Review of Prominent Strategies for Mapping CNNs onto Embedded Systems

M. Arredondo-Velázquez, J. Diaz-Carmona, *Member, IEEE,* A. Barranco-Gutiérrez, *Member, IEEE,* and C. Torres-Huitzil, *Senior, IEEE*

***Abstract*—Convolutional neural networks (CNN) have turned into one of the key algorithms in machine learning for content classification of digital images. Nevertheless, the CNN computational complexity is considerable larger than classic algorithms, thus, CPU- or GPU-based platforms are generally used for CNN implementations in many applications, but often do not fulfill portable requirements due to resources, energy and real-time constrains. Therefore, there is a growing interest on real time processing solutions for object recognition using CNNs mainly implemented on embedded systems, which are limited both in resources and energy consumption. An updated review of prominent reported approaches for mapping CNNs onto embedded systems is described in this paper. Two main solutions trends for reducing the hardware CNN workload are distinguished through a deduced taxonomy. One is focused on algorithm level solutions to reduce the number of multiplications and CNN coefficients. On the other hand, hardware level solutions goal is to achieve processing time, power consumption and hardware resources reduction. Two dominant hardware level design strategies are pointed out as oriented to either reducing the energy consumption and resources utilization meeting real-time requirements or increasing the throughput at the expense of resources utilization. Finally, two identified design strategies for CNN hardware accelerators are proposed as opportunity research areas.**

***Index Terms*—Convolutional neural networks (CNN), Deep learning, Embedded systems, Field programmable gate arrays (FPGAs), Hardware accelerators, Layer operation chaining, Machine learning, Single computation engine, Streaming architectures.**

I. Introducción

LAS Redes Neuronales de Convolución (RNC) son uno de los algoritmos claves dentro del área del aprendizaje profundo (AP) ya que tienen la capacidad de aprender representaciones numéricas de imágenes digitales en diferentes niveles de abstracción [1]. Desde el surgimiento del AP en 2006 [2], las RNCs se han convertido en un tema de interés para los investigadores debido a la precisión que puede ser obtenida en tareas de clasificación [3]. En décadas pasadas las técnicas clásicas de reconocimiento de objetos consistían en seleccionar la técnica más apropiada para la descripción de una imagen como lo son los *Patrones Locales Binarios* [4], *Descriptores de Haar* [5], [6], *Análisis de los componentes principales* [7], *Momentos de Hu* [8], *Histograma orientado a gradientes* [9], [10], *Transformación de características invariante a la escala* [11], [12], etc. Sin embargo, estas técnicas están limitadas en la capacidad de obtener información de los datos de entrada, ya que se requiere que la técnica sea cuidadosamente seleccionada y ello depende principalmente de la habilidad y experiencia del diseñador [13].

En el 2012 [14], se demostró en el evento *Imagenet Large Scale Visual Recognition Challenge* (ILSVRC) [15] que una RNC tiene el potencial de aumentar la precisión en tareas de clasificación con respecto a los métodos de clasificación tradicionales, ya que se alcanzó una precisión top-5[1] de 84.7%. A partir de entonces las RNCs han incrementado constantemente su precisión en la clasificación de imágenes mediante la propuesta de nuevas topologías [16], logrando obtener precisiones top-5 en el ILSVRC de 88.87% [17], 93.3% [18], 96.4% [19] y 97.7% [20] en los años 2013, 2014, 2015 y 2017, respectivamente. Esta precisión obtenida supera las capacidades de reconocimiento un ser humano [21]. Por ello, las RNCs se han convertido en una de las mejores soluciones a problemas de aprendizaje complejos como los encontrados en la visión artificial.

Actualmente, la tendencia general es explorar RNCs más profundas, lo cual se traduce en una mayor complejidad computacional, ya que requieren realizar una gran cantidad de operaciones (mayor a mil millones por imagen) sobre una gran cantidad de pesos (mayor a 50 millones) [17], [18], [19], [22]. Esta complejidad aunada al uso de un mayor número de capas y conexiones irregulares tiene como consecuencia el surgimiento de desafíos significativos en el diseño de aceleradores hardware. En algunos casos el procesamiento en la nube a través de servidores locales puede ser una solución siempre y cuando se encuentre disponible una conexión a la red. Sin embargo, la transmisión de datos a la nube o incluso a otro dispositivo en muchas ocasiones no es práctico debido a factores como la potencia de transmisión, el volumen de datos, la latencia, la confiabilidad y la privacidad [23]. Por otro lado, el procesamiento local permite que los dispositivos

Esta investigación fue financiada por CONACYT, número de concesión 434864.

M.A. Velázquez Instituto Tecnológico de Celaya, Tecnológico Nacional de México, México, Av. Tecnológico y G. Cubas, s/n, 38010 Celaya, GTO, Mexico. email: D1603003@itcelaya.edu.mx.

J.D. Carmona Instituto Tecnológico de Celaya, Tecnológico Nacional de México, México, Av. Tecnológico y G. Cubas, s/n, 38010 Celaya, GTO, Mexico. email: javier.diaz@itcelaya.edu.mx.

I.A.B. Gutiérrez Instituto Tecnológico de Celaya, Tecnológico Nacional de México, México, Av. Tecnológico y G. Cubas, s/n, 38010 Celaya, GTO, Mexico. email: israel.barranco@itcelaya.edu.mx.

C.T. Huitzil Tecnologico de Monterrey, Escuela de Ingeniería y Ciencias, Campus Puebla, Av. Atlixcayotl 5718, Puebla C.P. 72453 Puebla, Mexico. email: torresc@tec.mx.

Correspondence author: Moisés Arredondo Velázquez.

[1]El error top-5 es medido considerando si la respuesta correcta se encuentra dentro de las 5 probabilidades mas altas obtenidas por el algoritmo de clasificación

funcionen con mayor eficiencia y cuenten con una mayor autonomía, lo que reduce la carga de trabajo y el ancho de banda para los servidores locales [24]. Debido a ello, existe una creciente demanda de arquitecturas hardware eficientes que puedan solventar los problemas originados de estas nuevas RNCs [25]. En este sentido, los investigadores han propuesto una serie de soluciones a los principales inconvenientes de las RNCs, los cuales son: 1) El elevado número de operaciones en la capas de convolución, que representan el 90% de las operaciones totales de la RNC [26], [27]. 2) El gran ancho de banda requerido principalmente en la etapa de clasificación. Si los datos provienen de memorias externas, la transferencia de datos aumenta el consumo de energía; llegando incluso a ser mayor que la energía requerida para los cálculos. [28], [29], [30].

En este artículo se presenta una revisión de las estrategias más representativas sobre implementación en hardware de RNCs. Dichas estrategias se describen de acuerdo a la taxonomía propuesta de soluciones para el alto peso computacional en RNCs mostrada en la Figura 1. En esta clasificación destacan las soluciones a nivel algoritmo que emplean diferentes técnicas enfocadas en atenuar el alto peso computacional mediante la reducción del número de multiplicaciones y/o coeficientes de la RNC. Por otro lado, las soluciones hardware están orientadas a reducir ya sea el número de recursos, el consumo de energía o el tiempo de ejecución de la RNC. Dentro de este último enfoque se identifican como principales tendencias: las arquitecturas hardware con un convolucionador único (AHCU) y las arquitecturas hardware con múltiples convolucionadores (AHMC). A partir de las tendencias hardware identificadas y el análisis de sus soluciones propuestas se describen las estrategias de diseño que aún están poco exploradas y que tienen el potencial de ofrecer ventajas tanto en utilización de recursos como de aceleración del tiempo de procesamiento. Las áreas de oportunidad identificadas representan tendencias de diseño diferentes a las que existen actualmente.

El resto del documento está organizado de la siguiente manera. En la sección II se describen de manera general los conceptos básicos de RNCs, así como algunas de sus aplicaciones. En la sección III se detallan las técnicas para reducir el peso computacional, ésto como soluciones a nivel de algoritmo. La sección IV describe las estrategias de diseño para la implementación hardware de RNCs. Después, en la sección VI se comentan dos principales áreas de oportunidad en el diseño de aceleradores para RNCs. Finalmente, la sección VII concluye este artículo.

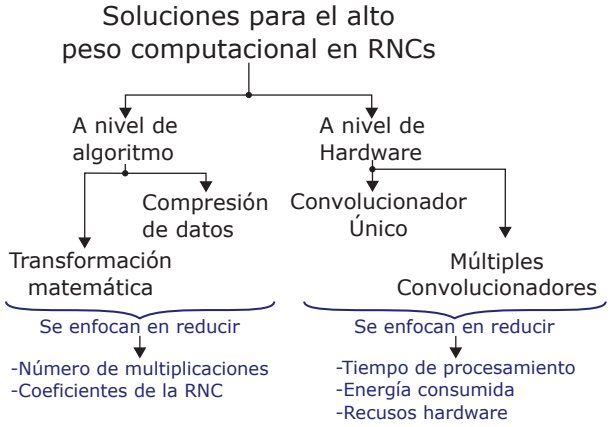


Fig. 1. Taxonomía propuesta de soluciones para la implementación de RNCs sobre hardware.

II. RNCs

En los últimos años, el número de aplicaciones que usan RNCs ha crecido considerablemente. Dentro de éstas se encuentran la localización de objetos en imágenes [31], [32], [33], [34], [35], [36], seguimiento de objetos [37], transferencia de conocimiento [38], vehículos autónomos [39], [40], reconocimiento óptico de caracteres (OCR) [41], [42], estimación de pose [43], [44], [45], reconocimiento de rostros [46], medicina [47], etc. Las RNC empleadas en estas aplicaciones se dividen en tres categorías 1) RNCs con capas subsecuentes [48], [14], [34], [17], [49] 2) RNCs con capas paralelas [18] y RNCs con conexiones residuales [50], [22]. Sin importar estas diferencias, una RNC está compuesta típicamente por capas convolucionales y una etapa de clasificación que consiste en una Red Completamente Conectada (RCC), las cuales son descritas a continuación.

A. Capas de Convolución

Una RNC está conformada por N_l capas convolucionales y cada una de ellas recibe como entrada un conjunto de mapas de características $\mathcal{F} \in \mathbb{R}^{M^{(l)} \times N^{(l)} \times CH^{(l)}}$, los cuales son procesados por múltiples kernels $\mathcal{K} \in \mathbb{R}^{w^{(l)} \times w^{(l)} \times CH^{(l)} \times K^{(l)}}$ para generar un nuevo conjunto de mapas de características $\mathcal{G} \in \mathbb{R}^{M^{(l+1)} \times N^{(l+1)} \times K^{(l)}}$, expresamente $f(\mathcal{F} \otimes \mathcal{K}) = \mathcal{G}$. El superíndice $l \; \forall \; 1, 2, \ldots, N_l$ denota la capa convolucional a la que pertenece la variable. Los hiper-parámetros asociados a la capa l son: número de filas en la imagen ($M^{(l)}$), columnas ($N^{(l)}$), mapas de características ($CH^{(l)}$), kernels ($K^{(l)}$) y tamaño del kernel ($w^{(l)}$). La dimensión de cada elemento involucrado en una capa de convolución se ilustra en la Figura 2.

Una función de activación $f(x)$ es empleada para limitar la amplitud de los valores de salida de la capa convolucional a un rango determinado, siendo la función ReLU $f(x) = max(x, 0)$ la más utilizada en RNCs. Dependiendo de la configuración de la RNC, una capa convolucional puede estar asociada a un componente de agrupamiento, donde la

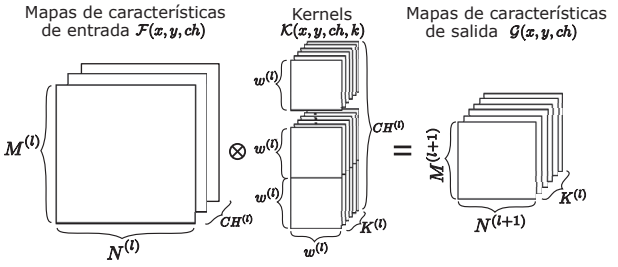


Fig. 2. Dimensión de los elementos involucrados en la operación de convolución.

dimensión de la salida de la capa de convolución puede ser reducida a través del cálculo del valor máximo o promedio de diferentes regiones de la imagen.

B. Etapa de Clasificación

Los mapas de caracteristicas ($\mathcal{G}$) obtenidos en la capa N_l son una representación de la imagen de entrada a la RNC y son generalmente clasificados por medio de una RCC. Una RCC está compuesta por N_{lf} capas y l representa el índice de la capa donde $1 < l < L_{N_{lf}}$. El número de neuronas en la capa l es S_l y la capa N_{lf} tiene C neuronas correspondientes al número de clases de la RCC. La activación de la $i-esima$ neurona ubicada en la capa l está representada por $a_i^{(l)} = f(z_i^{(l)})$, donde $z_i^{(l)}$ es la suma ponderada de los valores de datos de entrada a la neurona $x_1, x_2, \ldots, x_n = \mathbf{x}$ más un término adicional $x_0 = 1$, por lo tanto $z_i^{(l)} = x_0 w_{i0}^{(l)} + x_1 w_{i1}^{(l)} + \ldots + x_n w_{in}^{(l)}$. El término adicional x_0 está ponderado por $w_{i0}^{(l)}$, la adición de este término permite que $f(z_i^{(l)})$ sea una función no lineal.

III. Soluciones a Nivel de Algoritmo

De manera general, las plataformas basadas en CPU o GPU son usadas para implementar RNCs en muchas aplicaciones, ya sea por sus capacidades de cómputo o debido a la existencia de numerosas librerías que permiten compartir los coeficientes de redes previamente entrenadas como *Caffe*, *Theano*, *Keras* o *Tensorflow*, entre otras [51]. No obstante, prevalece la necesidad de soluciones hardware eficientes particularmente en sistemas embebidos. En esta sección se presentan soluciones destacadas a nivel de algoritmo enfocadas en reducir el alto peso computacional en RNCs.

A. Transformación Matemática

Una tendencia habitual es modificar el algoritmo convencional a través de una transformación matemática para así obtener una reducción del número de multiplicaciones en las capas convolucionales. Dentro de estas trasformaciones destacan las siguientes.

1) FFT: La transformada rapida de Fourier (*FFT*) ([52], [53]) es un conocido método que puede ser empleado para reducir el número de multiplicaciones en la operación de convolución, ya que, a comparación del método clásico donde el número de multiplicaciones es $\mathcal{O}(N_{out}^2 w^2)$, la convolución mediante *FFT* solo requiere $N_{out}^2 log_2 N_{out}$ multiplicaciones. Obteniendo la *FFT* del mapa de características de entrada $\mathbf{F}$ y del filtro $\mathbf{K}$ es posible calcular la convolución mediante el producto en el dominio de la frecuencia de $\mathbf{F} \times \mathbf{K}$ y posteriormente obtener el resultado final en el dominio del espacio mediante la FFT inversa. Sin embargo, existen algunos inconvenientes en el uso de la *FFT*, uno de ello es que solo se muestra una pequeña reducción en la complejidad del cómputo con filtros pequeños como es el caso de 3×3 [54]. Debido a ello, algunas librerías especializas en plataformas GPU tales como *cuDNN* [55] y *fbfft* [56] eligen de forma dinamica el algoritmo apropiado para el tamaño de filtro dado (por ejemplo, *FFT* para $w = 5$ y *Winograd* [57] para $w = 3$). Esto para acelerar el proceso de entrenamiento e inferencia.

2) Transformada Winograd: En [57] se evalua la *FFT* y la *Transformada Winograd* (TW) ([58]) en plataformas GPU, y se reporta que la TW es una mejor elección en filtros pequeños (3×3). En [59] se propone una arquitectura basada en TW para implementar la convolución en FPGA, se hace uso de una estructura con líneas de retardo para habilitar el re-uso de datos entre los diferentes componentes de procesamiento. Con ello, se demuestra que dicho algoritmo puede reducir la complejidad aritmética, además de mejorar el rendimiento de las RNCs en FPGAs. De manera similar, en [60] se aplica la TW en una plataforma *Arria* 10, con lo cual se logra una mejora significativa en el rendimiento.

En estos trabajos, la TW ha sido empleada para realizar la convolución 2D. Sin embargo, en [61] se implementa este algoritmo para realizar convoluciones 3D para una RNC empleada en la clasificación de videos, los resultados son comparados con los obtenidos en una implementación GPU utilizando la librería *cuDNN*, donde se obtiene el doble de aceleración del tiempo de procesamiento. Una de las mayores desventajas que presenta la TW es que se requiere un algoritmo diferente para cada una de las posibles dimensiones del filtro de convolución. Debido a ello, nuevas variantes de esta trasformada siguen siendo propuestas [62].

3) Descomposición de valores singulares (DVS): La descomposición de valores singulares (DVS) consiste en factorizar una matriz real o compleja, con lo cual se puede reducir la dimensión de una de las variables implicadas en la convolución y con ello lograr un costo computacional menor. En [63] se demuestra que las capas en la etapa de clasificación requieren un mayor ancho de banda, así que se utilizó el método DVS en esa etapa para poder reducir su complejidad y la cantidad de memoria necesaria para almacenar los coeficientes. Para implementar la convolución 2D de manera eficiente en [64] se propone una nueva técnica llamada *Aproximación de la Descomposición en Valores Singulares* (ADVS). El método ADVS descompone la convolución 2D en pares de convoluciones de una dimensión de menor complejidad mediante la aproximación *low rank* [65]. los resultados experimentales muestran que este esquema puede reducir la complejidad del hardware en un rango de entre 14.46% a 37.8% dependiendo del tamaño de kernel de convolución.

4) Convolución como multiplicación de matrices: Una de las alternativas que se han explorado para implementar la convolución es mapear esta operación en una multiplicación de matrices usando la forma de *matriz Toeplitz*. En [66] se implementa un acelerador en FPGA basado en *OpenCL*, donde las convoluciones 3D en las capas de convolución son mapeadas como multiplicación de matrices mediante el reordenamiento de los mapas de entrada, obteniendo con ello una redución del número de operaciones. Otro trabajo es el reportado en [67], donde se describe una representación uniforme para las capas de convolución y las capas en la etapa de clasificación. Dicha representación consiste en una multiplicación de matrices para procesar ambos tipos de capas en la que además es posible compartir los mismos recursos de cómputo. Sin embargo, en esta transformación no se considera la optimización potencial en las operaciones de convolución y el tiempo de procesamiento obtenido al realizar

la transformación puede ser considerable elevado. Esta técnica es ampliamente utilizada en plataformas GPU, pero en FPGAs esto puede resultar en un uso de memoria excesivo.

B. Reducción de Datos

Un enfoque diferente consiste en emplear métodos que permitan reducir la cantidad de memoria necesaria para almacenar los coeficientes de una RNC. Las técnicas más importantes de compresión de datos se describen a continuación:

1) Baja precisión en la representación de datos: Los números en punto fijo son una elección común para representar los parámetros de una red neuronal. Sin embargo, éstos pueden contener información redundante [68]. Para reducir dicha redundancia, se hace uso de las *Redes Neuronales Binarizadas* (RNB) para restringir algunas operaciones aritméticas involucradas en procesar los valores de salida. Hay tres aspectos en la binarización de redes neuronales: entradas binarias, pesos de sinapsis binarios, y activaciones de salida binarias. En [69] se considera una binarización completa con una porción predeterminada de sinapsis, teniendo un valor de cero y todas las demás sinapsis con un valor de uno. La red opera utilizando solo la operación lógica XNOR y contadores de bits. El uso de este método reporta una precisión del 98.7% utilizando la base de datos *MNIST*. La RNC binarizada XNOR-Net [70] que está inspirada en las topologías AlexNet, ResNet y GoogLeNet, reporta una precisión top-1 del 51.2% con binarización completa y del 65.5% con una binarización parcial utilizando la base de datos Imagenet.

2) Compresión de datos: Algunos métodos utilizados son la cuantización vectorial [71] y el *hashing* [72]. La cuantización vectorial consiste en mapear un conjunto grande de datos a un conjunto más pequeño [73]. En [74] se demuestra la presencia de información redundante en los parámetros de las redes neuronales, por ello se emplea cuantización vectorial para reducir de manera significativa el número de los parámetros dinámicos en topologías de RNCs con muchas capas. En [75] se encuentra que con los métodos de cuantización vectorial se obtiene una clara ganancia sobre los métodos existentes de factorización de matrices. Además, se concluye que los métodos de cuantización vectorial estructurada, como la cuantización de productos, permiten un mejor rendimiento en comparación a los métodos de cuantización escalar [76] y cuantización residual [77].

Diseñar una técnica *hashing* apropiada para acelerar el entrenamiento en RNCs o ahorrar espacio en la memoria también es un reto interesante. El uso de *HashedNets* [78] permite reducir el tamaño del modelo de una red neuronal mediante el uso de la función *Hash* para la agrupación de las conexiones de sinapsis. Las conexiones agrupadas entre sí comparten un solo parámetro, de esta manera se reduce la cantidad de datos a procesar mientras se preserva la generalización en el reconocimiento de imágenes. En [79] se propone un método basado en diccionarios que codifica las convoluciones en un gran conjunto de datos que se utiliza para el proceso de entrenamiento. Los datos obtenidos del diccionario son compartidos entre todas las capas de la red, esto permite que la red aprenda con un menor número de muestras. Con este método se puede alcanzar una mejor precisión en un menor número de iteraciones comparado con las RNCs tradicionales.

3) Conexiones escasas: Una alternativa a la compresión de datos es la técnica denominada *Pruning* [80], [81]. Ésta reduce el número de parámetros y de operaciones en la RNC con la eliminación permanente de las conexiones menos importantes [82], [83], lo cual permite tener redes de menor complejidad manteniendo un rendimiento competitivo. En [84] se introduce el método *fine-grained sparsity*, en el cual cada conexión entre neuronas puede ser eliminada en base la magnitud absoluta de cada peso. En [85] extienden este método basado en la magnitud para permitir la restauración de los pesos eliminados en iteraciones previas durante el entrenamiento. En [86] se propone remover los filtros que generan un valor de cero en las funciones de activación en el conjunto de validación. En [87] se plantea una nueva representación de los filtros basandose en la redundancia *inter-canal* e *intra-canal* presente en los kernels de convolución, con lo cual se logra eliminar el 90% de las conexiones de la RNC. Por otro lado, en [88] se propone el método *Structured Sparsity Learning* (SSL) para optimizar de manera simultanea los hiper-parámetros tales como la dimensión del filtro y su conectividad local.

IV. Soluciones a Nivel de Hardware

Desde la perspectiva hardware, el diseño de las arquitecturas está enfocado ya sea a reducir el número de accesos a memoria externa con el objetivo de aminorar el consumo energético, o bien acelerar la operación de convolución mediante el uso de distintos tipos de paralelismo. Por un lado, el cuello de botella se encuentra en el acceso a memoria, cada multiplicación y acumulación requiere tres lecturas de memoria (coeficiente del filtro, dato del mapa de características, y la suma parcial) y una escritura (actualización de la suma parcial). En Alexnet se requieren 724 millones de multiplicaciones y acumulaciones, lo que implica alrededor de 3000 millones de accesos a memoria, lo cual tiene un gran impacto en el rendimiento y la energía consumida puesto que los accesos a memoria externa demandan niveles de energía de orden mucho mayor a los accesos a memoria interna [89].

Debido a que no existe aleatoriedad en el procesamiento de una RNC, es posible diseñar un flujo de datos que se pueda adaptar a diferentes topologías de RNCs y mejorar así la eficiencia en términos de energía por medio de la reducción del número de accesos a memoria. En RNCs existen diferentes formas de flujo de datos, como los son: Reuso de los pesos (WR), Reuso de las entradas (IR), Reuso de las salidas (OR) [90]. Cabe destacar que no son los únicos patrones de computo reportados, sin embargo se considera que son los más representativos.

Por otro lado, existen diferentes estrategias que pueden ayudar a acelerar los cálculos en una RNC, estas estrategias consisten en realizar de manera paralela cada una de las operaciones presentes en una RNC. Dentro de las capas convolucionales se identifican cuatro tipos de paralelismo que pueden ser explotados, los cuales son: 1) Paralelismo de operación (PO) que se refiere al número de multiplicadores (nm) empleados en una convolución entre un kernel y una

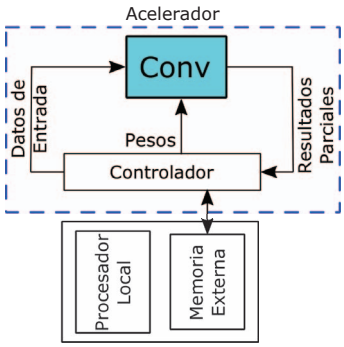


Fig. 3. Sistema acelerador de operación de convolución mediante un solo convolucionador.

imagen de tamaño $w \times w$. 2) Paralelismo de lazo (PL), en el cual es posible procesar nf_i mapas de entrada y nf_o kernels de forma paralela, por ello este tipo de paralelismo se divide en dos: *paralelismo intra-output* y *paralelismo inter-output*, respectivamente [91]. 3) Paralelismo de datos (PD) donde nv máscaras de convolución se procesan en paralelo. Finalmente, 4) Paralelismo de capa (PC) donde más de una capa se procesa a la vez. Cabe destacar que el tipo de paralelismo disponible está directamente relacionado con el algoritmo a emplear [92] y [93].

No obstante, los trabajos reportados sobre acceleradores hardware para RNCs se han realizado a través de dos distinguidas estrategias de diseño, las cuales son descritas a continuación:

A. Arquitecturas Basadas en un Solo Convolucionador

El gran número de capas en las RNCs representa un desafió al mapear una red completa a un solo chip, ya que la cantidad de recursos hardware necesarios puede ser mayor a los disponibles en un dispositivo. Por ello, muchos trabajos se han enfocado en diseñar sistemas que aceleren capa por capa de manera individual por medio de AHCU [94], [95], [29], [96], [97], [98], [99]. Dichas capas se ejecutan secuencialmente y la salida producida por la capa actual es almacenada y leída en memoria externa previo a procesar la siguiente capa. La Figura 3 muestra un sistema de aceleración típico usado en algunos trabajos. El sistema consiste en un único convolucionador ($Conv$), una memoria dinámica externa y un procesador.

Debido a que el procesamiento esencial de una RNC es realizado en las capas de convolución, las AHCU son una de las principales tendencias. Esta estrategia de diseño se enfoca en desarrollar componentes flexibles para operar sobre diferentes hiper-parámetros. Técnicas tales como arreglos sistólicos de elementos de procesamiento [100], [24] o árboles de multiplicadores se utilizan para lograr dicha flexibilidad.

Recientemente, los FPGAs han ganado popularidad debido al gran rendimiento obtenido al implementar aceleradores hardware para tareas de AP. Algunas soluciones en FPGA se enfocan en implementar las convoluciones utilizando el método convencional para realizar la convolución [64], donde se calcula la convolución 2D tomando una ventana de la imagen de entrada y multiplicando cada uno de los elementos por su parámetro correspondiente (resultando en n^2 multiplicaciones) para posteriormente obtener la sumatoria mediante un árbol de sumadores (n^2 sumas). Las propuestas [101], [102], [94], [103], [104] hacen uso de dicho método convencional inspirándose en [105], siendo en [101] donde se utilizó por primera vez en una RNC.

Otro ejemplo de una arquitectura aceleradora RNC programable y flexible se presenta en [95]. Su principal componente computacional es una matriz de elementos de procesamiento (PE), donde cada PE consiste de un banco de multiplicadores, un árbol sumador y una ruta de agrupación opcional. Esta propuesta utiliza una estrategia de cuantificación de datos para reducir el número de bits a ocho con una pérdida de precisión insignificante. Otro enfoque relacionado se reporta en [106], donde la reutilización de datos se emplea encadenando diversos operadores dentro de la capa de convolución tal como la función de activación y el agrupamiento, lo que resulta en un procesamiento con un número reducido de accesos a la memoria externa. Para lograr flexibilidad, los operadores de hardware se agrupan en un módulo denominado colección. El operador de convolución dentro de cada colección se puede configurar para realizar en paralelo una convolución de máscara 12×12, cuatro convoluciones de máscara 6×6 o dieciséis convoluciones de máscara 3×3. Esta idea de flexibilidad también fue adoptada en [107] y [104]. La ruta de datos reconfigurables descrita en la arquitectura propuesta en [90] permite el uso de un patrón híbrido de reutilización de datos para diferentes tamaños de capa, incluida la reutilización de datos de entrada, núcleo de convolución y resultados parciales.

En [63] se implementa un acelerador en FPGA descrito en VERILOG para la RNC VGG. Para el diseño de la arquitectura, el acelerador emplea un *buffer* de entrada programable y un número fijo de multiplicadores debido a que el tamaño de las ventanas de convolución también es fijo. Además, se hace uso de PD donde se procesa en paralelo un número fijo de filtros. Por otra parte, el patrón de almacenamiento de datos en las capas de convolución y en la etapa de clasificación esta optimizado para reducir la latencia de acceso a memoria. Finalmente, en la etapa de clasificación se reduce el número de pesos mediante DVS. En [94] se implementa un acelerador FPGA empleando herramientas de síntesis en alto nivel [108], explotando el PD, el convolucionador recibe T_m mapas de características y produce T_m salidas lo que corresponde al número de kernels que se procesan de manera paralela. El PO se realiza mediante un árbol de multiplicadores, donde se utilizan buffers especiales para el reuso de datos. Adicionalmente, se propone una exploración en el espacio de diseño para optimizar el rendimiento considerando los recursos disponibles y ancho de banda mediante la herramienta *Roofline Model* [109]. Para este último trabajo no se consideran los componentes de agrupamiento ni la etapa de clasificación. Un trabajo similar se presenta en [110], en el que se muestra el diseño de un componente de convolución genérico considerando la exploración de espacio de diseño, obteniendo con ello diversas soluciones. En una de ellas, cada capa es optimizada de manera independiente, una más donde el número de mutliplicadores T_k es fijo para cada una de las

capas de convolución y finalmente, una donde el número de mapas de entrada T_n al convolucionador, el número de filtros paralelos T_m y el número de multiplicadores son fijos para el procesamiento de todas las capas de convolución. En el trabajo presentado en [111] se propone un acelerador de propósito general empleando el método de partición de kernel empleando los métodos de paralelismo *intra-kernel* e *inter-kernel* para acelerar las capas de convolución. En este método se divide el kernel de convolución original en pequeños sub-kernels para eliminar el traslape entre las diferentes posiciones de la máscara de convolución.

Todos estos trabajos pueden lograr un rendimiento sobresaliente preservando la baja utilización de recursos. Sin embargo, el tamaño de la máscara de convolución varía para cada capa, lo que provoca una disminución de la eficiencia del componente.

B. Arquitecturas Basadas en Múltiples Convolucionadores

Otra estrategia de diseño se basa en la premisa de que los FPGA modernos poseen un mayor número de recursos computacionales y mayor velocidad de memoria que en el pasado, lo que proporciona un mayor espacio de diseño. Debido a ésto, los investigadores pueden aprovechar el PC mediante el uso de AHMC (Figura 4). Este otro enfoque corresponde a la segunda estrategia de diseño principal. En ésta, un convolucionador está dedicado a ejecutar cada capa de una RNC dada, por ello su diseño es altamente optimizado para su capa correspondiente. Por lo tanto, se procesa más de una capa en paralelo con una latencia fija y cada capa procesa información de diferentes imágenes de entrada. Este método se explota en [93], donde los convolucionadores asociados se pueden configurar para adaptarse a las RNCs más utilizadas. Esto es posible empleando un método novedoso para determinar el grado de paralelismo óptimo para cada capa de convolución, el cual consiste en un análisis de diferentes tipos de paralelismo que se pueden explotar en capas convolucionales. En el trabajo [16], se emplea a una serie de PEs con los que es posible procesar múltiples imágenes a través de la arquitectura. Los componentes básicos que se emplean consisten en unidades de convolución y agrupación, así como estructuras basadas en ventanas deslizantes que proporcionan la funcionalidad de almacenamiento. Esta propuesta emplea bloques de hardware especializados para mapear redes con conexiones irregulares [18], [19], [22]. La asignación de recursos en la fase de exploración del espacio de diseño, ajusta cada bloque instanciado por separado para satisfacer las necesidades de cada capa. Actualmente, esta característica requiere la reconfiguración completa del FPGA cuando se consideran nuevas imágenes de entrada, existiendo la posibilidad de utilizar múltiples FPGA para cada nueva instanciación.

En [112] se emplea una estructura *pipeline*, así, diferentes capas puede trabajar simultáneamente. Se utilizan *buffers* internos para almacenar los resultados intermedios entre las capas, mientras que los pesos son almacenados en memoria externa. Adicionalmente, se aplica el método de procesamiento basado en lotes en la etapa de clasificación, de esta manera se puede procesar en paralelo múltiples imágenes utilizando los mismos

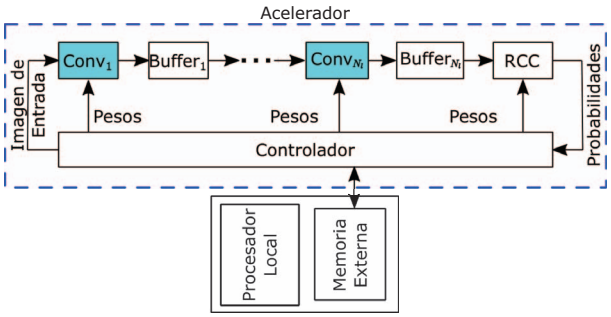


Fig. 4. Arquitectura típica para el procesamiento en paralelo de múltiples capas de convolución para múltiples imágenes.

parámetros de la red. En [103] se implementa una RNC de dos capas de convolución y se aplica en el reconocimiento de objetos utilizando la base de datos *CIFAR-10*. En este trabajo la etapa de clasificación fue remplazada por una red neuronal recurrente en lugar de la clásica red completamente conectada, aunque en este trabajo se implementan las diferentes capas de convolución, no se reporta a detalle los tipos de paralelismo explotados.

V. Tendencias en Técnicas de Diseño de Aceleradores Hardware

De acuerdo con las tendencias identificadas en el estado del arte relacionadas con los aceleradores de hardware, la Figura 5 muestra el diagrama de tiempo característico de una RNC de cinco capas de convolución empleando las arquitecturas AHCU y AHMC. En el primer caso, el enfoque de diseño se centra en permitir la flexibilidad en el convolucionador por medio de la actualización de rutas de datos a fin de procesar todas las capas de convolución con un solo convolucionador. Esta operación secuencial considerando una RNC de cinco capas convolucionales se muestra en la Figura 5a. En cada intervalo de tiempo Tn a $Tn+1$, se procesa una imagen I_h bajo la capa correspondiente, después de calcular la última capa convolucional, la imagen I_{h+1} se puede comenzar a procesar. Por otro lado, existen las AHMC cuyo diagrama de tiempo representativo se ilustra en la Figura 5b. Este tipo de arquitecturas se caracterizan por emplear un convolucionador por capa convolucional, con lo cual es posible procesar la imagen de entrada a través de todos los convolucionadores. Una vez que se calcula la primera capa de convolución, su primer convolucionador (Conv1) esta disponible para procesar una nueva imagen de entrada. Esta característica se cumple para los convolucionadores restantes, por ello, a partir de cierta latencia, el convolucionador de cada capa procesa una imagen diferente.

En ambos enfoques, el procesamiento de cada capa convolucional se realiza en cada intervalo de tiempo. Aunque este intervalo es visualmente igual para AHCU y AHMC en la figura, el tiempo de procesamiento solo podría ser el mismo si ambas arquitecturas de convolucionadores fueran equivalentes. La capacidad de aceleración de los convolucionadores depende de los tipos de paralelismo y patrones de computo seleccionados en función de los criterios y restricciones de la aplicación. $Tp_{conv}^{(l)}$, expresado en la Ecuación 1, representa el

tiempo de procesamiento en un convolucionador que calcula la capa l para una frecuencia de operación f. La Tabla I resume de manera general los beneficios e inconvenientes obtenidos al emplear estas estrategias de diseño identificadas, ademas se muestra el tiempo de procesamiento total por imagen Tp alcanzable por cada una de estas estrategias de diseño.

$$Tp_{conv}^{(l)} = \frac{1}{f} \times \left\lceil \frac{(w^{(l)})^2}{nm^{(l)}} \right\rceil \times \left\lceil \frac{CH^{(l)}}{nf_i^{(l)}} \right\rceil \times \left\lceil \frac{K^{(l)}}{nf_o^{(l)}} \right\rceil \times \left\lceil \frac{(N^{(l+1)})^2}{nv^{(l)}} \right\rceil \quad (1)$$

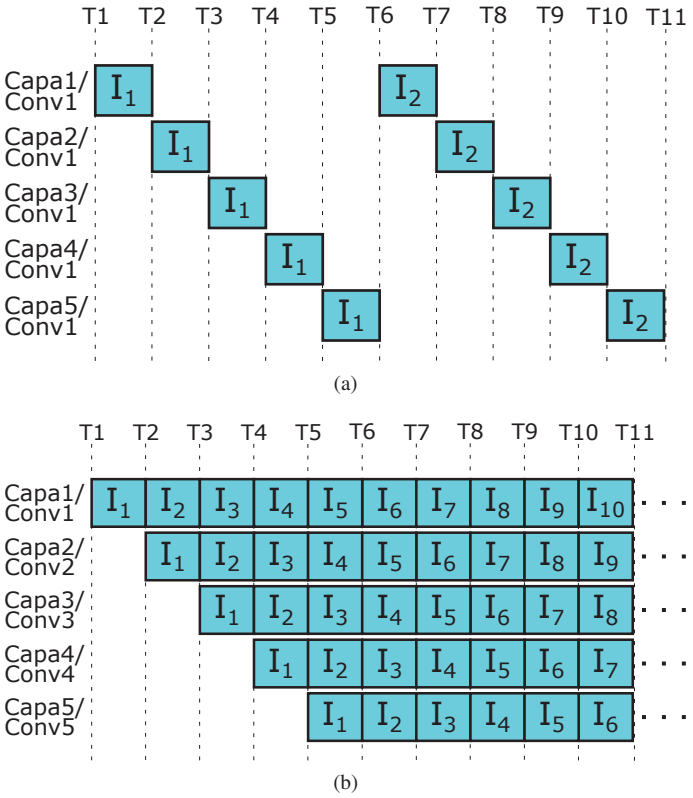


Fig. 5. Diagrama de tiempo característico de una RNC de cinco etapas de convolución con una arquitectura basada en a) AHCU y b) AHMC.

A. Técnicas Empleadas en Aceleradores HW

En la Tabla II se muestra un resumen de las estrategias reportadas para implementar aceleradores hardware en RNCs. Se presentan las técnicas para transferir los datos hacia el acelerador (que comúnmente provienen de memoria externa), el número de multiplicadores empleados en cada capa de convolución y su estrategia de operación. Para el paralelismo a nivel de lazo se incluye el número de filtros que se procesan en paralelo. Así mismo se presentan también las estrategias empleadas en las capas pertenecientes a la etapa de clasificación. Para este propósito, se respetan los términos manejados por cada uno de los trabajos reportados.

Se puede apreciar que para generar cada una de las máscaras de convolución es común emplear líneas de retardo, ya que a partir de una determinada latencia los valores a procesar estarán disponibles de manera paralela. El principal inconveniente de este esquema es que dificulta la flexibilidad, ya que la variedad de tamaños de filtros e imágenes implicaría inferir una gran cantidad de conexiones para poder adaptarse a cualquier configuración de convolución. Este esquema es más adecuado para aquellas topologías que emplean un tamaño fijo de filtro, como es el caso de *VGG* que se implementa en [63]. En aquellas topologías que emplean filtros de diferentes dimensiones, los investigadores han optado por hacer uso de *buffers* de entrada al convolucionador, donde se genera una subimagen de mayor dimensión al kernel de convolución (*Tile*). De esta manera el convolucionador se configura dependiendo de la dimensión del filtro y subimagen de entrada. Así, la convolución comúnmente se realiza mediante un árbol de multiplicadores de la misma dimensión que el kernel de convolución, algunos ejemplos de este modo de operación se reportan en [94], [106]. Cabe destacar que en todos los trabajos reportados este número de multiplicadores es fijo y depende específicamente de las características de la capa de convolución, así, cada técnica de optimización en los diferentes trabajos adopta un número diferente de multiplicadores para una misma capa, incluso para una misma topología de RNC. De igual manera la selección del número de filtros que se procesan de manera paralela depende de la estrategia de optimización empleada y usualmente el número de filtros a procesar depende de las características de la capa de convolución para una RNC en particular. La descripción del diseño de la etapa de clasificación es un aspecto que algunos trabajos no dan detalle en su implementación o simplemente no se contempla debido a que el peso computación de estas etapas no es significativo en comparación con las capas de convolución, aunque es común tratar a estas capas como una multiplicación de matrices y utilizando un árbol de multiplicadores o Multiplicadores Acumuladores (MACs) para realizar esta operación.

VI. Áreas de Investigación Potenciales

De acuerdo a la clasificación propuesta y al análisis realizado de los trabajos reportados recientemente en el área de aceleradores hardware para RNCs, se identifican dos principales áreas de oportunidad, las cuales se describen a continuación:

A. Arquitecturas con Múltiples Convolucionadores

Al utilizar una AHMC (donde $N_c = N_l$) se obtiene una mejora en rendimiento, ya que es posible procesar N_c imágenes simultáneamente. No obstante, este enfoque no es adecuado cuando la implementación se realiza en dispositivos hardware de gama baja, puesto que la cantidad de recursos requerida por la arquitectura en muchas ocasiones puede ser mayor a la disponible en estos dispositivos. Por lo tanto, es posible explorar arquitecturas con un número reducido de convolucionadores de tal forma que se obtenga un número de convolucionadores menor al número de capas de la RNC tal que $1 < N_c < N_l$. En este nuevo enfoque algunos convolucionadores procesan más de una capa de convolución mientras que otros procesan solo una capa, ésto dependiendo de la relación N_l/N_c, así, las N_l capas de una RNC se procesan con los N_c convolucionadores disponibles. En la Figura

TABLA I
COMPARACIÓN DE ESTRATEGIAS DE DISEÑO IDENTIFICADAS EN TÉRMINOS DE RENDIMIENTO Y FLEXIBILIDAD

Estrategia de diseño	Ventajas	Desventajas	Tiempo de procesamiento
AHCU	• Flexibilidad para operar sobre diferentes capas de una RNC • Mínima cantidad de memoria interna requerida	• Desempeño variable del acelerador (Debido a la flexibilidad) • Solo se procesa un capa convolucional a la vez	$Tp = \sum_{l=1}^{N_l} Tp_{conv}^{(l)}$
AHMC	• Alto rendimiento al procesar múltiples imágenes • Alta eficiencia en cada convolucionador • Mínima cantidad de accesos requeridos a memoria externa	• Nula flexibilidad para procesar diferentes tipos de RNC • Bajo rendimiento al procesar una imagen • Elevada utilización de recursos hardware	• Una imagen $Tp = \sum_{l=1}^{N_l} Tp_{conv}^{(l)}$ • Múltiples imágenes [2] $Tp = Tp_{conv}^{(j)}$

TABLA II
COMPARACIÓN DE LAS TÉCNICAS DE PARALELISMO EMPLEADAS EN LA IMPLEMENTACIÓN DE RNC EN TRABAJOS RECIENTES

Trabajo	Paralelismo de operación			Paralelismo de lazo	Paralelismo en capas FC		Re-uso
	Buffer de entrada	No. Mult	Estr. Mult.	No. filtros paralelos	Técnica mult	Estrategia	-
AHCU							
Aimar2018 [99]	DMA	128	MACs	1	-	-	IR
B. liu 2019 [24]	Tile maps	$Tm \times Tn$	Arbol Mult	$Tm \times Tn$	-	-	-
Ardakani 2017 [113]	Sistolico 1D	$p \times 3$	MACs	1	M. Clásico	Sistolico 1D	WR
Zhou 2018 [40]	-	48	Arbol Mult	-	M. Clásico	Arbol Mult	-
Du 2018[107]	Líneas de retardo	-	3×3 convolver array	1	-	-	IR
Qiu 2016 [63]	Líneas de retardo	9×16	Arbol Mult	64	DVS	Arbol mult	-
Tu 2017 [90]	Tile maps	256	Sistolico 2D	1	Matriz	Sistolico 2D	WR
Zhang2015 [94]	Tile maps	$T_m \times T_n$	Arbol Mult	48,20,96,95,32	-	-	NLR
Dundar 2017 [106]	Tile maps	576	3×3 convolver array	4	-	-	OR
Suda 2016[66]	-	-	Matriz mult.	1	Matriz	MACs	WR
Song 2016 [111]	-	$ceil(w/Stride)^2$	Arbol Mult	1	-	-	-
Zhang 2016 [67]	-	-	Arbol mult	-	Matriz mult.	Arbol Mult	-
Motamedi 2016[110]	-	T_k	Arbol mult	T_m	-	-	-
Lu2017 [59]	Líneas de retardo	3×3	Winograd	1	Winograd	Arbol mult	-
Nakahara 2015 [104]	Líneas de retardo	144	3×3 convolver array	1,4,16	-	-	WR
Chang 2017 [64]	Líneas de retardo	$\eta(w + w)$	ADVS	1	-	-	WR
AHMC							
Venieris 2018 [16]	Líneas de retardo	N_{fine}	Arbol Mult	N_{Coarse}	-	-	OR
N.Li 2016 [103]	Líneas de retardo	$w \times w$	Arbol Mult	3,15	RNN	MACs	WR
H.Li 2016 [112]	Sistolico 1D	$Size_{kernel}/Stride$	MACs	2,3,4,48	Matriz mult.	MACs	WR
Z. Liu 2017 [93]	Unrolling	$w \times w$	Arbol Mult	2,3,4	Matriz mult.	Arbol mult	-

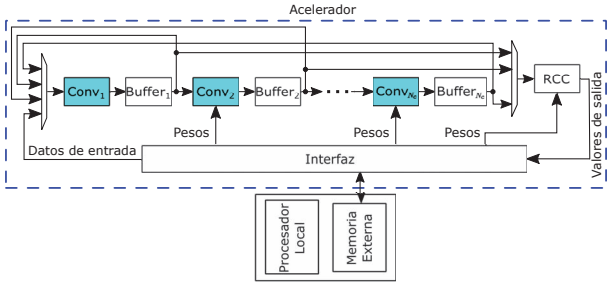


Fig. 6. Arquitectura hardware para una RNC con N_l capas de convolución y N_c convolucionadores, donde $1 < N_c < N_l$.

6 se ilustra el concepto descrito mediante una arquitectura hardware que utiliza N_c convolucionadores ($Conv_l$) donde $1 < N_c < N_l$. De esta manera, los resultados de salida del $l-$esimo convolucionador se almacenan en la memoria interna $Buffer_l$.

[2]La capa convolucional j es la de mayor complejidad computacional

El primer convolucionador $Conv_1$ recibe la primera imagen para procesar la primera capa de convolución. Después de ello las capas subsecuentes son procesadas por los siguientes N_c-1 convolucionadores. Una vez que la capa N_c sea procesada por el convolucionador N_c, los datos del $Buffer_{N_c}$ son dirigidos al primer convolucionador para que la capa N_c+1 sea procesada. De igual manera las siguientes capas son procesadas por los siguientes convolucionadores. El proceso finaliza hasta que el procesamiento de todas las capas de la RNC se realiza. De esta manera existe un reuso de los convolucionadores disponibles, el cual representa la flexibilidad buscada en las AHCU. Finalmente, los datos de salida de la última capa de convolución se transfieren a la etapa de clasificación. Con esta nueva técnica de diseño es posible obtener un compromiso entre el uso de recursos (Propiedad característica de AHCU) y rendimiento (Propiedad característica de AHMC), el cual está determinado directamente por el número de convolucionadores utilizados. Dada una aplicación en particular, los investigadores podrían enfocarse en encontrar el número de

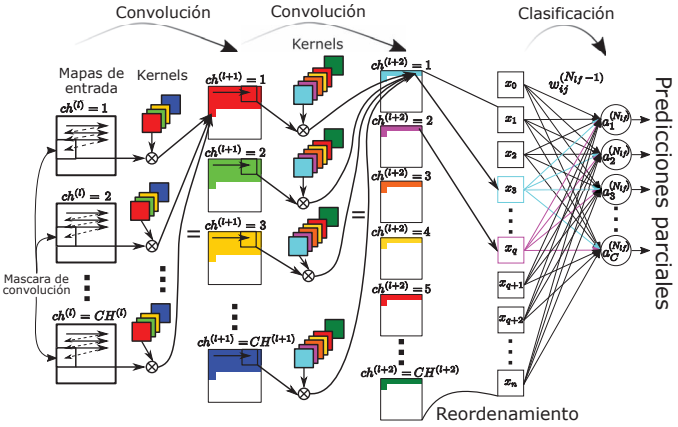


Fig. 7. Encadenamiento de operaciones a través de capas para una RNC de dos capas de convolución.

convolucionadores apropiado para la aplicación, considerando restricciones tal como recursos disponibles y subutilización.

B. Encadenamiento de Operaciones a Través de Capas

Otro enfoque aún poco explorado consiste en la experimentación con técnicas de diseño para implementar arquitecturas hardware basadas en el encadenamiento de operaciones a través de capas, principalmente de convolución. Esta arquitectura tiene un convolucionador destinado a cada capa de convolución, su funcionamiento permite obtener un procesamiento en paralelo de cada una de las capas considerando el análisis de una sola imagen. Esta técnica de encadenamiento de operaciones a través de capas se ilustra en la Figura 7 para el caso de una RNC de dos capas de convolución. Después de cierta latencia, los datos de salida resultantes de la primera capa de convolución pueden ser procesados por el primer componente de agrupamiento. Posteriormente los datos de salida son procesados por la segunda capa de convolución. Esta operación se repite hasta alcanzar la etapa de clasificación. En este punto todos los elementos de procesamiento están en operación sobre la misma imagen de entrada. Este enfoque es considerado como una opción para acelerar el tiempo de procesamiento al considerar solo una imagen.

VII. Conclusión

En este artículo se presenta una revisión exhaustiva de las estrategias de diseño actuales más importantes para solventar el alto peso computacional en RNCs, para ello se sigue una taxonomía propuesta de las mismas a fin de proveer una descripción organizada del estado del arte. Por un lado las soluciones a nivel de algoritmo se enfocan en reducir ya sea el número de multiplicaciones en capas de convolución o reducir la cantidad de memoria requerida para almacenar los coeficientes de la red por medio de técnicas de compresión de datos. Desde la perspectiva hardware, los diseñadores se enfrentan a dos principales retos. Uno de ellos consiste en reducir el número de accesos a memoria externa a través una adecuada reutilización de datos, lo cual se traduce en un menor consumo de la energía. Por otro lado, acelerar los cálculos haciendo un uso eficiente de los recursos hardware es otro de los mayores desafíos. Como consecuencia existen dos remarcadas tendencias en el diseño de aceleradores hardware. La primera de ellas consiste en utilizar un solo convolucionador flexible que pueda procesar secuencialmente cada capa de convolución, con ello se logra reducir la cantidad de recursos empleados y así poder ser aplicado en escenarios que utilicen dispositivos de gama baja. La segunda tendencia es diseñar arquitecturas basadas en múltiples convolucionadores buscando incrementar el rendimiento al procesar múltiples imágenes en paralelo. Este último enfoque no se concentra en la reducción de recursos, por lo cual puede ser aplicado en dispositivos de gama alta o bien, en servidores locales para procesamiento en la nube. A partir de la taxonomía propuesta, se identifican dos áreas aún poco exploradas orientadas a la aceleración de cálculos a través de hardware. La primera de ellas consiste en proponer una arquitectura basada en múltiples convolucioandores, donde el número de convolucionadores sea menor al número de capas de convolución, es decir $1 < N_c < N_l$. Así, cada convolucionador debe ser reutilizable y se lograría procesar una mayor cantidad de imágenes en comparación a utilizar un solo convolucionador. Otra posibilidad es emplear múltiples aceleradores con un encadenamiento de operaciones a través de capas en la RNC para el procesamiento de una imagen. Lo anterior podría permitir procesar una sola imagen en un menor tiempo que las estrategias de diseño identificadas en el estado del arte. Finalmente, se espera que la revisión presentada en este artículo ayude al lector a tener un panorama general y actualizado del tema y con ello motivar a investigadores a generar más ramas dentro de la taxonomía de aceleradores hardware para RNC ya existentes.

Referencias

[1] Y. Lecun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.
[2] L. Zhang, S. Wang, and B. Liu, "Deep learning for sentiment analysis: A survey," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 8, no. 4, p. e1253, 2018.
[3] A. Shawahna, S. M. Sait, and A. El-Maleh, "Fpga-based accelerators of deep learning networks for learning and classification: A review," *IEEE Access*, vol. 7, pp. 7823–7859, 2018.
[4] M. Anthimopoulos, B. Gatos, and I. Pratikakis, "A two-stage scheme for text detection in video images," *Image and Vision Computing*, vol. 28, no. 9, pp. 1413–1426, 2010.
[5] R. Lienhart and J. Maydt, "An extended set of haar-like features for rapid object detection," in *Proceedings. international conference on image processing*, vol. 1. IEEE, 2002, pp. I–I.
[6] M. Khammari, Y. Tlili, and C. Bencheriet, "Robust face detection using haar features with weber local descriptor and local binary pattern," *International Journal of Knowledge-based and Intelligent Engineering Systems*, vol. 19, no. 2, pp. 117–123, 2015.
[7] R. Khokher, R. C. Singh, and R. Kumar, "Footprint recognition with principal component analysis and independent component analysis," in *Macromolecular Symposia*, vol. 347. Wiley Online Library, 2015, pp. 16–26.
[8] D. Žunić and J. Žunić, "Shape ellipticity from hu moment invariants," *Applied mathematics and computation*, vol. 226, pp. 406–414, 2014.
[9] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," 2005.
[10] R. Hu and J. Collomosse, "A performance evaluation of gradient field hog descriptor for sketch based image retrieval," *Computer Vision and Image Understanding*, vol. 117, no. 7, pp. 790–806, 2013.
[11] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[12] E. Fidalgo, E. Alegre, V. González-Castro, and L. Fernández-Robles, "Compass radius estimation for improved image classification using edge-sift," *Neurocomputing*, vol. 197, pp. 119–135, 2016.

[13] Z.-Q. Zhao, P. Zheng, S.-t. Xu, and X. Wu, "Object detection with deep learning: A review," *IEEE transactions on neural networks and learning systems*, 2019.

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[15] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, Dec 2015. [Online]. Available: https://doi.org/10.1007/s11263-015-0816-y

[16] S. I. Venieris and C.-S. Bouganis, "fpgaconvnet: Mapping regular and irregular convolutional neural networks on fpgas," *IEEE transactions on neural networks and learning systems*, no. 99, pp. 1–17, 2018.

[17] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*. Springer, 2014, pp. 818–833.

[18] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[19] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[20] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7132–7141.

[21] V. Sze, Y.-H. Chen, T.-J. Yang, and J. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *arXiv preprint arXiv:1703.09039*, 2017.

[22] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[23] A. Jafari, A. Page, C. Sagedy, E. Smith, and T. Mohsenin, "A low power seizure detection processor based on direct use of compressively-sensed data and employing a deterministic random matrix," in *Biomedical Circuits and Systems Conference (BioCAS), 2015 IEEE*. IEEE, 2015, pp. 1–4.

[24] B. Liu, D. Zou, L. Feng, S. Feng, P. Fu, and J. Li, "An fpga-based cnn accelerator integrating depthwise separable convolution," *Electronics*, vol. 8, no. 3, p. 281, 2019.

[25] M. T. Hailesellasie and S. R. Hasan, "Mulnet: A flexible cnn processor with higher resource utilization efficiency for constrained devices," *IEEE Access*, vol. 7, pp. 47 509–47 524, 2019.

[26] J. Cong, Z. Fang, M. Huang, P. Wei, D. Wu, and C. H. Yu, "Customizable computing—from single chip to datacenters," *Proceedings of the IEEE*, vol. 107, no. 1, pp. 185–203, 2018.

[27] J. Cong and B. Xiao, "Minimizing computation in convolutional neural networks," in *Artificial Neural Networks and Machine Learning – ICANN 2014*, S. Wermter, C. Weber, W. Duch, T. Honkela, P. Koprinkova-Hristova, S. Magg, G. Palm, and A. E. P. Villa, Eds. Cham: Springer International Publishing, 2014, pp. 281–290.

[28] Y. Ma, Y. Cao, S. Vrudhula, and J.-s. Seo, "Optimizing the convolution operation to accelerate deep neural networks on fpga," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 7, pp. 1354–1367, 2018.

[29] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2017.

[30] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: efficient inference engine on compressed deep neural network," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2016, pp. 243–254.

[31] D. Zang, Z. Chai, J. Zhang, D. Zhang, and J. Cheng, "Vehicle license plate recognition using visual attention model and deep learning," *Journal of Electronic Imaging*, vol. 24, no. 3, pp. 033 001–033 001, 2015.

[32] X. Wang, H. Liu, J. Wang, and H. Wang, "Vehicle-logo recognition algorithm based on convolutional neural network," *JOURNAL OF INFORMATION &COMPUTATIONAL SCIENCE*, vol. 12, no. 18, pp. 6945–6952, 2015.

[33] D. Marmanis, M. Datcu, T. Esch, and U. Stilla, "Deep learning earth observation classification using imagenet pretrained networks," *IEEE Geoscience and Remote Sensing Letters*, vol. 13, no. 1, pp. 105–109, 2016.

[34] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "Overfeat: Integrated recognition, localization and detection using convolutional networks," *arXiv preprint arXiv:1312.6229*, 2013.

[35] X. Qi, C.-G. Li, G. Zhao, X. Hong, and M. Pietikäinen, "Dynamic texture and scene classification by transferring deep image features," *Neurocomputing*, vol. 171, pp. 1230–1241, 2016.

[36] L. Xie, T. Ahmad, L. Jin, Y. Liu, and S. Zhang, "A new cnn-based method for multi-directional car license plate detection," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 2, pp. 507–517, 2018.

[37] E. Shelhamer, J. Long, and T. Darrell, "Fully convolutional networks for semantic segmentation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 4, pp. 640–651, 2017.

[38] M. Oquab, L. Bottou, I. Laptev, and J. Sivic, "Learning and transferring mid-level image representations using convolutional neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 1717–1724.

[39] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "Deepdriving: Learning affordance for direct perception in autonomous driving," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 2722–2730.

[40] Y. Zhou, Y. Lyu, and X. Huang, "Roadnet: An 80-mw hardware accelerator for road detection," *IEEE Embedded Systems Letters*, vol. 11, no. 1, pp. 21–24, 2018.

[41] S. Zheng, X. Zeng, G. Lin, C. Zhao, Y. Feng, J. Tao, D. Zhu, and L. Xiong, "Sunspot drawings handwritten character recognition method based on deep learning," *New Astronomy*, vol. 45, pp. 54–59, 2016.

[42] B. Shi, X. Bai, and C. Yao, "Script identification in the wild via discriminative convolutional neural network," *Pattern Recognition*, vol. 52, pp. 448–458, 2016.

[43] Y. Kim and T. Moon, "Human detection and activity classification based on micro-doppler signatures using deep convolutional neural networks," *IEEE Geoscience and Remote Sensing Letters*, vol. 13, no. 1, pp. 8–12, 2016.

[44] H. A. Perlin and H. S. Lopes, "Extracting human attributes using a convolutional neural network approach," *Pattern Recognition Letters*, vol. 68, pp. 250–259, 2015.

[45] J. Tompson, M. Stein, Y. Lecun, and K. Perlin, "Real-time continuous pose recovery of human hands using convolutional networks," *ACM Transactions on Graphics (ToG)*, vol. 33, no. 5, p. 169, 2014.

[46] O. M. Parkhi, A. Vedaldi, A. Zisserman *et al.*, "Deep face recognition." in *bmvc*, vol. 1, no. 3, 2015, p. 6.

[47] E. Le, Y. Wang, Y. Huang, S. Hickman, and F. Gilbert, "Artificial intelligence in breast imaging," *Clinical radiology*, 2019.

[48] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[49] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[50] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 37, no. 9, pp. 1904–1916, 2015.

[51] B. J. Erickson, P. Korfiatis, Z. Akkus, T. Kline, and K. Philbrick, "Toolkits and libraries for deep learning," *Journal of digital imaging*, vol. 30, no. 4, pp. 400–405, 2017.

[52] M. Mathieu, M. Henaff, and Y. LeCun, "Fast training of convolutional networks through ffts," *arXiv preprint arXiv:1312.5851*, 2013.

[53] C. Dubout and F. Fleuret, "Exact acceleration of linear object detectors," in *European Conference on Computer Vision*. Springer, 2012, pp. 301–311.

[54] C. Zhang and V. K. Prasanna, "Frequency domain acceleration of convolutional neural networks on cpu-fpga shared memory system." in *FPGA*, 2017, pp. 35–44.

[55] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cudnn: Efficient primitives for deep learning," *arXiv preprint arXiv:1410.0759*, 2014.

[56] N. Vasilache, J. Johnson, M. Mathieu, S. Chintala, S. Piantino, and Y. LeCun, "Fast convolutional nets with fbfft: A GPU performance evaluation," *CoRR*, vol. abs/1412.7580, 2014. [Online]. Available: http://arxiv.org/abs/1412.7580

[57] A. Lavin and S. Gray, "Fast algorithms for convolutional neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 4013–4021.

[58] S. Winograd, *Arithmetic complexity of computations*. Siam, 1980, vol. 33.

[59] L. Lu, Y. Liang, Q. Xiao, and S. Yan, "Evaluating fast algorithms for convolutional neural networks on fpgas," in *Field-Programmable Custom Computing Machines (FCCM), 2017 IEEE 25th Annual International Symposium on*. IEEE, 2017, pp. 101–108.

[60] U. Aydonat, S. O'Connell, D. Capalija, A. C. Ling, and G. R. Chiu, "An opencl™deep learning accelerator on arria 10," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '17. New York, NY, USA: ACM, 2017, pp. 55–64. [Online]. Available: http://doi.acm.org/10.1145/3020078.3021738

[61] Q. Lan, Z. Wang, M. Wen, C. Zhang, and Y. Wang, "High performance implementation of 3d convolutional neural networks on a gpu," *Computational intelligence and neuroscience*, vol. 2017, 2017.

[62] L. Meng and J. Brothers, "Efficient winograd convolution via integer arithmetic," *arXiv preprint arXiv:1901.01965*, 2019.

[63] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song *et al.*, "Going deeper with embedded fpga platform for convolutional neural network," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2016, pp. 26–35.

[64] J. Chang and J. Sha, "An efficient implementation of 2d convolution in cnn," *IEICE Electronics Express*, vol. 14, no. 1, pp. 20 161 134–20 161 134, 2017.

[65] X. Zhang, J. Zou, X. Ming, K. He, and J. Sun, "Efficient and accurate approximations of nonlinear convolutional networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1984–1992.

[66] N. Suda, V. Chandra, G. Dasika, A. Mohanty, Y. Ma, S. Vrudhula, J.-s. Seo, and Y. Cao, "Throughput-optimized opencl-based fpga accelerator for large-scale convolutional neural networks," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2016, pp. 16–25.

[67] C. Zhang, Z. Fang, P. Zhou, P. Pan, and J. Cong, "Caffeine: Towards uniformed representation and acceleration for deep convolutional neural networks," in *Computer-Aided Design (ICCAD), 2016 IEEE/ACM International Conference on*. IEEE, 2016, pp. 1–8.

[68] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[69] M. Kim and P. Smaragdis, "Bitwise neural networks," *arXiv preprint arXiv:1601.06071*, 2016.

[70] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European Conference on Computer Vision*. Springer, 2016, pp. 525–542.

[71] H. Lee, Y.-H. Wu, Y.-S. Lin, and S.-Y. Chien, "Convolutional neural network accelerator with vector quantization," in *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2019, pp. 1–5.

[72] T.-q. Peng and F. Li, "Image retrieval based on deep convolutional neural networks and binary hashing learning," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2017, pp. 1742–1746.

[73] G. J. Sullivan, "Efficient scalar quantization of exponential and laplacian random variables," *IEEE Transactions on information theory*, vol. 42, no. 5, pp. 1365–1374, 1996.

[74] M. Denil, B. Shakibi, L. Dinh, N. de Freitas *et al.*, "Predicting parameters in deep learning," in *Advances in Neural Information Processing Systems*, 2013, pp. 2148–2156.

[75] Y. Gong, L. Liu, M. Yang, and L. Bourdev, "Compressing deep convolutional networks using vector quantization," *arXiv preprint arXiv:1412.6115*, 2014.

[76] R. Balasubramanian, C. A. Bouman, and J. P. Allebach, "Sequential scalar quantization of color images," *Journal of Electronic Imaging*, vol. 3, no. 1, pp. 45–59, 1994.

[77] Y. Chen, T. Guan, and C. Wang, "Approximate nearest neighbor search by residual vector quantization," *Sensors*, vol. 10, no. 12, pp. 11 259–11 273, 2010.

[78] W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen, "Compressing neural networks with the hashing trick," in *International Conference on Machine Learning*, 2015, pp. 2285–2294.

[79] H. Bagherinezhad, M. Rastegari, and A. Farhadi, "Lcnn: Lookup-based convolutional neural network," *arXiv preprint arXiv:1611.06473*, 2016.

[80] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," *arXiv preprint arXiv:1611.06440*, 2016.

[81] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," *arXiv preprint arXiv:1608.08710*, 2016.

[82] S. J. Hanson and L. Y. Pratt, "Comparing biases for minimal network construction with back-propagation," in *Advances in neural information processing systems*, 1989, pp. 177–185.

[83] Y. L. Cun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Advances in Neural Information Processing Systems 2*, D. S. Touretzky, Ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1990, pp. 598–605.

[84] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in Neural Information Processing Systems*, 2015, pp. 1135–1143.

[85] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient dnns," in *Advances In Neural Information Processing Systems*, 2016, pp. 1379–1387.

[86] H. Hu, R. Peng, Y.-W. Tai, and C.-K. Tang, "Network trimming: A data-driven neuron pruning approach towards efficient deep architectures," *arXiv preprint arXiv:1607.03250*, 2016.

[87] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Pensky, "Sparse convolutional neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 806–814.

[88] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Advances in Neural Information Processing Systems*, 2016, pp. 2074–2082.

[89] M. Horowitz, "1.1 computing's energy problem (and what we can do about it)," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International*. IEEE, 2014, pp. 10–14.

[90] F. Tu, S. Yin, P. Ouyang, S. Tang, L. Liu, and S. Wei, "Deep convolutional neural network architecture with reconfigurable computation patterns," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2017.

[91] S. Chakradhar, M. Sankaradas, V. Jakkula, and S. Cadambi, "A dynamically configurable coprocessor for convolutional neural networks," in *ACM SIGARCH Computer Architecture News*, vol. 38. ACM, 2010, pp. 247–257.

[92] G. Lacey, G. W. Taylor, and S. Areibi, "Deep learning on fpgas: Past, present, and future," *CoRR*, vol. abs/1602.04283, 2016.

[93] Z. Liu, Y. Dou, J. Jiang, J. Xu, S. Li, Y. Zhou, and Y. Xu, "Throughput-optimized fpga accelerator for deep convolutional neural networks," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 10, no. 3, p. 17, 2017.

[94] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing fpga-based accelerator design for deep convolutional neural networks," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2015, pp. 161–170.

[95] K. Guo, L. Sui, J. Qiu, J. Yu, J. Wang, S. Yao, S. Han, Y. Wang, and H. Yang, "Angel-eye: A complete design flow for mapping cnn onto embedded fpga," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 1, pp. 35–47, 2017.

[96] K. Abdelouahab, M. Pelcat, J. Sérot, C. Bourrasset, and F. Berry, "Tactics to directly map cnn graphs on embedded fpgas," *IEEE Embedded Systems Letters*, 2017.

[97] M. Peemen, A. A. Setio, B. Mesman, and H. Corporaal, "Memory-centric accelerator design for convolutional neural networks," in *Computer Design (ICCD), 2013 IEEE 31st International Conference on*. IEEE, 2013, pp. 13–19.

[98] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," in *ACM Sigplan Notices*, vol. 49. ACM, 2014, pp. 269–284.

[99] A. Aimar, H. Mostafa, E. Calabrese, A. Rios-Navarro, R. Tapiador-Morales, I.-A. Lungu, M. B. Milde, F. Corradi, A. Linares-Barranco, S.-C. Liu *et al.*, "Nullhop: A flexible convolutional neural network accelerator based on sparse representations of feature maps," *IEEE transactions on neural networks and learning systems*, vol. 30, no. 3, pp. 644–656, 2018.

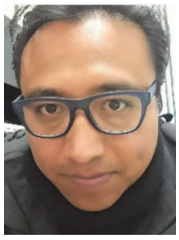
[100] H.-T. Kung, "Why systolic architectures?" *IEEE computer*, vol. 15, no. 1, pp. 37–46, 1982.

[101] C. Farabet, C. Poulet, J. Y. Han, and Y. LeCun, "Cnp: An fpga-based processor for convolutional networks," in *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*. IEEE, 2009, pp. 32–37.

[102] M. Sankaradas, V. Jakkula, S. Cadambi, S. Chakradhar, I. Durdanovic, E. Cosatto, and H. P. Graf, "A massively parallel coprocessor for convolutional neural networks," in *Application-specific Systems, Architectures and Processors, 2009. ASAP 2009. 20th IEEE International Conference on*. IEEE, 2009, pp. 53–60.
[103] N. Li, S. Takaki, Y. Tomiokay, and H. Kitazawa, "A multistage dataflow implementation of a deep convolutional neural network based on fpga for high-speed object recognition," in *Image Analysis and Interpretation (SSIAI), 2016 IEEE Southwest Symposium on*. IEEE, 2016, pp. 165–168.
[104] H. Nakahara and T. Sasao, "A deep convolutional neural network based on nested residue number system," in *Field Programmable Logic and Applications (FPL), 2015 25th International Conference on*. IEEE, 2015, pp. 1–6.
[105] R. G. Shoup, "Parameterized convolution filtering in a field programmable gate array," in *Selected papers from the Oxford 1993 international workshop on field programmable logic and applications on More FPGAs. Oxford, United Kingdom: Abingdon EE&CS Books*, 1994, pp. 274–280.
[106] A. Dundar, J. Jin, B. Martini, and E. Culurciello, "Embedded streaming deep neural networks accelerator with applications," *IEEE transactions on neural networks and learning systems*, 2017.
[107] L. Du, Y. Du, Y. Li, J. Su, Y.-C. Kuan, C.-C. Liu, and M.-C. F. Chang, "A reconfigurable streaming deep convolutional neural network accelerator for internet of things," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 1, pp. 198–208, 2018.
[108] R. Nane, V.-M. Sima, C. Pilato, J. Choi, B. Fort, A. Canis, Y. T. Chen, H. Hsiao, S. Brown, F. Ferrandi *et al.*, "A survey and evaluation of fpga high-level synthesis tools," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 10, pp. 1591–1604, 2015.
[109] S. Williams, A. Waterman, and D. Patterson, "Roofline: An insightful visual performance model for floating-point programs and multicore architectures," Lawrence Berkeley National Lab.(LBNL), Berkeley, CA (United States), Tech. Rep., 2009.
[110] M. Motamedi, P. Gysel, V. Akella, and S. Ghiasi, "Design space exploration of fpga-based deep convolutional neural networks," in *Design Automation Conference (ASP-DAC), 2016 21st Asia and South Pacific*. IEEE, 2016, pp. 575–580.
[111] L. Song, Y. Wang, Y. Han, X. Zhao, B. Liu, and X. Li, "C-brain: A deep learning accelerator that tames the diversity of cnns through adaptive data-level parallelization," in *Design Automation Conference (DAC), 2016 53nd ACM/EDAC/IEEE*. IEEE, 2016, pp. 1–6.
[112] H. Li, X. Fan, L. Jiao, W. Cao, X. Zhou, and L. Wang, "A high performance fpga-based accelerator for large-scale convolutional neural networks," in *Field Programmable Logic and Applications (FPL), 2016 26th International Conference on*. IEEE, 2016, pp. 1–9.
[113] A. Ardakani, C. Condo, M. Ahmadi, and W. J. Gross, "An architecture to accelerate convolution in deep neural networks," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 4, pp. 1349–1362, 2017.

Moisés Arredondo-Velázquez recibió el grado de Ingeniero en Mecatrónica por parte de la Universidad Tecnológica del Norte de Guanajuato, Guanajuato, México, en 2013, y su grado de Maestro en Ciencias en Ingeniería Electrónica en el Instituto Tecnológico de Celaya, Guanajuato, México, en 2015. Actualmente está cursando el Programa de Doctorado en Ciencias de la Ingeniería en el Instituto Tecnológico de Celaya. Sus intereses de investigación actuales incluyen: aprendizaje profundo, aceleradores hardware, computación reconfigurable y procesamiento de imágenes.

Javier Díaz-Carmona recibió el grado de Ingeniero en Electrónica en 1992 por parte del Instituto Tecnológico de Celaya (ITC), Celaya, Gto, México, y realizó una Maestría en 1997 y un Doctorado en Ciencias en Electrónica en 2003 en el Instituto Nacional de Astrofísica Óptica y Electrónica (INAOE), Puebla, México. Desde 1991 ha sido miembro del Departamento de Ingeniería Electrónica en el ITC, actualmente como profesor e investigador a tiempo completo. Sus principales intereses de investigación incluyen procesamiento digital de señales, técnicas de diseño de filtros digitales, técnicas de diseño de hardware reconfigurable y microcontroladores, aplicaciones de sistemas embebidos DSP / FPGA.



Alejandro-Israel Barranco-Gutiérrez es Ingeniero en Telemática. Recibió los títulos de Maestría y Doctorado en Tecnología Avanzada en CICATA IPN y la Universidad de Meijo, respectivamente. Actualmente, el Prof. Barranco Gutiérrez está trabajando para Catedras-CONACyT en el Instituto Tecnológico de Celaya dentro del Departamento de Ingeniería Electrónica, Celaya, México. Sus intereses de investigación incluyen metrología visual, inteligencia artificial, circuitos electrónicos y telemática.

Cesar Torres-Huitzil Obtuvo sus grados de Maestría en Electrónica y Doctorado en Ciencias Computacionales por parte del Instituto Nacional de Astrofísica Óptica y Electrónica en 1998 y 2003, respectivamente. Actualmente es investigador en la Escuela de Ingeniería y Ciencias del Instituto Tecnológico y de Educación Superior de Monterrey, dentro de la unidad de investigación ubicada en Puebla, México. Sus principales líneas de investigación se centran en el computo reconfigurable y las aplicaciones computacionales de dispositivos FPGA en diferentes dominios, como la visión por computadora, procesamiento digital señales y la computación neuronal.