CLASIFICADOR COMPLETO DE CÉLULAS SANGUÍNEAS MEDIANTE UNA FPGA DE BAJO COSTE, MATLAB Y SIMULINK

DANIEL FARIÑA FERNÁNDEZ

MÁSTER EN INGENIERÍA INFORMÁTICA, FACULTAD DE INFORMÁTICA,
UNIVERSIDAD COMPLUTENSE DE MADRID

Trabajo Fin Máster en Ingeniería Informática

Curso 2016-2017

Convocatoria junio 2017

Calificación obtenida: 8.0

Directores.:
Alberto del Barrio García / Guillermo Botella Juan

Autorización de Difusión

DANIEL FARIÑA FERNÁNDEZ

19 de junio de 2017

El abajo firmante, matriculado en el Máster en Ingeniería Informática de la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Máster: "CLASIFICADOR COMPLETO DE CÉLULAS SANGUÍNEAS MEDIANTE UNA FPGA DE BAJO COSTE, MATLAB Y SIMULINK", realizado durante el curso académico 2016-2017 bajo la dirección de Alberto del Barrio García y Guillermo Botella Juan en el Departamento de Informática, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

Resumen en castellano

Actualmente la creación de algoritmos para FPGA tiene un auge considerable gracias a las prestaciones que presentan estos chips para el procesamiento de información. Para mejorar la experiencia en el desarrollo de hardware compañías como MathWorks ofrecen herramientas de síntesis de alto nivel para acelerar el proceso de diseño. En el presente trabajo se analizó el uso de Matlab y Simulink, así como toolbox, enfocados al desarrollo de hardware. Para este análisis se crearon dos casos de estudio para el reconocimiento de células en imágenes de microscopía utilizando redes neuronales. Para cada caso se implementaron modelos basados en el algoritmo DCT, pero empleando diferentes estrategias de extracción de características.

Una vez generados los modelos se utilizó el entorno de co-simulación FPGA in-the-loop ofrecido por Simulink para ejecutar los modelos directamente en la FPGA. Se analizaron y presentaron los resultados obtenidos por precisión y por tiempos de ejecución. Adicionalmente se realizan comparaciones entre los modelos generados y su contraparte en software con el fin de validar el grado de mejora obtenido en los tiempos de ejecución con uso de hardware para la aceleración de cómputo.

Palabras clave

FPGA, Desarrollo en hardware, Matlab, Simulink, Clasificación, Redes Neuronales, Células Sanguíneas.

Abstract

Developing algorithms for hardware, especially for FPGAs, is gaining popularity nowadays. This is thanks to the performance provided by these chips in the different areas. To improve the experience of hardware development, companies as MathWorks offer high-level synthesis tools to speed up the design process. In this work, we analyzed the use of Matlab and Simulink, as well as the hardware development toolbox. For this analysis, two different study cases were created. Both studies focused on the recognition of cells in microscopy images using neural networks. For each case, a model based on the DCT algorithm was implemented, but using different strategies for the feature extraction.

Once the models were generated, we used the FPGA in the loop environment offered by Simulink to run the models directly into the FPGA. The results obtained in terms of precision and the execution times were analyzed and presented. In addition, several comparisons were made between the generated models and their equivalent in software to validate the degree of achieved improvement.

Keywords

FPGA, Hardware Development, Matlab, Simulink, Classification, Neural Networks, Blood Cells.

Índice de contenidos

Índice de figuras

Índice de tablas

Listado de acrónimos

ALM	Adaptive Logic Module
ASIC	Application-specific integrated circuit
DCT	Discrete Cosine Transform
DFT	Discrete Fourier Transform
DSP	Digital Signal Processing
DUT	Device Under Test
FPGA	Field Programmable Gate Array
GPU	Graphic Processing Unit
HLD	Hardware Description Language
Kpps	Kilo Pixel Por Segundo
RNA	Redes Neuronales Artificiales
SIFT	Scale Invariant Feature Transform
SVM	Support Vector Machines

Agradecimientos

A mis directores de TFM Guillermo Botella y Alberto Del Barrio por mostrarme el camino del desarrollo en hardware para FPGA y enseñarme un nuevo campo en mi profesión.

Agradecimiento al Proyecto DESCARTES (DETECCION TEMPRANA DE DESARROLLO TUMORAL MEDIANTE COMPUTACIÓN DE ALTAS PRESTACIONES). Referencia: PR26/16-20B-1 (Santander- UCM).

A Angel Sierra (University Program) de MathWorks por otorgarme las licencias de Matlab necesarias para el desarrollo de todo el proyecto.

Capítulo 1 - Introducción

Actualmente existen diversos dispositivos como las GPUs que permiten la aceleración de cómputo, la mayoría enfocados en tareas de alto consumo como el renderizado de imágenes o la edición de video, sin embargo, éstas tienen un precio considerable y un consumo energético alto debido a la necesidad de una elevada velocidad de reloj y de memoria.

Por otra parte, se está popularizando otro tipo de componente para la aceleración de cómputo, como la Field-Programmable Gate Arrays (FPGAs). Estos dispositivos de hardware contienen una gran cantidad de recursos lógicos -más simples que los de una GPU- que permiten la ejecución eficiente de algoritmos enfocados en la ejecución paralela. Gracias a la arquitectura de este tipo de hardware, el consumo energético se ve considerablemente reducido, sin embargo, los recursos a la disposición del desarrollador son más básicos por lo que el coste de implementación de los algoritmos suele ser superior lo que redunda en una mayor curva de aprendizaje.

Existen diversas herramientas que permiten modelar algoritmos para su implementación en hardware específicas de cada placa como Quartus II de Altera o Vivado de Xilinx y otras más genéricas como Matlab y Simulink. Aunque éstas son herramientas de desarrollo general, poseen módulos que permiten desarrollar algoritmos en hardware sin tener conocimientos profundos en lenguajes HDL (Verilog o VHDL).

Con el fin de implementar algoritmos en hardware para la aceleración de cómputo, en este trabajo, se realizará una investigación para obtener cuáles son los algoritmos disponibles para la extracción de características de células en imágenes de microscopía y su posterior clasificación mediante redes neuronales.

Para la implementación se utilizará Matlab y Simulink con sus correspondientes paquetes para desarrollo con FPGAs. Una vez obtenida la implementación adecuada se realizará un estudio del rendimiento y coste de recursos del algoritmo, así como la dificultad de su implementación con la herramienta. En cuanto al hardware a utilizar será la FPGA Cyclone V presente en la placa Altera DE1-SoC y como entorno principal de desarrollo Matlab R2016b con Simulink.

1.1 Motivación

El reconocimiento y clasificación de células ha sido una necesidad latente en diferentes ámbitos científicos como la medicina o la biología. Aunque existen diversos algoritmos y herramientas para este fin, actualmente en la mayoría de los casos los biólogos siguen utilizando el método de inspección manual para efectuar este reconocimiento y clasificación celular, lo que genera una pérdida considerable de tiempo para el investigador. Por otro lado, la principal limitación del procesado automático se debe a la alta complejidad de los algoritmos y a los tiempos de respuesta con equipos convencionales. La aceleración eficiente de estos algoritmos puede generar un nuevo campo para la creación de nuevas aplicaciones en tiempo real que permita mejorar las respuestas en temas de reconocimiento y clasificación.

1.2 Objetivos

1.2.1 Objetivo general

El objetivo principal de este trabajo es validar Matlab y Simulink como un entorno eficaz para el desarrollo de algoritmos en hardware para FPGA. Como caso práctico se implementarán algoritmos de clasificación de células en sangre mediante la utilización de placa Altera DE1-SoC que posee una FPGA Cyclone V.

1.2.2 Objetivos específicos

- Implementar un algoritmo efectivo en hardware para la extracción de características en imágenes de células presentes en muestras de sangre para su posterior clasificación utilizando redes neuronales.

- Analizar la efectividad de Matlab y Simulink para el desarrollo efectivo de algoritmos en hardware sin realizar la implementación directamente en código HDL (Hardware Description Language).

- Analizar los algoritmos implementados con índices de efectividad en la clasificación de células en sangre.

- Probar la usabilidad de los toolboxes ofrecidos por la Matlab y Simulink para generar algoritmos complejos para hardware.

- Comparar el desempeño del algoritmo de clasificación implementado contra equipos convencionales empleando algoritmos similares con redes neuronales. Como referencia para la comparación se utilizará un portátil Asus N56jr con un procesador Core i7-4700HQ a 2.4Ghz y 16Gb DDR3 a 1600Mhz de memoria RAM.

Capítulo 2 - Estado del Arte

2.1 Clasificación Celular

Antes de implementar cualquier algoritmo es necesario conocer la naturaleza del problema a considerar, así como sus parámetros característicos. Como caso práctico de este proyecto, se buscará validar el entorno de desarrollo para hardware ofrecido Matlab y Simulink mediante la implementación de un clasificador de células sanguíneas.

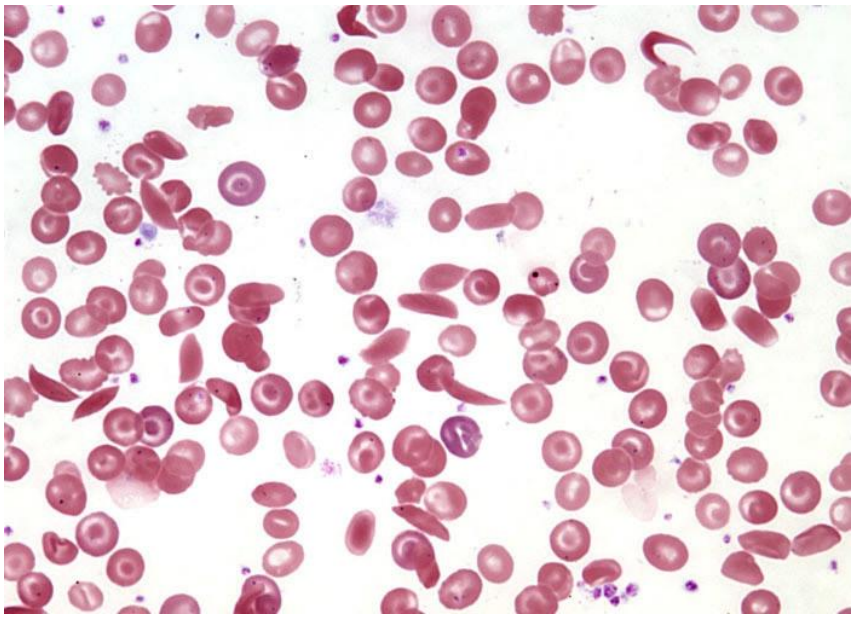
Con el objetivo de acotar el proyecto y reducir el amplio espectro de células clasificables, solo se emplearán dos grupos de imágenes de muestra con el fin de validar diferentes estrategias para la extracción de características. El primer grupo consiste en imágenes de microscopio de muestras sanguíneas, las cuales presentan diferentes tipos de células como trombocitos, hematíes con poiquilocitosis (anormalidades en la forma) y hematíes sanos. La clasificación y conteo de este tipo de células puede ayudar a detectar problemas en sangre como anemias hipocrómicas, cáncer metastásico óseo, síndrome mieloproliferativo (posibles causas de la poiquilocitosis), trombocitemia primaria (alto conteo de plaquetas) u otras enfermedades. El segundo grupo, contiene imágenes más simples de un único tipo de células tumorales.

Existen trabajos previos de clasificación de células en sangre, como el expuesto en 2016 por Loddo [1], en cual se logra la clasificación de células filtrando las imágenes por color, calculando el umbral mediante su histograma para posteriormente segmentar la imagen, y mediante un clasificador basado en Máquinas de soporte Vectorial (SVM) y Búsqueda de Vecinos Cercanos lograr la clasificación efectiva de las células.

En un trabajo más reciente Baker y Balhaf [2] exponen la clasificación de células blancas en imágenes de microscopía mediante el uso de algoritmo k-means, con especial interés en el uso de GPUs para acelerar los cálculos. En este trabajo se muestra un claro beneficio de la reducción de los tiempos de ejecución hasta un factor de tres al hacer uso de hardware gráfico.

Debido a que la placa Altera DE1-SoC a utilizar para el desarrollo es un modelo educacional de bajo costo y recursos, se buscará implementar algoritmos de baja complejidad que permitan un grado de clasificación de células aceptable.

Figura 2.1 Imagen de microscopía de muestra sanguínea[1]



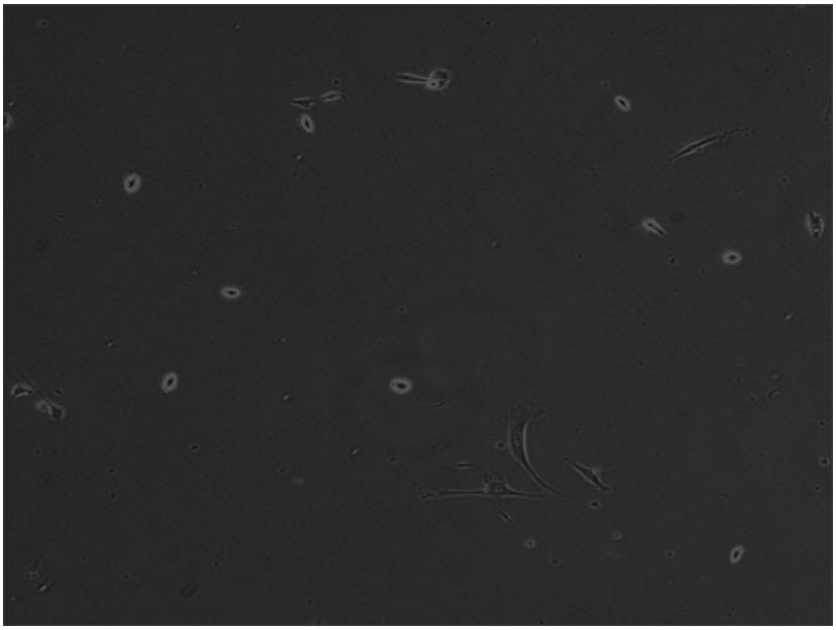
Debido a que se dispone de dos grupos de imágenes de naturaleza distinta, se propondrán 2 casos de estudio diferentes. La Figura 2.1 muestra un ejemplo de una imagen de microscopía de una muestra de sangre correspondiente al primer grupo. En ésta se puede observar que la imagen es en color y presenta variedad en los tipos de células. Para este grupo los algoritmos de clasificación se enfocarán en identificar los 3 tipos de células en concreto: hematíes sanos (glóbulos rojos sanos), hematíes con poiquilocitosis y trombocitos como los indicados en la Figura 2.2.

En el segundo grupo las imágenes de prueba se encuentran en escala de grises y contienen una mejor densidad de células. Estas solo poseen un tipo de célula y poseen características distintas a las del primer grupo. La Figura 2.3 muestra el ejemplo de una imagen del segundo grupo. Se puede observar la baja densidad celular y las características de estas células.

[1] Fuente: https://www.microscopyu.com/gallery-images/sickle-cell-anemia-at-40x-magnification

Figura 2.2 Hematíe sano, Hematíe con Poiquilocitosis y Trombocito

Figura 2.3 Imagen de ejemplo de células tumorales



2.2 Extracción de características

Las características de una imagen pueden definirse como funciones de una o más mediciones, la cual describe una propiedad cuantificable del objeto. Existen diversas aproximaciones tanto en software como en hardware para la extracción de características en imágenes, sin embargo, antes de realizar alguna implementación es necesario conocer cuál algoritmo se adapta mejor al problema planteado.

Según R. S. Choras [3], las características en imágenes pueden ser catalogadas de la siguiente manera:

- Características generales: son características independientes tanto del color, la textura o la forma, las cuales pueden subcatalogarse en características a nivel de píxel (contiene información única del píxel como el color y posición del píxel), características a nivel local

(funciones aplicadas a un segmento de la imagen), características a nivel global (funciones aplicadas a la imagen como un todo).

- Características de dominio específico: son características más complejas y conceptuales como por ejemplo un rostro humano, huellas dactilares, etc.

En este trabajo se realizará la implementación de algoritmos para la extracción de características a nivel de píxel o a nivel local. Entre las técnicas a nivel local más conocidas para la extracción de características, se pueden mencionar los siguientes: momentos invariantes, características por gradientes, características por densidad, DCT, características estadísticas, componentes de rotación invariante de energía espectral, estadísticas de matriz de coocurrencia, entre otros.

El trabajo presentado por Wiliem [4] se enfoca en la clasificación de células humanas en muestras de microscopía proponiendo 2 algoritmos para la extracción a bajo nivel de características en las imágenes: SIFT (Scale Invariant Feature Transform) y DCT (Discrete Cosine Transform).

Otro estudio realizado por A. Suyyagh y G. Abandah [5] sobre el reconocimiento de escritura arábiga demostró la efectividad de algunos de los algoritmos comentados anteriormente, arrojando el algoritmo DCT como el de mejor efectividad tanto en tiempo de ejecución como en errores en la predicción.

Debido a la complejidad de algunos de los algoritmos anteriormente mencionados y la efectividad del DCT para aportar características suficientemente discriminatorias en imágenes, se ha tomado la dedición se implementar dicho algoritmo para la extracción de características.

2.3 DCT

La transformada de coseno discreta, es una operación basada en la DFT (Discrete Fourier Transform), pero sólo actúa sobre funciones periódicas con simetría par y el resultado es una secuencia de números reales. Esta transformada expresa una secuencia finita de varios puntos como resultado de la suma de distintas señales cosenoidales en frecuencias múltiplo.

La DCT [5–9] se suele usar para representar este registro empleando las componentes espectrales más representativas de tal forma que la señal reconstruida aún tenga semejanza con la señal original. Actualmente es empleado para la compresión de múltiples formatos de imágenes y video como: DV, AC-3, JPEG, MJPEG, Vorbis, entre otros. La Figura 2.4 muestra el resultado de aplicar la DCT a la imagen, se observa como la mayoría de la información se concentra en la esquina superior izquierda del histograma.

Figura 2.4 Imagen de muestra con Histograma de DCT[2]

Aunque la función es ampliamente utilizada en diversos ámbitos, la implementación más común de este algoritmo suele utilizarse en 2 dimensiones y consiste en tomar una ventana de *NxN* píxeles que se desplaza por toda la imagen, a esta ventana se le aplica la función indicada en la Figura 2.4 y se obtienen *NxN* coeficientes de DCT que representan la mayor concentración de energía del bloque *NxN* de la imagen. Con una pequeña parte de estos coeficientes y la aplicación de una función inversa es posible la reconstrucción del bloque correspondiente de la imagen original. Gracias a que cada grupo de coeficientes representa a una porción de la imagen en particular es posible construir una red neuronal utilizándolos como características tal y como lo indica el estudio de Ashraf Suyyagh [5]. La Figura 2.5 muestra la fórmula general de la transformada del coseno discreto en 2 dimensiones:

Figura 2.5 Fórmula general de DCT 2D

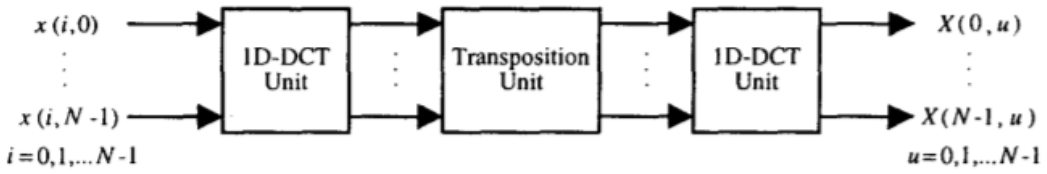
$$f(u,v) = \sqrt{\frac{2}{n}}\sqrt{\frac{2}{n}}\sum_{i=0}^{n-1} A(i) * \cos\left(\frac{u(2i+1)\pi}{2N}\right) * \sum_{j=0}^{M-1} A(j) * \cos\left(\frac{u(2j+1)\pi}{2M}\right) * f(j,i)$$

$$A(i) = \begin{cases} \frac{1}{\sqrt{2}}, & u = 0 \\ 1, & u \neq 0 \end{cases} \qquad A(j) = \begin{cases} \frac{1}{\sqrt{2}}, & u = 0 \\ 1, & u \neq 0 \end{cases}$$

[2] Fuente: https://es.wikipedia.org/wiki/Transformada_de_coseno_discreta

La principal razón de la elección del algoritmo DCT para la extracción de características en imágenes es la simplicidad de su implementación en hardware y su alta precisión para el posterior reconocimiento. Adicionalmente permite un alto grado de paralelismo que se traduce en mejores tiempos de ejecución. El paralelismo se extrae gracias a la separabilidad del algoritmo original en operaciones basadas en filas y columnas. Esto consiste en aplicar la DCT por columnas y al resultado de este aplicar el algoritmo por filas, como lo implementa Fernández [6] en su trabajo. En la Figura 2.6 muestra un esquema general para la aplicación de la DCT en 2 dimensiones.

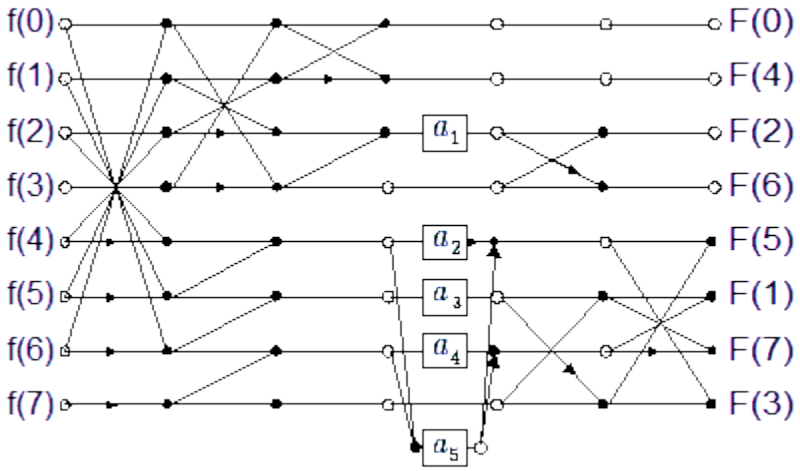
Figura 2.6 Esquema de aplicación de la DCT 2D en 2 fases [6]



El principal problema del uso la DCT para la extracción de características es que ésta se basa en funciones trigonométricas como el coseno. Dichas funciones no son sintetizables en la placa Altera DE1-SoC ya que dependen de tipos de datos con una precisión de punto flotante. Sin embargo, para poder sintetizar sobre esta placa es necesario conseguir aproximaciones que solo contengan operaciones sintetizables en hardware.

Arai et al. [7] publicaron en 1988 una de las primeras aproximaciones de la DCT. Esta aproximación está limitada únicamente a 8 parámetros de entrada, sin embargo, logran eliminar todas las operaciones trigonométricas y únicamente el algoritmo se basa en sumas, multiplicaciones y cambios de signo de los operandos. La Figura 2.7, muestra el esquema general de la aproximación.

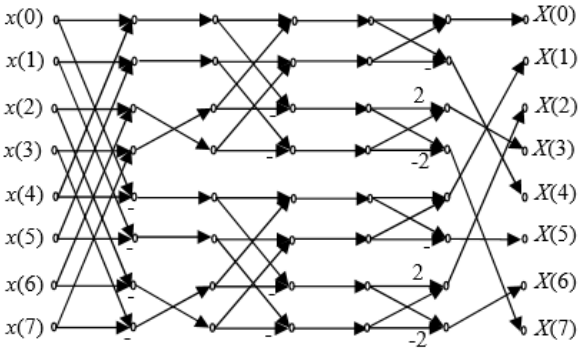
Figura 2.7 Implementación DCT de 8 parámetros de Arai [7]



Las flechas negras indican cambio de signo del operador, los puntos negros indican una operación de suma y las cajas indican una multiplicación, donde $a_1 \ldots a_5$ son parámetros constantes.

En otro trabajo presentado en 2010 por Bouguezel et al. [8] se realiza otra aproximación a la DCT. En este trabajo se muestran 4 implementaciones para los diferentes números de parámetros de entrada 2^N donde N=2...5. Solo se requiere 8 sumas y 2 desplazamientos de bit, 24 sumas y desplazamientos de bit, 64 sumas y 8 desplazamientos de bit, y 160 sumas 16 desplazamientos de bit para aplicar el algoritmo a secuencias de entrada 4, 8, 16 y 32 respectivamente. A continuación, en las Figuras 2.8 y 2.9 se muestran los diagramas de la implementación del DCT de 8 y 16 puntos respectivamente presente en el trabajo de Bouguezel.

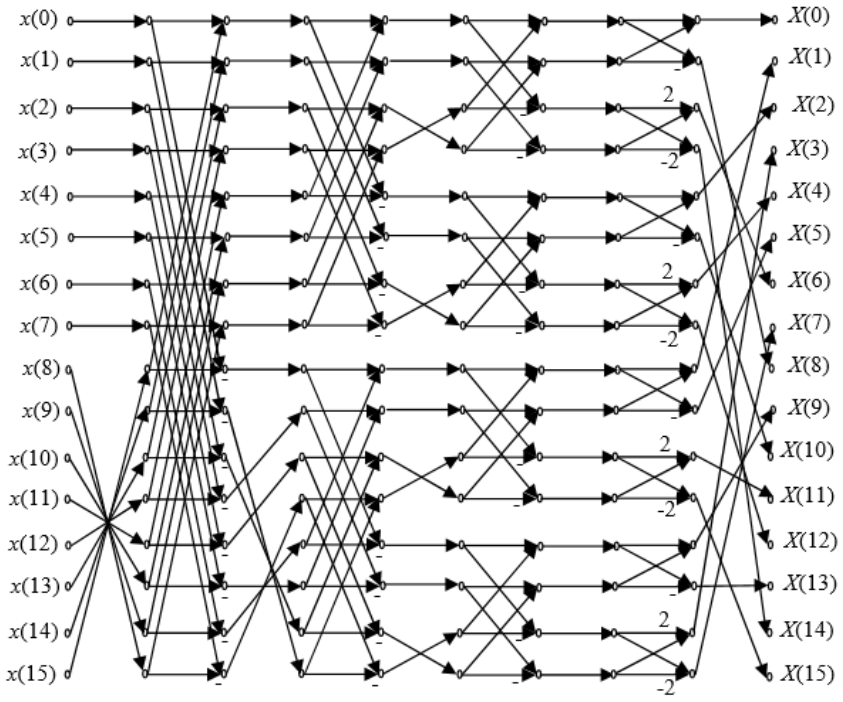
Figura 2.8 Implementación DCT de 8 parámetros Bouguezel [8]



Como se ha observado, existen diversos trabajos enfocados a encontrar una aproximación bastante buena de dicha transformada y la forma de conocer su efectividad es comparando su

PSNR (Relación Señal a Ruido). El PSNR es una expresión que define la relación entre el posible valor máximo de una señal y la distorsión causada por el ruido que afecta a su representación. Uno de los usos habituales de esta métrica es como medida cuantitativa de la calidad de la reconstrucción en el ámbito de la compresión de imágenes.

Figura 2.9 Implementación DCT de 16 parámetros Bouguezel [8]



Para calcular el PSNR de la señal, basta con aplicar el algoritmo en cuestión sobre una determinada imagen y posteriormente a cada grupo de coeficientes se le aplica la inversa del algoritmo para obtener una aproximación a la imagen original. Con esta nueva imagen obtenida y la imagen original se puede aplicar el PSNR que viene obtenido por la siguiente expresión:

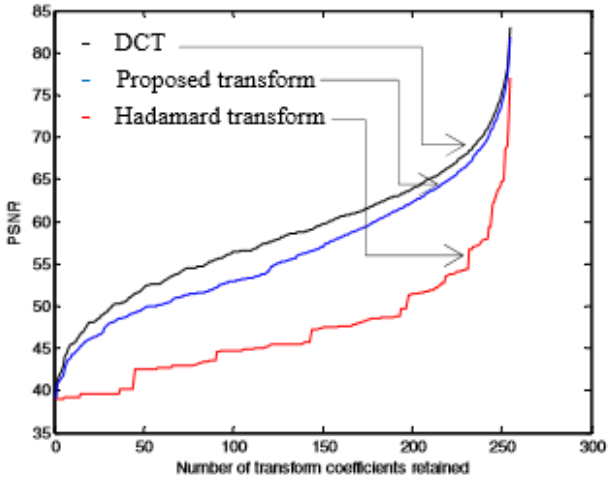
$$PSNR = 20\log([\text{Fondo de la escala} / \text{MSE}])$$

Donde "Fondo de escala" indica el mayor valor de pico que se obtiene (a modo de ejemplo con 8 bits de precisión, tendríamos 255) y MSE indica el valor cuadrático medio de la diferencia de la señala original y la obtenida.

Aunque varía dependiendo de su implementación, los valores típicos que adopta este parámetro están entre 30 y 50 dB, siendo mayor cuanto mejor es la codificación.

Bouguezel en su trabajo también aporta comparativas del PSNR obtenido de su aproximación de DCT con la función real y contra la transformada de Hadamard, observándose una diferencia de entre 2 y 5 dB cuando se utilizan entre 50 y 200 coeficientes. La Figura 2.10 muestra la gráfica de la comparativa del PSNR.

Figura 2.10 PSNR de DCT, Hadamard y la aproximación de Bouguezel [8]



Finalmente, para este trabajo se decide utilizar ambas aproximaciones de la DCT presentadas anteriormente. Se procederá a la validación de qué aproximación ofrece mejor precisión en su implementación, con los recursos disponibles en la placa objeto de este TFM.

2.4 Matlab y Simulink para el desarrollo de Hardware

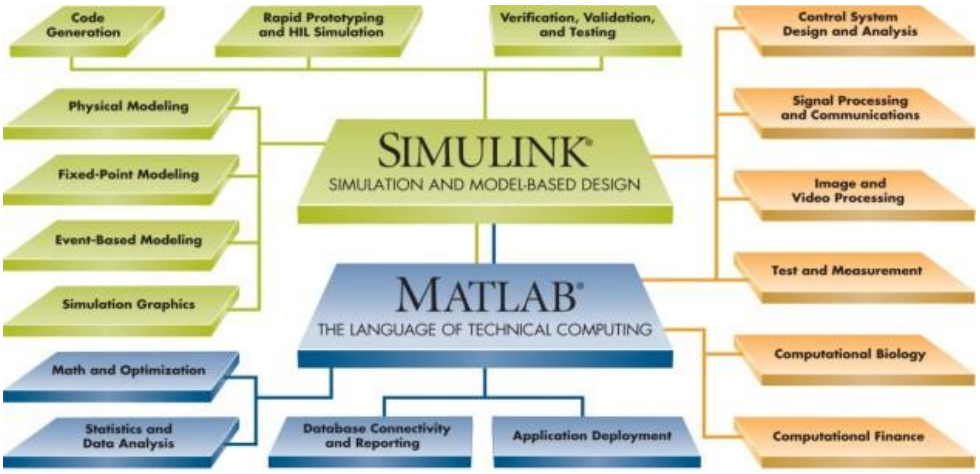
2.4.1 Matlab

Matlab (Matrix Laboratory) es un entorno de computación numérica multiparadigma, desarrollado por la empresa MathWorks. El entorno fue desarrollado principalmente para la manipulación de datos numéricos en matrices, mostrar gráficos de datos e implementación de funciones para validación de algoritmos. Actualmente cuenta con múltiples interfaces para diferentes lenguajes de programación como C/C++, Java, Fortran o Python entre otros.

Matlab también ofrece una serie de paquetes adicionales, comercializados como toolbox que incluye múltiples funcionalidades y componentes hacen que la herramienta sea usada actualmente en ámbitos como ingeniería, economía, investigación académica, así como el área industrial. Dispone de un potente lenguaje de programación propio -el cual también recibe el

nombre de la herramienta- orientado a capacidad de computación simbólica, creación de interfaces gráficas y muchas más opciones lo cual lo convierten en una herramienta básica en muchos procesos de ingeniería. La Figura 2.11 muestra el esquema general de Matlab y, así como sus diferentes utilidades y componentes.

Figura 2.11 Esquema general Simulink/Matlab[3]



2.4.2 Simulink

Simulink es un componente añadido a Matlab, desarrollado también por MathWorks, que proporciona un entorno de programación gráfico para el modelado, la simulación y el análisis de sistemas dinámicos multidominio.

Por sus características, Simulink es ampliamente usado en ámbitos como la ingeniería automática, el procesamiento digital de señales o el diseño basado en modelos. Además de ofrecer los bloques estándar ofrecidos por la herramienta, también existen en la comunidad infinidad de colecciones de bloques tanto de la propia Simulink como de empresas externas que permiten extender la funcionalidad de Simulink hacia nuevos dominios, como puede ser por ejemplo la electrónica analógica, el modelado de máquinas eléctricas. Otra de las características a destacar de Simulink, es que ofrece la generación automática de código C, VHDL o Verilog para implementaciones en tiempo real. El origen de esta herramienta proviene de la necesidad de realizar simulaciones de modelos de forma más rápida. Simulink también permite convertir el

[3] Fuente: http://innovate.dreamcatcher.asia/page_02mw.html

modelo o parte de él a un binario ejecutable sobre un sistema operativo propio de MathWorks, conocido como xPC Target.

Simulink dispone de diferentes herramientas que ayudan en la validación y verificación del modelo que se esté desarrollando. De esta forma, Simulink es capaz de detectar problemas de división por cero, desbordamiento de números enteros, análisis de números en coma fija, etc.

Para este proyecto se utilizaron varios toolbox no incluidos en la versión por defecto de Matlab, cada uno contiene componentes especializados en diversas tareas del proyecto. A continuación, se muestra una pequeña descripción de cada uno de los toolbox utilizados en el proyecto.

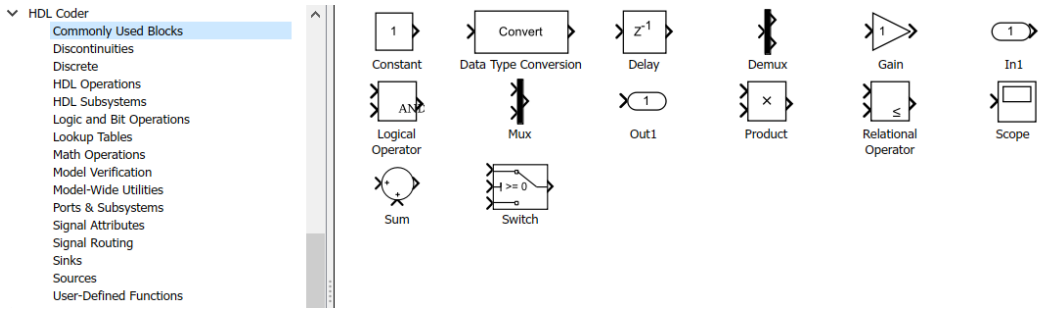
2.4.3 HDL Coder

El HDL Coder permite realizar las transformaciones y compilaciones de los modelos de Simulink, funciones definidas en Matlab y Máquinas de estados a código Verilog y VHDL. Este código generado puede ser utilizado por la herramienta específica de la FPGA para la compilación a un binario ejecutable por la placa.

El HDL Coder provee de un asistente que automatiza la creación y configuración necesaria de modelos para compilar en placas FPGA de marcas como Xilinx o Altera FPGAs. También permite controlar la arquitectura e implementación del código HDL, indicando el camino crítico del modelo, así como ofrece generar un reporte de estimación de los recursos necesarios para compilar el proyecto a una placa en particular [10].

Un aspecto importante a resaltar es que ofrece trazabilidad del código, es decir, que permite visualizar el código VHDL o Verilog generado relacionándolo con la parte del modelo diseñado. Para este proyecto se utilizó principalmente los bloques de este toolbox, ya que la herramienta permite la compilación de estos a código HDL de manera directa. En la Figura 2.12 muestra un ejemplo de las operaciones y bloques disponibles en el HDL coder Toolbox organizadas por sus diferentes categorías.

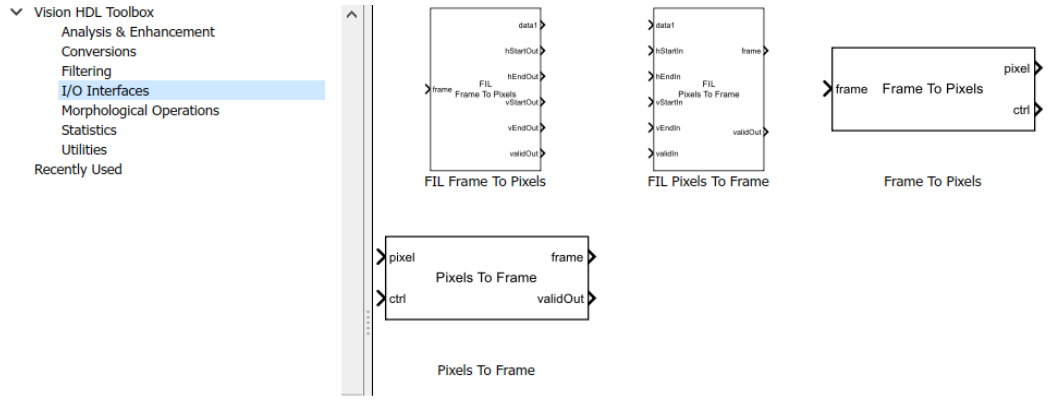
Figura 2.12 HDL Coder Toolbox



2.4.4 Vision HDL Toolbox

El Vision HDL Toolbox provee de componentes básicos para realizar diseño de algoritmos que impliquen manipulación de píxeles, específicamente para su uso en placas FPGA. El framework de diseño que ofrece soporta diferentes interfaces, tipos de datos, extensiones de ficheros y tamaños de imagen y video que incluyen alta definición de 1080p. Todos los componentes incluidos en el toolbox utilizan una arquitectura apropiada para su síntesis a HDL por lo que no requieren modificaciones adicionales para realizar su compilación. La Figura 2.13 muestra las categorías y los bloques disponibles del Vision HDL Toolbox [11].

Figura 2.13 Vision HDL toolbox



2.4.5 Neural Network Toolbox

El Neural Network Toolbox provee algoritmos, modelos de redes neuronales pre entrenadas, y aplicaciones para crear, entrenar, visualizar y simular redes neuronales. Ofrece la posibilidad de crear diferentes tipos de redes para clasificación, regresión, clustering, reducción de dimensiones, previsión de series de tiempo, sistemas de control y modelado dinámico.

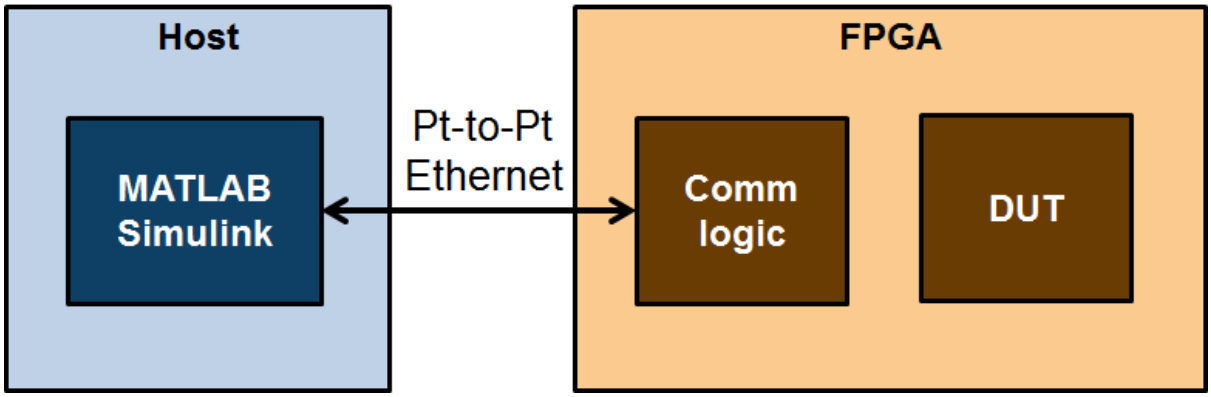
Las redes de aprendizaje profundo incluyen redes neuronales convolucionales y decodificadores para la clasificación, regresión y aprendizaje de características en imágenes. Para pequeños grupos de muestra, permite aplicar entrenamiento profundo para realizar la transferencia del aprendizaje a redes neuronales pre-entrenadas. Para aumentar la velocidad del entrenamiento de redes neuronales Matlab ofrece otro toolbox denominado Parallel Computing Toolbox, el cual permite distribuir los cálculos y la información de entrenamiento a los diferentes procesadores multinúcleo y las GPUs disponibles en el sistema [12]. Aunque Matlab permite la creación de redes específicas para clasificación de imágenes, éstas no son fácilmente sintetizables a HDL.

2.4.6 HDL Verifier

El HDL Verifier permite generar automáticamente bancos de prueba para la verificación de diseños sintetizados desde lenguajes de alto nivel de descripción de hardware como Verilog o VHDL. Es posible utilizar Matlab o Simulink directamente para ejecutar una simulación del modelo y en una etapa anterior analizar los resultados del modelo utilizando la co-simulación con HDL o el paradigma FPGA in-the-loop con las placas Xilinx y Altera. HDL Verifier también permite crear componentes generados por modelos en Matlab y Simulink para ejecutarse de forma nativa en simuladores de otras compañías especializadas en circuitos integrados complejos como Cadence, Mentor Graphics y Synopsys [13].

Además de emplear de los elementos disponibles en los diferentes toolbox, también se utilizó el FPGA in-the-loop del HDL Verifier. Este modo de co-simulación permite ejecutar los bloques del modelo previamente compilado directamente en la FPGA, sin la necesidad de realizar configuraciones en herramientas externas. Adicionalmente permite alimentar la placa con la información y los parámetros de entrada establecidos en el modelo y recibir la información procesada por ésta. Es decir, permite establecer un ciclo de Host-FPGA-Host de la información, empleando la FPGA para la aceleración de los cálculos que emplean más tiempo en el modelo. En la Figura 2.14 se muestra un diagrama general de la interconexión entre la FPGA y el anfitrión. El *Comm logic* es un componente que incluye Matlab al diseño durante la compilación de los bloques a código HDL. Este permite la comunicación de los datos entre la placa y Matlab. El *DUT* (Device Under Test) corresponde al diseño sintetizado dentro de la FPGA.

Figura 2.14 Conexión FPGA in-the-loop[4]



2.5 Redes Neuronales

Una Red Neuronal Artificial (RNA) se define como un modelo matemático que se inspira en el comportamiento biológico de las neuronas y en cómo se interconectan para realizar el paso de impulsos nerviosos [15].

En 1943 el psiquiatra y neuro-anatomista Warren McCulloch y el matemático Walter Pitts realizan el primer modelo matemático de una neurona artificial, creado con el fin de llevar a cabo tareas simples. Las características principales de las RNA son las siguientes:

- Procesado no lineal: ofrece la capacidad de realizar aproximaciones a funciones para las clasificaciones de patrones lo que aumenta su inmunidad frente al ruido.
- Autoorganización y adaptabilidad: se emplean algoritmos de aprendizaje adaptativo, por lo que ofrecen mejores posibilidades de procesado robusto y adaptativo.
- Procesado paralelo: por lo general se utiliza un gran número de nodos generando una red con un alto nivel de interconectividad. El elemento básico de computo se le llama habitualmente nodo o neurona. Esta recibe una entrada desde otras neuronas o de una fuente externa de datos. Cada nodo tiene un peso asociado *w*, que se modifica en el proceso de aprendizaje. Cada neurona aplica una función de normalización, típicamente una sigmoide, sobre la suma ponderada de los parámetros de entrada. La Figura 2.15 muestra la fórmula de la suma ponderada de los pesos. En esta fórmula *w* representa los pesos definidos en la red e *y* los valores de entrada del nodo.

[4] Fuente: https://es.mathworks.com/help/hdlverifier/ug/fpga-in-the-loop-fil-simulation.html

Figura 2.15 Suma pondera de los pesos en RNA

$$y_i = \sum_j w_{ij} y_j$$

Las características de las RNA juegan un importante rol en el procesado de señales e imágenes, ya que permiten utilizar arquitecturas complejas que emplean elementos de procesado adaptativo paralelo, combinados con estructuras de interconexiones jerárquicas.

La creación de un modelo de red neuronal consiste de 2 fases:

- Fase de entrenamiento: se emplean conjuntos de datos o patrones de entrenamiento para determinar los pesos que definen el modelo de red neuronal. Se calculan de manera iterativa, de acuerdo con los valores de entrenamiento, con el fin de minimizar el error cometido entre la salida obtenida por la red neuronal y la salida deseada.
- Fase de prueba: al finalizar la fase previa, el modelo puede encontrarse demasiado ajustado a las particularidades presentes en los patrones de entrenamiento, lo que ocasiona una perdida en su habilidad de generalizar su aprendizaje a casos nuevos (sobreajuste). Para evitar el problema del sobreajuste, se aconseja utilizar un segundo grupo de datos diferentes a los de entrenamiento que permita controlar el proceso de aprendizaje.

Existen diversos tipos de redes neuronales, estas generalmente se clasifican según sus algoritmos de entrenamiento como: redes de pesos fijos, redes no supervisadas, y redes de entrenamiento supervisado [16].

- Para las redes de pesos fijos no existe ningún tipo de entrenamiento, simplemente se indican los pesos de cada neurona de acuerdo a un criterio particular.
- Las redes neuronales de entrenamiento supervisado son las más utilizadas, en estas redes, los datos para el entrenamiento están constituidos por varios pares de patrones de entrenamiento de entrada y de salida. El hecho de conocer la validez del dato de salida implica que el entrenamiento se beneficia de la supervisión. En este tipo de redes dado un nuevo patrón de entrenamiento genera una nueva adaptación de los pesos, como lo indica la Figura 2.16. En la fórmula *w* representa los pesos para un nodo *i, j* en una iteración de entrenamiento *m*, el peso de la

siguiente iteración es calculado con los pesos de la iteración actual más un delta de ese mismo peso

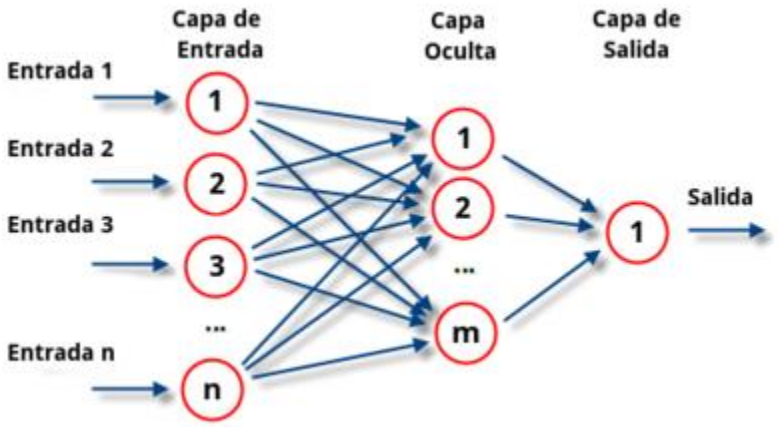
- Para los modelos de entrenamiento no supervisado, el conjunto de datos de entrenamiento consiste únicamente de los patrones de entrada.

Figura 2.16 Cálculo de pesos de redes de entrenamiento supervisado

$$w_{ij}^{m+1} = w_{ij}^{m} + \Delta w_{ij}^{m}$$

Un sistema típico de entrenamiento no supervisado son la Regla de Aprendizaje de Hebb, y la Regla de Aprendizaje Competitivo. El aprendizaje de Hebb consiste en reforzar el peso que conecta a dos nodos que se estimulan en paralelo. Para el aprendizaje competitivo, si un patrón nuevo pertenece a una clase reconocida previamente, entonces la inclusión de este nuevo patrón a esta clase matizará la representación de la misma. En el caso de que el nuevo patrón no pertenezca a ninguna de las clases reconocidas anteriormente, entonces la estructura y los pesos de la red neuronal se modificarán para reconocer a la nueva clase. En la Figura 2.17 se muestra la estructura general de una red neuronal.

Figura 2.17 Estructura de una red neuronal de 3 capas[5]

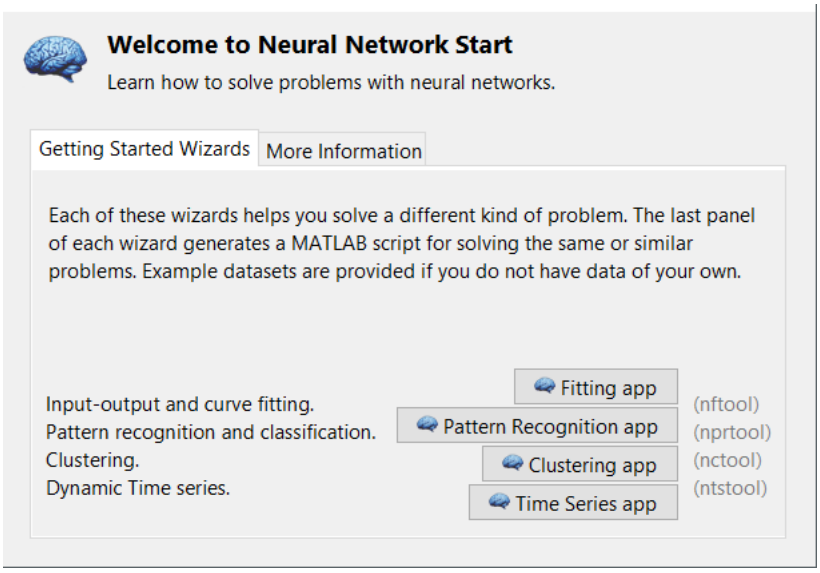


Matlab ofrece diferentes tipos de arquitecturas de redes neuronales, cada una enfocada a resolver un problema distinto, desde redes para problemas de adaptación que buscan ajustar entre

[5] Fuente: https://medium.com/espanol/avances-en-redes-neuronales-705c2efe53d2

un conjunto de datos de entradas numéricas y un conjunto de objetivos numéricos; pasando por redes para clustering que buscan crear conjuntos de datos basados en su similitud y por último las redes de tiempo dinámico, en el que se utilizan valores anteriores de una o más series temporales para predecir valores futuros. En la Figura 2.18 se muestra el asistente de Matlab para la creación de redes neuronales, en ésta se observan las diferentes arquitecturas de redes que ofrece la aplicación.

Figura 2.18 Asistente de Matlab para la creación de redes neuronales[6]



En el caso particular del proyecto se decide utilizar la denominada “Pattern Recognition and Classification”, este tipo de red supervisada está enfocada directamente a la clasificación de patrones, con un tipo de entrenamiento offline.

La red neuronal seleccionada se crea alimentando al constructor de la red con una cantidad *N* de muestras que contienen una cantidad *M* de patrones. Se indica para cada muestra a qué clase final pertenecen.

2.6 FPGA

Aunque las FPGAs se comercializan desde 1985, actualmente la mayoría de los sistemas informáticos desde mediados de los años 90 cuentan con procesadores basados en la arquitectura

[6] Fuente: https://es.mathworks.com/help/nnet/gs/classify-patterns-with-a-neural-network.html

de Intel x86. Con los años estos procesadores han aumentado sus prestaciones incrementando el número de núcleos, así como la frecuencia del reloj, sin embargo, el consumo energético ha aumentado o se ha mantenido. Las FPGAs son chips de silicio reprogramables, estos utilizan bloques de lógica preconstruidos y recursos de interconexión programables, que permiten configurarlos en un orden deseado para implementar soluciones en hardware.

A pesar de ser soluciones no tan extendidas como los procesadores de propósito general, hoy en día las FPGA están empezando a ganar notoriedad por su versatilidad y flexibilidad, siendo principalmente usada en campos como sistemas de imágenes médicas, reconocimiento de voz, codificación y cifrado, entre otros. El incremento en su popularidad ha logrado que recientemente Intel, el principal fabricante de procesadores, adquiriera a Altera, una de las empresas dominantes del mercado de las FPGA, y ha logrado incorporar la arquitectura de las FPGAs de Altera en algunos de sus chips [17-18].

Antiguamente estas placas eran empleadas únicamente por ingenieros con conocimientos en diseño de hardware digital debido a la complejidad de su programación. Sin embargo, gracias al auge de múltiples herramientas de diseño de alto nivel la tendencia está cambiando, y la programación de FPGAs se hace más sencilla gracias a nuevas tecnologías que convierten los diagramas a bloques gráficos, o hasta el código ANSI C a circuitos de hardware digital.

La adopción de las FPGAs en la industria ha sido impulsada por el hecho de que los FPGAs combinan lo mejor de los ASICs y de los sistemas basados en procesadores, es decir, ofrecen velocidades temporizadas por hardware y fiabilidad, pero sin requerir altos volúmenes de recursos para compensar el gran gasto que genera un diseño personalizado de ASIC.

La arquitectura de este chip reprogramable tiene la misma capacidad de ajustarse que un software que se ejecuta en un sistema basado en procesadores, pero no está limitado por el número de núcleos disponibles. A diferencia de los procesadores, las FPGAs llevan a cabo diferentes operaciones de manera paralela, por lo que éstas no necesitan competir por los mismos recursos. Cada tarea de procesos independientes se asigna a una sección dedicada del chip, y puede ejecutarse de manera autónoma sin ser afectada por otros bloques de lógica. Como resultado, el rendimiento de una parte de la aplicación no se ve afectado cuando se agregan otros procesos.

Las FPGAs se programan con lenguajes HDL. Este tipo de lenguajes de programación están especializados en describir la estructura y el comportamiento de circuitos electrónicos y más comúnmente circuitos lógicos digitales. Al igual que los lenguajes de programación para software concurrentes, la semántica y sintaxis de los lenguajes HDL incluyen notaciones explicitas para expresar la concurrencia, sin embargo, a diferencia de los primeros, los HDL también incluyen nociones explícitas de tiempo.

Aunque existan similitudes entre los lenguajes de programación de software y los HDL, existe una diferencia muy marcada en cuanto a su funcionamiento, por ejemplo, muchos lenguajes de programación en software están limitados por la concurrencia que aportan los hilos de un procesador (y el sistema operativo), por otra parte, los HDL ofrecen un alto grado de paralelismo ya que estos definen la estructura del procesador en función del algoritmo. Ambos tipos de lenguajes de programación pasan por un proceso de compilado (usualmente denominado sintetizado en lenguajes HDL). En HDL esto significa la transformación de una colección de código a una lista redes de puertas lógicas físicamente realizable. Por otra parte, la compilación en software convierte el código fuente en código específico para su ejecución en el microprocesador. En cuanto al nivel de abstracción, los lenguajes HDL son comparados al lenguaje ensamblador de máquina, es decir, a un muy bajo nivel lo que dificulta el desarrollo de algoritmos complejos. Entre los lenguajes HDL de más amplio uso destacan VHDL y Verilog. Altera ofrece soporte para ambos lenguajes en su conjunto de herramientas para diseño de hardware denominado Quartus II.

Existen diversas compañías enfocadas en mejorar la experiencia de creación de algoritmos en hardware mediante la implementación de herramientas de síntesis de alto nivel [9]. Este nuevo tipo de sintetizadores buscan compilar algoritmos creados en lenguajes como C/C++ a código HDL. En el caso de Matlab y su herramienta de simulación Simulink, ofrecen una colección de operaciones básicas en bloques interconectables para representar los algoritmos.

Como este proyecto tiene como objetivo validar la eficacia de Matlab y Simulink para la implementación de hardware, todo el desarrollo se realizará mediante la utilización de los bloques que ofrecen las herramientas de Matlab. Los bloques utilizados para el modelado en Simulink, se traducen a código HDL, sin embargo, se desconoce el nivel de optimización que estos posean dejando para un trabajo futuro el análisis del código compilado.

En cuanto al hardware, en este proyecto se desarrollarán los algoritmos para la placa Altera DE1-SoC. Dicha placa contiene la FPGA Cyclone V de Altera, Debe notarse que al ser una placa con propósito educacional orientada a bajo coste y bajo consumo cuenta con menores recursos de los que se pudiesen conseguir en una placa comercial con chips basados en Stratix o Arria, por ejemplo

Tabla 2.1 Características Placa Altera DE1-SoC

FPGA	Cyclone V SoC 5CSEMA5F31C6
Puertas lógicas Programables	85000
Memoria embebida	4,450 Kbits
SDRAM Para FPGA	64MB (32Mx16)

En la Tabla 2.1 se muestra las características principales de la placa FPGA. se puede observar que cuenta con 85000 elementos programables, 4,450 Kbits de memoria embebida y 64MB de SDRAM. Adicionalmente también cuenta con un procesador ARM Cortex-A9 de 2 núcleos embebido dentro del procesador de la placa. Para este proyecto, solo se desarrollarán algoritmos para la FPGA de la placa.

La arquitectura desarrollada por Altera se basa en Módulos Lógicos Adaptativos (ALM, Adaptative Logic Module), este tipo de módulos es la unidad básica utilizada para el desarrollo de diseños en la familia de placas con los siguientes chips: Serie Arria, Cyclone V, Stratix IV, y Stratix V. Cada ALM puede soportar hasta 8 parámetros de entrada y 8 valores de salida, contiene dos o cuatro celdas de registros lógicos, y dos celdas para combinación lógica, dos sumadores completos dedicados, una cadena de transporte, una cadena de registro y una máscara de 64 bits para tablas de búsqueda [19].

Con un solo ALM pueden implementar las siguientes funciones:

- Dos funciones independientes con 4 parámetros de entrada.
- Una función de 5 parámetros de entrada y una función de 3 parámetros.
- Una función de 5 parámetros de entrada y una de función de 4, si comparten al menos un parámetro.

- Dos funciones de 5 parámetros, si comparten 2 parámetros.
- Una función de 6 parámetros de entrada.
- Dos funciones de 6 parámetros de entrada, si comparten 4 parámetros y es la misma función.
- Una función de 7 parámetros de entrada.

Aunque la información suministrada por Altera sobre la arquitectura de la FPGA ofrece una idea general de los recursos que puede utilizar cada componente de un modelo, la realidad es que los cálculos del uso de estos recursos difieren mucho al momento de la síntesis del modelo, probablemente se deba a optimizaciones automáticas que realiza la herramienta sobre sectores de la FPGA. La Figura 2.19 muestra los recursos disponibles de la FPGA, para este proyecto se utiliza el modelo 5CSEMA5F31C6, que corresponde a la columna A5, se observa la cantidad de bloques ALM disponibles, elementos lógicos y registros, sin embargo, es interesante resaltar la cantidad de multiplicadores y de bloques DSP disponibles, ya que es la principal limitante en los algoritmos a desarrollar.

Figura 2.19 Características FPGA Cyclone V Familia SE [20]

Resource		Member Code			
		A2	A4	A5	A6
Logic Elements (LE) (K)		25	40	85	110
ALM		9,434	15,094	32,075	41,509
Register		37,736	60,376	128,300	166,036
Memory (Kb)	M10K	1,400	2,700	3,970	5,570
	MLAB	138	231	480	621
Variable-precision DSP Block		36	84	87	112
18 x 18 Multiplier		72	168	174	224
FPGA PLL		5	5	6	6
HPS PLL		3	3	3	3
FPGA GPIO		145	145	288	288
HPS I/O		181	181	181	181
LVDS	Transmitter	32	32	72	72
	Receiver	37	37	72	72
FPGA Hard Memory Controller		1	1	1	1
HPS Hard Memory Controller		1	1	1	1
ARM Cortex-A9 MPCore Processor		Single- or dual-core	Single- or dual-core	Single- or dual-core	Single- or dual-core

Para las operaciones complejas, Altera ofrece otro tipo de elementos denominados bloques DSP (Digital Signal Processing). Estos bloques permiten sintetizar operaciones como multiplicaciones, divisiones, etc., sin perder rendimiento. Ofrecen tres tipos de modo de uso dependiendo de su precisión, el "Standard-precision mode" ofrece una precisión de 18-bit de punto fijo para multiplicadores, el "High-precision mode" hasta 27 bits de precisión, y el "Floating-point mode" [21].

Este último permite a la FPGAs utilizar de forma nativa operadores de punto flotante, eliminando la necesidad de hacer una conversión del modelo a punto fijo en cada síntesis. Adicionalmente cada bloque DSP permite efectuar más de una multiplicación en paralelo si la precisión utilizada es menor del máximo según su modo. En la Tabla 2.2 se muestran cuantas multiplicaciones en paralelo se pueden realizar utilizando 1, 2 ó 4 bloques DSP.

Tabla 2.2 Cantidad de multiplicadores según precisión y Bloques DSP en Cyclone V

Within 1 DSP Block	
Quantity	**Multiplier Mode**
3	9x9
2	12x12
2	16x16
2	18x19
1	18x25
1	27x27
1	18x36

Within 2 DSP Block	
Quantity	**Multiplier Mode**
1	18x18 complex multiply
1	36x36 complex multiply
Within 4 DSP Block	
1	18x25 complex multiply
1	18x36 complex multiply
1	27x27 complex multiply
1	54x54 complex multiply

Capítulo 3 - Metodología

En esta sección, se describe en detalle la metodología utilizada para el reconocimiento y clasificación de células, el tipo de imágenes de entrada utilizadas, implementación en Matlab y Simulink.

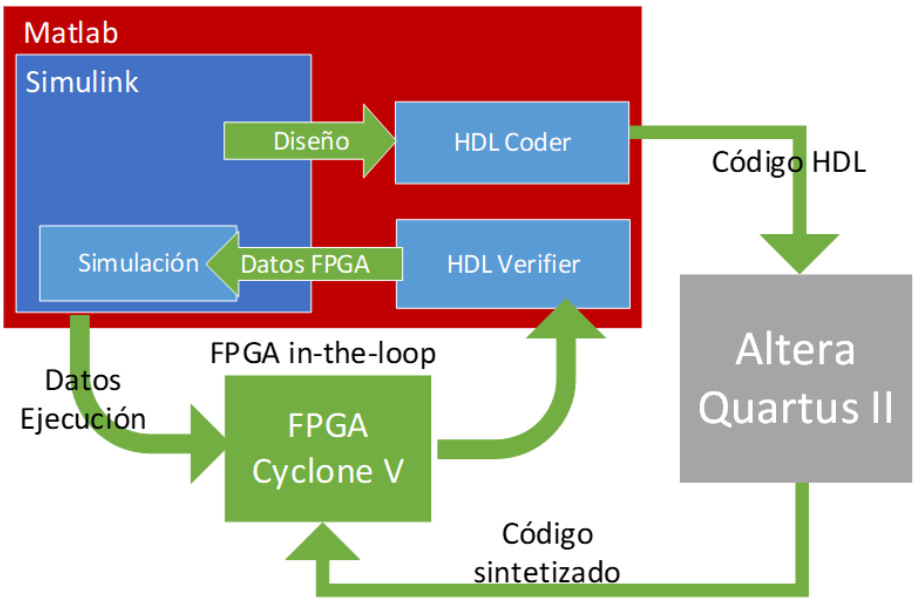
3.1 Fases del proyecto

La primera etapa del proyecto consistió en analizar cuáles son las posibles características extraíbles de las imágenes ejemplo. Estas imágenes consisten en muestras de microscopio de células. Para facilitar el uso de las imágenes con Matlab se ha decidido normalizar el tamaño de estas imágenes, reduciéndose a imágenes de 640x480 píxeles. En estas imágenes se observa que las células presentan un tamaño aproximado de 16x16 píxeles por lo que se toma este tamaño como patrón de objetivo para la clasificación.

Durante la etapa de investigación se analizaron diferentes estudios para identificar cuáles son los algoritmos más idóneos para realizar la extracción de características en imágenes. Una vez obtenido el algoritmo más adecuado tomando en cuenta las limitaciones causadas por los recursos de la FPGA, fue necesario validar su viabilidad de implementación en hardware. Este estudio consistió en analizar las diferentes operaciones realizadas en el algoritmo comprobando su disponibilidad en Matlab R2016 y con el compilador a HDL (ejemplo: funciones como coseno, arco tangente, raíces, etc.).

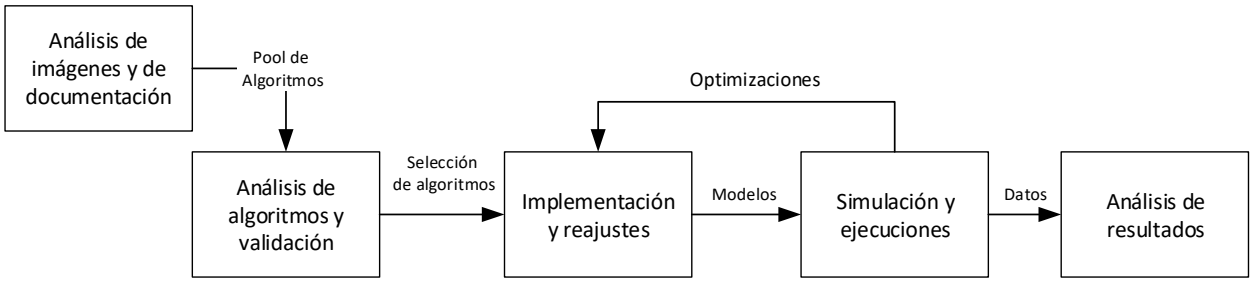
En la etapa de implementación se utilizaron todos los recursos disponibles de Matlab y Simulink para su desarrollo, teniendo a disposición los diferentes toolbox para el manejo de imágenes, así como los correspondientes para el desarrollo en FPGAs. Hay que mencionar que, aunque Matlab permite realizar el diseño del algoritmo, la compilación y el despliegue en la placa se realiza a través del software propietario de Altera, Quartus II. La Figura 3.1 muestra el framework de trabajo utilizado en el proyecto, se observa la interconexión de las diferentes herramientas, así como los datos e información que se intercambian.

Figura 3.1 Framework utilizado en el proyecto



Por último, se realiza el análisis de los resultados obtenidos de las ejecuciones realizadas, así como el coste de implementación de los algoritmos mediante el uso de Matlab. En la Figura 3.2 se muestra la metodología empleada para la realización de este proyecto.

Figura 3.2 Metodología del proyecto



3.2 Análisis de imágenes

En esta sección se realizará una breve descripción del análisis realizado a las imágenes de muestra utilizadas en el proyecto.

La Figura 3.3 muestra una de las imágenes objetivo del proyecto que corresponde al grupo de imágenes de células tumorales. Se observa que la imagen se encuentra en escala de grises y presenta una marcada diferencia en los bloques de la imagen que tienen y las que no tienen células. Las células a clasificar resaltan en la imagen por poseer una forma aproximada a un círculo con

un núcleo obscuro. Por otro lado, en Figura 3.4 se observa una imagen de microscopía muy diferente a la anterior, además de encontrase en color se observan más elementos clasificables y con una densidad mayor. En este último escenario se muestran diferentes tipos de células, con diferente forma y tonalidad, sin embargo, se comparte un tamaño similar.

Figura 3.3 Imagen de muestra de células tumorales

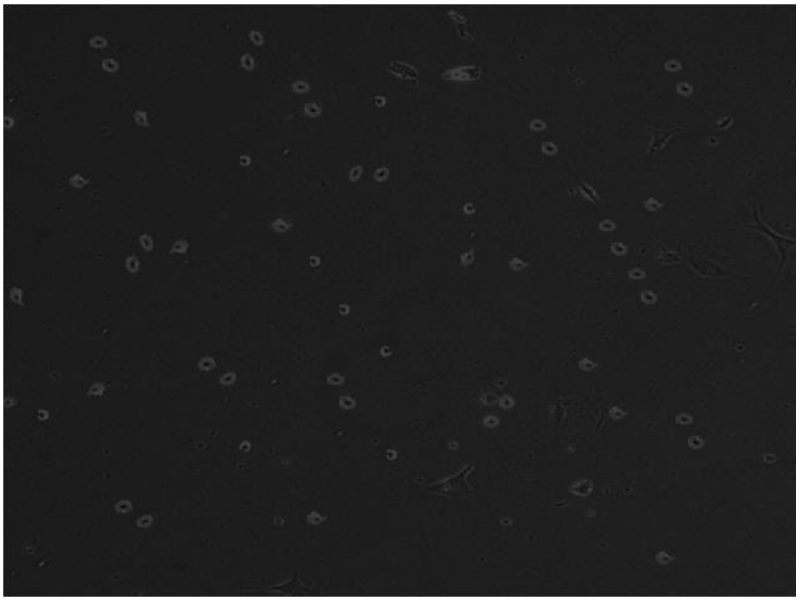
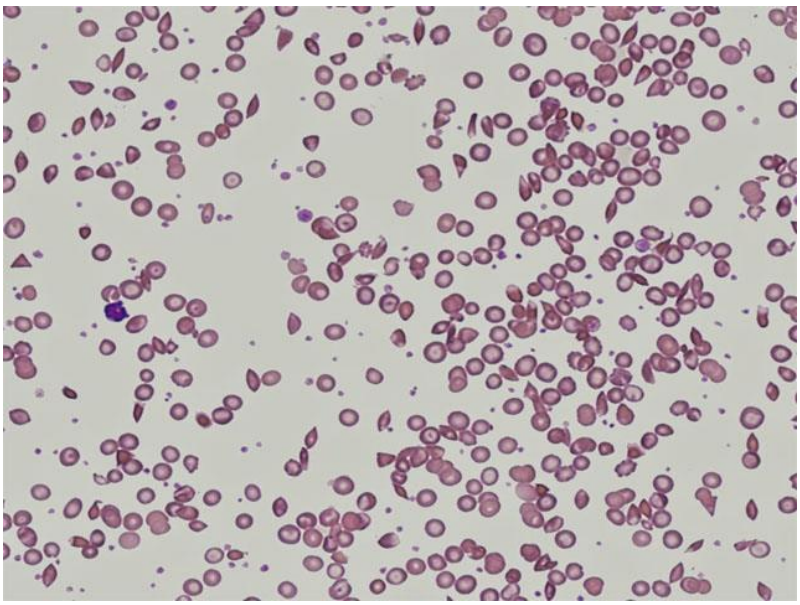


Figura 3.4 Imagen de muestra de células sanguíneas



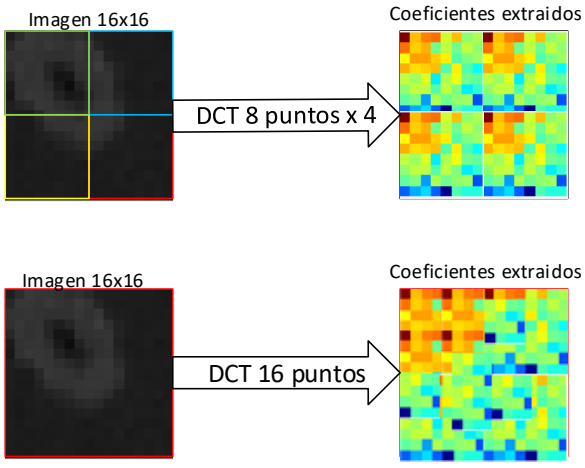
Al tratarse imágenes muy diferentes en cuanto a contenido y forma, se ha decido tratar el proyecto de clasificación de células en ambas imágenes como dos casos de estudio diferentes.

Se observa que para ambas imágenes las células presentes muestran un tamaño aproximado de 16x16 píxeles y aprovechando que se cuenta con algoritmos de DCT para 8 y 16 puntos de entrada, se decide realizar dos estrategias diferentes para la extracción de sus características basadas en el mismo algoritmo DCT. Utilizando dicha medida en píxeles como tamaño objetivo de bloques de imágenes objetivo a clasificar.

Estas implementaciones consistirán en la aplicación del algoritmo DCT a bloques de la imagen, sin embargo, la primera aproximación extraerá las características de la imagen mediante el algoritmo de DCT de 8 puntos que posteriormente los agrupará en bloques para su posterior procesado. Esta aproximación es la utilizada por Fariña [14] para la clasificación celular en su investigación.

La otra implementación extraerá las características aplicando la DCT directamente en bloque de la imagen de 16x16. Esto permitirá evaluar, si una agrupación de coeficientes de DCT obtenido utilizando el algoritmo de 8 puntos, es capaz de extraer características para la clasificación como una de 16 puntos. La Figura 3.5 muestra una idea general de ambas estrategias a implementar para la extracción de características.

Figura 3.5 Estrategias para la extracción de características

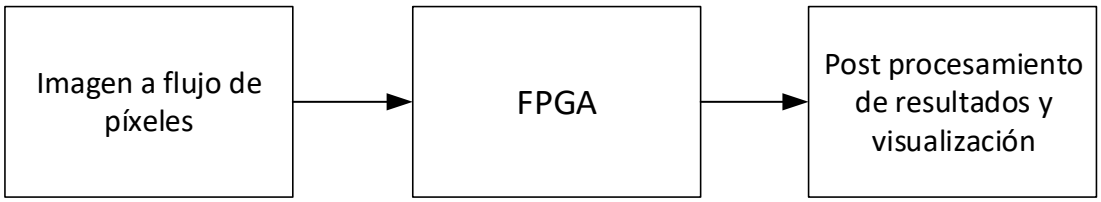


3.3 Esquema general

Uno de los principales beneficios del uso de Matlab y Simulink es que permite diseñar el modelo en hardware y en la misma herramienta desarrollar su contraparte en software para realizar las comparaciones pertinentes. En la sección de anexos se muestran en detalle las capturas de pantalla de la implementación realizada para ambos casos de estudio.

Con el fin de validar la efectividad del DCT para la detección y clasificación de células se decide realizar la implementación de 2 diferentes modelos en Simulink. Un modelo para cada caso de estudio. Estos modelos seguirán las estrategias descritas en el apartado anterior por lo que cada uno empleará una aproximación de la DCT (de 8 y de 16 puntos) diferente, así como utilizarán grupos de imágenes de prueba diferentes. La Figura 3.6 muestra el diagrama general de algoritmos de ambos modelos. La imagen muestra las operaciones realizadas por el host, así como la transformación de la imagen a flujo de píxeles y la recepción y procesado de los resultados de la FPGA.

Figura 3.6 Diagrama general de los modelos



3.4 Envío y recepción de datos de la FPGA

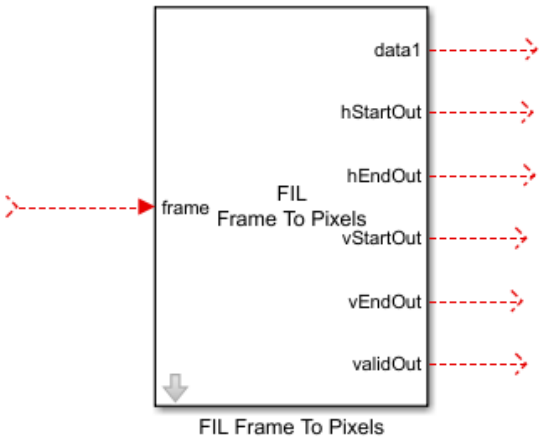
Para poder realizar el envío de la imagen a la placa, es necesario realizar una transformación de ésta a algo que pueda ser interpretado fácilmente por la FPGA, como por ejemplo un flujo de números enteros.

Para realizar esta transformación se utilizaron los bloques pertenecientes al HDL Vision toolbox, específicamente el bloque de Fil Frame to Pixels. Este permite transformar una matriz *NxM* en valores enteros de 8 bits de precisión. Estos valores son discretizados y enviados uno a uno a la FPGA. La Figura 3.7 muestra un ejemplo del bloque Fil Frame to Pixels, indicando cuales son los parámetros de entrada y de salida.

Además de realizar la transformación de la matriz a flujo de pixeles, este bloque permite indicar el modo de envió de los datos a la FPGA. Estos modos corresponden a la cantidad de píxeles que se envían por vez, permitiendo enviar píxel a píxel, una línea completa de pixeles de la imagen, o la imagen completa. Eso mejora los tiempos de procesamiento por la FPGA ya que es ésta la encarga de gestionar el consumo de los datos del buffer de entrada.

Para poder llevar un control sobre que píxel se está recibiendo en la FPGA, también es necesario enviar 5 valores booleanos por cada píxel. Cada valor representa una posición particular del píxel, por ejemplo, si éste es el primer píxel de la imagen (vStart), o si es el último (vEnd), así como indicar el píxel inicio (hStart) y fin de cada línea (hEnd), o indicar si es un píxel valido de la imagen (valid).

Figura 3.7 FIL Frame to Pixels



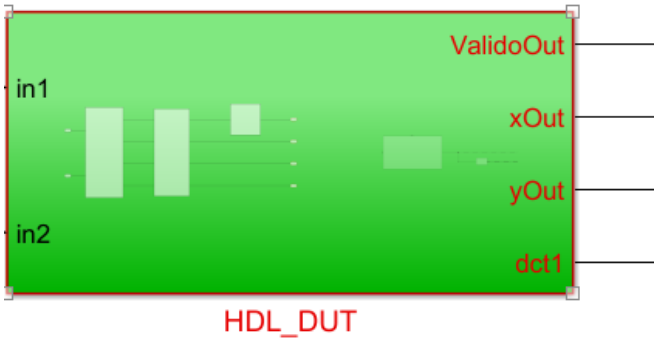
Para el caso de las imágenes en color, es necesario realizar una transformación adicional para llevarlas a escala de grises, ya que este tipo de imágenes cuentan con hasta 3 matrices, cada una correspondiente a la componente RGB.

En cuanto a la recepción por parte del anfitrión de los datos procesados por la FPGA, para ambos modelos también se emplea la misma estrategia, con la única diferencia de la cantidad de datos arrojados por la red neuronal. Los modelos cuentan con 4 valores de salida de los cuales 3 son compartidos por ambos modelos, identificados como “validoOut”, “xOut” y “yOut”. Xout yOut son valores enteros de 16 bits de precisión que corresponden a la ubicación del bloque de la imagen procesado. ValidoOut es un valor booleano que indica que la información retornada por el resto de los parámetros corresponde a información valida. Este mecanismo de notificación es necesario, ya que el anfitrión no es capaz de diferenciar cuando la FPGA envía información valiosa, porque esta se encuentra muestreando constantemente la placa.

Por último, el cuarto parámetro de salida llamado “DCT1” corresponde a la respuesta de la red neuronal, ésta varía entre los modelos y consiste en un array valores de precisión de punto fijo (2 valores para el primer modelo, 3 valores para el segundo modelo o estrategia) que varían entre 0 y 1. Cada valor en el array corresponde a una clase en particular, y cuando un valor toma un máximo de un umbral definido, se considera que el bloque de la imagen pertenece a una clase especifica. La Figura 3.8 se muestran los 4 parámetros de salida del bloque que se sintetizará en la FPGA.

El envío y recepción de información hacia y desde la placa se realiza a través del puerto USB del anfitrión y de un único USB tipo B de la placa. Cabe destacar que este proceso de transferencia de información añade un coste en tiempos de ejecución considerable. Para comprobar los tiempos que tarda el envío y recepción de datos a la FPGA, se creará un modelo adicional que no realice ninguna operación, y que contenga los mismos parámetros de entrada y de salida que los modelos a desarrollar.

Figura 3.8 Bloque de sintetizado a FPGA



3.5 FPGA, caso de estudio 1: imágenes en escala de grises + 2 clases

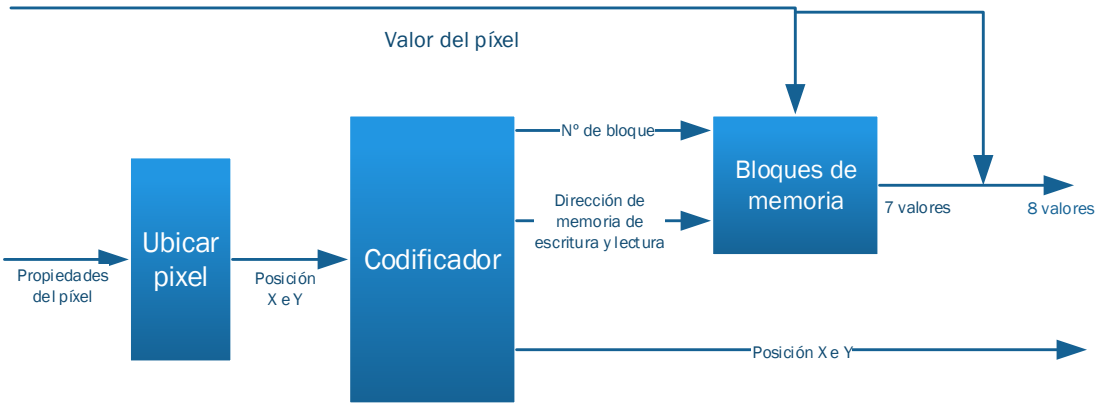
Para el primer caso de estudio, el modelo desarrollado tenía la finalidad de comprobar la validez para extracción de características del algoritmo DCT con 8 parámetros de entrada en la FPGA, así como realizar una primera aproximación con el entorno Matlab y Simulink. Con el fin de realizar diferentes pruebas, se emplearon las imágenes de microscopía en escala de grises en las que se encontraban presentes un único tipo de célula (corresponden a la Figura 3.3).

La primera operación en el modelo es la transformación de la imagen a un flujo de píxeles con las que se alimentará directamente a la FPGA (como lo indicado en la sección 3.4). Dentro de la FPGA un controlador de memoria realiza el almacenamiento por fila de los píxeles de la imagen.

El controlador de memoria diseñado al efecto consta de 3 etapas, la primera encontrar la ubicación del píxel de entrada, la segunda almacenar el píxel de entrada y la tercera extraer un grupo de 7 píxeles de memoria. La etapa de ubicación consiste en utilizar los parámetros de propiedades del píxel (hStart, hEnd, vStart, vEnd, valid) para calcular las coordenadas *X* e *Y* del píxel entrante. Los valores obtenidos junto con el valor del píxel son utilizados en la etapa de almacenamiento.

Se calcula la dirección de memoria en donde se guardará el valor del píxel utilizando la coordenada *X* y con el valor coordenada *Y* se identifica el bloque de memoria. Una vez se almacenan 7 filas (cada fila corresponde a un bloque de memoria), en la etapa de extracción se obtienen los 7 valores de los píxeles almacenados utilizando la coordenada *X* del primer píxel de la octava línea como dirección de memoria de lectura de todos los bloques de memoria. Los 7 píxeles extraídos más el píxel de la entrada se utilizarán como parámetros de entrada para la siguiente etapa del modelo. La Figura 3.9 muestra una visión general de la estructura del controlador de memoria. En esta se pueden apreciar los parámetros de entrada y de salida de cada operación.

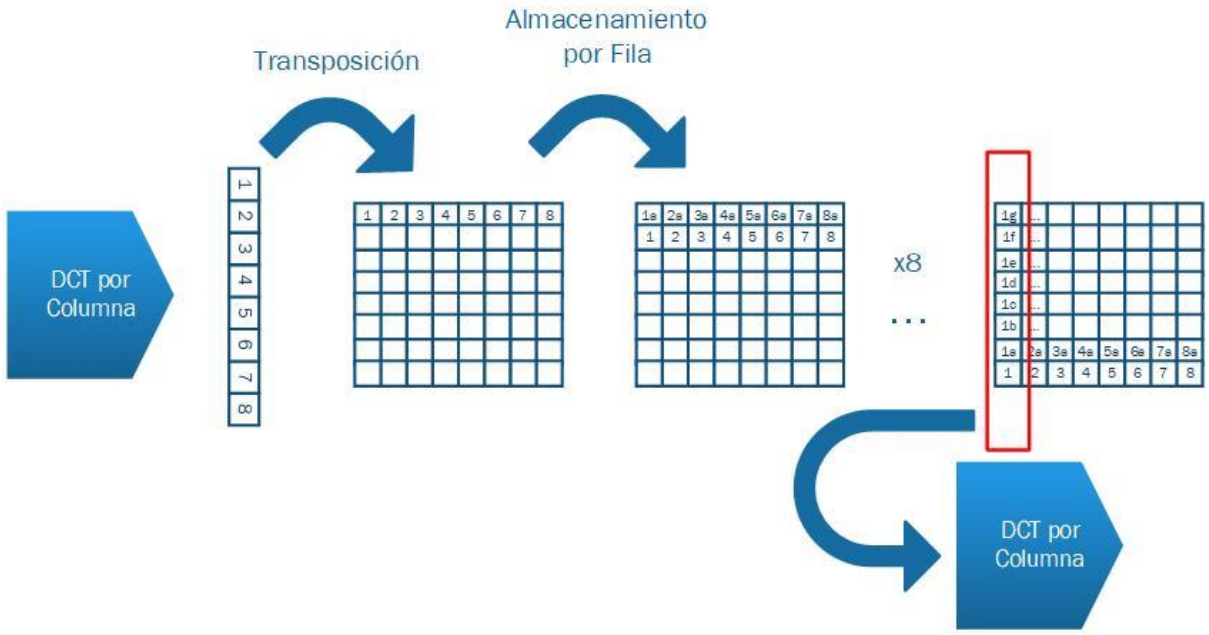
Figura 3.9 Controlador de memoria



3.5.1 2D-DCT de 8 puntos

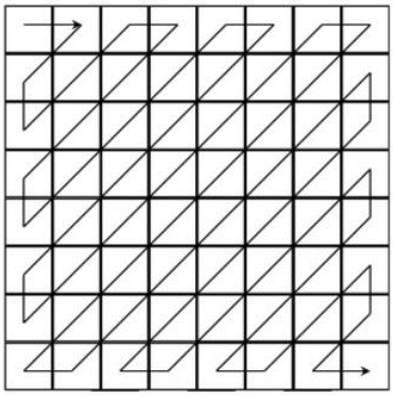
El siguiente paso consiste en efectuar la DCT con los 8 píxeles extraídos de la etapa anterior. Para esta aproximación de extracción de características en imágenes se implementó el DCT de 8 parámetros de entrada presentado por Arai [7]. Esta implementación consiste en sumar, restar y multiplicar los parámetros de entrada en un orden determinado hasta obtener el coeficiente correspondiente a cada pixel.

Figura 3.10 Transposición de los coeficientes del DCT



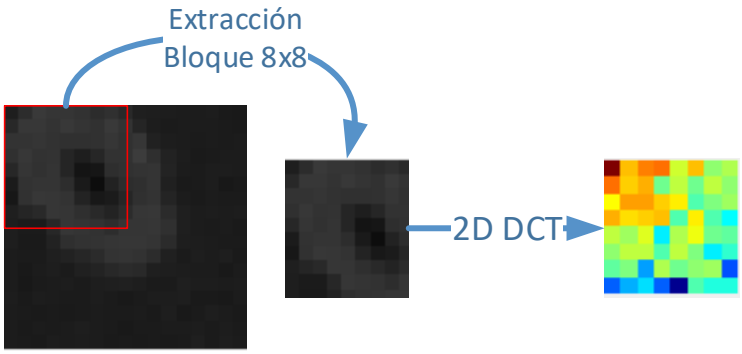
Una vez obtenidos los primeros 8 coeficientes se transponen y se acumulan en memoria hasta que se realizan otras 8 ejecuciones de la DCT. Este proceso se realiza hasta obtener 8 columnas transpuestas en memoria. Después de que los 8x8 coeficientes se encuentren en memoria se efectúa nuevamente la DCT esta vez utilizando como parámetros de entrada las columnas transpuestas. La Figura 3.10 muestra el proceso de transposición de los coeficientes de DCT.

Figura 3.11 Extracción de coeficientes en zigzag



Una vez se obtiene la matriz 8x8 de coeficientes DCT se extraen los primeros 6 valores de la matriz siguiendo un patrón de zigzag como indica la Figura 3.11. Se extrae este número de coeficientes por cada bloque de imagen con el fin de minimizar los parámetros de entrada de la red neuronal. Este número se obtuvo analizando las matrices arrojadas por la DCT 2D. La Figura 3.12 muestra un ejemplo del análisis realizado, en el que se extrajo un bloque de 8x8 de una sección de la imagen que contenía una célula y se aplicó el algoritmo DCT 2D. El resultado que se obtuvo fue que los valores más significativos se encontraban en el cuadrante superior izquierdo de la matriz.

Figura 3.12 Matriz de coeficientes de un bloque de 8x8



3.5.2 Agrupación de coeficientes

La etapa de agrupación consistió en almacenar los coeficientes obtenidos de la etapa anterior, y extraerlos de manera agrupada para conseguir los coeficientes de la DCT correspondientes a una imagen de 16x16.

Este agrupador emplea una estrategia similar a la utilizada en el controlador de memoria inicial. Se utiliza la coordenada *X* del ultimo píxel del bloque como dirección de memoria tanto de escritura como de lectura. Una vez que se almacena una línea de coeficientes, al obtenerse los coeficientes de la siguiente línea de bloques se extraen de memoria los ya almacenados y se agrupan formando conjuntos de 2 filas y 2 columnas de coeficientes, lo que corresponde a 24 características (2x2x6=24). Estos coeficientes se utilizan como parámetros de entrada para la red neuronal.

3.5.3 Red neuronal de 2 clases

Esta etapa consiste en utilizar los coeficientes de la etapa anterior para la clasificación de los bloques de imágenes utilizando la red neuronal. El proceso de creación de la red neuronal puede consultarse en el apartado 3.7

La red neuronal consiste de 3 etapas. En la primera etapa se realiza una normalización de los 24 valores de entrada. Esta normalización consiste en restar cada parámetro de entrada por sus valores mínimos, al valor obtenido se multiplica por un valor constante precalculado (2 dividido por el valor máximo que puede alcanzar el parámetro menos el valor mínimo que puede alcanzar el mismo parámetro) y por último se le resta 1 a cada valor. Esta normalización genera que los parámetros de entrada sean llevados a un rango definido entre -1 y 1. Esta función está definida en la siguiente expresión:

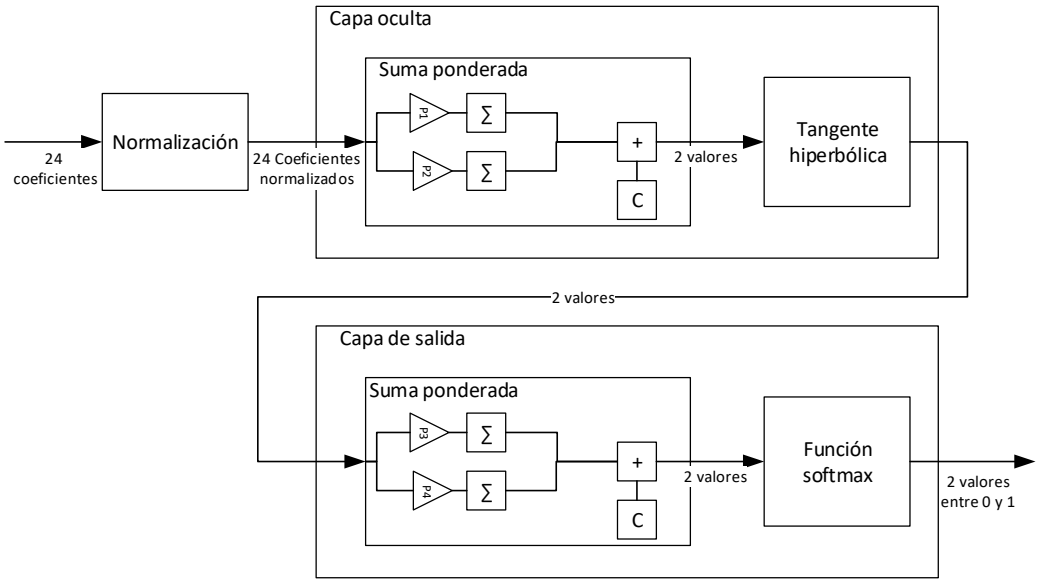
$$norm(x_j) = (x_j - min(x)) * \left(\frac{2}{\max(x) - \min(x)}\right) - 1$$

La siguiente operación corresponde a la capa oculta de la red neuronal. Consiste en la multiplicación de los parámetros obtenidos en la normalización por los pesos correspondientes de cada nodo, para este modelo se realizaron pruebas con 2 y 4 nodos. Cada nodo posee 24 valores constantes que se multiplicarán por los 24 valores normalizados obtenidos, posteriormente esos 24 valores se sumarán hasta obtener un único valor por cada nodo. A estos valores se les vuelve a sumar valores constantes. A continuación, estos valores son utilizados como parámetros de entrada de la función de activación sigmoidea, en este caso corresponde a la función tangente hiperbólica.

Con los valores obtenidos de la capa oculta de la red se alimenta la capa de salida de la red neuronal. Esta capa consiste nuevamente en la suma ponderada de los parámetros de entrada

usando pesos precalculados. Para esta capa solo se utilizan 2 nodos, cada uno correspondiente a un tipo de clase. El primer nodo de la capa de salida corresponde a la clase de las imágenes que contiene células tumorales, la segunda clase corresponde al fondo la imagen. Después de estas multiplicaciones y sumas de los pesos, los valores se suman nuevamente a constantes, para después ser utilizados como parámetros de entrada de la función SoftMax La función permite normalizar nuevamente los valores entre 0 y 1. La Figura 3.13 se muestra el esquema general de la red neuronal del modelo 1, en esta se puede apreciar los parámetros de entrada de cada operación, así como las operaciones que se realizan por cada capa.

Figura 3.13 Esquema red neuronal modelo 1



3.5.4 Recursos utilizados y optimizaciones

A continuación, se especificará cuáles son los recursos necesarios en hardware para la implementación del modelo 1 (operaciones matemáticas y memoria) utilizados durante el proyecto y las posteriores optimizaciones realizadas al modelo.

La primera etapa del controlador de memoria requirió al menos 7 bloques de RAM de al menos 640 (ancho de la imagen) segmentos disponibles. La implementación del algoritmo DCT por columnas consta de 27 sumas, 11 multiplicaciones y 6 desplazamientos de bits. La aplicación de la DCT por filas requiere de 37 sumas, 11 multiplicaciones y 8 desplazamientos de bits. Cabe destacar que para aprovechar el paralelismo del hardware se realiza la DCT por filas de manera

simultánea al obtener las 8 columnas por lo que se multiplica por 8 el coste total de la DCT por filas.

La etapa del agrupado de los coeficientes del DCT, se necesita almacenar al menos 6 filas de coeficientes de la DCT del tamaño de la imagen por lo que se requieren 6 bloques de RAM del ancho de la imagen.

La primera fase de la red neuronal correspondiente a la normalización de los valores requiere consta de 4 sumas, 1 multiplicación por cada característica de entrada lo que da un total de 96 sumas y 24 multiplicaciones. La segunda Fase de la red neuronal denominada “Capa oculta” en esta se realizan la multiplicación de los pesos de los nodos ocultos por los valores normalizados tiene un coste de 24 multiplicaciones y 48 sumas por cada nodo. La aplicación de la representación de la tangente hiperbólica como función de activación requiere de 5 sumas y 7 multiplicaciones por cada nodo.

En la última capa de la red neuronal tiene un costo de *N* multiplicaciones por cada nodo y *2xN* Sumas siendo *N* el número de nodos. Por último, la aplicación de la función softmax dentro de la última capa tiene un coste de 2 multiplicaciones y 2 sumas. También debe tomarse en cuenta el espacio en memoria que ocupa la lookup table de esta última función.

Utilizando 2 nodos para la red neuronal los recursos necesarios totales del diseño representan en hardware unas 782 operaciones aritméticas y un poco más de 9000 registros de memoria RAM repartidos en 14 bloques. Aunque los resultados no representan un porcentaje alto de los recursos del sistema existe otro factor que impidió la compilación del modelo a hardware.

La FPGA utilizada para el desarrollo cuenta con un número limitado de 87 bloques DSP; las multiplicaciones especificadas en el diseño donde los operandos tengan un tamaño superior a los 18 Bits son transformadas durante la compilación a bloques DSP. Teniendo 187 multiplicaciones en el diseño (con un máximo permitido de 87) fue necesario revisar lo implementado para realizar optimizaciones.

El principal problema del desarrollo en hardware para este proyecto ha sido la escasez de recursos que posee la FPGA utilizada, específicamente en el apartado de multiplicadores ya que si se utilizan algoritmos que empleen muchas operaciones de multiplicación y con una alta precisión es probable que el modelo no pueda ser sintetizado para la FPGA en cuestión.

Este modelo sin optimizaciones utiliza los recursos presentados en la Tabla 3.1. A estos se le deben incluir los bloques de memoria RAM para el almacenamiento de los píxeles tanto en la etapa del DCT como en el agrupado de características, así como los valores de la tabla de búsqueda utilizada en la función exponencial.

Tabla 3.1 Operaciones utilizadas modelo 1

Algoritmo	Sumas	Mult.	<<
DCT por columnas	27	11	6
DCT por filas	37 x 8	11 x 8	8 x 8
Normalización	96	24	0
Capa 1 RNA.	34	62	0
Capa 2 RNA.	4	2	0
Total de operaciones	457	187	70

Las siguientes optimizaciones fueron realizadas con el fin de reducir los recursos necesarios para compilar el diseño. En una primera revisión del diseño, se pudo eliminar un grupo considerable de operaciones, reduciendo a 3 el número de bloques de la DCT por filas utilizados, ya que solo se extraen 6 valores de las primeras 3 filas, obteniendo 55 multiplicaciones menos.

Por otro lado, Simulink proporciona herramientas para optimizar los bloques de operaciones de manera individual, permitiendo serializar las operaciones similares. Sin embargo, esto incurre en un incremento en los tiempos de respuesta. Esta optimización fue utilizada en el bloque de DCT por filas, logrando reducir a solo 22 multiplicaciones para todo el algoritmo DCT.

Junto con las optimizaciones anteriores, se realizaron mejoras revisando los bits necesarios para cada entrada y salida de las operaciones, reduciendo cuando fuese posible el número de bits de los valores resultantes por cada bloque. Esto permite una mejor utilización de los bloques DSP, ya que la arquitectura de FPGA de Altera permite hasta 3 multiplicaciones de 9x9 bits en un solo bloque DSP. Esta disminución de los bits utilizados en los parámetros de las operaciones permite que el diseño entre en la FPGA, sin embargo, conlleva pérdida de precisión en el modelo. Subsanar la pérdida de precisión sería posible utilizando una FPGA con mayores recursos. La Tabla 3.2 muestra las operaciones totales utilizadas en el modelo 1 ya optimizado.

Tabla 3.2 Operaciones utilizadas modelo 1 optimizado

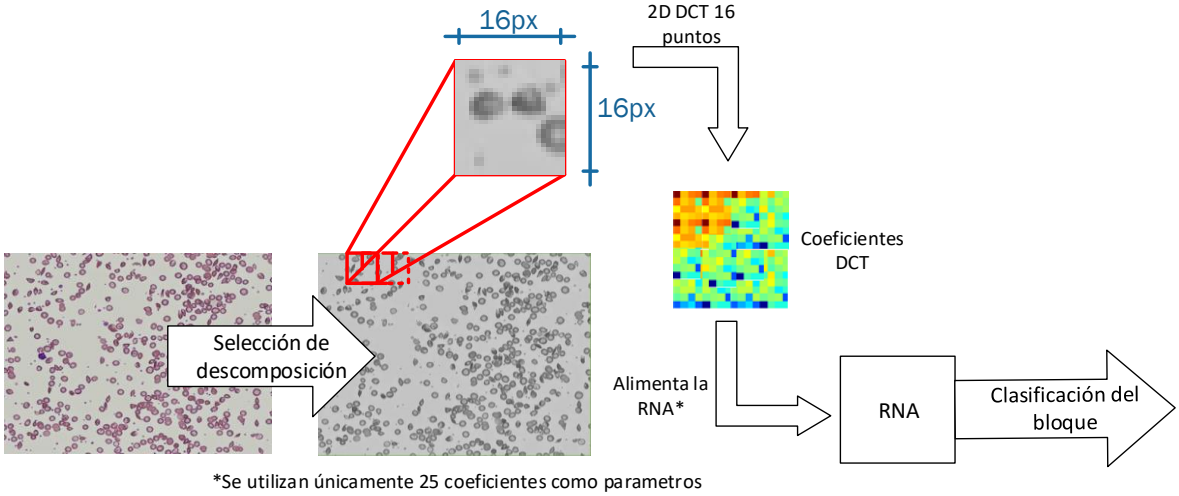
Algoritmo	Sumas	Mult.	<<
DCT por columnas	27	11	6
DCT por filas	37	11	8
Normalización	96	24	0
Capa 1 RNA.	34	62	0
Capa 2 RNA.	4	2	0
Total de operaciones	198	110	14

3.6 FPGA, caso de estudio 2: imágenes en color + 3 clases

Para el segundo caso de estudio se implementó un segundo modelo. En éste se realizó la segunda aproximación para la extracción de características, así como el uso de una red neuronal para la clasificación de 3 tipos de células.

Las imágenes a utilizar en este modelo corresponden al grupo de imágenes de muestras sanguíneas. Estas no se encuentran en escala de grises (Figura 3.4) este hecho afecta al modelo propuesto para su clasificación, ya que, si bien esta información adicional del color se podría utilizar para otros mecanismos de extracción de características. Sin embargo, si no se realiza la transformación a escala de grises implica enviar 3 valores correspondientes a cada color (RGB) a la FPGA. La Figura 3.14 muestra una visión general de las etapas del modelo 2

Figura 3.14 Visión general de etapas del modelo 2



3.6.1 Conversión a escala de grises

Esta operación se realiza previo al envió píxel por píxel de la imagen a la FPGA por lo que esta transformación es efectuada por el anfitrión.

Matlab almacena en memoria las imágenes como una colección de 3 matrices del tamaño de la imagen, independientemente de su formato. Cada matriz corresponde a los valores de intensidad de cada componente RGB de la imagen.

Para simplificar esta transformación a escala de grises se decide utilizar una de las descomposiciones de color de la imagen. Para ello se utiliza un componente en Simulink denominado “selector” que permite seleccionar una colección de valores de un conjunto de matrices. Después de analizar las diferentes componentes de la imagen, se optó por utilizar la segunda componente correspondiente a la intensidad del color verde por ser la que mejores resultados obtiene.

Con la matriz obtenida de la transformación se alimentará el componente que permite enviar píxel por píxel la imagen a la FPGA. De manera similar al modelo implementado en el primer caso de estudio, los píxeles recibidos por la FPGA son almacenados en un controlador de memoria para su posterior procesamiento.

Para este modelo, el controlador de memoria permite almacenar y extraer 16 filas de la imagen cada vez, ya que para la ejecución de la implementación DCT se requieren 16 parámetros de entrada.

3.6.2 2D-DCT de 16 puntos

La siguiente etapa del procesamiento consiste en utilizar los 16 píxeles extraídos del controlador de memoria y utilizarlos como parámetros de entrada de la DCT.

Esta implementación de la DCT [8] consiste una serie de sumas, restas y desplazamientos de bits entre los mismos parámetros y en un orden predefinido, como lo muestra la Figura 2.8. Los coeficientes obtenidos de esta DCT se almacenan de manera transpuesta en memoria, de igual forma que el modelo 1, hasta haberse realizado y almacenado el resultado de 16 DCT por columna. Con esta matriz 16x16 de coeficientes DCT por columna se realiza nuevamente la DCT, esta vez por los coeficientes correspondientes a una fila. Para aprovechar los recursos de la placa las 16 operaciones DCT por filas se realizan en paralelo.

Al finalizar estas ejecuciones de la DCT se obtiene como resultado una matriz de 16x16 correspondiente a los coeficientes DCT de un bloque de imagen de 16x16 píxeles. Al igual que en el modelo 1, solo se extraerán 25 coeficientes siguiendo el mismo patrón que en el modelo 1 y como lo indica la Figura 3.5. Este número de 25 coeficientes se seleccionó siguiendo el mismo criterio de análisis de la concentración de valores en el cuadrante superior izquierdo de la matriz utilizada en el modelo 1. Con los 25 coeficientes extraídos en esta etapa se alimenta la red neuronal definida en la siguiente etapa del modelo.

3.6.3 Red neuronal de 3 clases

Esta etapa consiste utilizar los coeficientes extraídos de la DCT y utilizarlos como parámetros de entrada en la red neuronal.

Esta red neuronal sigue el mismo esquema que la utilizada en el modelo 1, sin embargo, el número de nodos tanto como para la capa oculta como la capa de salida varía. Para la capa oculta se realizaron pruebas únicamente utilizando 5 nodos empleando la misma función de activación que en el modelo 1. En la capa de salida se utilizaron 3 nodos, cada nodo representa cada tipo de clase. El primer nodo corresponde a los hematíes sanos, el segundo a los hematíes con anormalidades y el tercero a los trombocitos.

Al igual que en el modelo 1, esta red neuronal arroja valores entre 0 y 1 para cada uno de los 3 datos de salida. De estos valores el que más se aproxime a 1, superando un umbral definido, indica tipo de clase a la que pertenece el bloque de la imagen.

3.6.4 Recursos utilizados y optimizaciones

Con el fin de contabilizar los recursos empleados para el segundo caso de estudio, para el modelo 2 también se realizará una breve explicación de las operaciones utilizadas en la elaboración de los algoritmos.

En la primera etapa el controlador de memoria se necesita 16 bloques de memoria RAM de 640 de tipo entero de 8 bits. En el caso de la DCT, el costo de la aplicación tanto por columnas como por filas es el mismo y se implementó el algoritmo propuesto por Bouguezel [8] con 16 parámetros de entrada. La implementación consiste de 64 sumas y 8 desplazamientos de bits por cada aplicación de la DCT. Este coste debe multiplicarse por 16 para obtener el costo real de la aplicación de la DCT por filas, ya que estos algoritmos se ejecutan en paralelo.

En la primera capa de la implementación de la red neuronal, se realiza la misma normalización utilizada en el modelo 1, con la diferencia que se utilizan 25 parámetros de entrada lo que da un total de 100 sumas y 25 multiplicaciones para dicha implementación. La capa oculta de red neuronal, esta consta de 5 nodos ocultos lo que implica un coste de 25 multiplicaciones y 25 sumas por cada nodo, incluyendo 5 sumas adicionales antes de la aplicación de la función de activación. A diferencia del modelo 1, la función de activación tangente hiperbólica fue remplazada por una tabla de búsqueda (lookup table) con el fin de reducir operaciones por lo que el único coste que tendrá esta función será el almacenamiento de los valores de la función en un rango determinado.

La segunda capa de la red neuronal consta de 3 nodos, cada uno de eso tiene un costo de 5 multiplicaciones y 5 sumas, lo que da un total de 15 multiplicaciones y 15 sumas. A esto hay que añadir 3 sumas adicionales antes de efectuar la función “SoftMax”.
En la función “SoftMax”, se incluye una tabla de búsqueda que contiene los valores previamente calculados de la función exponencial según los valores de salida de los pesos efectuados en la

segunda capa de la red neuronal. Esta función tiene un coste total de 3 multiplicaciones y 3 sumas, añadiendo el espacio de almacenamiento en memoria necesario para la tabla de búsqueda.

Para este modelo también se realizaron optimizaciones. Gracias a que la aproximación del algoritmo DCT utilizado no emplea multiplicaciones, se pudo utilizar más nodos (hasta 5) para la capa oculta y 3 nodos para la capa de salida de la red neuronal, lo que incrementa significativamente las operaciones efectuadas. La Tabla 3.3 muestra los recursos utilizados por el modelo sin realizar ningún tipo de optimización.

Tabla 3.3 Operaciones utilizadas modelo 2

Algoritmo	Sumas	Mult.	<<
DCT por columnas	64	0	8
DCT por filas	64x 16	0	8 x16
Normalización	100	25	0
Capa 1 RNA	(25x5) +5	25x5	0
Capa 2 RNA.	((3*5) +3) +3	(5*3) +3	0
Total de operaciones	1344	168	136

Se puede observar que las multiplicaciones de este modelo siguen superando los bloques DSP indicados en la FPGA (87), en especial si estas superan los 18 bits de precisión. Para intentar reducir estas operaciones se aplicó una aproximación similar a la del modelo 1, sin embargo, realizar eliminaciones de operaciones innecesarias en la implementación del algoritmo DCT no impactará en la cantidad de multiplicaciones.

Para ello se utilizaron las herramientas de optimización de Matlab y Simulink, concretamente las que permiten la serialización de operaciones. Para este modelo se aplicó la serialización en varias operaciones como la normalización de valores y en la multiplicación de los pesos de la primera capa de la red neuronal. Las Figuras 3.15 y 3.16 muestran los bloques de Simulink sin la optimización de serialización y con la optimización.

Figura 3.15 Normalización sin serialización

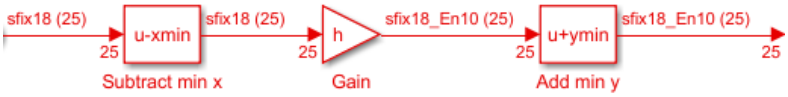
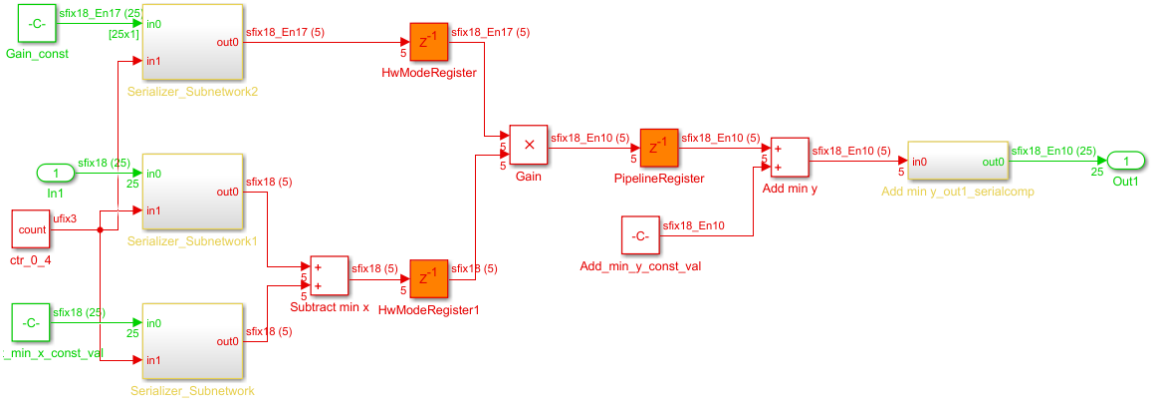


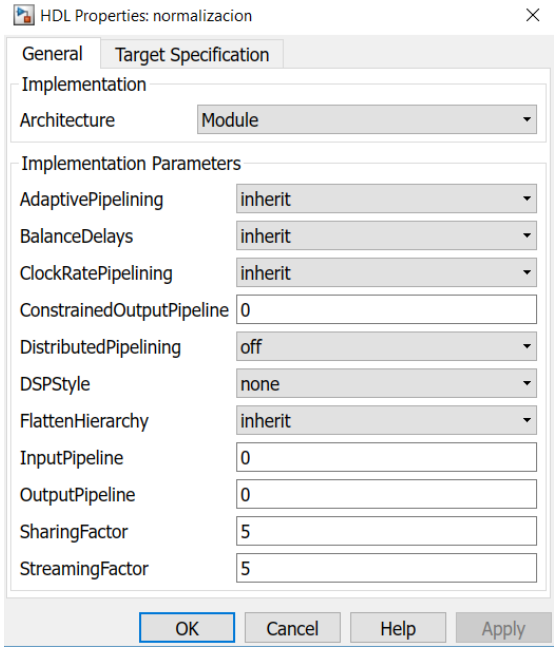
Figura 3.16 Normalización con serialización



Se observa como esta optimización reduce la cantidad de multiplicaciones (bloques Gain en ambas figuras) de 25 a solo 5, modificando únicamente los parámetros del bloque de normalización e indicando un “SharingFactor” y un “StreamingFactor” de 5, como se muestra en la Figura 3.17. Este tipo de optimización crea bloques adicionales que permiten serializar los parámetros de entrada y las constantes definidas en la multiplicación, sin embargo, añade un retraso adicional en los tiempos de ejecución.

Adicionalmente a este tipo de optimizaciones, también se modificó el esquema general de la función de activación, cambiando la aproximación basada en divisiones y sumas por una tabla de búsqueda (Lookup table). Se aprovecha el hecho de que Matlab cuenta con esta función implementada en su código y se establece la tabla de búsqueda utilizando como rango mínimo y máximo los valores extremos que puedan arrojar la multiplicación de los pesos por los parámetros de entrada.

Figura 3.17 Propiedades del bloque normalización



Al igual que en el modelo 1, también se realizó un exhaustivo seguimiento a los tipos de datos utilizados en todas las operaciones, intentando reducir al máximo los bits de precisión con el fin de reducir la cantidad de bloques DSP necesarios por el modelo al momento de compilación.

Gracias a estos cambios, se logró reducir considerablemente los bloques DSP utilizados por el modelo, sin embargo, la serialización de casi todas las operaciones tendrá un impacto negativo en los tiempos de ejecución. La Tabla 3.4 muestra las operaciones realizadas en el modelo luego de su optimización

Tabla 3.4 Operaciones utilizadas modelo 2 optimizado

Algoritmo	Sumas	Mult.	<<
DCT por columnas	64	0	8
DCT por filas	64x 7	0	8 x7
Normalización	10	5	0
Capa 1 RNA	(25) +5	25	0
Capa 2 RNA.	((3*5) +3) +3	(5*3) +3	0
Total de operaciones	573	48	64

3.7 Creación de las redes neuronales

Ambos modelos utilizan el mismo tipo de red neuronal, sin embargo, se diferenciarán en la cantidad de nodos ocultos y los parámetros de entrada. Esto implica que cada modelo de red neuronal debe ser entrenada y probada con los datos de entrada suministrados.

Para poder obtener las muestras de entrenamiento, se requirió ejecutar los modelos con las imágenes de entrenamiento con la finalidad de extraer los coeficientes DCT de cada subbloque de la imagen. Para el primer modelo de las 5 imágenes correspondientes al grupo de células tumorales se utilizaron 215 muestras, de las cuales 50 corresponden coeficientes de bloques de la imagen con presencia de células y otras 165 que no. Para el segundo modelo de las 2 imágenes disponibles se utilizó un lote de 350 muestras donde 106 corresponden a hematíes con poiquilocitosis, 128 hematíes sanos, y 116 muestras de trombocitos.

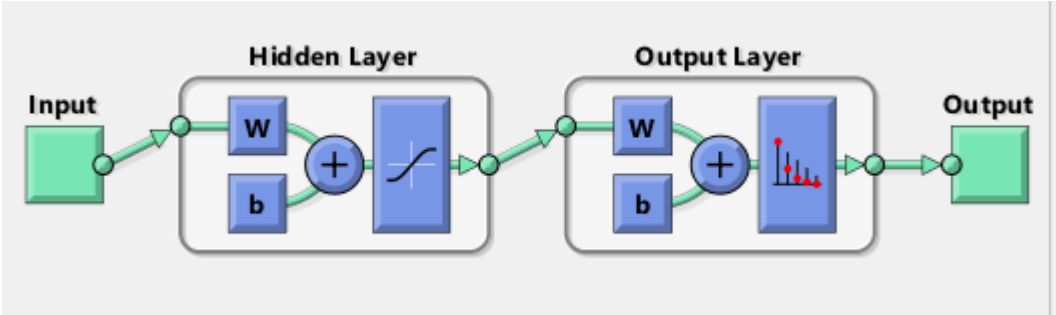
Después de la obtención de las muestras de las características se ejecutó el "Neural Network Toolbox" de Matlab, esta permite la creación automática de la red neuronal. Sin embargo, la herramienta solo permite modificar la cantidad de nodos ocultos de la red. Se utilizaron los parámetros por defecto durante la creación utilizando el 70% de las muestras para entrenamiento, 15% para validación y 15% para pruebas.

La red generada para el primer modelo tiene 24 parámetros de entrada correspondiente a las 24 características por muestra y 2 parámetros de salida en la capa de salida, que indican la célula identificada y una que representa espacio vacío de la imagen. Es decir que cada nodo en la capa de salida corresponde a un tipo de clase. En el caso de la red neuronal del segundo modelo consiste en 25 parámetros de entrada por muestra y 3 parámetros de salida correspondiente a cada tipo de célula a clasificar.

La Figura 3.18 muestra el diagrama general para los modelos de red neuronal de clasificación generado por Matlab. Se puede observar que la red neuronal consta de 2 capas, una capa con nodos ocultos (Hidden Layer) que consta de una multiplicación de pesos por los valores

de entrada y una función de activación sigmoidea. Para la segunda capa se tiene nuevamente una multiplicación de pesos acompañada de una función SoftMax.

Figura 3.18 Diagrama general de una red neuronal para clasificación en Matlab[7]



Para comprobar que los coeficientes contenidos en las muestras de entrenamiento ofrecen una buena diferenciación para lograr una clasificación efectiva, Matlab ofrece una serie gráficas y diagramas que indican la precisión lograda por el entrenamiento. En el apartado 4.1 se muestran los resultados obtenidos por el asistente de Matlab para la creación de redes neuronales para ambos modelos.

La Figura 3.19 muestra un ejemplo de las matrices de confusión por cada etapa del entrenamiento de las muestras (entrenamiento, validación, pruebas) y una matriz final agrupando todos los resultados. La Output Class, representa las clases obtenidas como resultado de la ejecución de la red neuronal con las muestras de entrenamiento, por otra parte, la Target Class indica cual es la clase a la cual debía clasificarse la muestra. Para una muestra si el resultado de la red neuronal coincide con la clase real de la muestra se incrementa en uno el valor de la diagonal de la matriz (celdas verdes) correspondiente a la clase y en el caso de que no, se incrementa la celda donde la Output Class fue la clase obtenida y la Target Class la clase real de la muestra. Los valores de la diagonal de cada matriz significan el número de aciertos obtenidos para una determinada clase, los cuadros rojos indican una clasificación errónea de una clase y su ubicación con respecto a la diagonal representa a que clase fue clasificado erróneamente. Los cuadros azules indican el porcentaje de aciertos/fallos para el total de muestras, y por último los cuadros grises indican el porcentaje de acierto/fallo para la clasificación de una determinada clase.

[7] Fuente: https://es.mathworks.com/help/nnet/gs/classify-patterns-with-a-neural-network.html

Figura 3.19 Ejemplo de matriz de confusión

Además de estos gráficos, el asistente para la creación de la red neuronal ofrece otros datos sobre la creación de los modelos de la red neuronal como: grafica de entropía-cruzada, histograma de error, características operativas de recepción, entre otros. Dado que se trata de información relativa al entrenamiento y no a la precisión que ofrece la red, se incluirán estos diagramas para cada modelo en el apartado de anexos.

Una vez que las redes neuronales han sido generadas, Matlab permite exportarlas a diferentes entornos, como Simulink, código Matlab, Código C, entre otros. Aprovechando esta característica, además de realizar la exportación a los diseños en Simulink, se almacenó el código en Matlab de las redes neuronales para crear el algoritmo en software de los modelos.

Es necesario destacar que las implementaciones en Simulink de las redes neuronales generadas por Matlab no son directamente compilables a código HDL por los siguientes motivos: las funciones de activación sigmoidea y la función de normalizado exponencial correspondientes a las capas de la red neuronal 1 y 2 respectivamente, contienen operaciones que requieren de un tipo de dato double, ya que ambas utilizan la función exponencial que no es sintetizable a hardware. Para poder compilar el modelo y ejecutarlo en la FPGA fue necesario conseguir aproximaciones de estas funciones mantengan un alto grado de precisión.

La función de activación sigmoidea se trata de una tangente hiperbólica (TanH) que normaliza los valores entre -1 y 1. Esta función es fácilmente aproximable a una serie infinita de divisiones y sumas, como lo muestra la Figura 3.20. Para este proyecto la acotación de la función a 6 divisiones ofrece la precisión suficiente para su implementación en hardware.

Figura 3.20 Representación infinita en fracciones de la función tangente hiperbólica

$$\tanh z = \cfrac{z}{1 + \cfrac{z^2}{3 + \cfrac{z^2}{5 + \cdots}}}$$

Para función de normalizado exponencial, que regulariza los parámetros de entrada arrojando valores de resultado entre 0 y 1, se opta por emplear una tabla de búsqueda (Lookup table) para la función exponencial e implementar el algoritmo de la red neuronal sustituyendo la función por la tabla. La Figura 3.21 muestra la función softmax empleando la función exponencial.

Figura 3.21 Función Softmax

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad donde\ j = 1, \ldots, K.$$

3.8 Desarrollo en Software

Con el fin de mostrar la efectividad de los modelos ejecutándose sobre la placa, se decidió crear algoritmos en software para realizar comparativas de los tiempos de ejecución de éstos con objeto de compararlos con los obtenidos para la FPGA.

Estos algoritmos consisten en desarrollar en software lo especificado en los 2 modelos. El primer algoritmo se basó en descomponer en escala de grises las imágenes de entrada según se necesite, posteriormente se efectuó la extracción de los bloques de imágenes almacenándolos en un directorio para su posterior procesamiento.

El siguiente algoritmo consistió en la implementación de ambas aproximaciones a la función DCT y su posterior extracción de las características realizando un patrón de zigzag sobre la matriz de coeficientes, de igual manera, aplicando la DCT en 2 dimensiones mediante la descomposición por columnas y luego por fila. Y por último la aplicación del código previamente generado de las redes neuronales.

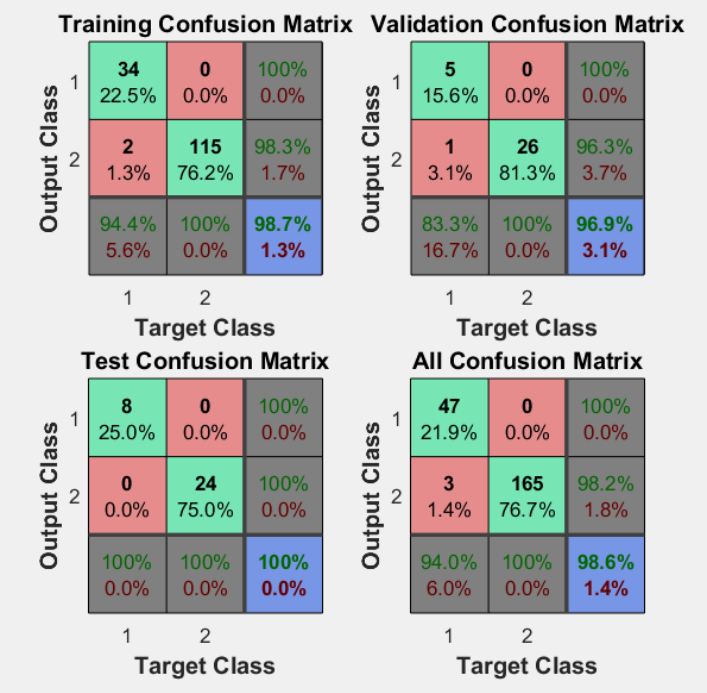
Se desarrolló utilizando como lenguaje de programación Matlab generando 3 scripts ejecutables. En todos los scripts se utilizaron tipos de datos de precisión double.

Capítulo 4 - Análisis experimental

4.1 Resultados de entrenamiento de las RNA

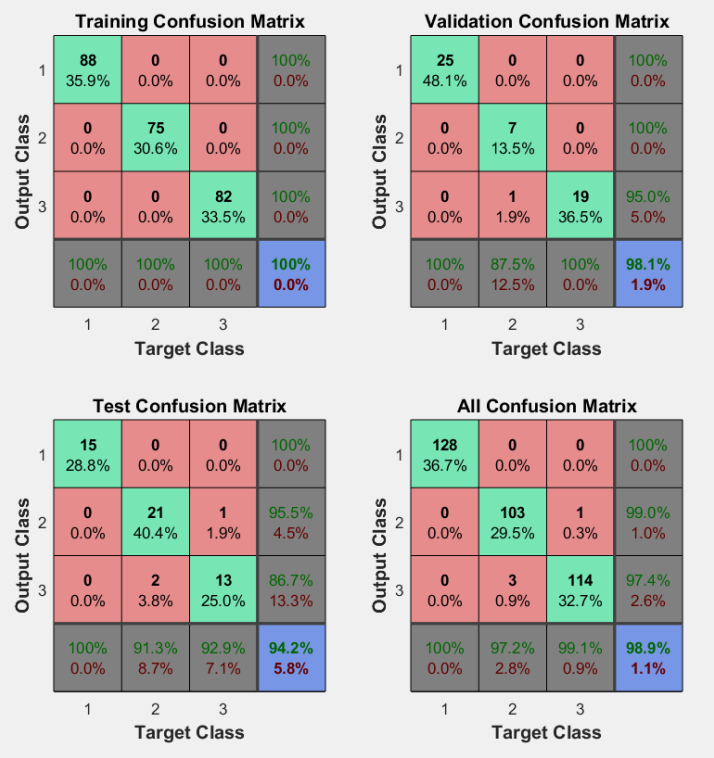
En la siguiente sección se muestran los resultados obtenidos por el asistente para redes neuronales de Matlab. Como muestras de entrenamiento se utilizaron los coeficientes de la DCT obtenidos directamente de las ejecuciones de los modelos en las FPGA.

Figura 4.1 Matrices de confusión modelo 1



La Figura 4.1 muestra la matriz de confusión del modelo 1, y presenta la precisión lograda por el algoritmo de entrenamiento con las muestras de entrenamiento. La clase 1 representa los bloques de las imágenes que contienen células tumorales y la clase 2 las que no. Se observa un porcentaje de clasificación correcta del 98.6% para ambas clases, sin embargo, para la clasificación correcta de la célula este valor se reduce al 94%. Cabe destacar que estos porcentajes altos de clasificación no significan que el 94% de las imágenes se clasificarán de manera correcta, ya que existen muchos factores que disminuyen ese porcentaje, como lo puede ser la precisión del muestreo y que los valores de entrenamiento no incluyan algunas características significativas de gran parte de las muestras.

Figura 4.2 Matrices de confusión modelo 2



La Figura 4.2 muestra las matrices de confusión generado por en el entrenamiento de la red neuronal para el modelo 2, a diferencia del modelo 1 en esta se puede observar una fila y una columna adicional que corresponde una tercera clase a clasificar. La primera clase representa los hematíes sanos, la segunda clase los hematíes con anormalidades y la tercera los trombocitos. Se muestra igualmente la cantidad de acierto/fallo por cada clase, así como su porcentaje de clasificación.

Se puede observar que el porcentaje de clasificación de la red es del 98.9% y que, para las clases de Hematíe Sano, Hematíe con Anormalidad y Trombocitos corresponde un porcentaje de 100%, 97.2% y 99.1% respectivamente. Al igual que para el modelo 1, este alto porcentaje no implica directamente una alta precisión en la clasificación correcta, más aún en el caso particular del modelo 2, ya que no se están incluyendo muchas otras muestras que estarán presentes en las imágenes, como espacios sin células, bloques en donde converjan más de una célula, células sin clasificar, entre otros.

4.2 Tiempos de ejecución del modelo 1

El "FPGA in-the-loop" del toolbox HDL Verifier, permite ejecutar los modelos en co-simulación con la FPGA, esto es, que el equipo anfitrión al momento de iniciar la simulación del modelo también envía los parámetros de entrada directamente a la FPGA sin la necesidad de realizar ninguna configuración adicional. Sin embargo, el envío y recepción de la información de la imagen a la FPGA impacta negativamente en los tiempos reales de ejecución de la FPGA.

Como lo indicado en la sección 3.4 con respecto a los tiempos de envío y recepción de los datos desde y hacia la FPGA, se creó un modelo denominado "modelo de control". Con dicho modelo se midió el tiempo que tomó el envío a la placa de imágenes con mismas dimensiones que las imágenes de entrenamiento, de igual manera se midió los tiempos de la recepción en el anfitrión de información constante predefinida en la FPGA, utilizando los mismos tipos de datos empleados en el diseño original. De esta manera es posible identificar el tiempo empleado en el envío y recepción de la información a la FPGA.

Simulink permite controlar el envío de la imagen a la FPGA ya sea por píxel por píxel, línea de píxeles, o la imagen completa (frame). Las siguientes pruebas se realizaron ejecutando 10 veces el diseño compilado en la FPGA. Para comprobar la eficacia del modelo se validó el mejor control de envío de datos óptimo para el diseño. Las Tablas 4.1 y 4.2 muestran los tiempos de ejecuciones en segundos para cada modo de envío de la imagen.

Tabla 4.1 Tiempos de ejecución modelo de control (en segundos)

Diseño Control	Min.	Max.	Avg.
Píxel	96.99	106.57	101.53
Línea	4.43	4.56	4.58
Frame	6.89	7.01	6.96

Tabla 4.2 Tiempos de ejecución modelo 1 (en segundos)

Diseño DCT + RNA	Min.	Max.	Avg.
Píxel	97.00	107.23	102.36
Línea	4.49	5.37	5.04
Frame	6.94	7.72	7.37

Los resultados de las pruebas demuestran la velocidad real de los cálculos en la FPGA permitiendo procesar en promedio una imagen de 640x480 cada 5.04 segundos cuando se envía línea a línea, sin embargo, para normalizar este tiempo es necesario restar el tiempo del envío, arrojando una ejecución real de 0.53 segundos (579.62 Kpps), por imagen 0.41 segundos (749.26 Kpps) cuando se realiza el envío de la imagen completa y siendo el peor resultado el envío de píxel por píxel con un tiempo de ejecución de 0.83 segundos (379.25 Kpps).

Estos resultados no incluyen el tiempo del Host en procesar la información recibida y realizar el pintado sobre la imagen de las células identificadas, por lo que se realizó otra batería de pruebas sobre el diseño con los bloques adicionales para el pintado, arrojando en promedio un incremento del 0.20 segundos por imagen para cada modo de envío.

4.3 Efectividad de la clasificación modelo 1

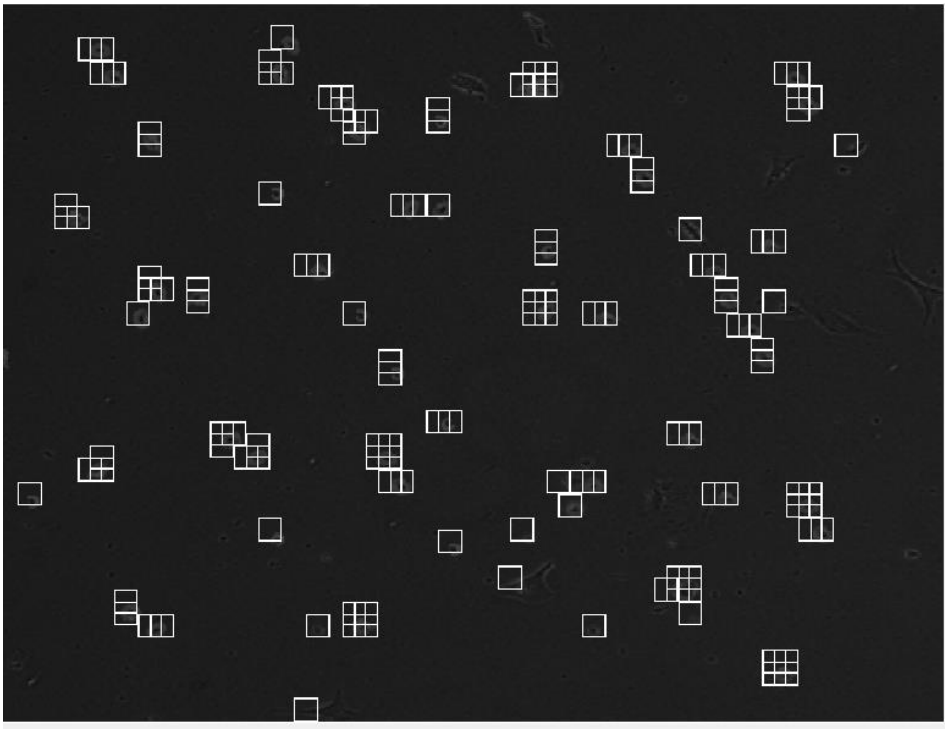
Con respecto a la precisión en la clasificación, para todos los cambios en la red neuronal realizados (con 2 ó 4 Nodos) se detectaron correctamente todas las células, sin embargo, también se incluyeron sectores de las imágenes que no representaban células.

Tabla 4.3 Resultado clasificación de células

Nodos	Falsos Positivos	Falsos Negativos	Detecciones totales	Células presentes	Precisión	Recall
2	9	0	61	52	52/61	1
4	12	0	64	52	52/64	1

La Tabla 4.3 muestra la precisión en la detección de las células utilizando 2 y 4 nodos en la red neuronal. Se puede identificar claramente como el diseño es capaz de detectar todas las células presentes, sin embargo, a una mayor cantidad de nodos en la red neuronal se introducen más errores en la clasificación de las imágenes. La falta de precisión en la red neuronal se debe a varios factores: por un lado, las optimizaciones necesarias para su síntesis en la FPGA y por otro lado la falta de más imágenes el entrenamiento de la red neuronal. En la Figura 4.3 se muestra el resultado de ejecutar el modelo 1 sobre la imagen de entrenamiento inicial.

Figura 4.3 Imagen resultado Modelo 1 con 2 nodos ocultos



4.4 Tiempos de ejecución modelo 2

Para los cálculos de los tiempos de ejecución de este modelo, se empleó la misma metodología que en el modelo anterior, ejecutando el modelo 10 veces por cada modo de envió de datos, utilizando en la FPGA el diseño sintetizado y enviando las imágenes de muestra mediante el FPGA in-the-loop de Simulink. La Tabla 4.4 muestra los tiempos de ejecución obtenidos para el modelo 2 por cada uno de los modos de envió disponibles en Simulink.

Tabla 4.4 Tiempos de ejecución modelo 2 (en segundos)

Diseño DCT + RNA	Min.	Max.	Avg.
Píxel	125.53	132.73	127.00
Línea	5.29	7.77	6.81
Frame	9.55	12.72	11.44

Para este modelo la FPGA es capaz de procesar en promedio una imagen de 640x480 cada 6.81 segundos cuando se envía línea a línea. Al igual que en el modelo anterior es necesario normalizar estos valores para obtener el tiempo real empleado por la placa, por lo que se realiza la

resta de los tiempos del modelo control a tiempos reales obtenidos de la FPGA. Se obtiene un tiempo de procesamiento promedio de 2.01 segundos (152.83 Kpps) cuando se envía la imagen línea por línea, 4.48 segundos (68.57 Kpps) cuando se realiza el envío de la imagen completa y siendo el peor resultado el envío de píxel por píxel con un tiempo de ejecución de 25.47 segundos (12.06 Kpps).

Se observa una marcada diferencia con respecto al modelo 1, principalmente porque este modelo contiene más optimizaciones basadas en la serialización de bloques de operaciones. Se muestra como el modelo generado pierde velocidad de procesamiento. Esta disminución es causada principalmente por la serialización de la mayoría de sus operaciones ocasionando que los datos en la FPGA tengan que esperar más ciclos de reloj entre operaciones.

4.5 Efectividad de la clasificación modelo 2

La precisión obtenida para este modelo difiere un poco a la mostrada en el modelo anterior, principalmente porque existen más clases que clasificar, esto implica que la precisión de los datos con los que se alimenta de la red neuronal no es suficiente para lograr un buen grado de clasificación.

Como se mencionó anteriormente, para este modelo no se realizaron pruebas variando la cantidad de nodos de la capa oculta de la red neuronal, por lo que los resultados mostrados se obtuvieron utilizando 5 nodos ocultos en la red neuronal. Se utilizaron esta cantidad de nodos ya que es el minino valor divisible de los 25 coeficientes extraídos de la DCT, esto con el fin de poder serializar las operaciones que incluyen multiplicaciones.

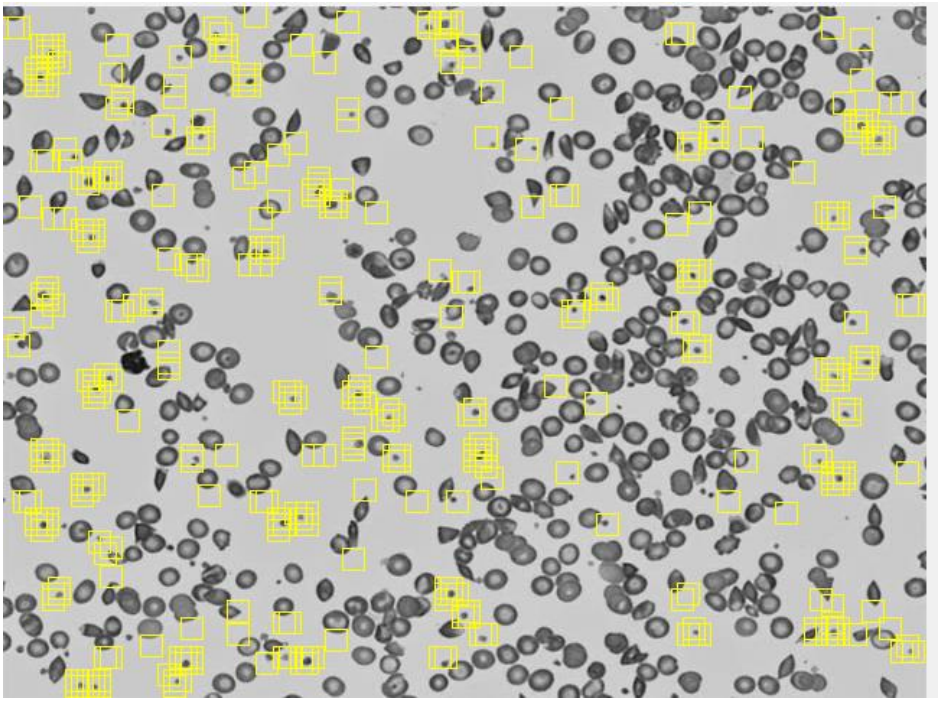
Tabla 4.5 Resultado clasificación de células

Clase	Falsos Positivos	Falsos Negativos	Clasificaciones correctas	Detecciones totales	Células presentes	Precisión	Recall
Hematíe Sano	28	0	265	293	265	265/293	1
Hematíe con poiquilocitosis	92	24	24	120	48	24/120	24/48
Trombocito	34	23	93	127	116	93/127	93/116

La Tabla 4.5 muestra la precisión en la detección de las células utilizando 5 nodos ocultos en la red neuronal. Se observa como el modelo es capaz de clasificar correctamente el 73.2% de los trombocitos, sin embargo, solamente fue capaz de clasificar correctamente el 20% de las muestras de los hematíes con anormalidades con un alto porcentaje de errores. Para los hematíes sanos hubo un 89.41% de clasificación de acierto, con una cantidad baja de falsos positivos correspondientes principalmente a hematíes con poiquilocitosis.

El modelo posee un alto grado de clasificación para los trombocitos y hematíes sanos, sin embargo, en el caso de los hematíes con anormalidades disminuye considerablemente este grado de precisión en la clasificación. Esto se puede deber directamente al proceso de entrenamiento, ya que este proceso no incluyó muestras que contuviesen otros elementos presentes en la imagen como espacios en blanco, otros tipos de células como el solapamiento de diferentes células. La Figura 4.4 muestra un ejemplo de clasificación para el modelo 2, se puede ver en los cuadros amarillos la identificación de los trombocitos.

Figura 4.4 Imagen resultado Modelo 2 clasificación de trombocitos



4.6 Tiempos de ejecución algoritmo en software

Para medir los tiempos de ejecución de los algoritmos utilizados en los modelos, pero implementados en software, se empleó el mismo criterio que con los modelos, se ejecutó 10 veces cada algoritmo tomando los tiempos totales de ejecución. A diferencia de los modelos en hardware, los modos de envió de datos: por píxel, línea, o frame, carecen de aplicación para estas ejecuciones.

El equipo utilizado para estas ejecuciones fue un portátil marca Asus n56jr con un procesador Intel i7-4700HQ a 2.4Ghz de velocidad junto con 16 GB de memoria RAM a 1600Mhz.

La Tabla 4.6 muestra los tiempos de ejecución arrojados por los algoritmos de los modelos implementados para software, se observa como para el modelo 1 el tiempo de ejecución ronda los 14.50 segundos, mientras que el del modelo 2 puede llegar a sobre pasar el minuto. Para ambos modelos en software la clasificación fue mejor que la obtenida en hardware debido a la precisión que aportan los cálculos con tipos de datos double. Se muestra una marcada diferencia entre cada modelo, ya que para el modelo 1 la ventana que extrae los bloques de 8x8 píxeles se desplaza cada 8 píxeles solapando las muestras de cada muestra, para el caso del modelo 2 la ventana que extrae los bloques de 16 píxeles de la imagen se desplaza píxel por píxel. Quiere decir que el modelo 1 analiza menos muestras que el modelo 2.

Tabla 4.6 Tiempos de ejecución de los modelos en software (en segundos)

Modelo en software	Min.	Max.	Avg.
Modelo 1	14.25	15.69	14.50
Modelo 2	57.62	61.02	58.03
Modelo en hardware[8]	Min.	Max.	Avg.
Modelo 1	4.49	5.37	5.04
Modelo 2	5.29	7.77	6.81

[8] Corresponde a los tiempos de ejecución cuando se envía la imagen a la FPGA línea por línea

4.7 Recursos finales de los modelos

Para el modelo 1 debido a la necesidad de realizar optimizaciones, el diseño final de la red neuronal solo pudo contener un máximo de 4 nodos, ya que al incrementar el número de nodos el modelo sobrepasa los recursos disponibles en la placa. Se realizaron diversas pruebas incrementando y disminuyendo la cantidad de nodos en la red neuronal, arrojando como mejor candidato 2 nodos para la clasificación de células. Utilizando esta cantidad de nodos en la capa oculta de la red neuronal los recursos arrojados por Quartus II al momento de la síntesis son los siguientes: 15,919 / 32,070 (50 %) Bloques ALM, 39 / 397 (10 %) Bloques de RAM y 77 / 87 (89%) Bloques DSP.

Para el modelo 2, diferencia del modelo 1, no se realizaron pruebas alterando el número de nodos ocultos de la red, por lo que podría obtenerse mejores resultados en la clasificación probando con diferentes combinaciones de valores. Una vez aplicados las optimizaciones indicadas en la sección 3.6.4 del presente trabajo, los recursos arrojados por el proceso de “fitter” posterior al de análisis y síntesis del Quartus II fueron los siguientes: 16,884 / 32,070 (53 %) Bloques ALM, 54 / 397 (14 %) Bloques de RAM, y 87 / 87 (100 %) Bloques DSP

Capítulo 5 - Conclusiones

Se han presentado dos implementaciones de un algoritmo clasificatorio de imágenes de microscopía de células basada en redes neuronales utilizando la FPGA Cyclone V de la placa DE1-SoC de Altera. Aunque la FPGA cuenta con velocidades de solo 50Mhz es capaz de realizar cálculos casi en tiempo real para el proceso de clasificación de imágenes.

Herramientas de alto nivel como Matlab y Simulink demuestran ser de gran utilidad para el desarrollo de algoritmos complejos, a pesar de que los toolbox que se ofrecen solo contengan componentes de bajo nivel. Además, a estos toolbox, el uso de simulaciones en Simulink acelera la capacidad de probar los modelos previo paso a su compilación.

Las implementaciones en FPGA de algoritmos de altas prestaciones siempre se verán afectados por la precisión que puedan ofrecer los recursos disponibles en el hardware. Siempre es recomendable el análisis previo del diseño con el fin de redistribuir los recursos de segmentos de código que puedan tener poco impacto en el resultado a otros que requieran una mayor precisión.

Los casos de estudio implementados demuestran la efectividad del algoritmo DCT para la extracción de las características, de igual manera ambas estrategias utilizadas ofrecen un alto grado de precisión para la clasificación de células.

5.1 Trabajo Futuro

A pesar de los resultados obtenidos, siempre existe capacidad de mejora para estos algoritmos, principalmente en la precisión obtenida en los modelos. Podría realizarse un exhaustivo análisis de los tipos de datos con el fin de lograr el nivel óptimo de bits necesarios para cada operación del modelo.

También podría mejorarse el grado de clasificación probando con diferente cantidad de nodos en las redes neuronales o empleando otras funciones para las capas de la red. Adicionalmente se podría realizar pruebas extrayendo más coeficientes de la DCT de 2 dimensiones con las que alimentar las redes neuronales.

El uso de hardware con mejores recursos elimina la necesidad de limitar los algoritmos a implementar, por lo que usar una FPGA no orientada al ámbito educativo puede facilitar la elección de los algoritmos.

Como lo indicado en la sección 2.6, Altera también ofrece un modo de ejecución que permite utilizar operaciones de punto flotante en los diseños, esto podría mejorar los modelos eliminando la necesidad de hacer una transformación de las operaciones a punto fijo e inclusive mejorar la precisión de los algoritmos. Para esto se necesitaría una FPGA con mayores recursos.

Introduction

Nowadays there exists several devices such the GPUs that allows computing acceleration. These devices are mostly focused on high consumption tasks like image rendering or video editing, however, they possess a high price and a high energy consumption due to the need a high clock speed and memory.

On the other hand, another type of device is gaining popularity for computing acceleration: the Field-Programmable Gate Arrays (FPGAs). These hardware devices contain a large amount of logical resources - simpler than those in a GPU - that allow the efficient execution of algorithms focused on parallel execution. Thanks to the architecture of this type of hardware, the energy consumption is considerably reduced. However, from the developer point of view it is harder to develop programs for these platforms.

There are several tools focused on algorithm modeling for hardware. Each FPGA vendor offers its own tool like Quartus II for Altera boards or Vivado by Xilinx for their own FPGAs. On the other hand, there are some generic tools like Matlab and Simulink. Although these are tools for general development, there are modules that allow developing algorithms in hardware without the necessity of having a deep knowledge in HDL languages (Verilog or VHDL).

To implement the algorithms in hardware, in this work a research will be carried out to determine which algorithms are available for the extraction of cell characteristics in images of microscopy and its subsequent classification through neural networks.

The model implementation will be realized through Matlab and Simulink, by using their corresponding packages for FPGAs development. Once the implementation is finished, a study will be made of the performance and cost of the resources utilized by the algorithms. The target platform will be the FPGA Cyclone V, which is embedded within the Altera DE1-SoC board, and Matlab R2016b with Simulink will be used as main development environment.

Motivation

The recognition and classification of cells has been a necessity in different scientific fields such as medicine or biology. Although there are several algorithms and tools for this purpose, nowadays in most cases biologists continue using the manual method to carry out this cell recognition and classification, which provokes a considerable loss of time for the researcher. On

the other hand, the main limitation of automatic processing is due to the high complexity of the algorithms and their response times using conventional computers. The efficient acceleration of these algorithms can generate a new field for the creation of new applications in real time that allows improving recognition and classification.

Objectives

General objective

The main objective of this work is to validate Matlab and Simulink as an efficient environment for the development of FPGA hardware algorithms. As a practical case, two cell classification algorithms will be implemented using the Altera DE1-SoC board that contains a FPGA Cyclone V.

Specific objectives

• Implementing an effective algorithm in hardware for the extraction of characteristics in images that contain blood cells for their later classification using neural networks.

• Analyzing the effectiveness of Matlab and Simulink for developing algorithms in hardware without directly using HDL (Hardware Description Language) code.

• Analyzing the algorithms implemented by measuring the effectiveness in the classification of cells.

• Reviewing the usability of the toolboxes offered by Matlab and Simulink to generate complex algorithms for hardware.

• Comparing the performance of the classification algorithms with a software implementation run on a conventional station. As reference for the comparison the following station will be used: an Asus N56jr notebook with a Core i7-4700HQ processor at 2.4Ghz and 16Gb DDR3 at 1600Mhz of RAM.

Conclusions

Two implementations of a classification algorithm for cell-based microscopy images have been presented. These have been deployed using neural networks and the Cyclone V FPGA as target board. Although the FPGA possesses a reduced 50Mhz frequency, it can perform almost real-time calculations for the image classification process.

High-level tools like Matlab and Simulink have proved to be very useful for the development of complex algorithms, even though the toolboxes just offer low-level components. The model simulations with Simulink accelerate the ability to test the models prior to their compilation.

FPGA implementations of high performance algorithms will always be affected by the accuracy of available hardware resources. Hence, it is always advisable to pre-analyze the design to distribute the resources of code segments that have a moderate impact on the result, rather than others requiring greater precision.

The two cases of studies demonstrate the efficiency of the DCT algorithm for the extraction of characteristics. Furthermore, the applied strategies for feature extractions offered the required high precision for cell classification.

Future Work

Despite the achieved results, there is always improvement space for these algorithms, mainly for the precision obtained by the models. A thorough analysis of the data types to achieve the optimum bitwidth could be performed to study the different tradeoffs and reduce the number of required resources.

The accuracy of the classification could also be improved by configuring the neural networks with a different number of nodes in the hidden layer.

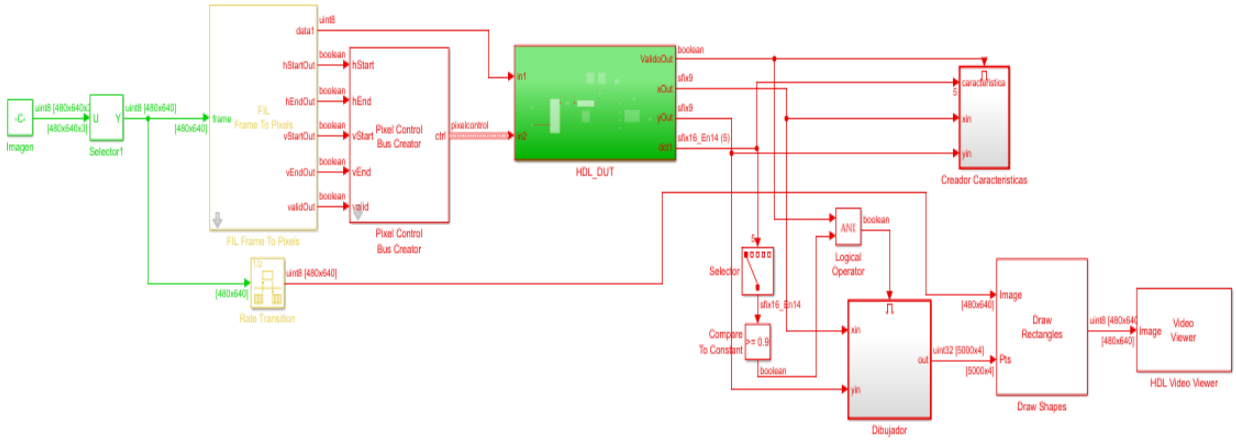
The use of hardware with more resources eliminates the limitations of the algorithms to be synthetized. Hence, a high-end FPGA may facilitate the algorithms deployment.

As indicated in section 2.6, Altera also provides an execution mode that allows using floating point operations in designs. This could improve the model design by eliminating the need to transform operations to fixed point format. This would improve the precision of the algorithms. Nevertheless, to use floating point operations, an FPGA with more resources is needed.

Apéndice A - Modelos desarrollados en Simulink

Con el fin de explicar la utilidad de cada bloque utilizado para diseñar los modelos, en esta sección se mostrarán diferentes capturas de pantalla de la herramienta Simulink con ambos modelos desarrollados al efecto en las siguientes 16 figuras.

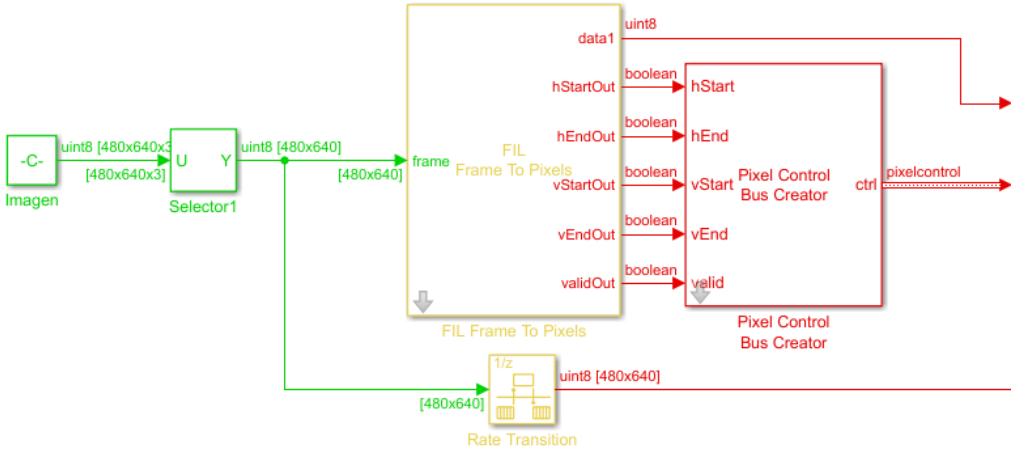
Figura A.1 Esquema general de ambos modelos



La Figura A.1 muestra el esquema general de ambos modelos. Es común a ambos modelos ya que emplean los mismos bloques para realizar la transformación de una imagen de 640x480 a un flujo de bits para su posterior envió a la FPGA, así como el post procesamiento de la información de la posición y las clases de las células en la imagen devueltas por la FPGA: Con el fin de una mejor visualización la Figura A.1 se divide en las Figuras A.2 y A.3.

En la Figura A.2 se muestran los componentes iniciales del proyecto, estos no se sintetizan ni se compilan en la FPGA ya que corresponde al equipo anfition de realizar la lectura inicial de la imagen. En el bloque “Imagen” hace referencia la funcion “imread” de Matlab, esta función realiza la carga en memoria de cualquier formato de imagen, y la transmite al resto de componentes del modelo. Al realizar la carga en memoria de cualquier imagen se obtienen 3 matrices 640x480 correspondiente a cada componente RGB. El siguiente bloque “Selector” permite realizar selecciones especificas de la información de entrada, para el caso de ambos modelos se utiliza la segunda componente, a pesar de que la imagen del mode 1 ya se encuentra en escala de grises.

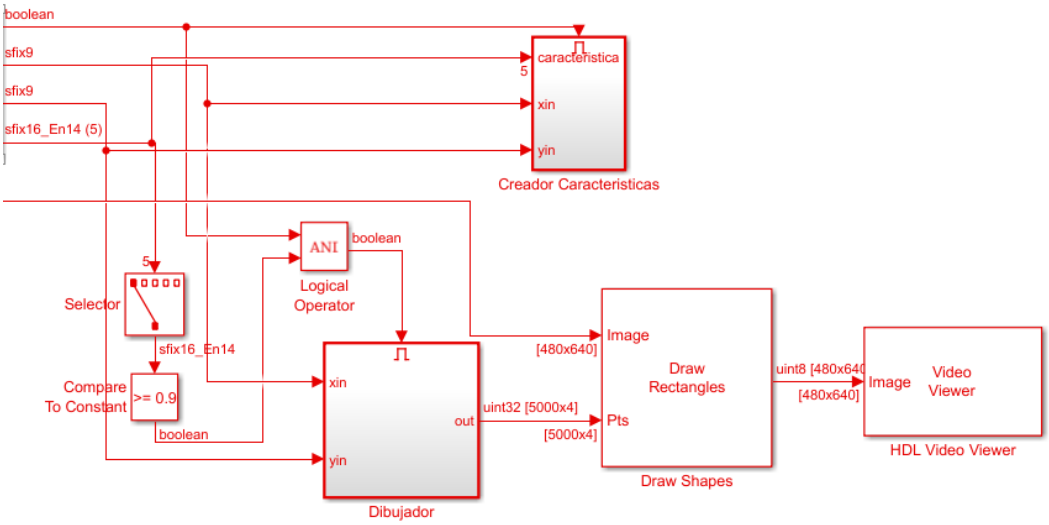
Figura A.2 Modelo general preprocesado



El siguiente componente corresponde a FIL Frame to Pixels. Este se encarga de la transformación de la imagen a un flujo de bits, cada salida de este bloque corresponde a propiedades para identificar la ubicación del píxel en la imagen. El Pixel Control Bus Creator permite agrupar múltiples parámetros en un solo elemento de entrada. Los colores de los componentes del modelo (verde, amarillo, rojo) corresponden a la tasa de muestreo dentro de la simulación, siendo el verde el tiempo de muestreo más lento y el rojo el más rápido. Es por ello que se incluye un componente llamado Rate Transmission que permite unir componentes con tiempo de muestreo diferente.

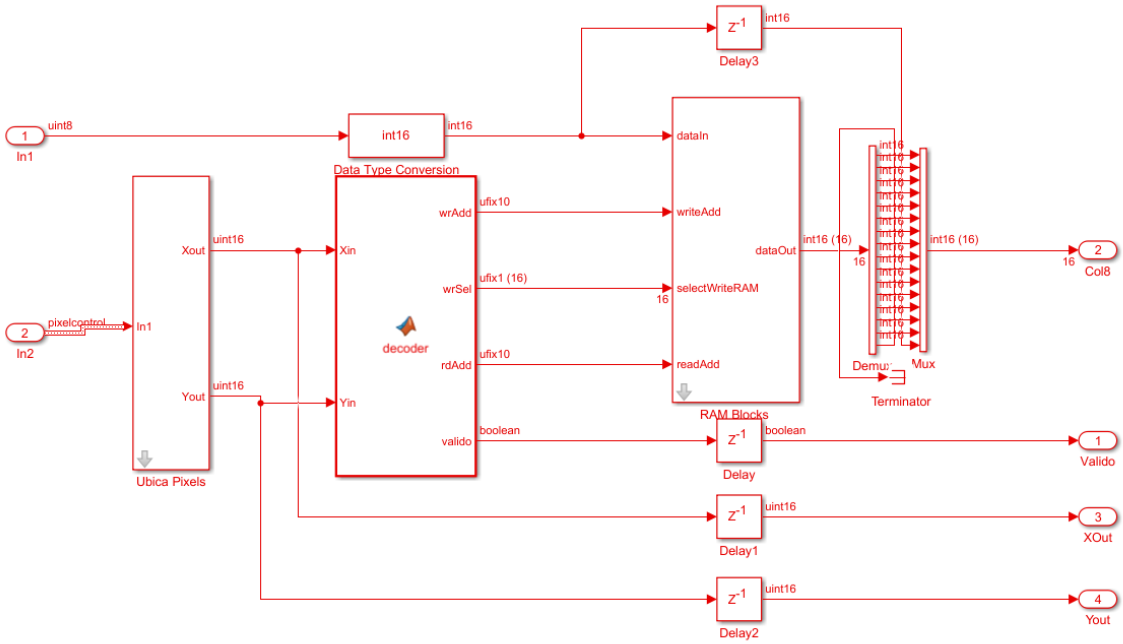
La Figura A.3 muestra el postprocesado del modelo, en este se obtienen los valores arrojados por la FPGA y se procesan para obtener las posiciones en la imagen de cada clase de célula clasificada. Se observan diferentes bloques, cada uno con una funcionalidad especifica. El bloque "Creador Características" permite almacenar los valores arrojados por el algoritmo DCT, así como la posición correspondiente al subbloque de la imagen. El bloque "Dibujador" junto con "Draw Shapes" se encargan de crear las estructuras necesarias para el pintado de los recuadros sobre la imagen. Por último, el bloque "HLD video viewer" permite visualizar la imagen con los cuadros identificando las células.

Figura A.3 Modelo general postprocesado



La Figura A.4 muestra el diseño del controlador de memoria utilizado para ambos modelos. Este consta de 2 etapas, la primera consiste en recibir el valor del píxel (in1) y las propiedades que definen su ubicación (in2). Estas propiedades son analizadas en el bloque "Ubica píxeles" arrojando la posición X e Y del píxel de entrada. La siguiente etapa, se encuentra el bloque "Decoder" y consiste en procesar esos valores de posición para generar los parámetros necesarios con los que alimentar los bloques de memoria. Estos parámetros incluyen la dirección de memoria en la que se inscribirá el píxel, la dirección de lectura y un booleano que indica si se escribe el dato. En el primer modelo, el controlador de memoria almacena hasta 8 filas por vez, mientas que el segundo almacena 16.

Figura A.4 Controlador de memoria



La Figura A.5 muestra la visión general de la implementación del modelo 1. Se muestran los datos de entrada (in1, in2) como lo son el entero correspondiente al valor del píxel y un conjunto de datos denominado "pixelcontrol". Este último dato contiene la información sobre la ubicación del píxel en la imagen. Como se puede apreciar el modelo consiste en aplicar el algoritmo DCT en 2 dimensiones y los datos de salida de este se introducen en el agrupador de características con las que se alimentaran la red neuronal. Se muestran también los 4 parámetros de salida correspondientes a la ubicación del bloque de la imagen procesado, así como los datos obtenidos de la red neuronal.

La Figura A.6 al igual que el modelo 1, cuenta con los bloques de "2D-DCT" y el bloque correspondiente a la red neuronal, sin embargo, no cuenta con el bloque agrupador ya que la aplicación del algoritmo devuelve todos los datos necesarios para alimentar la red neuronal.

Figura A.5 Visión general del modelo 1

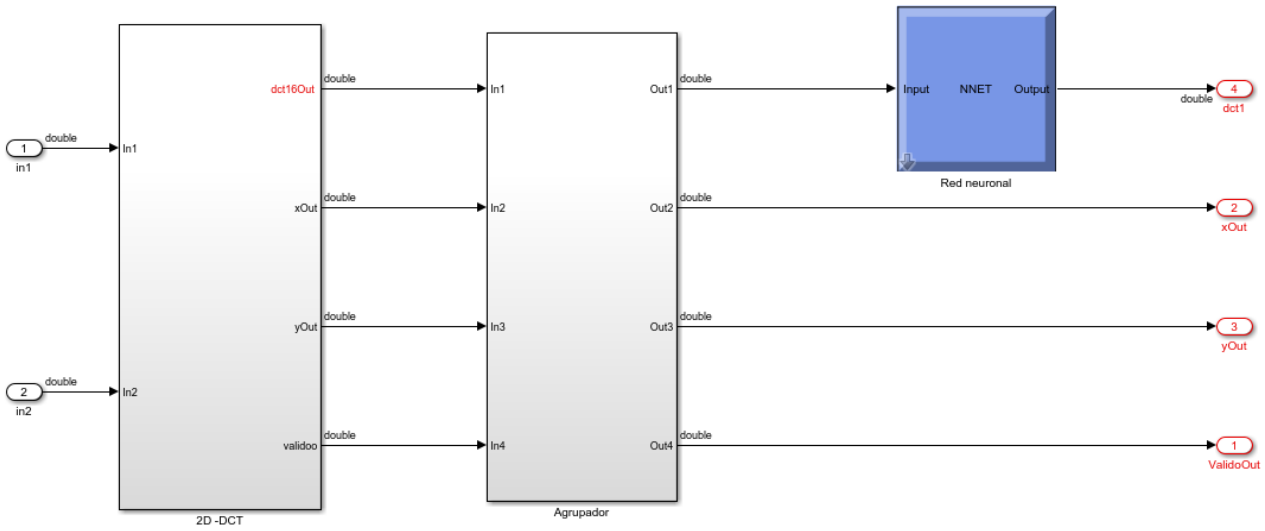
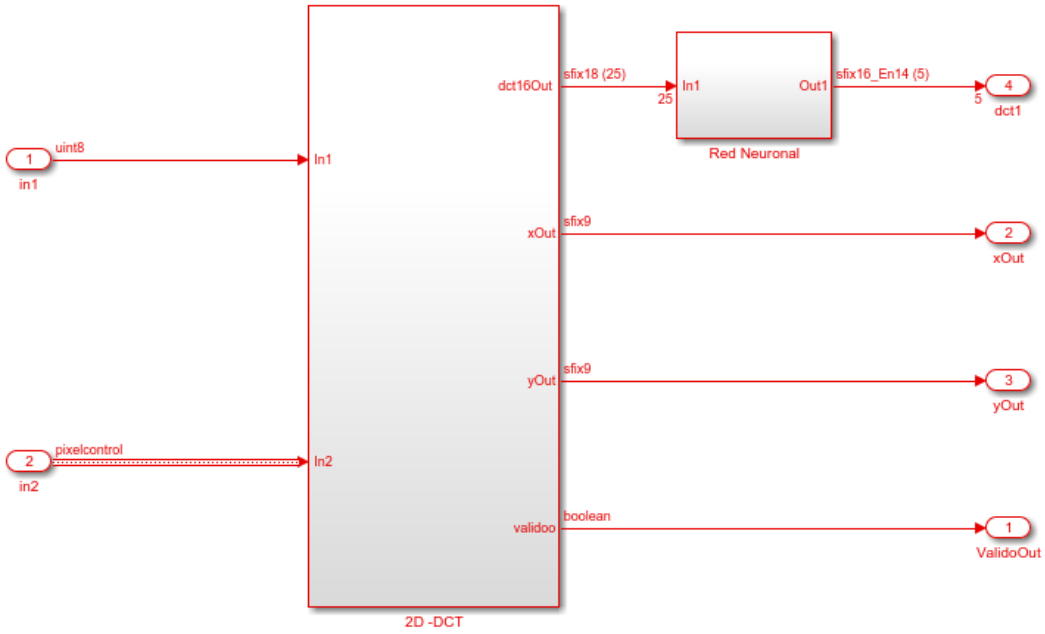


Figura A.6 Visión general del modelo 2

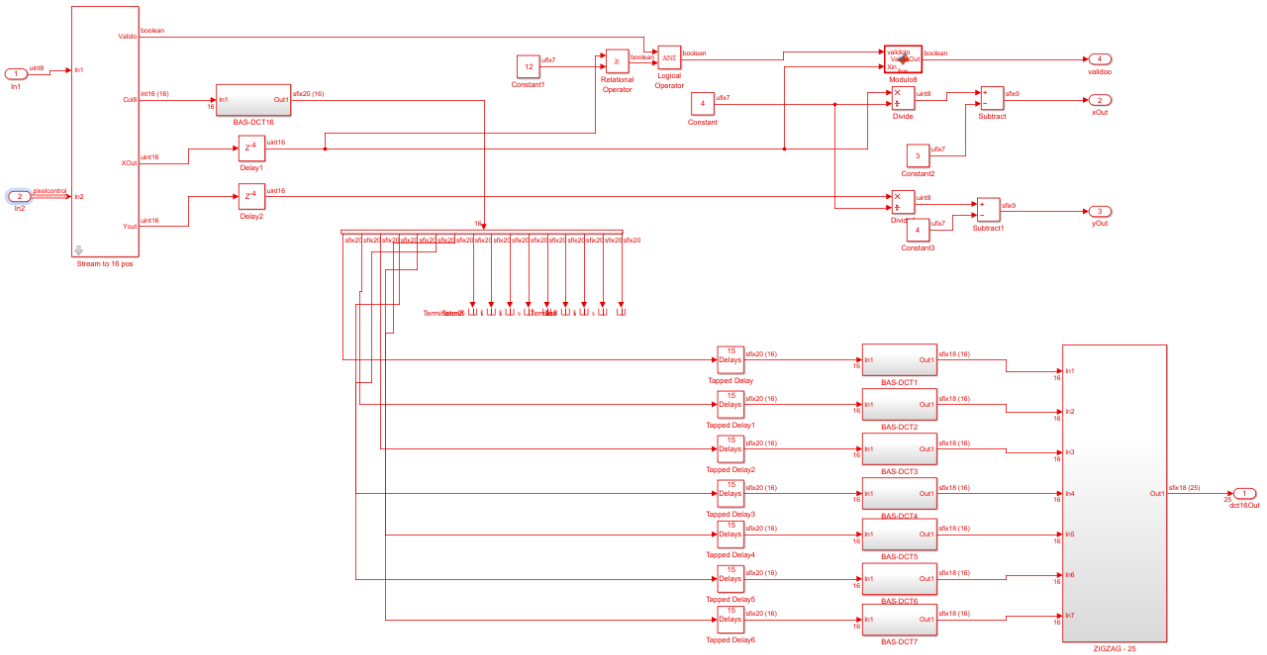


Dentro del bloque “2D-DCT" se muestra el contenido de la imagen A.7, en esta se aprecian las diferentes operaciones para aplicar la DCT en 2 dimensiones. Los datos de entrada pasan

directamente al controlador de memoria (descrito en la Figura A.5), este ofrece como salida una columna de 16 píxeles que es procesada por el bloque "COL DCT".

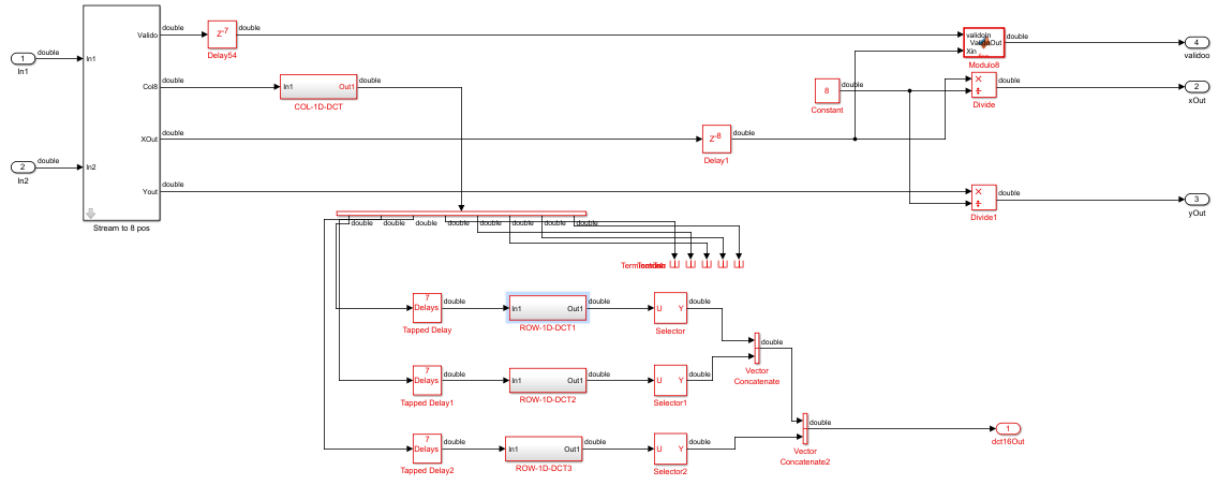
Este se encarga de aplicar el algoritmo DCT correspondiente y devolver los coeficientes obtenidos. Los bloques "Target Delay" permiten almacenar en memoria los coeficientes obtenidos hasta llegar a 16, los cuales se utilizan como parámetros de entrada de los bloques "ROW DCT". Estos aplicaran de nuevo el algoritmo DCT. Se observa que en esta última etapa se aplican en simultaneo varios algoritmos DCT. El siguiente bloque "Zigzag", consiste en extraer los valores correspondientes a la segunda aplicación del DCT y devolverlos en un orden especifico. Los demás bloques no mencionados en la imagen corrigen los valores de ubicación del bloque que se está procesando.

Figura A.7 DCT 2 dimensiones modelo 2



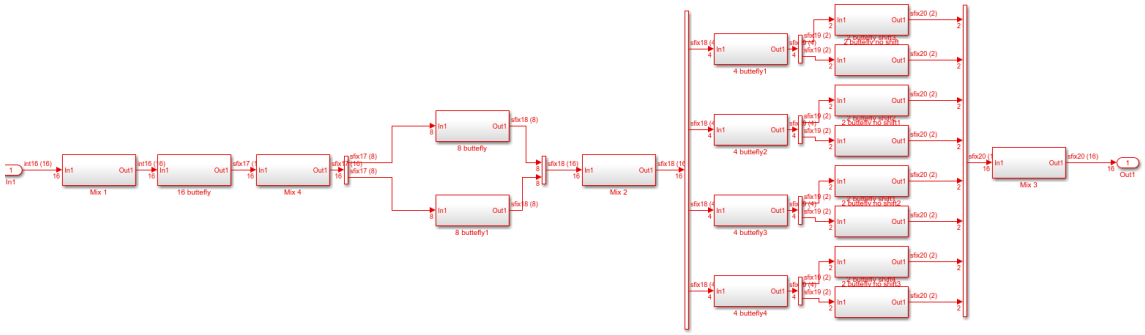
La Figura A.8 muestra la implementación del algoritmo DCT de 2 dimensiones utilizado en el modelo 1. Se aprecia la misma estructura que en la Figura A.7. Los valores se reciben por el controlador de memoria y posteriormente se efectua la DCT primero por columna y despues por fila.

Figura A.8 DCT 2 dimensiones modelo 1



La Figura A.8 muestra una visión global de la implementación del algoritmo DCT de Bouguezel [8]. En cada bloque se realizan diversas operaciones de sumas, restas, y desplazamientos de bits según a lo dispuesto por la aproximación de la DCT.

Figura A.9 Implementación DCT modelo 2 Bouguezel [8]



Las Figuras A.10 y A.11 son las implementaciones de la DCT para el modelo 1, los cuadros representan sumas o restas, mientras que los triangulos corresponden a multiplicaciones, los rectacngulos corresponden a desplazamientos de bits.

Figura A.10 Implementación DCT por columnas modelo 1

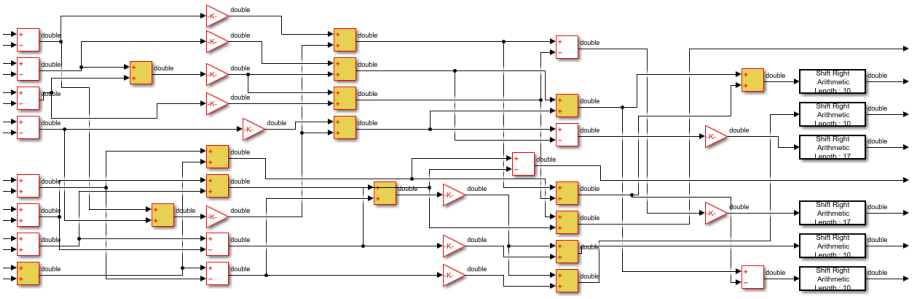
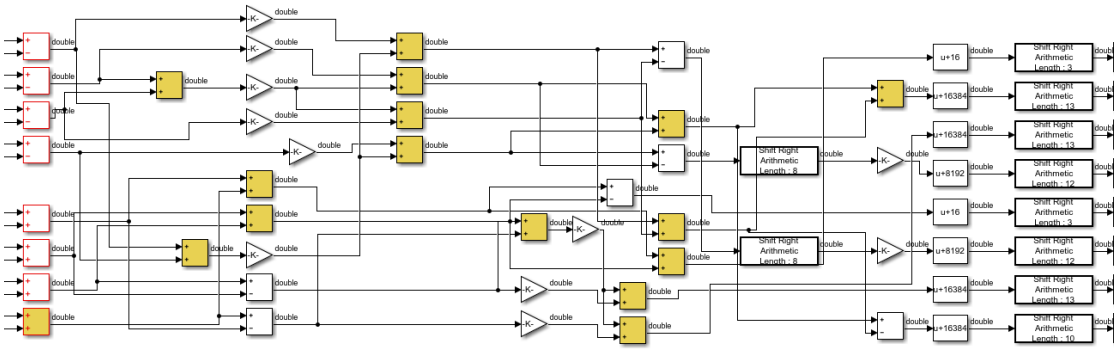
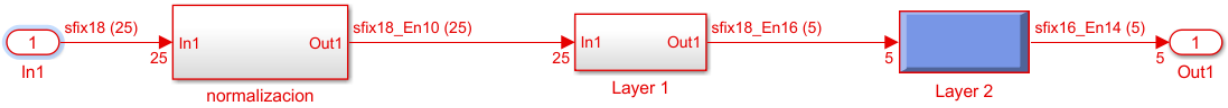


Figura A.11 Implementación DCT por filas del modelo 1



La figura A.12 muestra la captura de pantalla general de las redes neuronales en los modelos. Se puede observar las diferentes capas de la red, así como el bloque previo de normalización.

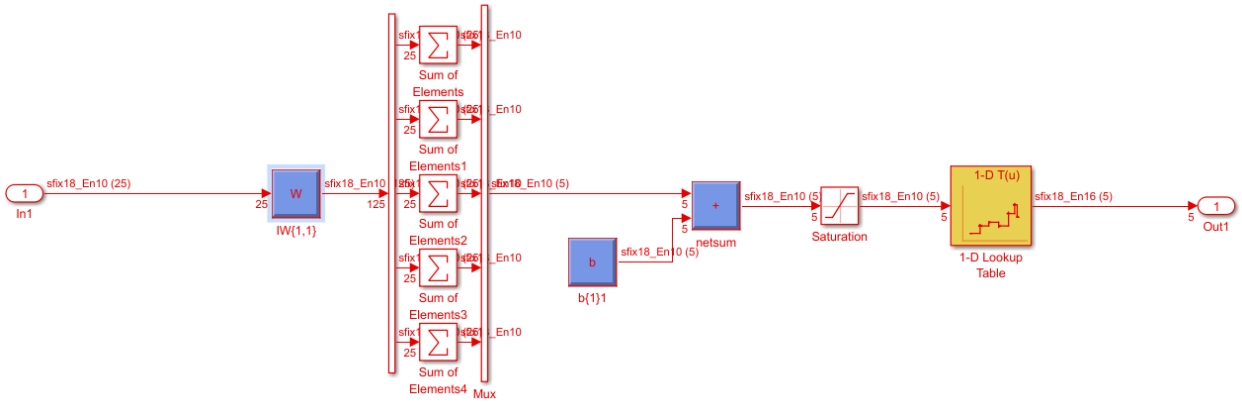
Figura A.12 Implementación de las redes neuronales ambos modelos.



La Figura A.13 muestra la captura de pantalla del diseño realizado para modelo 2, concretamente la capa 1 de la red neuronal. El bloque "IW{1,1}" consiste en 5 nodos donde cada

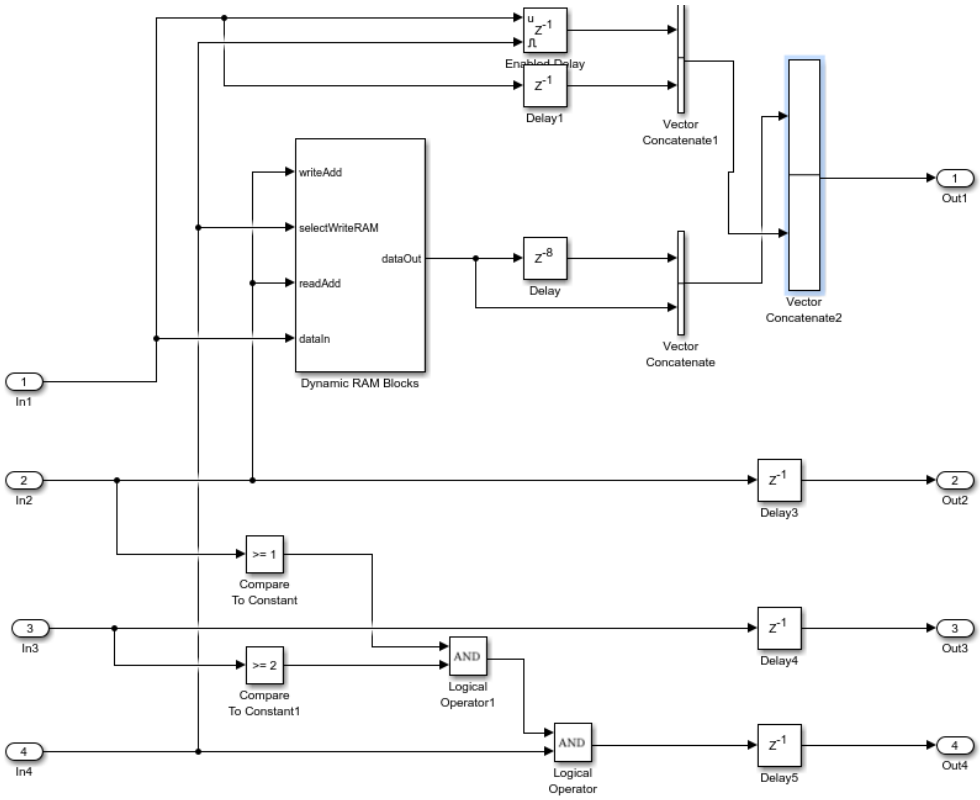
uno realiza la multiplicación de pesos por los 25 parámetros de entrada. El resultado son 125 valores que se suman en grupos de 25 como lo muestra el grupo de bloques "Sum of Elements", el resultado de estas sumas grupales son 5 parámetros con los que se alimentará la función activación denotada en la imagen por el bloque "Lookup Table". Para el modelo 1, este bloque se sustituye por el grupo de bloques definidos la Figura A.15.

Figura A.13 Capa oculta red neuronal modelo 2



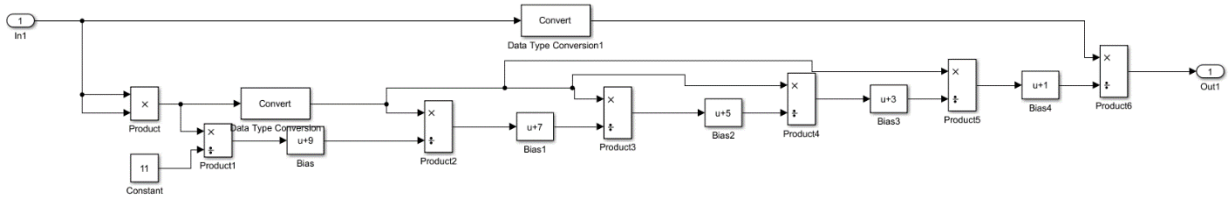
La Figura A.14 muestra la implementación del bloque "Agrupador" indicado en la Figura A.10. En la imagen se muestran los diferentes bloques para el almacenamiento de los coeficientes extraídos de la DCT para el modelo 1.

Figura A.14 Implementación bloque agrupador



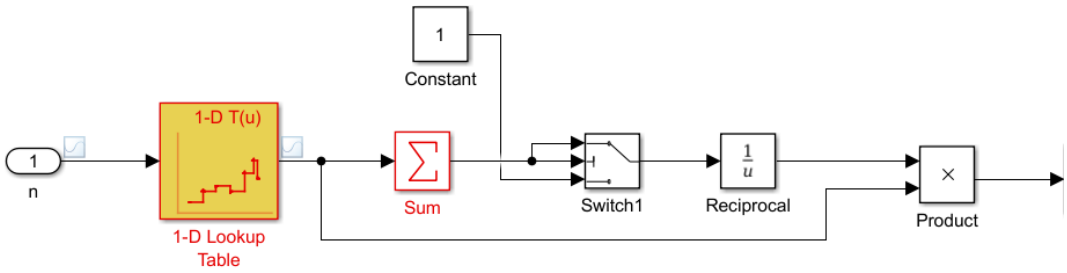
La Figura A.15 muestra la representación en sumas y divisiones de la función tangente hiperbólica. Estos bloques corresponden a la función de activación del modelo 1.

Figura A.15 Representación tangente hiperbólica en sumas y divisiones



La Figura A.16 corresponde a la función SoftMax utilizada en ambos modelos. Se Observa como la función exponencial fue substituida por una lookup table.

Figura A.16 Implementación función Softmax



Apéndice B - Resultados del entrenamiento de las RNA

En esta sección se describirán brevemente los diferentes gráficos generados por el asistente para redes neuronales de Matlab, específicamente en la creación de las redes para ambos modelos.

Las Figuras B.1 y B.2 muestran el gráfico de entropía cruzada del modelo 1 y 2 respectivamente. En esta imagen se muestran para cada iteración de entrenamiento (epochs) de la red neuronal los errores obtenidos por cada etapa del proceso de entrenamiento. Para el primer modelo (Figura B.1) resalta la iteración 37 con un error de $3.82x10^{-7}$ como la mejor para la clasificación según los datos de entrada ya que corresponde al menor error obtenido para la etapa de validación.

En el caso del segundo modelo, el error mínimo de la etapa de validación es considerablemente mayor, de 0.047 y es alcanzado en la iteración 16 del proceso de entrenamiento.

Figura B.1 Gráfico de entropía cruzada modelo 1

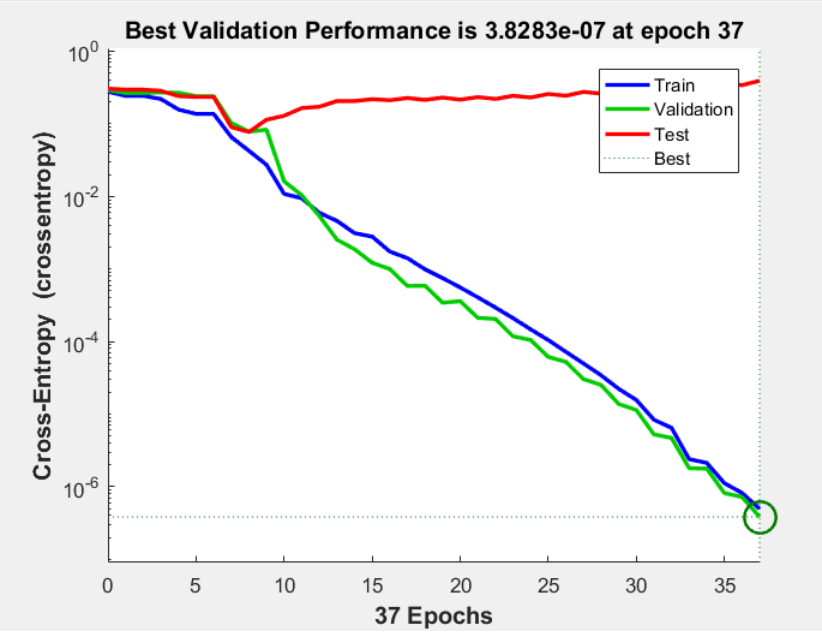
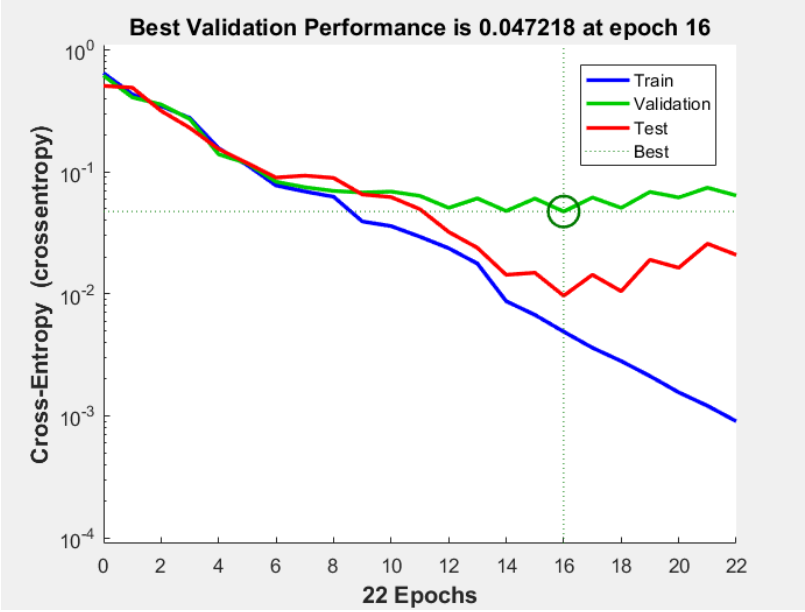
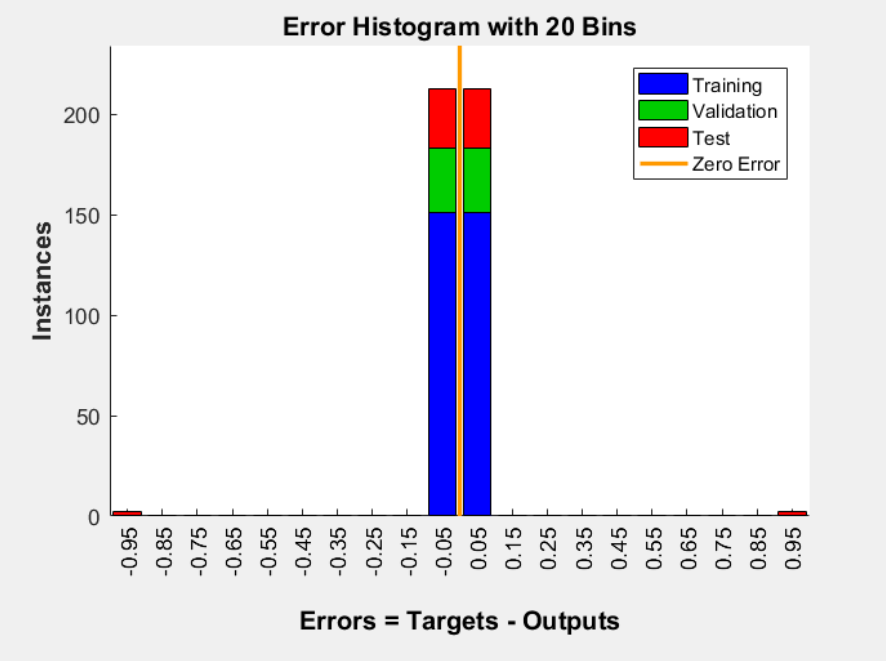


Figura B.2 Gráfico de entropía cruzada modelo 2



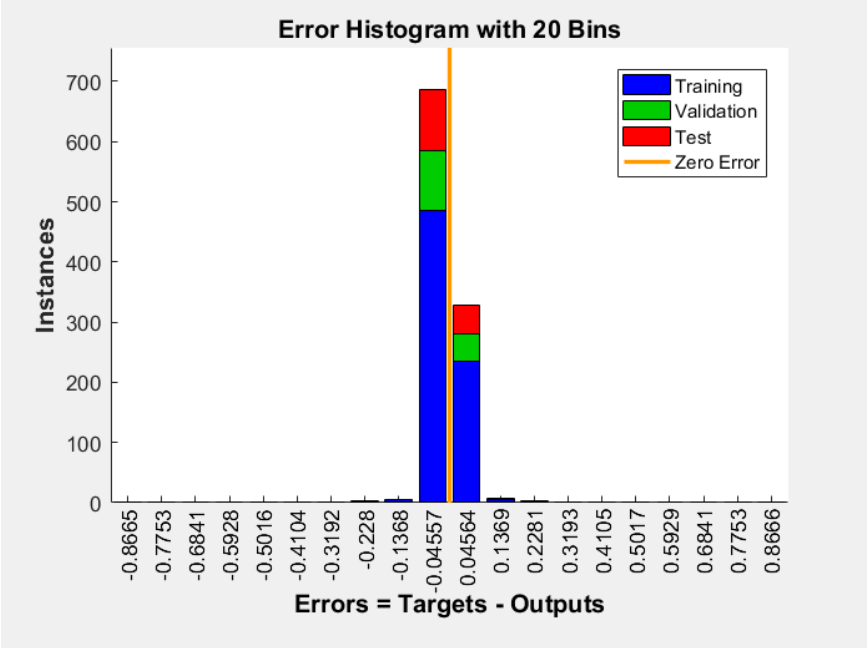
La Figura B.3 muestra el histograma de errores para la red neuronal del modelo 1 para cada una de las fases del entrenamiento. El error es calculado restando el valor obtenido para una muestra de la red neuronal al valor real de la muestra. La parte central del gráfico representa la zona de error 0 del modelo, hacia los extremos se encuentran rangos en los cuales se catalogan los valores de los errores obtenidos. Para este modelo, se muestran que los valores de error se concentran en el rango de –0.05 y 0.05 con algunos pocos valores en los extremos.

Figura B.3 Histograma de errores modelo 1



En el histograma del modelo 2 (Figura B.4) se observa que la mayoría de los valores de errores se encuentran en el rango de -0.045 y 0.045, pero se perciben más valores fuera de este rango que los encontrados en el modelo 1.

Figura B.4 Histograma de errores modelo 2



Las Figura B.5 y B.6 muestra el resumen del entrenamiento de la red neuronal de los modelos. Se puede apreciar el diseño de la red, indicando el número de parámetros de entrada y de salida, así como los nodos de cada capa.

En la sección de algoritmos se observan los diferentes parámetros utilizados como la estrategia de segmentación de las muestras, la función utilizada para el entrenamiento, el algoritmo utilizado para calcular la eficiencia.

Adicionalmente se muestran valores como las iteraciones utilizadas para el entrenamiento, el tiempo empleado en crear la red neuronal y la eficiencia lograda por la misma. Para el modelo 2 (Figura B.5) se muestran 25 parámetros de entrada con 5 nodos en la capa oculta, 3 en la capa de salida correspondientes a cada una de las clases a clasificar. Se observa que para crear la red realizaron 22 iteraciones obteniendo un desempeño de 0.000904.

En el modelo 1 (Figura B.6) se muestran 24 parámetros de entrada con 2 nodos en la capa oculta y 2 en la capa de salida correspondiente a los dos tipos de clases. Se requieren 37 iteraciones para obtener el desempeño de 4.95×10^{-7}.

Figura B.5 Resumen de entrenamiento red neuronal modelo 1

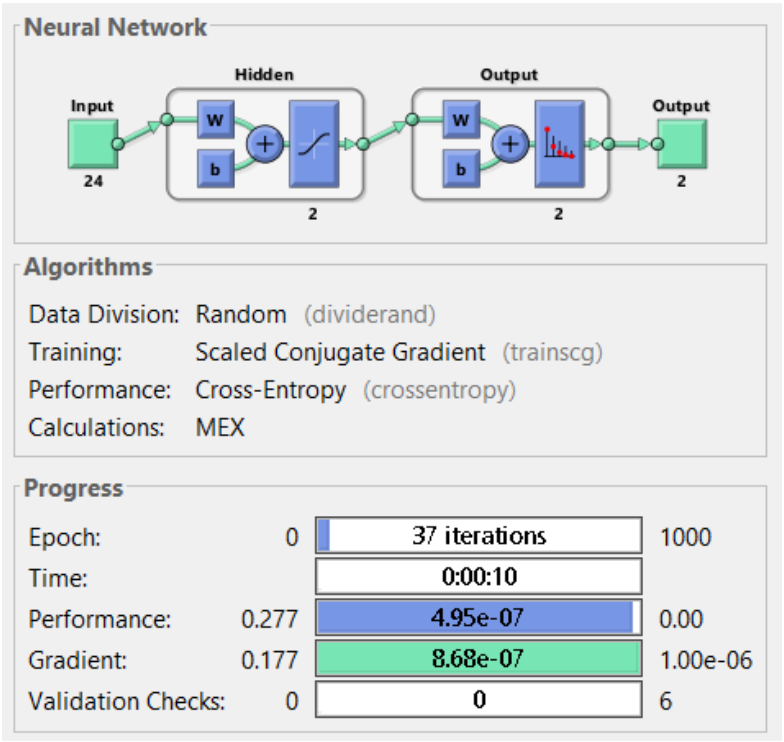
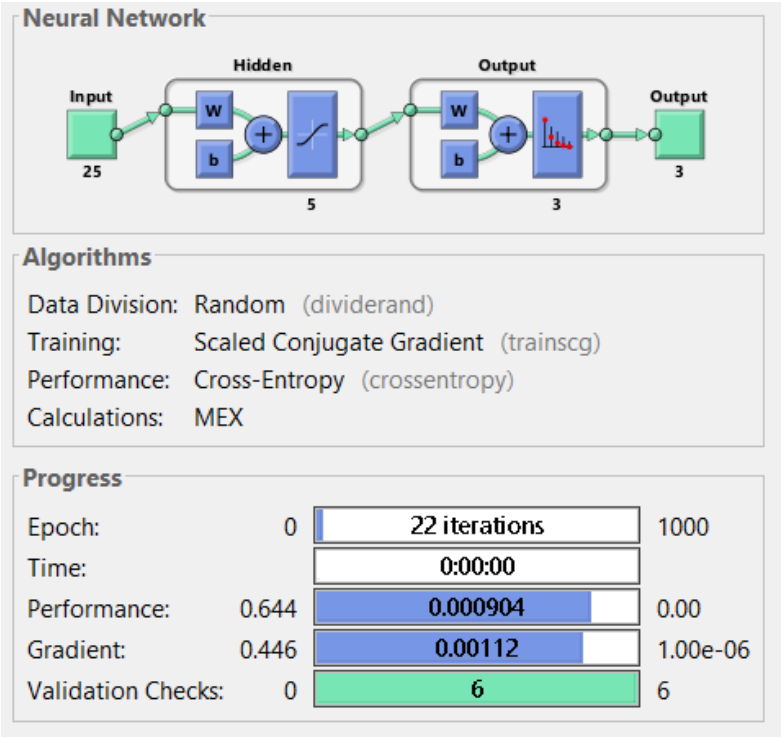


Figura B.6 Resumen de entrenamiento red neuronal modelo 2



Apéndice C - Imágenes de resultado y síntesis con Quartus

En esta sección se mostrarán los resultados obtenidos por Quartus durante la síntesis de los modelos. También se mostrarán las imágenes resultados del modelo 2.

La Figura C.1 y C.2 muestran la captura de pantalla de los recursos indicados por la herramienta Quartus en análisis y síntesis de los modelos. Se observa los recursos totales utilizados en la síntesis, así como un desglose de los elementos implementados,

Para el primer modelo se observa un total de 31893 recursos utilizados con 31392 celdas lógicas utilizadas, 416 segmentos de memoria RAM y 77 elementos DSP. Para el modelo 2, se muestra una leve disminución de los recursos totales, hasta los 30349, sin embargo, se observa un incremento considerable de los elementos DSP.

Figura C.1 Resumen de recursos Quartus modelo 1

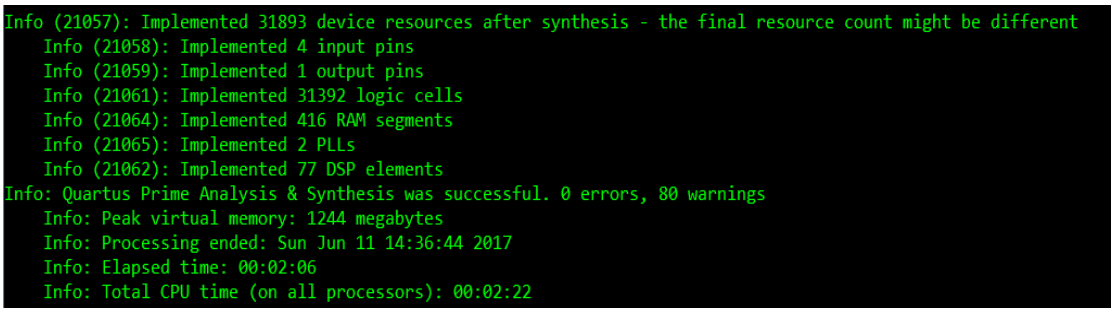
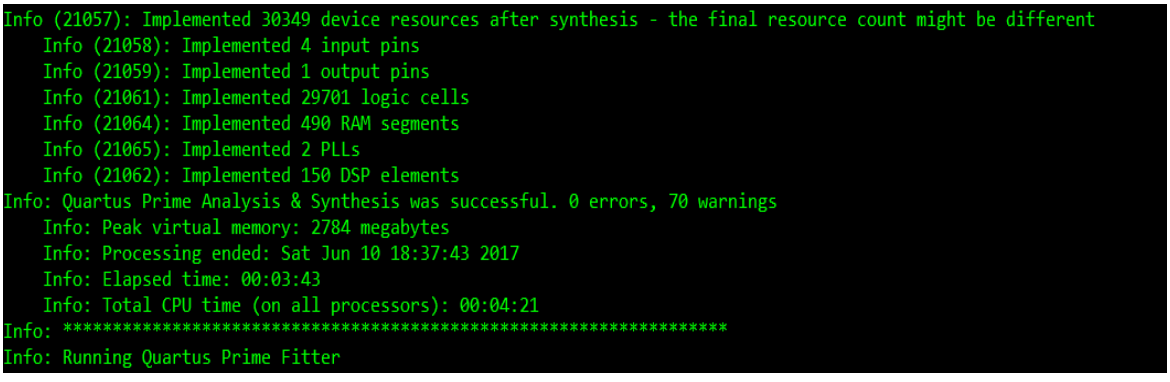
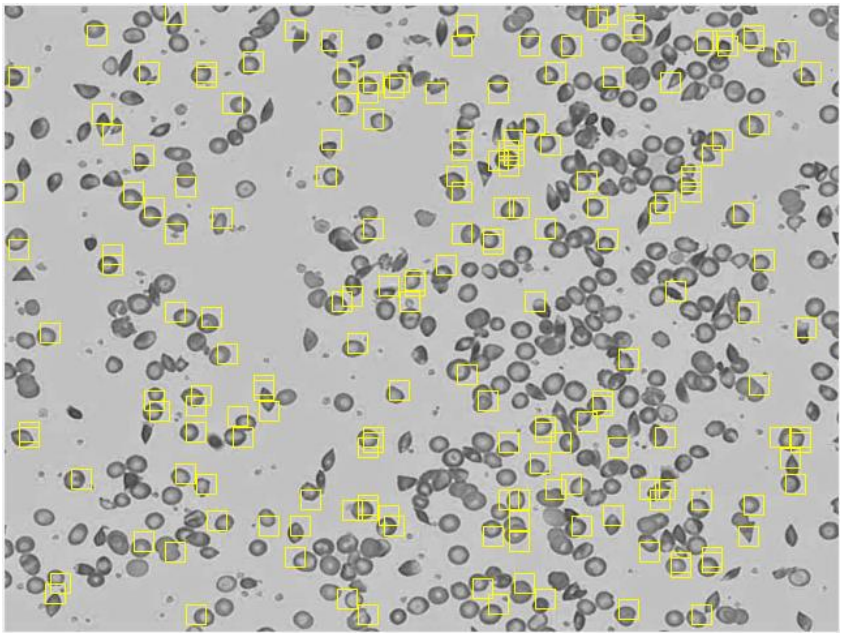


Figura C.2 Resumen de recursos Quartus modelo 2



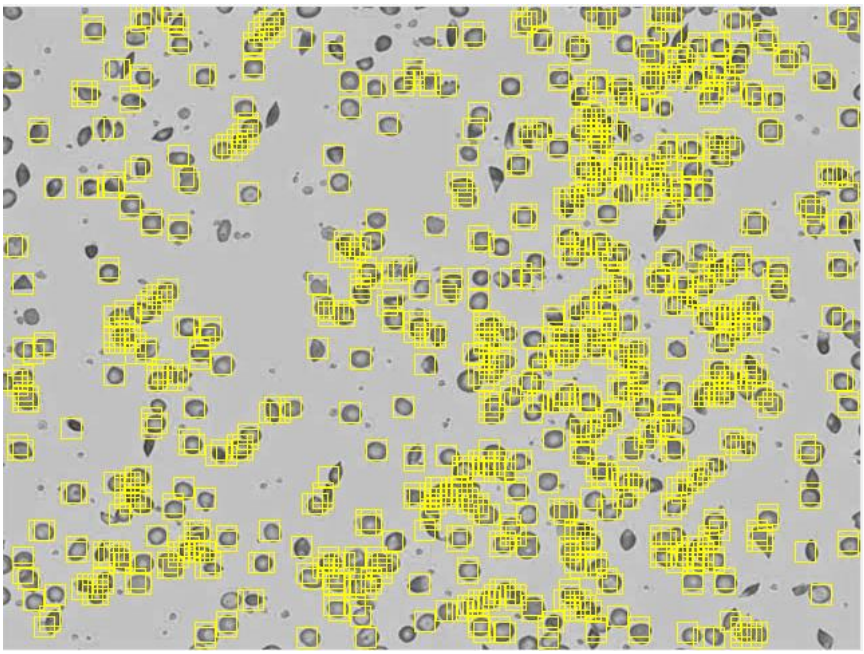
La Figura C.3 muestra el resultado de ejecutar el modelo 2 con las imágenes de muestras sanguíneas. Esta corresponde a la clasificación de hematíes con anormalidades. Se observa que el modelo es capaz de detectar algunas de los hematíes con poiquilocitosis, sin embargo, posee un alto grado de error, ya que también incluye en este grupo a un gran número de células sanas.

Figura C.3 Resultado del modelo 2 clasificación hematies con poiquilocitosis



La Figura C.4, al igual que la Figura C.3 muestra el resultado del modelo 2. Esta imagen corresponde a la identificación de células sanas de muestras sanguíneas. Se observa que el modelo ofrece un gran nivel de precisión de clasificación de células sanas, a pesar de incluir algunas células con anormalidades, logra identificar todas las células sanas de la imagen.

Figura C.4 Resultado del modelo 2 clasificación hematíes sanos



Referencias y Bibliografía

[1] A. Loddo, L. Putzu, C. Di Ruberto, and G. Fenu, “A Computer-Aided System for Differential Count from Peripheral Blood Cell Images,” in *2016 12th International Conference on Signal-Image Technology & Internet-Based Systems (SITIS)*, 2016, pp. 112–118.

[2] Q. B. Baker and K. Balhaf, “Exploiting GPUs to accelerate white blood cells segmentation in microscopic blood images,” in *2017 8th International Conference on Information and Communication Systems (ICICS)*, 2017, pp. 136–140.

[3] R. S. Choras, “Image feature extraction techniques and their applications for CBIR and biometrics systems,” *Int. J. Biol. Biomed. Eng.*, vol. 1, no. 1, pp. 6–16, 2007.

[4] A. Wiliem, Y. Wong, C. Sanderson, P. Hobson, S. Chen, and B. C. Lovell, “Classification of Human Epithelial type 2 cell indirect immunofluoresence images via codebook based descriptors,” *Proc. IEEE Work. Appl. Comput. Vis.*, pp. 95–102, 2013.

[5] A. Suyyagh and G. Abandah, “FPGA Parallel Recognition Engine for Handwritten Arabic Words,” *J. Signal Process. Syst.*, vol. 78, no. 2, pp. 163–170, 2013.

[6] P. G. Fernández, A. García, J. Ramírez, and A. Lloris, “Fast RNS-based 2D-DCT computation on field-programmable devices,” *IEEE Work. Signal Process. Syst. SiPS Des. Implement.*, pp. 365–373, 2000.

[7] Y. ARAI, T. AGUI, and M. NAKAJIMA, “A Fast DCT-SQ Scheme for Images,” *IEICE Trans.*, vol. E71–E, no. 11, pp. 1095–1097, 1988.

[8] S. Bouguezel, M. O. Ahmad, and M. N. S. Swamy, “A novel transform for image compression,” *2010 53rd IEEE Int. Midwest Symp. Circuits Syst.*, pp. 509–512, 2010.

[9] A. A. Del Barrio, S. O. Memik, M. C. Molina, J. M. Mendias, and R. Hermida, “A Distributed Controller for Managing Speculative Functional Units in High Level Synthesis,” *IEEE Trans. Comput. Des. Integr. Circuits Syst.*, vol. 30, no. 3, pp. 350–363, Mar. 2011.

[10] Matlab, “HDL Coder Features,” 2015. [Online]. Available: https://es.mathworks.com/help/hdlverifier/index.html?s_tid=srchtitle. [Accessed: 20-Mar-2017].

[11] Matlab, “Vision HDL toolbox,” 2015. [Online]. Available:

https://es.mathworks.com/products/vision-hdl/features.html. [Accessed: 20-Mar-2017].

[12] Matlab, “Neural Network Toolbox,” 2015. [Online]. Available: https://es.mathworks.com/products/neural-network/features.html. [Accessed: 22-Mar-2017].

[13] Matlab, “HDL Verifier,” 2015. [Online]. Available: https://es.mathworks.com/help/hdlverifier/index.html?s_tid=srchtitle. [Accessed: 20-Mar-2017].

[14] D. Fariña, G. Botella, and A. A. Del Barrio, “A DCT AND NEURAL NETWORK BASED SYSTEM TO OBTAIN THE CHARACTERISTICS OF BIOLOGICAL IMAGES,” *Summer Simul. Multi-Conference,* 2017.

[15] C. Tablada and G. Torres, “Redes Neuronales Artificiales,” *Rev. Educ. Matemática,* pp. 22–30, 2009.

[16] J. M. Marín Diazaraque, “Introducción a las redes neuronales aplicadas,” *Man. Data Min.,* pp. 1–31, 2007.

[17] “Dispositivos FPGA.” [Online]. Available: https://www.intel.la/content/www/xl/es/fpga/devices.html. [Accessed: 18-Jun-2017].

[18] A. Corporation, “Cyclone V Device Overview CV-51001 Key Advantages of Cyclone V Devices,” 2016.

[19] Altera, “Adaptive Logic Module (ALM),” 2015. [Online]. Available: http://quartushelp.altera.com/15.0/mergedProjects/reference/glossary/def_alm.htm. [Accessed: 10-May-2017].

[20] S. Feedback and S. Jose, “Cyclone V Device Handbook,” vol. 1, 2015.

[21] Altera, “Hardened DSP Blocks in FPGAs,” 2015. [Online]. Available: https://www.altera.com/solutions/technology/dsp/overview.html. [Accessed: 10-May-2017].

[22] “About Place and Route.” [Online]. Available: http://quartushelp.altera.com/14.1/mergedProjects/comp/comp/comp_about_place.htm. [Accessed: 16-Jun-2017].