ISSN 1870-4069

Análisis de funcionamiento de una red neuronal implementada sobre una tarjeta SoC Zynq

Alberto Martínez Contreras, Juan Iván Díaz Reyes,
Alam Armando Minor González, David Tinoco Varela

Universidad Nacional Autónoma de México, FESC, Departamento de Ingeniería,
Ingeniería en Telecomunicaciones Sistemas y Electrónica (ITSE),
México

{phama_contra26,elec_st,dativa19}@hotmail.com, teeniisgool@gmail.com

Resumen. Hoy en día el uso de dispositivos tales como las FPGA, ha tomado fuerza para el diseño de diferentes proyectos tecnológicos y computacionales, estos dispositivos tienen características que las hacen atractivas para la ejecución de diferentes esquemas de desarrollo. En este trabajo se presenta la implementación y análisis de una red neuronal tipo Backpropagation sobre una tarjeta de desarrollo SoC Zynq®-7000. La placa de trabajo consta de un elemento FPGA y una unidad de procesamiento, los cuales pueden emplearse de manera autónoma. Se generaron los modelos neuronales basados en el lenguaje de descripción VHDL. Cuando se ha tenido la red neuronal completamente descrita dentro de la FPGA, se han realizado experimentos de clasificación de imágenes ejecutándose en ambos dispositivos (FPGA y unidad de procesamiento), comparando y analizando la eficiencia y velocidad de ejecución en cada uno de ellos. Se utiliza la tarjeta de desarrollo PYNQ-Z1 de Digilent, ya que proporciona los recursos necesarios para cumplir los objetivos del proyecto.

Palabras clave: Redes neuronales, FPGA, tarjetas de desarrollo.

Functional Analysis of a Neural Network Implemented using a Zynq SoC Board

Abstract. Nowadays the use of devices such as FPGA, have had relevance for the design of different technological and computational projects, these devices have characteristics that make them attractive for the execution of different development schemes. In this paper, the implementation and analysis of a Backpropagation-type neural network on a Zynq®-7000 SoC development board is presented. The development board consists of an FPGA element, and a processing unit, both can be used autonomously. Neural models based on the description language VHDL were generated. When the neuronal network has been completely described within the FPGA, image classification experiments have been carried out on both devices (FPGA and processing unit), comparing and analyzing the efficiency and the execution time in each one of them. The Digilent PYNQ-Z1 development board is used, as it provides the necessary re-courses to meet the project's objectives.

Keywords: Neural network, FPGA, development boards.

1. Introducción

Las redes neuronales artificiales, o clasificadores conexionistas, son sistemas de computación masivamente paralelos que se basan en modelos simplificados del cerebro humano. Sus complejas capacidades de clasificación, combinadas con propiedades tales como generalización, tolerancia a fallas y aprendizaje, las hacen atractivas para una gama de aplicaciones en las que las computadoras convencionales encuentran dificultades de cálculo. Ejemplos de estos incluyen detección de rostros [1], reconocimiento de caracteres escritos a mano [2], clasificación de patrones [3], y tareas de control [4].

Las redes neuronales actuales son utilizadas cada vez más en diferentes áreas del conocimiento [5], lo que implica la necesidad de sistemas computacionales cada vez más potentes tanto en procesamiento como en almacenamiento. Como es de imaginarse, no es barato conseguir un equipo de cómputo con los requerimientos necesarios para ejecutar una red neuronal de gran potencia, por lo que una alternativa económica es de suma importancia para el desarrollo e investigación de este campo.

En este proyecto se buscó una alternativa para poder diseñar una red neuronal en paralelo sin equipos muy costosos, esta opción es usar un SoC que combina una unidad de procesamiento y un FPGA (*Field-Programmable Gate Array*) de alta gama que nos permite hacer diseños propios usando un lenguaje de descripción de *hardware*, se empleara un SoC Zynq®-7000 de *Xilinx*, con la tarjeta de desarrollo PYNQ-Z1. En la figura 1, podemos observar la tarjeta sobre la cual se han realizado los experimentos.

Se utilizó la FPGA de la placa para implementar una red neuronal en hardware y verificar las características que presenta este tipo de redes con respecto a una red neuronal implementada sobre una unidad de procesamiento, también embebida en la tarjeta. La red neuronal implementada, fue probada con diferentes imágenes para poder medir la velocidad de procesamiento tanto de la FPGA como del microprocesador de la placa PYNQ-Z1.

Fig. 1. Placa PYNQ-Z1 de *Digilent*, las características, especificaciones y una descripción completa de la tarjeta puede verificarse en el manual [21].

2. Estado del arte

Las redes neuronales se comenzaron a desarrollar desde el siglo pasado cuando *Warren McCulloch* y *Walter Pitts* [6] crearon un modelo computacional en 1943 para redes neuronales, basado en matemáticas y algoritmos, llamado lógica de umbral.

En 1958, el concepto del *perceptrón* fue mostrado por *Rosenblatt* [7], este fue uno de los más importantes conceptos en este campo, ya que representaba un modelo matemático de una neurona biológica que podía ser o no activada de acuerdo a los valores de sus entradas. Este modelo podía resolver problemas que tuvieran un comportamiento linealmente dependiente, sin embargo problemas como la resolución de una función de tipo *Or-Exclusive*, no puede ser resuelta por este modelo. La representación básica del perceptrón puede verse en la figura 2.

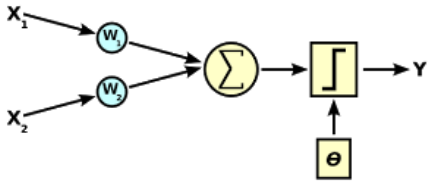


Fig. 2. Representación de un perceptrón con diferentes entradas con función de activación de tipo escalón.

Cuando los primeros conceptos relacionados a las redes neuronales se publicaron, se tuvieron dificultades con su implementación, principalmente por el hecho de que en ese momento no existían ordenadores lo suficientemente potentes en procesamiento y memoria para poder ejecutar una red neuronal compleja. Estas dificultades mantuvieron detenidos los avances relacionados a este campo de conocimiento, sin embargo, un puno clave para un renovado interés en las redes neuronales y el aprendizaje fue el algoritmo *Backpropagation* de *Werbos* [8] que podía resolver eficazmente el problema de la or exclusiva , y en términos más generales, logró acelerar el entrenamiento de redes neuronales multicapa.

Otro gran factor para el desarrollo de las *Redes Neuronales Artificiales* (RNA) fue que a mediados de la década de 1980, el procesamiento distribuido paralelo se hizo popular bajo el nombre de conexionismo. *Rumelhart* y *McClelland* describieron el uso del conexionismo para simular procesos neuronales [9].

A partir de 2011, las técnicas de redes de aprendizaje progresivo alternaron capas convolucionales y capas de máximo aprovechamiento [10, 11] encabezadas por varias capas totalmente, o escasamente conectadas, seguidas por una capa de clasificación final.

Por otro lado, se ha tenido un gran avance tecnológico en el desarrollo de microcomputadoras y distintos dispositivos embebidos, entre ellos los FPGA, que cuentan con suficiente poder para ejecutar estos algoritmos de formas distribuidas y aceleradas.

Recientemente, las RNA se han implementado con FPGA reconfigurables. Estos dispositivos combinan la facilidad de ser programados, con una mayor velocidad de operación asociada al paralelismo de hardware.

Se han generado varias herramientas de diseño de FPGA en la última década, entre las que se encuentran la *Intel FPGA SDK* para *OpenCL* [12] y Xilinx SDSoC [13], lo que ha provocado una reducción en los costos de producción y adquisición de este tipo de dispositivos.

Existen gran cantidad de experimentos y aplicaciones montadas sobre FPGAs, sin embargo, debido a la línea principal de este artículo, solamente se mencionan algunas de ellas, relacionadas al desarrollo de aceleradores y de implementación neuronal.

Recientemente, en el año 2017 [14], *Zhao* y otros autores implementaron un clasificador de tipo red neuronal binaria (BNN, por sus siglas en inglés) en una placa de desarrollo FPGA de bajo costo (*ZedBoard*) y mostraron, según sus propias palabras, mejoras en comparación con las líneas de base de CPU (*Central Processing Unit*) y GPU (*Graphics Processing Unit*) incorporadas, así como con los aceleradores de FPGA existentes.

En su trabajo, *Zhao* et al. [14] evaluaron el diseño en un ZedBoard, que utiliza un Xilinx Zynq-7000 SoC de bajo costo que contiene un FPGA XC7Z020 junto con un procesador integrado ARM Cortex-A9. Compararon su diseño con dos plataformas informáticas de servidor: un procesador multinúcleo (CPU) Intel Xeon E5-2640 y una GPU NVIDIA Tesla K40 (GPU). Según los autores, ellos fueron los primeros en implementar un acelerador para redes neuronales binarias en FPGA.

Por otro lado, *Jin, Gokhale* y otros autores [15] presentaron una implementación en tiempo real de redes neuronales convolucionales profundas (DCNN, por sus siglas en inglés) acelerada por *hardware*. Su sistema fue implementado en una plataforma Xilinx Zynq-7000.

En [16], *Lysaght, Stockwood* y otros autores describen la implementación de una RNA en una matriz de compuerta programable de campo *Atmel AT6005* (FPGA). En propias palabras de los autores, "el trabajo se llevó a cabo como un experimento en el mapeo de una aplicación lógicamente intensiva a nivel de bit en los recursos lógicos específicos de un FPGA de grano fino. Al explotar las capacidades de reconfiguración del FPGA de Atmel, las capas individuales de la red se multiplexan en el tiempo en la matriz lógica. Esto permite implementar una RNA más grande en una sola FPGA a expensas de una operación más lenta del sistema en general".

Gadea, Cerdá y otros autores [17] describen la implementación de una matriz sistólica para un perceptrón multicapa en una FPGA Virtex XCV400 con un algoritmo de aprendizaje amigable con el *hardware*. Ellos muestran una adaptación segmentada del algoritmo de tipo *Backpropagation* en línea. Según los autores, el paralelismo se explota mejor porque las fases hacia adelante y hacia atrás se pueden realizar simultáneamente.

Según este grupo de científicos, las simulaciones de *software* son útiles para investigar las capacidades de los modelos de redes neuronales y crear nuevos algoritmos; pero las implementaciones de *hardware* siguen siendo esenciales para aprovechar al máximo el paralelismo inherente de las redes neuronales.

Venieris, y Bouganis [18] presentaron *fpgaConvNet*, un marco para mapear redes neuronales convolucionales en FPGA.

Según su evaluación experimental, fpgaConvNet ofrece predicciones de rendimiento bastante precisas y logra mejoras en la densidad y la eficiencia del rendimiento en comparación con los trabajos existentes de FPGA y GPU incrustado.

En [19] se presentaron tres aceleradores de hardware para RNN en Zynq SoC FPGA de Xilinx para mostrar cómo superar los desafíos involucrados en el desarrollo de aceleradores RNN. En sus experimentos, el hardware produjo con éxito texto de *Shakespeare* utilizando un modelo de nivel de personaje.

Una aplicación interesante de la mezcla de RNA y FPGA se da en [20], en este trabajo se propone un sistema de clasificación de gas en tiempo real de baja latencia, que utiliza una red neuronal artificial de perceptrón multicapa (MLP) para detectar y clasificar los datos del sensor de gas por medio de un MLP (*Multi Layer Perceptron*) paralelo, implementado en una plataforma de sistema sobre chip SoC de Xilinx.

Para un resumen más amplio relacionado a las implementaciones dentro de sistemas embebidos y FPGAs, pueden visualizarse las referencias [23-26].

Como se puede ver a través de esta sección, se han generado una gran cantidad de tarjetas de desarrollo, entre ellas, las tarjetas FPGA. Tales dispositivos embebidos, han sido de utilidad para la implementación de diferentes modelos de redes neuronales, con la intención de mejorar el rendimiento y/o acelerar los procesos de cálculo. En el presente trabajo, se ha tomado como punto de partida la tarjeta de desarrollo PYNQ-Z1 de *Digilent* que incluye una FPGA y una unidad de procesamiento. Se ha buscado implementar un mismo modelo neuronal tanto en la unidad de procesamiento como en la FPGA de la placa, con la finalidad de poder determinar cuál de los dos dispositivos presenta una mejor eficiencia temporal, y definir las principales diferencias de una implementación en *hardware* con respecto a la implementación en *software*.

3. Conceptos básicos

3.1. FPGA

Un FPGA (*Field Programable Gate Array*) es un dispositivo semiconductor que contiene componentes lógicos programables (CLP) e interconexiones programables entre ellos. Los CLP pueden ser programados para duplicar la funcionalidad de puertas lógicas básicas tales como AND, OR, XOR, NOT o funciones combinacionales más complejas tales como decodificadores o simples funciones matemáticas.

En muchos FPGA, los CLP (o bloques lógicos, según el lenguaje comúnmente usado) también incluyen elementos de memoria, los cuales pueden ser simples *flip-flops* o bloques de memoria más complejos. La figura 3, muestra un esquema básico de un FPGA. Estos dispositivos son programados por medio de lenguaje VHDL.

3.2. VHDL

VHDL es un lenguaje de especificación definido por el IEEE (*Institute of Electrical and Electronics Engineers*), bajo el esquema ANSI/IEEE 1076-1993, utilizado para describir circuitos digitales y para la automatización de diseño electrónico. VHDL es acrónimo proveniente de la combinación de dos acrónimos: VHSIC (*Very High Speed Integrated Circuit*) y HDL (*Hardware Description Language*). Aunque puede ser usado de forma general para describir cualquier circuito digital, se usa principalmente para programar PLD (*Programmable Logic Device*), FPGA (*Field Programmable Gate Array*), ASIC (*Application-specific Integrated Circuit)* y similares.

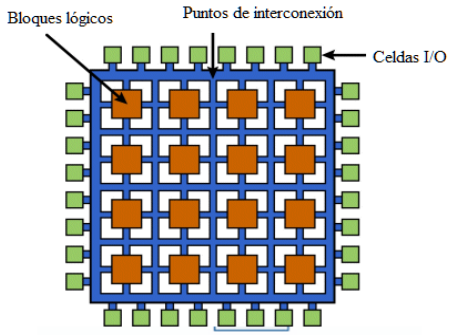


Fig. 3. Estructura interna de un FPGA: donde se muestran los bloques lógicos que pueden ser configurables para generar la función a realizar.

Originalmente, el lenguaje VHDL fue desarrollado por el departamento de defensa de los Estados Unidos a inicios de los años 80 basado en el lenguaje de programación *Ada* con el fin de simular circuitos eléctricos digitales.

Posteriormente se desarrollaron herramientas de síntesis e implementación en hardware a partir de los archivos VHD. En la siguiente sección se presentan algunos detalles extra de este tipo de lenguaje de programación, haciendo hincapié en su importancia dentro del proyecto.

3.3. Neuronas artificiales con VHDL

Antes de crear una red en VHDL, se deben analizar las características y ventajas que nos proporciona este lenguaje de descripción. La primera característica, y la más importante, es que a diferencia de los lenguajes de programación estándar que necesitan compilarse o interpretarse en instrucciones que un microprocesador pueda ejecutar, una descripción con VHDL se convierte directamente en hardware, este dinamismo nos permite crear módulos que trabajen intrínsecamente en forma paralela.

Un ejemplo sencillo para ver las ventajas que nos proporciona VHDL sería hacer dos contadores, que cada tiempo t_x incrementen su valor en uno, usando un lenguaje de programación clásico como C, primero se aumentaría el valor de un contador, y posteriormente del otro, pero nunca los dos a la vez, pues las instrucciones en C están siendo ejecutadas por un microprocesador que solamente puede hacer una tarea a la vez, pero usando VHDL, nosotros podemos diseñar dos circuitos independientes que hagan la tarea, en pocas palabras, los dos contadores pueden incrementar su valor al mismo tiempo pues diseñamos dos circuitos iguales que trabajan en paralelo y que no dependen uno del otro para poder ejecutarse.

Ahora que sabemos las ventajas que nos puede proporcionar VHDL, se realiza un modelo de red neuronal que pueda aprovechar estas ventajas, para hacer este diseño se debe pensar en las neuronas que componen la red como pequeños procesadores que tienen todos los recursos necesarios para trabajar en su arquitectura y no los comparten.

El proceso que hace una neurona para obtener un valor en su salida es multiplicar cada entrada por el valor de su peso y sumarlos todos, después sumar el offset y a esta sumatoria aplicar una función sigmoidea, escalón, rampa o algún otra, en este caso se

ha utilizado la función rampa, por ser más fácil implementarla en VHDL, pero puede utilizarse cualquier función deseada.

En este esquema tenemos que p_i son las entradas de la neurona, estas entradas pueden estar conectadas a un valor específico o la salida de otra neurona, es importante que una entrada esté conectada únicamente a un elemento, de lo contrario se podría producir un comportamiento inesperado, w_i son los pesos y cada entrada tiene el propio, estos valores son los que se irán modificando para hacer que la red neuronal converja, b es el offset, este valor también puede ser modificado por la neurona para ajustar su salida, y le permite tener una salida diferente de cero, aun cuando todas sus entradas sean cero, esta descripción está dada en la ecuación (1):

$$a = f(n) = f\left(\sum_{i=0}^{i=q}(p_i * w_i) + b\right). \tag{1}$$

Cabe mencionar que los pesos y el *offset* son valores aleatorios generados dentro de un rango y asignados a los enlaces la primera vez que se ejecuta una red.

Ahora veremos el proceso de corrección de los pesos, para poder acercarse al resultado que esperamos. Donde W_{ni} son los nuevos pesos, W_{ai} son los pesos anteriores, I_i es el valor de la respectiva entrada, E es el valor del error y f el factor de aprendizaje, lo podemos ver en la ecuación (2):

$$W_{ni} = W_{ai} + (I_i * E * f). \tag{2}$$

Esta fórmula de corrección es aplicada a cada uno de los pesos en los enlaces donde la neurona tenga una entrada valida. Para corregir el offset se utiliza la ecuación (3). Donde O_n es el nuevo offset calculado, O_a es el offset actual, E es el valor del error y f el factor de aprendizaje:

$$O_n = O_a - (E * f). \tag{3}$$

La corrección de pesos y offset se hace tantas veces como sea necesario, hasta obtener en la salida un resultado aceptable, pero este proceso de corrección únicamente funciona para una neurona que tenga conectadas las entradas y salidas directamente, pero no será efectivo si se tiene una red neuronal multicapa con varias neuronas conectadas entre sí.

3.4. Redes multicapa

En una red multicapa es importante entender que las neuronas de salida, son a las que se les indica el valor esperado y pueden compararlo con el valor actual y corregir sus pesos, pero a las neuronas de capas anteriores no les podemos indicar cuál es el valor esperado, este valor se lo deben dar las neuronas que tengan conectadas a la salida.

Las neuronas de la última capa, después de corregir sus pesos y offset, calculan que valor para cada una de sus entradas podría beneficiales, después la proporcionan según el peso que le dan a esta entrada, y este valor lo pasan como valor ideal a la neurona que tienen conectada en esa entrada.

Cuando una neurona de capa intermedia recibe todos los valores de corrección de las neuronas de la capa de salida, los pondera y decide a que valor deberá aproximarse, calcula su error, corrige sus pesos y nuevamente hace el proceso anterior para poder pasar a la capa anterior un valor de corrección, y así sucesivamente hasta que el valor de corrección llegue a la capa que tiene conectadas las entradas.

4. Implementación de red

Tomando en cuenta las ecuaciones y procesos explicados anteriormente, se implementó el siguiente algoritmo dentro de la FPGA considerando cada neurona como una unidad de procesamiento individual, capaz de generar la salida y corrección de errores por sí misma, los cambios en la salida o en el valor de error de la neurona se consideran como eventos que se ejecutan de forma concurrente al percibir un cambio y no como funciones que se ejecutan cada vez que se las llama.

Evento 0: Creación de la neurona. Este proceso lo lleva a cabo únicamente una vez cuando se comienza la ejecución.

1. Poner con valor de cero la salida y la señal de error.
2. Inicializa con un valor aleatorio o precargado los pesos y el offset.
3. Inicializa con un valor aleatorio o precargado el factor de aprendizaje.
4. Queda a la espera de percibir un cambio en las entradas.

Evento 1: Cambio en la entrada. Este proceso se ejecuta de forma concurrente, cada vez que la neurona detecta un cambio en sus entradas.

1. Multiplica cada entrada por su respectivo peso.
2. Realiza la sumatoria de todas las entradas y suma el valor del offset.
3. Aplica la función programada, puede ser rampa, sigmoidea o escalón.
4. Cambia el valor de la salida.

Evento 2: Cambio en la señal de error. Este proceso se ejecuta de forma concurrente cada vez que la neurona detecta un cambio en la señal de error.

1. Calcula el valor esperado considerando todas las neuronas que tiene a su salida.
2. Calcula el valor de error comparando el valor esperado contra el obtenido.
3. Hace la corrección de los pesos, ya sea aumentándolos o disminuyéndolos en proporción al error que aporto cada entrada y el factor de aprendizaje.
4. Hace la corrección del offset.
5. En base al error obtenido, hace el cálculo del valor ideal en cada entrada y lo pasa a la neurona en esa entrada.

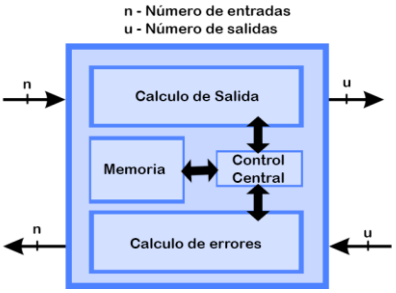


Fig. 4. Modelo de una neurona en una FPGA.

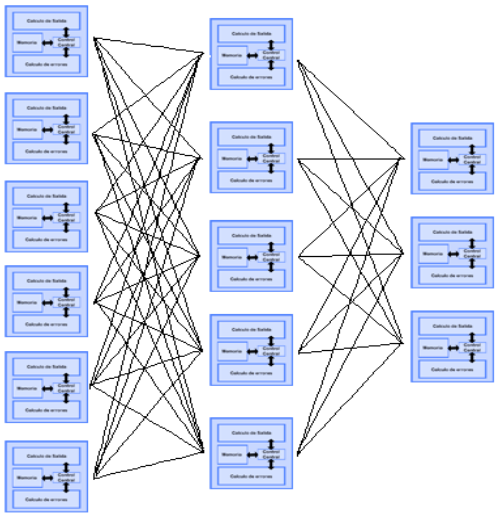


Fig. 5. Modelo de la red dentro de la FPGA.

En base a este algoritmo, el diseño de la neurona seria parecido al mostrado en la figura 4.

Ahora que se tiene el diseño de cada neurona individual, se procede a conectarlas entre sí, en VHDL se vería cada neurona como un módulo o procesador que comparte entradas y salidas con otros semejantes comunicándose entre sí, parecido al diseño del tejido neuronal de un ser vivo, pero con muchas menos conexiones y capacidad.

Antes de implementar el algoritmo para conectar las neuronas entre sí, se debe tomar en cuenta la imposibilidad de ejecutar en paralelo todas las neuronas en la red, sólo las de la misma capa, pues la capa dos depende del resultado obtenido de la capa uno, la capa tres depende del resultado de la capa dos y así sucesivamente. Una vez que todas las capas se ejecuten y obtengamos el error, tendremos el mismo problema, pero a la inversa pues el error lo calcula la última capa, y lo pasa a la capa anterior, está a la anterior y así sucesivamente hasta que todas las capas conozcan su parte de error y puedan corregirlo. El algoritmo de la red neuronal completa se describe a continuación:

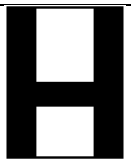
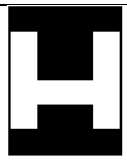
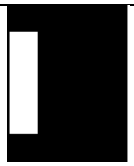
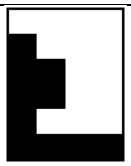
	Patrón	Salida 1	Salida 2	Salida 3
Prueba1		85%	-	-
Prueba2		75%	-	-
Prueba3		30%	-	-
Prueba4		97%	-	-
Prueba1		80%	-	-
Prueba2		73%	-	-
Prueba3		26%	-	-
Prueba4		45%	-	-
Prueba1		-	70%	-
Prueba2		-	54%	-
Prueba3		-	76%	-
Prueba4		-	69%	-
Prueba1		-	45%	-
Prueba2		-	32%	-
Prueba3		-	58%	-
Prueba4		-	47%	-
Prueba1		-	-	60%
Prueba2		-	-	76%
Prueba3		-	-	54%
Prueba4		-	-	65%
Prueba1		-	-	90%
Prueba2		-	-	87%
Prueba3		-	-	92%
Prueba4		-	-	86%

Fig. 6. Patrones de prueba que se ejecutaron en la implementación dentro de la FPGA, donde se muestran tres columnas de salida, cada una de ellas representando una salida distinta. Cada una de estas salidas de la red expresa el porcentaje que representa el parecido del patrón de entrenamiento por salida.

1. Se ejecuta en paralelo la capa uno, que está conectada a las entradas de la red.
2. Se ejecuta en paralelo la capa dos de la red con los valores provenientes de la capa uno.
3. Se ejecuta en paralelo la capa tres de la red, con los valores obtenidos de la capa dos.
4. Se compara la salida obtenida en la capa tres con la salida que se espera obtener.

5. La capa tres ejecuta en paralelo el cálculo del error entre la salida esperada y la obtenida, corrige sus pesos y pasa el error a la capa dos.

6. La capa dos recibe el error de la capa tres, corrige sus pesos y pasa el valor de error a la capa uno.

7. La capa uno recibe el error de la capa dos y corrige sus pesos de ser necesario.

8. Comienza nuevamente el flujo de ejecución hasta que el error sea mínimo.

El modelo implementado se puede observar en la figura 5.

4.1. Pruebas realizadas en la red neuronal

La primera prueba realizada, es una prueba muy simple que solamente consiste en la identificación de caracteres especiales. Con una red que está conectada a una imagen representada por una matriz de 24 posiciones, cada posición puede contener un 1 que se representa con color negro, o un -1 que se representa con color blanco, la capa de salida tuvo tres neuronas, por lo que puede reconocer tres patrones. La red neuronal fue entrenada con tres patrones base: A, I, E. Con estos patrones se obtuvo una respuesta satisfactoria de su ejecución, ya que lograba clasificar correctamente la mayoría de los casos de prueba. En la figura 6 podemos observar el comportamiento de esta red neuronal cuando ingresamos diferentes imágenes matriciales para poder reconocer o identificar el matriz origen. Este primer experimento solamente fue realizado para verificar el funcionamiento de la red en su implementación en FPGA.

Un segundo experimento fue realizado para la verificación y clasificación de diferentes imágenes, y por medio de este experimento, analizar el funcionamiento y ejecución de una implementación neuronal en un FPGA y en una unidad de procesamiento.

Para comprobar las ventajas que nos proporciona un modelo implementado en hardware contra uno implementado en software, compararemos ambos modelos en la tarjeta de desarrollo PYNQ-Z1 que tiene un dispositivo SoC Zynq®-7000, será utilizado el modelo FINN que implementa una red neuronal binarizada programable en Phyton para clasificación de imágenes.

El modelo que usaremos esta previamente entrenado para clasificar imágenes. Sera implementado en el procesador del SoC, que es un Dual-Core ARM® Cortex®-A9, y también en el FPGA Artix-7 de la misma comparando ambos resultados.

De acuerdo a las observaciones relacionadas a la tabla 1, vemos que los tiempos de clasificación de las distintas imágenes son diferentes, debido principalmente a los tamaños de cada una de ellas, sin embargo, a pesar de las diferencias en tamaños, podemos notar que el promedio de tiempo de clasificación del software sobre el hardware es 1300 veces más rápido en todos los casos.

Tabla 1. Tiempos de clasificación de diferentes imágenes, ejecutadas dentro del FPGA y dentro del procesador.

Muestra	Tiempo total	Tiempo por imagen
1 (hardware)	.0063 seg	.00063 seg
1 (software)	8.32 seg	.83245 seg
2 (hardware)	.0062 seg	.00062 seg
2 (software)	8.34 seg	.83459 seg
3 (hardware)	.0061 seg	.00061 seg
3 (software)	8.35 seg	.83562 seg
4 (hardware)	.0064 seg	.00064 seg
4 (software)	8.34 seg	.83412 seg
5 (hardware)	.0067 seg	.00067 seg
5 (software)	8.28 seg	.82894 seg

Tabla 2. Clasificaciones correctas y erróneas de las imágenes base, modificadas con características diferentes tales como colores, fondos, iluminación, y direccionamientos aleatorios.

Muestra	Número de imágenes	Muestras correctamente clasificadas.	Muestras erróneamente clasificadas
1	10	8	2
2	10	9	1
3	10	8	2
4	10	7	3
5	10	8	2

Tabla 3. Características de las imágenes analizadas con la red neuronal para verificar su nivel de exactitud.

Características de las imágenes probadas.	Resultado
Colores estandar.	Correctamente clasificada.
Imágenes con elementos visuales agregados.	Incorrectamente clasificada.
Cambio de color en las imágenes.	Incorrectamente clasificada.
Imágenes con pequeñas variacion de colores.	Correctamente clasificada.
Imágenes en mabientes cargados de elementos extras.	Incorrectamente clasificada.

Es importante mencionar que en el modelo implementado en la PYNQ no se hace corrección de errores, simplemente se cargan los pesos pre programados.

Esta red fue sometida a ejemplos distorsionados y con características diferentes (como colores y fondos aleatorios) de cada una de las imágenes base. En la tabla 2, podemos observar el número de muestras correctamente clasificadas y el número de muestras no clasificadas correctamente por medio de la RNA.

En basa a la tabla 2, los principales factores que generan una clasificación errónea son el cambio de color de la imagen base, la orientación de la imagen (dependiendo del grado de orientación o inclinación, la red puede identificarla o no identificarla), y cuando hay muchos elementos extra, tales como fondos aleatorios o caracteres adicionales sobre la muestra. La iluminación también ha jugado un papel interesante, ya que si se cambia la iluminación de la imagen, la red confunde los colores y los identifica como colores diferentes a la muestra, generando una mala clasificación. La tabla 3, describe algunos ejemplos de imágenes correctamente clasificadas y erróneamente clasificadas.

5. Conclusiones

En este trabajo se ha presentado la implementación de una red neuronal dentro de una FPGA, con esta implementación se han obtenido las métricas relacionadas al tiempo y a la eficiencia de clasificación de imágenes de la red neuronal. Las pruebas realizadas dentro de la FPGA, han sido realizadas también dentro de una unidad de procesamiento (Ambas, FPGA y unidad de procesamiento, embebidas dentro de una tarjeta de trabajo Soc Zynq de Xilinx) con la intención de comparar sus velocidad y eficiencia, logrando observar que la implementación en la FPGA es, por mucho, más rápida que la ejecución cobre la unidad de procesamiento.

A pesar de que las diferencias en tiempo de ejecución entre la unidad de procesamiento y la FPGA, la clasificación de las imágenes llevada a cabo dentro de la unidad de procesamiento, tiene la misma eficiencia que la llevada a cabo dentro de la FPGA.

Las FPGA son dispositivos que permiten una ejecución de algoritmos intrínsecamente paralela, esto permite que las redes neuronales artificiales, puedan potenciar su tiempo de ejecución cuando son adecuadamente implementadas, sumado a esta característica, está el hecho de su bajo coste y gran accesibilidad, logrando posicionar a las FPGA como dispositivos bien capacitados para la investigación relacionada a inteligencia artificial y redes neuronales.

6. Trabajo futuro

Como trabajo futuro, se buscará implementar una misma red en diferentes tipos de sistemas embebidos y diferentes tipos de tarjetas de desarrollo de *Digilent*, para poder definir en cual, de toda esta gama de posibilidades existentes, se tiene una mejor respuesta tanto temporal como de cálculo, sumando al estudio una comparativa en relaciones tiempo de ejecución-precio.

Se plantea la generación de un *Cluster* realizado con diferentes FPGA, para verificar su funcionamiento en este esquema.

Agradecimientos. Este trabajo fue en parte financiado por el proyecto PAPIIT IN 113316 y el proyecto PIAPI 1634, de la UNAM.

Referencias

1. Curran, K.; Li, X.; McCaughley, N.: Neural network face detection. The Imaging Science Journal, 53(2), pp. 105–115 (2005)
2. Patil, V.; Shimpi, S.: Handwritten English character recognition using neural network. Elixir Comput Sci Eng, 41, pp. 5587–5591 (2011)
3. Bartlett, P. L.: The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network. IEEE transactions on Information Theory, 44(2), pp. 525–536 (1998)
4. Åkesson, B. M.; Toivonen, H. T.: A neural network model predictive controller. Journal of Process Control, 16(9), pp. 937–946 (2006)
5. Timotheou, S.: The random neural network: a survey. The computer journal, 53(3), pp. 251–267 (2010)
6. McCulloch, W.; Walter, P.: A Logical Calculus of Ideas Immanent in Nervous Activity. Bulletin of Mathematical Biophysics. 5(4), pp. 115–133 (1943)
7. Rosenblatt, F.: The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. Psychological Review, 65(6), pp. 386–408 (1958)
8. Werbos, P. J.: Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences. PhD thesis, Harvard University (1975)
9. Rumelhart, D. E; McClelland, J.: Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Cambridge: MIT Press (1986)
10. Ciresan, D. C.; Meier, U.; Masci, J.; Gambardella, L. M.; Schmidhuber, J.: Flexible, High Performance Convolutional Neural Networks for Image Classification. International Joint Conference on Artificial Intelligence (2011)
11. Martines, H.; Bengio, Y.; Yannakakis, G. N.: Learning Deep Physiological Models of Affect. IEEE Computational Intelligence, 8(2), pp. 20–33 (2013)
12. Czajkowski, T. S.; Aydonat, U.; Denisenko, D.; Freeman, J.: Kinsner, M; Neto, D.; Wong, J.; Yiannacouras, P.; Singh D. P.: From OpenCL to High-Performance Hardware on FPGAs. In: Int'l Conf. on Field Programmable Logic and Applications (FPL), pp. 531–534 (2012)
13. Kathail, V.; Hwang, J.; Sun, W.; Chobe, Y.; Shui, T.; Carrillo, J.: SDSoC: A Higher-level Programming Environment for Zynq SoC and Ultrascale+ MPSoC. In: Int'l Symp. On Field-Programmable Gate Arrays (FPGA), pp. 4–4 (2016)
14. Zhao, R.; Song, W.; Zhang, W.; Xing, T.; Lin, J. H.; Srivastava, M.; Zhang, Z.: Accelerating binarized convolutional neural networks with software-programmable FPGAs. In: Proceedings of the 2017 (ACM/SIGDA), International Symposium on Field-Programmable Gate Arrays, pp. 15–24 (2017)
15. Jin, J.; Gokhale, V.; Dundar, A.; Krishnamurthy, B.; Martini, B.; Culurciello, E.: An efficient implementation of deep convolutional neural networks on a mobile coprocessor. In: Circuits and Systems (MWSCAS), IEEE 57th International Midwest Symposium on, pp. 133–136 (2014)
16. Lysaght, P.; Stockwood, J.; Law, J.; Girma, D.: Artificial neural network implementation on a fine-grained FPGA. In: International Workshop on Field Programmable Logic and Applications, pp. 421–431, Springer (1994)
17. Gadea, R.; Cerdá, J.; Ballester, F.; Mocholí, A.: Artificial neural network implementation on a single FPGA of a pipelined on-line backpropagation. In: Proceedings of the 13th international symposium on System synthesis, IEEE Computer Society, pp. 225–230 (2000)
18. Venieris, S. I.; Bouganis, C. S.: A framework for mapping convolutional neural networks on FPGAs. In: Field-Programmable Custom Computing Machines (FCCM), IEEE 24th Annual International Symposium on, pp. 40–47 (2016)

19. Chang, A. X. M., Culurciello, E.: Hardware accelerators for recurrent neural networks on FPGA. In: Circuits and Systems (ISCAS), IEEE International Symposium on, pp. 1–4 (2017)
20. Zhai, X.; Ali, A.; Amira, A., Bensaali, F.: MLP neural network based gas classification system on Zynq SoC. IEEE Access, 4, pp. 8138–8146 (2016)
21. Digilent PYNQ-Z1: Board Reference Manual. https://reference.digilentinc.com /_media/reference/programmable-logic/pynq-z1/pynq-rm.pdf
22. PYNQ: Python Productivity for ZYNQ. Available: http://www.pynq.io/
23. Guo, K.; Zeng, S.; Yu, J.; Wang, Y.; Yang, H.: A Survey of FPGA Based Neural Network Accelerator. arXiv preprint arXiv:1712.08934 (2017)
24. Zhu, J.; Sutton, P.: FPGA implementations of neural networks a survey of a decade of progress. In: International Conference on Field Programmable Logic and Applications, pp. 1062–1066 Springer (2003)
25. Misra, J.; Saha, I.: Artificial neural networks in hardware: A survey of two decades of progress. Neurocomputing, 74(1-3), pp. 239–255 (2010)
26. Liu, J.; Liang, D.: A survey of FPGA-based hardware implementation of ANNs. In: Neural Networks and Brain, (ICNN&B'05) IEEE International Conference on, 2, pp. 915–918 (2005)