elektron ámbar

Original Article

Revista elektron, Vol. 8, No. 2, pp. 61-70 (2024)
https://doi.org/10.37537/rev.elektron.8.2.200.2024

Comparación de Vitis-AI y FINN para implementar redes neuronales convolucionales en FPGA

Comparison of Vitis-AI and FINN for implementing convolutional neural networks on FPGA

Nicolás Urbano Pintos*†[1], Héctor A. Lacomi*‡[2] y Mario B. Lavorato†[3]

*División Radar Láser, CITEDEF
J. B. de la Salle 4397, Villa Martelli- Argentina
[1]nurbano@frh.utn.edu.ar
‡Grupo ASE, UTN FRH
París 532, Haedo- Argentina
[2]hlacomi@citedef.gob.ar
†Grupo TAMA, UTN FRH
París 532, Haedo- Argentina
[3]mlavorato@frh.utn.edu.ar

Recibido: 30/09/24; Aceptado: 11/11/24

***Resumen*—Las redes neuronales convolucionales (CNN) son esenciales para la clasificación y detección de imágenes, y su implementación en sistemas embebidos resulta cada vez más atractiva debido a su tamaño compacto y bajo consumo energético. Los FPGA (Field-Programmable Gate Arrays) han surgido como una opción prometedora, gracias a su baja latencia y alta eficiencia energética.**

Vitis AI y FINN son dos entornos de desarrollo que automatizan la implementación de CNN en FPGA. Vitis AI utiliza una unidad de procesamiento de aprendizaje profundo (DPU) y aceleradores de memoria, mientras que FINN se basa en una arquitectura de transmisión de datos (streaming) y ajusta la paralelización. Ambos entornos implementan técnicas de cuantización de parámetros para reducir el uso de memoria.

Este trabajo extiende comparaciones previas al evaluar ambos entornos mediante la implementación de cuatro modelos con diferentes cantidades de capas en la plataforma FPGA Kria KV260 de Xilinx. Se describe en detalle el proceso completo, desde el entrenamiento hasta la evaluación en FPGA, incluyendo la cuantización y la implementación en hardware.

Los resultados muestran que FINN proporciona menor latencia, mayor rendimiento y mejor eficiencia energética que Vitis AI. No obstante, Vitis AI destaca por su simplicidad en el entrenamiento de modelos y facilidad de implementación en FPGA. El hallazgo principal del estudio es que, al aumentar la complejidad de los modelos con más capas, las diferencias de rendimiento y eficiencia energética entre FINN y Vitis AI se reducen notablemente.

Palabras clave: FPGA; CNN; FINN; Vitis-AI; Cuantización.

***Abstract*— Convolutional neural networks (CNNs) are essential for image classification and detection, and their implementation in embedded systems is becoming increasingly attractive due to their compact size and low power consumption. Field-Programmable Gate Arrays (FPGAs) have emerged as a promising option, thanks to their low latency and high energy efficiency.**

Vitis AI and FINN are two development environments that automate the implementation of CNNs on FPGAs. Vitis AI uses a deep learning processing unit (DPU) and memory accelerators, while FINN is based on a streaming architecture and fine-tunes parallelization. Both environments implement parameter quantization techniques to reduce memory usage.

This work extends previous comparisons by evaluating both environments by implementing four models with different numbers of layers on the Xilinx Kria KV260 FPGA platform. The complete process from training to evaluation on FPGA, including quantization and hardware implementation, is described in detail.

The results show that FINN provides lower latency, higher throughput, and better energy efficiency than Vitis AI. However, Vitis AI stands out for its simplicity in model training and ease of implementation on FPGA. The main finding of this study is that as the complexity of the models increases (with more layers in the neural networks), the differences in terms of performance and energy efficiency between FINN and Vitis AI are significantly reduced.

Keywords: FPGA; CNN; FINN; Vitis-AI; Quantization.

I. Introducción

Las redes neuronales convolucionales (CNN) son fundamentales en aplicaciones como la clasificación y detección de imágenes [1], debido a su capacidad para alcanzar precisiones de clasificación altas mediante el uso de múltiples capas y filtros complejos. Sin embargo, esta complejidad conlleva un alto consumo de memoria y un gran número de operaciones en punto flotante, lo que supone un desafío en escenarios donde la inferencia debe realizarse en tiempo real y en dispositivos compactos y portátiles [2].

Los sistemas embebidos han surgido como una opción viable para la inferencia de este tipo de modelos de aprendizaje profundo [3]. No obstante, debido los recursos de procesamiento y memoria limitados de estos dispositivos, la ejecución de estos modelos en tiempo real plantea una dificultad significativa. Entre las técnicas más utilizadas para abordar este desafío se encuentra la cuantización de los parámetros de los modelos, que reduce el tamaño de los pesos y activaciones, normalmente entrenados en punto flotante de 32 bits (FP32), a representaciones de menor precisión. Algunos enfoques destacados incluyen el uso de enteros de 8 bits [4], así como cuantizaciones ternarias y binarias [5].

A pesar de los avances en hardware embebido, como las GPU integradas en kits de desarrollo como es el caso de la placa Jetson Nano fabricada por Nvidia [6], que ofrecen aceleración en operaciones tensoriales, el consumo energético sigue siendo un desafío crítico. En este contexto, los dispositivos FPGA representan una alternativa destacada para la inferencia de modelos de aprendizaje profundo, ya que combinan una baja latencia con una alta eficiencia energética [7].

Entre los entornos de desarrollo automatizados para facilitar la implementación de CNN en FPGA, podemos destacar a: Vitis AI [8] y FINN [9], ambos desarrollados por Xilinx. Vitis AI utiliza un enfoque secuencial para resolver las operaciones de las CNN, en cambio, FINN utiliza se basa en un flujo de datos, permitiendo operaciones en paralelo.

Estudios recientes han comparado la performance de estos entornos [10] [11] utilizando parámetros como la latencia, el rendimiento y la eficiencia energética, concluyendo que FINN supera a Vitis AI, tanto en rendimiento como en eficiencia. Estas comparaciones se realizan en modelos específicos de CNN para tareas de clasificación y detección de objetos.

El presente trabajo tiene como objetivo ampliar esta comparación, aportando una contribución clave: el análisis de la performance de modelos de CNN con diferentes grados de complejidad (cantidad de capas). Para ello, se evaluaron tres modelos que combinan capas convolucionales y totalmente conectadas, junto con un modelo adicional compuesto únicamente de capas totalmente conectadas. Si bien en la comparativa se observa que performance de implementaciones en FINN son superiores a las del entorno Vitis AI, una observación fundamental de este estudio es que, a medida que aumenta la complejidad del modelo, la diferencia en rendimiento y eficiencia energética entre FINN y Vitis AI se reduce significativamente, aportando un nuevo entendimiento sobre el comportamiento de estos entornos a distintas escalas de complejidad.

Este artículo detalla los pasos necesarios para la implementación de cada modelo en ambos entornos, abarcando desde la arquitectura, el entrenamiento, la cuantización y la implementación en la placa de desarrollo FPGA Kria KV260 [12]. Además, se lleva a cabo un análisis exhaustivo de las métricas obtenidas, en función de la complejidad de los modelos, destacando las fortalezas y debilidades específicas de FINN y Vitis AI. Los autores han creado un repositorio público poder evaluar los modelos estudiados en FINN y VITIS AI en la placa de desarrollo Kria KV260. El mismo se accede desde: https://github.com/nurbano/finn_vs_vitisai_kv260

El resto del documento está organizado de la siguiente manera: en la sección II se detallan los antecedentes; en la sección III se describe la metodología utilizada; en la sección IV se presentan los resultados; en la sección V la discusión, en la sección VI se esbozan las conclusiones y por último en la sección VII se detallan las direcciones futuras del trabajo.

II. Antecedentes

II-A. Entornos de Desarrollo Automatizados para implementación de CNN en FPGA

Vitis AI [8] es un entorno de desarrollo que implementa un módulo optimizado para la inferencia de redes neuronales convolucionales, basado en DPU (Deep-Learning Processing Unit) [13] y aceleradores de memoria. La arquitectura del DPU se compone de un planificador de trabajo, un módulo matricial de cómputo híbrido y un módulo de memoria tipo pool. El DPU realiza un enfoque secuencial a partir de la ejecución de un microcódigo descripto en un archivo con extensión xmodel, el cual es generado por el compilador de Vitis AI. En la Fig. 1 se ilustra la arquitectura del DPU implementado por Vitis AI.

El DPU busca instrucciones en la memoria externa (SD), analiza y programa dichas instrucciones, y opera el motor de cálculo. Este motor de convolución se encarga de ejecutar las operaciones de convolución, así como también del agrupamiento y la normalización de lotes. Las operaciones fundamentales, como multiplicaciones, sumas y acumulaciones, están implementadas en elementos de procesamiento dedicados.

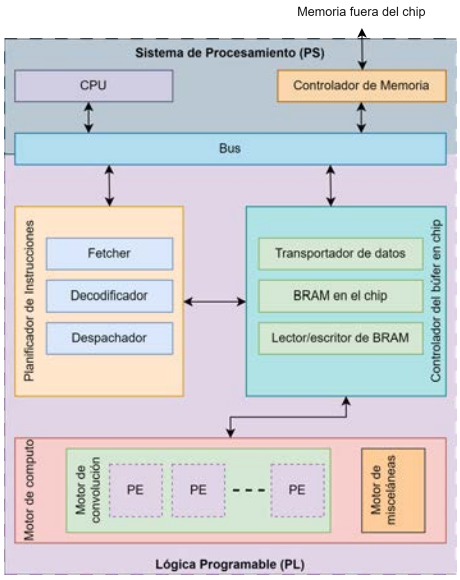


Figura 1: Arquitectura del DPU implementado en Vitis AI

El procedimiento de implementación en Vitis AI se realiza a través de un contenedor Docker que opera en un sistema Linux, preferentemente con distribución Ubuntu. El modelo puede ser entrenado en cualquier entorno de aprendizaje profundo, como PyTorch [14], TensorFlow [15] o Caffe [16]. Tras el entrenamiento, es necesario cuantizar el modelo de punto flotante a INT8, un proceso que se lleva a cabo en el entorno de PyTorch utilizando $torch\_quantization$. Posteriormente, se compila el modelo cuantizado utilizando la función vai_c_xir, definiendo el modelo DPU compatible con las diferentes familias de FPGA. Este proceso genera un archivo con extensión xmodel que se copia en la memoria SD de la placa de desarrollo de la FPGA. Para evaluar el modelo, se crea una aplicación, generalmente en Python, qué instancia el DPU y carga el modelo utilizando el conjunto de instrucciones del archivo xmodel.

Por otro lado, FINN utiliza una arquitectura de flujo de datos y permite la implementación de CNN mediante tres componentes principales: una unidad de umbral vectorial de matriz, una unidad de ventana deslizante y una unidad de agrupamiento [17]. La configuración de recursos en FINN no es fija, sino que se determina en función de requisitos específicos, como el rendimiento (cuadros por segundo) y la frecuencia de operación. La Fig. 2 presenta la arquitectura implementada por FINN. Los carriles SIMD (Single Instruction Multiple Data) se utilizan para realizar operaciones de multiplicación y suma en paralelo sobre múltiples datos de activación y peso. Esto permite que varias entradas sean procesadas simultáneamente. Por otra parte, los elementos de procesamiento (PE) son unidades de cómputo individuales dentro de la arquitectura de FINN que ejecutan operaciones básicas, como la multiplicación, suma y comparación de datos. Los PE reciben datos de los carriles SIMD, realizan las operaciones asignadas y luego envían los resultados a las etapas siguientes del flujo de datos. FINN permite ajustar la cantidad de PE disponibles en la FPGA de acuerdo con los requisitos de rendimiento, lo que se denomina "plegado" o "folding". Un mayor número de PE aumenta la capacidad de procesamiento paralelo, reduciendo la latencia y mejorando el rendimiento global, mientras que un número menor ahorra recursos de hardware.

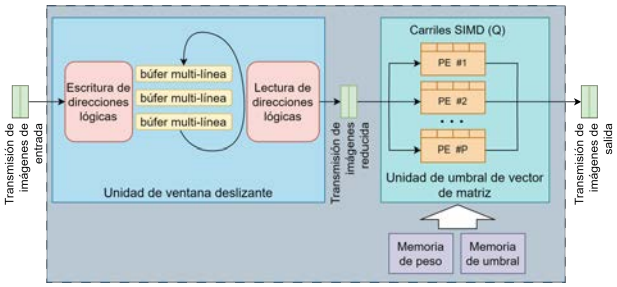


Figura 2: Arquitectura tipo flujo de FINN

Para implementar redes CNN en FINN, se debe construir el contenedor Docker FINN en un sistema operativo Linux, preferentemente en una distribución Ubuntu, y también es necesario instalar los software de Xilinx Vivado y Vitis HLS. Los modelos deben ser construidos y entrenados con capas cuantizadas utilizando Brevitas [18], siguiendo un entrenamiento de al menos 500 épocas. En esta etapa, se define el tipo de cuantización, ya sea binaria, ternaria, INT4 o INT8. El modelo entrenado se exporta al formato ONNX [19], y se evaluan los recursos utilizados de la FPGA, a partir del constructor $build\_dataflow$, donde se definen los parámetros de diseño, como el tiempo de reloj y el rendimiento en FPS deseado. Aunque el constructor ofrece un "auto-plegado" o "auto-folding" para automatizar el proceso, se pueden realizar ajustes manuales para optimizar el rendimiento. Finalmente, se genera el código HLS, el archivo con extensión bit y el controlador ejecutando los pasos completos del constructor $build_dataflow$. Estos archivos resultantes se copian a la memoria SD de la placa. Una vez en el sistema operativo Linux de la placa de desarrollo, se despliega el archivo .bit y se codifica una aplicación para verificar y evaluar el comportamiento.

II-B. Trabajos Relacionados

En la literatura actual, varios estudios han explorado la implementación de modelos de CNN en entornos como FINN y Vitis AI. Umuroglu et al. [9] implementaron tres modelos de clasificación basados en CNN utilizando FINN, analizando los recursos empleados, la latencia, la potencia, el rendimiento y la eficiencia energética. En una FPGA ZC706, FINN logra hasta 12.3 millones de clasificaciones por segundo y 95.8 % de precisión en el conjunto MNIST, y 21,906 clasificaciones por segundo en CIFAR-10 y SVHN, alcanzando precisiones de 80.1 % y 94.9 % respectivamente.

Por su parte, Ducasse et al. [20] experimentaron con un perceptrón multicapa (MLP- Multi Layer Perceptron) entrenado en los conjuntos de datos MNIST [21] y Fashion-MNIST [22], evaluando la implicancia de la cantidad de bits de cuantización en activaciones y pesos en una FPGA Zynq 7020. Sus resultados indican que las implementaciones con menor precisión pueden alcanzar una precisión comparable a las de mayor precisión, siempre que se realice un entrenamiento adecuado, manteniendo además una baja utilización de hardware y alta eficiencia.

Utilizando tanto FINN, Vitis AI y un acelerador ad hoc, Machura et al. [10] implementaron dos modelos de detección de objetos basados en CNN, "YoloFINN" y "LittleNet", evaluando el uso de energía, el rendimiento y la precisión en la detección de objetos en una placa de desarrollo Avnet Ultra96-V2 en el conjunto de datos VOT [23]. Sus hallazgos sugieren que aunque el modelo personalizado proporciona un mejor rendimiento, su desarrollo es considerablemente más complejo.

Hamanaka et al. [11] analizaron el modelo de clasificación ResNet-8, una versión reducida de ResNet [24] implementado en una placa creada ad hoc que tiene montado una FPGA SOM K26 de Xilinx, encontrando que FINN supera al acelerador basado en Vitis AI en términos de latencia, eficiencia energética y rendimiento.

Finalmente, Urbano Pintos et al. [25] implementaron una red VGG16 [26] en Vitis AI, con el conjunto de datos CIFAR10 [27]. Evaluando la precisión y el rendimiento. A su vez, los mismos autores implementaron con Brevitas [18] la red B-VGG16 [28], la misma no pudo ser compilada en FINN para en Kria KV260, debido a que no alcanzan los LUTs y/o BRAM disponibles.

III. Metodología

La metodología descripta en este estudio se centra en la implementación y evaluación de diferentes arquitecturas de clasificación de imágenes basadas redes neuronales convolucionales y en redes de capas totalmente conectadas en la FPGA Kria KV260. Cada modelo fue implementado a partir de los entornos de desarrollo automatizados Vitis AI y FINN, de principio a fin, esto se refiere al entrenamiento del modelo, la cuantización, la compilación y la evaluación de performance en la FPGA.

En este contexto, se seleccionaron tres modelos convolucionales: CNV [9], VGG11 (versión reducida de VGG16 [29]) y mini CNN (mini CNN) junto con un modelo de capas totalmente conectadas (LFC) [9]. En la tabla I se detalla la cantidad de capas totalmente conectadas, la cantidad de capas de convolución y la cantidad capas totales de cada

modelo. Este enfoque permite analizar cómo la variación en la profundidad de la red y la estructura arquitectónica impactan en el uso de recursos, el rendimiento, la eficiencia energética y la latencia.

Modelo	LFC	mCNN	CNV	VGG11
Capas totalmente conectadas	4	2	3	4
Capas de convolución	0	6	6	8
Capas Totales	**4**	**8**	**9**	**11**

Tabla I: Cantidad de capas de cada modelo

El modelo mCNN, es un modelo sencillo de menor cantidad de capas que fue desarrollado por los autores de este trabajo con el fin de aumentar los modelos a comparar, con los ya establecidos CNV y VGG11.

Aunque las redes totalmente conectadas no utilizan filtros de convolución, las capas totalmente conectadas son fundamentales en la etapa de clasificación de imágenes en las CNN. Por ello, se incluyó el modelo LFC en nuestra comparación para ofrecer una perspectiva completa sobre las capacidades de clasificación de las distintas arquitecturas.

A continuación, se detallan los conjuntos de datos utilizados, la arquitectura de cada modelo, los hiperparámetros de entrenamiento, los procedimientos en FINN y VITIS-AI, así como los detalles para la preparación de la placa Kria KV-260 y las métricas empleadas en la comparación.

III-A. Conjuntos de datos

Se utilizaron los conjuntos de datos CIFAR10 [27], MNIST [21] y SVHN [30]. CIFAR10 contiene 60 mil imágenes color de 32x32 píxeles de objetos y animales en 10 clases diferentes. MNIST contiene 60 mil números manuscritos del cero al nueve en escala de grises con un tamaño de 28x28 píxeles. Y SVHN, contiene más de 120 mil imágenes color de números de direcciones de casas recortadas, del cero al nueve con un tamaño de 32x32 píxeles.

III-B. Arquitecturas

En las Fig. 3a, 3b, 3c y 3d se presentan las arquitecturas de los modelos LFC, mCNN, CNV y VGG11 respectivamente. Estas figuras destacan el número y tipo de capas, y el tamaño de los filtros utilizados en cada arquitectura. Las capas de convolución y activación se encuentran representadas en color azul, las capas de agrupamiento en rojo y las capas totalmente conectadas en verde, por ejemplo en la Fig. 3c, se observa que la primera capa "Conv1-64" es una capa de convolución de 64 filtros. A su vez, se indica en cada arquitectura el conjunto de datos utilizado para el entrenamiento con una imágen de muestra.

El modelo **LFC** [9], representado en la figura 3a, tiene como entradas imágenes de 28x28 píxeles y 1 canal, correspondientes al conjunto de datos MNIST. A diferencia de los otros modelos, LFC está diseñado para trabajar con imágenes en escala de grises y una resolución más baja y tiene una arquitectura más simple, solamente con capas totalmente conectadas.

El modelo **mCNN**, ilustrado en la figura 3b, está diseñado para procesar imágenes de 32x32 píxeles y 3 canales, específicamente del conjunto de datos SVHN. Esta red fue creada por los autores de este trabajo ad hoc.

Por otro lado, el modelo **CNV** [9], representado en la figura 3c, también acepta imágenes de 32x32 píxeles y 3 canales, y está optimizado para el conjunto de datos CIFAR10. A diferencia de mCNN, CNV incorpora un mayor número de capas convolucionales, lo que permite una extracción más profunda de características.

Finalmente, el modelo **VGG11** [29], mostrado en la figura 3d, sigue una arquitectura similar a CNV, con imágenes de entrada de 32x32 píxeles y 3 canales. Sin embargo, VGG11 se distingue por su enfoque en capas convolucionales más profundas y un uso más intensivo de capas de agrupación.

En resumen, se observa que LFC es el modelo menos complejo, solo tiene capas de clasificación. En el caso de mCNN y CNV, tienen una misma cantidad de capas de convolución, pero CNV tiene mayor tamaño de filtro, y además posee una capa totalmente conectada más. Por último, se observa que VGG11 es la red de mayor complejidad, ya que posee la mayor cantidad de capas de convolución, con mayor tamaño de filtro. Además, las capas totalmente conectadas poseen más neuronas que los modelos antes descriptos. De este modo, se observa la diversidad de complejidad de los modelos analizados.

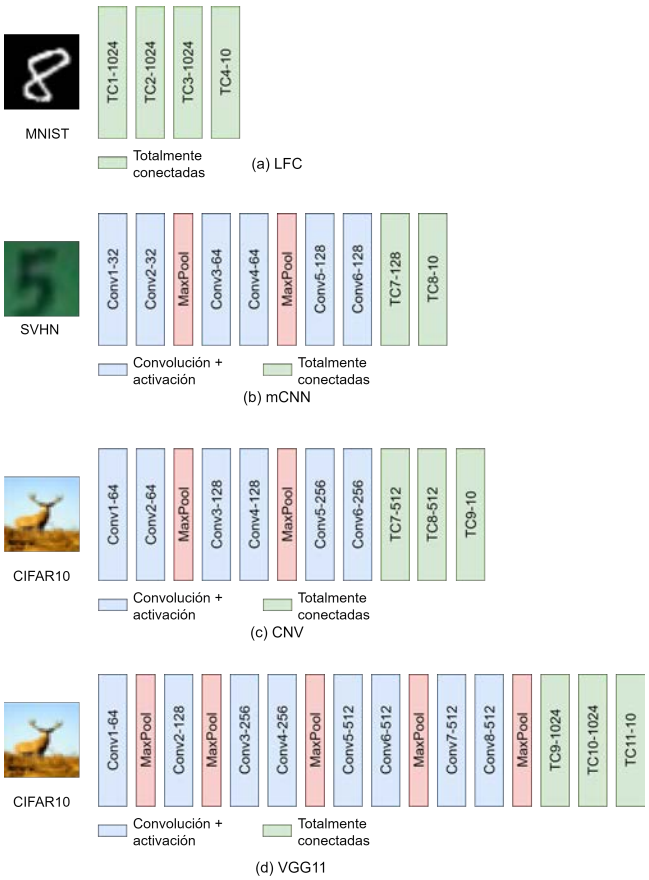


Figura 3: Arquitecturas implementadas

III-C. Procedimiento en FINN

El procedimiento en FINN se lleva a cabo en el contenedor Docker FINN 0.9 dev [31] en Ubuntu 18.04. Se utilizó la versión 2022.2 de Vivado y Vitis HLS. En la Fig. 4 se detalla el flujo de trabajo en FINN.

A continuación se detalla cada uno de los pasos

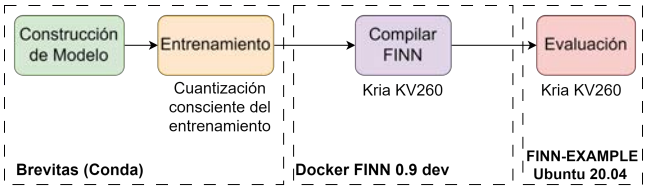


Figura 4: Flujo de desarrollo en FINN

- **Construcción de modelo**: Se realiza la construcción del modelo utilizando capas cuantizadas de Brevitas [18], de acuerdo a la arquitectura de los modelos ya mencionados.
- **Entrenamiento**: De acuerdo a las recomendaciones de la documentación de FINN, y debido a la cuantización consciente del entrenamiento se entrena cada modelo 1000 épocas, en todas las implementaciones realizadas para este trabajo en FINN se cuantiza de forma binaria. Una vez entrenado, el modelo se almacena en un archivo comprimido de extensión tar y es exportado al formato ONNX con las herramientas que proporciona la biblioteca Brevitas.
- **Compilación**: Dentro del contenedor de FINN, en primer lugar, se realiza una estimación de los recursos (LUTs, BRAM, DSP y URAM) y del rendimiento a partir del constructor $build\_dataflow$. Para este paso se definen los parámetros de diseño, cómo el tiempo de reloj, el dispositivo objetivo, y los cuadros por segundo (FPS) deseados. Para todos los casos se utilizó 10 ns como tiempo de reloj, $KV260\_SOM$ como dispositivo y 10000 FPS. Vale la pena aclarar que los FPS definidos en esta es solo una condición de diseño, los FPS resultantes del modelo dependerán de la optimización de los recursos. Para este trabajo, se utilizó el "auto plegado" o "auto folding" que ofrece el constructor, el cual define la paralelización. Aunque es posible mejorar el rendimiento de estos aceleradores ajustando manualmente el plegado, este trabajo se centra en el estudio de entornos de desarrollo automatizados.
  Por último, se llama nuevamente al constructor, pero en este caso realizando todos los pasos (generación de código hls, generación de IP, síntesis de archivo .bit y generación del driver). Este proceso es totalmente automático y se obtiene un archivo de descripción de hardware de extensión hwh y un archivo de extensión bit. Para el caso de la red VGG11 fue necesario habilitar la opción de mapear en memoria Ultra RAM, ya que, los BRAM y LUTS distribuidos no alcanzaban para implementar el modelo.
- **Evaluación**: Los archivos generados en el paso anterior son copiados a la memoria SD de la placa y utilizando un cuaderno de Jupyter, y la biblioteca FINN-EXAMPLE [32], se carga el archivo de extensión bit generado por el compilador. Además, es necesario cargar del conjunto de datos utilizado para evaluar cada modelo. Por último, se lleva a cabo la evaluación de la latencia, el rendimiento, y la medición potencia.

III-D. Procedimiento en Vitis-AI

En Vitis-AI, el procedimiento se llevó a cabo en el contenedor Docker Vitis-AI-2.5.0 en Ubuntu 18.04. En la Fig. 5 se describe el flujo de trabajo de Vitis AI.

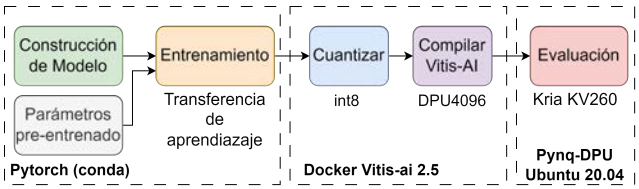


Figura 5: Flujo de desarrollo en Vitis AI

A continuación se detallan cada uno de los pasos:

- **Construcción del modelo**: Se construyo el modelo con capas de Pytorch. En el caso de mCNN, CNV y VGG11, se cargaron los parámetros preentrenados de VGG11 de Torchvision [33]. Para estos modelos, se utilizó la técnica de transferencia de aprendizaje [34], que consiste en congelar las capas de convolución y entrenar solamente las capas de clasificación, este proceso permite obtener precisiones altas, entrenando la red una menor cantidad de épocas.
- **Cuantización**: En esta etapa, dentro de Docker de Vitis AI se carga el modelo de punto flotante generado en el paso anterior y se realiza la cuantización a INT8 con la función $torch\_quantization$ de PyTorch. Luego se evalúa la precisión modelo cuantizado.
- **Compilación**: Se compila el modelo cuantizado para generar las instrucciones de la DPU utilizando vai_c_xir. Para ello, es necesario definir el modelo del DPU, en este caso se utilizó DPUCZDX8G_ISA1_B4096_MAX [13], el cual es compatible con la versión 2.5 de Vitis AI, y está optimizada para la familia Zynq UltraScale+ MPSoC de Xilinx a la cual pertenece la Kria KV260. En este punto se genera un archivo extensión xmodel y se copia en la memoria SD de la placa de desarrollo FPGA.
- **Evaluación**: La evaluación se lleva a cabo en la placa de desarrollo a partir del uso de un cuaderno Jupyter y la biblioteca PYNQ-DPU [35]. En primer lugar, se despliega el archivo de extensión bit del DPU en cuestión, y luego se carga el archivo de extensión xmodel. A partir de los conjuntos de datos seleccionados para cada modelo, se evalúa la latencia, el rendimiento, y la potencia utilizada.

Si bien Vitis AI ofrece Vitis AI Optimizer [36] para reducir la complejidad del modelo, a partir de técnicas de poda, reduciendo pesos redundantes y manteniendo la pérdida de precisión lo más baja posible. Esta herramienta no fue utilizada en este trabajo de comparación, ya que al podar el modelo se está cambiando la cantidad de filtros, pesos y activaciones.

III-E. Preparación de placa de desarrollo Kria KV260

Para evaluar ambos métodos en un mismo sistema operativo, se optó por instalar en la tarjeta SD de la placa de desarrollo Xilinx Kria KV260 el sistema operativo Ubuntu 20.04.3 LTS, especialmente desarrollado para estas placas.

Además, se instalaron las bibliotecas KRIA-PYNQ [37], el cual contiene PYNQ-DPU [35], que permite instanciar una DPU, cargando primero el archivo de extensión bit de la DPU y el archivo con extensión xmodel compilado por el Vitis AI. Y por último, se instaló la biblioteca FINN-EXAMPLES [32], la cual contiene los drivers para cargar los overlays en extensión bit generados por FINN.

Es menester aclarar que para este trabajo se utilizó la versión de Vitis AI 2.5, debido a que es la última versión que tiene compatibilidad con PYNQ-DPU.

III-F. Métricas

Para la evaluación de cada uno de los modelos implementados en cada entorno se utilizaron las métricas de precisión, latencia, rendimiento y potencia media. A continuación se detalla cada una:

- Precisión TOP1: Se calcula como el cociente de imágenes para las cuales el modelo predice correctamente la clase verdadera como la primera opción entre todas las clases posibles y la cantidad total de imágenes. Se expresa en porciento ($\%$).
- Latencia: Se mide el tiempo desde el inicio hasta el final de la inferencia para una sola imagen, se expresa en milisegundos (ms).
- Rendimiento: Se define como la cantidad de imágenes que se infieren en un segundo. Para ello, se utilizaron lotes de 10.000 imágenes. Se expresa en Cuadros por segundos (FPS).
- Potencia media: Se calcula la potencia instantánea a partir de la lectura de los sensores de corriente y tensión de la placa de desarrollo. Luego, se realiza la inferencia completa del conjunto de datos de evaluación correspondiente a cada modelo analizado y se calcula la potencia media. Se expresa en Watts (W)
- Eficiencia Energética: Se calcula a partir del cociente entre el rendimiento y la potencia media, se expresa en cuadros por segundo sobre Watts (FPS/W).
- Recursos Utilizados: Para cuantificar los recursos utilizados, en el caso de FINN, se utilizó los informes que proporciona el constructor $build\_dataflow$ del compilador de FINN, en donde se detalla la cantidad de recursos utilizada en cada compilación. En el caso de VITIS AI, la cantidad de recursos fue extraída de la documentación de Xilinx [13]. Además, la cantidad de recursos disponible de la placa KRIA KV260 se extrajo de la hoja de datos del fabricante [12]. En todos los casos, los recursos evaluados son LUTs, BRAM, DSP y URAM.

IV. Resultados

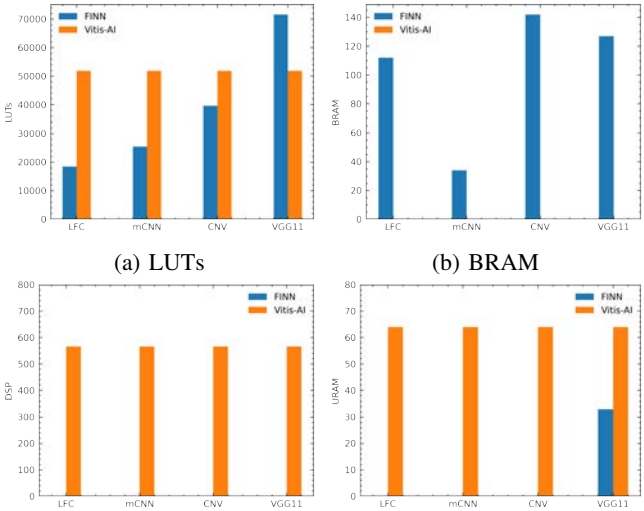
En la tabla II se observan los recursos utilizados para cada modelo. En Vitis AI, todos los modelos son ejecutados en el DPU (4096), en este caso se utiliza el 44 % de los LUTs disponibles. La versión de DPU utilizada no utiliza BRAM, pero utiliza el 45 % de los DPS y el 100 % de la URAM. En la misma tabla se observan los recursos utilizados para cada modelo en la implementación de FINN.

En la Fig. 6a se observa que en FINN a medida que los modelos tienen más capas, se requiere de más LUTs, en cambio, como se dijo anteriormente, la cantidad de recursos

	Disp.	Vitis-AI	FINN			
	KV260	DPU	mCNN	CNV	VGG11	LFC
LUTs	117120	51843	25492	39589	71574	18445
BRAM	144	0	34	142	127	112
DSP	1248	566	0	0	0	0
URAM	64	64	0	0	33	0

Tabla II: Recursos disponibles y utilizados por cada modelo

que usa el DPU de Vitis AI es fija. De forma análoga, se visualiza en la Fig. 6b los bloques BRAM utilizados para cada modelo, donde se observa que VGG11 utiliza menos bloques, eso se debe, a que al habilitar la opción de URAM, balancea ambos recursos, tal como se observa en la Fig. 6d. En el caso de FINN, ningún modelo utilizó DSP, en cambio, el DPU de Vitis AI, si utiliza este recurso, tal como se muestra en la figura 6c. Vale destacar que en FINN solo se utilizan LUTs y BRAM, aunque también existe la posibilidad de permitir el uso de URAM para ciertas capas, en el caso de VGG11, se habilitó esa opción, para que pueda ser compilado, esto se visualiza en la Fig. 6d.



(a) LUTs (b) BRAM

(c) DSP (d) URAM

Figura 6: Comparación de recursos utilizados en cada modelo

En la tabla III se observa las métricas obtenidas en FINN y en la tabla IV se observan los resultados obtenidos en Vitis AI. En ambos casos se muestran la precisión TOP1, la latencia, el rendimiento y la eficiencia energética.

Modelo	Dataset	Prec. TOP1 [%]	Latencia [ms]	Rend. [FPS]	P [W]	Efic. [FPS/W]
mCNN	SVHN	90.07	0.16496	6062	3.8	1595
CNV	CIFAR10	82.62	0.3291	3038	3.9	779
VGG11	CIFAR10	83.51	1.9801	505	4.3	116
LFC	MNIST	98.69	0.0862	11597	3.7	3134

Tabla III: Resultados obtenidos en FINN

Modelo	Dataset	Prec. TOP1 [%]	Latencia [ms]	Rend. [FPS]	P [W]	Efic. [FPS/W]
mCNN	SVHN	96.05	0.922	1315	3.7	355
CNV	CIFAR10	87.57	0.648	2455	5.9	413
VGG11	CIFAR10	88.92	2.400	476	7.2	66
LFC	MNIST	98.83	0.9011	1368	6.5	210

Tabla IV: Resultados obtenidos en Vitis-AI

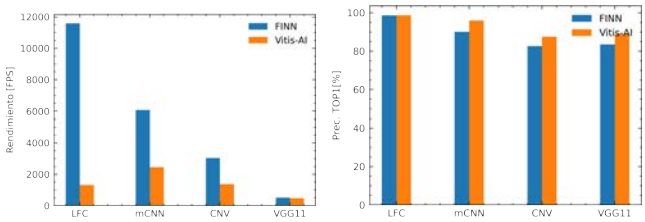
En la figura 7 se observan gráficas de barra que comparan las métricas obtenidas por la implementación en FINN y en Vitis AI de cada modelo, siendo FINN representado por el color azul y Vitis AI por el color naranja.

En la figura 7a se observa que FINN tiene un mayor rendimiento que Vitis AI, pero también se nota que a medida que aumentan la cantidad de capas de convolución, la diferencia de rendimiento disminuye, como el caso de VGG11. En el caso de LFC, modelo que solo contiene capas totalmente conectadas, el rendimiento mejora en más de 8 veces.

En la figura 7b se observa que en la mayoría de los casos, los modelos desarrollados en Vitis AI obtienen mayor precisión, esto es debido a que en FINN se utilizó una cuantización binaria, y esto produce una perdida en la precisión, respecto a la cuantización INT8 de Vitis AI.

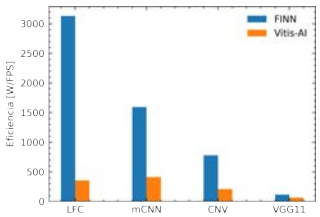
En la figura 7c se observa que la eficiencia es notablemente mayor en los modelos implementados en FINN, pero en los modelos convolucionales a medida que aumentan las capas, las diferencias se acortan.

En la figura 7d, se observa que en todos los casos FINN obtiene menor latencia que Vitis AI. En el caso de las CNN, a medida que aumentan las capas, se disminuye la diferencia.

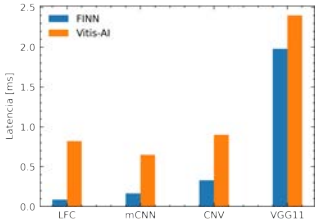


(a) Rendimiento

(b) Precisión TOP1



(c) Eficiencia



(d) Latencia

Figura 7: Comparación de resultados FINN versus Vitis AI

V. Discusión

En este trabajo, se ha realizado una comparativa exhaustiva entre los entornos Vitis AI y FINN para la implementación de redes neuronales convolucionales (CNN) en FPGAs, utilizando la plataforma Kria KV260 de Xilinx. Los resultados muestran diferencias significativas en términos de rendimiento, eficiencia energética y latencia entre los dos entornos, que proporcionan información valiosa para la selección de herramientas adecuadas según los requisitos específicos de las aplicaciones.

- Rendimiento y eficiencia energética: En general, FINN demostró un mayor rendimiento en términos de cuadros por segundo (FPS) en la mayoría de los modelos implementados, especialmente en aquellos con menor complejidad, como el modelo mCNN y el modelo CNV. Esta mayor eficiencia de FINN puede atribuirse a su arquitectura de flujo de datos, que permite una paralelización más fina y una mayor optimización de los recursos hardware disponibles. Además, los modelos implementados en FINN mostraron una eficiencia energética notablemente superior, particularmente en modelos con un menor número de capas. Esto sugiere que FINN es más adecuado para aplicaciones que priorizan tanto el bajo consumo energético como el alto rendimiento en sistemas embebidos. Esto es coincidente con la bibliografía actual.
- Precisión: No obstante, los modelos implementados con Vitis AI presentaron mejores resultados en cuanto a precisión, en parte debido al uso de la cuantización INT8 frente a la cuantización binaria utilizada en FINN. Esto refleja una ventaja de Vitis AI para aplicaciones en las que la precisión es un factor crítico. Además, la facilidad de integración de Vitis AI con entornos de entrenamiento como PyTorch, junto con la posibilidad de emplear técnicas de transferencia de aprendizaje, facilita la obtención de modelos con alta precisión en menor tiempo de entrenamiento.
- Latencia: En cuanto a la latencia, FINN mostró mejores resultados en todos los modelos, con una latencia significativamente menor en la inferencia, lo que la convierte en una opción preferible para aplicaciones de tiempo real. A medida que los modelos se vuelven más complejos (como en el caso de VGG11), la diferencia de latencia entre FINN y Vitis AI disminuye, lo cual indica que las ventajas de FINN en términos de latencia podrían reducirse para modelos más profundos y complejos.
- Recursos: En la comparativa de recursos, se observó que Vitis AI utiliza una cantidad fija, que se debe al DPU, en cambio, FINN asigna los recursos según la complejidad del modelo. Esto permite que en modelos de menor cantidad de capas, FINN utilice una cantidad de recursos significativamente menor. Aunque en VGG11, se observa que los recursos son proporcionalmente similares.
- Escalabilidad: Un hallazgo destacable es que, a medida que aumenta la complejidad de los modelos (número de capas), las diferencias de rendimiento y eficiencia energética entre FINN y Vitis AI disminuyen, como se observa en las Fig. 8a y 8b. En modelos más profundos, como VGG11, las ventajas iniciales de FINN en términos de rendimiento y eficiencia energética se reducen considerablemente. Esto puede influir en la elección del entorno en función del tipo de red que se quiera implementar.

  FINN, utiliza una arquitectura de flujo de datos, ajusta recursos de hardware para maximizar la paralelización del procesamiento, lo que implica que cada capa y operación tiene una unidad de procesamiento dedicada o compartida de manera óptima, según los recursos hardware disponibles. Cuando se requieren más capas de procesamiento, y mayor memoria, esto presenta una limitación, ya que para mantener el alto rendimiento, es necesario mantener la paralelización, y por lo tanto es necesario una mayor cantidad de recursos de hardware,

Tarea	Modelo	Autores	Dataset	Entorno	Dispositivo	Cuantización (bits)	Rendimiento (FPS)	Latencia (ms)	Eficiencia (FPS/W)
Clasificación	mCNN	**Urbano Pintos et al.**	SVHN	FINN	Xilinx Kria KV260	1	**6062**	**0,165**	**1595,26**
				VITIS AI		8	1315	0,922	355,41
	CNV		CIFAR10	FINN		1	**3038**	**0,329**	**778,97**
				VITIS AI		8	2455	0,648	416,10
	VGG11			FINN		1	**505**	**1,980**	**117,44**
				VITIS AI		8	476	2,401	66,11
	LFC		MNIST	FINN		1	**11597**	**0,086**	**3134,32**
				VITIS AI		8	1368	0,901	210,46
	Resnet8	He et al.	CIFAR10	FINN	Xilinx K26	4	**13475**	**0,154**	**2287,78**
				VITIS AI		8	4458	1,293	694,39
Detección	YoloFINN	Machura et al.	VOT	FINN	Avnet Ultra96-V2	4	**111**	**9,009**	**37,58**
				VITIS AI		8	53	18,796	17,10

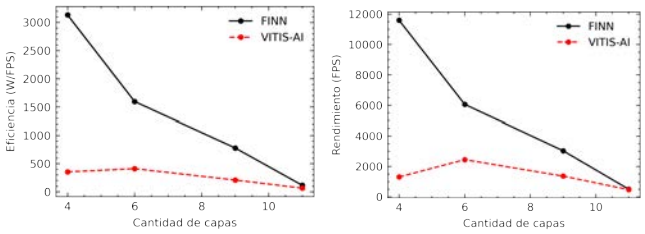
Tabla V: Comparación de FINN y VITIS AI de diferentes implementaciones

elevando el costo y aumentando la complejidad del diseño.

Por otro lado, Vitis AI emplea una arquitectura de overlay en la que el procesamiento de las capas se realiza mediante operaciones secuenciales optimizadas. Este enfoque le permite manejar modelos de mayor complejidad al utilizar una combinación de procesamiento paralelo limitado y acceso a memoria externa para cargar y almacenar datos según sea necesario. Aunque esto implica una mayor latencia y menor eficiencia energética en comparación con FINN, Vitis AI es capaz de manejar modelos grandes de manera más sencilla, sin requerir tantas optimizaciones manuales o ajustes de folding. La combinación de acceso a memoria externa y procesamiento secuencial hace que Vitis AI sea más adecuado para modelos complejos, donde el rendimiento se ve menos impactado por el tamaño del modelo en comparación con FINN, que enfrenta más limitaciones de escalabilidad debido a su dependencia de recursos de hardware.

Por lo tanto, para redes neuronales más profundas, Vitis AI tiende a ser una opción más eficiente, ya que su arquitectura secuencial optimiza el uso de hardware en modelos complejos, mientras que FINN experimenta una caída en rendimiento debido a las limitaciones de recursos.

- Limitaciones: A pesar de los buenos resultados obtenidos con FINN en términos de rendimiento y eficiencia, se observó que para obtener una optimización completa del hardware es necesario realizar ajustes manuales en el proceso de plegado, lo cual puede requerir un mayor esfuerzo de desarrollo. En cambio, Vitis AI ofrece una mayor simplicidad en el flujo de trabajo, lo que lo hace más accesible para desarrolladores que buscan una solución más rápida, aunque esto puede resultar en un uso menos eficiente de los recursos hardware.
- Implicaciones: Los resultados obtenidos sugieren que FINN es una mejor opción para aplicaciones embebidas que demanden alta eficiencia energética y rendimiento con bajo consumo, mientras que Vitis AI puede ser más adecuado para proyectos donde la precisión y la facilidad de desarrollo son prioritarias. Ambos entornos tienen ventajas claras dependiendo de los requerimientos específicos de cada aplicación, lo que refuerza la necesidad de seleccionar la herramienta más adecuada en función de las características del sistema y los objetivos de diseño.



(a) Eficiencia versus capas (b) Rendimiento versus capas

Figura 8: Comparación de Rendimiento y Eficiencia según la cantidad de capas para las implementaciones en Vitis AI y FINN

Este trabajo amplía la comparación existente de estos entornos en la bibliografía, que comparan un modelo en ambos entornos. Nuestra comparación incluye modelos con distintas configuraciones y complejidades, proporcionando una visión más amplia de sus capacidades y limitaciones. En la tabla V se realiza un recuento de comparaciones de la bibliografía de ambos entornos y de los resultados de este trabajo, donde se detalla la tarea que cumple el modelo, los autores que lo implementaron, el conjunto de datos utilizado, el entorno con el que se desarrolló, la cantidad de bits de cuantización, el rendimiento, la latencia y la eficiencia. En todas las comparaciones se destaca con la tipografía negrita la mejor métrica, se puede observar que los modelos implementados en FINN superan a los implementados en VITIS AI en rendimiento, latencia y eficiencia.

VI. Conclusiones

Este estudio ha presentado una comparativa detallada entre los entornos Vitis AI y FINN para la implementación de redes neuronales convolucionales en FPGAs, utilizando la plataforma Xilinx Kria KV260. Se implementaron y evaluaron tres modelos convolucionales y uno basado únicamente en capas totalmente conectadas, analizando aspectos clave como rendimiento, latencia, precisión y eficiencia energética.

Los resultados obtenidos muestran que FINN destaca por su mayor rendimiento y eficiencia energética, siendo especialmente adecuado para aplicaciones que requieren alto rendimiento en sistemas embebidos con restricciones de consumo. En particular, los modelos más simples, como el mCNN y el CNV, se beneficiaron de la arquitectura de flujo

de datos de FINN, mostrando una latencia significativamente menor y una mayor eficiencia energética en comparación con Vitis AI.

Por otro lado, Vitis AI presentó ventajas en términos de precisión, especialmente en modelos más complejos como el VGG11, lo que se debe en parte a su uso de la cuantización INT8. Además, la integración de Vitis AI con entornos de desarrollo como PyTorch, junto con su flujo de trabajo más simplificado, facilita el entrenamiento y despliegue de modelos, lo que lo convierte en una opción adecuada para aplicaciones donde la facilidad de desarrollo y la precisión son prioritarias.

A medida que los modelos aumentan en cantidad de capas (complejidad), se observó una disminución en las diferencias de rendimiento entre ambos entornos. Esto sugiere que, si bien FINN muestra un rendimiento ampliamente superior en modelos con menor profundidad debido a su arquitectura de flujo de datos y su capacidad para minimizar el acceso a la memoria externa, la ventaja se reduce a medida que se implementan modelos más complejos, como VGG11. Esta reducción de rendimiento se debe principalmente al tipo de arquitectura utilizada por cada entorno automatizado.

En resumen, FINN ofrece ventajas claras en términos de rendimiento y eficiencia energética, mientras que Vitis AI sobresale en facilidad de uso y precisión, lo que subraya la importancia de seleccionar el entorno de desarrollo más adecuado en función de las necesidades específicas de cada aplicación.

VII. Trabajo a futuro

Como continuación de este trabajo, se propone ampliar la investigación hacia la implementación de modelos de aprendizaje profundo más avanzados, como autocodificadores y arquitecturas orientadas a la segmentación de imágenes o la estimación de profundidad monocular. Modelos como la red U-Net podrían ser evaluados para tareas de segmentación y reconstrucción de imágenes, lo cual permitiría explorar el comportamiento de FINN y Vitis AI en aplicaciones más complejas y con mayores exigencias de procesamiento.

Otra línea de investigación futura es la optimización manual del proceso de plegado en FINN, con el objetivo de maximizar el rendimiento en modelos complejos. Este proceso permitiría alcanzar una optimización más fina y obtener mejoras significativas en la eficiencia energética sin comprometer la precisión. Y también, sumar a la comaparativa el uso de la herramienta Vitis AI optimizer que permite reducir (a través de la poda) y optimizar los modelos.

Agradecimientos

Los autores expresan su agradecimiento al Departamento de Investigaciones en Láseres y sus Aplicaciones del Instituto de Investigaciones Científicas y Técnicas para la Defensa (CITEDEF), dependiente del Ministerio de Defensa de la República Argentina, por haber facilitado el uso de sus instalaciones y servicios, indispensables para la realización de este trabajo.

Asimismo, agradecen a la Universidad Tecnológica Nacional (UTN) por el financiamiento otorgado a través del Proyecto de Investigación y Desarrollo AST-CHA0008788TC, y extienden su gratitud a la Facultad Regional Haedo de la UTN por el constante apoyo brindado, así como por la disposición de sus recursos e instalaciones.

Referencias

[1] X. Zhao, L. Wang, Y. Zhang, X. Han, M. Deveci, and M. Parmar, "A review of convolutional neural networks in computer vision," *Artificial Intelligence Review*, vol. 57, no. 4, pp. 1–43, 2024.

[2] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, "A survey of convolutional neural networks: Analysis, applications, and prospects," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 12, pp. 6999–7019, 2022.

[3] Z. Zhang and J. Li, "A review of artificial intelligence in embedded systems," *Micromachines*, vol. 14, no. 5, p. 897, 2023.

[4] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *Journal of Machine Learning Research*, vol. 18, pp. 1–30, 2018.

[5] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1," *arXiv: Learning*, 2016. [Online]. Available: http://arxiv.org/abs/1602.02830

[6] T. P. Swaminathan, C. Silver, and T. Akilan, "Benchmarking deep learning models on nvidia jetson nano for real-time systems: An empirical investigation," 2024. [Online]. Available: https://arxiv.org/abs/2406.17749

[7] K. P. Seng, P. J. Lee, and L. M. Ang, "Embedded intelligence on fpga: Survey, applications and challenges," *Electronics*, vol. 10, no. 8, 2021. [Online]. Available: https://www.mdpi.com/2079-9292/10/8/895

[8] XILINX, "Vitis ai - adaptable & real-time ai inference acceleration," 2022. [Online]. Available: https://github.com/Xilinx/Vitis-AI

[9] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "FINN: A framework for fast, scalable binarized neural network inference," *FPGA 2017 - Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, no. February, pp. 65–74, 2017, doi: 10.1145/3020078.3021744.

[10] M. Machura, M. Danilowicz, and T. Kryjak, "Embedded object detection with custom littlenet, finn and vitis ai dcnn accelerators," *Journal of Low Power Electronics and Applications*, vol. 12, no. 2, 2022. [Online]. Available: https://www.mdpi.com/2079-9268/12/2/30

[11] F. Hamanaka, T. Odan, K. Kise, and T. V. Chu, "An exploration of state-of-the-art automation frameworks for fpga-based dnn acceleration," *IEEE Access*, vol. 11, pp. 5701–5713, 2023.

[12] Xilinx, "Kria kv260 vision ai starter kit," 2021. [Online]. Available: https://www.amd.com/en/products/system-on-modules/kria/k26/kv260-vision-starter-kit.html

[13] ——, "Dpuczdx8g for zynq ultrascale+ mpsocs product guide (pg338)," 2023. [Online]. Available: https://docs.xilinx.com/r/en-US/pg338-dpu/Core-Overview

[14] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An imperative style, high-performance deep learning library," *Advances in Neural Information Processing Systems*, vol. 32, no. NeurIPS, 2019.

[15] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[16] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," 2014. [Online]. Available: https://arxiv.org/abs/1408.5093

[17] M. Blott, T. B. Preuber, N. J. Fraser, G. Gambardella, K. O'Brien, Y. Umuroglu, M. Leeser, and K. Vissers, "FinN-R: An end-to-end deep-learning framework for fast exploration of quantized neural networks," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 11, no. 3, 2018, doi: 10.1145/3242897.

[18] A. Pappalardo, "Xilinx/brevitas," 2021, doi: 10.5281/zenodo.3333552.

[19] F. Manca, F. Ratto, and F. Palumbo, "Onnx-to-hardware design flow for adaptive neural-network inference on fpgas," 2024. [Online]. Available: https://arxiv.org/abs/2406.09078

[20] Q. Ducasse, P. Cotret, L. Lagadec, and R. Stewart, "Benchmarking quantized neural networks on fpgas with finn," *arXiv preprint arXiv:2102.01341*, 2021.

[21] 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, doi: 10.1109/5.726791.

[22] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," 2017. [Online]. Available: https://arxiv.org/abs/1708.07747

[23] M. Kristan, J. Matas, A. Leonardis, T. Vojir, R. Pflugfelder, G. Fernandez, G. Nebehay, F. Porikli, and L. Čehovin, "A novel performance evaluation methodology for single-target trackers," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 11, pp. 2137–2155, Nov 2016.

[24] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015. [Online]. Available: https://arxiv.org/abs/1512.03385

[25] N. Urbano Pintos, H. Lacomi, and M. Lavorato, "Implementación de red neuronal de convolución vgg16 en fpga con vitis ai," in *Libro de resúmenes de la 109a Reunión de la Asociación Física Argentina*, 2024, pp. 47–48.

[26] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, doi: 10.48550/ARXIV.1409.1556. [Online]. Available: https://arxiv.org/abs/1409.1556

[27] A. Krizhevsky, "Learning multiple layers of features from tiny images," *University of Toronto Department of Computer Science*, 2009.

[28] N. Urbano Pintos, H. Lacomi, and M. Lavorato, "B-vgg16: Red neuronal de convolución cuantizada binariamente para la clasificación de imágenes," *Elektron*, vol. 6, no. 2, pp. 107–114, 2022.

[29] S. Liu and W. Deng, "Very deep convolutional neural network based image classification using small training sample size," *Proceedings - 3rd IAPR Asian Conference on Pattern Recognition, ACPR 2015*, pp. 730–734, 2016, doi: 10.1109/ACPR.2015.7486599.

[30] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011.

[31] XILINX, "Fast, scalable quantized neural network inference on fpgas," 2024. [Online]. Available: https://github.com/Xilinx/finn

[32] ——, "Dataflow qnn inference accelerator examples on fpgas," 2024. [Online]. Available: https://github.com/Xilinx/finn-examples

[33] T. maintainers and contributors, "Torchvision: Pytorch's computer vision library," https://github.com/pytorch/vision, 2016.

[34] A. Farahani, B. Pourshojae, K. Rasheed, and H. R. Arabnia, "A concise review of transfer learning," 2021. [Online]. Available: https://arxiv.org/abs/2104.02144

[35] XILINX, "Dpu on pynq," 2022. [Online]. Available: https://github.com/Xilinx/DPU-PYNQ

[36] Xilinx, "Vitis ai optimizer," 2023. [Online]. Available: https://docs.amd.com/r/en-US/ug1414-vitis-ai/Vitis-AI-Optimizer

[37] XILINX, "Kria-pynq," 2022. [Online]. Available: https://github.com/Xilinx/Kria-PYNQ