UNIVERSIDAD TECNOLÓGICA DE LA MIXTECA

Arquitectura hardware de un sistema de reconocimiento utilizando histogramas de gradientes orientados y una red neuronal DEN

TESIS

PARA OBTENER EL TÍTULO DE:

INGENIERO EN MECATRÓNICA

PRESENTA:

ISAAC ALVAREZ NAJERA

DIRECTOR DE TESIS:

DR. ENRIQUE GUZMÁN RAMÍREZ

HUAJUAPAN DE LEÓN, OAXACA. FEBRERO, 2020

Resumen

El campo de la visión artificial o por computadora es el encargado de emular las acciones biológicas que se presentan en el sistema visual del ser humano, por ello la visión artificial comprende un conjunto de técnicas para adquirir, procesar, segmentar, clasificar, analizar y comprender datos provenientes de un entorno tridimensional. De esta forma, esta disciplina conlleva el surgimiento de diversas ramas de aplicación, donde una de las tareas más importantes es la de reconocimiento, especialmente la de clasificación o reconocimiento de objetos múltiples.

En este trabajo de tesis se propone la implementación en hardware de un sistema de reconocimiento utilizando el descriptor HOG para la extracción de características y una red neuronal DEN como clasificador. Para realizar dicha tarea, primeramente, se realizó un modelado conceptual de los algoritmos en lenguaje de alto nivel, con el objetivo de identificar las fases de los algoritmos y poder realizar algunas modificaciones para su posterior implementación en hardware. Por su parte, la implementación en hardware se desarrolló dentro del FPGA Spartan 6-XC6SLX16 utilizando una metodología descendente, el desarrollo de las arquitecturas hardware de los algoritmos HOG y DEN-*network* fueron modeladas utilizando un lenguaje descriptor de hardware VHDL.

Finalmente, en las pruebas de desempeño realizadas tras evaluar un conjunto de 2, 3, 5, 10, 15 y 20 clases de las bases de datos COIL-20 y ALOI, se obtuvo una eficiencia superior al 90% para la base de datos ALOI y para conjuntos inferiores a 20 clases de la base de datos COIL-20. Así mismo, se estimó que el tiempo para la extracción de características y clasificación de un objeto es de 1.398ms.

Dedicatoria

A mis padres Martha y Antonio

En mi búsqueda del agradecimiento ideal no eh podido encontrar algo que se adecuara a ustedes, porque darles las gracias me parece poco debido a todo lo que me brindaron, por ello me quedo sin palabras y no me queda más que dedicarles este trabajo y decirles gracias por la educación que me brindaron, su apoyo y sus enseñanzas, ustedes me convirtieron en el hombre que soy y dejaron el camino para el hombre en el que espero convertirme.

A mi abuelita Valentina De La Cruz

Porque tu amor y memoria siempre estarán en nuestros corazones.

A mi abuelita Gloria Coronel

Por el amor que nos tienes a tus nietos, pues tu cariño, ejemplo y ayuda, nos fortalecen para recorrer nuestro propio camino.

A mi abuelito Luis Alvarez

Por tu amor y cariño incondicional que nos proporciona calma y seguridad.

A mi hermana Zayra

Por ser un ejemplo a seguir.

Isaac Alvarez Najera.

Agradecimientos

En el caminar de la obscuridad de la ignorancia, en la que solo veía sombras, me eh encontrando con gigantes que en su infinita bondad me han acogido, y al elevarme y dejarme ver a través de sus ojos me han motivado para ir hacia mundos desconocidos.

Los gigantes presentes en mi vida a quienes les debo mi gratitud.

Martha, mi madre, por apoyarme, guiarme y enseñarme a tener un orden relativo no solo en mi cuarto, sino en mi vida, y así como preocuparse por mi constantemente.

Antonio, mi padre, por apoyarme, guiarme, enseñarme que es la disciplina y mostrarme como es tener pasión por una carrera y profesión, animarme a ser mejor y llegar más lejos cada día.

Zayra, mi hermana, por mostrarme que la excelencia es algo a lo que hay que aspirar y que podemos alcanzar.

Mis abuelos Gloria, Valentina y Luis, así como los familiares que siempre me motivaron y apoyaron de diversas maneras.

Dr. Enrique Guzmán, por siempre tener la disposición de enseñarme, apoyarme, permitirme trabajar con él, así como dirigir esta tesis.

Dr. Ivan García, por brindarme su apoyo y guiarme en los pasos que eh dado durante la carrera, pues sin él no me imagino haber tomado los caminos que curse en la universidad.

Sin dejar al lado, es grato mencionar que sin su compañía y amistad la vida no sería igual, gracias a mis amigos Pedro, Hugo, Noel, Carlos, Magda y Celina, por su apoyo y aguantar mi mal humor, les agradezco desde lo más profundo de mi ser.

Finalmente, gracias a los profesores de la UTM que siempre me brindaron su apoyo y conocimiento.

Índice

Contenido

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

1.1 Contexto

La visión es uno de los sistemas sensoriales más complejos e indispensables que el ser humano posee y utiliza para la realización de sus actividades diarias, las cuales van desde tareas tan simples como el identificar y evitar obstáculos o el reconocimiento de objetos o personas, hasta complejas tareas como el poder manejar un automóvil, o desarrollar un trabajo de tipo industrial (como lo son el maquinado, el cableado y la soldadura de piezas). El sistema visual biológico es un sistema bastante complejo, en este la percepción se origina en el lóbulo occipital en la corteza visual primaria lo cual ocasiona que, como lo menciona Gimenez, "*los impulsos luminosos estimulan fotorreceptores hiperpolarizando su potencial de membrana, lo que permite que la información nerviosa alcance, tras varias sinapsis, las células ganglionares de la retina; finalmente, estas descargan potenciales de acción y sus axones inician una complicada ruta anatómica para alcanzar la corteza cerebral*" [1]. Este conjunto de procesos sensoriales, cognitivos y cinestésicos otorgan la capacidad de procesar información del entorno, lo cual brinda el sentido espacial, el color, la forma, el relieve o posición de los objetos [2].

Por otro lado, la visión artificial o por computadora, de acuerdo con [3] y [4], son un conjunto de técnicas para adquirir, procesar, segmentar, clasificar, analizar y comprender datos tomados de un entorno tridimensional, procesados a partir de una imagen bidimensional y definidos en un vector de características, cuya clasificación puede obtenerse mediante el uso de la inteligencia artificial, la cual se basa en propiedades brindadas por la imagen. De igual manera se puede observar que el campo de estudio de la visión artificial ha tenido un mayor crecimiento en las

últimas décadas, dando como resultado la existencia de diversas ramas de aplicación y desarrollo entre los sectores de la industria e investigación, permitiendo que cada vez procesos repetitivos y monótonos para el ser humano puedan ser automatizados [5], [6]. Además, una importante propiedad de algunos de estos procesos es que el sistema de visión artificial pueda ser integrado de manera compacta en un dispositivo de procesamiento, al igual que tener un alto grado de eficiencia en la tarea que desempeña. De esta forma, una importante especificación del sistema propuesto en esta investigación es que debe formar parte de un sistema autónomo, por lo que se ha elegido modelar una arquitectura hardware de un sistema de reconocimiento dentro de un dispositivo reconfigurable, específicamente, en un arreglo de compuertas programables en el campo **(**FPGA, *Field Programable Gate Array***)**. Esto es debido a las prestaciones que un FPGA ofrece, destacando la concurrencia, característica que permite modelar arquitecturas que presentan una alta eficiencia energética y temporal.

1.2 Planteamiento del problema

Se sabe que en la actualidad una gran cantidad de los problemas electrónicos pueden ser resueltos por cualquier dispositivo de procesamiento existente, como son microprocesadores, microcontroladores, procesadores de señales digitales (DSP, *Digital Signal Processor*), ASICs, unidades de procesamiento grafico (GPU, *Graphics Processing Unit*) y FPGAs. Sin embargo, existen ciertos problemas en los cuales el procesamiento ofrecido por estos dispositivos puede no ser suficiente, tal es el caso donde existe gran cantidad de datos de entrada/salida que deben ser procesados en un lapso de tiempo corto, o cuando el número de operaciones por muestra es elevado, ambos ejemplos se presentan en las tareas que un sistema de visión debe realizar.

Por otra parte, durante un proceso de reconocimiento es deseable que un sistema de visión artificial sea robusto a diversas transformaciones geométricas que un objeto puede sufrir, como la traslación, rotación y escala, así como robusto a problemas generados por el entorno, tales como la iluminación y, además que tenga un alto porcentaje de eficiencia al cumplir su tarea de reconocimiento.

Finalmente se pretende que el sistema propuesto en este trabajo de tesis forme parte de un sistema de visión autónomo. Resulta claro que, debido a la naturaleza de un sistema autónomo, se requiere de un elemento central de procesamiento que, además de realizar las tareas para las que fue concebido en forma eficiente, permita diseñar un sistema que sea compacto y tenga bajos requerimientos de consumo de recursos y de potencia. Además, el elemento central de procesamiento debe ser capaz cumplir eficientemente tareas relacionadas con el procesamiento de imágenes, las cuales resultan complejas y con alta demanda de recursos en la mayoría de los casos.

1.3 Justificación

La mayor parte de los problemas que plantea un sistema de visión artificial autónomo tiene que ver con la demanda de una gran cantidad de recursos, la necesidad de realizar un gran número de operaciones y requerimientos de altas velocidades de procesamiento. Al ser un sistema autónomo hay que agregar la necesidad de contar con un sistema que tenga bajos requerimientos de consumo de potencia. En este sentido, existen dispositivos que pueden ser usados para dar solución a esta problemática. Dispositivos como microprocesadores, microcontroladores y DSP's, son descartados debido a que utilizan un paradigma secuencial para cumplir con su tarea, lo que implica tiempos de ejecución largos, además que es difícil explotar el paralelismo existente en algunos algoritmos. Los ASIC's, al tener la mejor relación área-velocidad-consumo, son el

dispositivo ideal para esta tarea. Sin embargo, su costo los hace una opción que solo grandes compañías son capaces de costear. Además, también presentan las desventajas de tener largos tiempos de diseño y ser dispositivos no reconfigurables, lo que dificulta una actualización del sistema. Por su parte, los GPU's resultarían una alternativa adecuada, ya que al poseer una gran cantidad de unidades de procesamiento podrían realizar tareas con cierto grado de concurrencia y ofrecer altas velocidades de procesamiento. Sin embargo, el GPU tiene el inconveniente de la carga de un sistema operativo.

Para dar solución a la problemática planteada, en este trabajo de tesis se plantea utilizar un FPGA como elemento central de procesamiento. Un FPGA resulta ser la opción más equilibrada debido a las siguientes razones, 1. La característica más importante de un FPGA es la capacidad de realizar tareas en forma concurrente, permitiendo explotar el paralelismo existente en los algoritmos, lo que genera altas velocidades de procesamiento. 2. La arquitectura de un FPGA brinda una gran flexibilidad de diseñar, lo que permite modelar fácilmente una arquitectura para una tarea específica. 3. El costo de estos dispositivos y de las herramientas de diseño resultan bastante accesibles para el ámbito académico. 4. Las herramientas de diseño permiten modelar, implementar y probar un sistema en lapsos de tiempo relativamente cortos. 5. Son fácilmente adaptables para trabajar en un sistema autónomo, debido a su uso moderado de energía y su portabilidad.

Finalmente, el presente trabajo tiene por objetivo principal generar arquitecturas hardware para las tareas de extracción de características y clasificación pertenecientes a un sistema de visión artificial. Se propone utilizar el descriptor HOG y una DEN-*network*, para cubrir las mencionadas tareas. El uso de estas técnicas se debe principalmente a que HOG ha mostrado una eficiencia en sistemas de reconocimiento de personas superior al 83.3%, dada su robustez a factores como la iluminación, rotación y posición [7], [8], mientras que la DEN-*network* permite obtener un alto porcentaje de eficiencia ocupando pocos recursos del sistema [9].

1.4 Hipótesis

La implementación en hardware del descriptor HOG y la DEN-*network*, para las fases de extracción de características y clasificación de un sistema de visión artificial, en conjunto pueden alcanzar una eficiencia en el reconocimiento multi-objetos superior al 90 %.

1.5 Objetivo

1.5.1 Objetivo general

Diseñar y modelar una arquitectura hardware para el descriptor HOG y otra para la DEN-*network* e implementarlas en un FPGA para que realicen tareas de reconocimiento.

1.5.2 Objetivos específicos

- Modelado conceptual del descriptor HOG mediante un lenguaje de alto nivel.
- Modelado conceptual de la DEN-*network* mediante un lenguaje de alto nivel.
- Diseño y modelado de una arquitectura hardware del descriptor HOG.
- Diseño y modelado de una arquitectura hardware de la DEN-*network*.
- Implementación en un FPGA de las fases de extracción de características y clasificación de un sistema de visión con base en las arquitecturas modeladas.

1.6 Solución propuesta

En el sistema propuesto de este trabajo de tesis se implementan las fases de extracción de características y clasificación de un sistema de visión artificial dando como resultado un sistema de reconocimiento. De esta forma se busca que, dada una imagen de entrada, se lleve a cabo dentro de una arquitectura hardware el procesamiento necesario para extraer un vector de características mediante el descriptor HOG, y a través de la red neuronal DEN-*network* obtener como resultado a que clase pertenece la imagen de entrada. La metodología implementada se muestra en el diagrama a bloques de la Figura 1.1, la cual expresa las fases en las que consistirá el desarrollo de este proyecto de tesis.

1.- Modelado conceptual. Dada la facilidad que un lenguaje de alto nivel ofrece al implementar algoritmos, un lenguaje de este tipo será utilizado para modelar tanto la red neuronal como el extractor de características.

- Red neuronal: Modelado de la red neuronal DEN-*network* utilizando Matlab.
- Descriptor: Modelado del descriptor HOG utilizando Python, el modelado y verificación de este algoritmo será hecho en forma modular.
- Evaluación de modelado: Se evaluará su desempeño interfazando los componentes anteriores y utilizando un conjunto de entrenamiento generado por el descriptor HOG a partir de imágenes predefinidas que será procesado por la red neuronal. Para este fin, se utilizarán las bases de datos estándares COIL y ALOI.

2.- Modelado en Hardware. Con base a los resultados obtenidos en el modelado conceptual del algoritmo HOG y la DEN-*network* se determinó cuales elementos y técnicas de modelado de hardware resultan idóneos para modelar las arquitecturas hardware correspondientes. Con este fin, se utilizará el lenguaje de descripción de hardware VHLD y nivel de abstracción RLT y algorítmico.

- Red neuronal: Modelado en VHDL.
- Descriptor HOG: Modelado en VHDL.
- Evaluación de modelado: Se interfazaran los dos módulos anteriores dentro del FPGA para realizar las pruebas correspondientes.

3.-Evaluación de resultados: Se comparará el desempeño que tuvo el modelado en hardware con el modelado en software utilizando las mismas bases de datos.

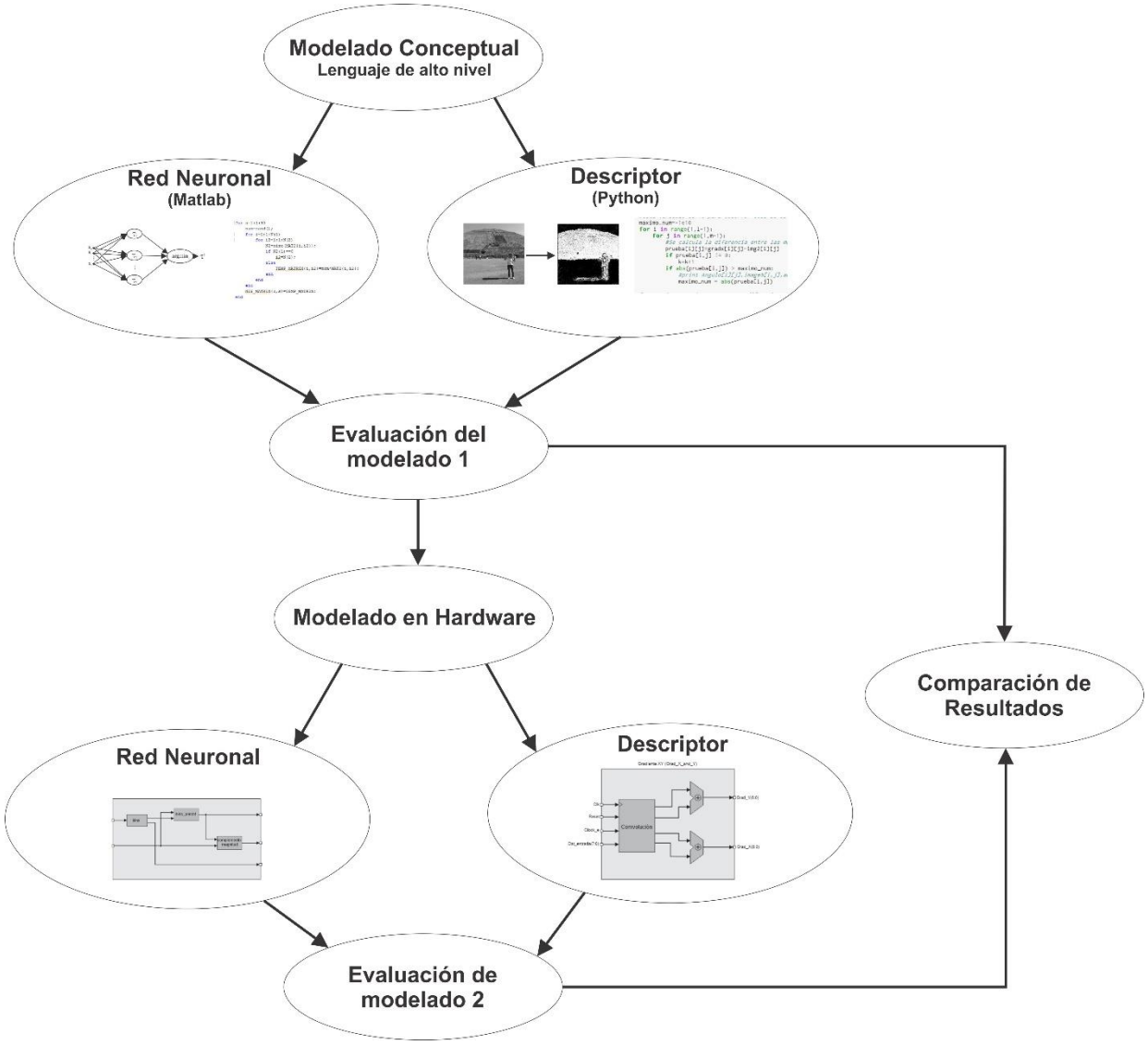

Figura 1.1. Diagrama de bloques de las fases del desarrollo del sistema de reconocimiento.

1.7 Contribuciones

Las principales contribuciones del presente trabajo de tesis son las siguientes:

- Implementar un sistema de reconocimiento en forma modular en una arquitectura hardware.
- Integrar el descriptor HOG en una arquitectura hardware.
- Integrar el clasificador DEN-*network* en una arquitectura hardware.
- Generar un sistema de reconocimiento multi-objeto en un FPGA.

1.8 Organización del documento

La organización del presente documento consta de 5 capítulos cuya descripción se presenta a continuación.

Capítulo 1. Introducción. Capitulo presente, en donde se explica los aspectos generales en los que consistirá el trabajo de tesis.

Capítulo 2. Marco Teórico. En este capítulo se presentan la descripción de los conceptos, algoritmos y bases teóricas necesarias para la comprensión y desarrollo de los algoritmos necesarios para elaborar el sistema de reconocimiento en el modelado conceptual y la implementación en hardware. Se inicia ampliando la explicación de un sistema de visión y se procede a describir ampliamente como están conformados el descriptor de HOG y el clasificador DEN-*network.*

Capítulo 3. Modelado Conceptual. Este capítulo presenta la implementación de los algoritmos del descriptor HOG y el clasificador DEN-*network* siguiendo un paradigma secuencial y utilizando un lenguaje de alto nivel. Una implementación de este tipo permite probar y verificar como afectan las modificaciones propuestas en el algoritmo de HOG al desempeño global del sistema hardware.

Capítulo 4. Implementación en Hardware. Se presentan los esquemas y elementos de hardware utilizados para el desarrollo de las operaciones involucradas en el algoritmo de HOG y la DEN-*network*, así mismo se muestran las técnicas utilizadas para la reducción del tiempo de procesamiento entre cada fase del sistema.

Capítulo 5. Conclusiones y Trabajo Futuro. En dicho capítulo, se expresan las conclusiones obtenidas en este trabajo de tesis dado los resultados de los capítulos anteriores. Además, se mencionan las perspectivas o trabajos futuros que se plantean para la continuación y complemento de la investigación presentada.

Finalmente, se presentan las referencias bibliográficas de este trabajo.

Capítulo 2

Marco Teórico

En este capítulo se encuentran los fundamentos teóricos de los temas, principios, técnicas y algoritmos implementados en este trabajo de tesis. En el primer apartado se describe una breve comparativa entre la visión biológica y el proceso de visión artificial. En el apartado siguiente se exponen los conceptos y algoritmos referentes a la extracción de características que se planea implementar y al método de clasificación por el que se optó. Posteriormente, se realiza la descripción de la composición del dispositivo implementado. Al final del capítulo se incluye una compilación de resultados de otras investigaciones que versan sobre el tema de investigación escogido.

2.1 Paradigma biológico

El funcionamiento de los cuerpos biológicos de los animales y del ser humano han sido una fuente de inspiración para una gran variedad de trabajos a lo largo de los años, p. ej., dicha inspiración se puede encontrar en los primeros intentos de vuelo del hombre, el cual consistía en tratar de imitar las alas de las aves. Dado el gran desempeño que tienen los sistemas biológicos aún en la actualidad existen trabajos que tratan de igualar la eficacia del sistema en cuestión. Para conseguirlo han surgido una gran variedad de campos, como es el caso de la robótica, en la cual algunos trabajos tratan de imitar el comportamiento animal, p. ej., el guepardo creado por Boston Dynamics, que pretende imitar el sistema de desplazamiento del animal mencionado, llegando a desplazarse a una velocidad de 32 km/h [10]. Todos estos sistemas incluyen áreas y sub-áreas que implementan mecanismos de control para poder manipular dichos sistemas y cuyos

principios pueden ser utilizados en otras áreas. Otra rama que trata de igualar la eficacia de un sistema biológico y que es de particular interés para este trabajo de tesis es la visión artificial, cuyo propósito es igualar la capacidad de reconocimiento que tiene el ser humano, pero conseguirlo representa un gran reto dado al alto costo computacional que conlleva el poder reconocer o clasificar objetos.

Como se sabe el sistema visual de los humanos es bastante complejo, por lo que tratar de hacer un análogo artificial consta de una ardua tarea que requiere gran cantidad de recursos de cómputo y exige altas velocidades de procesamiento. A continuación, se describirá resumidamente como son las etapas del sistema de visión humano para estar en contexto con la complejidad que conlleva imitarlo.

2.1.1 Sistema de visión biológico humano

Usualmente cuando se hace mención al sistema de visión sólo se suele involucrar a los ojos, sin embargo, el sistema de visión se compone de varias etapas en los cuales suceden diversos tipos de procesos físicos y químicos. La primera de estas etapas es la captura, esto ocurre cuando la luz pasa a través del ojo y estimula una serie de células sensoriales que se encuentran en la parte posterior de éste, una vez que los sensores son estimulados crean una serie de señales que viajan por el nervio óptico hasta llegar a la corteza visual primaria que se sitúa en el lóbulo occipital [1].

Como se puede observar en la Figura 2.1, el sistema de visión humano se compone esencialmente de los ojos, el nervio óptico, el núcleo geniculado lateral (LGN, *Lateral Genticulate Nucleus*) y la corteza occipital en la zona V1 del cerebro, la cual se encarga del procesamiento de la imagen captada.

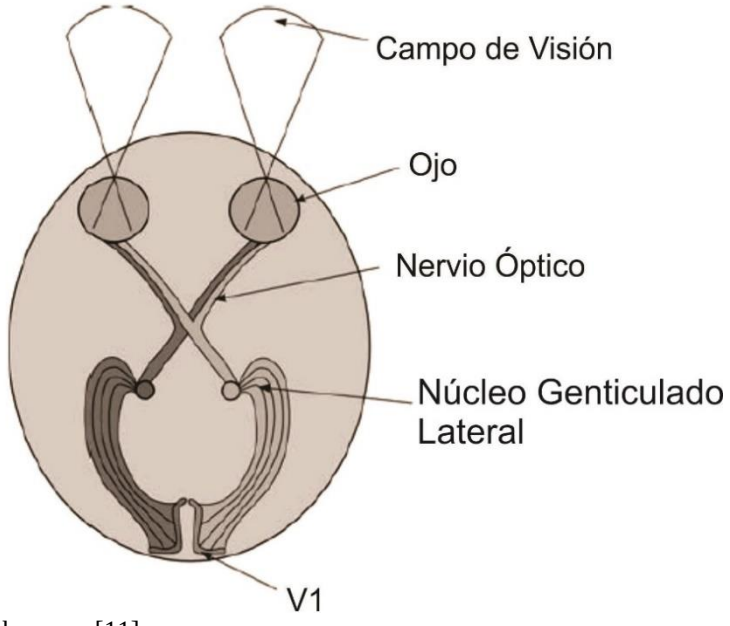

Figura 2.1. Partes del ojo humano [11].

La operación del sistema de visión comienza cuando a partir del campo visual la luz atraviesa la retina y llega hasta los nervios sensoriales ubicados en la parte posterior del ojo, estos nervios sensoriales están compuestos por células fotosensibles, las cuales provocan un estímulo eléctrico que pasa a través del nervio óptico y llega al núcleo geniculado lateral, que se encuentra en un área del tálamo, la cual recibe y reenvía información a la corteza visual primaria mediante capas de neuronas [11].

A pesar de que existen diversas zonas del cerebro encargadas del proceso visual, existe un conjunto de regiones que son las encargadas de las principales funciones que cumple la visión, tales regiones han sido denominadas con la letra "V". A este respecto, la más grande de estas regiones, y donde se cree que se lleva a cabo la mayor parte del procesamiento preliminar, es la región V1. Por su parte, la llamada región V2 es aquella que está más relacionada con la memoria visual. Mientras que la región V3 tiene relación directa en la percepción de las formas dinámicas, la V4 lo tiene en la percepción del color y sus formas. Finalmente, la región V5 es la encargada de la percepción del movimiento. El conjunto de interconexiones entre las zonas de la corteza visual integra distintos atributos que conforman la información visual. Una vez obtenida la información esencial, compuesta por impulsos, se logra estimular las neuronas para que de esta forma el cerebro pueda ordenar, clasificar y relacionar objetos [11], [1].

2.1.2 Estructura y funcionamiento de una neurona

Las neuronas son las unidades celulares básicas que comprenden una amplia red y cuyo propósito es procesar el flujo de información del sistema nervioso. Esta transferencia de información que ocurre entre neuronas se debe gracias a señales químicas en forma de neurotransmisores. El flujo de información a través de la neurona se conduce eléctricamente a las zonas pre-sinápticas donde los neurotransmisores se liberan para comunicarse con las siguientes neuronas, en el transcurso de información entre las neuronas, y dado que no son conexiones pasivas, recopilan la información de diferentes fuentes, por lo que al pasar la información entre una de ellas determinan si la información recibida puede ser excitatoria o inhibitoria, además de que la respuesta acumulativa a estas aportaciones es inequívoca para disparar un potencial de acción o permanecer inhibida [12]. La Figura 2.2 muestra la estructura básica de una neurona.

Los elementos que integran a una neurona biológica son los siguientes:

- **Dendritas:** Capturan y cotejan información local proveniente de los axones.
- **Axón:** Es la entrada y salida de información.
- **AIS:** El segmento inicial axonal (*axonal initial segment*) entradas inhibidoras en el cuerpo de la neurona.
- **Barrera de difusión:** Permite el paso a químicos dentro de las conexiones neuronales.
- **Nodo de Ravier:** Son interrupciones que ocurren a lo largo del axón.

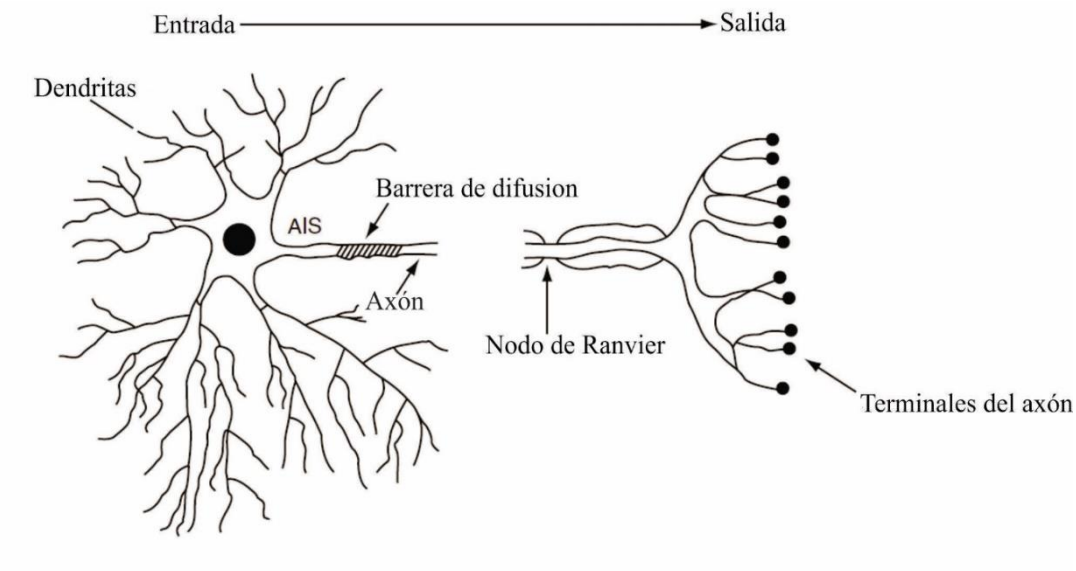

Figura 2.2. Estructura de una neurona [12].

2.2 Sistema de visión artificial

En 1963 se crea el primer sistema de visión artificial, hecho por Lawrence Roberts, cuyo trabajo es conocido como "*Block World*", el cual consistía en que una máquina tuviera la percepción de sólidos tridimensionales. Esto se logró poniendo parámetros específicos para obtener una percepción de profundidad, logrando construir una matriz tridimensional a partir de una fotografía bidimensional [13].

Un sistema de visión está definido por las fases de captura, preprocesamiento, segmentación, extracción de características y clasificación como se puede ver en la Figura 2.3.

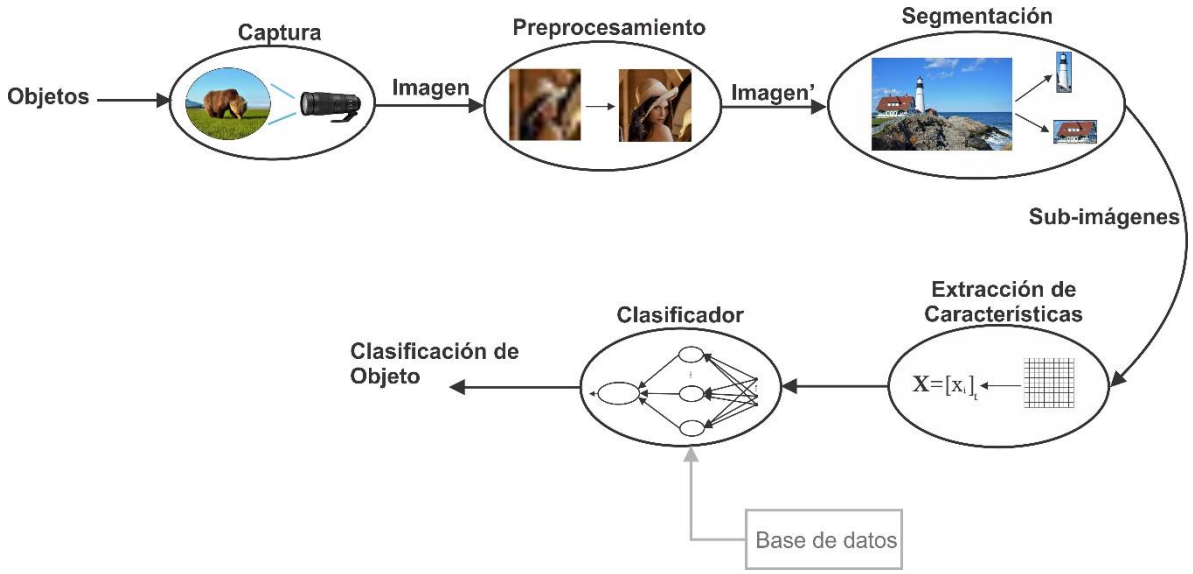

Figura 2.3. Etapas de un sistema de visión artificial.

Considerando lo mencionado en los apartados anteriores se puede hacer una analogía básica de un sistema de visión biológico y un sistema de visión artificial. El proceso de estimulación de los receptores del ojo humano es emulado por un sistema artificial mediante un proceso de captura de imagen realizado mediante una cámara. Las tareas de preprocesamiento y segmentación que el sistema artificial realiza pueden ser comparadas con las funciones que el LGN y la región V1 del cerebro cumplen. Por otro lado, las funciones desarrolladas por las regiones V3, V4 y V5 son análogas al proceso de extracción de características que el sistema artificial realiza. Finalmente, el proceso de clasificación e identificación de objetos se encuentra relacionado con la región V2 y las neuronas que son las encargadas de procesar la información.

Dado que la visión artificial es una disciplina que conlleva diversas ramas de aplicación, resulta un tanto difícil mencionarlas en su totalidad. Sin embargo, estas pueden clasificarse de acuerdo con el tipo de actividad o tarea que desempeñan, como se puede ver en [14].

- **Automatización:** Es el proceso mediante el cual actividades repetitivas, pudiendo ser complejas o no, son realizadas por máquinas programadas permitiendo un mejor control de calidad y eficiencia en los procesos. Ejemplos de este tipo de tarea son [15], [16].
- **Medición o calibración:** Se refiere a la correlación cuantitativa con los datos del diseño, asegurando que las dimensiones o mediciones cumplan con las especificaciones del diseño. Por ejemplo, el comprobar que un cable tenga el espesor recomendado. Otros ejemplos de esta actividad pueden ser encontrados en [17], [18].
- **Detección de fallas:** Consiste en un análisis cualitativo que involucra la detección de defectos o artefactos no deseados, con forma desconocida en una posición desconocida. Por ejemplo, encontrar desperfectos en el chasis de un auto nuevo, también [19], [20] representan ejemplos de este tipo de tarea.
- **Verificación:** Es la comprobación cualitativa de si una operación de ensamblaje o fabricado ha sido llevada a cabo correctamente. Por ejemplo, que no falte ninguna tecla en un teclado o que no falten componentes en un circuito impreso, o el uso de soldadura en acero. En [21], [22] se documentan tareas de verificación.
- **Reconocimiento:** Involucra la identificación de un objeto con base en características o descriptores asociados a él. Por ejemplo, la clasificación de cítricos (limones, naranjas, mandarinas, etc.) por color o tamaño. Algunos otros ejemplos relacionados con el reconocimiento son documentados en [23], [24].
- **Identificación:** Es el proceso de identificar un objeto por el uso de símbolos en el mismo. Por ejemplo, el código de barras, o códigos de perforaciones empleados para distinguir hule espuma de asientos automotrices. En [25], [26] presentan ejemplos de este tipo de tarea.
- **Análisis de localización:** Es la evaluación de la posición de un objeto. Por ejemplo, determinar la posición donde debe insertarse un circuito integrado o la identificación de un lugar en una zona determinada para el desplazamiento y ubicación de un robot autónomo. Ejemplos adicionales de este tipo de actividad son [27], [28].
- **Guía:** Significa proporcionar adaptativamente información posicional de retroalimentación para dirigir una actividad. El ejemplo típico es el uso de un sistema de visión para controlar un brazo robótico mientras suelda o manipula partes. Otro ejemplo seria la navegación en vehículos autónomos. Adicionalmente, en [16], [28] se reportan ejemplos de este tipo de tarea.

Como se mencionó, la diversidad de ramas de aplicación que tiene la visión artificial ha generado que esta disciplina experimente un incremento en su desarrollo y aplicación, repercutiendo en diversas soluciones a las tareas planteadas.

Bajo este contexto, en el presente trabajo de tesis se plantea una solución a la que quizá es la tarea más importante en el área de la visión artificial, el reconocimiento, poniendo especial atención en la clasificación o reconocimiento de múltiples objetos.

2.3 Reconocimiento de objetos

El reconocimiento de objetos es el conjunto de técnicas que tienen por finalidad la identificación de un objeto inmerso en una imagen utilizando características intrínsecas a él. Estas características deben permitir discriminar el objeto en estudio de otros objetos y es deseable que sean invariantes a cambios de posición, color, iluminación y forma del objeto.

Los investigadores involucrados en esta área consideran al reconocimiento de objetos un gran desafío debido principalmente a las siguientes razones:

- La gran variación que los objetos pueden presentar en su apariencia, debiéndose a causas como transformaciones visuales, diferentes poses, variaciones intrínsecas, oclusiones, ruido, etc.
- La similitud entre clases de objetos causada por algunos patrones visuales compartidos por diferentes clases.
- Se trata de una tarea en demasía compleja y que requiere una gran cantidad de tiempo y recursos para poderse implementar.

El reconocimiento de objetos al ser una terea derivada de la visión artificial se compone de las mismas etapas que esta contiene; captura, preprocesamiento, segmentación, extracción de características y clasificación.

Para el presente trabajo de tesis es de particular interés hacer énfasis en las últimas dos etapas del sistema de visión, la extracción de características y la clasificación. Se trabajará con bases de datos estándares de imágenes procesadas y segmentadas, lo que permitirá enfocarse en modelar e implementar arquitecturas hardware para las fases de extracción de características, mediante histogramas de gradientes orientados (HOG, *Histogram of Oriented Gradients*), y de clasificación o reconocimiento, con una red neuronal basada en neuronas con dendritas elipsoidales (DEN, *Dendrite Ellipsoidal Neuron*). Antes de profundizar en estos temas, se describirán brevemente cada una de las fases que integran a un sistema de reconocimiento de objetos.

2.3.1 Sistema de captura

En la actualidad, un sistema de captura de una imagen posee una estructura bien definida. En la Figura 2.4 se puede apreciar el esquema general de un sistema de captura, el cual está constituido por los siguientes elementos,

- **Lente**: Su principal función consiste en atrapar la luz y enfocarla para lograr una mayor perspectiva de la imagen tridimensional.
- **Filtro Bayer**: Es una matriz de filtros, rojos, verdes y azules cuya principal función es dejar pasar la luz a cada célula de sensores, estimulándolos de forma diferente por el uso de filtros, permitiendo poder reconstruir una imagen bidimensional de un entorno tridimensional.

Figura 2.4. Proceso de captura de una imagen.

- **Sensor**: Su función consiste que al estimular pequeños materiales fotosensibles envié una serie de pulsos eléctricos al procesador que dependen de la imagen a capturar.
- **Procesador de Imagen**: Su función es interpretar y construir una imagen digital bidimensional, principalmente con 3 canales (RGB), a partir de una perspectiva tridimensional. Esta reconstrucción permite tener una percepción de un entorno tridimensional.

Para que una escena del mundo real, definida en 3D, pueda ser procesada por un sistema digital, resulta imperativo discretizarla y representarla en 2D, obteniendo como resultado lo que se conoce como imagen. El sistema de captura es el encargado de efectuar este proceso. La discretización de la escena se realiza tanto en coordenadas espaciales (x, y), denominado muestreo, como en intensidad o color, denominado cuantificación. La imagen resultante es representada por una función $\boldsymbol{F} = f(x, y)$ cuyo valor es proporcional a la intensidad o al color de la escena en la coordenada espacial definida por x y y. Así, la escena es aproximada mediante una representación matricial de $m \times n$ elementos, denominados píxeles, los cuales son muestras de la escena igualmente espaciadas. La Figura 2.5 ilustra el proceso de discretización y la aproximación discreta de una imagen.

2.3.2 Preprocesamiento o acondicionamiento de la imagen

Dentro del proceso de captura existen diversas técnicas de fotografía que permiten mejorar la imagen capturada, p. ej. realizar un buen enfoque, mejorar la perspectiva del objeto, etc. A pesar de esto, es necesario considerar que la tarea de captura será realizada por una máquina, la cual carece de propiedades o habilidades desarrolladas para tomar una fotografía óptima. Esto en muchas ocasiones significa que se tiene como producto final una imagen cuyas características han sido afectadas significativamente a causa del enfoque del dispositivo de captura, del procesador, o de las condiciones climáticas, lo que puede llegar a inducir en la imagen ruido, efecto de granulado, manchas en la imagen, saturación de luz, o deformación de la imagen.

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & \cdots & f(0,n-1) \\ f(1,0) & f(1,1) & \cdots & f(1,n-1) \\ \vdots & \vdots & & \vdots \\ f(m-1,0) & f(0,0) & \cdots & f(m-1,n-1) \end{bmatrix}$$

Figura 2.5. Aproximación discreta de una imagen.

La segunda etapa del sistema de visión, el preprocesamiento, tiene por finalidad subsanar los problemas generados por la etapa de captura, es decir, la imagen es sometida a un proceso de mejoramiento para minimizar estos problemas y pueda ser usada de forma eficiente en las etapas posteriores. El mejoramiento de contraste, la reducción del ruido y realce de características son algunos procesos que son aplicados a imágenes para adecuarlas para las siguientes etapas

2.3.3 Segmentación

Como se mostró en el esquema general de un sistema de visión, en la Figura 2.4, la etapa de segmentación consiste principalmente en aislar elementos de una imagen general, es decir, en esta etapa se crea un conjunto de sub-imágenes normalizadas y en ocasiones se acondiciona el fondo de éstas, permitiendo sólo resaltar el objeto en cuestión.

2.3.4 Extracción de características

La extracción de características es una parte fundamental e indispensable para todo sistema de visión artificial debido a que, primero, una imagen contiene una gran cantidad de datos, aun si ésta ha sido segmentada. Segundo, la mayor parte de estos datos proporciona muy poca información útil para interpretar la escena. Entonces, el objetivo de la extracción de características es buscar, seleccionar y expresar en forma adecuada las características que proporcionen información precisa del contenido de la escena. Este proceso hace que el reconocimiento o clasificación de objetos sea más eficiente al determinar características específicas de un objeto inmerso en una imagen y hacerlas disponibles a través del llamado "vector de características". Además, el vector generado es más pequeño que la imagen en cuestión, es decir disminuye la dimensionalidad de la imagen haciendo de la clasificación un proceso más simple.

El vector de características de una imagen, dependiendo del algoritmo usado para generarlo, ayuda a que el sistema tenga robustez a diversos factores, tales como al tamaño, iluminación, rotación, posición, etc. Es decir, si el sistema es robusto al tamaño, significa que la imagen puede ser pequeña o grande y el vector de características generado tendrá los mismos valores o valores similares en ambos casos. De igual manera un descriptor puede ser robusto a varios factores, pero ser poco robusto o nulo a otros, como es el caso los momentos de HU, el cual nos brinda robustez en rotación y posición, pero no en iluminación.

Diversos enfoques han sido utilizados para crear algoritmos de extracción de características de objetos inmersos en imágenes. Cada enfoque contempla características diferentes y se basa en operaciones específicas, p. ej., existe un enfoque que agrupa algoritmos que basan su operación en el cálculo de momentos geométricos de la imagen, los momentos de Hu es uno de estos algoritmos. Un enfoque en particular es importante para este trabajo de investigación, aquel que incluye algoritmos que fundamentan su operación en el uso de características locales codificadas en un descriptor. Algunos de estos algoritmos son HOG [29], SIFT (*Scale-Ivariant Feature Transform*) [30], SURF (*Speeded Up Robust Features*) [31], DAISY [32], LBP (*Local Binary Patterns*) [33], BASIS (*Basis Sparce-coding Inspired Similarity*) [34].

El descriptor HOG es parte integral del sistema propuesto en este trabajo de investigación, principalmente por su robustez a factores como la iluminación, rotación y posición. Una descripción detallada de este algoritmo es presentada en el apartado 2.4.

2.3.5 Clasificación de patrones

La clasificación de patrones es el proceso de asignar cualquier muestra de datos medidos u observados como miembro de una de las varias clases o categorías existentes dentro de un conjunto de datos mediante el uso de algoritmos. En problemas de clasificación de patrones, los datos de entrenamiento del sistema consisten en tomar un conjunto de vectores de entrada, donde cada vector se obtiene al extraer una serie de características de los objetos de interés. Cuando en un sistema de clasificación los datos de entrenamiento comprenden ejemplos de vectores de entrada en conjunto con sus valores objetivo correspondientes, se conocen como problemas de aprendizaje supervisados. En problemas donde los datos de entrenamiento contienen un conjunto de vectores de entrada sin ningún valor objetivo, se conocen como problemas de aprendizaje no supervisados, cuyo objetivo puede ser el de formar nuevos conjuntos o subconjuntos a partir de los datos de entrada [35], [36].

Matemáticamente, la clasificación es un proceso que se encarga de asignar una instancia representada por un vector de características o atributos $\mathbf{x} = [x_1, x_2, \dots, x_n] \in \mathbb{R}^n$ a una clase c de un conjunto de clases $\boldsymbol{C}$, mediante una función de clasificación f que asigna etiquetas a las observaciones de los objetos de entrada.

Los principales enfoques utilizados en la clasificación de patrones son, el estadístico o teoría de decisión, sintáctico o estructural, lógico combinatorio y neuronal [37].

- Estadístico o teoría de decisión. En este enfoque se hace uso de argumentos probabilísticos dados por la naturaleza cuantitativa de las características generadas por el objeto estudiado, y a partir de estas observaciones se da un conjunto de distribuciones de probabilidad que permiten hacer la clasificación.
- Sintáctico o estructural. Se basan en la representación explicita de la estructura geométrica de una clase, donde representa conceptualmente la forma característica de los patrones de entrenamiento y a partir de su representación matemática realizar el reconocimiento de objetos.
- Lógico combinatorio. Este enfoque funciona partiendo de las descripciones de las características de los objetos a partir de probabilidades de observación, esta observación se caracteriza en dar atributos al objeto sin hacer suposiciones que no estén fundamentadas.
- Neuronal. Su principio se basa en los supuestos de comportamiento de las estructuras neuronales del cerebro, tratando de emular alguna de las funciones de la sinapsis donde las neuronas involucradas tienen un umbral de excitación. Uno de los esquemas que permite imitar las funciones del cerebro con pequeñas unidades de cálculo son las redes neuronales artificiales (RNAs).

En este trabajo de tesis se emplea un clasificador que utiliza un enfoque neuronal, específicamente una red de neuronas con dendritas elipsoidales (DEN-*network*, *Dendrite Ellipsoidal Neuron* - *Network*) [9], la cual se explicará detalladamente en el apartado 2.5.

2.4 Histograma de gradientes orientados

En el año 2005, Navneet Dalal y Bill Triggs propusieron un sistema de visión artificial donde presentaron al algoritmo HOG y lo aplicaron a la detección de humanos [29]. Este trabajo consistía, además de presentar un nuevo descriptor, en la detección de bordes para localizar peatones, los resultados de esta implementación mostraron que el descriptor HOG es robusto a

diversos factores, como la iluminación, posición y fondos desordenados, lo que implica un alto grado de eficiencia al momento de implementar un sistema de reconocimiento.

En la detección de peatones es de suma importancia que un descriptor sea robusto a diversas condiciones climáticas y físicas, debido a que estas pueden distorsionar la imagen. En este sentido HOG se planteó como una reminiscencia de los histogramas de orientación en los bordes, además del uso de celdas uniformemente espaciadas y superpuestas, en las cuales se hace uso de la normalización de contraste, lo que brinda una mejor descripción de un objeto especifico en una imagen de gran tamaño [29]. Una celda es una submatriz de la imagen que representa una región espacial local donde se calcula un histograma, en la sección 2.4.3 el concepto de celda será definida con precisión.

HOG es un descriptor que basa su operación en el cálculo de histogramas. Este algoritmo se compone de 5 fases, partiendo de una imagen en escala de grises se realiza el cálculo de los gradientes de la imagen y su respectiva magnitud y orientación, enseguida se divide la imagen en pequeñas regiones y se obtiene su histograma, consecutivamente se realiza el proceso de ventaneo para obtener un conjunto de propiedades a partir de bloques de características que agrupan a los histogramas, finalmente se realiza un proceso de normalización y se conforma el vector de características mediante la concatenación de las propiedades de los bloques pertenecientes a la ventana de detección, ver Figura 2.6 y 2.7.

Este vector de características puede utilizarse en la parte de clasificación de un sistema de reconocimiento. La formación del vector de características se explicará más ampliamente en el apartado 2.4.3. El motivo por el cual la fase de ventaneo se lleva acabo después de la obtención de histogramas es para evitar repetir las operaciones en el traslape de bloques de características.

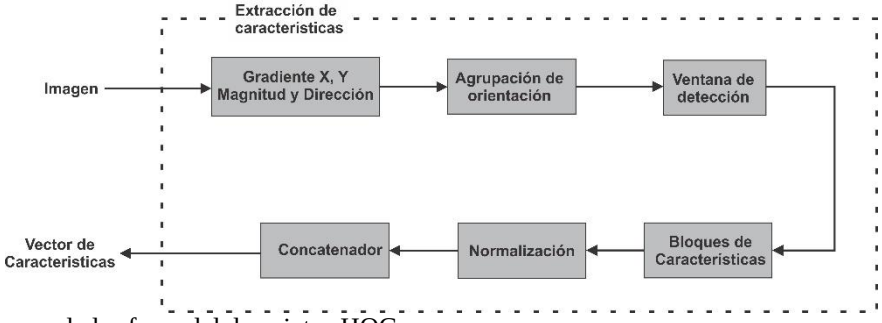

Figura 2.6. Esquema de las fases del descriptor HOG.

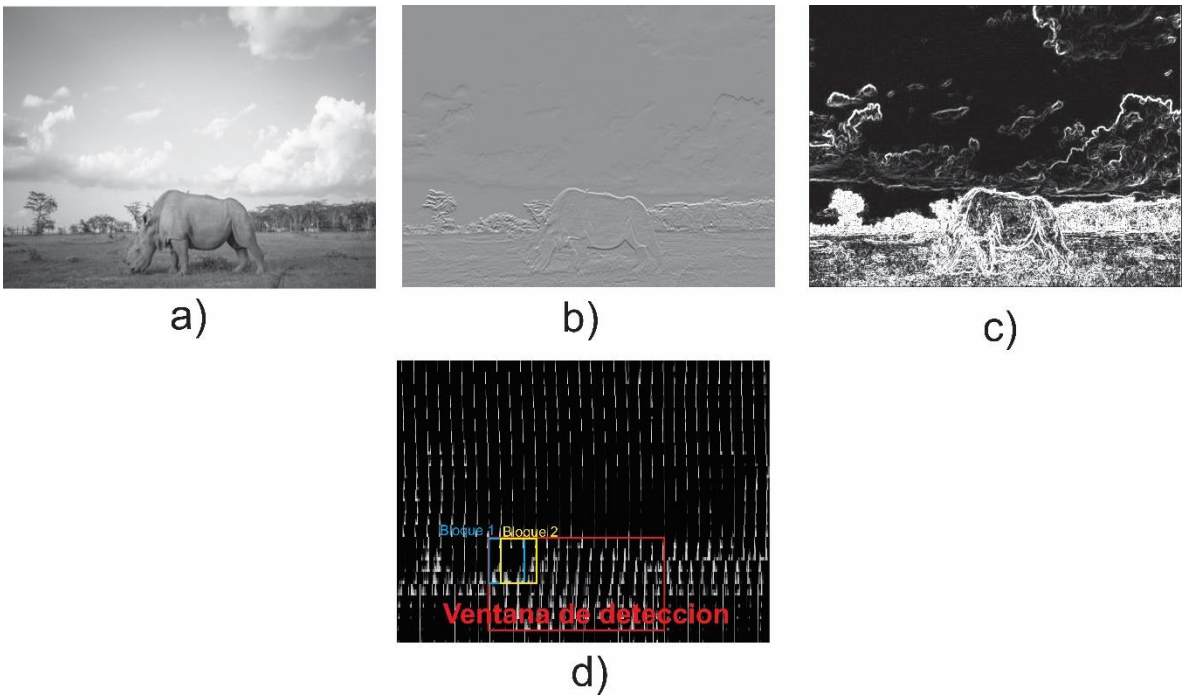

Figura 2.7. Fases de HOG, a) Imagen original, b) Gradiente de la imagen, c) Magnitud del gradiente, d) Orientación de gradiente y ventana de detección (con fines de visualización se modificó el contraste en la imagen b).

Considerando que esta investigación propone la implementación de una arquitectura hardware, y buscando optimizar recursos del dispositivo que será utilizado previo a la implementación del algoritmo HOG, la imagen a procesar será transformada en una imagen en tonos de grises (ver Figura 2.8).

A continuación, se explica detalladamente cada una de las fases que componen a HOG, dicha explicación se basará principalmente en los trabajos de García Ayax [23] y Navneet Dalal con Bill Triggs [29], sin embargo, se modificará el orden de alguna de las fases para facilitar su comprensión.

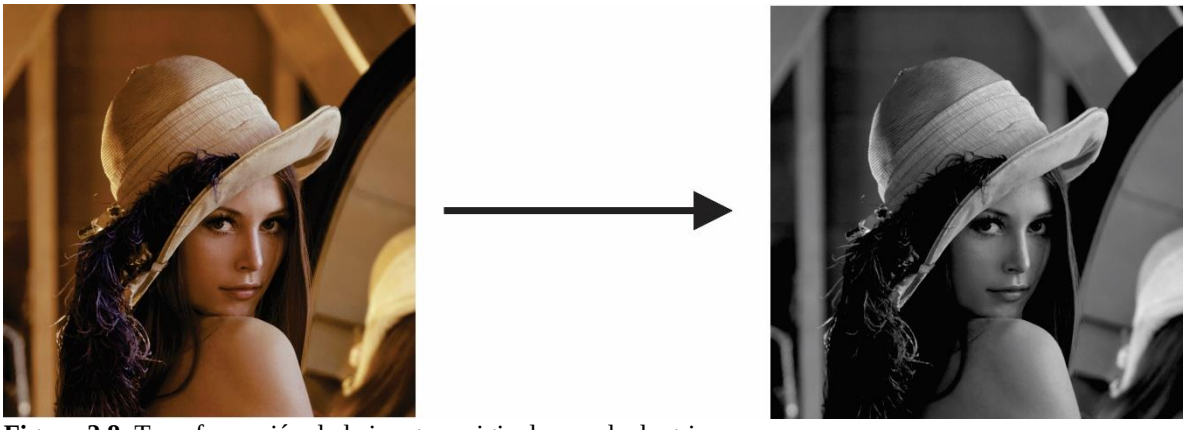

Figura 2.8. Transformación de la imagen original a escala de grises.

2.4.1 Cálculo de gradientes

Una imagen puede ser vista como una matriz $\boldsymbol{A} = [a_{i,j}]_{wxh}$, donde *w* y *h* representan el ancho y la altura de la imagen respectivamente, a representa el valor del i,j-ésimo píxel de la imagen, para $a \in \{0,1,2,\dots,2^b - 1\}$, donde b representa el número de bits utilizados para representar el valor de un píxel, para una imagen en escala de grises $b = 8$.

Se considerará una imagen de tamaño $N \times M$ píxeles, como la ilustrada en la Figura 2.9, para auxiliarse en la explicación de esta fase.

El cálculo de gradientes de una imagen se realiza mediante un proceso de convolución entre la imagen y un filtro. Para este propósito, el algoritmo HOG utiliza un operador derivador simple unidimensional del tipo $[-1,0,1]$, aplicado a cada píxel tanto de forma horizontal como de forma vertical. Es decir, considerando como centro al $a_{i,j}$-ésimo píxel se extrae un segmento de 3×3 píxeles de la imagen y se realizan las operaciones correspondientes con la máscara (ver figura 2.10). Este proceso se repite para todos los píxeles de la imagen.

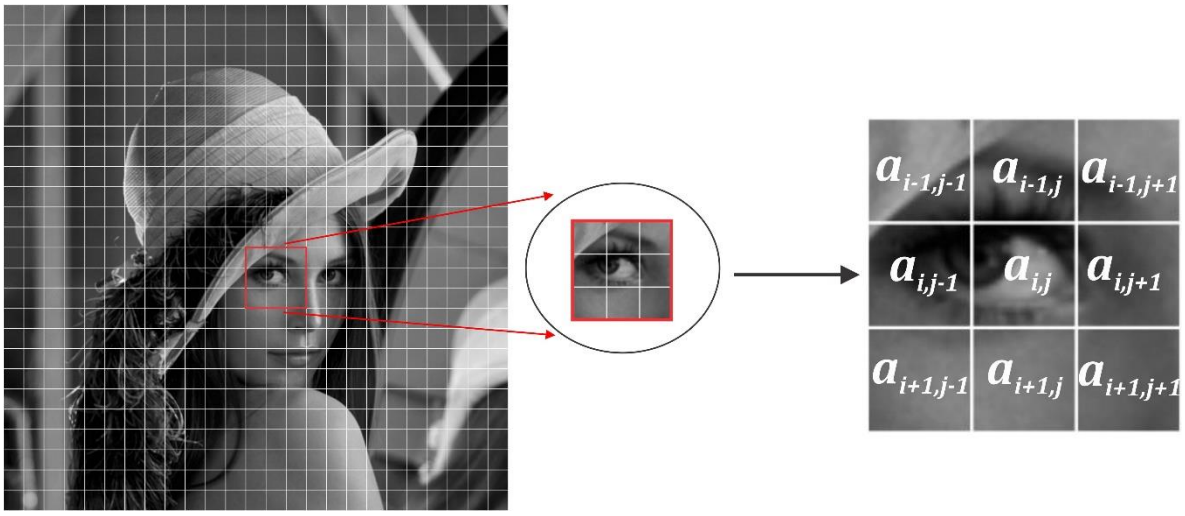

Figura 2.9. Seccionamiento de píxeles de una imagen y cuadro de cálculo de gradiente del $\boldsymbol{i,j}$-ésimo píxel.

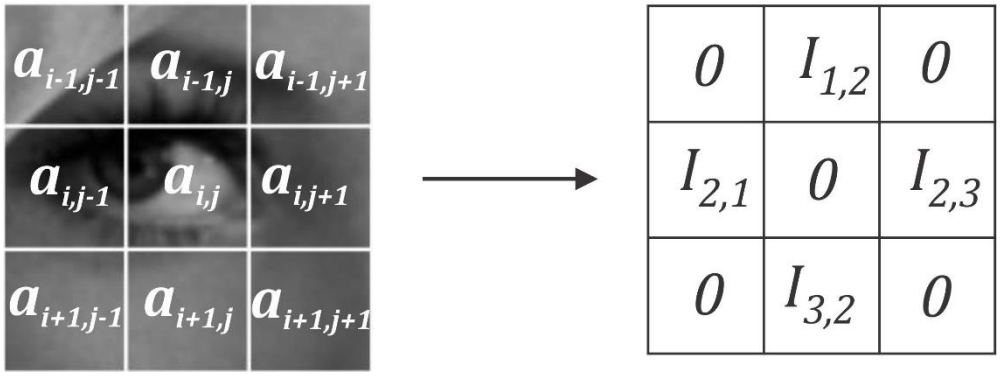

Figura 2.10. Enmascaramiento de píxeles para el cálculo del gradiente.

De esta forma, el cálculo de los gradientes mediante un operador derivador puede ser expresado de la siguiente forma:

$$\frac{\partial \boldsymbol{A}}{\partial x} = \left(a_{i,j+1}\right)\left(I_{2,3}\right) - \left(a_{i,j-1}\right)\left(I_{2,1}\right)$$
$$\frac{\partial \boldsymbol{A}}{\partial y} = \left(a_{i+1,j}\right)\left(I_{3,2}\right) - \left(a_{i-1,j}\right)\left(I_{1,2}\right) \quad (1)$$

Así, los gradientes en x y y están definidos por:

$$\boldsymbol{Gx}_{i,j} = \frac{\partial \boldsymbol{A}}{\partial x} = a(i+1,j) - a(i-1,j)$$
$$\boldsymbol{Gy}_{i,j} = \frac{\partial \boldsymbol{A}}{\partial y} = a(i,j+1) - a(i,j-1) \quad (2)$$

para $x = 1,2,\ldots,w$ y $y = 1,2,\ldots,h$, generando dos matrices, $\boldsymbol{Gx}$ y $\boldsymbol{Gy}$, de dimensiones $w \times h$. La figura 2.11 muestra un segmento de los $\boldsymbol{Gx}_{i,j}$ y $\boldsymbol{Gy}_{i,j}$, mientras que la Figura 2.12 muestra ambos gradientes completos.

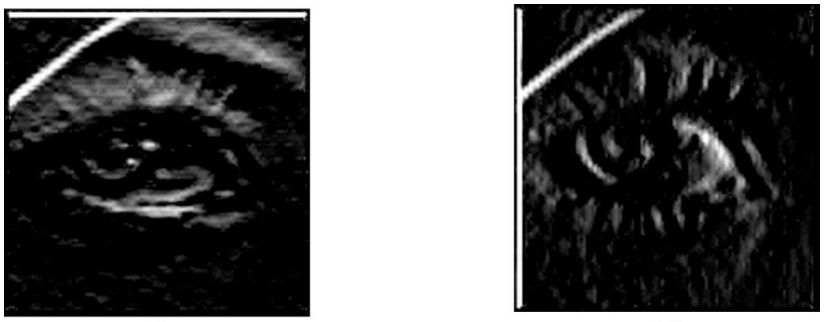

Figura 2.11. Gradiente en un área local, sobre la máscara. a) Gradiente en $\boldsymbol{x}$, b) Gradiente en $\boldsymbol{y}$.

Figura 2.12. Gradientes calculados sobre toda la imagen, a) Gradiente en $\boldsymbol{x}$, b) Gradiente en $\boldsymbol{y}$.

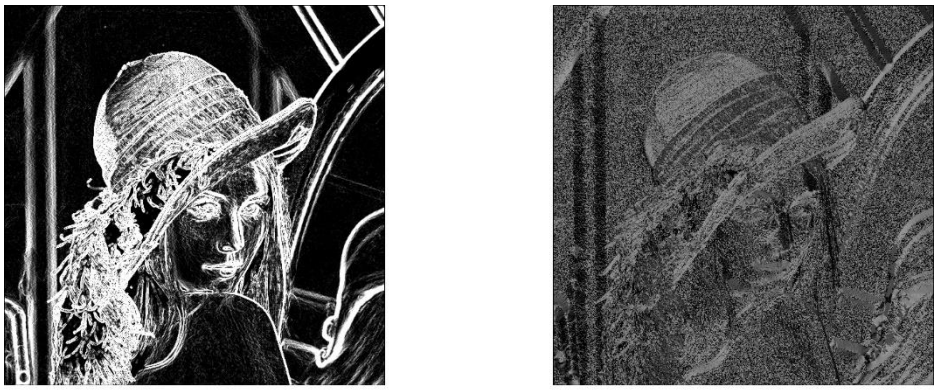

Figura 2.13. a) Magnitud de los gradientes de la imagen, b) Dirección de los gradientes de la imagen.

Finalmente, la magnitud y la dirección del gradiente son definidas por,

$$\boldsymbol{Mag}_{i,j} = \sqrt{\boldsymbol{Gx}_{i,j}{}^2 + \boldsymbol{Gy}_{i,j}{}^2} \tag{3}$$

$$\boldsymbol{\theta}_{i,j} = tan^{-1}\left(\frac{\boldsymbol{Gy}_{i,j}}{\boldsymbol{Gx}_{i,j}}\right) \tag{4}$$

Estas operaciones son puntuales y tanto $\boldsymbol{Mag}$ como $\boldsymbol{\theta}$ son matrices de dimensión $w \times h$. La Figura 2.13 muestra el resultado de este proceso.

2.4.2 Orientación del gradiente

El histograma de una imagen representa las distribuciones de cada nivel de intensidad en la misma, por lo tanto, sirve principalmente para verificar que filtro es adecuado implementar, qué preprocesamiento se le dará a la imagen, o, cómo es el caso de estudio que nos concierne, generar un vector de características de la misma. Por lo tanto, dada una imagen con valores de píxel en el rango $[0, L]$, si la profundidad de color es np, entonces $L = 2^{np} - 1$, su histograma es definido por la función discreta dada por:

$$h(r_k) = n_k \text{ Para } k = 0,1, \dots, L \tag{5}$$

donde r_k es el k-ésimo nivel de intensidad y n_k es el número de píxeles de la imagen que tiene el nivel de intensidad r_k.

A su vez $h(r_k)$ es definida como: $h(r_k) = \sum_{x,y} \delta(f(x,y) - r_k), \forall\, k = 0,1, \dots, L$ donde $f(x,y)$ es la intensidad del píxel en la posición (x,y) y entonces:

$$\delta(\alpha) = \begin{cases} 1, & \alpha = 0 \\ 0, & \text{en otro caso} \end{cases} \tag{6}$$

El histograma de una imagen se muestra de la Figura 2.14.

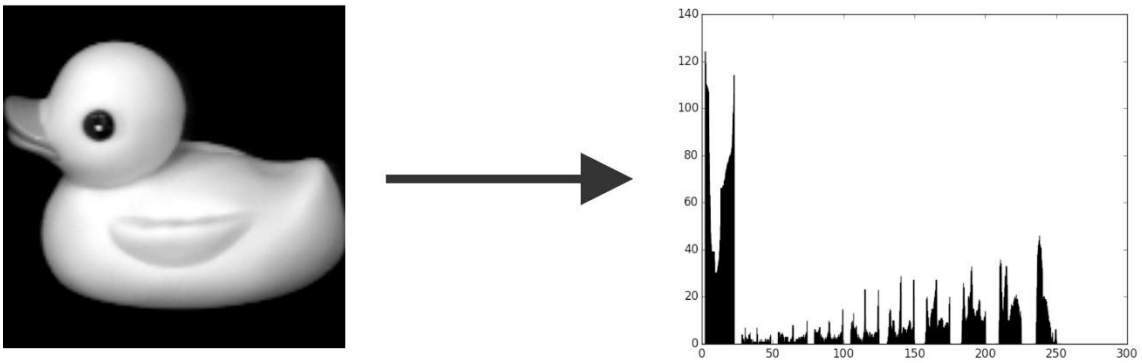

Figura 2.14. Histograma de una imagen en escala de grises.

En el algoritmo HOG, el histograma representa la distribución de la magnitud del gradiente en una celda, considerando la dirección del mismo (ver figura 2.15). En otras palabras, cada magnitud del gradiente representa un valor ponderado para el histograma de orientación de borde, basado en la orientación del gradiente de cada elemento. Cada valor de la magnitud se acumula dentro de un rango de contenedores (*bins*). En [29] se define que para el algoritmo HOG, solamente es necesario 9 contenedores por celda.

Los contenedores están espaciados uniformemente según el rango de la dirección del gradiente, esto es, sin signo se usa un rango de 0° a 180°, con signo el rango es de 0° a 360°. Los contenedores acumularán el valor de la magnitud del gradiente, sin embargo, también puede usarse el cuadrado de la magnitud, su raíz cuadrada, magnitud recordada, etc. En la construcción del histograma, durante el proceso de acumulación, los valores de la magnitud del gradiente son interpolados bilinealmente entre los centros de los contenedores vecinos, con el fin de reducir el *"aliasing"* entre valores (para el contexto en el que se implementa HOG el *"aliasing"* es la mala distribución de los valores dentro del histograma). La interpolación bilineal de la magnitud del gradiente se hace en función de la dirección del gradiente, p. ej., si se usa una dirección del gradiente sin signo y existen 9 contenedores, definidos en {10°, 30°, 40°, 70°, 90°, 110°, 130°, 150°, 170°}, al procesar un valor orientado en 15°, el 75% del valor de la magnitud será acumulado en el contenedor de 10° y el 25% en el contenedor de 30°.

Para realizar la distribución proporcional entre cada uno de los contenedores se consideran las matrices $\boldsymbol{Mag}$ y $\boldsymbol{\theta}$ que representa la magnitud de los gradientes y su dirección, respectivamente. De estas matrices se extraen las submatrices $\boldsymbol{cm}_{lk} = \left[c\_m_{ij}^{lk}\right]_{npx\times npy}^{cx\times cy}$ $\boldsymbol{cth}_{lk} = \left[c\_th_{ij}^{lk}\right]_{npx\times npy}^{cx\times cy}$, donde $cx = w/npx$ y $cy = h/npy$, que corresponden a una celda.

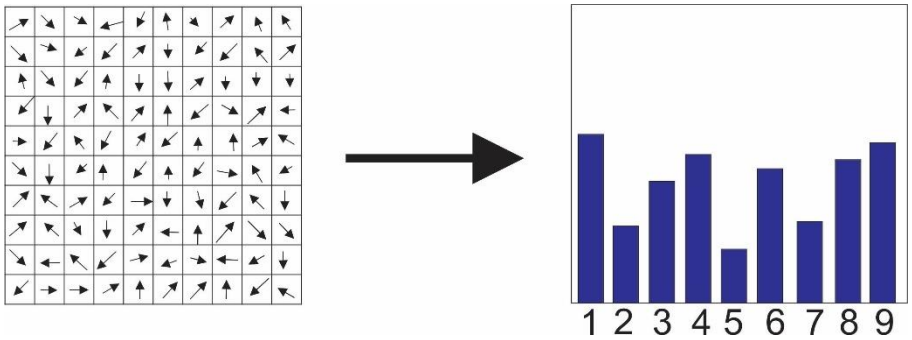

Figura 2.15. Direcciones en una celda de $\mathbf{10 \times 10}$ píxeles y su respectivo histograma.

A su vez se asocia un elemento $\boldsymbol{mc^{lk}} = [mc_o^{lk}]_p^{cx \times cy}$ que representa el vector de características de la celda correspondiente; p representa el número de contenedores existentes. La separación entre contenedores es representada por d_{bins}, y d_{o+1} es el contenedor superior al ángulo $c\_th_{ij}^{lk}$, por lo que se tiene la siguiente ecuación [23].

$$mc_o^{lk} = \begin{cases} mc_o^{lk} + (1 - \dfrac{c\_th_{ij}^{lk} - (d_{o+1} - 20)}{d_{bins}}) \cdot c\_m_{ij}^{lk}, & \text{si } bin_o < c\_th_{\text{ij}}^{lk} \\ mc_o^{lk} + c\_m_{\text{ij}}^{lk}, & \text{si } bin_o = c\_th_{ij}^{lk} \end{cases} \quad (7)$$

$$mc_{o+1}^{lk} = \begin{cases} mc_{o+1}^{lk} + (\dfrac{c\_th_{ij}^{lk} - (d_{o+1} - 20)}{d_{bins}}) \cdot c\_m_{ij}^{lk}, & \text{si } c\_th_{ij}^{lk} < bin_{o+1} \\ mc_{o+1}^{lk} + c\_m_{ij}^{lk}, & \text{si } bin_{o+1} = c\_th_{ij}^{lk} \end{cases}$$

donde, bin_o y bin_{o+1} son contenedores continuos que cumplen $bin_o \leq c\_th_{ij}^{lk} \leq bin_{o+1}$.

Para el contenedor inferior mc_o^{lk}, si el ángulo $c\_th_{ij}^{lk}$ es igual al contenedor inferior todo el peso del píxel $c\_m_{\text{ij}}^{lk}$ se distribuye al contenedor correspondiente. En caso contrario se aplica la distribución proporcional mediante una interpolación bilineal. Para el contenedor superior mc_{o+1}^{lk}, si el ángulo $c\_th_{ij}^{lk}$ es igual al contenedor superior todo el peso del píxel $c\_m_{\text{ij}}^{lk}$ se distribuye al contenedor correspondiente. En otro caso se aplica la distribución proporcional.

2.4.3 Venta de detección y bloque de características

En un sistema de visión clásico, la fase de segmentación cumple con la tarea de encontrar objetos dentro de una imagen y asilarlos para extraer sus características y clasificarlos en las fases posteriores. Para cumplir esta tarea, el algoritmo HOG implementa el uso ventanas de detección de tamaño variable, las cuales están conformadas por bloques, que a su vez se componen por celdas, y que son desplazadas por toda lo imagen.

Retomando la representación de una imagen en forma matricial $\boldsymbol{A} = \left[a_{i,j}\right]_{wxh}$, el proceso de ventaneo comienza después de la formación de los histogramas en cada una de las celdas que componen la imagen. Una celda es un subconjunto de $\boldsymbol{A}$ que forma una matriz de $npx \times npy$ píxeles, existiendo $ncx \times ncy$ celdas en la imagen, donde $ncx = w/npx$ y $ncy = h/npy$ donde w es el ancho de la imagen y h el alto en píxeles. Un conjunto de celdas compone a un bloque y a su vez un conjunto de bloques conforma la ventana de detección. Un bloque se define como $\boldsymbol{B}_{C_x,C_y}$, que representa una matriz de celdas de la imagen, donde $C_x = 1,2,..,ncx$ y $C_y = 1,2,..,ncy$, y definen las coordenadas de un bloque dentro de la ventana.

Concluido el cálculo de los histogramas locales se procede a formar las ventanas de detección, compuestas por bloques los cuales se utilizarán para generar el vector de características de la ventana (ver Figura 2.16). Frecuentemente, el desplazamiento de la ventana incluye bloques y celdas traslapadas, es decir una celda puede formar parte de más de 1 bloque y este puede formar parte de más de 1 ventana de detección.

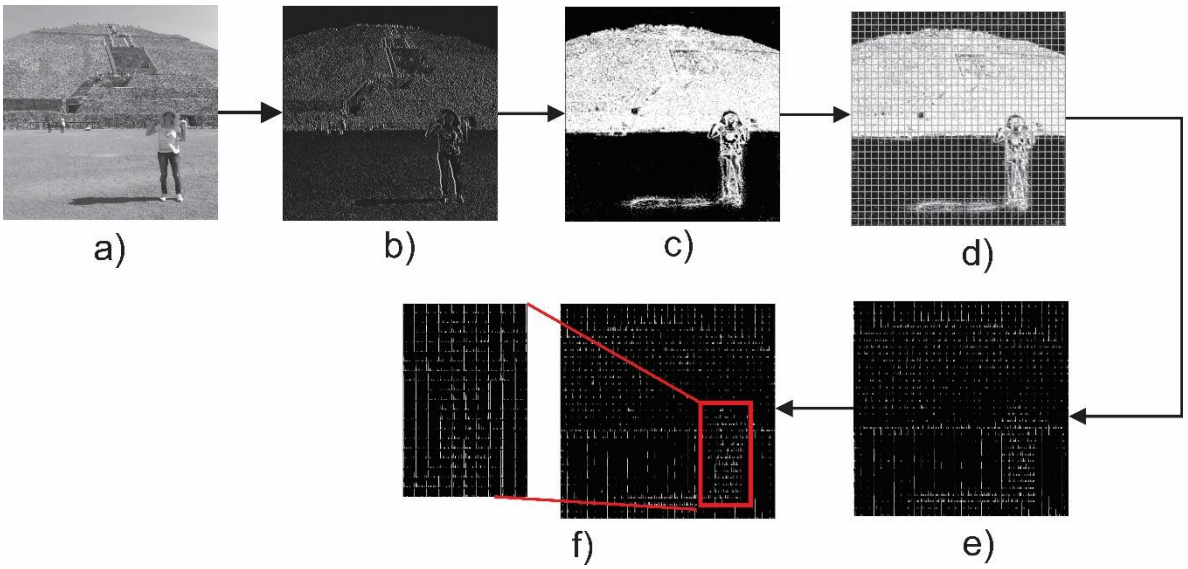

Figura 2.16. Fases de HOG (Por motivos de demostración se incrementó el contraste en la imagen). a) Imagen de prueba original. b) Gradiente en $\boldsymbol{x}$ de la imagen. c) Magnitud del gradiente. d) División de celdas, f) Histograma por celda, f) Ventana de detección.

El propósito de la ventana de detección es encontrar objetos específicos, que pueden ser de diferente tamaño, dentro de una imagen. Por lo tanto, la implementación inicia barriendo la imagen con una ventana formada por un conjunto pequeño de bloques, al concluir el barrido, el número de bloques de la ventana es incrementado y el proceso de barrido se repite. P. ej una ventana compuesta de $l \times m$ bloques, al terminar un proceso de barrido, su tamaño incrementara en $(l + a) \times (m + b)$ bloques, donde a y b es el incremento de tamaño tanto de largo y ancho de bloques en la ventana de detección para cada iteración. Los procesos de barrido están controlados por el usuario, esto es, el usuario determina el tamaño máximo que puede tener la ventana.

Como se describe en el trabajo de Dalal y Triggs, el algoritmo de HOG está diseñado para que el usuario determine el agrupamiento de píxeles óptimo según la aplicación en la que se desee implementar, es decir, la celda, el bloque y la ventana están determinados por los parámetros que defina el usuario.

El tamaño del bloque, definido por el número de celdas que contiene, y el traslape que los bloques pueden tener dentro de una ventana son los parámetros que determinan el número de bloques que serán formados en una ventana de detección (ver Figura 2.17), quedando definido por la ecuación 8.

$$B_{max} = \left(\frac{w}{npx * d_{Cx}}\right) \times \left(\frac{h}{npy * d_{Cy}}\right) \tag{8}$$

donde, B_{max} es el número máximo de bloques y d_{Cx}, d_{Cy} es la separación de los bloques en la ventada de detección en cantidad de celdas en x y en y respectivamente. Adicionalmente se tiene que cumplir que $d_{Cx}, d_{Cy} \in \mathbb{N} \mid \frac{w}{npx*d_{Cx}} \in \mathbb{N}\ y\ \frac{h}{npy*d_{Cy}} \in \mathbb{N}$.

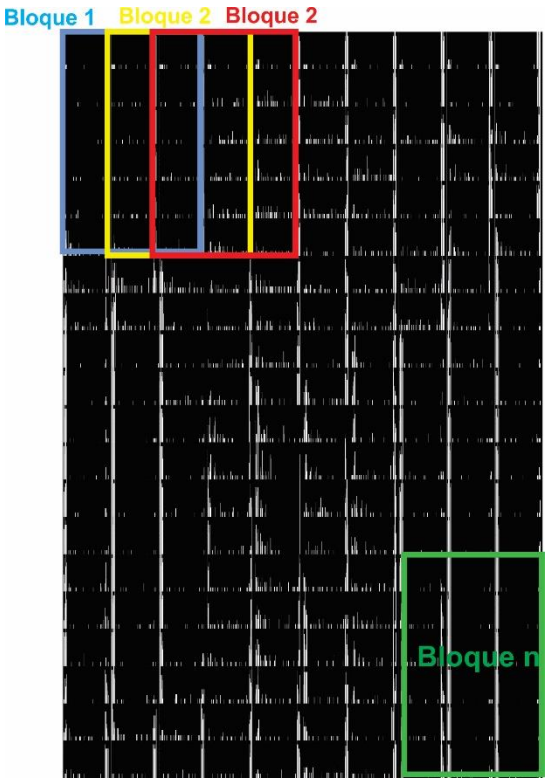

Figura 2.17. Generación del vector de características a partir de bloques en la ventana ***n***-ésima de detección.

Cada una de las celdas que componen al bloque tiene un conjunto de contenedores que almacenan la suma de las magnitudes del gradiente de una dirección específica. Es decir, los contenedores son los elementos que agrupan a los histogramas de gradientes orientados. Los contenedores de las celdas que componen a un bloque son concatenados en un vector parcial, obteniendo un vector por cada bloque que pertenece a la ventana de detección. Con la finalidad de disminuir la invariancia de iluminación, cada uno de estos vectores es normalizado utilizando los métodos descritos en el apartado 2.4.4. Los vectores parciales normalizados de los contenedores son concatenados para formar el vector de características de la ventana seleccionada. Por lo tanto, la dimensión del vector de características ($\mathbf{vc}_d$) está dado por $vc_d = \#Bloques\ x\ \#\ de\ celdas\ x\ \#Contenedores$. P. ej., al definir una ventana de 4×4 bloques formados por 2×2 celdas cada uno, considerando 9 contenedores, y sin traslape entre bloques, el vector de características consistirá en 576 elementos.

2.4.4 Normalización

Los valores del gradiente varían en un amplio rango debido a las variaciones locales de iluminación y al contraste con respecto al fondo. Para solventar este detalle al algoritmo HOG incluye una fase de normalización de los valores del gradiente a un rango {0,1}. Además, este proceso produce un mejor desempeño del algoritmo, resultando un proceso esencial de éste.

Dalal y Triggs evaluaron diversos esquemas de normalización, obteniendo los mejores resultados en cuatro de ellos. Cualquiera de estos esquemas puede ser aplicado al vector de características obtenido en la fase anterior. Los esquemas de normalización a los que se hace alusión son los siguientes,

$L2$-norm: $$\mathbf{v}' = \frac{\mathbf{v}}{\sqrt{||\mathbf{v}||_2^2 + \varepsilon^2}} \quad (9)$$

$L2 - Hys$: $L2$-norm, seguido por un recorte (se limita los valores de $\mathbf{v}$ a 0.2) y, por último, re-normalizar ($L2$-norm).

$L1$-norm:
$$\mathbf{v}' = \frac{\mathbf{v}}{||\mathbf{v}||_1 + \varepsilon} \tag{10}$$

$L1$-sqrt:
$$\mathbf{v}' = \sqrt{\frac{\mathbf{v}}{||\mathbf{v}||_1 + \varepsilon}} \tag{11}$$

donde $\mathbf{v}$ es el vector de características no normalizado, $||\mathbf{v}||_k$ su k-norma para $k = 1{,}2$ y ε una constante muy pequeña, usada para que no ocurra una indeterminación.

2.5 Red de neuronas con dendritas elipsoidales

En 1943 Warren McCulloch y Walter Pitt crean la primera neurona artificial implementando algoritmos matemáticos basados en supuestos del comportamiento neuronal del cerebro humano, contemplando el umbral de excitación de las neuronas y la unión de axones que estas contienen. [38].

A partir de ese momento, las redes neuronales artificiales han tenido diversas modificaciones, existiendo varios tipos y variantes, entre las cuales están, perceptrón multi-capa, los mapas de auto-organizados de Kohonen, las redes neuronales celulares, las redes neuronales de impulsos, las redes neuronales morfológicas, etc. Para la fase de clasificación del sistema propuesto en este trabajo de tesis, resulta de particular interés las redes neuronales con dendritas morfológicas (DMN, *Dendrite Morphological Neuron*) [39], más específicamente una DEN-*network* [9].

La red neuronal elegida para formar parte del sistema propuesto en esta tesis es una DEN-*network* [9], se trata de un modelo que basa su comportamiento en híper-dendritas elipsoidales. Este tipo de red requiere para su proceso de entrenamiento una base de datos procesada, donde los vectores de características estén ya definidos. Así, un vector de características de cada imagen determinara la dimensión de la neurona, es decir un vector de características de 2 elementos definirá una red neuronal de 2 dimensiones, un vector de características de 9 elementos, como es el caso del descriptor HOG, generará una red neuronal de dimensión 9.

La DEN-*network* se puede considerar la evolución natural de una DMN, que a su vez utiliza los principios y operaciones de una red basada en *Lattice Algebra.*

2.5.1 Lattice Algebra

La importancia de la red basada en el *Lattice Algebra* radica en la implementación de las estructuras dendríticas de las neuronas, pues de acuerdo con [40], [41] y [42], los árboles dendríticos que se encuentran unidos a las neuronas son primordiales pues ahí ocurre la mayoría de los procesos de la sinapsis. Por esto las dendritas pueden representar más del 50% de la membrana de procesamiento de la neurona, lo que implica que las dendritas son dispositivos informáticos elementales del cerebro y pueden funcionar como subunidades funcionales con capacidad de implementar operaciones lógicas.

De acuerdo con [42] para trasladar la propiedad de subprocesamiento las neuronas morfológicas resultan de gran importancia debido a que mientras las redes neuronales tradicionales implementan una acumulación múltiple y un umbral sobre un anillo $(\mathbb{R}, +, x)$ que este dado por,

$$\tau_j(\mathbf{x}) = \Sigma^n{}_{i=1} x_i \omega_{ij} - \theta_j \tag{12}$$

Donde $\mathbf{x} \in \mathbb{R}^n$, x_i denota valor de entrada de la i-ésima neurona, ω simboliza el peso sináptico entre la i-ésima y j-ésima neurona, θ_j representa el umbral de excitación de la j-ésima neurona y τ_j es el total del valor del indicador con respecto a la j-ésima neurona.

Al implementar una neurona morfológica se introducen operaciones de *Lattice Algebra* $\vee$ (máximo), o $\wedge$ (mínimo), además implementa el operando $+$ del semi-anillo $(\mathbb{R}_{-\infty}, \mathrm{V}, +)$ o $(\mathbb{R}_{-\infty}, \Lambda, +)$ donde $\mathbb{R}_{-\infty} = \mathbb{R} \cup \{-\infty\}$ y $\mathbb{R}_{\infty} = \mathbb{R} \cup \{\infty\}$. El computo de la neurona en una estructura morfológica que tiene un vector de entrada $\mathbf{x} = \{x_1, \dots, x_n\}$ está dado por,

$$\tau_j(\mathbf{x}) = p_j \Lambda^n_{i=1} r_{ij}\left(x_i + \omega_{ij}\right) \tag{13}$$

ó

$$\tau_j(\mathbf{x}) = p_j \mathrm{V}^n_{i=1} r_{ij}\left(x_i + \omega_{ij}\right) \tag{14}$$

donde $r_{ij} = \pm 1$ denota si la i-ésima neurona causa excitación o inhibición en la j-ésima neurona, y $p_j = \pm 1$ indica la respuesta de salida (excitación o inhibición) de la j-ésima neurona cuyos axones entran en contacto con las neuronas siguientes.

Contemplando a las dendritas como subunidades de procesamiento en las fibras terminales de la neurona la ecuación quedaría como,

$$\tau_k(\mathbf{x}) = p_k \Lambda_{i \epsilon I} \Lambda_{i \epsilon L} \tau^l{}_{ik}(x_i + \omega^l{}_{ik}) \tag{15}$$

I corresponde al conjunto de todas las neuronas de entrada con sus fibras terminales que hacen sinapsis con la k-ésima dendrita, mientras que L corresponde al conjunto de fibras terminales conectadas con la i-ésima neurona que hacen sinapsis con la k-ésima dendrita. $p_k \in \{-1,1\}$ y denota la respuesta de excitación o inhibición de la k-ésima dendrita. El valor $\tau_k(\mathbf{x})$ pasa por el cuerpo celular y el estado de la neurona *N* que está en función de las entradas recibidas por todas sus dendritas.

El subprocesamiento brindando por las neuronas ofrece información adicional que actúa como un filtro para la señal inhibidora o de excitación que mejora considerablemente el desempeño de la red. De esta forma la implementación de las dendritas implica directamente en la facilidad de resolver problemas linealmente no separables, como el problema XOR.

2.5.2 Red Neuronal Morfológica Basada en Híper-cajas

Como se presenta en [39] la DMN es modificada basándose en un modelo de híper-cajas para agrupar clases utilizando el principio de la ecuación (14), la expresión de salida está dada por,

$$\tau^j_k = \Lambda^n_{i=1}(\mathbf{x}_i + w^1_{ik})\Lambda - (\mathbf{x}_i + w^0_{ik}) \tag{16}$$

$\mathbf{x}_i$ es el vector de entrada y n es la dimensión del vector; $i \epsilon I$ y $I \epsilon \{1, \dots, n\}$ representa el conjunto de todas las neuronas de entrada, las cuales contienen fibras terminales, o sinapsis con todas las k dendritas; w^0_{ik} y w^1_{ik} representan los pesos sinápticos que corresponden al conjunto de las fibras terminales de la i-ésima neurona, lo que permite hacer sinapsis en la k-ésima dentrita para la j-ésima clase; w^1_{ik} es la fibra del terminal de activación y w^0_{ik} es la fibra del terminal de inhibición.

Para el entrenamiento de red DMN se tiene un conjunto de clases agrupadas en el espacio cuya dimensión es definida por el vector de entrada **x**. Para analizar lo mencionado se considerará un ejemplo donde el vector $\mathbf{x} \in \mathbb{R}^2$. La Figura 2.18 muestra la distribución de las clases en un espacio de 2 dimensiones.

El proceso inicia al seleccionar los patrones de una clase y crear un híper-cubo de tamaño tal que todos los elementos permanezcan dentro del híper-cubo. En dado caso que el híper-cubo generado abarque otra clase, se divide el híper-cubo en 2^n partes donde n representa la iteración correspondiente. Si se siguen abarcando elementos de otra clase se vuelve a dividir los híper-cubos. Este proceso se debe repetir hasta que solo se abarquen elementos de la clase correspondiente. Para una mejor agrupación y división de las cajas se aplica un algoritmo de evolución diferencial que permite adaptar de mejor forma la híper-caja al conjunto de datos que se tienen.

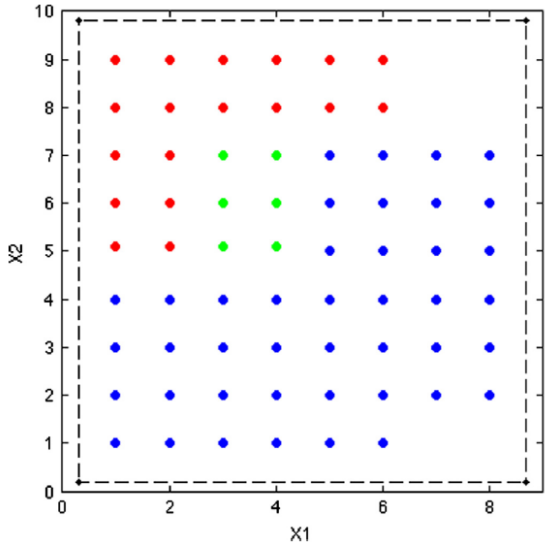

Figura 2.18. Conjunto de datos agrupados en el espacio, donde cada dato tiene 2 elementos [39].

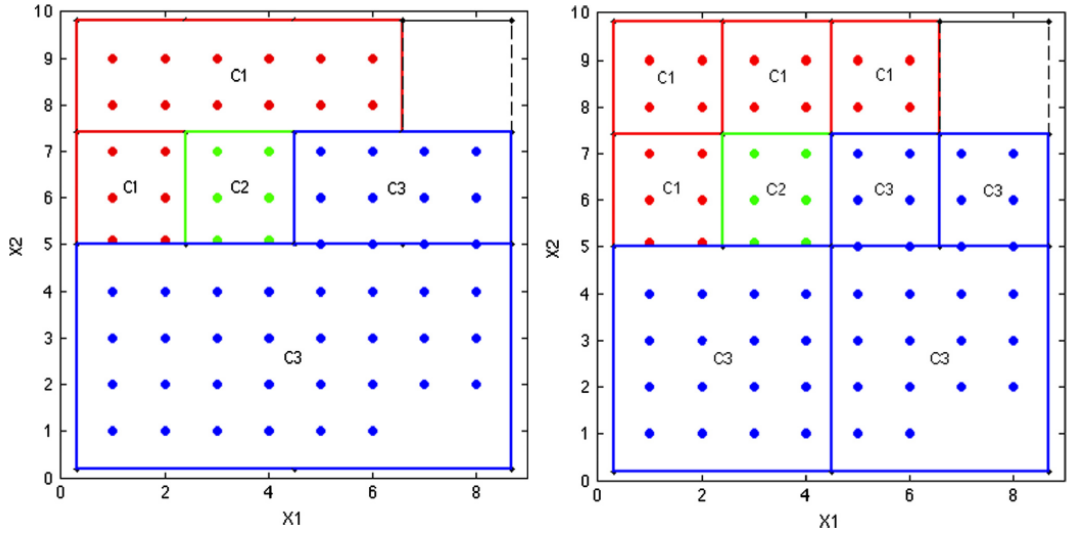

Figura 2.19. Agrupación y división de los híper-cubos [39].

Las coordenadas de los híper-cubos y los lados compartidos por estos definen los pesos sinápticos necesarios para la red.

2.5.3 DEN-*network*

De acuerdo con lo mencionado y lo observado en la Figura 2.19, una DMN fundamenta su funcionamiento en un conjunto de híper-cajas, cuya agrupación brinda a las dendritas información adicional de los elementos adyacentes, que a su vez incrementan la eficiencia de clasificación. Sin embargo, para casos donde los elementos de la base de datos son linealmente no separables la cantidad de híper-cajas aumenta considerablemente, como se puede observar en la Figura 2.20.

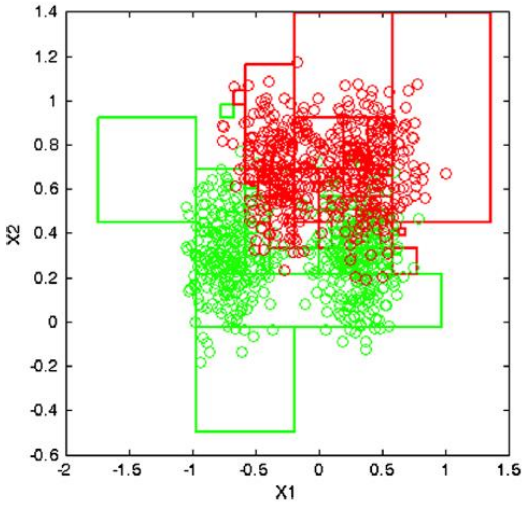

Figura 2.20. Híper-cajas formadas por la DMN [39].

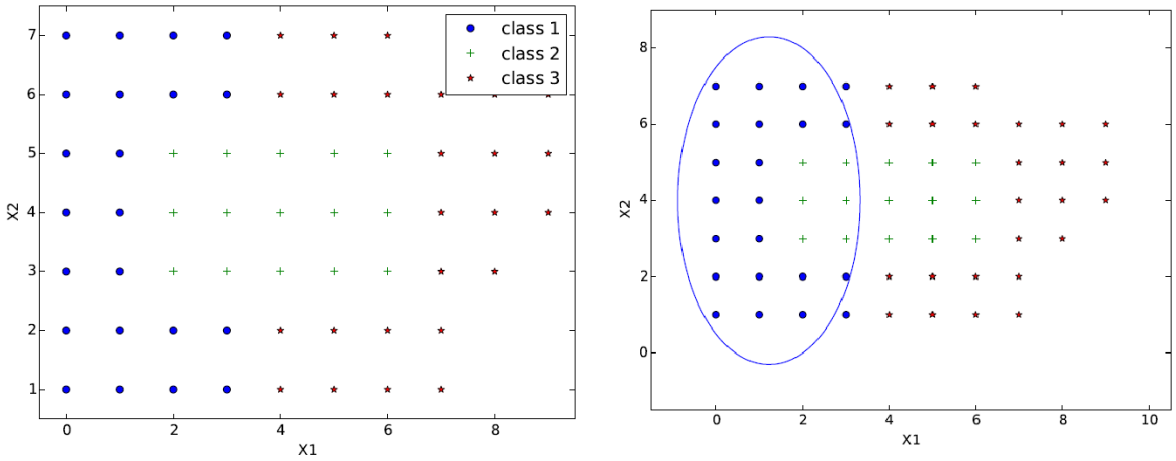

Figura 2.21. Creación del elipsoide dado un conjunto de datos [9].

Al cambiar el modelo de híper-cajas por un modelo de híper-elipsoides se obtiene una nueva neurona, la Neurona con Dendritas Elipsoidales. La DEN-*network* implementa el principio de agrupación de los elementos de una clase utilizando híper-elipses. Para agrupar los elementos de una clase en un elipsoide se toma en consideración la matriz de covarianza que pertenece a dichos elementos.

Así como el proceso de clasificación definido por la ecuación (15) toma en consideración las coordenadas cartesianas para los pesos sinápticos de las dendritas en la red, una DEN-*network* considera los aspectos geométricos de la híper-elipse, perímetro, área y distancia del centroide con respecto a un elemento. Para realizar la clasificación en la DEN-*network*, se requiere una ecuación que represente estos aspectos de la elipse. La distancia de Mahalanobis cumple estos requisitos y permite trabajar en el híper-espacio al permitir introducir vectores de entrada de *n* elementos. Entonces la operación de una DEN es definida como:

$$\tau_j = \left(\mathbf{x}_i - \boldsymbol{\mu}_j\right)^T \sum_j^{-1} \left(\mathbf{x}_i - \boldsymbol{\mu}_j\right) \tag{17}$$

donde $\boldsymbol{\mu}_j$ es el centroide de la hiper-elipse y $\boldsymbol{\Sigma}_j^{-1}$ representa la matriz inversa de covariancia asociada a la j-ésima neurona, que contiene toda la información de la elipse.

Al implementar la ecuación de la distancia de Mahalanobis en una híper-elipse se obtiene un valor constante que indica el grado de similitud de un vector de entrada con respecto a la matriz de covarianza de la neurona. Mientras este valor sea más pequeño indica que el vector de entrada tiene más similitud con el centroide de la neurona en cuestión. Por lo tanto, el vector será asignado a la neurona cuyo valor calculado sea más pequeño.

En la Figura 2.21 se puede apreciar que la elipse generada no agrupa a todos los elementos de la clase en cuestión y además contiene elementos de otra clase, lo que provoca un desempeño deficiente. Para resolver este problema se sigue un procedimiento similar al modelo de las hiper-cajas, dividir la clase en subgrupos para generar híper-elipses simétricamente más parecidas y que puedan ser adaptadas correctamente a clases linealmente no separables. El algoritmo de *Clustering k-means++* es utilizado para cumplir este fin, dividir una clase en subclases. El algoritmo genera *k* centroides, lo que divide la clase original en *k* subclases, cada subclase es evaluada y sus resultados son tomados en conjunto para determinar, de acuerdo con un parámetro introducido por el usuario, el cual hace referencia al error mínimo que la clase en cuestión puede

tener. Si el error ha disminuido, si es el caso, el algoritmo iterativo de la división de centroides termina, si no lo fuera, entonces la clase es ahora dividida en *k*+1 centroides. Cada subclase generada es considerada una unidad de subprocesamiento, referenciada como "dendrita".

De esta forma, el ajuste de las unidades de procesamiento permite una mejor relación de elementos para bases de datos cuyos elementos sean linealmente no separables (ver Figuras 2.22 y 2.23).

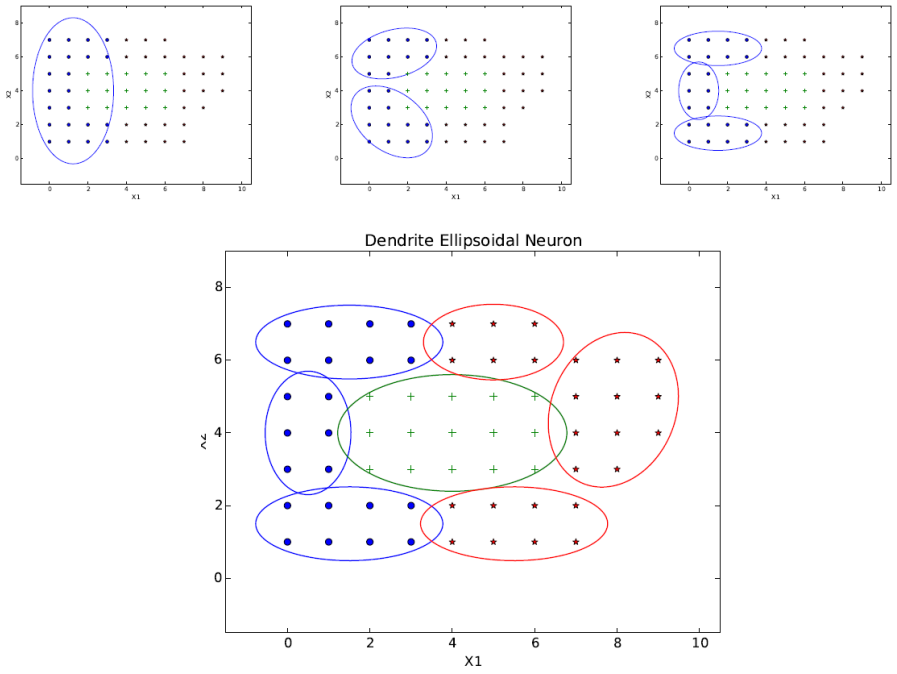

Figura 2.22. Construcción y división de las unidades de procesamiento de la red neuronal [9].

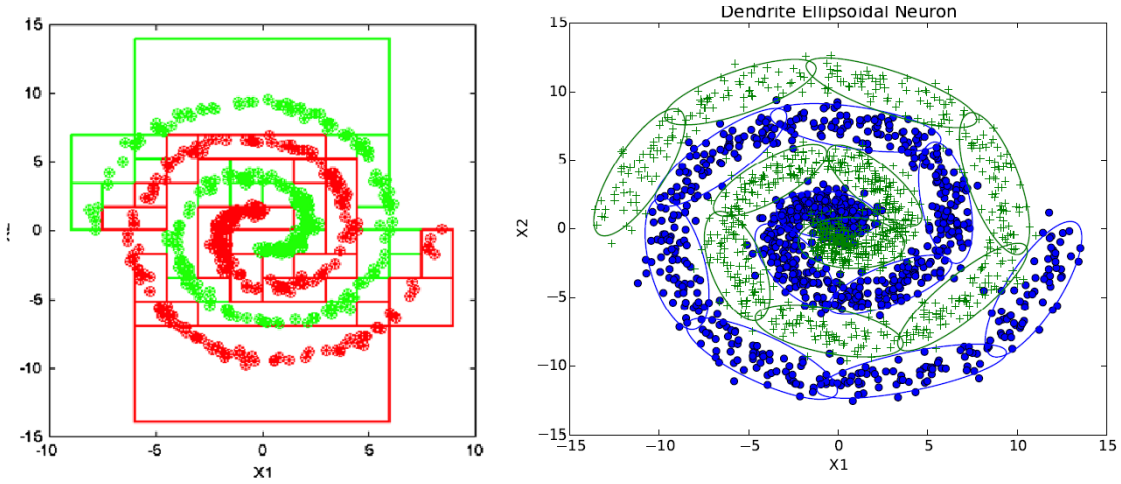

Figura 2.23. Comparación de formación de subunidades de procesamiento entre las redes DMN y DEN, [39] y [9].

Por ende, contemplando la modificación de *k* elipses la ecuación (17) quedaría como.

$$\tau_k^j = (\mathbf{x}_i - \boldsymbol{\mu}_k)^T \sum_k^{-1} (\mathbf{x}_i - \boldsymbol{\mu}_k) \tag{18}$$

donde $\boldsymbol{\mu}_k$ es el centroide de la hiper-elipse y $\boldsymbol{\Sigma_k^{-1}}$ representa la matriz inversa de covariancia asociada al k-ésimo clúster, donde k es el número máximo de *clusters* que pueden ser generados por la j-ésima neurona, y es un parámetro definido por el usuario. Cabe mencionar que al dividir una clase en subconjuntos se puede presentar el caso donde al obtener la matriz inversa de covarianza dé como resultado una indeterminación, para estos casos se opta por utilizar la semi-matriz de covarianza inversa de Moore-Penrose [43].

Para determinar cuál dendrita tiene el mejor desempeño se utiliza una función de umbral,

$$\tau_j = argmin_k(\tau_k^j) \tag{19}$$

donde $argmin_k$ solo selecciona la dendrita cuyo valor es menor. Y a su vez se determina el valor mínimo para cada clase.

$$\tau = argmin_j(\tau_j) \tag{20}$$

La Figura 2.24 ilustra este proceso en forma gráfica. Cada neurona está basada en un modelo elipsoidal que agrupa una clase, la distancia de Mahalanobis genera un valor escalar $\tau_k^j(\mathbf{x})$, si $\tau_k^j(\mathbf{x}) < 1$ el vector de entrada se encuentra dentro de la híper-elipse y si $\tau_k^j(\mathbf{x}) > 1$ se encuentra fuera de esta, es decir, el vector de entrada tiene mucha o poca similitud con la clase que representa la dendrita o elipse, respectivamente.

Una vez que la red está entrenada, un complemento que mejora su desempeño es la implementación de un algoritmo de evolución diferencial, el cual se realiza utilizando la siguiente ecuación

$$S(N) = \delta_a * a(N) + \delta_2 * (\delta_b * b(N) - \delta_c * c(N)) \tag{21}$$

donde a, b, c son redes neuronales (N) multiplicadas por un factor aleatorio diferente δ_a, δ_b, δ_c; δ_2 un factor aleatorio que multiplica la diferencia de b y c.

F representa la función original, es decir la función padre, *S* la función hijo y *err* el error al evaluar la red, entonces,

$$F(N) = \begin{cases} \text{si } F_{err} > S_{err}, & F = S \\ \text{si } F_{err} < S_{err}, & F = F \end{cases} \tag{22}$$

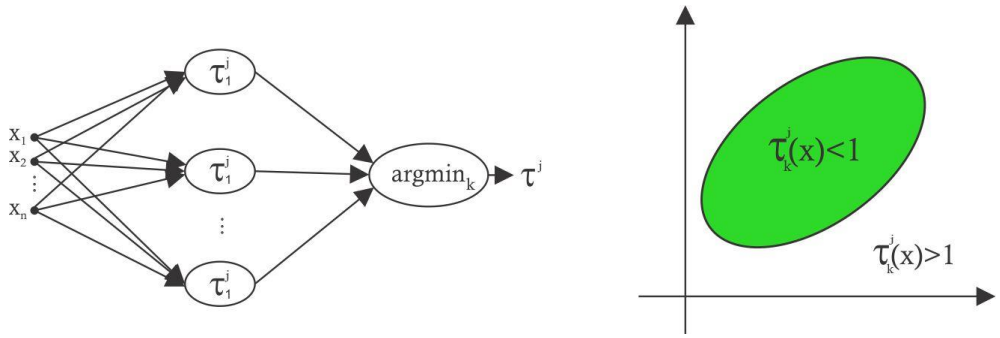

Figura 2.24. Esquema de una neurona y la relación de la distancia de Mahalanobis [9].

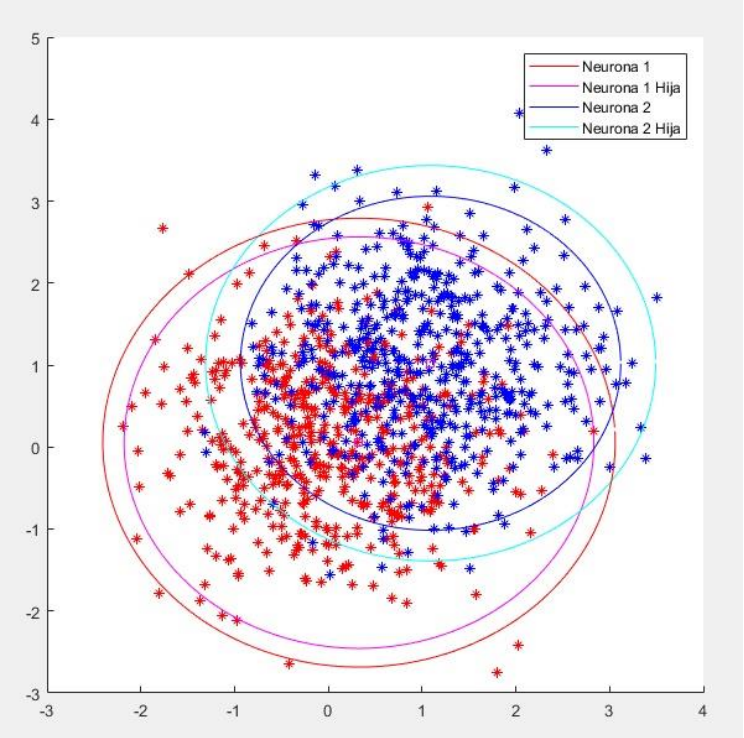

Figura 2.25. Representación de la aplicación del algoritmo de evolución diferencial al proceso de entrenamiento.

En la Figura 2.25 se puede apreciar como la aplicación del algoritmo de evolución diferencial modifica la forma de la elipse, y conserva aquella que hace que el desempeño general de la red sea el mejor. Para el ejemplo mostrado en la Figura 2.25, se puede apreciar que la neurona 1 se hizo más pequeña, mientras que la neurona 2 expandió su tamaño, de esta forma el desempeño de la red, para una base de datos genérica con 1000 elementos para la parte de entrenamiento y 200 para la parte de testeo se obtuvo una mejora del desempeño del 23% al 21.9%.

De acuerdo con los datos reportados en [9], las modificaciones hechas a la red generan 2 resultados significativos, primero, mejora el desempeño de la red y, segundo, la implementación incluye menos subunidades de procesamiento. Ver Tabla 2.1 donde la red DMN y sus modificaciones son denominadas DMN-R y DMN-P respectivamente.

Tabla 2.1. Comparación entre redes neuronales aplicadas a diferentes bases de datos [9].

	DMN-R		DMN-P			DEN		
Dataset	$\#Dendrites$	E_{test}	$\#Dendrites$	E_{train}	E_{test}	$\#Dendrites$	E_{train}	E_{test}
Iris	5	6.7	28	0.0	3.3	3	1.0	0.0
Mammographic	51	14.4	26	0.0	19.2	16	19.0	12.0
Liver Disorders	41	42.0	183	0.0	35.5	14	19.0	28.0
Glass Identification	60	36.7	82	0.0	31.8	7	0.0	9.0
Wine Quality	120	51.0	481	0.0	42.1	60	56.0	55.0
Mice Protein Expression	77	18.9	809	0.0	5.0	8	0.0	0.0
Hepatitis	19	53.3	49	0.0	46.7	7	40.0	20.0
NAO	231	13.3	74	0.0	2.2	8	6.0	2.0

El hecho de que la sencillez sea una característica de la organización de las DEN-*network* y que la estructura de sus unidades de procesamiento es simple, hace que su implementación no requiera gran cantidad de recursos. Además, las operaciones que ejecuta para cumplir su función tienen un grado bajo de complejidad computacional, generando tiempos de propagación reducidos, lo que incrementa la velocidad de procesamiento.

2.6 Unidad de procesamiento

2.6.1 Estructura de un FPGA

El elemento central de procesamiento del sistema propuesto en esta investigación es un FPGA, un dispositivo conformado por una matriz de bloques lógicos que pueden ser interconectados entre sí a través de interconexiones programables con la finalidad de implementar funciones complejas. Los FPGA tienen la particularidad de poder ser reconfigurados.

Un FPGA puede ser considerado como la evolución natural de los dispositivos lógicos programable (PLD, *Programmable Logic Devices*) y de los circuitos integrados de aplicación específica (ASIC, *Application-Specific Integrated Circuits*). Aunque los PLD´s eran dispositivos reconfigurables, su arquitectura no estaba diseñada para abarcar sistemas complejos. Los ASIC´s surgieron como una alternativa que podía satisfacer esta limitante; además, estos dispositivos son altamente fiables y tienen la mejor relación área-velocidad-consumo, es decir al realizar un sistema digital en un circuito integrado, el implementado en un ASIC será el que menos área de semiconductor requiera, el que realice la función en menos tiempo y el que menos corriente requerirá para funcionar. Sin embargo, los ASIC´s no son dispositivos que puedan reconfigurarse, tienen un alto costo y su diseño es complejo y en demasía tardado. Existen varios tipos de ASIC´s, entre éstos están los ASIC´s basados en arreglos de compuertas (*gate arrays*), que formados por filas de transistores sin interconexión y entre cada fila posee canales de ruteo. La idea central de estos ASIC´s es que, con 4 transistores e interconexiones puede construirse cualquier puerta lógica, los canales de ruteo son usados para conectar estas compuertas entre sí y conseguir funciones de mayor complejidad. En el año 1984 Ross Freeman y Bernard Vonderschmitt, cofundadores de la compañía Xilinx Inc., tuvieron la idea de desarrollar un dispositivo basándose en los arreglos de compuertas de los ASIC's, y que además los elementos que lo formaran fueran reconfigurables. Es así como 1985 la compañía Xilins lanzó al mercado su primer FPGA, el XC2064 formado por un arreglo de 64 CLB's, cada uno integrado por 2 LUT's de 3 entradas, con una equivalencia de 10000 compuertas lógicas [44].

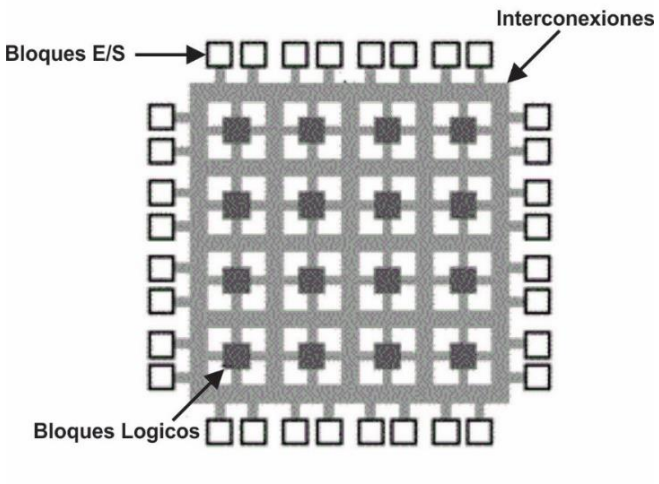

Figura 2.26. Estructura interna de un FPGA.

La Figura 2.26 muestra el esquema general de la estructura de un FPGA. Este dispositivo está integrado por los siguientes elementos,

- Bloques lógicos configurables (CLB, *Configurable Logic Block*). Los CLB´s son los recursos utilizados para implementar funciones lógicas y tienen una distribución matricial en el FPGA. Un CLB está compuesto principalmente por tablas de búsqueda (LUT, *Look-Up Table*) las cuales se combinan con un conjunto de biestables y elementos de control.
- Bloques de entrada/salida (*IOB, Input-Output Blocks*). Los IOB´s representan los recursos que permiten a las funciones implementadas en el FPGA interactuar con el exterior. Se componen principalmente por buffers de tercer estado, registros, latches, elementos de retardo, elementos de control de *skew*, entre otros.
- Recursos de interconexión. Se trata de recursos que permiten interconectar los CLB's entre sí para la formación de funciones complejas. Además, también cumplen la tarea de conectar CLB's o las funciones implementadas a partir de ellos con los bloques de entrada/salida.
- Elementos embebidos. Son elementos adicionales incluidos en el FPGA que realizan funciones específicas y que son de arquitectura fija. Aquellas funciones que requieren grades recursos para ser implementadas o que son frecuentemente requeridas en diversas aplicaciones son susceptibles para ser incluidas en un FPGA como un elemento embebido de arquitectura fija. Elementos que cumples estos requisitos y que comúnmente son incluidos en un FPGA son los DCM's, bloques RAM, multiplicadores y procesadores.

2.6.2 FPGA y GPU

Uno de los principales exponentes del procesamiento de imágenes es la unidad de procesamiento grafico (GPU, *Graphics Processing Unit*). Generalmente el trabajo de las tareas más intensivas de una CPU es desviado al GPU lo que conlleva que la tarea en cuestión se resuelva más rápida y eficientemente. Esto se debe principalmente a la composición que tiene un GPU respecto a un CPU, mientras que el CPU contiene una serie limitada de núcleos diseñados para administrar y operar diferentes tareas de forma secuencial, el GPU tiene una gran cantidad de núcleos conectados en paralelo capaces de ejecutar una tarea específica. Lo que brinda una alta eficiencia en operaciones relacionadas con imágenes [45].

Por otro lado, en los últimos años las aplicaciones que tienen los FPGA han abarcado el área de procesamiento de imagen, pues brinda una ventaja competitiva en tareas donde se requiere,

- Calculo con estructuras de control simple y regulables.
- Número de operaciones por muestra elevado.
- Las implementaciones aritméticas básicas son realizadas directamente en Hardware.
- En la existencia de grandes cantidades de datos entrada/salida procesados en un lapso de tiempo corto.
- Múltiples tareas en forma concurrente.
- Su arquitectura es fácilmente adaptable, así como su portabilidad.

Como se expone en [46], cuando se presentan métodos de cálculo complejos, donde se requiere la manipulación, operación y envió de gran cantidad datos el FPGA presenta una ventaja ante el GPU, mientras que el GPU muestra una superioridad al implementar cálculos simples. De igual manera para la implementación de las operaciones de punto flotante resultan más fáciles para el GPU y de punto fijo para el FPGA [47], [48].

En [49], [48] se concluye que, cuando se tiene una gran cantidad de E/S y operaciones MIMO, el GPU parece ser más sensible en la relación unidades de procesamiento - operaciones de computo, pues mientras más cantidad de elementos se requieran para realizar un procesamiento, su desempeño decae considerablemente. En cambio, los FPGA poseen una flexibilidad considerable en la implementación de algoritmos que utilizan estructuras de E/S personalizadas, permitiendo resolver operaciones MIMO en una forma más fácil.

Otro aspecto importante que considerar es la complejidad al implementar los algoritmos, pues mientras que en el GPU resulta relativamente fácil y rápido programarlos por su naturaleza de diseño en software, en el FPGA el programar los algoritmos consume más tiempo pues es una tarea de diseño en hardware [46], [47], [48].

Finalmente, cabe resaltar que de acuerdo con Fowers *et al*., la aplicación de algoritmos complejos que requieren una gran cantidad de operaciones sufren un mayor consumo de potencia en dispositivos que contienen multinúcleos [49]. Por otro lado, los FPGA proporcionan la mejor eficiencia de consumo de potencia en situaciones de procesamiento de imagen dada su arquitectura.

2.7 Estado del arte

La visión artificial y las redes neuronales son dos ramas de la ciencia que constantemente se vinculan en el desarrollo de sistemas complejos. Por su parte, las redes neuronales han sido desarrolladas desde la década de los 40’s, a partir de la primera red neuronal creada por Warren McCulloch y Walter Pitt, mientras que el primer sistema de visión fue concebido por Lawrence Roberts en la década de los 60’s, desde entonces han existido numerosos avances, algunos de los trabajos más relevantes y que tienen relación directa con el propuesto en esta investigación son brevemente descritos a continuación,

Bauer *et al.* presentaron un sistema híbrido para la detección de peatones [50]. El sistema utiliza un FPGA para implementar la fase de extracción de características y un GPU administrado por un CPU para implementar la fase de clasificación. HOG fue el algoritmo de extracción de características implementado en el FPGA, los autores propusieron métodos que simplifican los cálculos de la dirección del gradiente. Utilizaron dos dispositivos de la familia Spartan3, de la compañía Xilinx Inc., el XC3S-2000 y el XC3S-4000. El primer FPGA ofrece interfaces que

facilita la transferencia de datos con una cámara, mientras que del segundo se aprovecha su memoria integrada. La fase de clasificación fue implementada en un GPU y el algoritmo elegido fue la máquina de vectores de soporte (SVM, *Support Vector Machines*). El GPU es administrado por un CPU, el cual recibe el descriptor HOG proveniente del FPGA y lo pone disponible para el GPU para que este cumpla la tarea de clasificación.

En [51], se presenta una implementación del descriptor HOG en un FPGA que es utilizada en un sistema de reconocimiento peatonal. Buscando tener un uso más eficiente de los recursos del FPGA, los autores implementaron una versión simplificada del algoritmo HOG sin afectar su eficiencia y robustez. Normalmente, para obtener la dirección y la magnitud del gradiente son necesarios los cálculos de una raíz cuadrada y del arco tangente. Para simplificar la arquitectura de hardware, la raíz se realizó mediante una LUT y el arco tangente mediante un esquema de interpolación. En la agrupación del histograma se tomó en cuenta un valor constante para la distribución de valores en los contenedores. Para la normalización se tomó en cuenta la norma L2-*norm* y se realizó una aproximación de 2^n, y si n es impar entonces $\sqrt{2}$ se simplifica como 1.375. Se utilizó aritmética de punto fijo. La implementación física del sistema fue realizada en una tarjeta de la compañía Altera, que incluye a un FPGA Stratix II. El sistema propuesto puede procesar 30 fps (*frames per second*) para una calidad de video VGA. Adicionalmente, para probar el sistema propuesto, se implementó una fase de reconocimiento, mediante un SVM, en una computadora personal.

En el trabajo presentado por Yao *et al.*, se implementó el descriptor SIFT en un FPGA, este descriptor al igual que HOG basa su funcionamiento en el uso de características locales codificadas en un descriptor. Para simplificar la arquitectura hardware, los autores proponen dividir la imagen al aplicar un filtro gaussiano y aprovechar la concurrencia en el FPGA. Además, algunas operaciones fueron simplificadas mediante simples acumuladores y filtros, posteriormente se envían a 4 buffers FIFO que contiene la partición de la imagen para poder ser procesada [52]. La implementación utiliza aritmética de punto fijo, lo que reduce significativamente los recursos de hardware requeridos. Con la optimización que se propuso y utilizando como clasificador el microprocesador MicroBlaze incluido en el mismo FPGA, se pudo procesar una imagen de 640×480 píxeles en 31 ms.

En el año 2010 Farabet *et al.*, dando continuidad al trabajo presentado en [53], propusieron un sistema que utiliza una técnica de aceleración por hardware para modelar una red neuronal convoluciónal denominada ConvNets (*Convolutional Networks*), con el propósito final de aplicarlo en sistemas de visión artificial [54]. El dispositivo FPGA Virtex-4 SX35 de Xilinx fue elegido para albergar el sistema propuesto. Este modelo puede utilizarse para acelerar la ejecución y aprendizaje de algoritmos como SIFT y HMAX. El modelo se basa en usar el CPU, que incorpora la tarjeta de Xilinx, como una unidad de control de propósito general para administrar los procesos de filtrado y que, a través bus de configuración global, un bus de configuración global permite modificar los parámetros del sistema en tiempo de ejecución. Se hace uso de esta propiedad en el mapeo de datos en dos dimensiones que se almacena en la memoria externa, y a su vez permite realizar una operación de convolución en los bloques de entrada. Dada la naturaleza de la red es necesario implementar una arquitectura pipeline que permita facilitar el cálculo del filtro de convolución en las diferentes capas de la red, por lo que se utilizan las propiedades de paralelismo que ofrece el FPGA. De esta forma se logra un mayor porcentaje de eficiencia entre la capacidad de procesamiento y el consumo de energía; con un filtro de 19×19 píxeles se puede procesar aproximadamente 100 fps con un consumo de 15 W con respecto al CPU que con el mismo tamaño de filtro es capaz de procesar cerca de 10 fps con un consumo energético de 90 W.

Otra versión simplificada del algoritmo HOG fue presentada en [55], donde se implementó un sistema binarizado para la detección de humanos mediante tarjeta FPGA Virtex-5 VLX-50 FPGA de la compañía Xilinx. Previo al desarrollo de las fases de HOG se tiene como elemento de entrada una imagen RGB, por lo que, de acuerdo con su composición se le aplica una función de luminancia a cada píxel. Esta función tiene como objetivo tomar el promedio de valores máximos y mínimos, para que el valor resultante se trunque a un valor de 8 bits de parte entera. Una vez calculada la diferencia de los píxeles en el cálculo de los gradientes se expresan en valor absoluto, con el objetivo de seguir teniendo una variable de 8 bits de parte entera. En el cálculo de la magnitud de los gradientes se hace uso de aritmética sin signo; además, mediante el CORE Generator de Xilinx se realiza la operación de suma de cuadrados y su raíz cuadrada, por lo que el valor truncado máximo se encuentra en 9 bits de parte entera. Para evitar el cálculo complejo del arcotangente, se ocupan aproximaciones lineales por partes que distribuyen información en los contenedores del histograma a partir de los valores de luminancia. Esta distribución genera histogramas parciales, y mediante una arquitectura pipeline y el uso de bloques RAM se genera un histograma completo. En la normalización del histograma se implementó una aritmética de punto fijo, donde la división se realiza con una aproximación por desplazamientos de 2^n. Adicionalmente al método de HOG se le agrega una fase de binarización, donde al vector de salida se aplica una función cuyo umbral se encuentra en 0.08, valor obtenido de los experimentos realizados por los autores. Por ende, dada una imagen de entrada de 320×240 píxeles se crea una ventada de detección compuesta por 50×105 valores de luminancia utilizando el clasificador AdaBoost se obtiene una eficiencia del 96.6 % para la detección de personas con respecto al algoritmo original con una eficiencia de 80.9 %.

En Chen *et al*., con el propósito de tener un menor tiempo de respuesta y buscando hacer un uso eficiente de los recursos en una arquitectura hardware, propusieron una versión simplificada del algoritmo HOG para la detección de personas [56]. Este proceso se realizó en un FPGA de gama baja, Stratix II de la compañía Altera. Debido a que la obtención de los gradientes es una tarea que no conlleva una gran complejidad de cálculo, lo más resaltable de este trabajo se presenta en las fases de cálculo de magnitud, dirección y normalización. En el cálculo de la magnitud se opta por realizar una aproximación de la raíz cuadrada mediante una arquitectura pipeline que utiliza unidades de cambio para evitar las operaciones de multiplicación y optimizar la cantidad de recursos utilizados. En el cálculo de la dirección se sustituye la función arcotangente por aproximaciones lineales por partes correspondientes a cada contenedor del histograma, cuyo valor de entrada depende de la asociación de los gradientes y la función tangente; para el cálculo de la tangente utilizan una aproximación mediante el algoritmo de Volder. Finalmente, y debido a que la normalización hace uso de la división, la cual conlleva un alto grado de complejidad y que repercute directamente en los recursos utilizados del FPGA, se opta por realizar este proceso a través de una aproximación de Newton-Raphson. La parte de clasificación implementada mediante un SVM, obtuvo una eficiencia del 95.27 % para la base de datos INRA y un 99.37 % para una base de datos de personas del MIT.

En el año 2015 Zhang *et al.*, presentaron un modelo de optimización para una red neuronal convoluciónal que incrementa la velocidad de procesamiento mediante el uso de múltiples capas, paralelismo y la ventaja de un "sistema en un chip" (SoC, *System On a Chip*), aprovechando las características ofrecidas por un FPGA VC707 Virtex7 485t de Xilinx [57]. El sistema recibe una imagen RGB de 256×256 píxeles, es decir se tienen 3 matrices de 256×256, las cuales son transferidas a una capa de procesamiento para generar un mapa de características. La aceleración en los cálculos se basa en dos mejoras de optimización, el acceso a la memoria y la integración de multicapas. Para la optimización de la memoria se utiliza una combinación de memorias DDR

y DRAM las cuales actúan como buffers temporales en una arquitectura pipeline y que son accesadas mediante un bus común para que las todas las capas tengan disponible la información de los mapas de características. Para la estructura multicapa se implementa el uso de la recursividad para reducir los recursos del FPGA, además de la inclusión de las funciones de aceleración, que permiten hacer uso del paralelismo en los comparadores que se necesitan entre cada capa de la red, por lo que en combinación con las conexiones de la memoria el proceso de cálculo se ve acelerado. Esta implementación da como resultado la posibilidad de realizar 61.62 giga operaciones por segundo lo que sumado a su bajo consumo de potencia tiene un mayor grado de eficiencia comparado a un procesador Intel Xenon 2.20 GHz.

Un trabajo interesante fue propuesto por Rettkowski *et al*., donde se implementó un sistema de reconocimiento de personas utilizando HOG para la parte de extracción de características y un SVM o AdaBoost para la parte de clasificación, el sistema tuvo como objetivo ser aplicado en la rama de vehículos autónomos y detección de personas [58]. El desarrollo de este sistema se realizó en el dispositivo de gama alta Zynq de la compañía Xilinx, el cual contiene una arquitectura SoC, pues se compone de un procesador ARM y un FPGA. El trabajo descrito se basa en tres enfoques, software, hardware y software/hardware. Para los 3 enfoques las condiciones de entrada son una imagen RGB que debe ser transformarla al espacio de escala de grises. En el primer enfoque, se hace uso completo del procesador utilizando el sistema operativo Linux que permite hacer uso de la librería OpenCV para modificar las propiedades de la imagen; tamaño y espacio de color, además de hacer uso de una función que incorpora la estructura de un SVM. Adicionalmente, se implementan funciones de C++ en el desarrollo de las fases de HOG. La implementación en software permite procesar una imagen de 1920×1080 píxeles en 12.7 segundos. Al utilizando una función de escalamiento se reduce la imagen a 322×161 píxeles, por ende, el tiempo de procesamiento se redujo a 0.051 segundos. En el enfoque de software/hardware se utilizó el entorno de desarrollo SDSoC 2015.4 de Xilinx que permite hacer uso de funciones de aceleración, las cuales tienen mayor repercusión en el FPGA debido al uso del paralelismo que este ofrece, mientras que en el procesador las sentencias se reescriben en su lenguaje nativo, haciendo que se ejecuten más rápido. Para este enfoque se tiene en la aplicación de HOG una ventana de detección de 2×2 celdas, cada celda está conformada por 8×8 píxeles. La función que desempeña el procesador es la transformación de la imagen a escala de grises, la obtención de los gradientes, su magnitud y dirección, además de la ejecución del SVM. Las fases de obtención del histograma y la normalización son dedicadas al FPGA en donde notoriamente se ve un incremento en la velocidad de procesamiento, con respecto al enfoque en software. Sin embargo, debido a las limitaciones del entorno de desarrollo SDSoC 2015.4 solo fue posible procesar una imagen de 350×175 píxeles en 2.27 segundos, por lo que el enfoque en software fue más rápido. Finalmente, en el enfoque basado en hardware se realizan todas las etapas de las que se compone HOG y el clasificador en el FPGA. En el cálculo de magnitud se obtiene un valor aproximado de la raíz cuadrada implementando una tabla de búsqueda, cuyo elemento de entrada son los gradientes. Para el cálculo de la dirección y distribución de información del histograma, se optó en sustituir el arcotangente por intervalos de distribución en los contenedores del histograma, utilizando como entrada la división de los gradientes. Adicionalmente, después de la fase de normalización se implementó una función de binarización para la mejora del desempeño en el clasificador AdaBoost, pudiendo procesar una imagen de 1980×1020 en 0.0252 segundos. De acuerdo con las técnicas utilizadas, el enfoque en hardware tuvo un mejor desempeño considerando la relación entre el tamaño de la imagen y el tiempo de ejecución.

Utilizando un FPGA Zynq de Xilinx, Zhou *et al*., Implementaron el extractor de características HOG con el propósito de identificar señales de tránsito, en específico el estado de un semáforo

[59]. Se realizó un diseño hibrido entre software y hardware aprovechando la propiedad que brinda la arquitectura SoC. Antes de la extracción de características en la imagen de entrada RGB se aplica un prefiltro, haciendo un cambio al espacio de color HSV, debido a que este espacio de color permite identificar de mejor manera la saturación del color verde o rojo. Dada la capacidad del procesador, las fases de HOG son realizadas sin cambios, además debido a la propiedad de HOG que permite forma un bloque del tamaño deseado y dado a que solo se necesita encontrar una mancha (*blob*), la cual pertenece al foco del semáforo, se forma una ventana de detección de 2×2 celdas, donde cada celda está formada por 8×8 píxeles. Adicionalmente se ocupa un SVM aplicando una función de linealización, para reducir la dimensionalidad del vector de características de entrada. En la etapa de detección, se buscan los píxeles de una región que encierre al foco del semáforo, por ende, para encontrar la mancha del semáforo, se implementó una estructura pipeline, en la cual se hace un barrido píxel por píxel en la imagen para encerrar las posibles regiones donde se encuentra el objeto deseado, de esta forma si la clasificación detecta un píxel valido se almacena en un bloque de memoria para determinar la región a la que el píxel pertenece. Haciendo estas configuraciones se puede procesar una señal de video de 1024×768 píxeles a 60fps.

En [60] se muestran técnicas y algoritmos que pueden ser aplicados a las redes neuronales morfológicas con procesamiento dendrítico (MNNDPs, *Morphological Neuronal Network with Dendritic Processing*) para aumentar su desempeño, algunos de estos algoritmos tienen un enfoque adaptativo o evolutivo. Una MNNDPs basa su comportamiento en un modelo de hiper-cajas espaciales, pues dependiendo de la dimensión del vector de entrada es la dimensión de cada caja o dendrita. Dada la naturaleza espacial de la red, cada estructura neuronal se compone de un conjunto de dendritas o subunidades de procesamiento cuya estructura puede no estar adaptada de manera eficiente al conjunto de datos al que pertenece. Por esta razón, uno de los algoritmos adaptativos descritos se basa en la agrupación o división de las estructuras dendríticas para adaptar de mejor manera a los datos que encierra la neurona, teniendo un mayor grado de eficiencia. Otra técnica importante que resaltar es la linealización de los métodos aplicados a una red neuronal tipo DMN. Debido a que el algoritmo de formación de la red se basa en dividir 2^n hiper-cajas por cada iteración en la generación de dendritas, y donde la dimensión de la red reduce el desempeño de esta, el algoritmo es cambiado para realizar solo las hiper-cajas necesarias en un número limitado de iteraciones. Esto se logra mediante la reducción de la dimensionalidad del vector de entrada obteniendo un promedio entre valores mínimos y máximos que repercuten la estructura de la hiper-caja y su dimensión. Finalmente se presenta un método basado en la evolución diferencial (DEM, *Differential Evolution Method*) en donde se elige la neurona, agrupación de un conjunto de dendritas, que tenga el menor error de clasificación, cada dendrita original se le denominada dendrita padre, a la cual se le aplican 2 factores aleatorios que se restan, haciendo un operador de diferencia y creado una dendrita hija, si la dendrita hija tiene un menor error que la dendrita padre, la dendrita padre es sustituida por la hija. De esta forma cada dendrita, y por ende la red, queda mejor adaptada geométricamente al conjunto de datos que encierra. Dando como resultado una mejor eficiencia en la clasificación en diferentes bases de datos y diferentes clasificadores, con respecto a los métodos anteriores. Cabe resaltar que el autor menciona que la red MNNDP puede ser integrada más fácilmente por dispositivos que implementen el paralelismo, como lo son los GPUs, para la reducción en eltiempo de respuesta.

Algo importante a resaltar en el trabajo de Krips *et al*., es la inclusión de redes neuronales en un FPGA Virtex-EM de Xilinx. Cuyo objetivo fue implementar una técnica de detección y seguimiento de manos [61]. La red neuronal implementada, tiene un entrenamiento previo a la implementación en hardware, la estructura de la red está compuesta de 3 capas, una capa de

entrada, una capa oculta y una capa de salida. La capa de entrada es la encargada de distribuir los valores de entrada RGB, cada valor en un rango de [0,255], a una función de preprocesamiento que multiplica los valores de entrada por un valor ponderado diferente, con lo cual los 3 valores son procesados a un valor de 8 bits en un rango entre [0,1]. En la capa oculta se utiliza una función de activación no lineal, en este caso la tangente hiperbólica, sin embargo, debido a que su implementación en el FPGA requiere una gran cantidad de recursos, se opta en sustituir la tangente hiperbólica por una función de aproximación mediante una tabla de búsqueda. De esta forma la función hiperbólica brinda un valor cuyo estado se encuentra en "0" o "1", el cual es enviado a la capa de salida en donde se determina si la neurona fue excitada o no. Al implementar esta técnica en la red neuronal se pueden procesar imágenes RGB con un tamaño de 288×384 píxeles, ocupando el 34 % de los recursos de la tarjeta, y teniendo como resultados en la parte de clasificación una tasa de error de 0.780 % para aproximadamente 25 mil imágenes, el tiempo de clasificación obtenido fue inferior a los 10 ms.

Capítulo 3

Modelado Conceptual

Al mencionar modelado conceptual se hace referencia a la representación secuencial de un algoritmo y ésta puede ser desarrollada utilizando un lenguaje de software. Una implementación secuencial permite identificar las fases que integran un algoritmo y conocer su funcionamiento. El modelado conceptual de un algoritmo puede ser utilizado como base para el diseño de su arquitectura hardware. Tiene por finalidad facilitar la definición de los elementos de hardware requeridos en el diseño de la arquitectura e identificar técnicas idóneas para su implementación en un dispositivo específico. El presente capítulo describe el modelado conceptual correspondiente a los algoritmos del descriptor HOG y de la red neuronal DEN-*network*. Se hace énfasis en algunas modificaciones hechas a los algoritmos para disminuir los recursos requeridos en su implementación y se muestra como estos cambios influyen en la eficiencia del sistema.

3.1 Introducción

Considerando que para facilitar la comprensión de lo descrito en los primeros párrafos de esta sección es necesario poner en contexto la forma en que se definió el conjunto de entrenamiento, se dará inicio con este apartado mencionado que para evaluar el desempeño del sistema propuesto se utilizaron dos bases de datos de imágenes. La primera base de datos es obtenida de COIL-20 (*Columbia Object Image Library*) [62]. COIL-20 incluye imágenes en escala de grises de 20 objetos. Existen 72 imágenes por objeto, una imagen cada 5° cubriendo 360°, siendo un total de 1440 imágenes normalizadas en tamaño y con fondo negro. Por otro lado, la segunda base de datos fue extraída de ALOI (*Amsterdam library of object images*) [63]. ALOI es una colección

de imágenes de 1000 objetos, para este estudio únicamente se utilizan las imágenes de los primeros 20 objetos. Existen 72 imágenes de 192×144 píxeles por objeto, una imagen cada 5°, cubriendo 360°, con diferente ángulo de iluminación y con fondo negro, en las Figura 3.1 y 3.2 se muestran como están compuestas las imágenes de las bases de datos COIL-20 y ALOI respectivamente. Para homogeneizar las bases de datos, a las imágenes de ALOI se les aplico un redimensionamiento a 128×128 píxeles mediante una función integrada en Python que hace uso de una interpolación bicúbica.

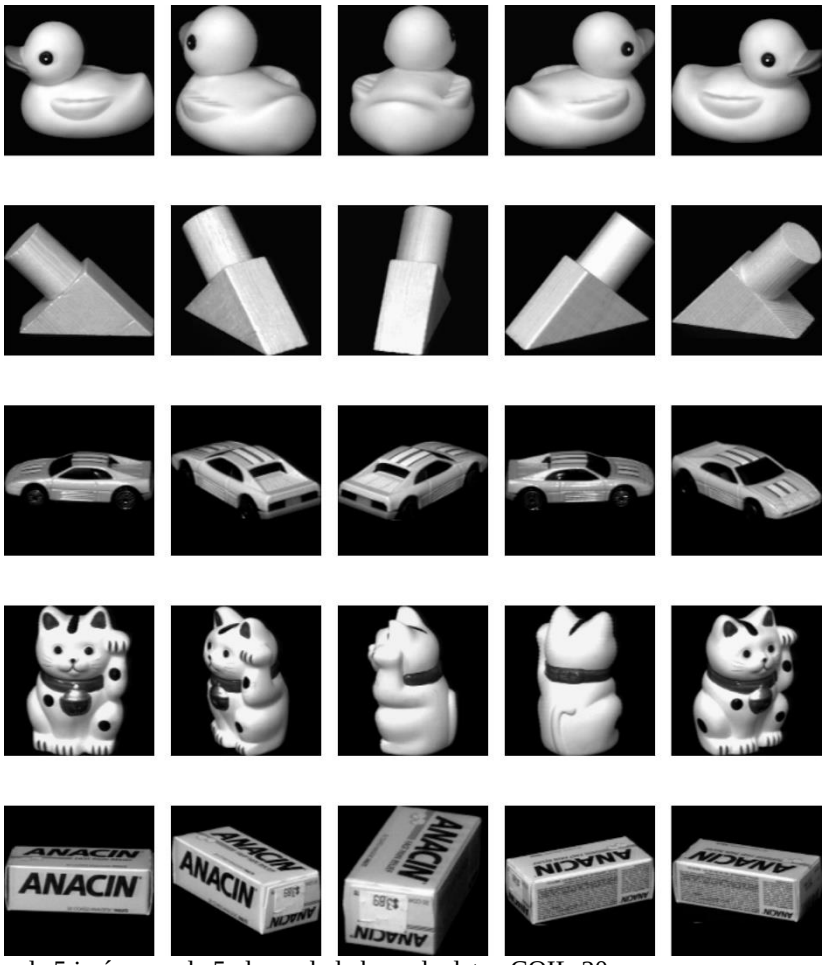

Figura 3.1. Muestra de 5 imágenes de 5 clases de la base de datos COIL-20.

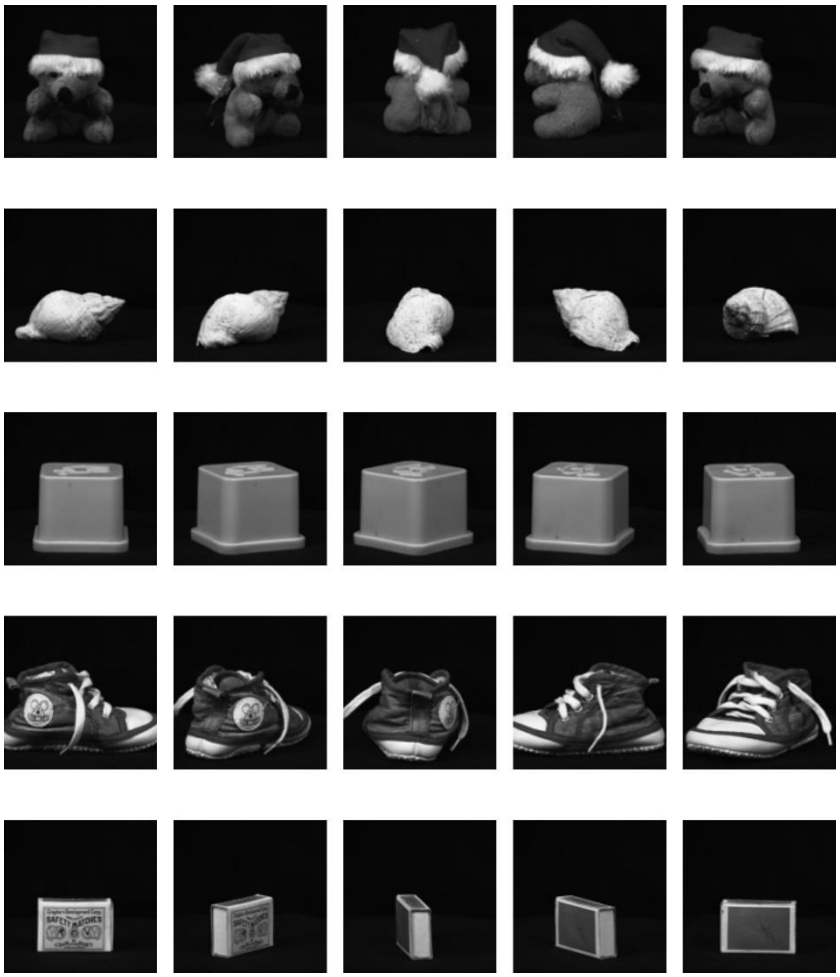

Figura 3.2. Muestra de 5 imágenes de 5 clases de la base de datos ALOI.

Además, los experimentos conducidos para evaluar el desempeño del sistema propuesto siguieron el siguiente protocolo:

- Primero, las bases de datos fueron utilizadas por separado, es decir se procesan primero imágenes de COIL-20 y después de ALOI. De la base de datos en cuestión fueron formados subconjuntos variando el número de objetos involucrados, dando origen a problemas de diferente número de clases.
- Segundo, se procesan los subconjuntos individualmente. En cada subconjunto, las imágenes de cada objeto son divididas en dos grupos, cada uno de 36 objetos. Se eligieron imágenes intercaladas, es decir, considerando que las imágenes están numeradas de acuerdo con el orden en que fueron adquiridas, un grupo contiene las imágenes de número impar y el otro las de número par.
- Tercero, se toma uno de los grupos de imágenes de cada objeto del subconjunto y se etiqueta como una clase, formando con esto el conjunto de entrenamiento que es utilizado para entrenar a la red DEN-*network*.
- Cuarto, el segundo grupo es utilizado para validar la generalización del comportamiento aprendido por la red.

El modelado conceptual consiste simplemente en desarrollar los algoritmos que forman parte del sistema, HOG y DEN-*network*, siguiendo un paradigma secuencial y utilizando un lenguaje de

programación. En el presente trabajo de tesis, para cumplir con esta actividad, y por simplicidad, se ha elegido utilizar el lenguaje de programación interpretado Python en el modelado del descriptor HOG y lenguaje de programación Matlab para modelar la red DEN-*network*.

Debido a la naturaleza del modelado de HOG se requiere tener acceso a diferentes imágenes, recibir y exportar archivos, modificar parámetros y visualizar los elementos resultantes, por lo cual, se desarrolló una interfaz básica usando el lenguaje Python, que permite realizar dichas acciones además de implementar el modelado del descriptor HOG. La Figura 3.3 muestra la pantalla principal de la interfaz desarrollada.

La interfaz permite seleccionar un grupo de imágenes, que puede ser utilizado en la fase de entrenamiento de la red neuronal (este grupo es el conjunto de entrenamiento) o para un proceso de clasificación de un grupo grande de imágenes, o únicamente seleccionar la imagen que se desea sea procesada mediante HOG. Además, permite definir parámetros como el tamaño de la celda (definido de $npx \times npy$ píxeles), el tamaño de bloque (definido por $Cx \times Cy$ celdas) y traslape entre bloques, e indicarle que calcule el descriptor HOG.

En respuesta, si se seleccionan un grupo de imágenes, la interfaz genera un archivo que contiene los vectores de características de las imágenes del grupo. En el caso que se trate de un conjunto de imágenes que serán utilizadas para el entrenamiento de la red, el archivo además incluye la clase a la que pertenece cada vector de características calculado. Si únicamente se selecciona una imagen, además de almacenar su vector de características en un archivo, la interfaz indica el tamaño del vector de características obtenido y muestra sus elementos en consola; si el número de elementos es mayor a 9 solo se muestran los primeros 9 elementos del vector. Además, a través de una ventana emergente se pueden observar las imágenes correspondientes al cálculo $\boldsymbol{Gx}$, $\boldsymbol{Gy}$, $\boldsymbol{Mag}$ y $\boldsymbol{\theta}$ (ver Figura 3.4). La información contenida en el archivo que será utilizado para entrenar a la red se denomina conjunto de entrenamiento.

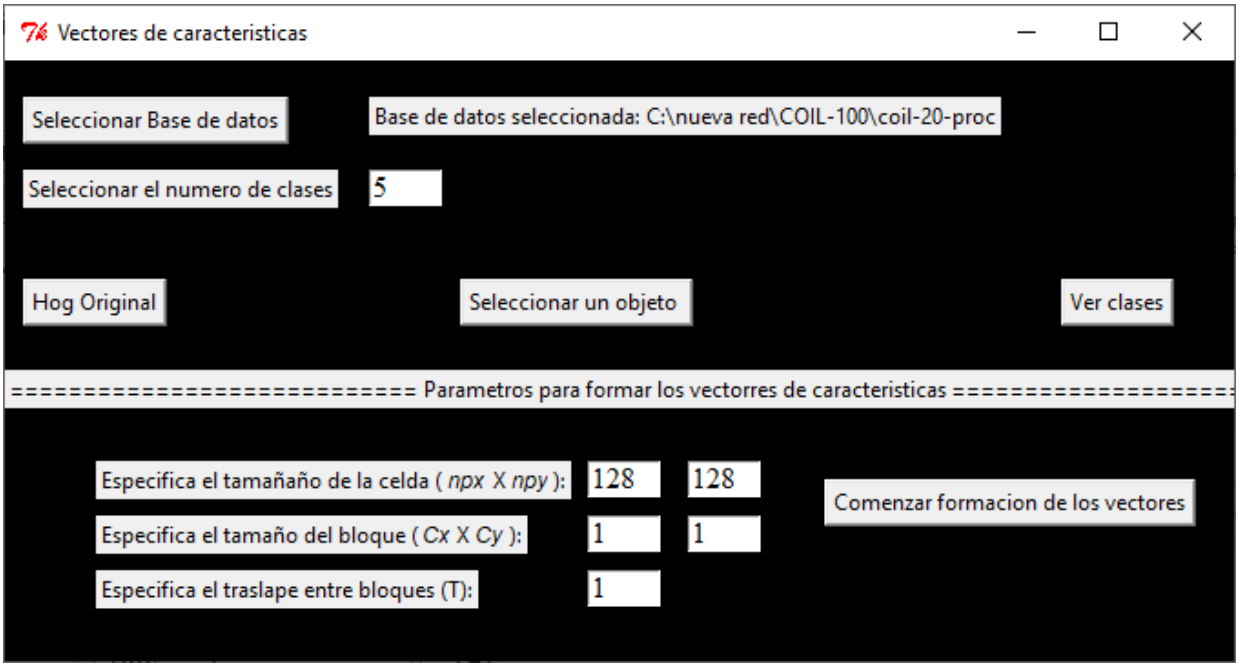

Figura 3.3. Interfaz de usuario para la implementación del algoritmo de HOG.

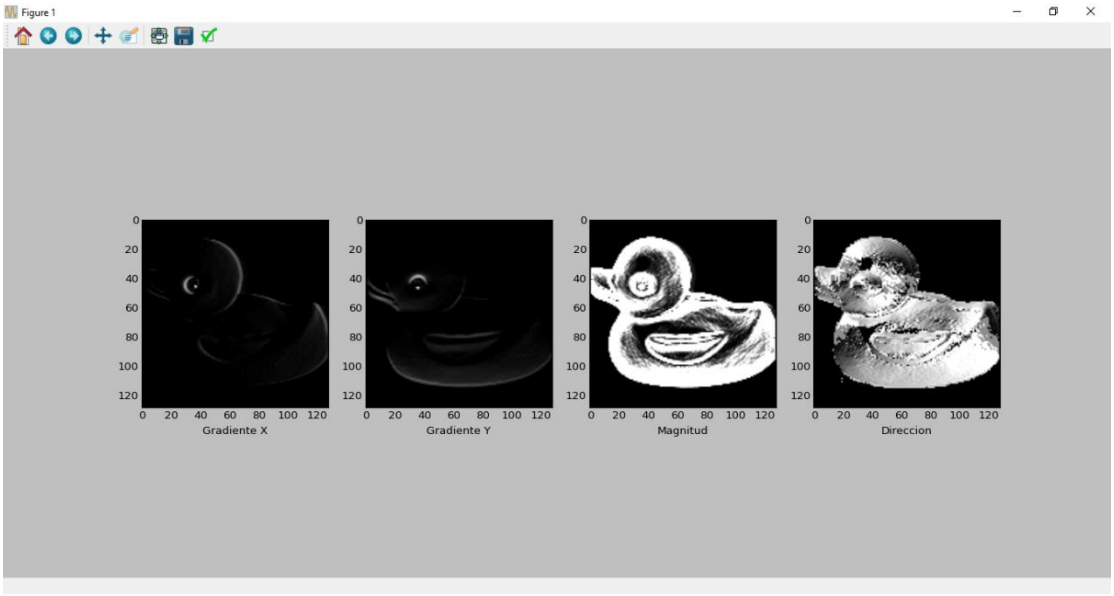

Figura 3.4. Imágenes de $\boldsymbol{Gx}$, $\boldsymbol{Gy}$, $\boldsymbol{Mag}$ y $\boldsymbol{\theta}$ (en la imagen correspondiente a la dirección con fines de visualización se incrementó el contraste).

Por otro lado, el modelado de la red DEN-*network*, realizado en Matlab, se desarrolló en dos fases, entrenamiento y operación o clasificación. El archivo que contiene los vectores de características del conjunto de entrenamiento es utilizado para entrenar a la red DEN-*network*. Es decir, este conjunto de entrenamiento especifica el problema de clasificación para el cual debe ser adaptada la red y es utilizado para determinar la estructura de la red.

El resultado de la fase de entrenamiento define la estructura de la red DEN-*network*. Esto es, especifica cuantas neuronas integran a la red, el número de dendritas que agrupa cada neurona, los pesos sinápticos asociados a cada dendrita (definidos por el centroide correspondiente) y el elemento utilizado para determinar el estado de la dendrita (definido por la matriz de covarianza correspondiente). Todos estos parámetros son almacenados en un archivo, el cual es accesado en la fase de clasificación para determinar la clase a la que pertenece un vector de entrada nuevo.

En la fase de clasificación, al recibir un archivo con un conjunto de n vectores que deben ser clasificados, se aplica la fase de operación a cada uno de ellos y los resultados obtenidos son almacenados en un archivo; el archivo contiene los n vectores y la clase a la que pertenece cada uno de ellos. Además, la interfaz calcula y muestra los porcentajes de reconocimiento y error que la red obtiene.

En el caso de un archivo que solo contiene un vector a clasificar, la interfaz únicamente aplica la fase de clasificación de la red y muestra en pantalla la clase a la que pertenece.

3.2 Modelado conceptual del algoritmo de Histograma de Gradientes Orientados

Tomando como referencia los métodos y ecuaciones descritos en el apartado 2.4, se implementan los algoritmos de forma secuencial basándose en el paradigma de programación estructurada. Debido a la naturaleza de las bases de datos ocupadas en este trabajo de tesis, la fase de ventaneo es omita, pues las imágenes contenidas en COIL-20 y ALOI tienen una fase de segmentación previa, por lo que una imagen se considera como una ventana de detección. Adicionalmente, buscando la optimización de recursos y menor tiempo de procesamiento en una arquitectura hardware, partes de los algoritmos de HOG fueron modificados en software para su posterior

implementación en el FPGA, con el propósito de tener mejores datos de referencia en el análisis de resultados. Cada modificación fue probada en el desempeño de la red neuronal, los resultados de estas modificaciones se mostrarán en el apartado 3.4.

Todas las fases de HOG son descritas e implementadas por funciones, las cuales son llamadas mediante una función principal denominada HOG(). Para la implementación del algoritmo de HOG es necesario un elemento de entrada, en este caso, una imagen en escala de grises de tamaño $h \times w$ introducida por el usuario, la imagen es representada por una matriz (*array* de 2 dimensiones) para su manipulación en Python. La función principal es la encargada de abrir la imagen, calcular su tamaño, y mandar a llamar a las funciones Gradiente(), Orientación(), Bloques(), pasando los parámetros necesarios de cada función. La fase de normalización es incluida en la función de Bloques(). En la Tabla 3.1 se describe la función principal en forma de pseudocódigo.

Tabla 3.1. Pseudocódigo de la función principal que implementa las fases de HOG.

```
00|Function HOG( )
01|Variables
02|     Global Int h, w
03|     float * Vec,** Mag,** θ,** Hist,* Vect
04|Begin
05|     Img ← Abrir("C:\\ ... ") // Se abre la imagen que se desea trabajar
06|     h ← Img(alto) // Se determina el ancho y el largo de la ventana
07|     w ← Img(ancho) // 'h' es el largo y 'w' el ancho
08|     //Se mandan a llamar a las funciones que implementan el algoritmo de HOG
09|     Mag, θ ← Gradiente(Img)
10|     Hist ← Orientación(Mag, θ)
10|     Vect ← Bloques(Hist)
11|     return Vect
12|end
```

3.2.1 Obtención del gradiente, magnitud y dirección

La función que realiza el cálculo de los gradientes, su magnitud y dirección es denominada Gradiente(). El parámetro de entrada de esta función es la imagen seleccionada por usuario, sus parámetros de salida son las matrices de magnitud y dirección. Los cálculos realizados en esta función se basan en las ecuaciones (2), (3) y (4) definidas en la Sección 2.4.1. Dado que la función tiene acceso al tamaño de la imagen, se realiza el procesamiento de cada píxel mediante 2 funciones iterativas que llevan el control del desplazamiento a través de la imagen. Primeramente, en el proceso iterativo se aplica a cada píxel el operador derivador de la ecuación (2) para formar las matrices de los gradientes $\boldsymbol{Gx}$, $\boldsymbol{Gy}$. Consecutivamente utilizando las ecuaciones (3) y (4) se obtienen la magnitud y dirección de cada píxel procesado, utilizando $\boldsymbol{Gx}$, $\boldsymbol{Gy}$ para conformar las matrices correspondientes a cada operación. Finalmente se regresan las matrices de magnitud y dirección a la función principal para que pueda implementarse la siguiente fase. El pseudocódigo de la función descrita se muestra en la Tabla 3.2.

Tabla 3.2. Implementación de los gradientes, magnitud y dirección en pseudocódigo

```
00|Function Gradiente(Img) //Calculo del algoritmo del gradiente
01|Variables //se definen todas las variables como globales
02|     Global int h, w
03|     int i, j
04|     float ** Gx,** Gy,** Mag,** θ
05|Begin
06|     for(i=1 to h-1)
07|          for(j=1 to w-1)
08|               // Se calcula la resta de los píxeles vecinos en X y Y
09|               Gx_{i,j} ← Img_{i,j+1} − Img_{i,j−1}
10|               Gy_{i,j} ← Img_{i+1,j} − Img_{i−1,j}
11|               Mag_{i,j} ← √(Gx²_{i,j} + Gy²_{i,j}) // Se calcula la magnitud de los gradientes
12|               θ_{i,j} ← tan⁻¹(Gx_{i,j}/Gy_{i,j}) //Se calcula la dirección del gradiente
13|     return Mag, θ
14|end
```

El resultado obtenido de $\boldsymbol{Gx}$, $\boldsymbol{Gy}$, $\boldsymbol{Mag}$ y $\boldsymbol{\theta}$, aplicando la función Gradiente() en Python al primer elemento de la primera clase de COIL-20, se puede observar en la Figura 3.5.

Adicionalmente, cabe recalcar que, dada la naturaleza del gradiente, el cual requiere un proceso de convolución, en el que se necesitan los píxeles superior, inferior, posterior y anterior al píxel i,j-ésimo que se está procesando, las matrices resultantes de gradientes, magnitud y dirección contienen $(w-2)\times(h-2)$ elementos, pues el procesamiento comienza en el píxel $(1,1)$ y termina en el píxel $(w-1, h-1)$. Para que las matrices sean del mismo tamaño al de a la imagen original, se agregó un contorno de píxeles con un valor de 0.

Teniendo en consideración que el algoritmo desarrollado será implementado en una arquitectura hardware y buscando mejorar los parámetros de velocidad de procesamiento y recursos consumidos, se realizan cambios para minimizar la complejidad de cálculo en las operaciones de magnitud, división de los gradientes y arcotrangente. Para esto se tienen en cuenta las propiedades de las bases de datos utilizadas en este trabajo de tesis. Como se describió en el apartado 3.1 las imágenes en las bases de datos están representadas en escala de grises y tienen un tamaño de 128×128 píxeles con una profundidad de 8 bits, existiendo un total de 16,384 píxeles, cuyo valor puede estar en el rango de [0,255], que deben ser procesados.

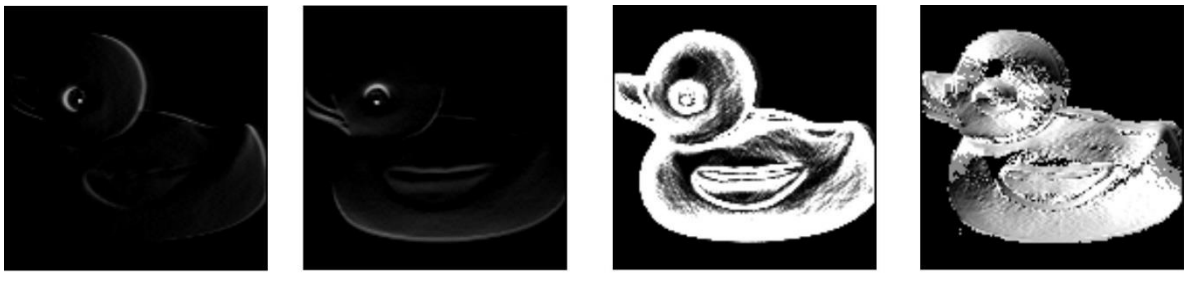

Figura 3.5. a) $\boldsymbol{Gx}$, b) $\boldsymbol{Gy}$, c) $\boldsymbol{Mag}$, d) $\boldsymbol{\theta}$ (Con propósitos de visualización se modificó el contraste de la figura d) correspondiente a la dirección).

En una arquitectura en hardware es importante minimizar la cantidad de bits involucrados en cada operación, por lo que es necesario determinar la resolución de bits para cada operación realizada en la primera fase de HOG implementada por Python, ver Tabla 3.3.

Tabla 3.3. Resolución de bits por cada tarea de la primera fase del algoritmo HOG.

	Rango de valor	Resolución de bits
Valor del píxel	(0 , 255)	32
$\boldsymbol{Gx}$ y $\boldsymbol{Gy}$	(-255 , 255)	32
$\boldsymbol{Gx_{i,j}/Gy_{i,j}}$	(-255 , 255)	64
$\boldsymbol{Mag_{i,j}}$	(0 , $\sqrt{130{,}050}$)	64
$\boldsymbol{\theta_{i,j}}$	$\left(-\frac{\pi}{2}, \frac{\pi}{2}\right)$	64

Las operaciones utilizadas en la implementación secuencial de HOG requieren el uso de aritmética de punto flotante, en precisión simple y doble. En hardware, esto genera una arquitectura que usa una gran cantidad de recursos. Buscando reducir estos recursos es necesario implementar alternativas que no afecten significativamente el rendimiento del sistema. Estas alternativas son mencionadas a continuación. Primero, es evidente que para almacenar el valor del píxel y los resultados del cálculo del gradiente solo se requieren 8 y 9 bits, respectivamente. Además de utilizar aritmética de punto fijo para estas operaciones. Segundo, en el caso de la magnitud, en donde la complejidad recae en el cálculo de la raíz cuadrada se optó por cambiar la ecuación de la magnitud a la forma $\boldsymbol{Mag_{i,j} = Gx_{i,j}^2 + Gy_{i,j}^2}$. Tercero, para el cálculo de la dirección del gradiente se realizaron simulaciones modificando el límite de los elementos de entrada, en este caso la división de los gradientes, en donde, dada la naturaleza de la función arcotangente se notó que el intervalo de valores de entrada de $(-2{,}2)$, al generar un vector de características y clasificarlo, genera un resultado donde el desempeño del sistema no se ve afectado de manera significativa. Por lo que la resolución de bits necesaria para el elemento de entrada se acota a 10 bits, un bit de signo, 1 bit de parte entera y 8 bits de parte decimal. De esta forma, el intervalo de valores de salida del arcotangente $\left(-\frac{\pi}{2}\ ,\ \frac{\pi}{2}\right) rad$, se acota a los límites (-$1.10714872 - 1.10632324) \, rad$. Este cambio significa que la distribución del peso del gradiente en los contenedores extremos del histograma es de menor proporción. Por lo tanto, la nueva resolución de bits requerida para una arquitectura hardware para cada operación, considerando las modificaciones, se muestra en la Tabla 3.4.

Tabla 3.4. Modificación en la resolución de bits para cada tarea de la primera fase de HOG.

	Rango de valor	Resolución de bits
Envió de píxel	(0 , 255)	8
$\boldsymbol{Gx}$ y $\boldsymbol{Gy}$	(-255 , 255)	9
$\boldsymbol{Gx_{,j}/Gy_{i,j}}$	(-2 , 2)	10
$\boldsymbol{Mag_{i,j}}$	(0 , 130,050)	17
$\boldsymbol{\theta_{i,j}}$	(-1.10714872 , 1.10632324)	16

Las modificaciones mencionadas inducen cambios en un segmento del pseudocódigo mostrado en el algoritmo de la Tabla 3.2, mismos que se expresan en la Tabla 3.5.

Tabla 3.5. Segmento del pseudocódigo modificado.

```
11|        Mag_{i,j} ← Gx_{i,j}^2 + Gy_{i,j}^2  // Se calcula la magnitud de los gradientes sin raiz
12|        if(Gy_{i,j} == 0)
13|            if(Gx_{i,j} < 0)
14|                θ_{i,j} ← −1.10714872
15|            else
16|                θ_{i,j} ← 1.1063232421875
17|        else
18|            if( Gx_{i,j} / Gy_{i,j} > 1.99609375 )
19|                θ_{i,j} ← 1.1063232421875
20|            else
21|                if( Gx_{i,j} / Gy_{i,j} < −1.99609375)
22|                    θ_{i,j} ← −1.10714872
23|                else
24|                    θ_{i,j} ← tan^{-1}( Gx_{i,j} / Gy_{i,j} )
```

3.2.2 Orientación del gradiente

De acuerdo a lo expresado en el pseudocódigo de la Tabla 3.1, la función Orientación() es la siguiente que debe ser evaluada. Esta función debe realizar el cálculo de los histogramas que representan la distribución de la magnitud del gradiente en cada una de las celdas (proceso descrito en el apartado 2.4.2). La función Orientación() recibe como parámetros de entrada las matrices $\boldsymbol{\theta}$ y $\boldsymbol{Mag}$, obtenidas en la fase anterior, y las divide en un conjunto de celdas de tamaño $npx \times npy$ píxeles. Entonces, por cada celda generada se obtiene un histograma de 9 elementos mediante la función Histograma(), obteniendo como resultado final una matriz de histogramas. La función encargada de la división de las matrices en celdas se expresa en pseudocódigo en la Tabla 3.6., la cual también hace un llamado de la función Histograma() por cada celda.

Tabla 3.6. Pseudocódigo de la función que implementa el histograma

```
00|Function Orientación(Mag, θ)
01|Variables
02|    Global int npx, npy
02|    int i, j
03|    float ** MagC,** θC,** Hist
04|Begin
05|    npx ← Leer(Numero de pixeles en x)
06|    npy ← Leer(Numero de pixeles en y)
07|    Dividir las matrices de Mag y θ en celdas de tamaño npx y npy
08|    Almacenar cada celda obtenida en MagC y θC, respectivamente
09|    for(i=0 to npy)
11|        for(j=0 to npx)
12|            Hist_{i,j} ← Histograma(MagC_{i,j}, θ_{i,j})
13|    return Hist
14|end
```

La implementación de la función Hitstograma() se basa en los métodos descritos en el apartado 2.4.2 y es mostrada mediante el pseudocódigo de la Tabla 3.7. Debido a que los valores de la función trigonométrica inversa arcotangente se ubican en el intervalo $\left(-\frac{\pi}{2}, \frac{\pi}{2}\right) rad$, los 9 contenedores de acuerdo al algoritmo de HOG están ubicados en $\{-80°,\ -60°,\ -40°,\ -20°,\ 0°,\ 20°,\ 40°,\ 60°,\ 80°\}$, con una separación de $\frac{1}{9}\pi\ rad$ o 20°. Basándose principalmente en la ecuación (7) donde $c\_th_{ij}^{lk} = \boldsymbol{\theta}_{i,j}$ y $c\_m_{ij}^{lk} = \boldsymbol{Mag}_{i,j}$, se distribuye el valor ponderado de cada píxel procesado en los contenedores de acuerdo con el ángulo e intervalo de contenedores al que pertenece. En el algoritmo descrito en la Tabla 3.7, y para facilitar la escritura y compresión de éste, a cada contenedor se le ha denominado *bin.*

Tabla 3.7. Pseudocódigo para la obtención del histograma.

```
00|Function Histograma(Mag_c, θ_c)
01|Variables
02|     int i, j                          // x es el factor de conversión de grados a radianes
03|     float d_{o+1}, d_bins, bins[9] ← {0}, x ← (π rad / 180°)
04|Begin
05|     d_bins ← 20°x //separación de los contenedores
06|     for(i=0 to npy)
07|          for(j=0 to npx)
08|               //Se obtiene a que contenedor (bin) superior pertenece
09|               d_{o+1} ← Asignar el valor del contenedor superior al angulo θ_{i,j}
10|               k ← Asignar el indice del contenedor inferior al angulo θ_{i,j}
11|                    si el angulo es mayor o menor al rango de los contenedores
12|                    asignar el indice del contenedor mas cercano
13|               // Se distribuye el peso de la magnitud entre cada contenedor
14|               if(θ_{i,j} < −80°x or θ_{i,j} > 80°x)
15|                    bins_k ← bins_k + Mag_{i,j}
16|               else
17|                    if(MOD(θ_{i,j}, 10°x) == 0)
18|                         bins_k ← bins_k + Mag_{i,j}
19|                    else
20|                         bins_k ← bins_k + Mag_{i,j} (1 − (θ_{i,j} + (d_{o+1} − 20°x)) / d_bins)
21|                         bins_{k+1} ← bins_{k+1} + Mag_{i,j} ((θ_{i,j} + (d_{o+1} − 20°x)) / d_bins)
22|     return bins
23|end
```

Nuevamente, buscando simplificar la implementación en hardware de este proceso, se analizó el desempeño del modelado conceptual cuando en la operación de división de la ecuación (7) el valor constante de la separación entre contenedores $d_{bins} = 20° = 0.34906\ rad$ es cambiado por su valor más cercado potencia de 2, $d_{bins} = 0.5\ rad$, y se determinó que esta modificación no afectaba en forma significativa el desempeño del algoritmo. Sin embrago, con respecto a la arquitectura hardware este cambio representa una mejora sustancial ya que en lugar de implementar una división cuyo divisor es un número real se realiza una simple operación de corrimiento.

3.2.3 Bloques de características

La última función que el pseudocódigo de la Tabla 3.1 debe evaluar es Bloques(). El pseudocódigo de esta función se muestra en la Tabla 3.8. La función Bloques() recibe la matriz de histogramas y generan bloques de características a partir de agrupaciones de histogramas (recordar que existe un histograma por celda), la cantidad de histogramas involucrados en un bloque es determina por el usuario, al igual que el traslape entre bloques. Todos los histogramas que agrupa un bloque son concatenados, formando un vector parcial, cada vector parcial es normalizado mediante la función Normalización(), y es concatenado para formar el vector final, que corresponde al vector de características de la ventana de detección.

Tabla 3.8. Pseudocódigo que realiza las agrupaciones de histogramas en bloques.

```
00|Function Bloques(Hist)
01|Variables
02|     Global int npx, npy
02|     int i, j
03|     float *Vect,*VectP, *VectP'
04|Begin
05|     Cx ← Leer(Numero de celdas en x), dCx ← Leer(Traslape en x)
06|     Cy ← Leer(Numero de celdas en y), dCy ← Leer(Traslape en y)
07|     /*Comenzando desde la celda [0,0] se hace un barrido por toda la matriz con
08|       incrementos dCy y dCx*/
09|     for(i=0 to npy, dCy)
10|             for(j=0 to npx, dCx)
11|                     Formar un bloque de Cx × Cy celdas partiendo de la
12|                     posicion [0,0]
13|                     VectP ← Concatenar histogramas del bloque
14|                     VectP' ← Normalización(VectP)
15|                     Vect ← Concatenar(Vect, VectP')
16|     return Vect
17|end
```

Como el algoritmo se desarrolla en forma secuencial, una vez determinado el tamaño del bloque, se hace un barrido bloque a bloque a través de toda la matriz de histogramas. El desplazamiento del bloque en x y y está dado por d_{Cx} y d_{Cy} respectivamente, lo cual corresponde al traslape entre bloques.

3.2.4 Normalización

Finalmente, para normalizar un vector parcial proveniente de un bloque de características, primero se determina el tamaño del vector, considerando que cada bloque agrupa una serie de vectores concatenados de 9 elementos. Posteriormente, se normaliza el vector parcial basándose principalmente en las ecuaciones de normalización, descritas en el apartado 2.4.4. Con este fin, se eligió la forma $L2-norm$, ecuación (9), donde ε es un elemento que permite evitar la división entre 0, por lo que generalmente es un número pequeño elegido por el usuario, en este caso se tomó $\varepsilon = 0.001$. El proceso descrito se implementó en la función Normalización(), que regresa el vector parcial que se concatena con los demás vectores parciales y de esta forma obtener el vector de características HOG. El algoritmo de dicha función se describe en la Tabla 3.9.

Tabla 3.9. Pseudocódigo para la normalización de los bloques.

```
00|Function Normalización(VectorP)
01|Variables
02|     int k, tam
03|     float D,* VectorP', ε ← 0.001
04|Begin
05|     tam ← size(VectorP)
06|     D ← 0
07|     for(k=0 to tam)
08|             D ← D + VectorP_k^2
09|     VectorP' ← VectorP/√(D + ε^2)
10|     return VectorP'
11|end
```

De manera similar a los algoritmos anteriores, para minimizar la complejidad de operaciones se optó por omitir la raíz cuadrada en la normalización, por lo que la línea de código 09 se reemplazaría por,

$$\boldsymbol{VectorP'} \leftarrow \boldsymbol{VectorP}/(D + \varepsilon^2) \tag{23}$$

El resultado del proceso de normalización entrega vectores tam-dimencionales, donde cada elemento del vector es de 64 bits: 1 bit de parte entera y 63 de parte decimal. Se realizó un análisis donde se observó que los vectores de características normalizados generados por HOG a partir de las bases de datos COIL-20 y ALOI presentaban un rango de valores muy pequeño, donde, para el caso de la base de datos COIL-20 el rango de valores se encuentra entre $[3.1346148977 \times 10^{-10}, 2.24248804355 \times 10^{-7}]$, por ende, los 21 bits más significativos tienen un valor de 0, mientras que para la base de datos ALOI el rango de valores esta entre $[4.02621150007 \times 10^{-10}, 1.15404817208 \times 10^{-5}]$, lo que implica que el valor de 0 se presenta en los 18 bits más significativos. La implementación de hardware aprovecha esta característica que puede ser implementada por una simple operación de desplazamiento a la izquierda de 21 y 18 posiciones, respectivamente, para luego solo considerar los 32 bits más significativos, esta acción corresponde a seleccionar los elementos (42:12) para la base de datos COIL-20 y (46:16) para la base de datos ALOI del bus de datos que corresponde al vector características, lo que reduce de manera significativa los recursos del FPGA requeridos y facilita las operaciones que se llevan a cabo en la parte de clasificación.

La implementación de estas operaciones fue evaluada en el modelado conceptual, añadiendo la línea de código de la expresión (24) entre las líneas 9 y 10 del pseudocódigo de la Tabla 3.9, determinándose que el desempeño global del algoritmo no presentaba cambios significativos.

$$\boldsymbol{VectorP'} \leftarrow \boldsymbol{VectorP'} << P \tag{24}$$

Para el desplazamiento de los vectores de la base de datos de COIL-20 $P = 21$ y para la base de datos ALOI el factor $P = 18$. Cabe aclarar que esta operación es válida únicamente para las bases de datos mencionadas, para una base de datos diferente es necesario realizar un análisis similar al descrito para determinar en qué condiciones se puede usar este procedimiento.

3.3 Modelado conceptual de la red neuronal DEN-*network*

De manera similar a lo descrito en el apartado 3.2, se utiliza el paradigma de programación estructurada para una mejor descripción de los algoritmos de este apartado. A partir de la función HOG() se generan un conjunto de vectores de características, provenientes de diferentes imágenes (objetos), correspondientes a las bases de datos COIL-20 y ALOI, y se almacenan en un archivo. El archivo generado se compone de 4 elementos principales; un par de conjuntos de vectores de características para entrenamiento y evaluación de la red, P y P_{test}; así como dos arrays, T y T_{test}, que contienen la información sobre a qué clase pertenecen los elementos contenidos en P y P_{test} respectivamente.

Para administrar las fases de formación y entrenamiento de la red neuronal, se implementa la función principal DEN_Network(), cuyo pseudocódigo se muestra en la Tabla 3.10. Esta función está formada por un conjunto de subrutinas que guardan dependencia entre sí, por lo que son llamadas en forma secuencial bajo el siguiente orden, el primer llamado es para la subrutina Neuronas() que cumple la tarea de formación de las neuronas, consecutivamente se llama a Clusstering(), encargada de crear las subunidades de procesamiento, y terminando con el llamado a EvDiff(), utilizada para aumentar el desempeño de la red mediante una función que aplica el algoritmo de evolución diferencial.

El proceso de formación de la red utiliza los vectores de características almacenados en P. Una vez determinada la estructura final de la red neuronal, se evalúa con los vectores de características de prueba P_{test}.

Tabla 3.10. Pseudocodigo de la función principal para la creación de una DEN-*network.*

```
00|Function Den_Network(P, P_test, T, T_test)
01|Variables
02|     Cell ** τ
03|     float err
04|Begin
05|     τ, P_class, T_class ← Neuronas(P, P_test)
06|     τ ← Clusstering(P, T, P_class, T_class, τ)
07|     τ ← EvDiff(τ)
08|     err ← Calculo_error_global(τ, P_test, T_test)
09|     Print err
10|end
```

Para facilitar el entrenamiento de la red neuronal se crea la unidad de procesamiento τ, que contiene la información de la red: su estructura, las unidades de procesamiento y los pesos sinápticos. Una vez obtenidos los elementos de entrada, se separan los objetos contenidos en P en subconjuntos, cada subconjunto corresponde a una clase específica, es decir, solo incluye imágenes de un mismo objeto. Para cada subconjunto de P, se obtiene un centroide y una matriz de covarianza, conformando así las neuronas de la red. Para contemplar los casos donde ocurre una indeterminación al obtener la matriz inversa de covarianza en un subconjunto, primeramente, se calcula el determinante de la matriz de covarianza y si el resultado es 0 se calcula la semi-matriz de covarianza inversa de Moore-Penrose. La subrutina Neuronas() implementa el proceso descrito; la Tabla 3.11 muestra su pseudocódigo.

Tabla 3.11. Pseudocódigo para la formación de las neuronas de la red.

```
00|Function Neuronas(P, T)
01|Variables
02|     int j, Nclass
03|     float ** Σ,* μ,* P_class,* T_class
04|Begin
05|     NClass ← Se determina el numero de clases de la base de datos "P"
06|     //Se separan los vectores de en subconjuntos correspondientes a cada clase
07|     P^j_class ← P //Para cada clase se tiene P^j_class que representa todos los vectores
08|     T^j_class ← T//asociados a la clase j
09|     for(j=1 to NClass)
10|     //se obtiene la matriz de covarianza y el centroide del subconjunto de datos P^j_class
11|             Σ_j ← cov(P^j_class)
12|             μ_j ← centroide(P^j_class)
13|             //para cada clase j se corresponde una neurona τ^j asociada a una
14|             //matriz de covarianza Σ_j^-1 y un centroide μ_k
16|             if(det(Σ_j) == 0)
17|                     Σ_j^-1 ← pinv(Σ_j)//matriz de covarianza inversa de moore-penrose
18|             else
19|                     Σ_j^-1 ← inv(Σ_j)
20|             τ^j ← Σ_j^-1, μ_j
21|     return τ, P_class, T_class
22|end
```

Debido a que el número de neuronas corresponde al número de clases, se tiene que τ^j es la unidad de procesamiento asociada a la *j*-ésima neurona y *j*-ésima clase. Sin embargo, para ciertos casos una unidad de procesamiento por neurona no es suficiente, p. ej., para conjuntos de datos que son linealmente no separables el desempeño de la red neuronal decae si se usa la estructura mencionada. La subrutina Clusstering() solventa un problema de este tipo mediante la implementación de un algoritmo de clusterización. Así, la subrutina Clusstering() genera varios clusters por cada clase, es decir, cada clase es separada en subclases mediante el algoritmo *Kmeans++* y a su vez se obtiene la matriz de covarianza y centroide de cada subclase, formando así, las dendritas de cada neurona. La creación de clusters de acuerdo con [9] se realiza mediante el cálculo del error. Partiendo del análisis de una clase, primeramente, se calcula el error global de la red neuronal y se compara con el error de la clase de interés, un error local, si este error es mayor al global, se realiza un cluster y se repite el proceso del cálculo del error. La cantidad máxima de clusters que pueden ser generados es determinada por un parámetro establecido por el usuario. El número de clusters creados contribuye al decremento del error por clase, que a su vez repercute en el incremento del desempeño de la red neuronal. Cada neurona puede contener *k* de clusters, donde no necesariamente *k* es igual para todas las clases. Para este trabajo de tesis se contempló que el número máximo de clusters por clase es 3. El seudocódigo del algoritmo de la subrutina Clusstering() *se* puede observar en la Tabla 3.12.

La modificación realizada a la DEN-*network* radica en recortar los elementos de la matriz de covarianza inversa para que puedan ser representados por una variable entera en el FPGA.

Tabla 3.12. Pseudocódigo para la división de clusters.

```
00|Function Clusstering(P, T, P_class, T_class, τ)
01|Variables
02|     int j, k, Nclass
03|     float error_global, error_local,** C
04|Begin
05|     error_global ← Calculo_error_global(τ, P, T)
06|     for(j=0 to Nclass)
07|          k = 0
08|          while(error_local > error_global)
09|               Crear k + 1 Clusters C^j para cada j Clase
10|               τ^j ← Σ_k^-1, μ_k     //se guardan los clusters en la red neuronal
11|               error_local ← Calculo_error_local(τ, P^j_class, T^j_class, j)
12|               if(error_local == 0 or k == 5)
13|                    break
14|               k ← k + 1
15|          error_global ← Calculo_error_global(τ, P, T)
16|     return τ
17|end
```

El cálculo del error local requiere del proceso de clasificación, para lo cual se quiere del subconjunto de datos de la clase que se está evaluando y de la información de toda la red ($\boldsymbol{P}^{j}_{class}, \boldsymbol{T}^{j}_{class}, j, \boldsymbol{\tau}$). La clasificación se realiza de acuerdo con lo descrito en el apartado 2.5.3, requiriendo determinar la distancia de Mahalanobis de todas las dendritas con respecto a un objeto de entrada. En consecuencia, el objeto será asociado a la clase que presente la menor distancia, para ello se considera la menor distancia de cada dendrita y consecutivamente se toma el menor valor por neurona, donde la *j*-ésima neurona corresponde a la *j*-ésima clase. Por lo tanto, el error local se calcula a partir de la cantidad de objetos clasificados correctamente con respecto a la cantidad de objetos totales de la clase de interés. El pseudocódigo del cálculo del error local se presenta en la Tabla 3.13.

Tabla 3.13. Pseudocódigo para el cálculo del error local por clase de la red neuronal.

```
00|Function Calculo_error_local(τ, P^j_class, T^j_class, j)
01|Variables
02|     Int i, j, k, N_objetos, Clase, Cont_err, n_c
03|     float error_local, n_c, d, * x
04|Begin
05|     N_objetos ← size(P^j_class) // Se determina el tamaño del vector de entrada P
06|     Cont_err ← 0
07|     for(i = 0 to N_objetos)
08|          x ← P^j_class(i) //Se obtiene el vector i-ésimo de la base de datos
10|          n_c ← Numero de clusters(τ^j)
11|          for(k = 0 to n_c)
12|               d^j_k ← (x − μ_k)^T Σ^{-1}_k (x − μ_k)
13|          d_j ← argmin_k(d^j_k) //se almacena el valor de la mejor dendrita
16|          Clase ← d_j(class) //se toma la clase a la que pertenece el valor mínimo
17|          if(Clase == T^j_class(i))
18|               Cont_err ← Cont_err + 1
19|     error_local ← Cont_err / N_objetos
20|     return error_local
21|end
```

Por otro lado, el cálculo del error global requiere de todos los vectores de características almacenados en *P*, de la información que indica a cuál clase pertenecen *T*, y de la estructura de la red neuronal τ incluyendo las neuronas y dendritas generadas en los algoritmos anteriores. De esta forma, haciendo uso de la distancia de Mahalanobis, cada objeto de entrada se asocia a la neurona que contenga la menor distancia y se compara con la clase contenida en *T*, si la neurona corresponde con la clase a la que pertenece el objeto, se incrementa un contador que lleva el control de la cantidad de objetos clasificados correctamente. Por lo tanto, el error global se obtiene a partir de la cantidad de objetos clasificados correctamente entre la cantidad de objetos totales de *P*. El pseudocódigo del algoritmo del cálculo del error global se puede consultar en la Tabla 3.14.

Tabla 3.14. Pseudocódigo para el cálculo del error global de la red.

```
00|Function Calculo_error_global(τ, P, T)
01|Variables
02|     Int i, j, k, n_c, N_objetos, Nclass, Clase, Cont_err
03|     float error_global, * x, d
04|Begin
05|     N_objetos ← size(P) // Se determina el tamaño del vector de entrada P
06|     Cont_err ← 0, Nclass ← size(τ)
07|     for(i = 0 to N_objetos)
08|           x ← P(i) //Se obtiene el vector i-ésimo de la base de datos
09|           for(j=0 to Nclass)
10|                 n_c ← Numero de clusters(τ^j)
11|                 for(k = 0 to n_c)
12|                       d_k^j ← (x − μ_k)^T Σ_k^{-1} (x − μ_k)
13|                 d_j ← argmin_k(d_k^j) // Se almacena el valor de la mejor dendrita
14|           d ← argmin_j(d_j) // Se toma el valor mínimo de las neuronas de la red
15|           Clase ← d(class)// se toma la clase a la que pertenece el valor mínimo
16|           if(Clase == T(i))
17|                 Cont_err ← Cont_err + 1
18|     error_global ← Cont_err / N_objetos
19|     return error_global
20|end
```

Una vez terminado el proceso de clusterización de la red neuronal, se tiene la estructura final de la red, es decir, se tienen n neuronas con k^j dendritas cada una. Entonces, la subrutina EvDiff() aplica el algoritmo de evolución diferencial definido por en la ecuación (21) a la estructura de la red neuronal, generada por el proceso de clusterización, τ. Este algoritmo establece que se deben crear 3 estructuras a partir de τ, cada una multiplicada por un factor aleatorio diferente (a, b y c). Además, se toma en cuenta un factor aleatorio δ que multiplica el operador de diferencia entre b y c. Como es un proceso iterativo donde la red original, red padre, se compara con las redes hijas y se determina cual tiene mejor desempeño de acuerdo con lo descrito en el apartado 2.5.3, se tiene un límite de iteraciones y un error de umbral que permite obtener la mejor estructura para el conjunto de datos en el entrenamiento de la red. El pseudocódigo de la subrutina EvDiff() se muestra en la Tabla 3.15.

Tabla 3.15. Pseudocódigo para la aplicación del método de evolución diferencial.

```
00|Function EvDiff( τ )
01|Variables
02|     int i, k
03|     float ** a,** b,** c
04|     Cell ** S,** F
05|Begin
06|     i ← 0
07|     F ← τ,  error_padre ← error_global
08|     while(i < 100) //número de iteraciones que puede ser modificado
09|             a, b, c ← τ * rand()
10|             δ ← 10 * rand()
11|             S ← a + δ * (b − c)
12|             Calcular error_hijo//Se cacula utilizando la función de error global
13|             If(error_hijo < error_padre)
14|                     F ← S
15|                     if(error_hijo ≤ 0.1)
16|                             break
17|             i ← i + 1
18|     τ ← F
19|     return τ
20|end
```

Finalmente, para obtener el desempeño de la red neuronal en la fase de *testing*, se implementa la función de error global, recibiendo como parámetros el conjunto de vectores P_{test}, el array T_{test} y la estructura final de la red.

Adicionalmente, se puede implementar la red neuronal para clasificar un solo objeto con la función descrita en la Tabla 3.16, donde a partir de la distancia de Mahalanobis solo es necesaria la información de la red neuronal y el vector de características perteneciente al objeto de interés.

Tabla 3.16. Pseudocódigo para clasificar un solo objeto.

```
00|Function Clasificar_obj (τ, x)
01|Variables
02|     Int i, j, k, Nclass , Clase
03|     float error_global, n_c, d
04|Begin
05|     Nclass ← size(τ) // Se determina el tamaño del vector de entrada P
06|     for(j=0 to Nclass)
07|             n_c ← Numero de clusters(τ^j)
08|             for(k = 0 to n_c)
09|                     d_k^j ← (x − μ_k)^T Σ_k^−1 (x − μ_k)
10|             d_j ← argmin_k(d_k^j) // Se almacena el valor de la mejor dendrita
11|     d ← argmin_j(d_j) // Se toma el valor mínimo de las neuronas de la red
12|     Clase ← d(class)// se toma la clase a la que pertenece el valor mínimo
13|     return Clase
14|end
```

3.4 Resultados del modelado conceptual

En las pruebas de desempeño realizadas al sistema propuesto (HOG + DEN-*network*) y debido a la naturaleza de las bases de datos COIL-20 y ALOI, una imagen se considera como la ventana de detección, y contemplando que las imágenes de entrada son de 128×128 píxeles se opta por utilizar bloques y celdas cuadrados en la configuración de HOG. Se realizaron pruebas modificando el tamaño de las celdas y bloques para determinar cuál configuración motiva el mejor desempeño global del sistema. Para realizar un mejor análisis de resultados en el desempeño del sistema, en este primer experimento se optó por utilizar 2 clases y contemplar una separación entre los bloques de una celda, por lo que solo se varió el tamaño de la celda y el bloque. En la Tabla 3.17 se reportan las pruebas de desempeño de la red para la base de datos COIL-20, y en la Tabla 3.18 las pruebas hechas para la base de datos ALOI.

Tabla 3.17. Variación del tamaño de la celda y bloque para la base de datos COIL-20.

Tamaño de la celda (en píxeles)	Tamaño del bloque (en celdas)	N° de bloques por ventana	N° de elementos del vector de características	$error_{traning}$	$error_{testing}$
16x16	1x1	64	576	50	50
	2x2	49	1764	50	50
	3x3	36	2916	45.83	50
	4x4	25	3600	50	50
	5x5	16	3600	50	50
	6x6	9	2916	50	50
	7x7	4	1764	50	50
	8x8	1	576	50	50
32x32	1x1	16	144	62.50	61.11
	2x2	9	324	48.61	50
	3x3	4	324	47.22	52.77
	4x4	1	144	5.55	12.5
64x64	1x1	4	36	43.05	43.05
	2x2	1	36	54.16	52.77
128x128	**1x1**	**1**	**9**	**1.38**	**2.77**

Tabla 3.18. Variación del tamaño de la celda y bloque para la base de datos ALOI.

Tamaño de la celda (en píxeles)	Tamaño del bloque (en celdas)	N° de bloques por ventana	N° de elementos del vector de características	$error_{traning}$	$error_{testing}$
16x16	**1x1**	**64**	**576**	**11.11**	**0**
	2x2	**49**	**1764**	**0**	**0**
	3x3	**36**	**2916**	**0**	**0**
	4x4	25	3600	19.44	69.44
	5x5	16	3600	50	50
	6x6	9	2916	30.55	23.61
	7x7	4	1764	15.27	11.11
	8x8	1	576	50	50
32x32	1x1	16	144	50	50
	2x2	9	324	29.16	26.38
	3x3	4	324	44.44	41.66
	4x4	1	144	30.55	37.50
64x64	1x1	4	36	0	50
	2x2	1	36	0	11.11
128x128	**1x1**	**1**	**9**	**1.38**	**1.38**

Las Tablas 3.17 y 3.18 muestran que el mejor desempeño global del sistema ocurre con la configuración de HOG que tiene un tamaño de celda de 128×128 píxeles y un tamaño de bloque de 1×1 celdas, generando un vector de características de 9 elementos. Esto se debe principalmente a que como se reporta en [9], el desempeño de una DEN-*network* va en decremento cuando la dimensión de los vectores de características que debe procesar se incrementa. Esto repercute directamente en la estructura de la DEN-*network* debido a que al tener un vector de características con menor número de elementos existirán neuronas más simples. Este hecho es de relevante importancia en el diseño de la arquitectura hardware de la DEN-*network*, ya que una neurona simple minimiza los recursos necesarios en su implementación y también presentan un menor tiempo de propagación.

Las pruebas de desempeño del sistema y sus modificaciones fueron realizadas contemplando un conjunto de 2, 3, 5, 10, 15 y 20 clases de las bases de datos COIL-20 y ALOI. En las Figuras 3.6 y 3.7 se muestra la primera imagen de cada una de las 20 clases utilizadas, para un conjunto de dos clases se utilizan las clases 1 y 2, para conjuntos de tres clases se utilizan las clases 1, 2 y 3, así sucesivamente.

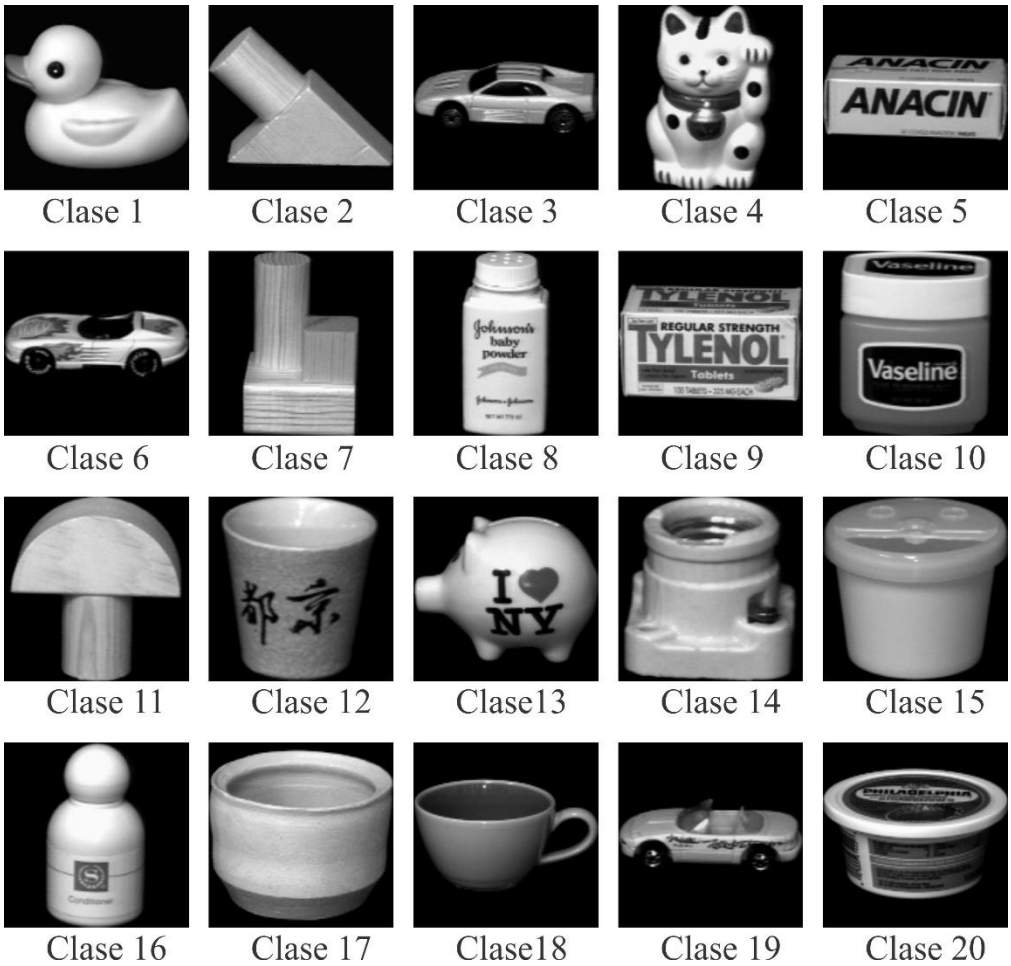

Figura 3.6. Muestra de las 20 clases de la base de datos COIL-20, utilizadas en la prueba de desempeño del modelado conceptual.

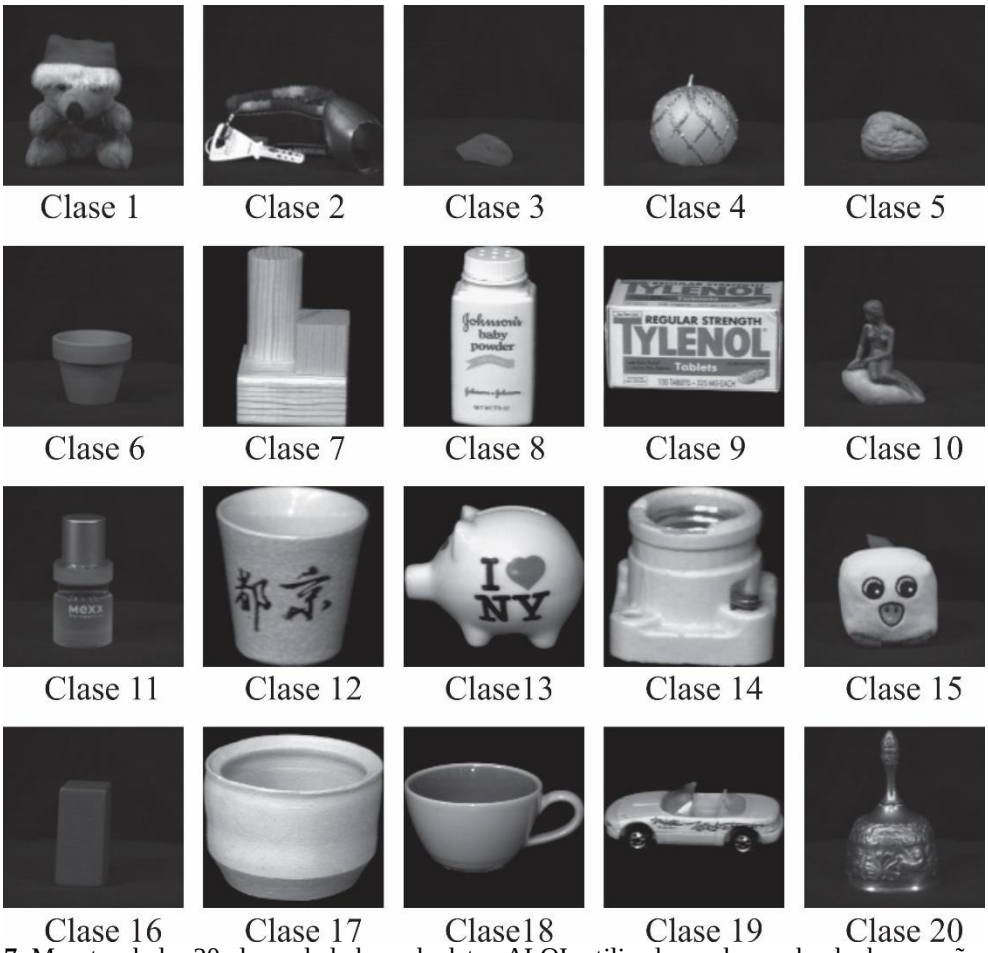

Figura 3.7. Muestra de las 20 clases de la base de datos ALOI, utilizadas en la prueba de desempeño del modelado conceptual.

Como se describió en el apartado 3.2. se hicieron modificaciones en partes de los algoritmos de las fases de HOG para favorecer su implementación en un FPGA. Cada cambio en el algoritmo fue probado previamente en el modelado conceptual del sistema para verificar como afectaba en su desempeño; la red neuronal con la que se probaron las modificaciones se forma mediante los vectores de características de 9 elementos y las pruebas se realizaron con 2, 3, 5, 10, 15 y 20 clases. La modificación 1, descrita en el apartado 3.4 hace referencia al truncamiento de los elementos de la matriz de covarianza inversa, en la Tabla 3.19 se puede observar que el desempeño del sistema, para las bases de datos COIL-20 y ALOI, mejora en la mayoría de los casos.

Tabla 3.19. Comparación de resultados entre el sistema original y la modificación en las matrices de covarianza inversa.

N° de clases	COIL-20				ALOI			
	Sin modificación		Modificación 1		Sin modificación		Modificación 1	
	$error_{traning}$	$error_{testing}$	$error_{traning}$	$error_{testing}$	$error_{traning}$	$error_{testing}$	$error_{traning}$	$error_{testing}$
2	1.38	2.77	**1.38**	**1.38**	1.38	1.38	**0**	**0**
3	2.77	1.85	**0.92**	**2.77**	0.92	0.92	**0.92**	**0.92**
5	3.33	2.77	**3.33**	**2.22**	1.66	6.11	**1.66**	**6.11**
10	8.61	17.5	**5.55**	**16.94**	2.77	11.94	**2.77**	**11.11**
15	**5**	**10**	7.03	15	3.51	10.55	**3.33**	**9.25**
20	10.55	21.52	**9.58**	**19.58**	**6.38**	**14.86**	9.16	20

La modificación 2, descrita en el apartado 3.2.1 y realizada en la magnitud y dirección de los gradientes, acumulando la mediación anterior, dio como resultado un aumento en la eficiencia del sistema, tanto para la base de datos COIL-20 como para ALOI. Para sostener esta aseveración, en las Tabla 3.20 y 3.21 se muestran los resultados de la comparación del desempeño del sistema formado con algoritmo HOG sin modificaciones y con HOG modificado en el cálculo de la magnitud y dirección.

Tabla 3.20. Comparación de resultados entre el sistema original y HOG modificado para la magnitud y dirección para la base de datos COIL-20.

N° de clases	Base de datos COIL-20					
	Sistema original		Modificación 1		Modificación 2	
	$error_{traning}$	$error_{testing}$	$error_{traning}$	$error_{testing}$	$error_{traning}$	$error_{testing}$
2	1.38	2.77	1.38	1.38	**0**	**0**
3	2.77	1.85	0.92	2.77	**0.92**	**0.95**
5	3.33	2.77	3.33	2.22	**3.88**	**5.14**
10	8.61	17.5	5.55	16.94	**3.33**	**5.42**
15	5	10	7.03	15	**4.62**	**6.66**
20	10.55	21.52	9.58	19.58	**3.33**	**8.14**

Tabla 3.21. Comparación de resultados entre el sistema original y HOG modificado para la magnitud y dirección para la base de datos ALOI.

N° de clases	Base de datos ALOI					
	Sistema original		Modificación 1		Modificación 2	
	$error_{traning}$	$error_{testing}$	$error_{traning}$	$error_{testing}$	$error_{traning}$	$error_{testing}$
2	1.38	1.38	0	0	**0**	**0**
3	0.92	0.92	0.92	0.92	**0**	**0**
5	1.66	6.11	1.66	6.11	**0**	**0**
10	2.77	11.94	2.77	11.11	**1.11**	**1.71**
15	3.51	10.55	3.33	9.25	**1.11**	**2.09**
20	6.38	14.86	9.16	20	**1.25**	**4.42**

Por su parte, la modificación 3, realizada en la división para la distribución en los contenedores del histograma, presenta una disminución en el desempeño del sistema para la base de datos COIL-20 (ver Tabla 3.22), mientras que para la base de datos ALOI el desempeño mejora (ver Tabla 3.23). Los resultados presentados son acumulando la modificación anterior con la modificación actual y se comparan con el desempeño del sistema con la versión de HOG original y la versión que incluye la modificación 2.

Tabla 3.22. Comparación de resultados HOG original, modificación 2 y 3 para la base de datos COIL-20.

N° de clases	Base de datos COIL-20					
	Sistema original		Modificación 2		Modificación 3	
	$error_{traning}$	$error_{testing}$	$error_{traning}$	$error_{testing}$	$error_{traning}$	$error_{testing}$
2	1.38	2.77	**0**	**0**	4.16	8.57
3	2.77	1.85	**0.92**	**0.95**	2.77	5.71
5	3.33	2.77	**3.88**	**5.14**	2.77	2.28
10	8.61	17.5	**3.33**	**5.42**	3.61	8.57
15	5	10	**4.62**	**6.66**	3.51	6.47
20	10.55	21.52	**3.33**	**8.14**	2.63	7.28

Tabla 3.23. Comparación de resultados HOG original, modificación 2 y 3 para la base de datos ALOI.

N° de clases	Base de datos ALOI					
	Sistema original		Modificación 2		Modificación 3	
	$error_{traning}$	$error_{testing}$	$error_{traning}$	$error_{testing}$	$error_{traning}$	$error_{testing}$
2	1.38	1.38	0	0	**0**	**0**
3	0.92	0.92	0	0	**0**	**0**
5	1.66	6.11	0	0	**0**	**0**
10	2.77	11.94	1.11	1.71	**1.11**	**0.85**
15	3.51	10.55	1.11	2.09	**1.11**	**2.28**
20	6.38	14.86	1.25	4.42	**2.08**	**3.28**

La modificación 4, discutida en el apartado 3.2.4, se realiza al proceso de normalización omitiendo la raíz cuadrada de la ecuación (9). Comparado con las modificaciones anteriores, se observa que con esta modificación el desempeño del sistema disminuye para la base de datos ALOI, mientras que para la base de datos COIL-20 aumenta. Sin embargo, los resultados de eficiencia del sistema, para la base de datos ALOI, presentan un mejor desempeño con respecto a la versión original del algoritmo HOG. Los resultados del desempeño del sistema con la modificación 4 y la comparativa con las versiones anteriores pueden ser consultados en la Tabla 3.24 y 3.25.

Tabla 3.24. Comparación entre HOG original, la tercera y cuarta modificación para COIL-20.

N° de clases	Base de datos COIL-20					
	Sistema original		Modificación 3		Modificación 4	
	$error_{traning}$	$error_{testing}$	$error_{traning}$	$error_{testing}$	$error_{traning}$	$error_{testing}$
2	1.38	2.77	4.16	8.57	**0**	**2.77**
3	**2.77**	**1.85**	2.77	5.71	0	2.77
5	3.33	2.77	2.77	2.28	**0**	**1.11**
10	8.61	17.5	3.61	8.57	**3.61**	**8.33**
15	**5**	**10**	3.51	6.47	5.92	10.74
20	10.55	21.52	2.63	7.28	**6.11**	**12.22**

Tabla 3.25. Comparación entre HOG original, la tercera y cuarta modificación para ALOI.

N° de clases	Base de datos ALOI					
	Sistema original		Modificación 3		Modificación 4	
	$error_{traning}$	$error_{testing}$	$error_{traning}$	$error_{testing}$	$error_{traning}$	$error_{testing}$
2	1.38	1.38	**0**	**0**	0	0
3	0.92	0.92	**0**	**0**	0	0
5	1.66	6.11	**0**	**0**	0	0
10	2.77	11.94	**1.11**	**0.85**	2.5	5.27
15	3.51	10.55	**1.11**	**2.28**	2.22	5.37
20	6.38	14.86	**2.08**	**3.28**	4.58	8.05

Finalmente, la modificación 5 mostrada en el apartado 3.2.4, sobre el desplazamiento de los elementos del vector de características, se observa de acuerdo a las Tablas 3.26 y 3.27 que el desempeño para las bases de datos COIL-20 y ALOI sufre una mejora respecto a la modificación 4, principalmente para clases mayores a 5 elementos, sin embargo, el desempeño obtenido sigue siendo inferior a la modificación 3 para un número mayor a 5 de clases. Adicionalmente es importante recalcar que la eficiencia del sistema final modificado (contemplando las 5 modificaciones) es mayor a la del sistema original sin cambios en el algoritmo de HOG.

Tabla 3.26. Comparación entre HOG original, la cuarta y quinta modificación para COIL-20.

N° de clases	Base de datos COIL-20					
	Sistema original		Modificación 4		Modificación 5	
	$error_{traning}$	$error_{testing}$	$error_{traning}$	$error_{testing}$	$error_{traning}$	$error_{testing}$
2	1.38	2.77	0	2.77	**0**	**2.77**
3	**2.77**	**1.85**	0	2.77	**0**	**2.77**
5	3.33	2.77	0	1.11	**0**	**1.11**
10	8.61	17.5	3.61	8.33	**7.77**	**3.61**
15	5	10	5.92	10.74	**4.07**	**8.14**
20	10.55	21.52	6.11	12.22	**5.41**	**11.80**

Tabla 3.27. Comparación entre HOG original, la cuarta y quinta modificación para ALOI.

N° de clases	Base de datos ALOI					
	Sistema original		Modificación 4		Modificación 5	
	$error_{traning}$	$error_{testing}$	$error_{traning}$	$error_{testing}$	$error_{traning}$	$error_{testing}$
2	1.38	1.38	0	0	**0**	**0**
3	0.92	0.92	0	0	**0**	**0**
5	1.66	6.11	0	0	**0**	**0**
10	2.77	11.94	2.5	5.27	**0.83**	**5.83**
15	3.51	10.55	2.22	5.37	**7.22**	**9.25**
20	6.38	14.86	4.58	8.05	**1.25**	**6.11**

Para mostrar el desempeño del sistema propuesto, se hará uso de matrices de confusión. Una matriz de confusión es una tabla frecuentemente usada para describir el comportamiento que presenta un modelo de clasificación al ser aplicado sobre un conjunto de datos de prueba conocido y que representa el resumen de resultados de predicción generados por el clasificador. Considerando que un clasificador tiene por objetivo predecir a cuál clase pertenece una nueva instancia basándose en el conocimiento aprendido a partir de instancias anteriores, la estructura de una matriz de confusión define a las clases reales en las filas (cada fila incluye a todas las instancias de una clase particular, clasificadas correcta e incorrectamente) y a las clases previstas por el modelo en las columnas (cada columna incluye el número total de predicciones de cada clase, correctas e incorrectas). Esta estructura sirve para mostrar de forma explícita cuándo una clase es confundida con otra, a partir de un conteo de los aciertos y errores de cada una de las clases en el proceso de clasificación.

De esta forma, para cada clase, la matriz agrupa los casos positivos que fueron clasificados correctamente, conocidos como verdaderos positivos (TP, *True Positives*); los casos positivos que fueron clasificados incorrectamente, falsos positivos (FP, *False Positives*); los casos negativos que fueron clasificados correctamente, verdaderos negativos (TN, *True Negatives*); y los casos negativos que fueron clasificados incorrectamente, falsos negativos (FN, *False Negatives*).

Un aspecto importante de una matriz de confusión son sus métricas asociadas, siendo las más importantes las mencionadas a continuación,

Exactitud (*accuracy*). Indica la fracción de observaciones correctamente clasificadas.

$$\text{Exactitud} = \frac{TP + TN}{TP + TN + FP + FN} \tag{25}$$

Tasa de error (Te). Indica que tan frecuente el sistema se equivoca en sus predicciones.

$$Te = \frac{FP + FN}{TP + TN + FP + FN} \tag{26}$$

Tasa de verdaderos positivos (TPR, *True Positives Rate*), sensibilidad. Indica la fracción de observaciones positivas que han sido predichas correctamente.

$$\text{TPR} = \frac{TP}{TP + FN} \tag{27}$$

Tasa de verdaderos negativos (TNR, *True Negatives Rate*), especificidad. Indica la fracción de observaciones negativas que han sido correctamente clasificadas.

$$\text{TNR} = \frac{TN}{TN + FP} \tag{28}$$

Precisión. Indican la fracción de predicciones positivas que en realidad son positivas.

$$\text{Presición} = \frac{TP}{TP + FP} \tag{29}$$

Por lo tanto, a partir de la matriz de confusión para 5 clases de la base de datos COIL-20 (ver Tabla 3.28), con el sistema sin modificación en el algoritmo de HOG, y utilizando las ecuaciones (25), (26), (27), (28) y (29) se obtienen los parámetros de Exactitud, Te, TPR, TNR y Precisión de cada clase, por lo que además se obtiene el promedio general (avg, *average*) de cada parámetro, de esta forma en la Tabla 3.29 se puede visualizar los resultados obtenidos.

Tabla 3.28. Matriz de confusión del sistema original para 5 clases de la base de datos COIL-20.

Clase	Predicciones del sistema original				
	(a)	(b)	(c)	(d)	(e)
(a)	33	1	0	2	0
(b)	0	35	0	1	0
(c)	0	0	35	0	1
(d)	0	0	0	36	0
(e)	0	0	0	0	36

Tabla 3.29. Resultados de la matriz de confusión de la Tabla 3.28.

Clase\rate	TP	TPR	TN	TNR	Exactitud	Te	Precisión
(a)	33	0.9166	144	1	0.9833	0.0167	1
(b)	35	0.9722	143	0.9930	0.9888	0.0111	0.9722
(c)	35	0.9722	144	1	0.9944	0.0056	1
(d)	36	1	141	0.9791	0.9833	0.0166	0.9230
(e)	36	1	143	0.9930	0.9944	0.0056	0.9729
avg		0.9722		0.9930	0.9888	0.0111	0.9736

De igual manera, analizando la matriz de confusión para el mismo número de clases con el sistema modificado en el algoritmo de HOG (ver Tabla 3.30), se obtienen los parámetros correspondientes y se muestran los resultados en la Tabla 3.31.

Tabla 3.30. Matriz de confusión del sistema final modificado para 5 clases de la base de datos COIL-20.

Clase	Predicciones del sistema final modificado				
	(a)	(b)	(c)	(d)	(e)
(a)	36	0	0	0	0
(b)	0	36	0	0	0
(c)	0	0	36	0	0
(d)	0	2	0	34	0
(e)	0	0	0	0	36

Tabla 3.31. Resultados de la matriz de confusión de la Tabla 3.30.

Clase\rate	TP	TPR	TN	TNR	Exactitud	Te	Precisión
(a)	36	1	144	1	1	0	1
(b)	36	1	142	0.9861	0.9889	0.0111	0.9474
(c)	36	1	144	1	1	0	1
(d)	34	0.9444	144	1	0.9889	0.0111	1
(e)	36	1	144	1	1	0	1
avg		0.9889		0.9972	0.9956	0.0044	0.9895

Como se puede notar en las Tablas 3.29 y 3.31 el promedio general de los parámetros Exactitud, Te, TPR, TNR y Precisión, brinda la información necesaria para la comprensión del comportamiento en el desempeño del sistema, por lo que se optó en calcular dichos parámetros para el numero de clases mostradas en las modificaciones del algoritmo de HOG y brindar una mejor perspectiva de como el conjunto de modificaciones propuestas afecta el desempeño del sistema, ver Tabla 3.32 y 3.33.

Tabla 3.32. Comparación de los datos de la matriz de confusión de la base de datos COIL-20 para diferente número de clases.

N° de clases	Base de datos COIL-20									
	Sistema original (avg)					Sistema final modificado (avg)				
	TPR	TNR	Exact.	Te	Precisión	TPR	TNR	Exact.	Te	Precisión
2	0.9722	0.9722	0.9722	0.0277	0.9736	0.9722	0.9722	0.9722	0.0278	0.9737
3	0.9814	0.9907	0.9876	0.0123	0.9824	0.9722	0.9861	0.9815	0.0185	0.9744
5	0.9722	0.9930	0.9888	0.0111	0.9736	0.9889	0.9972	0.9956	0.0044	0.9895
10	0.8250	0.9805	0.9649	0.0350	0.8337	0.9222	0.9914	0.9844	0.0156	0.9322
15	0.9000	0.9928	0.9866	0.0133	0.9072	0.9185	0.9942	0.9891	0.0109	0.9251
20	0.7847	0.9886	0.9784	0.2152	0.8500	0.8819	0.9938	0.9882	0.0118	0.9148

Tabla 3.33. Comparación de los datos de la matriz de confusión de la base de datos ALOI para diferente número de clases.

N° de clases	Base de datos ALOI									
	Sistema original (avg)					Sistema final modificado (avg)				
	TPR	TNR	Exact.	Te	Precisión	TPR	TNR	Exact.	Te	Precisión
2	0.9861	0.9861	0.9861	0.0138	0.9864	1	1	1	0	1
3	0.9907	0.9537	0.9938	0.0061	0.9909	1	1	1	0	1
5	0.9388	0.9847	0.9755	0.0244	0.9398	1	1	1	0	1
10	0.8805	0.9867	0.9761	0.0238	0.8929	0.9417	0.9935	0.9883	0.0117	0.9484
15	0.8944	0.9924	0.9859	0.0140	0.9073	0.9074	0.9934	0.9877	0.0123	0.9521
20	0.8513	0.9921	0.9851	0.0148	0.9018	0.9389	0.9968	0.9939	0.0061	0.9498

De esta forma se puede corroborar que las modificaciones del sistema en el algoritmo de HOG, para ambas bases de datos, brindan un mejor desempeño de clasificación con respecto al sistema original. Cabe recalcar que la mejora en el desempeño se hace más notoria para un mayor número de clases.

Capítulo 4

Implementación en Hardware

La implementación de una arquitectura hardware es un proceso que, considerando el modelado conceptual que precisa el comportamiento de un algoritmo, inicia con el diseño de la estructura de un sistema digital que lo define, donde se especifican los diferentes componentes que la integran y la interacción existente entre ellos. Posteriormente, se debe desarrollar una descripción funcional de cada uno de los componentes mediante técnicas de modelado acordes al componente descrito. En el presente capitulo se muestra el diseño de las arquitecturas hardware de los algoritmos del descriptor HOG y clasificador DEN-*network*, así como los métodos utilizados para la disminución de recursos e incremento de la eficiencia en términos de tiempo de procesamiento.

4.1 Esquema general del sistema propuesto

Como se expresó en los apartados anteriores, el sistema propuesto en este trabajo de tesis consiste en el diseño, modelado e implementación de una arquitectura hardware de un sistema de reconocimiento integrado por un elemento extractor de características y un elemento clasificador. Con la finalidad de ofrecer al usuario una forma simple de interactuar con el sistema propuesto, fue necesario desarrollar una interfaz de usuario que permite cumplir con tareas básicas como el envío de los datos del objeto que se desea clasificar, en este caso todos los píxeles que componen la imagen de entrada, y la recepción del resultado del procesamiento realizado por el sistema, la clase a la que pertenece el objeto.

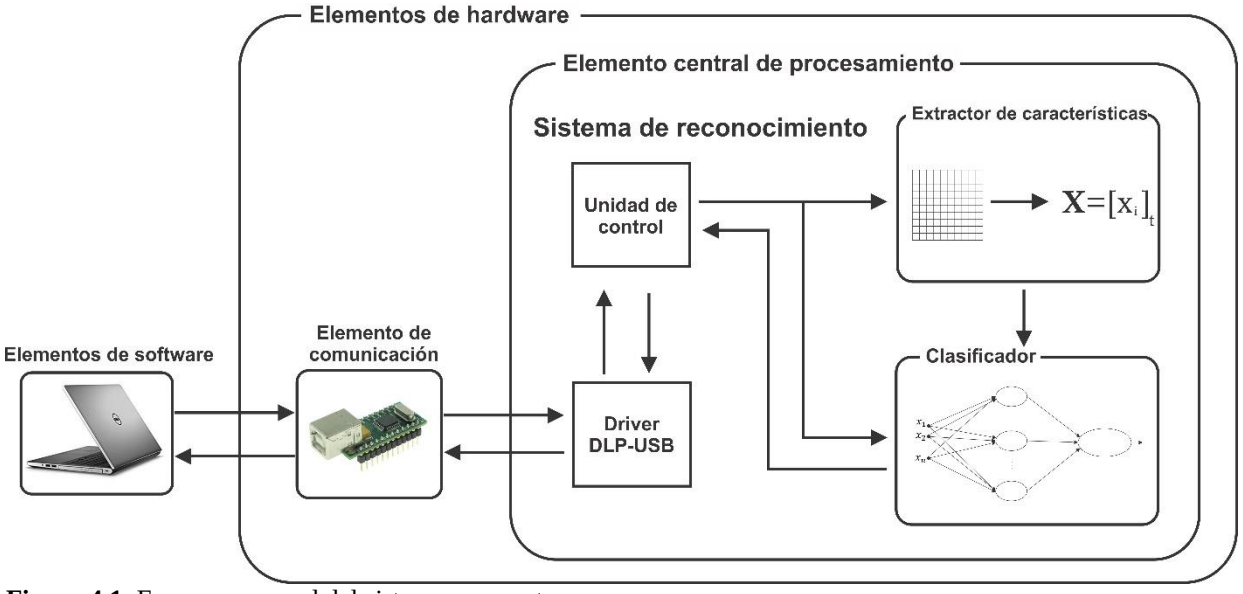

Figura 4.1. Esquema general del sistema propuesto.

De esta forma, mediante una interfaz desarrollada en un lenguaje de alto nivel para una PC, se toma un elemento de las bases de datos (COIL-20 y ALOI), que contienen imágenes previamente segmentadas, y se envía a través de un elemento de comunicación a la arquitectura hardware, por ende, los datos enviados son vistos como entradas del sistema en la arquitectura hardware, en respuesta el sistema regresa el resultado de clasificación del objeto.

Para facilitar la comprensión del esquema general del sistema (ver Figura 4.1), este se puede dividir en dos secciones. La primera integra a los elementos de hardware del sistema, el FPGA como elemento central de procesamiento y al controlador de comunicaciones USB. La segunda sección incluye a los elementos de software, la interfaz de usuario y el modelado mediante un HDL del descriptor y la red neuronal.

4.1.1 Elemento central de procesamiento

En capítulos anteriores se han vertido los argumentos que fundamentan el uso de un FPGA como elemento central de procesamiento del sistema propuesto en este trabajo de tesis. La implementación de un sistema basado en FPGA requiere el diseño de una arquitectura hardware del algoritmo elegido para dar soporte a la aplicación, para posteriormente describirlo mediante una herramienta de modelado. En la implementación de los algoritmos de HOG y la DEN-*network* en lógica reconfigurable, se ha elegido como herramienta EDA (*Electronic Design Automation*) hardware la plataforma de desarrollo Nexys 3 (ver Figura 4.2), diseñada y manufacturada por la compañía Digilent Inc. El dispositivo principal de esta tarjeta es un FPGA Spartan6-XC6SLX16 de la compañía Xilinx Inc. Además, la tarjeta Nexys 3 incluye diversos periféricos que facilitan en desarrollo de aplicaciones, dichos periféricos son mencionados a continuación,

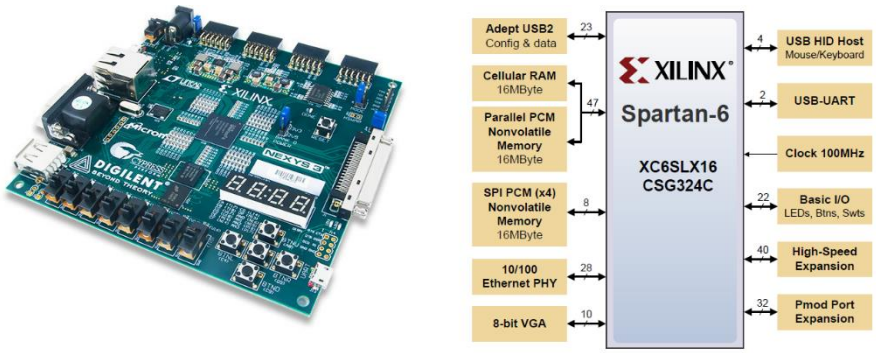

Figura 4.2. Estructura general de la tarjeta Nexys 3.

- Oscilador CMOS de 100MHz pudiendo elevar la frecuencia hasta más de 500MHz.
- 16 Mbytes de celula RAM y 16 Mbytes SPI (quad mode) PCM.
- Puerto USB para la programación y configuración de la tarjeta.
- Puerto USB-UART y USB-HID.
- 72 conectores de entrada/salida.
- Conectores de entrada y salida de propósito general.
- 8 LEDs, 4 displays de 7 segmentos, 5 botones, 8 switchs.

Adicionalmente, y debido al hecho que la tarjeta fue diseñada para ser un dispositivo de desarrollo, tiene la importante característica de permitir conectarle módulos adiciones de manera sencilla, tal es el caso del módulo de comunicación.

4.1.2 Elemento de comunicación

La comunicación entre la Interfaz de usuario y el elemento central de procesamiento es realizada mediante el protocolo de comunicaciones USB. El Elemento de comunicación se encarga de cumplir con esta tarea, permitiendo el intercambio de información entre una aplicación software y una la aplicación en el FPGA. Para cumplir con esta tarea, se ha elegido el controlador DLP-USB245M debido a que permite enviar y recibir hasta 8 millones de bits (1 Megabyte) por segundo mediante un puerto USB. El chip de comunicación utilizado por este dispositivo es el FT8U245AM, desarrollado por la compañía FTDI Ltd. El DLP-USB245M establece una interfaz simple con la aplicación software mediante bibliotecas de vínculos dinámicos, permitiendo accederlo mediante un lenguaje de alto nivel, Python en este caso, como simples funciones de envío y recepción de tramas de datos de 8 bits. Para el sistema de reconocimiento en el FPGA, el DLP-USB245M es visto como un simple componente de entrada y salida de propósito general que puede ser manipulado a través de un módulo de control.

4.1.3 Interfaz de usuario PC-FPGA

La interfaz de usuario es una aplicación software desarrollada mediante el lenguaje de programación Python que tiene como propósito principal fungir como complemento del sistema de reconocimiento, enviando los datos necesarios, de una imagen (objeto) de interés, para que pueda cumplir con su función. De igual manera la interfaz puede utilizarse como elemento evaluador de la arquitectura diseñada, pues al enviar n imágenes diferentes se puede obtener un porcentaje de eficiencia en el reconocimiento de objetos.

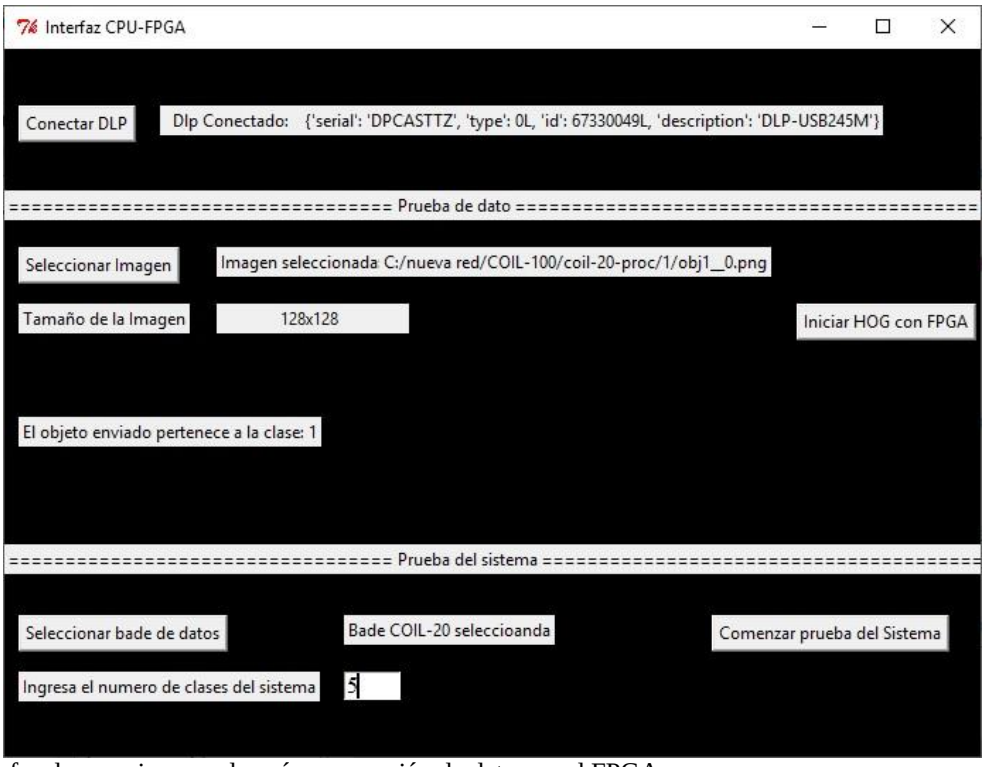

Figura 4.3. Interfaz de usuario para el envío y recepción de datos en el FPGA.

Como se puede observar en la Figura 4.3, la interfaz desarrollada se compone de 3 partes para la evaluación del sistema del FPGA. La primera de ellas se encarga de enlazar la conexión entre la PC y el elemento de comunicación (DLP-USB245M). La segunda parte sirve para enviar un objeto individual (imagen) al sistema de reconocimiento, obteniendo por respuesta la clase en la que fue clasificado. En esta sección la interfaz muestra el tamaño y dirección de la imagen seleccionada.

Finalmente, la tercera sección de la interfaz es la encargada de verificar el desempeño de clasificación del sistema diseñado, para realizar dicha acción se debe de seleccionar una base de datos y un determinado número de clases. Las imágenes seleccionadas son enviadas al FPGA, y al obtener la respuesta de cada objeto, esta es comparada con la clase de la cual se obtuvo, y si es igual se incrementa un contador, que permite, al final de la implementación, obtener el porcentaje de error del sistema para n clases, este resultado es mostrado en consola. Cabe recalcar que a pesar de que la interfaz fue diseñada para enlazar cualquier base de datos, el sistema en el FPGA solo está diseñado para efectuar las operaciones de las bases de datos COIL-20 y ALOI, debido a que el sistema espera recibir 16,384, correspondientes a una imagen de 128×128 píxeles.

La interfaz se compone de 5 botones para realizar las acciones descritas en la Tabla 4.1.

Tabla 4.1. Descripción de las funciones de la interfaz de acuerdo con el botón que se presiona.

Botón	Función
Conectar DLP	Se conecta con el DLP-USB245M.
Seleccionar Imagen	Abre la base de datos para seleccionar una imagen.
Iniciar HOG con FPGA	Envía una imagen en el FPGA y este regresa un vector de características.
Seleccionar base de datos	Toma la dirección de la base de datos seleccionada.
Comenzar prueba del sistema	Comienza la prueba del sistema para *n* clases en el FPGA.

4.1.4 Modelado de la arquitectura mediante un HDL

Como se ha mencionado, una vez diseñadas las arquitecturas hardware de los algoritmos HOG y DEN-*network*, estas son modelados utilizando el lenguaje descriptor de hardware VHDL. Debido a que se trata de un modelado de hardware que utiliza un enfoque de software, se considera que forma parte de la sección de software. Para cumplir con esta tarea, es necesario la ayuda de una herramienta de software que permita realizar diseños de electrónica digital, por lo que la herramienta seleccionada es *Xilinx ISE Desing Suite 14.7*. Esta herramienta permite diseñar, editar, simular, exportar, importar y sintetizar diseños basados en FPGA. Adicionalmente, la herramienta permite crear estructuras hardware mediante esquemáticos o lenguajes de descripción de hardware (VHDL o Verilog).

4.2 Diseño, modelado e implementación de arquitecturas hardware de los algoritmos HOG y DEN-*network*

El sistema de reconocimiento propuesto en este trabajo de tesis está integrado por dos fases de un sistema de reconocimiento, extracción de características, realizada mediante HOG, y clasificación, realizada mediante DEN-*network*, se trata de un sistema síncrono que trabaja con una frecuencia de reloj de 100MHz. El diseño de este sistema se basa en la metodología descendente (*top-down*). Esta metodología permite generar diseños jerárquicos y enfocados al reúso, facilitando la evolución del sistema. La primera división obtenida al aplicar la metodología genera 4 módulos: **Unidad de control general** (**UniCtrl_General**), **HOG**, **DEN-network** y **Controlador USB** (ver Figura 4.4).

La **UniCtrl_General** es la responsable de administrar el inicio de los procesos de cada fase del sistema, es decir, se encarga de indicarle al módulo de **HOG** cuando comenzar a crear el vector de características y al módulo **DEN-network** cuando está listo el vector para realizar su clasificación. Al estar interconectados los módulos **HOG**, **DEN-network** y el **Controlador USB**, el flujo de información entre cada proceso puede utilizarse de manera sencilla. Las funciones de las señales involucradas en esta interacción se muestran en la Tabla 4.2.

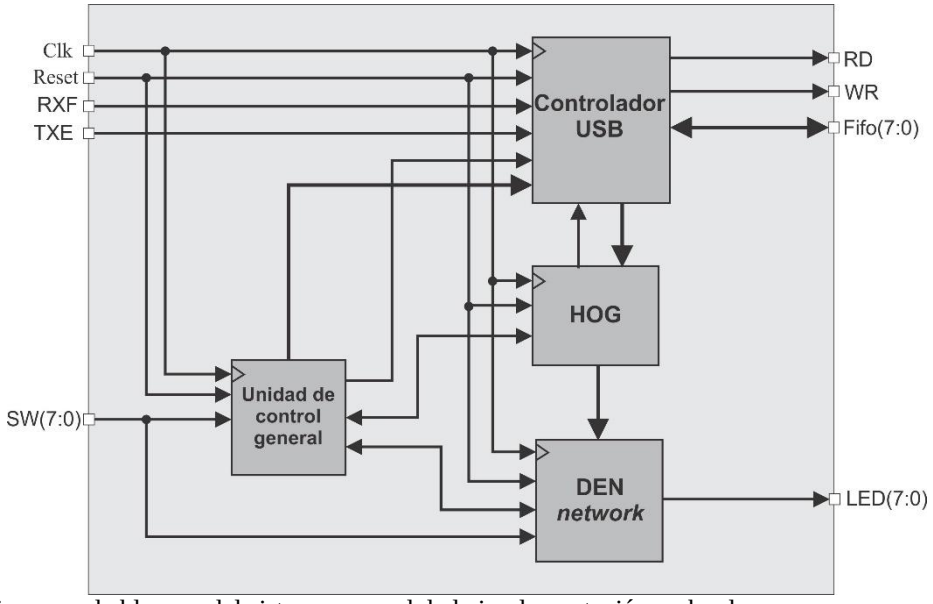

Figura 4.4. Diagrama de bloques del sistema general de la implementación en hardware.

Tabla 4.2. Descripción de las señales Entrada/Salida del esquemático superior del sistema de reconocimiento.

Señal	Descripción
Clk	(Entrada) Señal de reloj del sistema, 100MHz.
Reset	(Entrada) Señal de reset para todos los módulos.
RXF	(Entrada) Señal proveniente del DLP-USB245M, cuando el nivel lógico está en bajo indica que hay un dato disponible en el buffer FIFO.
TXE	(Entrada) Señal proveniente del DLP-USB245M, cuando el nivel lógico de la señal está en bajo indica que se puede escribir en el buffer FIFO.
SW[7:0]	(Entrada) Señal proveniente de los switches de la tarjeta Nexys 3.
RD	(Salida) Señal que va al DLP-USB245M para habilitar la secuencia de lectura.
WR	(Salida) Señal que va al DLP-USB245M para habilitar la secuencia de escritura.
LED[7:0]	(Entrada/Salida) Señal que va de a los LEDs de la tarjeta Nexys 3.
Fifo[7:0]	(Entrada/Salida) Bus de datos bidireccional proveniente del DLP-USB245M.

Para facilitar la compresión del proceso de reconocimiento de un objeto, se explicará primeramente el módulo encargado de la trasmisión de datos, consecutivamente se desarrollará la explicación de la **UniCtrl_General**, seguida del módulo encargado de implementar el algoritmo de HOG y finalmente el módulo clasificador que contiene a la DEN-*network*. En la Figura 4.5 se muestra el esquemático superior que se genera mediante el software *Xilinx ISE Desing Suite 14.7*; la función de los módulos y las señales que intervienen en ellos, serán explicados en los apartados posteriores.

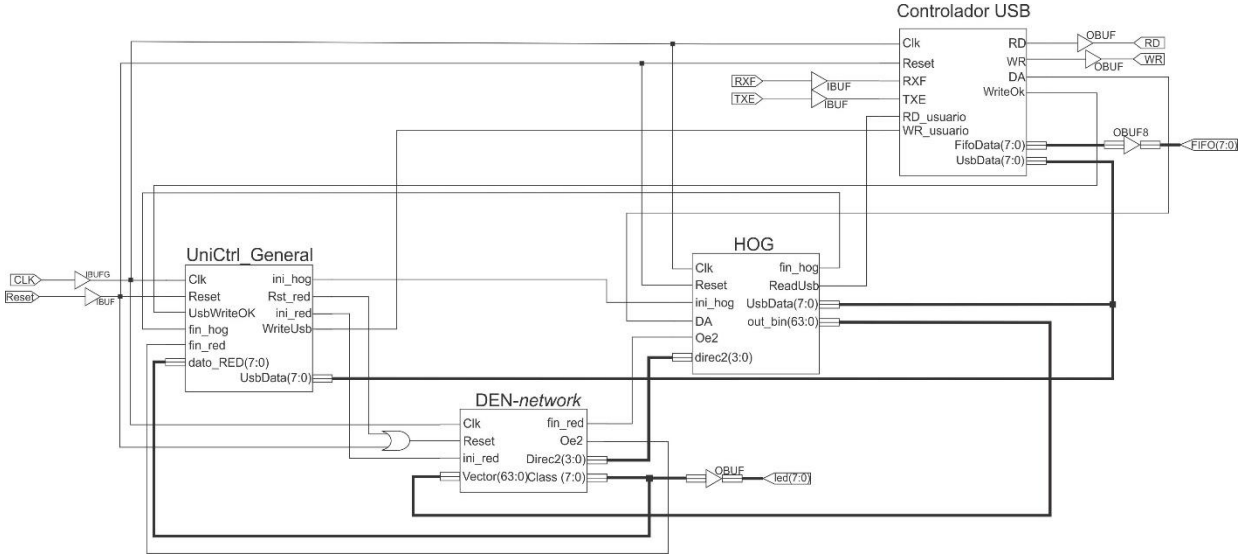

Figura 4.5. Esquemático superior del sistema de reconocimiento en el FPGA.

4.3 Controlador USB

El módulo **Controlador USB** administra la operación del DLP-USB245M, confiriendo al sistema de reconocimiento implementado en el FPGA la capacidad de intercambiar información con la Interfaz de usuario. Por este medio, el sistema recibe la imagen que debe ser procesada y envía los resultados parciales y finales obtenidos.

La Figura 4.3 muestra el símbolo del **Controlador USB** y en la Tabla 4.3 se describen las señales que definen la interfaz con el dispositivo DLP-USB245M y con el sistema de reconocimiento implementado en el FPGA.

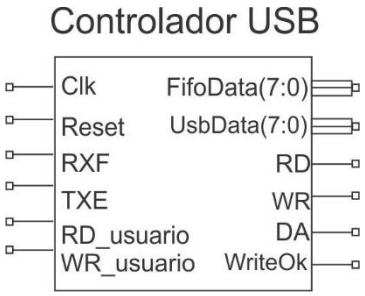

Figura 4.6. Símbolo del módulo **Controlador USB**.

Tabla 4.3. Descripción de las señales Entrada/Salida del módulo **Controlador USB.**

Señal	Descripción
Clk	(Entrada) Señal de reloj del sistema a 100MHz.
Reset	(Entrada) Señal de reset para las máquinas de estado de escribir y leer datos.
RXF	(Entrada) Señal proveniente del DLP-USB245M, cuando el nivel lógico está en bajo indica que hay un dato disponible en el buffer FIFO.
TXE	(Entrada) Señal proveniente del DLP-USB245M, cuando el nivel lógico de la señal está en bajo indica que se puede escribir en el buffer FIFO.
RD_usuario	(Entrada) Señal proveniente del sistema de reconocimiento que inicia el proceso de lectura en la máquina de estados.
WR_usuario	(Entrada) Señal proveniente del sistema de reconocimiento que inicia el proceso de escritura en la máquina de estados.
RD	(Salida) Señal que va al DLP-USB245M para habilitar la secuencia de lectura.
WR	(Salida) Señal que va al DLP-USB245M para habilitar la secuencia de escritura.
DA	(Salida) Indica que hay un dato listo para su lectura en el buffer FIFO.
WriteOK	(Salida) Indica que el proceso de escritura en el DLP-USB245M ha terminado.
UsbData[7:0]	(Entrada/Salida) Bus de datos proveniente del sistema de reconocimiento conectado al módulo de trasmisión de datos.
FifoData[7:0]	(Entrada/Salida) Bus de datos bidireccional proveniente del DLP-USB245M.

El **Controlador USB** está formado por los procesos de escritura y lectura del DLP-USB245M. A través del primero, el sistema de reconocimiento puede enviar información a la interfaz de usuario; mediante el segundo, recibe información proveniente de la misma fuente.

El diagrama de tiempos del proceso de lectura del dispositivo DLP-USB245M es mostrado en la Figura 4.7, la Tabla 4.4 contiene las especificaciones temporales que dicho diagrama debe cumplir. Esta información fue utilizada para realizar el modelado del proceso de lectura. El comportamiento de este proceso fue modelado mediante la máquina de estados finitos (FSM, *Finite State Machine*) ilustrada en la Figura 4.8. A continuación se detallan aspectos importantes de su funcionamiento.

Tabla 4.4. Tiempos para las señales para la lectura en el DLP-USB245M**.**

Tiempo	Descripción	Min	Max	Unidades
T1	Ancho de pulso activo para la señal RD.	50	-	ns
T2	Tiempo de espera entre habilitación y deshabilitación de RD.	50	-	ns
T3	Tiempo de espera para un dato valido después de deshabilitar RD.	-	30	ns
T4	Tiempo de retención de datos valido después de activar RD.	10	-	ns
T5	Tiempo de espera para activar RXF después de activar RD.	5	25	ns
T6	Tiempo de inactividad de RXF después de un ciclo de lectura.	80	-	ns

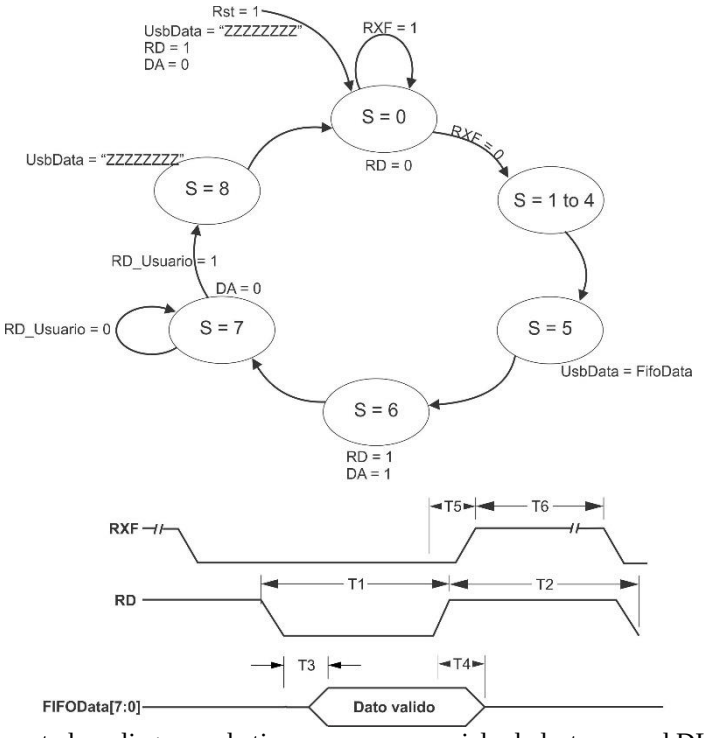

Figura 4.7. Máquina de estado y diagrama de tiempos y para un ciclo de lectura en el DLP-USB245M.

De acuerdo con el diagrama de tiempos de la Figura 4.7, cuando el DLP-USB245M ha recibido un dato proveniente de la interfaz de usuario, lo indica activando la señal RXF. En consecuencia, el **Controlador USB** activa la señal RD, 30 ns después el dato recibido está disponible en el bus de datos FIFOData[7:0]. Una vez que el Controlador USB ha leído el dato desactiva la señal RD, originando que el DLP-USB245M desactive la señal RXF. Ahora, el **Controlador USB** activa la señal DA para indicar al módulo HOG la existencia de un dato que debe ser procesado y que se encuentra disponible en el bus USBData[7:0]. En respuesta, el módulo HOG lee al dato e indica que la lectura ha concluido activando la señal ReadUsb, esta acción ocasiona que el **Controlador USB** desactive la señal DA, concluya la secuencia de estados de su FSM y regrese a condiciones iniciales.

Por otro lado, el proceso de escritura del DLP-USB245M también fue modelado mediante una FSM, considerando el diagrama de tiempos correspondiente, mostrados en la Figura 4.8, y cumpliendo los requisitos temporales especificados en la Tabla 4.5.

Tabla 4.5. Tiempos para las señales para la lectura en el DLP-USB245M.

Tiempo	Descripción	Min	Max	Unidades
T7	Ancho de pulso activo para la señal WR.	50	-	ns
T8	Tiempo de espera entre deshabilitación y habilitación de WR.	50	-	ns
T9	Tiempo de carga de dato antes de activar WR.	-	20	ns
T10	Retención de dato valido después de desactivar WR	10	-	ns
T11	Tiempo de espera para la activación TXE después de desactivar WR.	5	25	ns
T12	Tiempo de inactividad de TXE después de un ciclo de escritura.	80	-	ns

Cuando el módulo **UniCtrl_General** requiere enviar un dato a la interfaz de usuario, lo pone disponible a través del bus USBData[7:0] y activa la señal WriteUsb para solicitar al **Controlador USB** realice esta tarea. El controlador cumplirá la tarea siempre y cuando la señal TXE tenga un valor 0 lógico, lo que indica que el DLP-USB245M está disponible, es caso contrario espera hasta que esto ocurra.

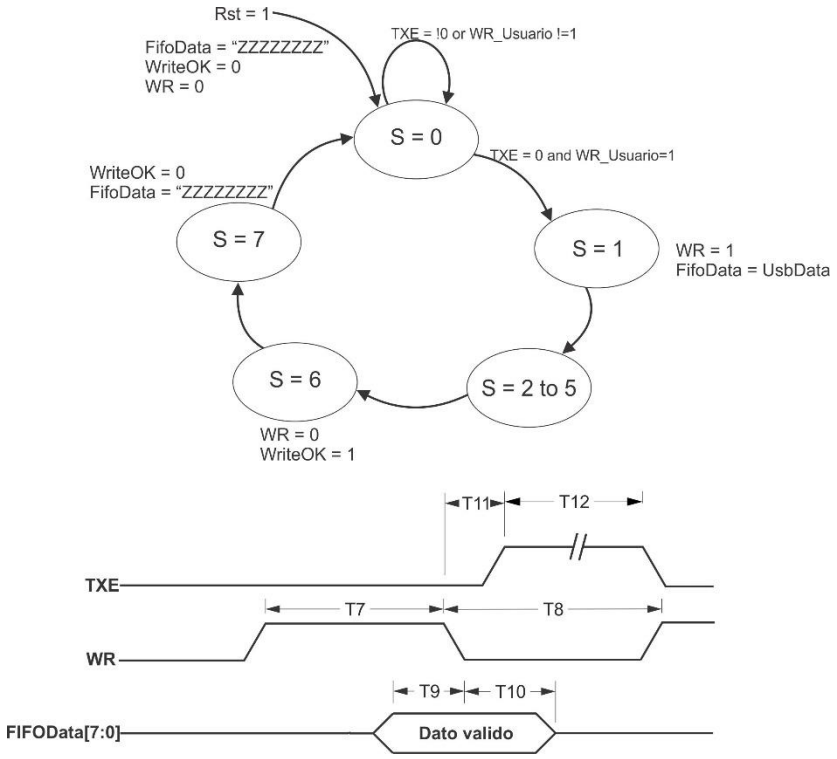

Figura 4.8. Máquina de estados y diagrama de tiempos para un ciclo de escritura en el DLP-USB245M.

Al inicio de la FSM del proceso de escritura del **Controlador USB**, se toma el dato del bus USBData[7:0] y se pone disponible para el DLP-USB245M mediante el bus FIFOData[7:0]. Ahora, se activa la señal WR y 50 ns después se desactiva, este evento indica al DLP-USB245M transmita el dato hacia la interfaz de usuario. El **Controlador USB** indica que la tarea fue concluida activando la señal WriteOK durante un ciclo de reloj. La señal WR debe permanecer desactivada por lo menos 50 ns.

4.4 Unidad de control general

El módulo de control general, **UniCtrl_General**, fue implementada mediante una FSM encargada de secuenciar la activación de los módulos de **HOG** y **DEN-network**. En la Figura 4.9 se muestra el símbolo de la **UniCtrl_General** y en la Tabla 4.6, se describen las señales que interfieren en su operación.

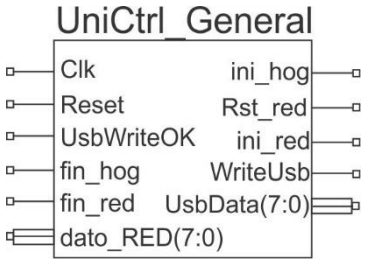

Figura 4.9. Símbolo de **UniCtrl_General** del sistema de reconocimiento en el FPGA.

Tabla 4.6. Descripción de las señales Entrada/Salida de **UniCtrl_General**.

Señal	Descripción
Clk	(Entrada) Señal de reloj del sistema a 100MHz.
Rst	(Entrada) Señal de reset.
fin_hog	(Entrada) Señal proveniente del módulo **HOG** el cual indica que el proceso ha terminado.
fin_red	(Entrada) Señal proveniente del módulo **DEN-network** el cual indica que el proceso ha terminado.
UsbWriteOk	(Entrada) Señal proveniente del módulo **Controlador USB** que indica que un dato ha sido escrito correctamente.
dato_RED(7:0)	(Entrada) Bus de datos proveniente del módulo **DEN-network** el cual contiene la clase a la que pertenece el objeto de entrada.
ini_hog	(Salida) Señal de control, indica al módulo **HOG** cuando iniciar su proceso.
ini_red	(Salida) Señal de control, indica al módulo **DEN-network** que iniciar su proceso.
Rst_red	(Salida) Señal de reset del módulo **DEN-network**.
WriteUsb	(Salida) Señal que va al módulo **Controlador USB** e indica que se quiere enviar un dato.
UsbData(7:0)	(Entrada/Salida) Bus de datos bidireccional proveniente del DLP-USB245M.

La FSM del módulo **UniCtrl_General** se presenta en la Figura 4.10. Cuando el usuario activa la señal de Reset, la FSM se posiciona en el primer estado donde se inicializan las señales de control. Una vez desactivado la señal de Reset, comienza el proceso de reconocimiento, por lo que se activa la señal ini_hog e indica el inicio de la extracción de características en el módulo **HOG**, durante este proceso se deshabilitan las señales de control del módulo **DEN-network** mediante la activación de la señal Rst_red. Al finalizar la extracción de características el módulo **HOG** activa la señal fin_hog, por lo que en respuesta **UniCtrl_General** desactiva las señales de control de **HOG** y habilita las señales de **DEN-network** para iniciar la clasificación del objeto de entrada con la activación de la señal ini_red. Concluido el proceso, indicado por el estado en alto de la señal fin_red, se tiene un dato de 8 bits correspondiente a la clase de clasificación, el dato es puesto en el bus de datos UsbData para que **Controlador USB** envié el dato a la PC, para ello se activa la señal de escritura UsbWrite. Finalizado él envió del dato, se deshabilita la señal de escritura, se pode en alta impedancia el bus de datos y se inicializan todas las señales de control para la entrada de una nueva imagen.

Cabe recalcar, que debido a que al módulo **HOG** se le asignaron las señales para la lectura de datos, el módulo **UniCtr_General** quedara indefinidamente en el estado 1 si no se ingresan los datos de un nuevo objeto. El proceso de lectura de datos de **HOG** se explicará en el apartado 4.5.4.

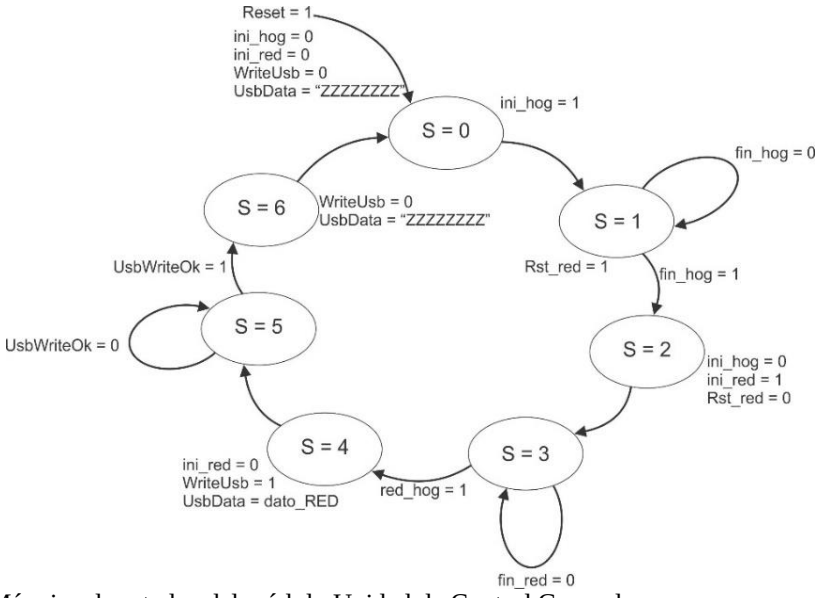

Figura 4.10. Máquina de estados del módulo Unidad de Control General.

4.5 HOG

El diseño, modelado e implementación del algoritmo de HOG sobre lógica reconfigurable se basó principalmente en los métodos y modificaciones descritas en el apartado 3.2, así como en los resultados de su modelado conceptual. La configuración de HOG elegida para implementar en el FPGA considera una imagen como la ventada de detección, formada por un bloque de características de 1×1 celda, donde cada celda se compone de 128×128 píxeles, dando como resultado, de acuerdo con el apartado 3.4, un vector de características de 9 elementos. Dada la configuración establecida, la fase de formación de bloques de características está incluida dentro de la fase del cálculo del histograma, de esta forma en la arquitectura solo se requieren 3 módulos para representar las 4 fases de HOG. En la Figura 4.11 se muestra el símbolo del módulo **HOG**, y en la Tabla 4.7 se describen las señales que interfieren con el módulo.

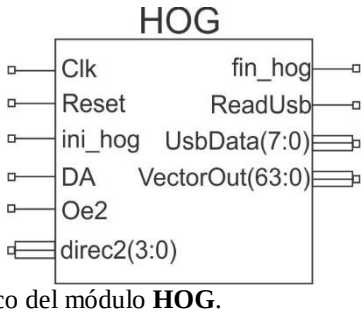

Figura 4.11. Símbolo de esquemático del módulo **HOG**.

Tabla 4.7. Descripción de las señales Entrada/Salida del módulo **HOG**.

Señal	Descripción
Clk	(Entrada) Señal de reloj del sistema a 100MHz.
Reset	(Entrada) Señal de reset.
ini_hog	(Entrada) Señal proveniente del módulo **HOG** el cual indica que el proceso debe comenzar.
DA	(Entrada) Señal proveniente del módulo **Controlador USB** el cual indica que existe un dato disponible en el bus de datos.
Oe2	(Entrada) Señal proveniente del módulo **DEN-network** el cual habilita un buffer de tercer estado de la memorial DualPort
direc2(3:0)	(Entrada) Bus de direcciones proveniente del módulo **DEN-network** y van a la memoria DualPort.
fin_hog	(Salida) Señal que va al módulo **UniCtrl_General** que indica el término del proceso.
ReadUsb	(Salida) Señal que va al módulo **Controlador USB** e indica que el dato disponible ha sido leído.
VectorOut(63:0)	(Salida) Bus de datos que va al módulo **DEN-network** el cual contiene información de la DualPort.
UsbData(7:0)	(Entrada/Salida) Bus de datos bidireccional proveniente del DLP-USB245M.

En la Figura 4.12 se muestra de manera interna, en forma de diagrama de bloques, los módulos que integran las fases de HOG. En el diagrama se puede visualizar que el módulo **HOG** se compone de una unidad de control local, **UniCtrl_HOG**, la cual además de administrar las fases del algoritmo, realiza la lectura de datos en el módulo **Controlador USB**. Los elementos restantes del esquema son los módulos operativos que realizan el cálculo de gradiente magnitud y dirección, la formación del histograma y la normalización de los vectores, denominados a partir de ahora y por simplicidad **Gradientes**, **Histograma** y **Normalización** respectivamente. Los dos últimos módulos comparten un módulo de memoria que almacena los elementos del vector de características. Para facilitar la compresión de la extracción de características, primeramente, se describirán los módulos operativos de **HOG** y finalmente se explicará la **UniCtrl_HOG**.

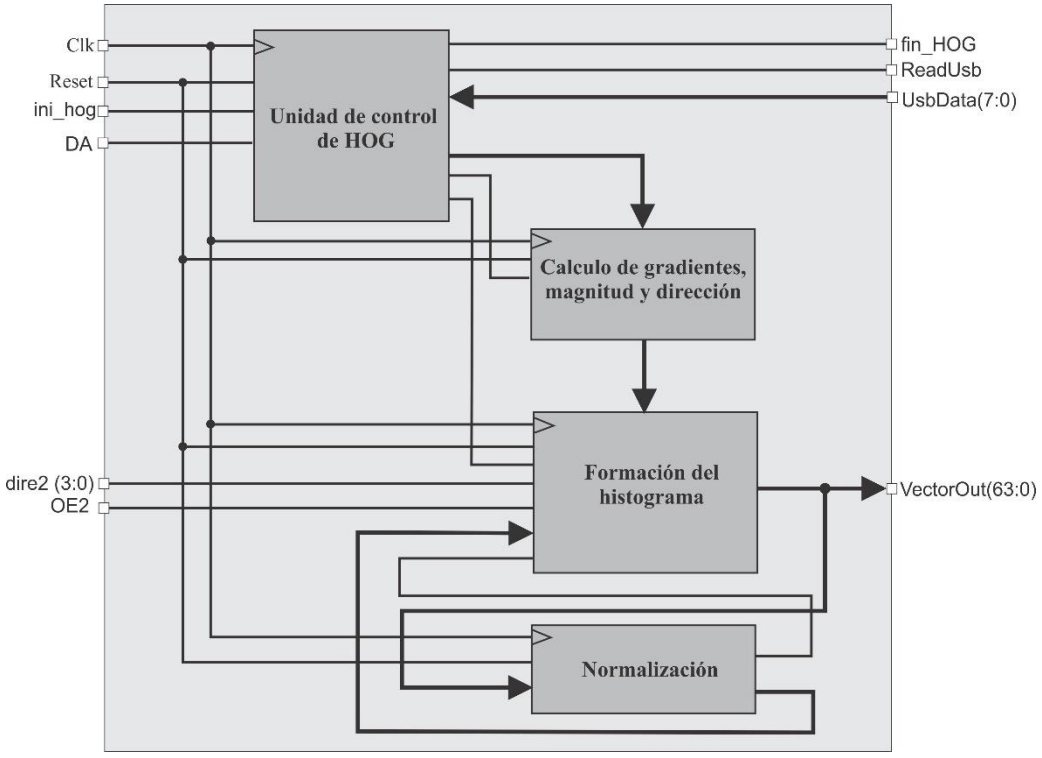

Figura 4.12. Diagrama de bloques del algoritmo de HOG en el FPGA.

Figura 4.13. Símbolo del módulo **Gradiente** que implementa la primera fase de HOG.

4.5.1 Obtención del gradiente, magnitud y dirección

El módulo **Gradiente** es el encargado de realizar los cálculos, dado un dato de entrada, de gradiente en X y Y, su magnitud y dirección. La Figura 4.13 muestra el símbolo de este módulo y en la Tabla 4.3 se describen las señales necesarias para realizar su tarea.

Tabla 4.8. Descripción de las señales Entrada/Salida del módulo **Gradiente**.

Señal	Descripción
Clk	(Entrada) Señal de reloj del sistema a 100MHz.
Reset	(Entrada) Señal de reset.
Clock_e	(Entrada) Señal proveniente la **UniCtrl_HOG** que va a los inestables de la consolación.
DatoIN(7:0)	(Entrada) Bus de datos proveniente de la **UniCtrl_HOG**.
Magnitud(16:0)	(Salida) Bus de datos que contiene la información de la magnitud del gradiente y va al módulo **Histograma**.
Direc(15:0)	(Salida) Bus de datos que contiene la información de la dirección del gradiente y va al módulo **Histograma**.

En la Figura 4.14 se muestra el diagrama de bloques del módulo **Gradiente**. Tras ingresar un dato valido al módulo de la **Convolución**, éste regresa 2 datos de 9 bits (1 bit de signo, 8 bits de parte entera) correspondientes a los gradientes en X y Y del *n*-ésimo píxel procesado.

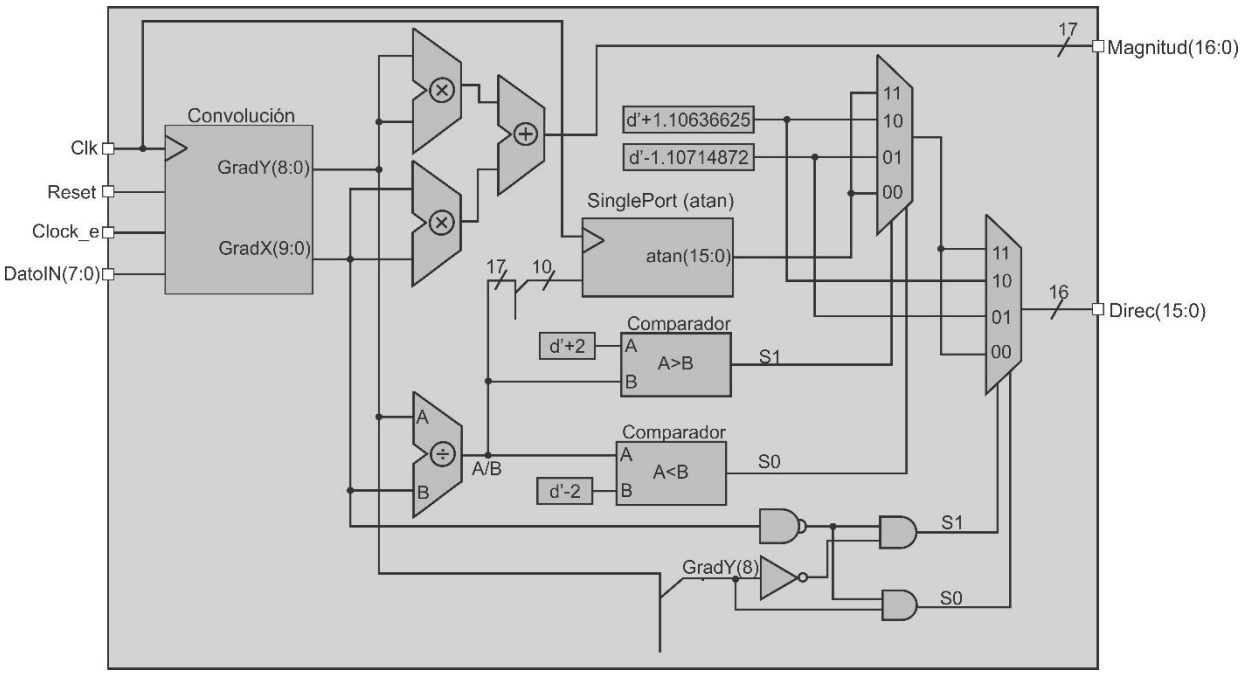

Figura 4.14. Diagrama de bloques del módulo **Gradiente**.

El ancho del bus de datos correspondiente a la magnitud se da bajo el siguiente análisis de datos, el valor máximo que puede tener cada gradiente es -255 o 255, por ende, el valor al cuadrado de un gradiente seria 65,025, contemplando que la operación requerida para la magnitud es una suma de cuadrados sin la raíz cuadrada, el valor máximo que se puede presentar es 130,050, el cual se puede expresar por una variable de 17 bits de parte entera. Como se hizo mención en el apartado 3.2.1, se realizaron las pruebas correspondientes para verificar que la omisión de la raíz cuadrada en la ecuación (3), no afectaba de manera significativa al desempeño final del sistema.

De manera similar a la magnitud, para determinar el ancho del bus de datos de la dirección, primeramente, se contemplan las siguientes consideraciones; el rango de valores de la división de gradientes puede encontrarse entre $[-255{,}255]$; y para representar la parte decimal de la división se determinó que 8 bits eran suficientes, por ello el resultado se expresa en un bus de 17 bits (1 bit de signo, 8 bits de parte entera y 8 bits de parte decimal). Como se mencionó en el apartado 3.2.1, el rango de valores de entrada óptimos en la función del arcotangente se encuentra en $(-2{,}2)$, de esta forma realizando una aproximación, solo se requiere de un bus de 10 bits, 1 bits de signo, 1 bit de parte entera y 8 bits de parte decimal, para representarla. Para determinar si los valores de la división exceden el rango establecido, la información pasa a 2 comparadores cuyo estado determinará si la dirección toma el valor del arcotangente, el del límite superior o el del límite inferior. Adicionalmente, si se presenta el caso donde la división de como resultado una indeterminación, cuando el gradiente en X es igual a 0, se toma el signo del gradiente en Y y mediante la lógica mostrada, se determina si la dirección toma el límite superior o inferior del arcotangente. En cualquier otro caso se toma el valor de la unidad que realiza el arcotangente, el elemento que realiza esta operación es una memoria SinglePort precargada con los valores que puede tener el arcotangente, dada la entrada de 10 bits la memoria se compone de 1024 localidades y cada localidad se compone de 16 bits (1 bit de signo, 1 bit de parte entera y 14 bits de parte decimal). Finalmente, cabe recalcar que, si se aumenta el número de bits de entrada en la memoria, esta incrementa su tamaño en potencia de dos, es decir, si se agrega un bit más, la memoria será de 2048 localidades, lo que repercute directamente en la cantidad de recursos utilizados.

4.5.1.1 Convolución

El módulo **Convolución** es el encargado de realizar las operaciones de gradiente en X y Y. Para que este módulo cumpla con su tarea es necesario que una ventada sea desplazada a lo largo y ancho de la imagen y que en cada desplazamiento se realice la operación que dicte el algoritmo en cuestión y que involucra a los píxeles de la imagen que son contenidos por la ventana. Una arquitectura de ventana deslizante basada en buffers de línea es típicamente utilizada para realizar esta tarea. Una arquitectura de este tipo está formada por un arreglo de $t \times t$ registros, donde t define al ancho y largo de la ventana, y existen $t - 1$ buffers de línea, donde cada buffer de línea está formado por $ancho\_i - t$ biestables, donde $ancho\_i$ es el ancho de la imagen. Para esta aplicación se realizó el esquema de la Figura 4.15, el cual está compuesto por un conjunto de 9 registros de 8 bits, y 2 buffers de datos que actúan como registros de corrimiento, cuyo ancho es de 1000 bits correspondientes a 125 datos de 8 bits, esto es debido que el conjunto de un registro de corrimiento y 3 registros forman un ancho de 128 datos, lo que corresponde al ancho de la imagen de entrada, de esta forma la configuración del sistema permite llevar acabo las operaciones previamente mencionadas. Una vez que la señal Clock_e es puesta en alto, se almacena el dato contenido en el bus de datos DatoIN(7:0) en el primer biestable y se recorren los demás datos en el biestable y registros consecutivos, de esta forma mediante restadores se obtiene los gradientes deseados.

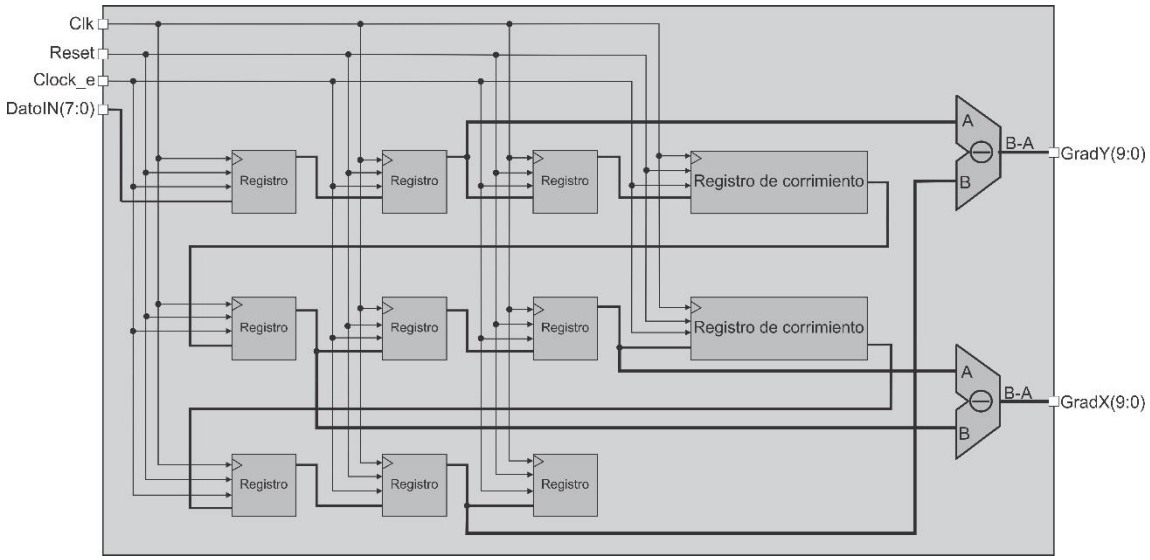

Figura 4.15. Diagrama de bloques del módulo **Convolución**.

4.5.2 Orientación del gradiente

El módulo **Histograma** es el encargado de distribuir y almacenar la información correspondiente a la orientación del gradiente de un píxel en contenedores formando el histograma del algoritmo de HOG. Debido a que el resultado del módulo **Histograma** se almacena en una memoria que también requiere ser accesada por los módulos de **Normalización** y **DEN-network**, se implementaron buffers de tercer estado que permitan a los buses de direcciones direc1(3:0) y direc2(3:0) acceder a la información que necesiten. Las señales que activan los buffers de tercer estado son Oe0, Oe1, Oe2, correspondientes a la **UniCtrl_HOG**, al módulo **Normalización** y al módulo **DEN-network** respectivamente. La Figura 4.16 muestra el símbolo del Histograma y en la Tabla 4.9 se describen las señales de control y respuesta del módulo.

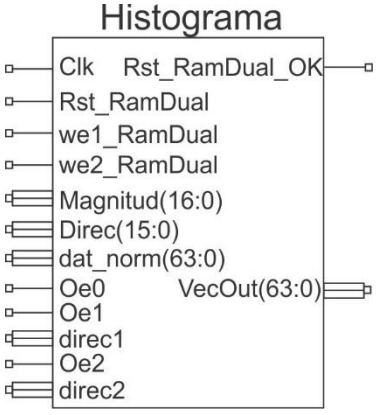

Figura 4.16. Símbolo del módulo **Histograma**.

Tabla 4.9. Descripción de las señales Entrada/Salida del módulo **Histograma.**

Señal	Descripción
Clk	(Entrada) Señal de reloj del sistema a 100MHz.
Rst_RamDual	(Entrada) Señal de reset de la memoria DualPort.
we1_RamDual	(Entrada) Señal de control 1 para escribir un dato en la memoria Dualport.
we2_RamDual	(Entrada) Señal de control 2 para escribir un dato en la memoria Dualport.
Magnitud(16:0)	(Entrada) Bus de datos que contiene la información de la magnitud del gradiente.
Direc(15:0)	(Entrada) Bus de datos que contiene la información de la dirección del gradiente.
dat_norm(63:0)	(Entrada) Bus de datos proveniente del módulo **Normalización** el cual va a la memoria DualPort.
Oe0	(Entrada) Señal de control proveniente de la **UniCtrl_HOG** para dar paso a la información del primer buffer de tercer estado.
Oe1	(Entrada) Señal de control proveniente de la **UniCtrl_HOG** para dar paso a la información del segundo buffer de tercer estado.
Oe2	(Entrada) Señal de control proveniente de la **UniCtrl_HOG** para dar paso a la información del tercer buffer de tercer estado.
direc1(3:0)	(Entrada) Bus de datos proveniente del módulo **Normalización** para poder acezar a los datos de la memoria DualPort.
direc2(3:0)	(Entrada) Bus de datos proveniente del módulo **DEN-network** para poder acezar a los datos de la memoria DualPort.
Rst_RamDual_OK	(Salida) Señal que indica que todas las localidades de la memoria DualPort están en 0.
VecOut(63:0)	(Salida) Bus de datos de la memoria DualPort que va a los módulos externos que hacen uso de los datos de la memoria.

La lógica diseñada para almacenar la información en los contenedores del histograma se muestra en la Figura 4.17. Los bus de datos Magnitud(15:0) y Direc(15:0) entran al módulo **Proporción bins** que distribuye la proporción de pesos ponderados de la magnitud de acuerdo al ángulo de entrada entre los contenedores superior e inferior mc_o^{lk} y mc_{o+1}^{lk}. Además, el módulo **Proporción bins** genera, mediante el bus de direcciones bin_selec(2:0), la dirección del contenedor inferior, y agregando un sumador se tiene un bus de direcciones adicional para el contenedor superior. Finalmente, la información del peso de los contenedores se acumula y almacena en las direcciones respectivas de la memoria DualPort.

El tamaño de la memoria DualPort es de 9 localidades de 64 bits debido a las operaciones realizadas por los módulos **Histograma** y **Normalización**. El valor máximo que se puede presentar en un contenedor del histograma se da cuando todos los gradientes X y Y dan como resultado 255 y por ende la magnitud 130,050, tomando en consideración los 15,876 píxeles procesados, correspondientes a una imagen de 128×128 píxeles pues no se procesan los pixeles de la periferia, el valor máximo que se puede almacenar es 2.064×10^9 el cual puede representarse con una variable de 32 bits de parte entera. En el caso de la normalización, dado a que se requiere precisión en la división, se optó por representar el resultado médiate una variable de 64 bits (1 bit de parte entera y 63 de parte decimal), por ello, con el objetivo de minimizar recursos, el ancho de la memoria es de 64 bits para que se pueda ser utilizada por ambas fases del sistema.

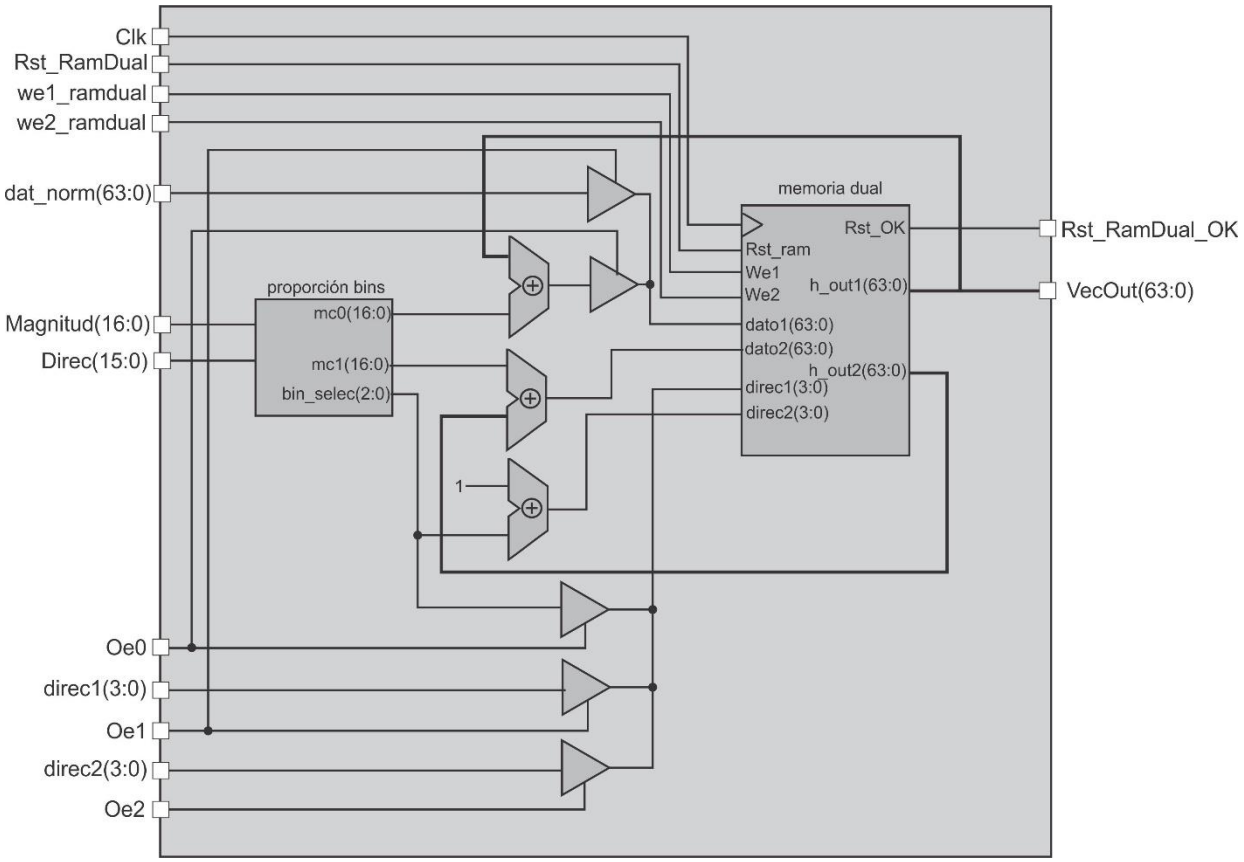

Figura 4.17. Diagrama de bloques del módulo **Histograma**.

4.5.2.1 Distribución de pesos ponderados

Para comprender de manera detallada como se realiza la distribución de pesos ponderados en lógica reconfigurable, se debe de analizar la ecuación (30) resultado de la ecuación (7) y su modificación del apartado 3.2.2. Donde x es el factor de conversión de grados a radianes $x = \left(\frac{\pi\ rad}{180°}\right)$.

$$mc_o^{lk} = \begin{cases} mc_o^{lk} + (1 - \dfrac{c\_th_{ij}^{lk} - (d_{o+1} - 20x)}{0.5}) \cdot c\_m_{ij}^{lk}, & \text{si } bin_o < c\_th_{ij}^{lk} \\ mc_o^{lk} + c\_m_{ij}^{lk}, & \text{si } bin_o = c\_th_{ij}^{lk} \end{cases}$$

$$mc_{o+1}^{lk} = \begin{cases} mc_{o+1}^{lk} + (\dfrac{c\_th_{ij}^{lk} - (d_{o+1} - 20x)}{0.5}) \cdot c\_m_{ij}^{lk}, & \text{si } c\_th_{ij}^{lk} < bin_{o+1} \\ mc_{o+1}^{lk} + c\_m_{ij}^{lk}, & \text{si } bin_{o+1} = c\_th_{ij}^{lk} \end{cases} \quad (30)$$

Se puede observar que la operación $(d_{o+1} - 20x)$ hace referencia al valor de ubicación del contenedor superior menos la separación entre contenedores, sin embargo, al realizar dicha operación da como resultado el valor de ubicación del contenedor inferior d_0, por ende, la operación es reemplazada directamente con dicho valor.

Otra modificación importante y la cual repercute principalmente en la reducción de recursos en el FPGA, es en el cálculo del valor ponderado del contenedor inferior, pues al ser complemento del valor ponderado del contenedor superior, basta con realizar una operación de resta entre la

magnitud y el valor obtenido del contenedor superior. De esta forma, reescribiendo la ecuación (30) se tiene que,

$$mc_o^{lk} = \begin{cases} mc_o^{lk} + c\_m_{ij}^{lk} - (\frac{c\_th_{ij}^{lk} - d_o}{0.5} \times c_{m_{ij}}^{lk}), & \text{si } bin_o < c\_th_{ij}^{lk} \\ mc_o^{lk} + c\_m_{ij}^{lk}, & \text{si } bin_o = c\_th_{ij}^{lk} \end{cases}$$

$$mc_{o+1}^{lk} = \begin{cases} mc_{o+1}^{lk} + (\frac{c\_th_{ij}^{lk} - d_o}{0.5}) \cdot c\_m_{ij}^{lk}, & \text{si } c\_th_{ij}^{lk} < bin_{o+1} \\ mc_{o+1}^{lk} + c\_m_{ij}^{lk}, & \text{si } bin_{o+1} = c\_th_{ij}^{lk} \end{cases} \quad (31)$$

La lógica utilizada para realizar las operaciones de la ecuación (31) se puede observar en la Figura 4.18. El bus de datos de la dirección del gradiente ingresa a una serie de comparadores (a, b, c, d, e, f, g) donde se evalúa si el valor es mayor o igual al valor de ubicación de los contenedores de correspondientes a $\{-60°, -40°, -20°, 0°, 20°, 40°, 60°\}$, los cuales son valores contantes almacenados en registros; la representación binaria del valor del contenedor (1 bit de signo, 1 bit de parte entera y 14 bits de parte decimal) se muestran en la Tabla 4.10. De esta forma mediante un codificador, cuya tabla de verdad se expresa en la Tabla 4.11, y un multiplexor se obtiene el valor de ubicación del contenedor inferior correspondiente, por lo que se procede a realizar la resta entre el ángulo y el contenedor, consecutivamente se realiza la división, sin embargo, dado a que el dividendo es un número en potencia de dos ($0.5 = 2^{-1}$), se puede expresar en lógica reconfigurable como un corrimiento a la izquierda.

Al realizar las operaciones mencionadas se obtiene un bus de datos de 17 bits (1 bit de signo, 2 bis de parte entera, y 14 bits de parte decimal), como es evidente el resultado siempre dará un valor positivo, por lo que se puede descartar el bit de signo, como el valor de proporción se puede expresar entre 0 y 1, de igual manera se puede descartar el bit más significativo de la parte entera, quedando un bus de datos de 15 bits (1 bit de parte entera y 14 de parte decimal). Multiplicando el bus de datos resultante por el valor del bus de datos de la magnitud, se obtiene un resultado de 32 bits (18 bits de parte entera y 14 bits de parte decimal) que corresponde al valor ponderado del contenedor superior mc_{o+1}^{lk}. Considerando que el valor resultante es igual o menor al valor de la magnitud y con el propósito de optimizar los recursos en las operaciones posteriores, se toman los 17 bits menos significativos de la parte entera, de esta forma el resultado junto con el valor de la magnitud pasan a un restador donde se obtiene el peso ponderado del contenedor inferior mc_o^{lk}.

Tabla 4.10. Valor de los contenedores (bin) del histograma.

Contenedor	Valor del contenedor	Representación binaria
bin0	-80° / $-\frac{4\pi}{9} rad$	10.10011010100110
bin1	-60° / $-\frac{\pi}{3} rad$	10.11110011111100
bin2	-40° / $-\frac{2\pi}{9} rad$	11.01001101010011
bin3	-20° / $-\frac{\pi}{9} rad$	11.10100110101010
bin4	0° / $0\ rad$	00.00000000000000
bin5	20° / $\frac{\pi}{9} rad$	00.01011001010110
bin6	40° / $\frac{2\pi}{9} rad$	00.10110010101101
bin7	60° / $\frac{\pi}{3} rad$	01.00001100000100

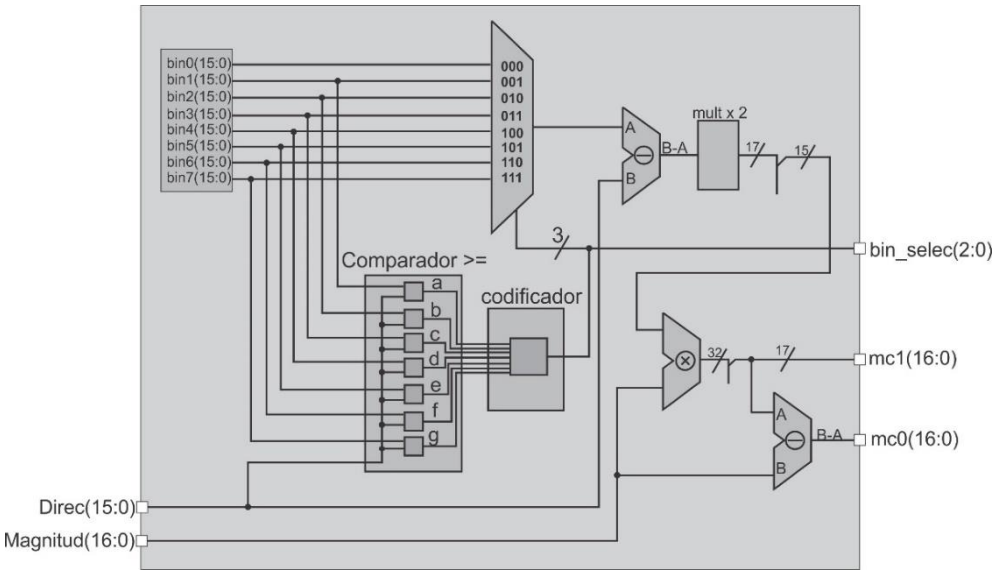

Figura 4.18. Lógica para la distribución de pesos ponderados en contenedores del histograma.

Tabla 4.11. Tabla de verdad del codificador de acuerdo con los valores de entrada de los comparadores.

a	b	c	d	e	f	g	O_2	O_1	O_0
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	1
1	1	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	1	1
1	1	1	1	0	0	0	1	0	0
1	1	1	1	1	0	0	1	0	1
1	1	1	1	1	1	0	1	1	0
1	1	1	1	1	1	1	1	1	1

4.5.3 Normalización

El módulo **Normalización** es el encargado de la última fase del algoritmo de HOG, donde toma cada elemento almacenado de la memoria DualPort y les aplica un proceso de normalización definida por la ecuación (9) y su modificación del apartado 3.2.4, por ello se le asignan las señales de control para tener el acceso a los elementos de la memoria. El símbolo del módulo **Normalización** se muestra en la Figura 4.19 y la descripción de sus señales se muestran en la Tabla 4.12.

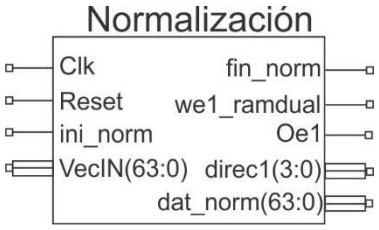

Figura 4.19. Símbolo del módulo **Normalización**.

Tabla 4.12. Descripción de las señales Entrada/Salida del módulo **Normalización**.

Señal	Descripción
Clk	(Entrada) Señal de reloj del sistema a 100MHz.
Reset	(Entrada) Señal de reset de la memoria DualPort.
ini_norm	(Entrada) Señal de control proveniente de **UniCtrl_HOG**.
VecIN(63:0)	(Entrada) Bus de datos proveniente de la memoria DualPort.
fin_norm	(Salida) Señal de respuesta que indica el término de la normalización.
we1_RamDual	(Salida) Señal de control 1 para escribir un dato en la memoria Dualport.
Oe1	(Salida) Señal de control que activa el segundo buffer de tercer estado y da paso a la información.
direc1(3:0)	(Salida) Bus de datos proveniente del módulo **Normalización** para poder acezar a los datos de la memoria DualPort.
dat_norm(63:0)	(Salida) Bus de datos el cual va a la memoria DualPort que contiene un valor normalizado.

La estructura de hardware diseñada para implementar el módulo **Normalización** se muestra en la Figura 4.20, donde se toman los 32 bits menos significativos en el bus de datos correspondiente al vector de características. Debido a que el proceso de normalización es iterativo se realiza mediante una FSM, la cual se puede visualizar en la Figura 4.23 y su lógica asociada en la Figura 4.24. Una vez que el módulo **UniCtrl_HOG** le indica al módulo de normalización que debe iniciar su proceso mediante la activación de la señal ini_norm, se pone en alto la señal Oe1, lo que activa el buffer de tercer estado que permite dar paso al bus de direcciones direc1(3:0), cuyo control depende del contador de 4 bits de la FSM.

La primera operación que debe ser realizada es la suma de los cuadrados de todos los elementos del histograma, por lo que, al leer, multiplicar y acumular el cuadrado de un elemento, dado al tiempo de propagación existente entre los componentes, se esperan 40ns para activar el Clock enable (Ce) del registro encargado de almacenar la información del dividendo de la ecuación (9). Culminado la suma de cuadrados, se procede a normalizar los elementos del histograma donde cada elemento es introducido a la unidad de división, y de manera similar a la suma de cuadrados, se esperan 60ns dado el tiempo de propagación, de esta forma se escribe el resultado en la memoria DualPort activando la señal we1_RamDual, el resultado de cada operación se almacena en la localidad del elemento correspondiente. Terminado el proceso completo, el módulo **Normalización** activa la señal fin_norm, se inicializa el contador y se desactiva la señal de control Oe1.

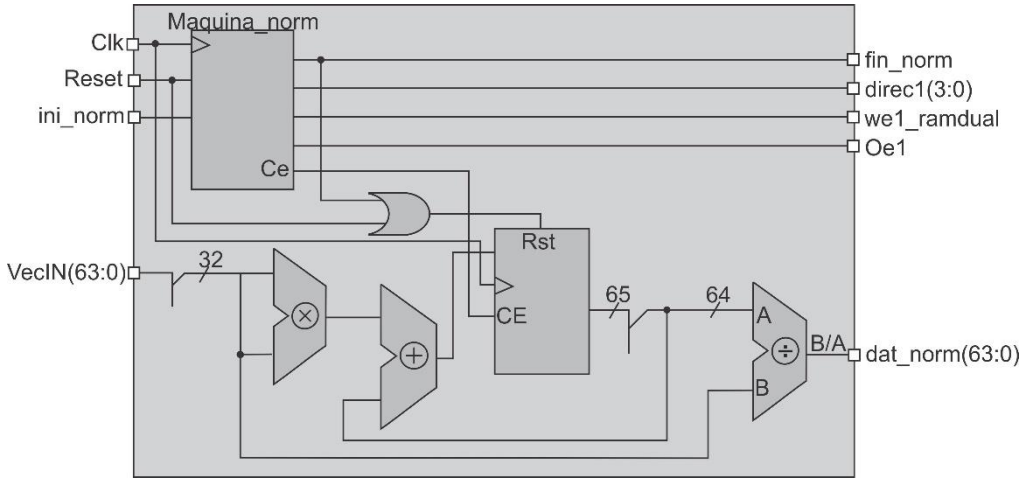

Figura 4.20. Lógica combinacional implementada para el proceso de normalización.

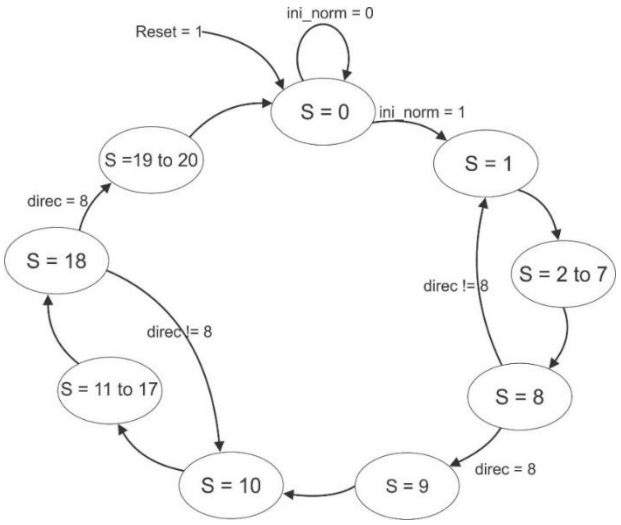

Figura 4.21. Máquina de estados del proceso de normalización.

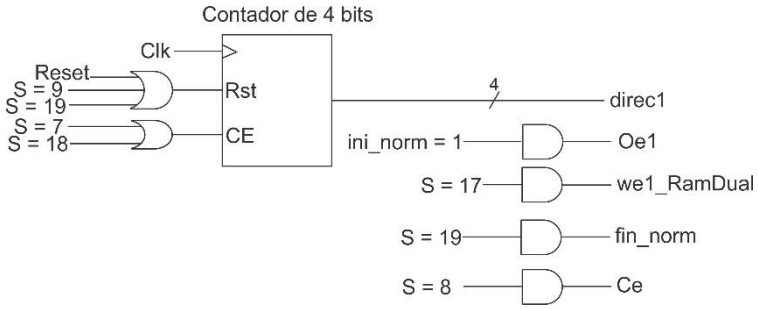

Figura 4.22. Lógica asociada a la máquina de estados de la normalización.

4.5.4 Unidad de control de HOG

Una vez comprendida la estructura de los módulos que implementan las fases del algoritmo de HOG, el módulo **UniCtrl_HOG** es la unidad de control local del módulo de **HOG** encargado de administrar, mediante una FSM, los procesos entre los componentes para la formación del vector de caracterizas. Además, el módulo se encarga de leer los datos provenientes de la PC y pasarlos al módulo **Convolución**. El símbolo de **UniCtrl_HOG** se muestra en la Figura 4.23 y las señales con las que interactúa se describen en la Tabla 4.13

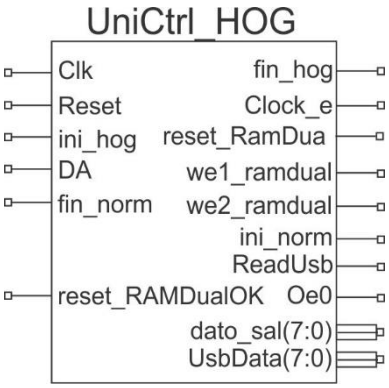

Figura 4.23. Símbolo del módulo **UniCtrl_HOG**.

Tabla 4.13. Descripción de las señales Entrada/Salida del módulo **UniCtrl_HOG**.

Señal	Descripción
Clk	(Entrada) Señal de reloj del sistema a 100MHz.
Reset	(Entrada) Señal de reset de los componentes del módulo de HOG.
ini_hog	(Entrada) Señal de control proveniente de la unidad de control general.
DA	(Entrada) Señal indicadora de datos disponible en el módulo **Controlador USB**.
fin_norm	(Entrada) Señal de respuesta que indica el término de la normalización.
reset_RAMDualOK	(Entrada) Señal de respuesta que indica el que todas las localidades de la memoria DualPort se han puesto en 0s.
fin_hog	(Salida) Señal indicadora que el algoritmo de HOG ha terminado.
Clock_e	(Salida) Señal que activa el Clock enable de los biestables de la convolución.
Reset_RAMDual	(Salida) Señal que pone en 0s todas las localidades de la memoria DualPort.
Oe0	(Salida) Señal de control que activa el primer buffer de tercer estado y da paso a la información.
we1_RamDual	(Salida) Señal de control 1 para escribir un dato en la memoria Dualport.
we2_RamDual	(Salida) Señal de control 2 para escribir un dato en la memoria Dualport.
ini_norm	(Salida) Señal de control que indica al módulo **Normalización** cuando inicial su proceso.
ReadUsb	(Salida) Señal de control que indica que el dato del **Controlador USB** ha sido leído.
dato_sal(7:0)	(Salida) Bus de datos que va al módulo **Gradiente**.
UsbData(7:0)	(Entrada/Salida) Bus de datos bidireccional proveniente del DLP-USB245M.

En la Figura 4.24 se muestra la FSM que modela el comportamiento del módulo **UniCtrl_HOG** y en la Figura 4.25 se muestra su lógica asociada. Cuando el proceso da inicio, mediante el estado en alto de la señal ini_hog puesto por la **UniCtrl_General**, la señal de control band_reset_mem actúa como una bandera que indica, con un nivel en bajo, si la memoria DualPort ha sido reseteada, puesto que se requiere que este vacía para comenzar la formación del vector de características, si no es el caso se activa la señal reset_RAMDual e inicia el proceso de reset en la memoria. Culminado el proceso se espera un dato proveniente de la PC, dicho dato debe corresponder a un píxel de una imagen. Capturando un dato se activa la señal clock_e, la cual permite al módulo **Convolución** agregar un nuevo dato a su esquema.

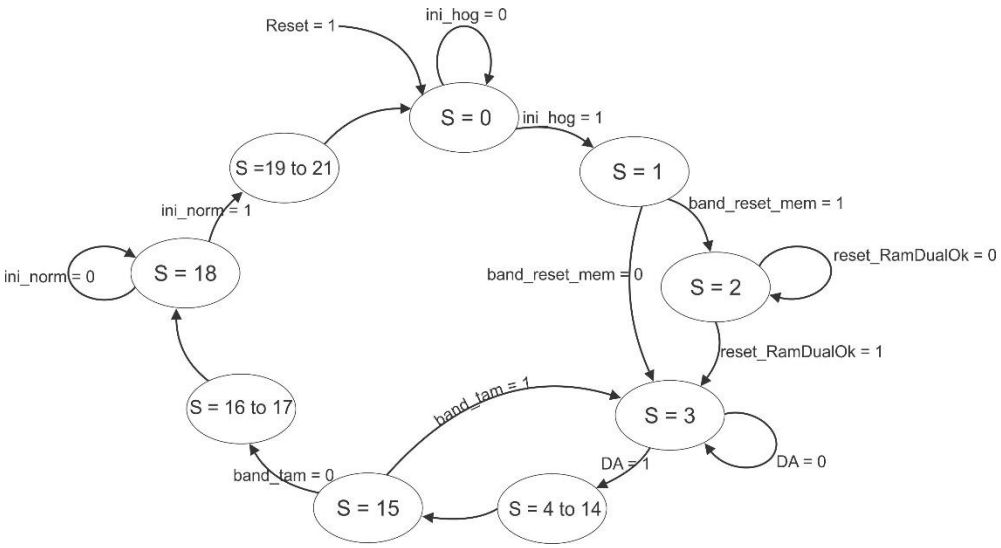

Figura 4.24. FSM del módulo **UniCtrl_HOG**.

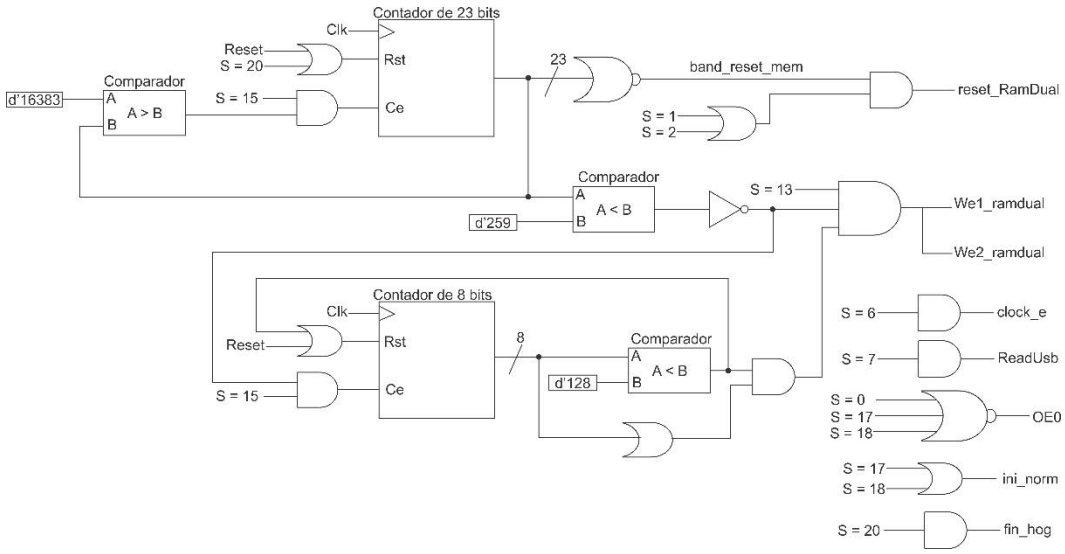

Figura 4.25. Lógica asociada a la FSM del módulo **UniCtrl_HOG**.

Tras esperar 8 ciclos de reloj, debido al tiempo de propagación resultante al realizar las operaciones de cálculo de gradientes, magnitud, dirección y distribución de pesos ponderados, se procede a acumular y guardar los valores mc_o^{lk} y mc_{o+1}^{lk} en la memoria DualPort, por lo que se activan las señales de escritura we1_RamDual y we2_RamDual y la señal Oe0 que activa el primer buffer de tercer estado. Cabe recalcar que el proceso de escritura en la memoria comienza después de la entrada del dato 259, debido a que, para que el esquema de convolución funcione se requiere que los 9 registros de 8 bits y los 2 registros de corrimientos contengan datos válidos, lo que ocurre cuando se ingresa el dato número 259, de esta forma un contador y comparador desactivan la opción de escritura mientras la cantidad de datos procesados sea inferior a la establecida, de igual manera el contador lleva el control de la cantidad de datos que se han ingresado, pues dada una imagen de 128×128 píxeles se deben de esperar 16,384 datos.

Adicionalmente, se debe en tener en consideración que al ingresar datos de manera serial se debe de llevar un control para determinar el ancho de la imagen, puesto que los píxeles que se encuentran en la periferia de esta no deben ser procesados pues sus resultados no representan datos válidos. Para realizar esta acción se utiliza un segundo contador que deshabilita la opción de escritura cuando se presentan el caso mencionado. Consecutivamente, una vez capturado y procesado todos los píxeles de la imagen recibida, se procede a activar la señal ini_norm para dar inicio al proceso de normalización, una vez culminado se reinician los contadores y se indica al módulo **UniCtrl_General**, a través de la activación de la señal fin_hog, que la extracción de características de la imagen ha culminado.

4.6 DEN-*network*

Debido a que el proceso de formación de la DEN-*network* es una tarea muy demandante, así lo corroboran los algoritmos y técnicas vistas en el apartado 3.3, y a que una vez definida la estructura de la res ésta no cambiara durante la fase de operación, se optó por realizar el entrenamiento en la PC con el programa Matlab y el proceso de evaluación con una estructura fija en el FPGA. De manera similar a la metodología empleada en el diseño del módulo de **HOG**, primeramente, se presentará la estructura, módulos y componentes hardware que conforman a la red y finalmente se describiría el módulo encargado de controlar las operaciones de los componentes utilizados.

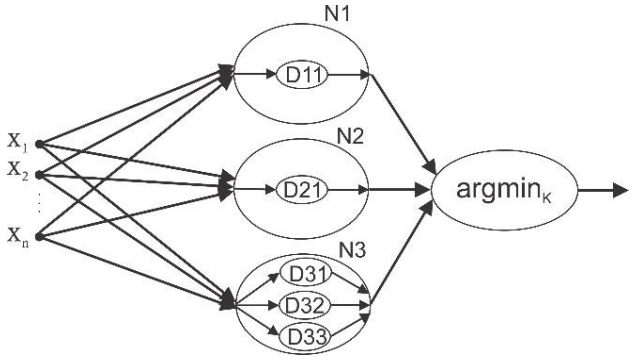

Figura 4.26. Esquema de 3 neuronas de la DEN-*network*.

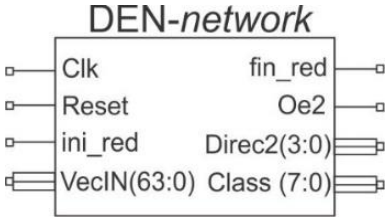

Figura 4.27. Símbolo del módulo **DEN-network.**

Para comprender de mejor manera como se realizó el diseño e implementación en hardware de la red neuronal, se explicará el modelado de la estructura de la Figura 4.26, la cual se compone de 3 neuronas correspondientes a 3 clases, dos de ellas contienen 1 dendrita y una contiene 3. El módulo **DEN-network** es el encargado de albergar dicha estructura, cuyo símbolo se muestra en la Figura 4.27 y la descripción de sus señales en la Tabla 4.14.

Tabla 4.14. Descripción de las señales Entrada/Salida del módulo que implementa la DEN-*network*.

Señal	Descripción
Clk	(Entrada) Señal de reloj del sistema a 100MHz.
Reset	(Entrada) Señal de reset.
ini_red	(Entrada) Señal de control proveniente de la **UniCtrl_General**, indica el inicio de la clasificación.
VecIN(63:0)	(Entrada) Bus de datos proveniente de la memoria DualPort.
fin_red	(Salida) Señal indicadora que el proceso de clasificación del módulo **DEN-network** ha terminado.
Oe2	(Salida) Señal de control que activa el tercer buffer de tercer estado y da paso a la información.
direc2(3:0)	(Salida) Bus de direcciones que va la memoria DualPort y permite leerla.
Class(7:0)	(Salida) Bus de datos el cual contiene la información de la clase perteneciente del objeto de entrada.

4.6.1 Estructura de la red neuronal

Al realizar el modelado de la estructura de la red propuesta, es necesario contemplar la cantidad de recursos disponibles en el FPGA después de implementar el módulo de la extracción de características, esto es debido a que el número de multiplicadores embebidos dicta la cantidad de estructuras que pueden calcular, de manera simultánea, la distancia de Mahalanobis. Para comprender como se realizó la estructura real en el FPGA se considerará a manera de ejemplo, una arquitectura que solo se tienen los recursos suficientes para formar 3 estructuras neuronales (neuronas), ver Figura 3.28, lo que implica que se deben de reutilizar y esto conlleva a que el proceso del cálculo de la distancia de Mahalanobis sea iterativo, seleccionando en cada iteración un conjunto diferente de dendritas. Estas estructuras neuronales son las encargadas de calcular la distancia de Mahalanobis y son identificadas como **Estructura_DEN*i*** en la Figura 4.29.

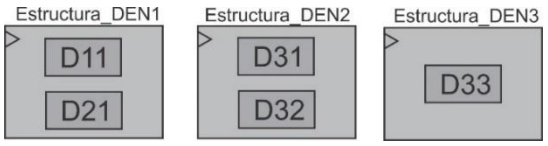

Figura 4.28. Distribución de las dendritas en las estructuras de la red neuronal.

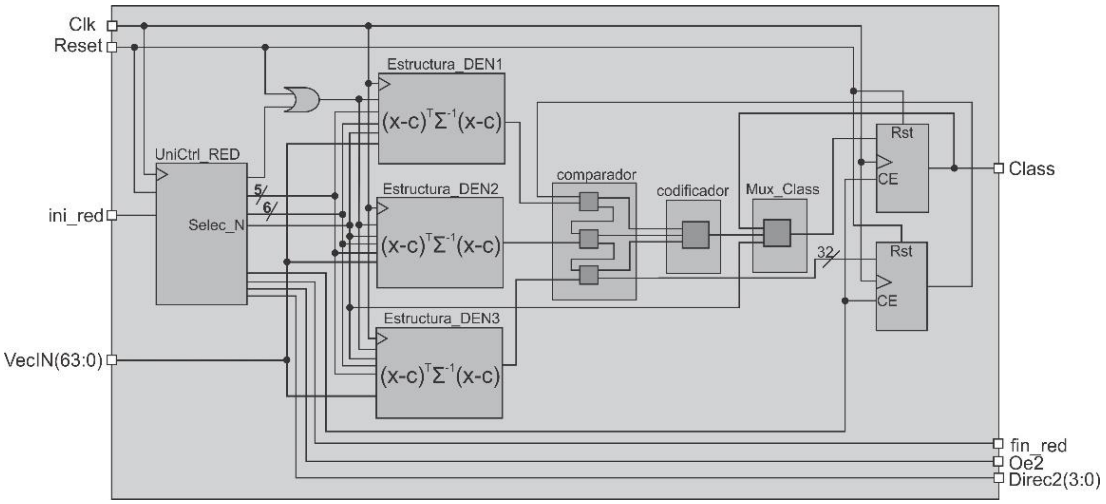

Figura 4.29. Estructura interna del módulo **DEN-network.**

Con el fin de minimizar la cantidad de iteraciones requeridas, se organiza la información de las dendritas de acuerdo con lo expresado en la Figura 4.28. De esta forma, de manera interna el módulo **DEN-network** implementa la lógica que se muestra en la Figura 4.29, donde además de integrar las estructuras que realizan el cálculo de la distancia deseada, se compone de una unidad de control local (**UniCtrl_RED**) y una serie de componentes para obtener el argumento mínimo del sistema.

4.6.1.1 Acondicionamiento del sistema

Con el objetivo de minimizar recursos en las operaciones que se requieren para el cálculo de la distancia de Mahalanobis, se optó por tomar una sección de 31 bits válidos del bus de datos VecIN(63:0) que corresponde al rango de valores validos de los vectores de características, $[3.1346148977 \times 10^{-10}, 2.24248804355 \times 10^{-7}]$ para la base de datos COIL-20 y $[4.02621150007 \times 10^{-10}, 1.15404817208 \times 10^{-5}]$ para la base de datos ALOI, esta modificación se expresó más ampliamente en el apartado 3.2.4. De esta forma, para la base de datos COIL-20 se tomó la sección (42:12) correspondiente a un desplazamiento de 21 bits, para la base datos ALOI se toma la sección (46:16), que corresponde a un desplazamiento de 18 bits, dando un bus resultante de 31 bits que está formado por 1 bit de parte entera y 30 bits de parte fraccionaria. Finalmente, para facilitar las operaciones necesarias de resta y multiplicación, al bus resultante se le agrega un bit de signo con un valor de 0, puesto que los elementos del vector de características siempre son positivos, dando así un bus de 32 bits (1 bit de signo, 1 bit de parte entera y 30bits de parte fraccionaria). Cabe mencionar que la acción de tomar una sección del bus VecIN(63:0) implica una pérdida de información, puesto que, se recortan los bits menos significativos de los elementos del vector de características, sin embargo, esta modificación no repercute de manera significativa en el resultado final de clasificación.

4.6.2 Distancia de Mahalanobis

Para comprender como se lleva a cabo el cálculo de la distancia de Mahalanobis dentro de una **Estructura_DEN*i***, primeramente, se presentará como está organizada la información de los pesos sinápticos y consecutivamente se mostrará el proceso del cálculo.

4.6.2.1 Organización y lectura de la información de una dendrita en un módulo de memoria RAM

Al realizar el entrenamiento de la red neuronal en la PC, se crea un archivo de texto a través de Matlab que contiene la información asociada a cada dendrita, el centroide y la matriz de covarianza inversa, de esta forma mediante VHDL se almacena esta información en una unidad de memoria RAM. Para comprender como está distribuida la información en la memoria, primeramente, se realizará un análisis basándose en una representación genérica de una matriz de covarianza inversa, cuyo tamaño es de 9×9 elementos dada los vectores de características de 9 elementos resultados del algoritmo de HOG, ver Tabla 4.15.

Tabla 4.15. Matriz de covariancia inversa de las dendritas (los números de la matriz representan el número de elemento, así $\boldsymbol{valor\ elemento\ 2 = valor\ elemnto\ 10, 3 = 19, 4 = 28, \ldots, 72 = 80}$).

Índices XY	j_1	j_2	j_3	j_4	j_5	j_6	j_7	j_8	j_9
i_1	**1**	2	3	4	5	6	7	8	9
i_2	**10**	**11**	12	13	14	15	16	17	18
i_3	**19**	**20**	**21**	22	23	24	25	26	27
i_4	**28**	**29**	**30**	**31**	32	33	34	35	36
i_5	**37**	**38**	**39**	**40**	**41**	42	43	44	45
i_6	**46**	**47**	**48**	**49**	**50**	**51**	52	53	54
i_7	**55**	**56**	**57**	**58**	**59**	**60**	**61**	62	63
i_8	**64**	**65**	**66**	**67**	**68**	**69**	**70**	**71**	72
i_9	**73**	**74**	**75**	**76**	**77**	**78**	**79**	**80**	**81**

Se observó que las matrices resultantes del entrenamiento de la DEN-*network* son matrices simétricas. Debido a esta característica, solo se requieren los valores de los 45 elementos marcados en negrita de la Tabla 4.15 para representar toda la matriz. Contemplando los 9 elementos del centroide, el tamaño de la memoria utilizada es de 54 localidades. La forma en la que se concatenan los elementos de manera lineal se expresa en la ecuación (32).

$$RAM = \{i_1, i_2, i_3, i_4, i_5, i_6, i_7, i_8, i_9, \mu_9\} \tag{32}$$

Donde i_l, $l = \{1,2,3, \ldots,9\}$, incluye todos los elementos seleccionados del i-ésimo renglón y μ_9 representa los 9 elementos del centroide asociado a una dendrita. Por tanto, $i_1 = \{1\}$, $i_2 = \{10,11\}$, $i_3 = \{19,20,21\}$, …, $i_9 = \{73,74,75,76,77,78,79,80,81\}$. Entonces, la distribución de los elementos en la memoria RAM se puede observar en la Tabla 4.16, donde se muestra la localidad y el elemento que almacena.

Tabla 4.16. Distribución de los elementos del centroide y de la matriz de covarianza inversa en una memoria RAM.

Loc.	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Elem. matriz	**1**	**10**	**11**	**19**	**20**	**21**	**28**	**29**	**30**	**31**	**37**	**38**	**39**	**40**	**41**	**46**	**47**	**48**	**49**	**50**
	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
Elem. matriz	**51**	**55**	**56**	**57**	**58**	**59**	**60**	**61**	**64**	**65**	**66**	**67**	**68**	**69**	**70**	**71**	**73**	**74**	**75**	**76**
	40	41	42	43	44	45	46	47	48	49	50	51	52	53						
Elem. matriz	**77**	**78**	**79**	**80**	**81**	$\boldsymbol{\mu_1}$	$\boldsymbol{\mu_2}$	$\boldsymbol{\mu_3}$	$\boldsymbol{\mu_4}$	$\boldsymbol{\mu_5}$	$\boldsymbol{\mu_6}$	$\boldsymbol{\mu_7}$	$\boldsymbol{\mu_8}$	$\boldsymbol{\mu_9}$						

El ancho de la memoria se determinó analizando los datos que contienen las matrices de covarianza inversa y su modificación del apartado 3.2.3. Se observó que, en la mayoría de los casos de las bases de datos COIL-20 y ALOI, los valores se podían representar con una variable de 31 bits de parte entera. En el caso de que se presente un elemento mayor a 31 bits, dado los algoritmos evolutivos, se puede forzar al sistema para obtener pesos sinápticos cuyos valores no excedan a una variable de 31 bits. De esta manera, se determinó que el ancho óptimo de la memoria seria de 32 bits (1 bit de signo y 31 bits de parte entera).

Para efectuar operaciones de multiplicación con la matriz, se requiere conocer la ubicación de los elementos columna en memoria, ver Tabla 4.17. De esta forma, para realizar dicha operación la lectura de la memoria obedece al algoritmo de la Tabla 4.18.

Tabla 4.17. Localidades de memoria correspondiente a los valores de la $\boldsymbol{j}$-ésima columna.

Índices XY	Localidades de memoria RAM								
	j_1	j_2	j_3	j_4	j_5	j_6	j_7	j_8	j_9
i_1	0	1	3	6	10	15	21	28	36
i_2	1	2	4	7	11	16	22	29	37
i_3	3	4	5	8	12	17	23	30	38
i_4	6	7	8	9	13	18	24	31	39
i_5	10	11	12	13	14	19	25	32	40
i_6	15	16	17	18	19	20	26	33	41
i_7	21	22	23	24	25	26	27	34	42
i_8	28	29	30	31	32	33	34	35	43
i_9	36	37	38	39	40	41	42	43	44

Tabla 4.18. Pseudocódigo para la lectura de la matriz de covarianza inversa para operaciones de multiplicación.

```
00|Metodo LecturaRam ( )
01|Variables
02|     Int i, j, k, addr_M, carga
03|Begin
04|     carga ← 0 // inicialización de la variable temporal
05|     for(i=0 to 9)
06|          addr_M ← carga
07|          for(j = 0 to 9)
08|               RAM(addr_M) //acceso a memoria RAM
09|               if j < i
10|                    addr_M ← addr_M + 1
11|               else
12|                    addr_M ← addr_M + 1 + j
13|          carga ← carga + i
14|end
```

La lógica necesaria para implementar el algoritmo de la Tabla 4.18 se muestra en la Figura 4.30. Donde, el control de la lectura del i,j-ésimo dato recae en 2 contadores, cuyo valor inicial es 1. La condición del incremento en el bus de dirección se realiza mediante una serie de comparadores y un multiplexor.

Como es evidente, para realizar las operaciones que involucran los elementos del centroide, la memoria se puede acceder de manera contigua, partiendo de la localidad 45, por ello, los incrementos en el bus de direcciones se dan de manera lineal, para realizar dicha acción la lógica necesaria se muestra la Figura 4.31, donde el contador utilizado tiene un valor inicial de 1.

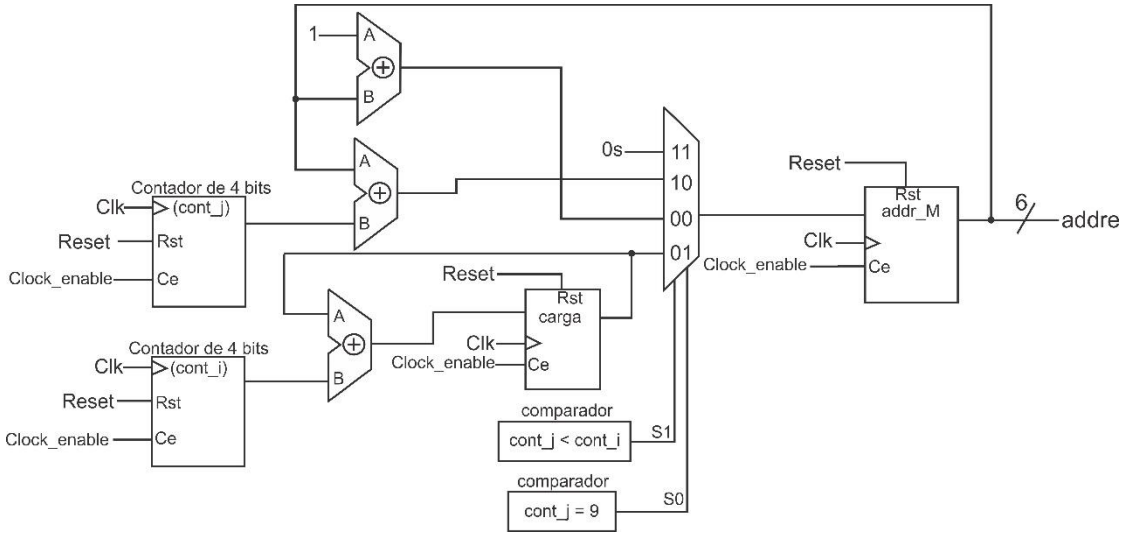

Figura 4.30. Lógica que controla la lectura de los elementos de la matriz de covarianza inversa de la memoria RAM.

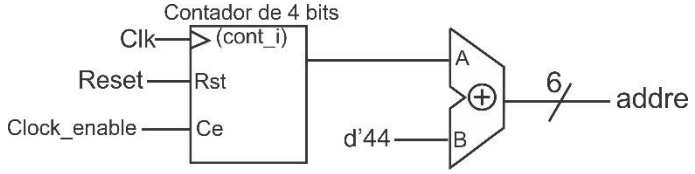

Figura 4.31. Lógica que controla la lectura de los elementos del centroide de la memoria RAM.

4.6.2.2 Arquitectura hardware para la distancia de Mahalanobis

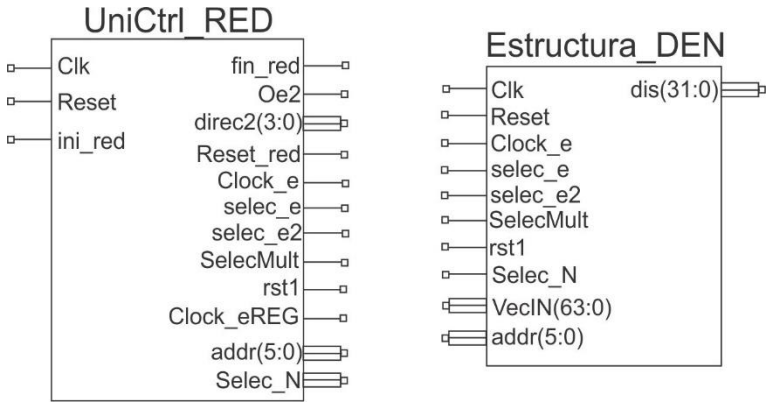

Figura 4.32. Símbolo de los módulos que intervienen en el cálculo de la distancia de Mahalanobis.

Para efectuar el cálculo de la distancia de Mahalanobis se considera la ecuación (18). En el cumplimiento de esta tarea se involucran 2 módulos, **Estructura_DEN**, el cual contiene los componentes necesarios para efectuar las operaciones y **UniCtrl_RED**, encargado de administra las señales de control de **Estructura_DEN** y responsable de realizar las iteraciones necesarias para el proceso completo de clasificación. Los símbolos de los módulos citados se pueden ver en la Figura 4.32 y la descripción de sus señales en las Tablas 4.19 y 4.20.

Tabla 4.19. Descripción de las señales Entrada/Salida del módulo **Estructura_DEN**.

Señal	Descripción
Clk	(Entrada) Señal de reloj del sistema a 100MHz.
Reset	(Entrada) Señal de reset.
Clock_e	(Entrada) Señal de control, activa el clock enable del registro X_C.
selec_e	(Entrada) Señal de control, activa el clock enable del registro VxM.
selec_e2	(Entrada) Señal de control, activa el clock enable del registro VxMxV.
SelecMult	(Entrada) Señal de control, indica que módulo se efectúa su multiplicación.
Selec_N	(Entrada) Señal de selección, indica que dendrita se ocupa por iteración.
rst1	(Entrada) Señal de reset para el registro VxM.
VecIN(63:0)	(Entrada) Bus de datos proveniente de la memoria DualPort.
addr(5:0)	(Entrada) Bus de direcciones de las memorias RAM.
dis(31:0)	(Salida) Bus de datos, cuya información india la distancia de Mahalanobis obtenida por el sistema.

Tabla 4.20. Descripción de las señales Entrada/Salida del módulo **UniCtrl_RED**.

Señal	Descripción
Clk	(Entrada) Señal de reloj del sistema a 100MHz.
Reset	(Entrada) Señal de reset.
ini_red	(Entrada) Señal de control, indica indicio del proceso de clasificacion.
fin_red	(Salida) Señal de control, activa el clock enable del registro X_C.
Reset_red	(Salida) Señal de reset para los registros de la red neuronal.
Oe2	(Salida) Señal de control, activa el tercer buffer te tercer estado de la memoria DualPort.
direc2(3:0)	(Salida) Bus de direcciones para la memoria DualPort.
Clock_e	(Salida) Señal de control, activa el clock enable del registro X_C.
selec_e	(Salida) Señal de control, activa el clock enable del registro VxM.
selec_e2	(Salida) Señal de control, activa el clock enable del registro VxMxV.
SelecMult	(Salida) Señal de control, indica que módulo se efectúa su multiplicación.
rst1	(Salida) Señal de reset para el registro VxM.
Selec_N	(Salida) Señal de selección, indica que dendrita se ocupa por iteración.
VectorIN(63:0)	(Salida) Bus de datos proveniente de la memoria DualPort.
Clock_eREG	(Salida) Señal de control, activa el clock enable de los registros de clase y distancia menor.
addr(5:0)	(Salida) Bus de direcciones para la memoria RAM.
dis(31:0)	(Salida) Bus de datos, cuya información indica la distancia de Mahalanobis obtenida por el sistema.

El módulo **Estructura_DEN** está formado por n-memorias que almacenan los pesos sinápticos de n-dendritas, una serie de registros que permiten almacenar las operaciones realizadas en cada etapa de la ecuación de Mahalanobis, y con el objetivo de optimizar recursos, un solo elemento multiplicador para efectuar las operaciones necesarias, ver Figura 4.33. Por su parte, el módulo **UniCtrl_RED** se construye a partir de una FSM y de un conjunto de lógica que integra y une los componentes de las Figuras 4.30 y 4.31, permitiendo, además de acceder a los pesos sinápticos, el acceso a los elementos del vector de características y la activación de señales de control para calcular la distancia deseada. La lógica utilizada depende principalmente de los bits de estado de la FSM cuyo código es representado por "S" (ver Figuras 4.34 y 4.35).

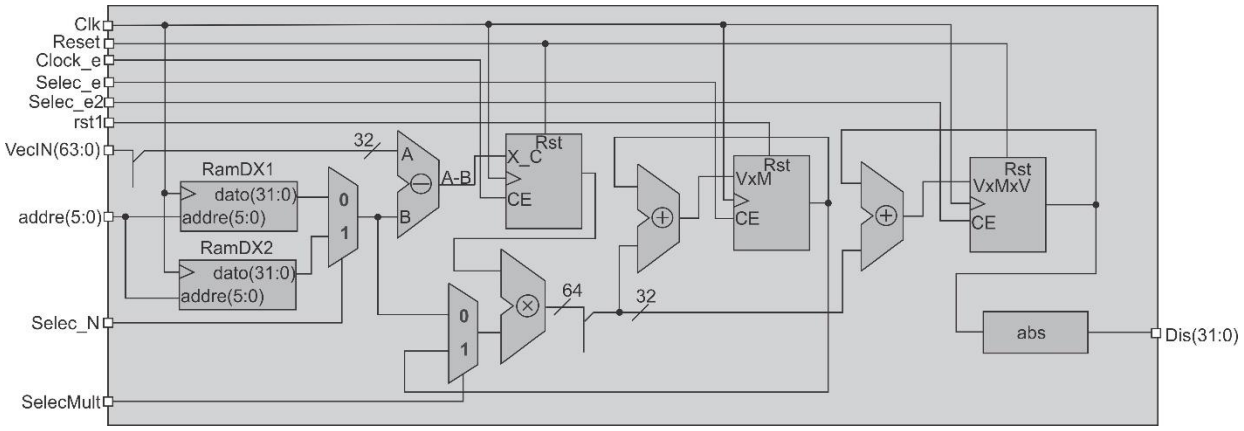

Figura 4.33. Diagrama de bloques del módulo **Estructura_DEN** cuya función es calcular la distancia de Mahalanobis.

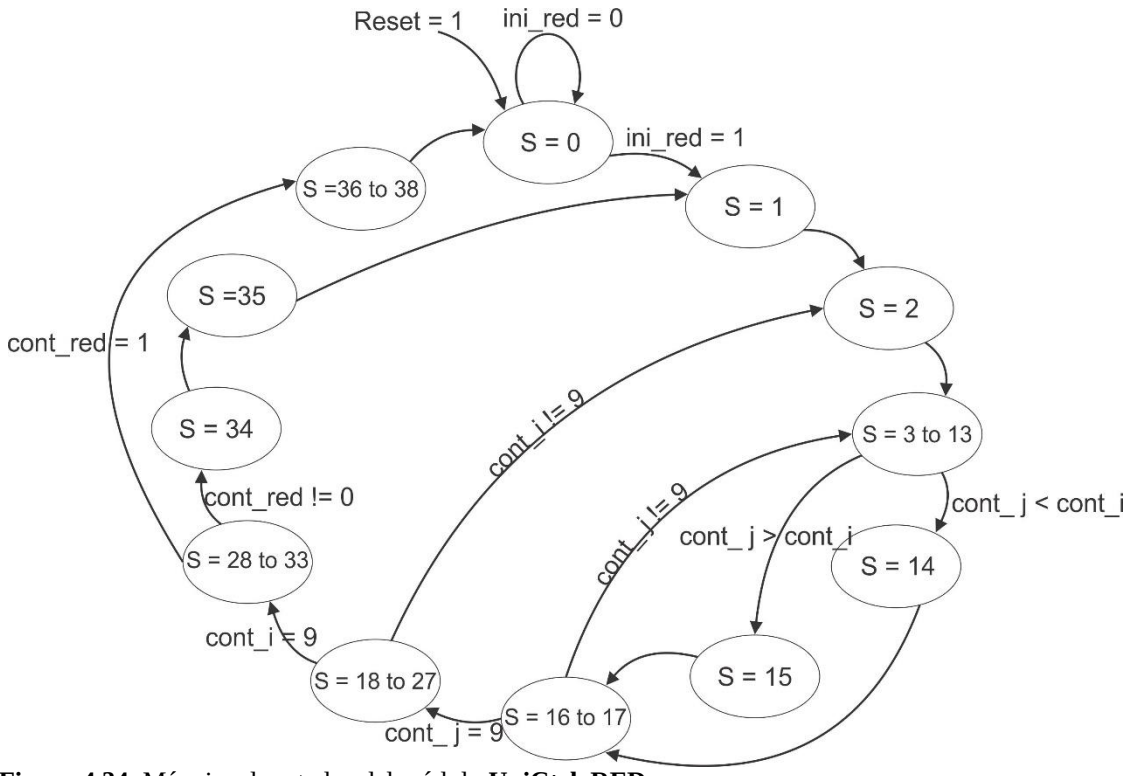

Figura 4.34. Máquina de estados del módulo **UniCtrl_RED**.

Al inicio de cada iteración, en el estado 1, los contadores son reseteados y los registros carga y addr_M son puestos en 0s. Además, se inicializan los registros que contienen las estructuras neuronales, X_C, VxM, y VxMxV, mediante la activación de la señal Reset_red.

Como lo expresa la ecuación (18), la distancia de Mahalanobis se puede dividir en 3 operaciones principales, la primera de ellas corresponde a la expresión $v_i = (x_i - \mu_i)^T$, por lo que se cargan los buses direc2(3:0) y addr(5:0) con las localidades del centroide y elemento del vector de características correspondiente, tras esperar 50ns a que se lleve a cabo la operación, debido al tiempo de propagación, el resultado es almenado en el registro X_C activando la señal Clock_e. El tamaño del registro corresponde al resultado de la operación, es decir, 32 bits, 1 bit de signo, 1 bit de parte entera y 30 bits de parte decimal.

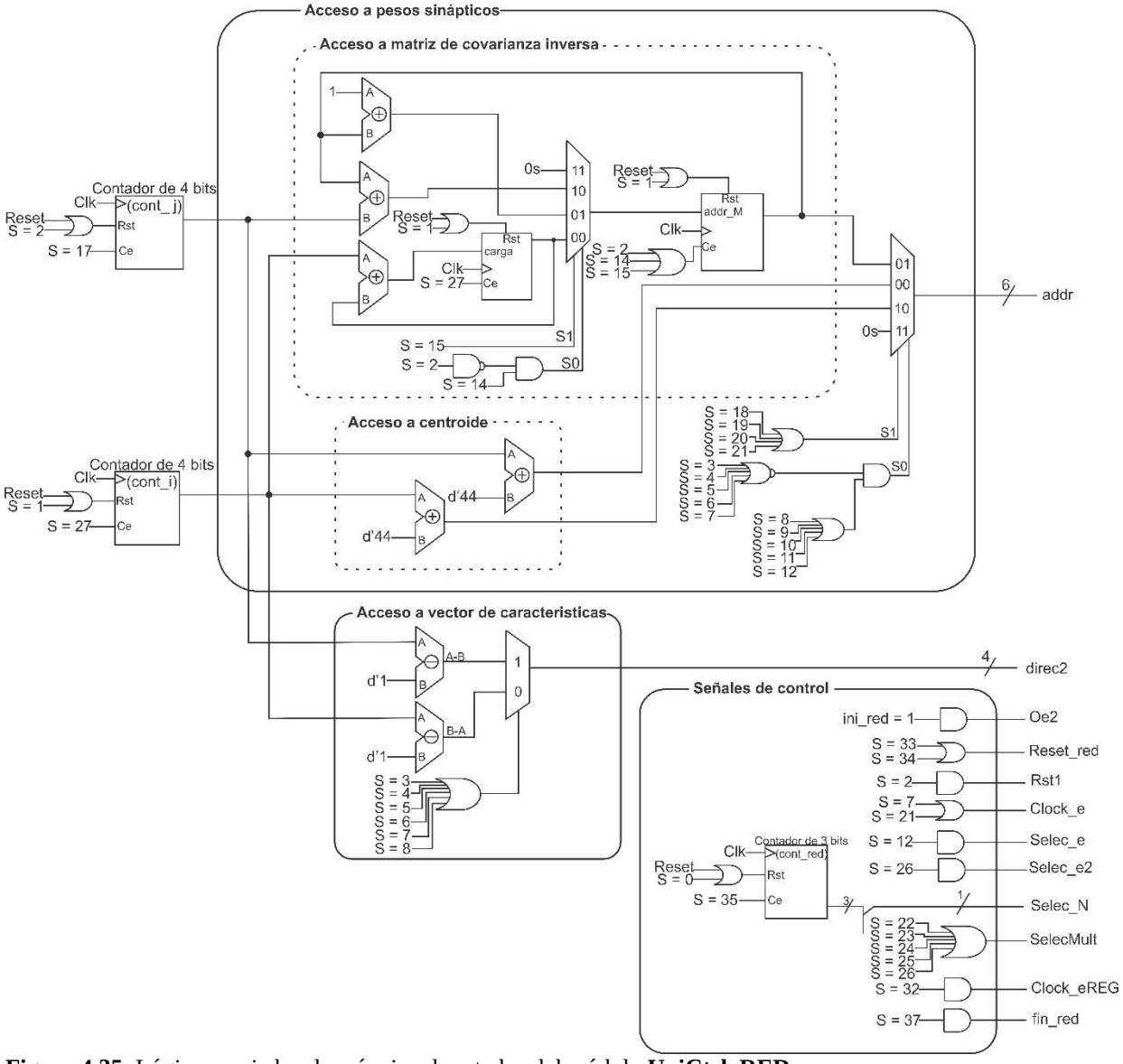

Figura 4.35. Lógica asociada a la máquina de estados del módulo **UniCtrl_RED**.

La segunda operación realiza la acción de multiplicar el vector resultante v_i por un elemento de la matriz de covarianza inversa, $(x_i - \mu_i)^T \times \Sigma^{-1}_{k_{i,j}}$. Una vez cargado el bus de direcciones de la memoria RAM con la localidad correspondiente al i,j-ésimo valor de la matriz, el valor acezado se multiplica con el valor del registro X_C, se esperan 50ns para que se efectué la operación y que el resultado sea acumulado y almacenado en el registro VxM activando la señal Selec_e.

Como se mencionó anteriormente, los elementos de la matriz de covarianza inversa están representados en una variable de 32 bits, por ello, al efectuar la operación de multiplicación, se tiene un resultado de 63 bits, 1 bit de signo, 32 bits de parte entera y 30 bits de parte decimal. En las pruebas realizadas con las bases de datos COIL-20 y ALOI, se observó que nunca puede presentarse el caso en el que el bit más significativo de la parte entera sea igual a uno, por ello, se optó en tomar un segmento del bus resúltate, correspondiente al bit de signo y los 31 elementos menos significativos de la parte entera, puesto que, son los elementos que representan el mayor peso en el cálculo de la distancia de Mahalanobis, esta acción resulta en realizar una aproximación de dicha distancia, de acuerdo a las pruebas de desempeño esta acción no repercute

de manera significativa en el resultado. Cabe mencionar que, al utilizar otras bases de datos se debe verificar si el bit más significativo representa un valor valido para las operaciones.

La tercera operación se realiza después de obtener la suma de productos correspondientes a la j-ésima columna, de esta forma, se calcula y almacena el elemento v_j en el registro X_C, por lo que el resultado es multiplicado con el valor del registro VxM, esta acción corresponde a la operación $(\mathbf{x}-\boldsymbol{\mu})^T \times \boldsymbol{\Sigma}_{k_j}^{-1} \times (x_j - \mu_j)$. 50ns después, el valor del bus resultante se acumula y almacena en el registro VxMxV activando la señal Selec_e2; debido a que se utiliza una diferente entrada en el elemento multiplicador, se activa la señal de selección SelecMult durante los 50ns, consecutivamente, el registro VXM es puesto en 0 para los cálculos de la siguiente columna. Tras efectuar la tercera operación 9 veces, se obtiene la distancia de Mahalanobis deseada, cabe mencionar que, al recortar los valores en las operaciones se pierde resolución y el valor final puede ser negativo, lo que a su vez afecta el desempeño del sistema, por ello, se implementa un módulo de valor absoluto en la salida.

Finalmente, a través del contador cont_red, se lleva el control de la cantidad de iteraciones necesarias para la clasificación completa del objeto de entrada, para el ejemplo de la red propuesta son requeridas 2 iteraciones del proceso del cálculo de la distancia de Mahalanobis.

4.6.2.2.1 Consideraciones de la DEN-*network* en el FPGA

Al realizar la multiplicación de 2 señales de 32 bits son requeridos 4 multiplicadores embebidos del FPGA utilizado. Como el descriptor HOG en su diseño hace uso de 12 multiplicadores, quedan disponibles 20, lo que permitió en el sistema real implementado utilizar 5 estructuras neuronales que realizan su operación de manera concurrente.

En el caso del sistema en el FPGA Spartan 6, la cantidad de iteraciones máximas son 7, las cuales corresponden al proceso de clasificar 20 clases de la base de datos COIL-20, por ello el contador que se muestra en la Figura 4.35, corresponde a un contador de 3 bits. Cabe mencionar que la salida del contador cont_red actúa como un bus de selección para el multiplexor que indica la información del conjunto de dendritas que se activan en la k-ésima iteración.

Al distribuir la información de las dendritas, de tal manera que permitan minimizar el tiempo de clasificación se pueden presentar 2 casos posibles, aquel donde la estructura neuronal tenga asociadas n dendritas; y aquel donde existen $n-1$ dendritas en una estructura neuronal. Si ocurre el segundo caso, se opta por realizar una conexión doble del último elemento en el multiplexor que permite indicar que dendrita se activa, por ende, el cálculo de la distancia de Mahalanobis de la última dendrita ocurrirá 2 veces en esa estructura neuronal. Para el ejemplo propuesto los casos presentados quedarían conforme a la Figura 4.36.

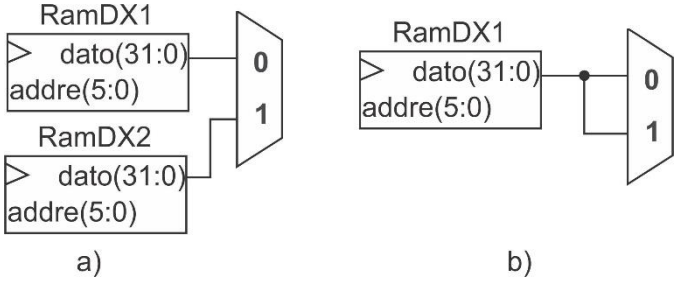

Figura 4.36. Configuración de la memoria de acuerdo con la estructura neuronal. a) Caso donde existen 2 dendritas por estructura. b) Caso donde existe solo 1 dendrita por estructura.

4.6.3 Evaluación de argumento mínimo

Terminado el proceso del cálculo de la distancia de Mahalanobis en cada iteración, la información de salida de cada estructura neuronal, así como la menor distancia de la iteración anterior, pasa a una serie de comparadores conectados en cascada cuyo resultado final corresponde a la menor distancia obtenida hasta la iteración correspondiente.

Cada comparador evalúa 2 elementos (a y b) y regresa la información de que entrada es la menor (un 1 si a < b) así como su respectivo valor, para el ejemplo propuesto se requieren 3 comparadores (A, B, C), por lo que tabla de verdad correspondiente se muestra en la Tabla 4.21. Terminando la evaluación se almacenan en registros la distancia menor y la clase a la que pertenece mediante la activación de la señal Clock_eREG generada por el módulo **UniCtrl_RED**. Para el caso de la primera iteración, el registro donde se almacena la menor distancia es precargado con la mayor distancia posible, es decir 2^{31}, y el registro que lleva el control de a qué clase pertenece dicha distancia, es precargado con un valor de 0.

Tabla 4.21. Tabla de verdad del codificador del módulo **DEN-*network***.

A	B	C	O_1	O_0
0	0	0	0	0
0	0	1	0	1
0	1	X	1	0
1	X	X	1	1

El módulo **Mux_Class** es el encargado de indicar la clase perteneciente a la menor distancia obtenida en cada iteración. Para realizar esta tarea, primeramente, las clases que contiene una estructura neuronal se encuentran asociadas a un multiplexor cuyo bus de selección es la señal Selec_N y permite indicar que dendrita se está activando, para el ejemplo propuesto el conjunto de multiplexores quedaría conforme a la Figura 4.37. Finalmente, mediante el bus de selección Cod(1:0) del codificador se indica la clase de la menor distancia. El bus de datos Class(1:0) pertenece al registro que guarda la clase de la iteración anterior.

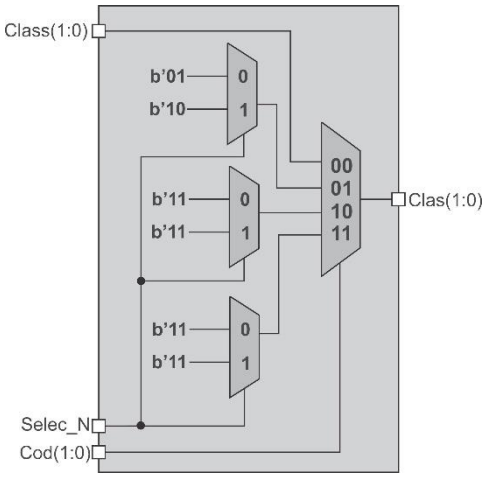

Figura 4.37. Esquema de multiplexores de las dendritas en unidades de procesamiento.

4.7 Resultados de la implementación en hardware

En esta sección se presentan una serie de experimentos que tienen como objetivo medir el desempeño de la implementación en hardware del sistema de reconocimiento descrito en los apartados anteriores. En el primer experimento, para implementar la estructura del módulo **DEN-network** se utilizaron los valores de las matrices de covarianza inversas y de los centroides generados en el modelado conceptual.

En las Tabla 4.22 y 4.23 se presentan los resultados del primer experimento. Con la finalidad de poder establecer una comparativa, se presentan los resultados obtenidos por el modelado conceptual y por el sistema implementado en el FPGA. La Tabla 4.22 permite apreciar que para la base de datos COIL-20 la eficiencia del sistema implementado en el FPGA decae con respecto al modelado conceptual, mientras que el desempeño con la base de datos ALOI, se ve incrementado para un alto número de clases (ver Tabla 4.23). En estas Tablas también se presenta el número de unidades de procesamiento requeridas por el número de clases, esto ayudara a comprender resultados posteriores.

Tabla 4.22. Comparación de desempeño entre el modelado conceptual y el sistema implementado en el FPGA para la base de datos COIL-20.

N° de clases	N° de dendritas	Porcentaje de error en training Modelado conceptual	Porcentaje de error en testing Modelado conceptual	Porcentaje de error en testing FPGA
2	2	0	**2.77**	5.55
3	3	0	**2.77**	4.62
5	8	0	**1.11**	**1.11**
10	17	7.77	**3.61**	7.77
15	22	4.07	**8.14**	8.70
20	33	5.41	**11.80**	15.69

Tabla 4.23. Comparación de desempeño entre el modelado conceptual y el sistema implementado en el FPGA para la base de datos ALOI.

N° de clases	N° de dendritas	Porcentaje de error en training Modelado conceptual	Porcentaje de error en testing Modelado conceptual	Porcentaje de error en testing FPGA
2	2	0	**0**	**0**
3	3	0	**0**	**0**
5	5	0	**0**	**0**
10	12	0.83	**5.83**	6.11
15	17	7.22	9.25	**7.59**
20	25	1.25	6.11	**3.75**

En el segundo experimento se utilizó la arquitectura hardware de HOG para obtener los vectores de características de los patrones pertenecientes al conjunto de entrenamiento y el modelado conceptual de la DEN-*network* para calcular los valores de las matrices de covarianza inversa y de los centroides que fueron utilizados en la implementación de la estructura del módulo **DEN-network**. Los resultados de este experimento para las bases de datos COIL-20 y ALOI se pueden consultar en las Tablas 4.24 y 4.25 respectivamente. Los resultados del modelado conceptual reportados en estos experimentos utilizaron, tanto en training como en testing, los vectores de características generados por la arquitectura hardware de HOG. De manera similar al experimento anterior, la eficiencia del sistema en el FPGA es menor a la del modelado conceptual para la

mayoría de los casos, sin embargo, se puede apreciar que, al realizar el entrenamiento con vectores creados en el FPGA, el desempeño del sistema aumenta con respecto al sistema formado con los vectores de características provenientes del modelado conceptual.

Tabla 4.24. Comparación de desempeño entre el modelado conceptual y el sistema implementado en el FPGA para la base de datos COIL-20 empleando vectores de entrenamiento generados por en el FPGA.

N° de clases	N° de dendritas	Porcentaje de error en training Modelado conceptual	Porcentaje de error en testing Modelado conceptual	Porcentaje de error en testing FPGA
2	2	0	**2.77**	5.55
3	3	0	**2.77**	4.62
5	7	0	1.11	**0.55**
10	15	3.88	**7.77**	10.55
15	24	5.18	**8.14**	9.07
20	31	9.44	15	**13.33**

Tabla 4.25. Comparación de desempeño entre el modelado conceptual y el sistema implementado en el FPGA para la base de datos ALOI empleando vectores de entrenamiento generados por en el FPGA.

N° de clases	N° de dendritas	Porcentaje de error en training Modelado conceptual	Porcentaje de error en testing Modelado conceptual	Porcentaje de error en testing FPGA
2	2	0	**0**	**0**
3	3	0	**0**	**0**
5	5	0	**0**	**0**
10	12	2.22	**5.27**	5.55
15	19	5.37	9.07	**7.59**
20	25	0.97	5.83	**4.86**

El tercer experimento tiene por objetivo mostrar el desempeño que presenta el sistema de reconocimiento implementado en el FPGA cuando es adaptado para reconocer diversos grupos de clases. Con este fin y similar a lo realizado con el modelado conceptual en el apartado 3.4, la evaluación del sistema implementado en el FPGA utilizó las matrices de confusión y sus métricas asociadas definidas por las ecuaciones (25), (26), (27), (28) y (29). A modo de ejemplo detallado, inicialmente se muestra el desempeño del sistema implementado en el FPGA que ha sido adaptado para reconocer 5 clases. La Tabla 4.26 muestra la matriz de confusión que genera este grupo de clases y la Tabla 4.27 agrupa el cálculo de las métricas asociadas a la mencionada matriz de confusión.

Tabla 4.26. Matriz de confusión obtenida al aplicar el sistema implementado en el FPGA al reconocimiento de 5 clases de la base de datos COIL-20.

Clase	Predicciones del sistema final modificado				
	(a)	(b)	(c)	(d)	(e)
(a)	36	0	0	0	0
(b)	0	36	0	0	0
(c)	0	0	36	0	0
(d)	0	0	0	36	0
(e)	0	0	1	0	35

Tabla 4.27. Métricas de desempeño obtenidas a partir de la matriz de confusión de la Tabla 4.26.

Clase\rate	TP	TPR	TN	TNR	Exactitud	Te	Precisión
(a)	36	1	144	1	1	0	1
(b)	36	1	144	1	1	0	1
(c)	36	1	143	0.9931	0.9944	0.0056	0.9730
(d)	36	1	144	1	1	0	1
(e)	35	0.9722	144	1	0.9944	0.0056	1
avg		0.9944		0.9986	0.9978	0.0022	0.9946

Como parte de este experimento, se determinó el desempeño del sistema implementado en el FPGA cuando es adaptado para reconocer otros grupos de clases y se comparó con los resultados obtenidos por el modela conceptual discutido en el capítulo anterior. Las Tablas 4.28 y 4.29 exponen los resultados de este comparativo para imágenes de las bases de datos COIL-20 y ALOI respectivamente.

Tabla 4.28. Comparación entre el modelado conceptual y la implementación en FPGA cuando se utilizan con la base de datos COIL-20 para diferente número de clases.

N° de clases	Base de datos COIL-20									
	Modelado Conceptual (avg)					Implementación FPGA (avg)				
	TPR	TNR	Exact.	Te	Precisión	TPR	TNR	Exact.	Te	Precisión
2	0.9722	0.9722	0.9722	0.0278	0.9737	0.9444	0.9444	0.9444	0.0556	0.9500
3	0.9722	0.9861	0.9815	0.0185	0.9744	0.9537	0.9769	0.9691	0.0309	0.9593
5	0.9889	0.9972	0.9956	0.0044	0.9895	0.9944	0.9986	0.9978	0.0022	0.9946
10	0.9222	0.9914	0.9844	0.0156	0.9322	0.8944	0.9883	0.9789	0.0211	0.9190
15	0.9185	0.9942	0.9891	0.0109	0.9251	0.9093	0.9935	0.9879	0.0121	0.9190
20	0.8819	0.9938	0.9882	0.0118	0.9148	0.8528	0.9923	0.9853	0.0147	0.8848

Tabla 4.29. Comparación entre el modelado conceptual y la implementación en FPGA cuando se utilizan con la base de datos ALOI para diferente número de clases.

N° de clases	Base de datos ALOI									
	Modelado Conceptual (avg)					Implementación FPGA (avg)				
	TPR	TNR	Exact.	Te	Precisión	TPR	TNR	Exact.	Te	Precisión
2	1	1	1	0	1	1	1	1	0	1
3	1	1	1	0	1	1	1	1	0	1
5	1	1	1	0	1	1	1	1	0	1
10	0.9417	0.9935	0.9883	0.0117	0.9484	0.9444	0.9938	0.9889	0.0111	0.9521
15	0.9074	0.9934	0.9877	0.0123	0.9521	0.9241	0.9946	0.9899	0.0101	0.9559
20	0.9389	0.9968	0.9939	0.0061	0.9498	0.9514	0.9974	0.9951	0.0049	0.9537

En el último experimento se muestra la comparación en la velocidad de procesamiento cuando el sistema propuesto es implementado en una PC (modelado conceptual) y el FPGA. La comparación se realizó utilizando el número máximo de dendritas que se presentó en los casos propuestos, ver Tabla 4.30. La PC utilizada cuenta con un procesador i7 de 5° generación de 2 núcleos y 4 procesadores lógicos cuya velocidad de operación máxima es de 2.4GHz. El proceso de clasificación en el FPGA se realiza en un tiempo de $12.39\mu s$, por cada conjunto de 5 dendritas en una iteración.

Tabla 4.30. Velocidades de procesamiento del proceso de clasificación implementado en una PC y en FPGA.

N° de clases	N° de dendritas en el procesador	N° de iteraciones en el procesador	Procesador i7 2.4GHz, 4 núcleos (μs)	N° de iteraciones en el FPGA	FPGA Spartan6 XC6LX16-CS324 (μs)
2	2	2	76	1	12.39
3	3	3	137	1	12.39
5	8	8	269	2	24.78
10	17	17	349	4	49.56
15	22	22	618	5	61.95
20	33	33	1,112	7	86.73

Cabe aclarar que las técnicas implementadas en el procesador son utilizando los algoritmos en forma secuencial, es decir, no se explotaron técnicas de paralelismo o multinúcleo en la evaluación del sistema.

De igual forma se obtuvo el tiempo de procesamiento (t_v) para obtener un vector de características valido, como se pudo observar en el apartado 4.5.4, la cantidad de ciclos de reloj que requiere procesar un píxel es de 8, sumando a los 139 ciclos que requiere realizar el proceso de normalización se tiene la siguiente ecuación,

$$t_v = \frac{1}{F}(8 \times P_T + 139) \tag{33}$$

Donde F, es la frecuencia de operación y P_T es el número de píxeles totales de la imagen. De esta forma, debido a que se trabajó a una frecuencia de operación de 100MHz y se tienen 16,384 píxeles correspondientes a una imagen de 128×128, el tiempo para obtener el vector de características de acuerdo con las características de bases de datos COIL-20 y ALOI es de $t_v = 1.312ms$. El tiempo obtenido más el tiempo máximo mostrado en la clasificación de un objeto da un máximo de $t = 1.398ms$ para efectuar el proceso de reconocimiento del sistema. Por su parte, el resultado obtenido en la extracción de características en la PC se da en $t = 218.99\,ms$, por ende, el tiempo de extracción de características y clasificación máximo en la PC es de $t = 221.102ms$.

Finalmente, la Tabla 4.31 muestra la cantidad de recursos utilizados del FPGA Spartan 6 XC6LX16-CS324 cuando es implementada la arquitectura hardware del sistema de reconocimiento propuesto. Los resultados mostrados corresponden al caso extremo de 33 dendritas generadas, el cual pertenece a la base de datos COIL-20 con 20 clases. Además, la Tabla 4.31 incluye el porcentaje de recursos utilizados por los módulos **HOG** y **DEN-network**.

Tabla 4.31. Porcentaje de recursos utilizados del FPGA Spartan6 XC6LX16-CS324 en la implementación del sistema de reconocimiento.

Lógica Utilizada	Recursos totales	Sistema Final (%)	**HOG** (%)	**DEN-network** (%)
Registros Slices	18224	5.76	1.50	3.95
LUTs Slices	9112	46.44	28.35	17.30
Pares de LUT-FF	4694	12.52	3.25	8.24
Bloques RAM	32	9.37	9.37	0
DSP48A1S	32	100	37.5	62.5

En la Tabla 4.32 se muestra el porcentaje de recursos utilizados por cada componente del módulo **HOG**, así como de la unidad de control al módulo asociado.

Tabla 4.32. Porcentaje de recursos utilizados del FPGA Spartan6 XC6LX16-CS324 en la implementación del módulo **HOG**.

Lógica Utilizada	Recursos totales	Unidad de Ctrl (%)	Gradiente (%)	Histograma (%)	Normalización (%)
Registros Slices	18224	0.04	0.57	0.03	0.85
LUTs Slices	9112	0.16	4.83	2.82	20.53
Pares de LUT-FF	4694	0.17	1.32	0.14	1.61
Bloques RAM	32	0	3.12	6.25	0
DSP48A1S	32	0	12.5	12.5	12.5

El porcentaje de recursos de la Tabla 4.33 hace referencia a la cantidad de recursos utilizados por la unidad de control asociada al módulo **DEN-network** y a la estructura neuronal, la cual hace uso de una mayor cantidad de recursos. La estructura neuronal mostrada hace uso de la información de 7 dendritas.

Tabla 4.33. Porcentaje de recursos utilizados del FPGA Spartan6 XC6LX16-CS324 en la implementación del módulo **DEN-network**.

Lógica Utilizada	Recursos Totales	Unidad de Ctrl (%)	Estructura Neuronal (%)
Registros Slices	18224	0.22	1.58
LUTs Slices	9112	1.82	4.66
Pares de LUT-FF	4694	0.68	5.51
Bloques RAM	32	0	0
DSP48A1S	32	0	12.5

Capítulo 5

Conclusiones y Trabajo Futuro

En este trabajo de tesis se planteó la implementación de una arquitectura hardware de un sistema de reconocimiento plasmando en lógica reconfigurable los algoritmos del descriptor HOG y del clasificador DEN-*network*. El presente capítulo documenta las conclusiones a las que se han llegado a partir de los resultados obtenidos de los experimentos y pruebas descritas y realizas en los apartados anteriores. Además, se presentan las perspectivas y/o trabajos futuros que plantea la continuación de esta investigación.

5.1 Conclusiones

1. Uniendo la extracción de características y la clasificación dentro de la arquitectura hardware se comprobó que para la base de datos ALOI se tiene una eficiencia superior al 90% para el numero de clases presentadas, mientras que para la base de datos COIL-20 se tiene una eficiencia superior al 90% para conjuntos inferiores a 20 clases, sin embargo, cabe recalcar que para 20 clases el sistema tiene una eficiencia mayor al 85%.
2. Dado el tiempo de procesamiento obtenido, se estimó que se pueden procesar 762 fps, donde cada fotograma debe estar compuesto por 128×128 píxeles. Cabe recalcar que se pueden procesar fotogramas de una dimensión mayor, cuyo tiempo es estimado por la ecuación (33).
3. El sistema propuesto en esta investigación utiliza aproximadamente el 50 % de los recursos que un FPGA Spartan6-XC6SLX16 posee. Esto implica que se tienen suficientes

recursos para incrementar las prestaciones del sistema o implementar otros procesamientos que pudieran requerirse.

4. Analizando el algoritmo de HOG se comprobó que es eficiente al brindar vectores de características adecuados para una clasificación multi-objetos.
5. Tras realizar el modelado conceptual del descriptor HOG se pudo comprobar que las modificaciones hechas, para facilitar las operaciones y minimizar los recursos en el FPGA, no dieron cambios significativos en el desempeño del sistema.
6. Una vez realizado el modelado conceptual de la DEN-*network* se observó que la matriz de covarianza inversa es una matriz simétrica lo que permitió el uso compacto de la información necesaria para ser utilizada en el FPGA, lo que a su vez implico una reducción de recursos.
7. El diseño en hardware del descriptor HOG en el FPGA permitió realizar las primeras fases del algoritmo de manera concurrente, así como el uso de registros y una memoria DualPort brindaron las herramientas necesarias para minimizar el tiempo de operación en los cálculos necesarios.
8. Debido a la naturaleza de la DEN-*network* y el conjunto de las modificaciones mostradas, se observó un aumento en la eficiencia de desempeño para las bases de datos COIL-20 y ALOI.
9. Al contemplar la cantidad de recursos disponibles en el FPGA para el diseño de la DEN-*network*, tras llevar acabo la extracción de características, se implementaron 5 estructuras neuronales fijas que pueden ser utilizadas por una cantidad de n-dendritas, donde las operaciones son realizadas de manera simultánea entre cada estructura, permitiendo reducir la cantidad de iteraciones necesarias entre cada operación y de esta forma disminuyendo el tiempo de clasificación del sistema.

5.2 Trabajos futuros

- Implementar la fase de segmentación dentro del FPGA.
- Incorporar una cámara a la tarjeta Nexys 3 para obtener una imagen y procesarla directamente dentro del FPGA, omitiendo a la PC en las operaciones restantes.
- Implementar en un sistema autónomo el sistema diseñado.
- Realizar el sistema de reconocimiento diseñado en un dispositivo de gama alta para tratar de reducir el tiempo de propagación existente en las operaciones realizadas debido a los componentes del sistema, además de aumentar el número de estructuras neuronales en el diseño de la DEN-*network*, permitiendo reducir el tiempo de clasificación.

Referencias

[1] Giménez, J.M. (2000). *Anatomía funcional de la corteza cerebral implicada en los procesos visuales,* REVNEUROL, 30(7), 656-662.

[2] Urtubia, C. (1999). *Neurobiología de la visión*, Cataluña, España, Edicions UPC. pp 55.

[3] Jahne,B. HauBecker, H. (2000). *Computer vision and aplications a guide for students and practitioners,* San Diego CA, USA, Academic Press.

[4] Veit, P. (2017). *Artificial Vision: A Clinical Guide*, Cham, Switzerland, Springer.

[5] Mundy, J. Zisserman, A. (1992). *Geometric Invariance in Computer Vision,* London, England, Cambridge, Massachusetts, The MIT press.

[6] Lu, H., & Li, Y. (Eds.). (2017). Artificial intelligence and computer vision. Springer.

[7] Ott, P., & Everingham, M. (2009, September). Implicit color segmentation features for pedestrian and object detection. In Computer vision, 2009 IEEE 12th international conference on (pp. 723-730). IEEE.

[8] Iamsa-at, S., & Horata, P. (2013, December). Handwritten character recognition using histograms of oriented gradient features in deep learning of artificial neural network. In IT Convergence and Security (ICITCS), 2013 International Conference on (pp. 1-5). IEEE.

[9] Arce, F., Zamora, E., & Sossa, H. (2017, May). Dendrite Ellipsoidal Neuron. In Neural Networks (IJCNN), 2017 International Joint Conference on (pp. 795-802). IEEE.

[10] Boston Dynamics (2013). WildCat: The Wold's Fastest Quadruped Robot https://www.bostondynamics.com/wildcat

[11] Bharath, A. A., & Petrou, M. (2008). Next generation artificial vision systems: Reverse engineering the human visual system. Artech House.

[12] Morris, B. (2004). Molecular biology of the neuron. Oxford University Press.

[13] Roberts, L. G. (1963). Machine perception of three-dimensional solids (Doctoral dissertation, Massachusetts Institute of Technology).

[14] Sobrado, E.A. (2003). *Sistema de visión artificial para el reconocimiento y manipulación de objetos utilizando un brazo robot.* Tesis de Maestría, Pontificia Univ. Católica del Perú, Lima, Perú.

[15] Aleksander, I. (1983). *Artificial Vision For Robots.* London: Kogan Page.

[16] Rodríguez, F. J., Mazo, M., & Sotelo, M. A. (1998). *Automation of an industrial fork lift truck, guided by artificial vision in open environments. Autonomous Robots*, 5(2), 215-231.

[17] Zhang, G., & Wei, Z. (2002). *A novel calibration approach to structured light 3D vision inspection*. Optics & Laser Technology, 34(5), 373-380.

[18] Broggi, A., Cerri, P., & Antonello, P. C. (2004, June). Multi-resolution vehicle detection using artificial vision. In Intelligent Vehicles Symposium, 2004 IEEE (pp. 310-314). IEEE

[19] Guyer, D., & Yang, X. (2000). Use of genetic artificial neural networks and spectral imaging for defect detection on cherries. Computers and electronics in agriculture, 29(3), 179-194.

[20] Xie, X. (2008). A review of recent advances in surface defect detection using texture analysis techniques. ELCVIA Electronic Letters on Computer Vision and Image Analysis, 7(3), 1-22.

[21] Aguilar-Torres, M. A., Argüelles-Cruz, A. J., & Yánez-Márquez, C. (2008, September). A real time artificial vision implementation for quality inspection of industrial products. In Electronics, Robotics and Automotive Mechanics Conference, 2008. CERMA'08 (pp. 277-282). IEEE.

[22] Cubero, S., Aleixos, N., Moltó, E., Gómez-Sanchis, J., & Blasco, J. (2011). Advances in machine vision applications for automatic inspection and quality evaluation of fruits and vegetables. Food and Bioprocess Technology, 4(4), 487-504.

[23] García, A. (2015). *Reconocimiento de objetos inmersos en imágenes estáticas mediante el algoritmo HOG y RNA-MLP*. Tesis de Maestría, Universidad Tecnológica de la Mixteca, Oaxaca, México.

[24] Bodor, R., Jackson, B., & Papanikolopoulos, N. (2003, June). *Vision-based human tracking and activity recognition*. In Proc. of the 11th Mediterranean Conf. on Control and Automation (Vol. 1).

[25] Ng, C. M., Ng, V., & Lau, Y. (1999, September). Regular feature extraction for recognition of Braille. In iccima (p. 302). IEEE.

[26] Akhtar, Z., Micheloni, C., & Foresti, G. L. (2015). Biometric liveness detection: Challenges and research opportunities. IEEE Security & Privacy, 13(5), 63-72.

[27] Wolf, J., Burgard, W., & Burkhardt, H. (2005). Robust vision-based localization by combining an image-retrieval system with Monte Carlo localization. IEEE transactions on robotics, 21(2), 208-216.

[28] Dickmanns, E. D. (2002, June). The development of machine vision for road vehicles in the last decade. In Intelligent Vehicle Symposium, 2002. IEEE (Vol. 1, pp. 268-281). IEEE.

[29] Dalal, N., & Triggs, B. (2005, June). Histograms of oriented gradients for human detection. In Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on (Vol. 1, pp. 886-893). IEEE.

[30] Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. International journal of computer vision, 60(2), 91-110.

[31] Bay, H., Ess, A., Tuytelaars, T., & Van Gool, L. (2008). Speeded-up robust features (SURF). Computer vision and image understanding, 110(3), 346-359.

[32] Tola, E., Lepetit, V., & Fua, P. (2008). A fast local descriptor for dense matching.

[33] Ojala, T., Pietikäinen, M., & Harwood, D. (1996). A comparative study of texture measures with classification based on featured distributions. Pattern recognition, 29(1), 51-59.

[34] Dan, S. G. F. D. L., & Archibald, V. J. K. (2011). Nature-Inspired BASIS Feature Descriptor and Its Hardware Implementation.

[35] Bishop, C. M. (2006). Pattern recognition and machine learning. Springer

[36] Bow, S. T. (Ed.). (2002). Pattern recognition and image preprocessing. CRC press.

[37] Theodoridis, S., & Koutroumbas, K. (2003). Pattern Recognition fourth edition.

[38] McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics, 5(4), 115-133.

[39] Sossa, H., & Guevara, E. (2014). Efficient training for dendrite morphological neural networks. Neurocomputing, 131, 132-142.

[40] Rall, W. (1960). Membrane potential transients and membrane time constant of motoneurons. Experimental neurology, 2(5), 503-532.

[41] Rall, W. (1967). Distinguishing theoretical synaptic potentials computed for different soma-dendritic distributions of synaptic input. Journal of neurophysiology, 30(5), 1138-1168.

[42] Ritter, G. X., & Urcid, G. (2003). Lattice algebra approach to single-neuron computation. IEEE Transactions on Neural Networks, 14(2), 282-295.

[43] Penrose, R. (1955, July). A generalized inverse for matrices. In Mathematical proceedings of the Cambridge philosophical society (Vol. 51, No. 3, pp. 406-413). Cambridge University Press.

[44] Xilinx , "XC2000 Logic Cell Array Families," 1985.

[45] nvidia (2019). ¿Qué es la computación acelerada por GPU? https://la.nvidia.com/object/what-is-gpu-computing-la.html.

[46] Asano, S., Maruyama, T., & Yamaguchi, Y. (2009, August). Performance comparison of FPGA, GPU and CPU in image processing. In 2009 international conference on field programmable logic and applications (pp. 126-131). IEEE.

[47] Chase, J., Nelson, B., Bodily, J., Wei, Z., & Lee, D. J. (2008, April). Real-time optical flow calculations on FPGA and GPU architectures: a comparison study. In 2008 16th International Symposium on Field-Programmable Custom Computing Machines (pp. 173-182). IEEE.

[48] Bodily, J., Nelson, B., Wei, Z., Lee, D. J., & Chase, J. (2010). A comparison study on implementing optical flow and digital communications on FPGAs and GPUs. ACM Transactions on Reconfigurable Technology and Systems (TRETS), 3(2), 6.

[49] Fowers, J., Brown, G., Cooke, P., & Stitt, G. (2012, February). A performance and energy comparison of FPGAs, GPUs, and multicores for sliding-window applications. In Proceedings of

the ACM/SIGDA international symposium on Field Programmable Gate Arrays (pp. 47-56). ACM.

[50] Bauer, S., Brunsmann, U., & Schlotterbeck-Macht, S. (2009, July). FPGA implementation of a HOG-based pedestrian recognition system. In Proc. MPC-Workshop (pp. 49-58).

[51] Kadota, R., Sugano, H., Hiromoto, M., Ochi, H., Miyamoto, R., & Nakamura, Y. (2009, September). Hardware architecture for HOG feature extraction. In Intelligent Information Hiding and Multimedia Signal Processing, 2009. IIH-MSP'09. Fifth International Conference on (pp. 1330-1333). IEEE.

[52] Yao, L., Feng, H., Zhu, Y., Jiang, Z., Zhao, D., & Feng, W. (2009, December). An architecture of optimised SIFT feature detection for an FPGA implementation of an image matcher. In 2009 International Conference on Field-Programmable Technology (pp. 30-37). IEEE.

[53] Farabet, C., Poulet, C., Han, J. Y., & LeCun, Y. (2009, August). Cnp: An fpga-based processor for convolutional networks. In Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on (pp. 32-37). IEEE.

[54] Farabet, C., Martini, B., Akselrod, P., Talay, S., LeCun, Y., & Culurciello, E. (2010, May). Hardware accelerated convolutional neural networks for synthetic vision systems. In Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on (pp. 257-260). IEEE.

[55] Negi, K., Dohi, K., Shibata, Y., & Oguri, K. (2011, December). Deep pipelined one-chip FPGA implementation of a real-time image-based human detection algorithm. In 2011 International Conference on Field-Programmable Technology (pp. 1-8). IEEE.

[56] Chen, P. Y., Huang, C. C., Lien, C. Y., & Tsai, Y. H. (2014). An efficient hardware implementation of HOG feature extraction for human detection. IEEE Transactions on Intelligent Transportation Systems, 15(2), 656-662.

[57] Zhang, C., Li, P., Sun, G., Guan, Y., Xiao, B., & Cong, J. (2015, February). Optimizing fpga-based accelerator design for deep convolutional neural networks. In Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (pp. 161-170). ACM.

[58] Rettkowski, J., Boutros, A., & Göhringer, D. (2017). HW/SW Co-Design of the HOG algorithm on a Xilinx Zynq SoC. Journal of Parallel and Distributed Computing, 109, 50-62.

[59] Zhou, Y., Chen, Z., & Huang, X. (2016, May). A system-on-chip FPGA design for real-time traffic signal recognition system. In 2016 IEEE International Symposium on Circuits and Systems (ISCAS) (pp. 1778-1781). IEEE.

[60] Sossa, H., Arce, F., Zamora, E., & Guevara, E. (2018). Morphological neural networks with dendritic processing for pattern classification. In Advanced Topics on Computer Vision, Control and Robotics in Mechatronics (pp. 27-47). Springer, Cham.

[61] Krips, M., Lammert, T., & Kummert, A. (2002, January). FPGA implementation of a neural network for a real-time hand tracking system. In Proceedings First IEEE International Workshop on Electronic Design, Test and Applications' 2002 (pp. 313-317). IEEE.

[62] Nene, S. A., Nayar, S. K., & Murase, H. (1996). Columbia object image library (coil-20).

[63] Geusebroek, J. M., Burghouts, G. J., & Smeulders, A. W. (2005). The Amsterdam library of object images. International Journal of Computer Vision, 61(1), 103-112.