa.
6	Segmentar(PlanarImage input)	Convierte la imagen de entrada, en este caso la placa, en un conjunto de segmentos, donde cada segmento es un carácter de la placa.
7	Segmento(Clustering c, entero apuntador, entero[][] labels)	Clase que contiene un segmento o carácter de la placa, para su posterior reconocimiento.
8	Archivo ( )	Realiza las operaciones de lectura y escritura de Archivos.
9	BackPropagation (entero inputCount, entero hiddenCount, entero outputCount, doble learnRate, doble momentum)	Red Neuronal de BackPropagation, su entrenamiento y aplicación.
10	Gestor ( )	Se realizan todas las operaciones necesarias en el programa. Provee una interfaz entre el usuario y los métodos del sistema.
11	Interfaz ( )	Interfaz de Usuario.

4.5.2 Especificación Formal de las Clases

Tabla 3 Especificación Formal de la Clase Preprocess

10/10/2008	Versión 1.0
1 clase Preprocess (PlanarImage entrada)	
{ Proceso para localizar y recortar la placa }	

1 2	Especificación de atributos: original, escala_grises, substracción, bottom_hat, closing, opening, clearing, placa_color: Planar Imagen. tProceso: long. isPlaca: booleano. puntos: punto[]. Especificación sintáctica: getPuntos (PlanarImagen imagen) procesar ( ) Declaraciones: Preprocess X	original: Imagen original o de entrada. escala_grises: Imagen en escala de grises. substracción: Imagen resta entre imagen original y escalada a grises. bottom_hat: Imagen binarizada luego de la resta. closing: Imagen que se le aplicó la operación morfológica closing. opening: Imagen que se le aplicó la operación morfológica opening. clearing: Imagen aclarada para facilitar el reconocimiento de la placa. placa_color: Imagen de la placa a color. tproceso: Tiempo que se toma la localización de la placa. isplaca: Determina si existe la placa en la imagen. puntos: Arreglo de puntos, que representan las 4 esquinas de la placa.

Tabla 4 Especificación Formal de la Clase Clustering

10/10/2008	Versión 1.0
3 clase Clustering (PlanarImage entrada)	
{ Segmentación de imágenes en blanco y negro en regiones o clases }	

	Especificación de atributos: input: PlanarImage. width, height, numberOfRegions: entero. pixels: byte[][] labels: entero[][]. position, tiempoEjecucion: long. count: Mapa. placa: punto[]. angplaca: doble. Especificación sintáctica: 1 resolver ( ) 2 setRecorte (entero MinArea, entero MaxArea) Declaraciones: Clustering X	input: Imagen de entrada. width: Ancho de la Imagen. height: Alto de la Imagen. pixels: Matriz que contiene la información de cada pixel de la imagen. labels: Matriz que contiene la información de cada pixel en forma binaria. position: La posición, numero de pasos que ha hecho el algoritmo el algoritmo. tiempoEjecución: El tiempo de Ejecución del proceso de segmentación. count: Mapeo de las regiones con sus respectivos números de pixeles por region. placa: Arreglo de los puntos de las esquinas de la placa. angPlaca: El ángulo de la placa.

Tabla 5 Especificación Formal de la Clase Filtrado_Segmentos

10/10/2008	Versión 1.0
5 clase Filtrado_Segmentos (Clustering c)	

{ Filtra los segmentos irrelevantes de la imagen de la placa }		
	Especificación de atributos: c: Clustering. colección: Lista Enlazada. media, puntoCorte, distorsion: entero. coordSegmentos: entero[]. Especificación sintáctica:	c: La segmentación en regiones realizada previamente. colección: Colección de todos los segmentos en la imagen de la placa. media: La media estadística del número de pixeles entre todos los segmentos de la imagen. puntoCorte: Es el número del segmento a eliminar en la colección de segmentos.
1	procesar (entero numSegmentos)	distorsion: Se utiliza para saber qué tan lejano está el número de pixeles de un segmento de la media, dependiendo su lejanía a la media será borrado o no de la colección.
2	OrdenarCaracteres (entero numSegmentos)	
3	getMinX (entero apuntador)	
4	getMedia ( )	
5	filtrar (entero media)	coordSegmentos: La coordenada de cada segmento para ser ordenado de acuerdo a su posición en la placa.
6	validar (entero numRegion, entero[][] labels, byte[][] pixels, Punto p)	
	Declaraciones: Filtrado_Segmentos X	

Tabla 6 Especificación Formal de la Clase Segmentar

10/10/2008		Versión 1.0
	6 clase Segmentar (PlanarImage input) { Convierte la imagen de entrada, en un conjunto de segmentos }	
1	Especificación de atributos: input, placa_binaria: PlanarImage. segmentos: Segmento. Especificación sintáctica: procesar ( ) Declaraciones: Segmentar X	input: Imagen de la placa. placa_binaria: Imagen de la placa binarizada. segmentos: Conjunto de segmentos en la imagen de la placa, estos segmentos representan cada uno de los caracteres en la placa.

Tabla 7 Especificación Formal de la Clase Segmento

10/10/2008		Versión 1.0
	7 clase Segmento(Clustering c, entero apuntador, entero[][] labels) { contiene un segmento o carácter de la placa, para su posterior reconocimiento }	
1 2	Especificación de atributos: width, height: entero. matriz, pixels: entero[][]. imagen_segmentada: PlanarImage. Especificación sintáctica: getSubMatrix ( ) getRecorte ( ) Declaraciones: Segmento X	width: El ancho del segmento: height: El alto del segmento. matriz: Matriz de los elementos del segmento. pixels: Matriz de los elementos del segmento en formato RGB.

Tabla 8 Especificación Formal de la Clase BackPropagation

10/10/2008		Versión 1.0
9 clase BackPropagation (entero inputCount, entero hiddenCount, entero outputCount, doble learnRate, doble momentum) { Red Neuronal de BackPropagation, su entrenamiento y aplicación }		
	Especificación de atributos: globalError, momentum: doble. inputCount, hiddenCount, outputCount, neuronCount, weightCount, learnRate: entero. fire, matrix, error, accMatrixDelta, thresholds, matrixDelta, accThresholdDelta, thresholdDelta, errorDelta: doble[]. **Especificación sintáctica:**	**globalError:** El error global del entrenamiento. **inputCount:** El número de neuronas de entrada. **hiddenCount:** El número de neuronas en la capa escondida, **outputCount:** El número de neuronas de salida. **neuronCount:** El número total de neuronas en la red. **wieghtCount:** El número de pesos en la red.
1 2 3 4	computeOutputs ( ) calcError ( ) learn ( ) reset ( )	**learnRate:** La tasa de aprendizaje. **fire:** La salida de los niveles de las capas en la red. **matrix:** La matriz de peso. **error:** Los errores desde el último cálculo de aprendizaje.
	Declaraciones: BackPropagation X	**accMatrixDelta:** Acumula los delta de la matriz de peso para el entrenamiento. **thresholds:** Los valores de umbral, conjuntamente con la matriz de peso, son la memoria de la red. **matrixDelta:** Los cambios que podrían ser aplicadosa la matriz de peso. **accThresholdDelta:** La acumulación de los deltas de umbrales. **thresholdDelta:** Arreglo que contiene los deltas de los umbrales. **momentum:** El ímpetu para el entrenamiento. **errorDelta:** Los cambios en los errores.

Tabla 9 Especificación Formal de la Clase Gestor

<table>
<tr><td colspan="3">10/10/2008 Versión 1.0

10 clase Gestor ( )

{ Se realizan todas las operaciones necesarias en el programa. Provee una interfaz entre el usuario y los métodos del sistema }</td></tr>
<tr><td></td><td>Especificación de atributos:

imagen_fuente: PlanarImage.

filename: Cadena de caracteres.

file, auxiliar: File.

segmentos: Segmentar.

preproceso: Preprocess.

network: BackPropagation.

caracteres: Cadena de caracteres[].

hilo: Hilo.

Especificación sintáctica:

run ( )

reconocer (Segmento s)

eq(doble[] a, doble[] b)

convIntRGBArrayToDouArray(entero[] entrada)

Declaraciones:

Gestor X</td><td>Imagen_fuente: La Imagen de entrada

filename: El nombre del archivo de entrada.

file: Archivo de entrada.

auxiliar: Auxiliar al archivo de entrada.

segmentos: Objeto que procesará en segmentos la imagen.

preproceso: Objeto que preprocesará la imagen de entrada.

Red Neuronal de network: backPropagation.

caracteres: Caracteres de la Placa.

hilo: Proceso del sistema.</td></tr>
</table>

4.5.3 Especificación Formal de Métodos

4.5.3.1 Métodos Clase Preprocess

Tabla 10 Especificación Formal del Método getPuntos

10/10/2009		versión 1.0
	1,1(Metodo,Privado) getPuntos() {obtener puntos}	
{Pre: PlanarImage im **}**		**{Pos:** Point[]**}**
1 2 3 4 5 6 7 8 9 10 11	puntos = nuevo Point[4] Para i=1 hasta 4 hacer puntos[i]=nuevo Point() fPara puntos[0].x=20 puntos[0].y=20 puntos[1].x=20 puntos[1].y= im.getHeight()-20 puntos[2].x=im.getWidth()-20 puntos[2].y=20 puntos[3].x=im.getWidth()-20 puntos[3].y=im.getHeight()-20 Retornar puntos;	**Puntos:** variable del tipo Point

4.5.3.2 Métodos Clase Clustering

Tabla 11 Especificación Formal del Método resolver

10/10/2009		versión 1.0
	3,1 (Metodo,Público) resolver() {escanear la imagen horizontal y verticalmente para reconocer los objetos que haya en ellos. la imagen debe ser binaria}	
{Pre: }		**{Pos:}**
1	numberOfRegions = 0	**numberOfRegions:** numero de regiones
2	Stack<Point> mustDo = new Stack<Point>()	**mustDo:** pila que contiene los pixeles vecinos iguales al pixel revisado
	Para h=1 hasta height hacer	
	Para w=0 hasta width hacer	
3	Position=position + 1	**Position:** numero de pasos que ha hecho el algoritmo
	Si labels[w][h] es menor que 0 entonces	
4	numberOfRegions= numberOfRegions+1	**Point:** variable tipo point
5	mustDo.add(new Point(w,h))	
6	labels[w][h] = numberOfRegions	**Labels:** matriz que contiene las regiones de la imagen
7	count.put(numberOfRegions,1)	
	fSi	
	Mientras mustDo.size() mayor que 0 hacer	
8	Point thisPoint = mustDo.get(0)	
9	mustDo.remove(0)	
	Para th =-1 hasta th menor o igual a 1 hacer	
10	Para tw=-1 hasta tw menor e igual a 1	
11	hacer	
	int rx = thisPoint.x+tw	
	int ry = thisPoint.y+th	

Tabla 12 Especificación Formal del Método resolver

<table>
<tr><td colspan="3">10/10/2009 versión 1.0

3,1 (Metodo,Público)

resolver()

{escanear la imagen horizontal y verticalmente para reconocer los objetos que haya en ellos. la imagen debe ser binaria}

{Pre: } {Pos:}</td></tr>
<tr><td>12

13

14

15

16</td><td>si rx < 0 ó ry < 0 ó ry>=height ó rx>=width entonces

continuar

si labels[rx][ry] < 0 entonces

si pixels[rx][ry] igual a pixels[thisPoint.x][thisPoint.y] entonces

mustDo.add(new Point(rx,ry))

labels[rx][ry] = numberOfRegions

count.put(numberOfRegions, count.get(numberOfRegions)+1)

fSi

fSi

fSi

fPara

position = width*height</td><td>numberOfRegions: numero de regiones

mustDo: pila que contiene los pixeles vecinos iguales al pixel revisado

Position: numero de pasos que ha hecho el algoritmo

Point: variable tipo point

Labels: matriz que contiene las regiones de la imagen</td></tr>
</table>

4.5.3.3 Métodos Clase Filtrado_Segmentos

Tabla 13 Especificación Formal del Método procesar

10/10/2009		Versión 1.0
	5,1(Método, Público) procesar (entero numSegmentos) { Solución Heurística para filtrar los segmentos necesarios que van a ser los caracteres a buscar} **{Pre:** Número de Segmentos a procesar**}**	**{Pos:** No retorna**}**
1 2 3	entero media = getMedia() hacer filtrar(media); mientras coleccion.size()>numSegmentos	**media:** Media estadística del número de pixeles de todos los segmentos en la imagen de la placa. **colección**: Lista de todos los segmentos en la imagen de la placa.

CAPITULO 5 IMPLEMENTACIÓN Y PRUEBAS

5.1. Diagrama de Componente

Un diagrama de componentes representa como un sistema de software es dividido en componentes y muestra las dependencias entre estos componentes. Los componentes físicos incluyen archivos, cabeceras, librerías compartidas, módulos, ejecutables, o paquetes. Los diagramas de Componentes prevalecen en el campo de la arquitectura de software pero pueden ser usados para modelar y documentar cualquier arquitectura de sistema.

Los diagramas de componentes describen los elementos físicos del sistema y sus relaciones, también muestran las opciones de realización incluyendo código fuente, binario y ejecutable. Pueden ser simples archivos, paquetes, bibliotecas cargadas dinámicamente, etc. [4]

La representación gráfica es la siguiente:

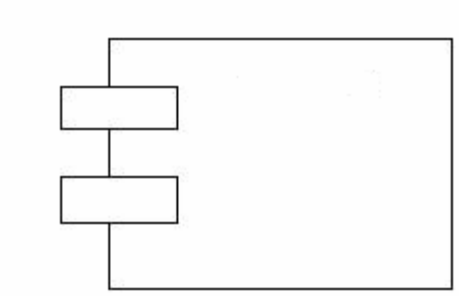


Figura 5.1 Representación Grafica del Diagrama de Componente

5.1.1 Diagramas de componentes del Sistema Reconocedor de Placas (SRP)

5.1.1.1 Diagrama de Componentes del Paquete filefilter

En la figura 5.2 se muestra el Diagrama de Componentes del paquete filefilter

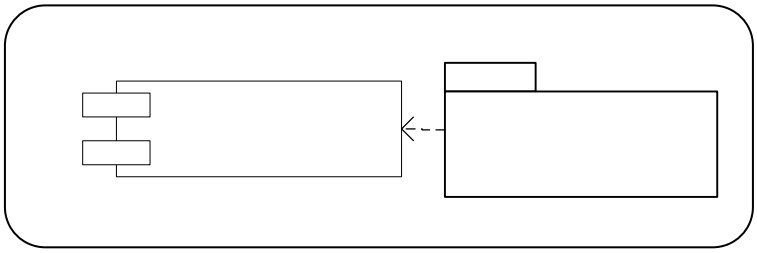


Figura 5.2 Diagrama de componente del paquete filefilter

5.1.1.2 Diagrama de Componentes del Paquete Preprocesar

En la figura 5.3 se muestra el Diagrama de Componentes del paquete Preprocesar

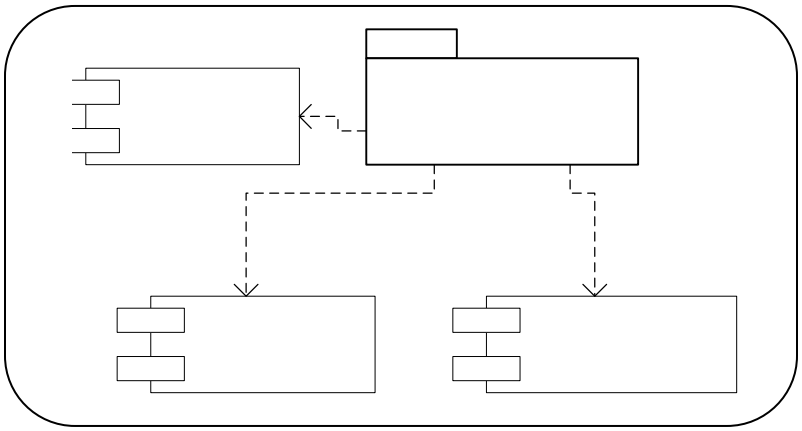


Figura 5.3 Diagrama de componente del paquete Preprocesar

5.1.1.3 Diagrama de Componentes del Paquete Segmentacion_Caracteres

En la figura 5.4 se muestra el Diagrama de Componentes del paquete Segmentacion_Caracteres

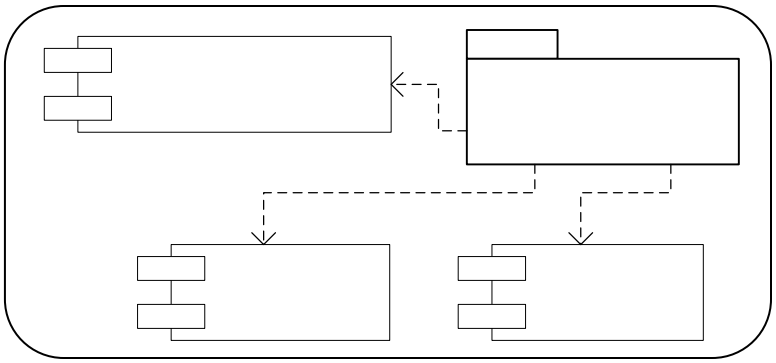


Figura 5.4 Diagrama de componente del paquete Segmentacion_Caracteres

5.1.1.4 Diagrama de Componentes del Paquete reconocedor

En la figura 5.5 se muestra el Diagrama de Componentes del paquete reconocedor

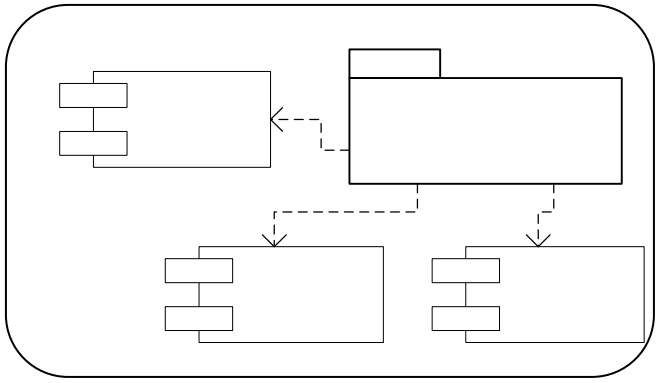


Figura 5.5 Diagrama de componente del paquete reconocedor

5.1.1.5 Diagrama de Componentes del Paquete BackPropagation_Net

En la figura 5.6 se muestra el Diagrama de Componentes del paquete BackPropagation_Net

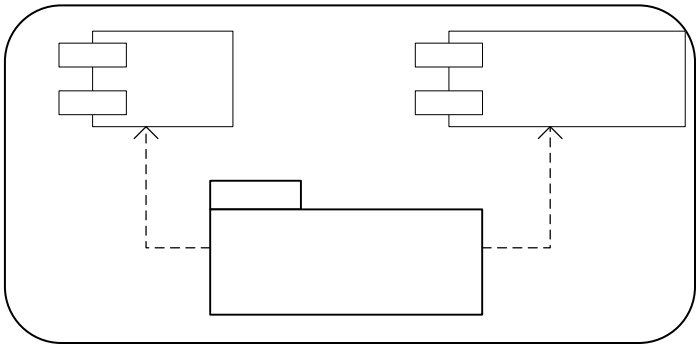


Figura 5.6 Diagrama de componente del paquete BackPropagation_Net

5.1.1.6 Relación del Componente Preprocess

En la figura 5.7 se muestra la relación entre el componente Preprocess y los c JaiOperations, Clustering, Segmentar, Gestor.

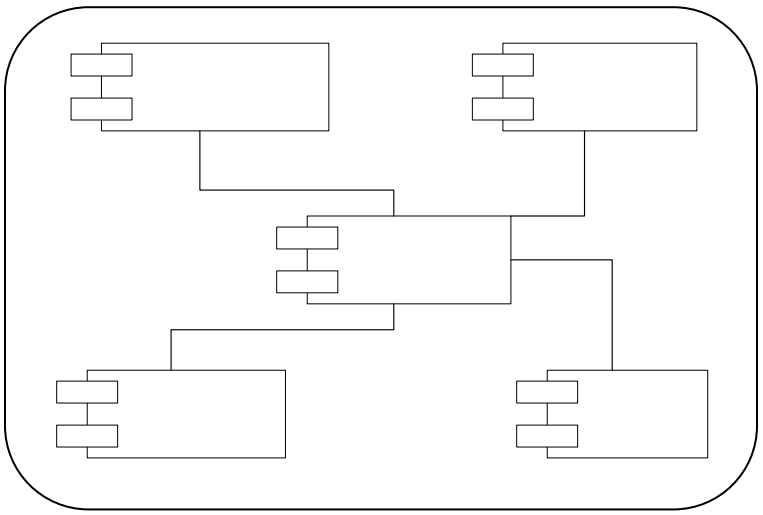


Figura 5.7 Relación entre el Componente Preprocess y los componentes JaiOperations, Clustering, Segmentar, Gestor

Código JAVA del Componente Preprocess

```
public void procesar(){

    escala_grises = JaiOperations.ToGrayScale(original);

    PlanarImage copia = JaiOperations.crop(original,300,100);
    escala_grises = JaiOperations.crop(escala_grises, 300,100);
    substraccion = JaiOperations.Substract(escala_grises, copia);

    bottom_hat = JaiOperations.Bottom_Hat(escala_grises, copia);

    closing = JaiOperations.dilate(bottom_hat, 140,1);
    closing = JaiOperations.erode(closing, 140,1);

    opening = JaiOperations.erode(closing, 1,30);
    opening = JaiOperations.dilate(opening, 1,30);

    clearing = JaiOperations.dilate(opening, 1,35);
    clearing = JaiOperations.dilate(clearing, 20,1);
    Clustering c = new Clustering(clearing);
    c.resolver();
    c.setRecorte(9750,12000);
   if(c.isPlaca()){
       isPlaca = true;
       placa_color =
        JaiOperations.crop(copia, c.getPlaca());

       placa_color =
        JaiOperations.rotate(
        placa_color,(float) c.getAngulo());

       placa_color =
        JaiOperations.crop(placa_color,
        getPuntos(placa_color) );
   }else
       JOptionPane.showMessageDialog(new JFrame(),
        "No existela placa en la foto","Error",1,
        new javax.swing.ImageIcon(getClass().getResource(
        "/Imagenes/Error Display.png")));
```

```
private Point[] getPuntos(PlanarImage im){
    Point [] puntos = new Point[4];
    for(int i=0;i<4;i++)
        puntos[i]=new Point();

    puntos[0].x=20;puntos[0].y=20;
    puntos[1].x=20;puntos[1].y=im.getHeight()-20;
    puntos[2].x=im.getWidth()-20;puntos[2].y=20;
    puntos[3].x=im.getWidth()-20;puntos[3].y=im.getHeight()-20;
    return puntos;
}
```

```
/**
 * Metodo que retorna la imagen original o la entrada
 * @return La imagen original
 */
public PlanarImage getOriginal(){
    return original;
}

public PlanarImage getGrises(){
    return escala_grises;
}

public PlanarImage getSubstraccion(){
    return substraccion;
}

public PlanarImage getBottomHat(){
    return bottom_hat;
}
```

```
public PlanarImage getClosing(){
    return closing;
}
```

```
public PlanarImage getOpening(){
    return opening;
}
```

```
public PlanarImage getClearing(){
    return clearing;
}
```

5.1.1.7 Relación del Componente Segmentar

En la figura 5.8 se muestra la relación entre el componente Segmentar y los otros componentes JaiOperations, Clustering, Segmentar, Gestor.

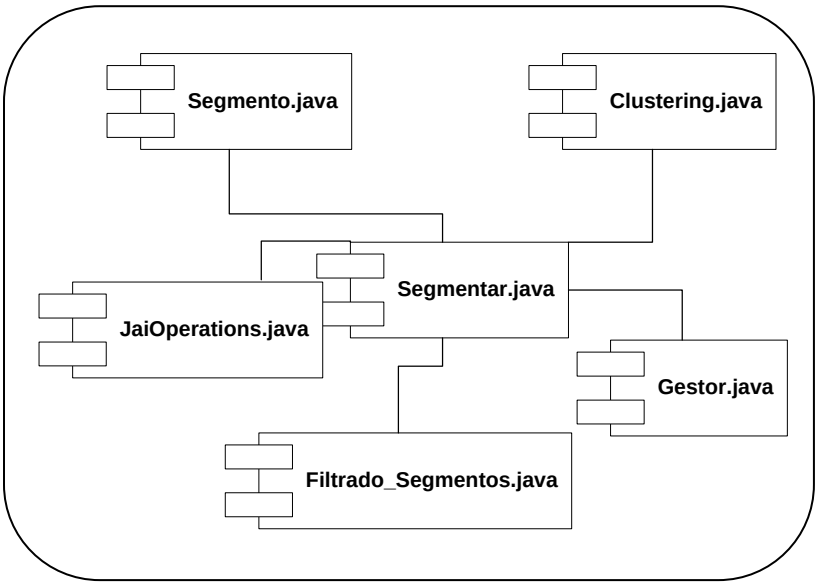


Figura 5.8 Relación entre el Componente Segmentar y los componentes JaiOperations, Clustering, Filtrado_Segmentos, Gestor, Segmento

Código JAVA del Componente Segmentar

```
/**
 * Convertir la imagen de entrada en un conjunto de segmentos
 * que serian las letras o caracteres de la placa
 */
```

```
public void procesar(){

    placa_binaria = JaiOperations.scale(input,750,380);
    placa_binaria = JaiOperations.binarize(placa_binaria);
    placa_binaria = JaiOperations.aclarar(placa_binaria,2,6);

     Clustering c = new Clustering(placa_binaria);
     c.resolver();



     Filtrado_Segmentos filtro = new Filtrado_Segmentos(c);
     filtro.procesar(7);//numero de caracteres de la placa
                        //Luego se verifica si en realidad
                        //existen 7 caracteres

     segmentos = new Segmento[filtro.getNumColeccion()];


     for(int i=0;i<filtro.getNumColeccion();i++){
         segmentos[i]= new Segmento(c,filtro.getPointer(i),
                 c.getLabels());


      public Segmento[] getSegmentos(){
          return segmentos;
      }

      public PlanarImage getPlacaBinaria(){
          return placa_binaria;
      }

      public int getSegmentosLength(){
          return segmentos.length;
      }
```

5.1.1.8 Relación del Componente Segmento

En la figura 5.9 se muestra la relación entre el componente Segmento y los otros componentes como son Segmentar y Gestor.

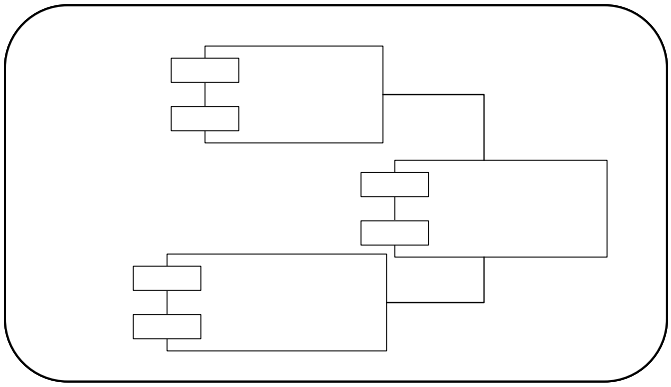


Figura 5.9 Relación entre el Componente Segmento y los componentes Segmentar y Gestor

Gestor.j

5.1.1.9 Relación del Componente Filtrado_Segmentos

En la figura 5.10 se muestra la relación entre el componente Filtrado_Segmentos y el componente Segmentar

Segmentar.j

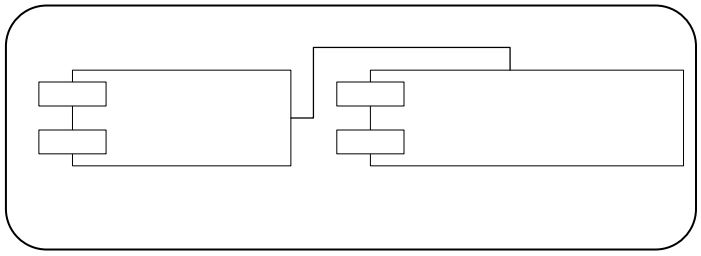


Figura 5.10 Relación entre el Componente Filtrado_Segmentos y componente Segmentar

5.1.1.10 Relación del Componente Clustering

En la figura 5.11 se muestra la relación entre el componente Clustering y los componentes Gestor y Preprocess.

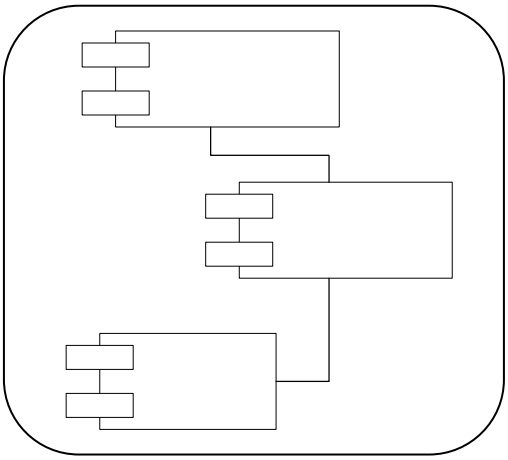


Figura 5.11 Relación entre el Componente Filtrado_Segmentos y los componentes Gestor y Preprocess

5.1.1.11 Relación del Componente JaiOperations

En la figura 5.12 se muestra la relación entre el componente JaiOperations y los componentes Segmentar, Gestor.

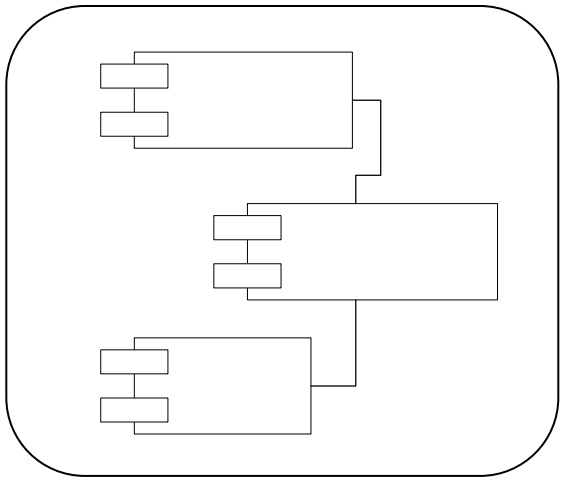


Figura 5.12 Relación entre el Componente JaiOperations y los componentes Gestor y Segmentar

Código JAVA del Componente JaiOperations

```
/**
 * Metodo para crear imagenes del tipo PlanarImage, a traves
 * de un archivo de entrada, q debio ser previamente validado
 * a ser una imagen posible
 * @param file Archivo de entrada
 * @return Imagen creada a partir del archivo de entrada
 */
public static PlanarImage createImage(File file){
    FileSeekableStream stream= null;
    PlanarImage image = null;
    try {

    stream = new FileSeekableStream(file.getAbsolutePath());
} catch (IOException ex) {
    Logger.getLogger(JaiOperations.class.getName()).
            log(Level.SEVERE, null, ex);
}
```

```
        image = JAI.create("stream", stream);

        return image;
    }


    /**
     * Metodo para crear imagenes a partir de un arreglo de entrada,
     * el arreglo debe estar pre-cargado con los valores RGB de la imagen
     * a crear
     * @param arreglo Arreglo con valores RGB
     * @return Imagen creada a partir del arreglo
     */
    public static PlanarImage createImage(int[][] arreglo,
            int width, int height){

        BufferedImage bi = new BufferedImage(width, height,
                BufferedImage.TYPE_INT_RGB);

            for(int y=0;y<height;y++){
                for(int x=0;x<width;x++){
                    bi.setRGB(x, y, arreglo[x][y]);
                }
            }
        return JaiOperations.scale(PlanarImage.wrapRenderedImage(bi),
                54,48);

    }
    /**
     * Metodo que sirve para clonar una imagen
     * @param imagen Imagen a clonar
     * @return Imagen clonada
     */
    public static PlanarImage cloneImage(PlanarImage imagen){
        return PlanarImage.wrapRenderedImage(imagen);
    }
    /**
     * Este Metodo convierte una imagen a color (indexada o no) a una imagen
     * de tonos grises
     * @param color Imagen de entrada
     * @return Imagen en escala de grises
     */
    public static PlanarImage ToGrayScale(PlanarImage color){
```

```
 if (color.getColorModel() instanceof IndexColorModel){

   IndexColorModel icm = (IndexColorModel)color.getColorModel();

   int mapSize = icm.getMapSize();

   byte[][] lutData = new byte[3][mapSize];

   icm.getReds(lutData[0]);
   icm.getGreens(lutData[1]);
   icm.getBlues(lutData[2]);

   LookupTableJAI lut = new LookupTableJAI(lutData);

   color = JAI.create("lookup", color, lut);
   }

 if (color.getNumBands() > 1) {

  double[][] matrix = {{ 0.114, 0.587, 0.299, 0 }};
  ParameterBlock pb = new ParameterBlock();
  pb.addSource(color);
  pb.add(matrix);
  color = JAI.create("bandcombine", pb, null);
  }
  return color;
}
public static PlanarImage binarize(PlanarImage color){
    PlanarImage gris = ToGrayScale(color);

     ParameterBlock pb = new ParameterBlock();
     pb.addSource(gris);
     pb.add(null); // The ROI
     pb.add(1); pb.add(1); // sampling
     pb.add(new int[]{256}); // bins stuff
     pb.add(new double[]{0});
     pb.add(new double[]{256});

     PlanarImage dummyImage = JAI.create(
             "histogram", pb);
     Histogram h = (Histogram)dummyImage.getProperty(
             "histogram");
     double[] thresholds = h.getMinFuzzinessThreshold() ;

     pb = new ParameterBlock();
     pb.addSource(gris);
     pb.add(thresholds[0]);
```

```
        // Creates the thresholded image.
        gris = JAI.create("binarize", pb);
        return gris;
    }
```

```
/**
 * Metodo para realizar la Operacion Morfologica de Bottom-Hat
 * el Bottom-Hat se realiza haciendo:
 * closing con un kernel de forma de disco+
 * resta entre imagen original y la imagen que fue tratada con closing
 *
 * @param im Imagen en escala de grises
 * @param patron Imagen original
public static PlanarImage Bottom_Hat(PlanarImage im,
        PlanarImage patron){

    ParameterBlock pb = new ParameterBlock();
    pb.addSource(im);
    pb.add(null); // The ROI
    pb.add(1); pb.add(1); // sampling
    pb.add(new int[]{256}); // bins stuff
    pb.add(new double[]{0});
    pb.add(new double[]{256});

    PlanarImage dummyImage = JAI.create("histogram", pb);
    Histogram h =
          (Histogram)dummyImage.getProperty("histogram");
    double[] thresholds = h.getMinFuzzinessThreshold() ;

    // Create the kernel using the array.
    KernelJAI kernel = new KernelJAI(19,19,kernelMatrix());

    //Se realizan las operaciones morfologicas para
    //encontrar la localizacion de la placa
    //Primero se realiza Bottom-Hat = closing+ resta
    //entre imagen original y actual
    //Operacion de Closing = dilate+erode

    PlanarImage location = dilate(im, kernel);
    location = erode(location, kernel);
```

```
    pb = new ParameterBlock();
    pb.addSource(location);
    pb.add(thresholds[0]);
    // Creates the thresholded image.
    return JAI.create("binarize", pb);

}
/**
 * Metodo para realizar la Operacion Morfologica de Substraccion
 * resta entre imagen original y la imagen que fue tratada con closing
 *
 * @param im Imagen en escala de grises
 * @param patron Imagen original
 * @return
 */


public static PlanarImage Substract(

PlanarImage im, PlanarImage patron){

ParameterBlock pb = new ParameterBlock();
pb.addSource(im);
pb.add(null); // The ROI
pb.add(1); pb.add(1); // sampling
pb.add(new int[]{256}); // bins stuff
pb.add(new double[]{0});
pb.add(new double[]{256});

PlanarImage dummyImage = JAI.create("histogram", pb);
Histogram h = (Histogram)dummyImage.getProperty("histogram");
double[] thresholds = h.getMinFuzzinessThreshold() ;

KernelJAI kernel = new KernelJAI(19,19,kernelMatrix());

//Se realizan las operaciones morfologicas para encontrar
//la localizacion de la placa
//Primero se realiza Bottom-Hat = closing+ resta entre
//imagen original y actual
//Operacion de Closing = dilate+erode
PlanarImage location = dilate(im, kernel);
location = erode(location, kernel);

    pb = new ParameterBlock();
    pb.addSource(location); pb.addSource(patron);
    return JAI.create("subtract", pb);
}
```

```
* Metodo para dilatar la imagen
* @param entrada Imagen a dilatar
* @param kernelHor Elementos q tendra la matriz
* //estructurante horizontalmente
* @param kernelVer Elementos q tendra la matriz
* //estructurante verticalmente
* @return Imagen dilatada
*/
public static PlanarImage dilate(PlanarImage entrada,
        int kernelHor, int kernelVer){

    KernelJAI kernel = new KernelJAI(kernelHor,kernelVer,
            getkernel(kernelHor*kernelVer));

 // Create a ParameterBlock with that kernel and image.
 ParameterBlock p = new ParameterBlock();
 p.addSource(entrada);
 p.add(kernel);
 // Dilate the image.
 PlanarImage salida = JAI.create("dilate", p);

      return salida;
  }

/**
 * Metodo para erosionar la imagen
 * @param entrada Imagen a erosionar
 * @param kernelHor Elementos q tendra la matriz
 * estructurante horizontalmente
 * @param kernelVer Elementos q tendra la
 * matriz estructurante verticalmente
 * @return Imagen erosionada
 */
public static PlanarImage erode(PlanarImage entrada,
        int kernelHor, int kernelVer){

    KernelJAI kernel = new KernelJAI(kernelHor,kernelVer,
            getkernel(kernelHor*kernelVer));

    // Create a ParameterBlock with that kernel and image.
    ParameterBlock p = new ParameterBlock();
    p.addSource(entrada);
    p.add(kernel);
    // Dilate the image.
    PlanarImage salida = JAI.create("erode", p);
    return salida;
}
```

```
public static PlanarImage aclarar(PlanarImage entrada,
        int kernelHor, int kernelVer){
    //Dilatacion Vertical
    entrada = dilate(entrada, 1, kernelVer);
    //Dilatacion Horizontal
    entrada = dilate(entrada, kernelHor, 1);
    return entrada;
}
```

```
public static PlanarImage crop(PlanarImage entrada,
        int x, int y){
    ColorModel colm = entrada.getColorModel();
    BufferedImage bim =
            entrada.getAsBufferedImage(
            new Rectangle(x,y,entrada.getWidth()-x*2,
                    entrada.getHeight()-y*2), colm);
    PlanarImage result = PlanarImage.wrapRenderedImage(bim);

    return result;
}
```

```
public static PlanarImage scale(PlanarImage ent,
        int width, int height){

    ColorModel colm = ent.getColorModel();
    BufferedImage aux =
            ent.getAsBufferedImage(new Rectangle(
            0,0,ent.getWidth(),ent.getHeight()), colm);

    BufferedImage bdest =
        new BufferedImage(width, height,
        BufferedImage.TYPE_INT_RGB);
    Graphics2D g = bdest.createGraphics();
    AffineTransform at =
    AffineTransform.getScaleInstance(
         (double)width/aux.getWidth(),(double)height/
         aux.getHeight());
    g.drawRenderedImage(aux,at);
    PlanarImage result = PlanarImage.wrapRenderedImage(bdest);
    return result;
}
```

```
public static PlanarImage rotate(PlanarImage im,
        float degrees){
    float newdeg = (float)Math.toRadians(360)+degrees;
    System.out.println("degrees en grados: "+
            Math.toDegrees(newdeg));

    ParameterBlock params = new ParameterBlock();
    params.addSource(im);
    params.add((float)im.getWidth());
    params.add(0.0f);
    if(degrees>=0)
        params.add(-degrees);
    else
        params.add((float)Math.toRadians(364.0d)+degrees);

    params.add(new InterpolationNearest());
    im = JAI.create("rotate", params);
    return im;
}
```

5.1.1.11 Relación del Componente Interfaz

En la figura 5.13 se muestra la relación entre el componente Interfaz y el componente Gestor.

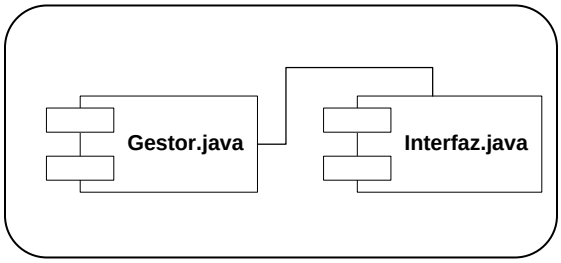


Figura 5.13 Relación entre el Componente Interfaz y el componente Gestor

5.1.1.12 Relación del Componente Archivo

En la figura 5.14 se muestra la relación entre el componente Archivo y el componente Gestor.

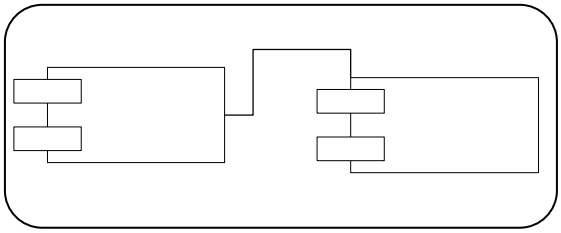


Figura 5.14 Relación entre el Componente Archivo y el componente Gestor

5.1.1.13 Relación del Componente BackPropagation

En la figura 5.15 se muestra la relación entre el componente BackPropagation y el componente Gestor.

Gestor.java

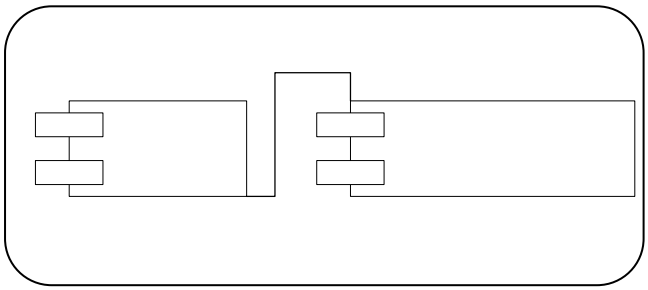


Figura 5.15 Relación entre el Componente BackPropagation y el componente Gestor

Código JAVA del Componente BackPropagation

```
 public double getError(int len)  {
     double err = Math.sqrt(globalError /
             (len * outputCount));
     globalError = 0; // clear the accumulator
     return err;
 }

public double threshold(double sum)
{
    return 1.0 / (1 + Math.exp(-1.0 * sum));
}
```

Gestor.java

```
public double[] computeOutputs(double input[])
{
    int i, j;
    final int hiddenIndex = inputCount;
    final int outIndex = inputCount + hiddenCount;

    for (i = 0; i < inputCount; i++)
    {
        fire[i] = input[i];
    }


    int inx = 0;



    for (i = hiddenIndex; i < outIndex; i++)
    {
        double sum = thresholds[i];

        for (j = 0; j < inputCount; j++)
        {
            sum += fire[j] * matrix[inx++];
        }
        fire[i] = threshold(sum);
    }

    double result[] = new double[outputCount];

    for (i = outIndex; i < neuronCount; i++)
    {
        double sum = thresholds[i];

        for (j = hiddenIndex; j < outIndex; j++)
        {
            sum += fire[j] * matrix[inx++];
        }
        fire[i] = threshold(sum);
        result[i - outIndex] = fire[i];
    }

    return result;
}
```

```
public void learn()    {
    int i;


    for (i = 0; i < matrix.length; i++)  {
        matrixDelta[i] = (learnRate * accMatrixDelta[i])
            + (momentum * matrixDelta[i]);
        matrix[i] += matrixDelta[i];
        accMatrixDelta[i] = 0;
    }

      for (i = inputCount; i < neuronCount; i++)  {
          thresholdDelta[i] = learnRate * accThresholdDelta[i]
              + (momentum * thresholdDelta[i]);
          thresholds[i] += thresholdDelta[i];
          accThresholdDelta[i] = 0;
      }
  }
```

5.1.1.14 Relación del Componente ExampleFileFilter

En la figura 5.16 se muestra la relación entre el componente ExampleFileFilter y el componente Gestor.

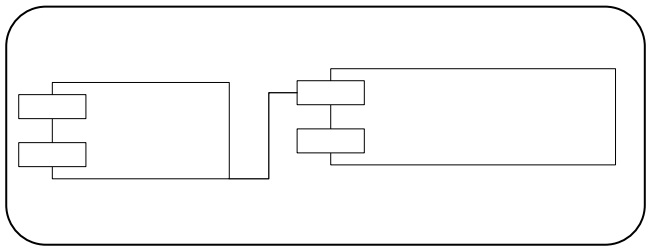


Figura 5.16 Relación entre el Componente ExampleFileFilter y el componente Gestor

5.1.1.15 Relación del Componente Gestor

En la figura 5.17 se muestra la relación entre el componente Gestor con todos los componentes.

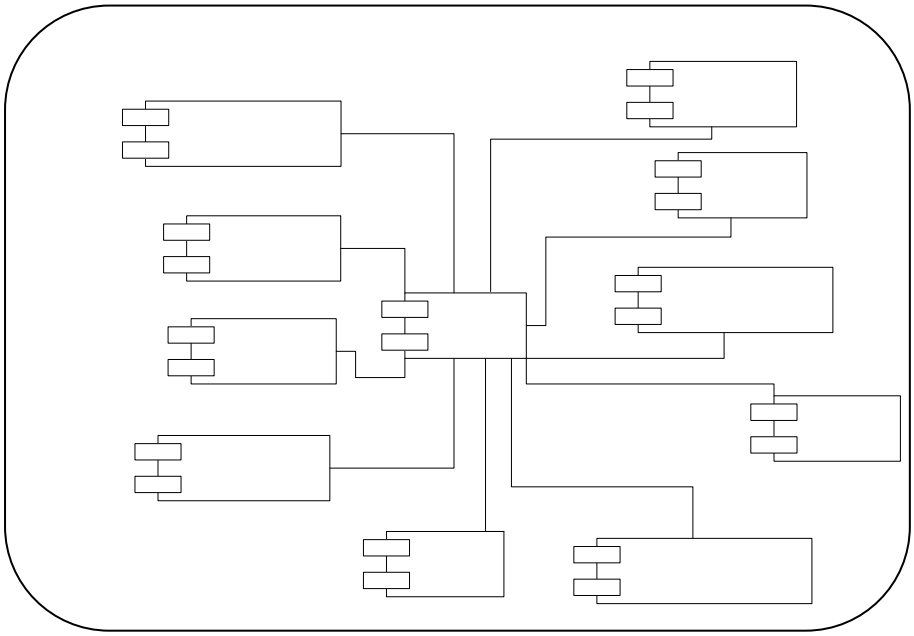


Figura 5.17 Relación entre el Componente Gestor con los componentes restantes

Código JAVA del Componente Gestor

ExampleFileFilter.java

Preprocess.java

```
public void run() {
    Thread miHilo = Thread.currentThread();
    if(imagen_fuente!=null){
        if(miHilo == hilo && preproceso==null){
            preproceso = new Preprocess(
                    Gestor.imagen_fuente);
            if(imagen_fuente.getWidth()==1024
                    && imagen_fuente.getHeight()==768){

                preproceso.setProgressBar(
                        jProgressBar_procesamiento);
                preproceso.setjLabel_procesos(
                        jLabel_procesos);
                preproceso.setjLabel_Preproceso(
                        jLabel_Preproceso);
                preproceso.procesar();
            }
```

```
else
    JOptionPane.showMessageDialog(new JFrame(),
        "Elija una imagen con Dimensiones 1024*768",
        "Dimensiones de la Imagen Incorrectas",1,
        new javax.swing.ImageIcon(getClass().getResource("" +
        "/Imagenes/Error Display.png")));
```

```
}
```

```
if(miHilo == hilo && segmentos==null && preproceso.isPlaca()){
    segmentos = new Segmentar(preproceso.getPlacaColor());
    segmentos.setProgressBar(jProgressBar_procesamiento);
    segmentos.setjLabel_procesos(jLabel_procesos);
    segmentos.setjLabel_Segmentar(jLabel_Segmentar);
    segmentos.procesar();
}
```

```
if(miHilo == hilo && caracteres==null && preproceso.isPlaca()){
    long Tinicial = System.currentTimeMillis();
    long Tfinal = System.currentTimeMillis();
    updateInterfaz("Inicializando Red BackPropagation",
    "BackPropagation: "+String.valueOf(Tfinal-Tinicial)+
            " mseg.", 0);

    caracteres = new String[7];
      if(network==null)
    try {

          network = getNetwork();
  } catch (IOException ex) {
      Logger.getLogger(Gestor.class.getName()).log(Level.SEVERE,
              null, ex);
  }

 caracteres = new String [7];
 for(int i=0; i<7;i++)
   try {
   caracteres[i] = reconocer(segmentos.getSegmentos()[i],
           network);
   Tfinal = System.currentTimeMillis();
   updateInterfaz("Reconociendo Caracter: "+String.valueOf(i),
           "BackPropagation: "+String.valueOf(Tfinal-Tinicial)+
           " mseg.", 2);

  } catch (IOException ex) {
      Logger.getLogger(Gestor.class.getName()).log(Level.SEVERE,
              null, ex);
  } catch (InterruptedException ex) {
      Logger.getLogger(Gestor.class.getName()).log(Level.SEVERE,
              null, ex);
  }
String[] array = getCaracteres();
String resultado ="";
for(int i=0;i<array.length;i++)
    resultado+=array[i];
jLabel_caracteres.setText(resultado);
```

```
  if(resultado.length()==6||resultado.length()==7)
      jLabel_tick.setIcon(
              new javax.swing.ImageIcon(getClass().getResource(
              "/Imagenes/Good.png")));
  else
      jLabel_tick.setIcon(new javax.swing.ImageIcon(
              getClass().getResource("/Imagenes/Warning.png")));

        }
    }

    hilo=null;
}

private String reconocer(Segmento s, BackPropagation network)
        throws IOException, InterruptedException{
          //escalar a 24*16 y esqueletizar

 int[] crudo = JaiOperations.scaleToInt(s.getImage(), 24, 16);

 double[] prueba = convIntRGBArrayToDouArray(crudo);

 Archivo arch = new Archivo();

  double[][] resultados = arch.leerResultados();

  for (int i=0; i<16; i++){
      for (int j=0; j<24; j++) {
          if(crudo[i*24+j]==-1)System.out.print(0);
          else System.out.print(1);
      }
      System.out.println();
  }

    double out[] = network.computeOutputs(prueba);

   for(int i=0; i<6;i++){
       System.out.print(out[i]+" ");
       if(out[i]<0.20d) out[i]=0.0d;
       if(out[i]>0.80d) out[i]=1.0d;

   }

 for(int i=0;i<resultados.length;i++)

if(eq(resultados[i],out)) resultado = arch.mapeoResultados[i];

return resultado;

}
```

```
private BackPropagation getNetwork() throws IOException{
    Archivo arch = new Archivo();
    arch.leerRecuerdos();
    double[][] resultados = arch.leerResultados();

NumberFormat percentFormat = NumberFormat.getPercentInstance();
percentFormat.setMinimumFractionDigits(4);

error =0.0d;
BackPropagation red = new BackPropagation(24*16,60,6,0.05,0.9);
int size = arch.getListaRecuerdos().size();

 for (int i=0;i<size*800;i++){
     double[] ideal = arch.getListaRecuerdos().get(i%size);

     red.computeOutputs(ideal);
     red.calcError(resultados[getIndexRes(
             arch.getListaMapeoRecuerdos(),
             arch.getMapRes(),i%size )] );
     red.learn();

     if(i%size ==0)
         error = red.getError(ideal.length);
 }
    return red;
}


  private double[] convIntRGBArrayToDouArray(int[] entrada){
      double[] recuerdos = new double[entrada.length];
      for(int i=0;i<entrada.length;i++){
          if(entrada[i]==-1) recuerdos[i]=0.0d;
          else recuerdos[i]=1.0d;
      }
      return recuerdos;
  }

private int getIndexRes(LinkedList<String> mapRecuerdos,
        String[] mapResultados, int index){


for(int i=0;i<mapResultados.length;i++)
    if(mapResultados[i].compareTo(mapRecuerdos.get(index))==0)
            return i;
    return -1;
}
```

5.1.2 Diagramas de Componentes del Paquete Imágenes

En la figura 5.18 se muestra al paquete Imágenes y los Componentes por los que está formado

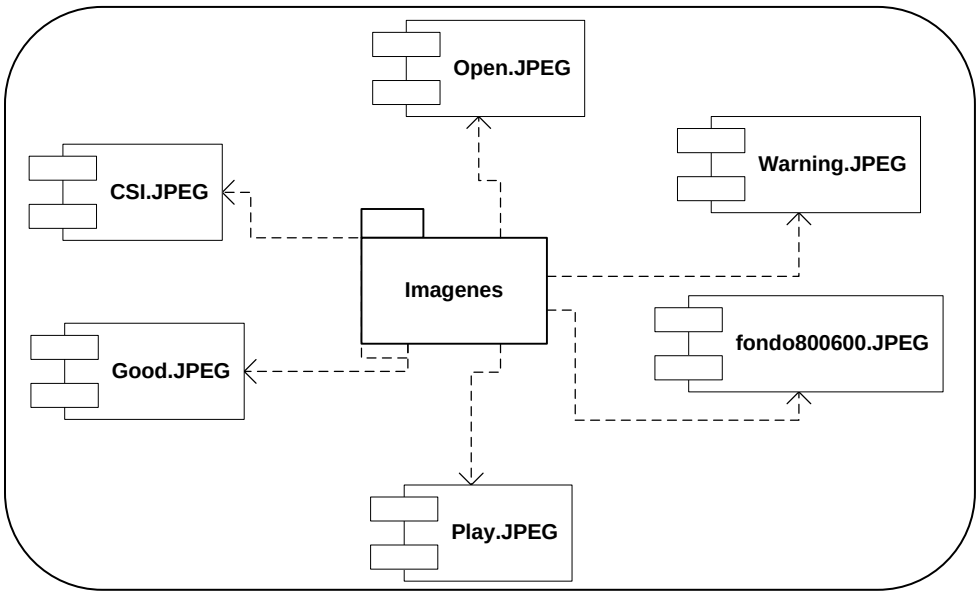


Figura 5.18 paquete Imágenes sus Componentes

5.2 Diagrama de Interfaz

La interfaz de usuario (IU) es uno de los componentes más importantes de cualquier sistema computacional, pues funciona como el vínculo entre el humano y la máquina. La interfaz de usuario es un conjunto de protocolos y técnicas para el intercambio de información entre una aplicación computacional y el usuario. La IU es responsable de solicitar comandos al usuario, y de desplegar los resultados de la aplicación de una manera comprensible.

5.2.1 Diagrama de Interfaz del Sistema Reconocedor de Placa

5.2.1.1 Diagrama Interfaz Principal

En la figura 5.19 se puede observar la Interfaz Principal de la aplicación.

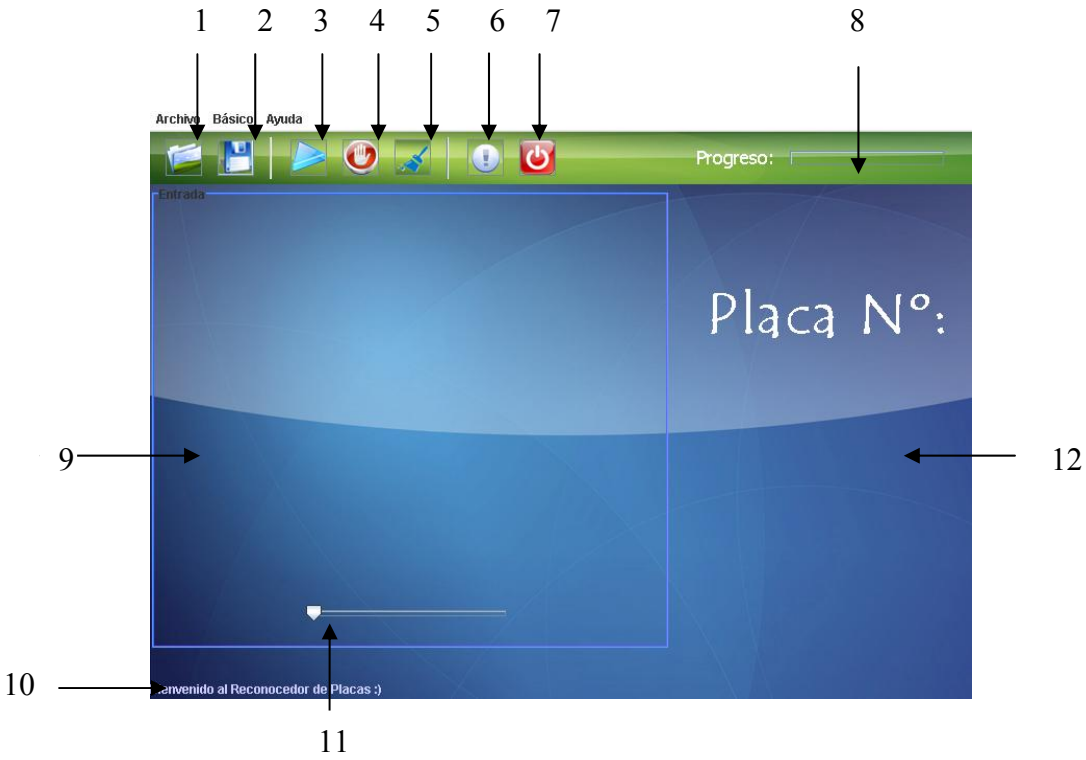


Figura 5.19 Diagrama Interfaz Principal

Leyenda Diagrama Interfaz Principal

1. Botón de acceso directo para cargar el archivo de entrada.
2. Botón de acceso directo para guardar la solución obtenida.
3. Botón de acceso directo para iniciar el reconocimiento de la placa.

4. Botón de acceso directo para detener el proceso de reconocimiento.
5. Botón de acceso directo para borrar la foto del panel de muestra, la placa obtenida y volver al estado inicial el panel de status del programa.
6. Botón de acceso directo para abrir el centro de ayuda.
7. Botón de acceso directo para cerrar el programa.
8. Barra de progreso cuando la aplicación está analizando la imagen de entrada.
9. Panel de muestra de la imagen cargada.
10. Panel de status de la aplicación.
11. Deslizador que permite visualizar los diferentes estados del procesado de la imagen.
12. Panel donde se muestra la placa resultado.

5.2.1.2 Interfaz Menú Archivo

En el menú archivo están comprendidas las acciones para abrir el archivo de entrada, guardar archivo de salida y salir de la aplicación, donde se muestra el icono del botón de acceso directo, y las teclas de atajo para activarla. (Ver Figura 5.20)

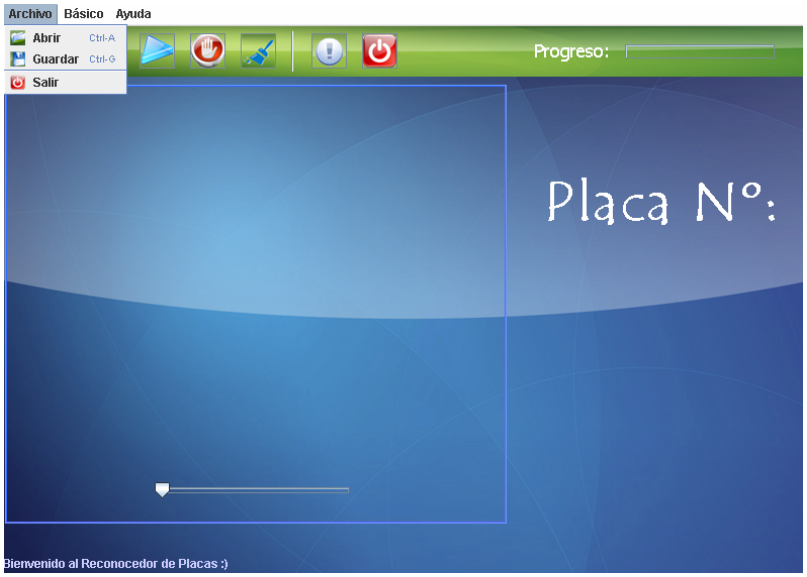


Figura 5.20 Diagrama Interfaz Menú Archivo

5.2.1.3 Interfaz Menú Básico

En la figura 5.21 se muestra la Interfaz del menú Básico, el cual contiene la acción de mostrar una ventana emergente donde se muestra la visualización del Histograma de la imagen de entrada, incluye el icono de acceso directo y las teclas de atajo para cada acción.

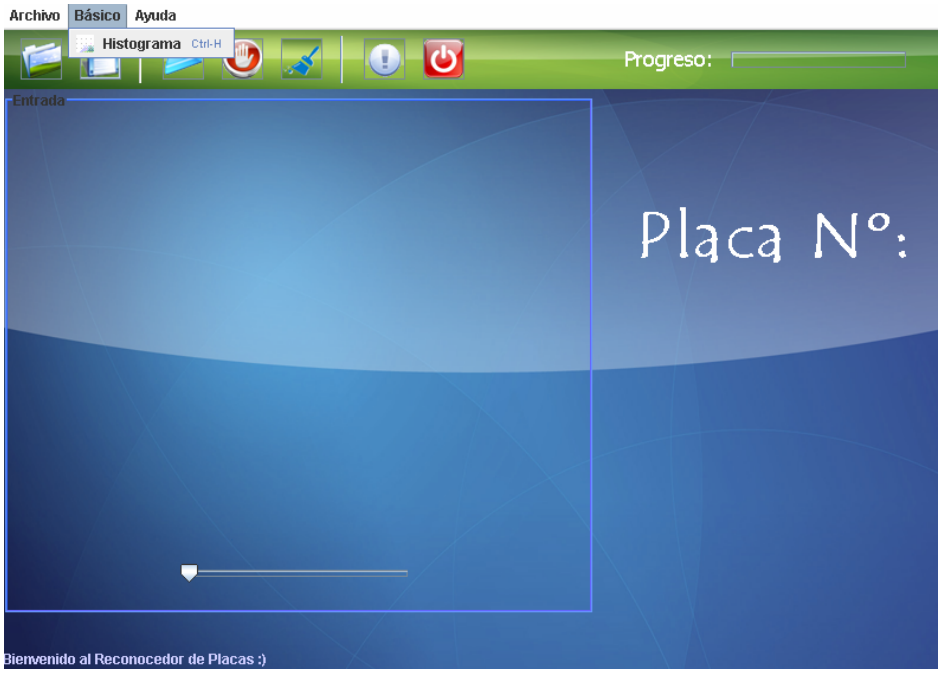


Figura 5.21 Diagrama Interfaz Menú Básico

5.2.1.4 Interfaz Menú Ayuda

En la figura 5.22 se muestra el contenido del menú Ayuda, el cual solo contiene la acción de mostrar la ayuda de la aplicación, incluye el icono de acceso directo y teclas de atajo.

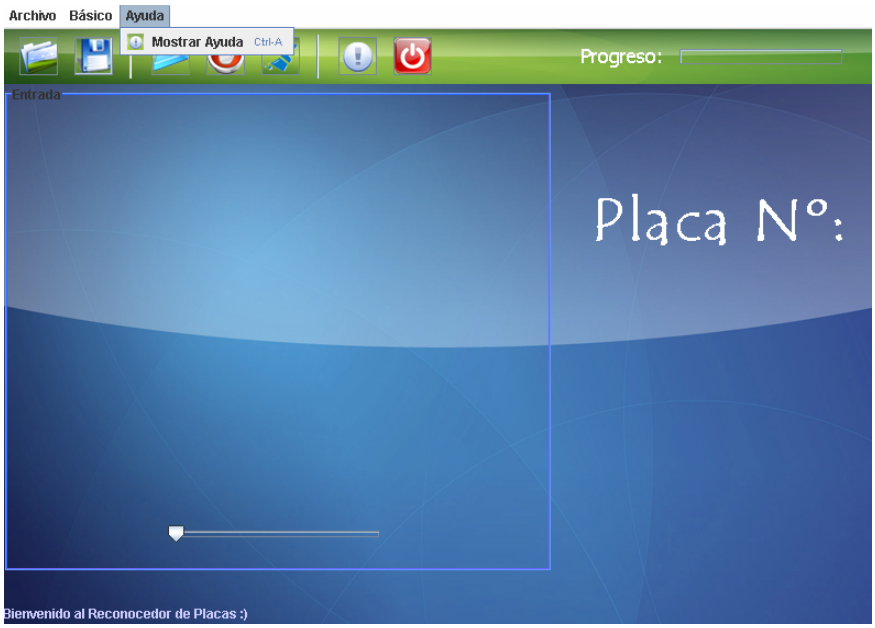


Figura 5.22 Diagrama Interfaz Menú Ayuda

5.3 Pruebas

Las pruebas se realizaron para demostrar el buen funcionamiento del software y como éste respondió tanto para los distintos métodos de la visión artificial utilizados como para la implementación de la red neuronal BackPropagation, se hizo necesario acotar la forma de captura de las placas quedando limitadas a un tamaño de 1024x768 pixeles y la distancia promedio entre la cámara fotográfica y el vehículo de 1,60 metros, todas estos cambios fueron el resultado de muchas pruebas las cuales permitieron llegar a la conclusión de limitar la forma de captura de la imágenes a procesar para el mejor desempeño tanto en tiempo como en resultados del programa a continuación en la figura 5.23 se muestra el resultado para el caso de prueba 5.3.1.

5.3.1 Caso de prueba

Figura 5.23. Lado izquierdo la imagen de entrada, lado derecho resultado.

5.3.2 Tablas de Resultado

La tabla que se presenta contiene los datos generados de las 61 pruebas realizadas para este software, las cuales contenían diversas variaciones al momento de captar la imagen, variaciones tales como hora a la que se tomó la foto, inclinación del vehículo, sombra sobre el mismo, distancia desde la cámara hasta el vehículo.

De los datos obtenidos en la Tabla de resultado (ver Tabla 14 y 15) fueron tomadas en consideración las siguientes penalizaciones: En una placa de 6 caracteres, cada carácter erróneo o no encontrado en la placa de prueba sería restado un 16,67% del porcentaje total de acierto; con respecto a las placas de 7 caracteres, por cada carácter erróneo se le restó un 14,29% del total de acierto de la aplicación. En cada placa probada, se comenzó con un 100% del total de acierto y a partir de ahí se fue penalizando carácter por carácter. Los porcentajes de penalización fueron considerados de la siguiente manera: Para las placas de 6 caracteres se hizo una sencilla regla de 3 donde el resultado fue 100 entre 6, donde el último valor es el número de caracteres encontrados en la placa, igualmente se realizó para las placas de 7 caracteres.

Los tiempos de Ejecución fueron tomados a partir de una misma computadora de modo de estandarizar los tiempos y tomar un promedio cercano a la realidad de cuánto tardaría la aplicación en procesar el reconocimiento.

Los datos tomados para realizar el análisis de los resultados que arrojó la tabla fueron los siguientes: Imagen, Placa Imagen, Tiempo Ejecución y el porcentaje de acierto. Dentro de los valores promedios de la tabla se encuentran:

- Promedio Tiempo Ejecución Aplicación: 2442,19672 (milisegundos).

- Promedio de Acierto Reconocimiento: 95,7059016%

Tabla 14 Resultados

Imagen	Placa Imagen	Tiempo Ejecución	Placa Resultante	% Acierto
1	AA668AE	3923	AA668AE	100
2	BBJ22Y	2719	BBJ22Y	100
3	BBZ11T	2594	BBZ11T	100
4	AGK47L	2423	AGK47L	100
5	21LDBC	2437	21LDBC	100
6	BBE82F	2390	BBE82F	100
7	JAR07I	2296	JAR07I	100
8	AFB49E	2641	AFB49E	100
9	BCG52H	2156	BCG52EH	83,33
10	BAK76R	3000	BAK76R	100
11	FBI33N	2328	FBI33N	100
12	BBS05D	2359	BBS05D	100
13	NAW22D	2313	NAW22D	100
14	BAH02J	2719	BA02J	83,33
15	MDR77G	2359	MDR77G	100
16	FBV63G	2297	FBV63G	100
17	BBS24S	2406	BBS24S	100
18	RAP44B	2235	RAP44B	100
19	AA566CS	3062	A566CS	85,72
20	MCV250	2625	MCVW250	83,33
21	29RGBA	2406	298GBA	83,33
22	BBU12K	2516	BBUA12K	83,33
23	AF020J	2359	AF20J	83,33
24	GDE00Y	2485	GDE00Y	100
25	AGG70B	2296	AGG70B	100
26	BAA900	2343	BAA900	100
27	BBW83H	2344	BBW83H	100
28	48RDAZ	2406	48RDAZ	100
29	BBI87Z	2470	BBI87Z	100
30	NAU47J	2469	NAU47J	100
31	GBN13X	2438	GBN13X	100
32	BBU86K	2203	BBU86K	100
33	BCA59R	2281	BCA59R	100
34	DDC03P	2203	DDC03P	100
35	LAP00G	2313	LAP00G	100

Tabla 15 Continuación Resultados

Imagen	Placa Imagen	Tiempo Ejecución	Placa Resultante	% Acierto
39	VCP93W	2313	VCP93W	100
40	BBG31E	2360	BBG31E	100
41	0AN36L	2578	0AN36L	100
42	A85AK0G	2360	A85AKG	85,72
43	LAS81W	2265	LASA81W	83,33
44	BCA42K	2266	BCA42K	100
45	RAP79W	2250	FRAP9W	66,66
46	AA925FK	2203	AA925FK	100
47	BBF48Z	2579	BF48Z	83,33
48	AFD03D	2312	AFD03D	100
49	BCD70X	2531	BCD70X	100
50	BAY26V	2500	BAY26V	100
51	FBU25J	2235	FBU25J	100
52	BCA54C	2265	CA54C	83,33
53	BCE50U	2578	BCE50U	100
54	BAS54N	2453	BAS54N	100
55	BAY10D	2563	BAY10D	100
56	NA059J	2219	NAA059J	83,33
57	CAB80A	2172	CAB80	83,33
58	BBM52A	2642	BBM52A	100
59	51UGBF	2391	51UGBF	100
60	MBD64L	2438	MBD64L	100
61	BBU44E	2312	BBU44E	100
		2442,19672		95,7059016
Promedio Tiempo Ejecución Aplicación:			2442,19672	
Promedio de Acierto Reconocimiento:			95,7059016	

CONCLUSIONES

- Algunas de las imágenes de prueba tomadas presentaron información no funcional en la parte exterior del vehículo de prueba, por lo cual se recorta la imagen antes de realizar cualquiera operación sobre la misma, para evitar falsos positivos.

- Cuando se localizó la zona donde se encontraba la placa, se presentó el problema que la misma no estaba completamente horizontal sino que tenía una inclinación leve, eso se corrigió calculando las esquinas del segmento de la placa y se realizaba una rotación a la imagen.

- En muchas ocasiones, se encontró el problema que para algunas placas donde la iluminación era excesiva y se le aplicaba una dilatación posterior el carácter se eliminaba por completo, así que la eliminación de ruido en la imagen se produjo muchas veces solamente por la binarización de la misma y se le aplicaba una sutil dilatación de esta.

- Se realizaron distintos escalamientos a los segmentos de los caracteres de la placa concluyendo que la resolución con la que estos segmentos mantenían la información fue de 24 unidades de ancho por 16 unidades de alto, con una resolución menor a esta algunos caracteres perdían información valiosa para su posterior reconocimiento, también se intentó esqueletizar cada segmento o carácter, pero ocurría el mismo problema de la pérdida de información así que se decidió no realizar esta operación.

- Se demostró el algoritmo BackPropagation es idóneo para este tipo de problemas, con el inconveniente que se deben tener un número elevado de

recuerdos por cada carácter, mientras más recuerdos tenga la red será más eficiente y exacta en el reconocimiento.

Sobre la base del análisis de 61 casos de prueba con diferentes niveles de dificultad se obtuvo las siguientes conclusiones:

- El tiempo de respuesta de la aplicación fue rápido para cada uno de los casos de prueba arrojando un tiempo total promedio de 2442,19672 (milisegundos).

- El promedio de reconocimiento total fue de 95,7059016% por lo que se puede decir que es un sistema confiable.

RECOMENDACIONES

- Se recomienda tener una cantidad de 150 patrones por cada carácter dentro de los recuerdos del programa, esto para volverlo más robusto y lograr que el porcentaje de acierto sea mayor a 95%.

- Las placas venezolanas actuales poseen en cada carácter la etiqueta de "Venezuela" impresa en relieve como medida de seguridad, por lo que se recomienda implementar autómatas celulares para reconocer este nuevo tipo de placas y reconstruir la imagen de los segmentos debido a que en el proceso de binarización los caracteres tienden a perder información.

- Se recomienda el uso de cámaras infrarrojas con el fin de eliminar las limitaciones que acarrea la captura de imágenes por medio de cámaras fotográficas con respecto a la iluminación, el efecto del flash.

- Se recomienda la reestructuración del algoritmo de nivelación del ángulo de la placa en la imagen para reconocer placas que estén con un grado de inclinación mayor.

BIBLIOGRAFIA

1. Wikipedia la enciclopedia libre "**Visión por Computador**" (2008) http://en.wikipedia.org/wiki/Computer_vision.

2. Sucre Desiree, Verde Abraham, "**Desarrollo de un Agente Inteligente que identifique objetos geométricos específicos según su forma o color dentro de un ambiente definido aplicando técnicas de Visión Artificial**" (2007).

3. Luna Raquel, Parada Henry, "**Desarrollo de un sistema reconocedor de carácter alfanuméricos, que transforma una imagen escaneada de un manuscrito en un archivo de texto mediante algoritmos de Visión Artificial**" (2007).

4. Craig Larman **"UML y Patrones"** segunda Edición (2006).

5. Sulbaran Charlie, Yánez Sioli, "**Desarrollo de un Agente Inteligente que intercepta la trayectoria de un proyectil que se desplaza en un ambiente limitado de 2 dimensiones, utilizando técnicas de Visión Artificial**" (2005).

6. Gerardo Moreno Martínez "**Ingeniería de Software UML"** (2005).

7. Díaz Juan, Fermín Lila, "**Desarrollo de un software para el reconocimiento de caracteres alfabéticos manuscritos mediante Algoritmos de Visión Artificial**" (2004).

8. Benjamín Rodríguez **"UML"** (2004).

9. Jose Ali Moreno **"Redes Neuronales Cibernéticas"** Postgrado de Computacion Emergente (2003).

10. Netherlands Organization for Scientific Research, "**Operaciones basadas en Morfología**" (2000) http://www.ph.tn.tudelft.nl/Courses/FIP/noframes/fip-Morpholo.html

11. Javier Gonzales Jiménez "**Visión por Computador**" (2000).

12. R. Lackes and D. Mack, en colaboración con J. Ziola y K. Ahern "**Neuronal Networks: Basics and Applications"**. (1998).

13. Castelman K.R "**Digital Image Processing**" (1996).

14. Rodriguez Morejón y Antonio Angel "FACE: Un Entorno para la Segmentación de Rasgos Faciales" (1995).

15. Frecman y Skapura " **Redes Neuronales una Visión**" (1993).

16. Freeman J.A y D.M Skapura "**Redes Neuronales Algoritmos, Aplicaciones y Tecnicas de Programacion**" (1993).

17. Horowitz, S.L Pavlidis "**Picture Segmentation by a Directed Split-and-Merge Procedure**" (1974).