ISSN 1870-4069

Evaluación de modelos DNN para la detección de objetos en dispositivos de cómputo en la frontera

Alberto Pacheco[1], Ever A. Flores[2], Edgar Trujillo[1]

[1] TecNM / Instituto Tecnológico de Chihuahua,
Posgrado e Investigación, Chihuahua,
México

Robert Bosch GmbH, Guadalajara,
México

{alberto.pg, edgar.tp}@chihuahua.tecnm.mx,
[2] Ever.FloresAvila@mx.bosch.com

Resumen. Dado el creciente interés en aplicar aprendizaje automático dentro del Internet de las Cosas, TinyML emerge para resolver problemas asociados con la adaptación de modelos de redes neuronales profundas para correr en dispositivos pequeños de bajo costo. En este trabajo, se enlistan algunos dispositivos de la frontera con potencial para ser empleados en TinyML. Se presenta una revisión de la literatura de trabajos relacionados sobre el tema a tratar. El trabajo consistió en abordar el problema de evaluar y seleccionar un modelo DNN para la detección de objetos que sea capaz de correr en diversos dispositivos de cómputo en la frontera con procesadores ARM de bajo costo y escasa memoria. Se logró correr un modelo DNN en todos los dispositivos de prueba; se detectó un rendimiento 40x superior en dispositivos ARMv8 respecto a ARMv7, y es posible mejorar su rendimiento hasta un factor 10x usando aceleradores neuronales. Para ejecutar estos modelos DNN en dispositivos pequeños se recomienda aplicar técnicas de optimización para reducir su tamaño y complejidad.

Palabras clave: Redes neuronales profundas, internet de las cosas, TinyML.

Benchmarking of Edge Computing for Object Detection DNN Models

Abstract. Given the growing interest in Machine Learning applied for IoT, TinyML arises as a new research area with the goal of performing on-device inferencing. This work reviews some potential inference computing edge devices. A literature review of recent works is also introduced. The present work carried out an object detection DNN model benchmark for diverse low-cost, low-memory ARM-based devices. The experimental results showed that one DNN model was able to run on all testing devices, where ARMv8-based test devices outperformed ARMv7 by 40 times, while DNN accelerators can boost

performance up to 10 times. To perform on-device inferences in smaller devices, it is also recommended to apply a compression technique.

Keywords: DNN, IoT, TinyML.

1. Introducción

Explosión de datos y dispositivos. Según [1-3] se proyecta para 2021 sea triplicado con respecto a 2018, el tráfico de red proveniente de la nube (*cloud computing*), así como la incorporación para 2030, de 20 a 50 mil millones de dispositivos IoT (*Internet of Things*), mismos que, aparte del creciente número de *smartphones*, se estima incrementen hasta 10 veces la cantidad de datos en Internet, y que el mercado global IoT pase de 16 a 186 mil millones de dólares durante el período 2016-2023.

Retos actuales. Al considerar este crecimiento, tanto de datos como dispositivos IoT, es probable que resulte insuficiente la infraestructura de telecomunicaciones, derivando en mayores latencias de red e insuficiencias en la capacidad de almacenamiento y procesamiento, privacidad y seguridad. Por ello, de acuerdo con [4-8], en el futuro inmediato, entre los principales retos para IoT figuran: ¿cómo extraer información contextual analizando una gran cantidad de datos proveniente de entornos físicos dinámicos y ruidosos? y ¿cómo procesar datos con algoritmos cada vez más inteligentes de una forma eficiente, segura y sustentable?

Estado del arte. El cómputo en la frontera (*edge computing*) ofrece la posibilidad de procesar información de forma distribuida y en tiempo real, con pequeños dispositivos alimentados por baterías, mismos que adquieren datos directamente de sus propios sensores. El cómputo en la frontera ofrece una forma muy eficiente en términos de consumo de energía, latencia y privacidad sin necesidad de saturar la red [4-6]. Por otra parte, las modelos de redes neuronales profundas (*deep neural network*, DNN) son algoritmos muy eficaces para el reconocimiento de patrones en información vasta, compleja y ruidosa.

Al construir y aplicar modelos DNN existen dos etapas: el entrenamiento y la inferencia (ejecución del modelo DNN). TinyML representa el estado del arte en IoT [7-10], al potenciar las capacidades del cómputo en la frontera para la etapa de inferencia dentro de dispositivos pequeños de bajo costo y memoria reducida, posibilitando el procesamiento de datos de manera inmediata y privada, evitando así, la latencia y congestionamiento de la red, cubriendo además, un amplio espectro de aplicaciones IoT, tales como: eHealth, smart spaces, transporte inteligente, industria 4.0 y agricultura de precisión, entre otros [13-14, 29-30].

Planteamiento del problema. El presente trabajo consiste en evaluar y seleccionar un modelo DNN para detección de objetos que sea capaz de correr en diversos dispositivos con procesadores ARM de bajo costo y escasa memoria, esperando obtener un rendimiento de 2-5 FPS para soportar la viabilidad de su aplicación [16]. La Tabla 1 plantea la arquitectura de un sistema TinyML orientado a la detección de objetos en secuencias de vídeo (a), usando modelos DNN pre-entrenados basados en redes convolucionales (b), para la etapa de inferencia (c), sin esquemas de simplificación o reducción de modelo DNN (d), usando distintos frameworks ML para comparar el rendimiento del modelo DNN corriendo en diferentes dispositivos y plataformas (f-h).

Tabla 1. Sistema TinyML basado en redes neuronales.

Dimensión	Características	Opciones
Aplicaciones	a) Visión por computadora	Detección de objetos
Modelos y arquitecturas	b) Modelos y arquitecturas de redes neuronales	CNN, FF, RNN, GAN, LSTM, SOM, SAE, SVM, ART, etc
	c) Entrenamiento e inferencia de modelos	Supervisado, no supervisado. Prueba, validación, aplicación.
	d) Optimización de modelo DNN	Cuantización, poda, SVD, compresión, destilación, etc.
	e) Frameworks y formatos	Keras, Caffe, TensorFlow, Pytorch, CoreML, MXNet, OpenVino, etc.
Hardware	f) Computadora (procesadores)	CPU, GPU, TPU.
	g) Sistema operativo	Linux, Windows, macOS, iOS, RT.
	h) Acelerador neuronal	FPGA, DSP, NPU, acelerador DNN (Movidius, Coral), etc.

El presente trabajo es la segunda de tres etapas de un proyecto concluido recientemente [15-16], para la cual se establecieron los siguientes subproblemas:

Comparar el rendimiento de una red neuronal DNN entrenada para la detección de objetos a partir de imágenes de vídeo, cuya etapa de inferencia sea verificada en distintos dispositivos.

- Determinar el flujo de trabajo necesario para portar y habilitar modelos DNN pre-entrenados (deployment) en diversos dispositivos.
- Elegir un modelo DNN para detección de objetos con base en los criterios de eficiencia y portabilidad para el ecosistema TinyML heterogéneo.
- Caracterizar y determinar cuál configuración ofrece el mejor rendimiento para el modelo DNN elegido.

2. Trabajos relacionados

La ejecución optimizada de modelos ML en dispositivos de bajo costo es un campo emergente de desarrollo e investigación. En 2015, DeepEar [13], un trabajo pionero, ejecutó un modelo DNN para reconocimiento de voz en un DSP de baja potencia. El consorcio de benchmarks para sistemas embebidos (EEMBC), liberó en 2019 la primera prueba TinyML denominada MLMark [17]. Como revelan [18-19] y eventos como TinyML Summit 2019-2021 [20], existe un creciente número de fabricantes y desarrolladores de dispositivos y aplicaciones para TinyML.

A continuación, revisamos la evidencia existente en la literatura de investigación. Se realizó una búsqueda en marzo de 2021 en la biblioteca digital de ACM, en la colección *The ACM Guide to Computing Literature* (2,923,187 registros) buscando dentro de los resúmenes (*abstract*) de "*research articles*", con una expresión de búsqueda para

Tabla 2. Arquitecturas para realizar inferencias ML [14].

	Cloud ML	**Mobile ML**	**Tiny ML**
Memoria (activación)	>16GB	4GB	200-400Kb
Memoria (pesos DNN)	>TB	<256GB	<1MB

trabajos sobre inferencia ML/CN/DL o TinyML en dispositivos móviles, edge, fog, IoT, embebidos con CPUs ARM, NPUs, FPGAs con/sin técnicas de compresión de modelos DNN, excluyendo cómputo en la nube, big data, TPUs, etc. para 2010-2021 [21]:

```
[[inferenc* OR optim*] AND [ml OR cnn OR dl OR ai OR "machine learning" OR tinyml
OR "deep learning"] AND [edge OR fog OR "mobile device" OR "on-device" OR fpga
OR arm OR smartphone OR npu OR iot OR embed*] AND NOT [cloud OR "large-scale"
OR cluster OR server* OR "big data" OR tpu] AND [Publication Date: (01/01/2010
TO 12/31/2021)]
```

Se encontraron 506 artículos (Tabla 6), distribuidos como sigue: 327 en *Proceedings* y 153 en *Journals* y revistas periódicas, destacando *ACM Trans. on Embedded Computing Systems* con 11 artículos y *Proc. ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* con 9 artículos. Entre las principales organizaciones editoriales se encuentran: ACM con 323, IEEE con 101 y Elsevier con 26 artículos.

Dentro del Top 10 de las organizaciones participantes destacan: 2 universidades chinas (Pekín y Tsinghua con 26), 3 universidades americanas (Stanford, Illinois y California con 19), 2 empresas americanas (Intel y Microsoft con 17), 2 universidades de Singapur (Nayuang y Singapur con 12) y una universidad inglesa (8 Cambridge).

También, en la misma fecha, se realizó la misma búsqueda en IEEEXplore (5,271,145 ítems disponibles), donde se encontraron 1,817 artículos (Tabla 6) con 1,352 artículos de Proceedings, 443 de Journals y 22 de Magazines IEEE entre los años 2010 y 2021, usando la misma expresión de búsqueda [22].

Destacan IEEE Access (135), seguido de IEEE Internet of Things Journal (31). Entre los países con más artículos aparecen: 23 artículos americanos, 17 chinos, 8 ingleses y 4 de Singapur. Como muestra la Tabla 6, el promedio de crecimiento anual es alrededor de 30% para 2019-2020 considerando ambas (IEEE y ACM).

En cuanto a los artículos con un mayor número de citas por año en IEEE más citados se encuentran [23-25] con 130, 88 y 83 citas/año, respectivamente. Los más citados en ACM DL son [26-28] con 130, 38 y 38 citas/año, respectivamente.

Trabajos relacionados. En síntesis, existen tres categorías de trabajos relacionados con la ejecución de modelos DNN en dispositivos pequeños y limitados [28-29]: 1) los modelos DNN ligeros (*small footprint DNN models*) entrenados para tareas muy simples y específicas, tales como GooLeNet, ResNet, SENet, SuffleNet; 2) la incorporación de unidades de aceleración de modelos DNN, como DeepX, DeepSense, DeepMon, ProjectionNet, QNNPack, Movidius, Coral; 3) la aplicación de diversas técnicas para re-entrenar, reducir tamaño y complejidad de modelos DNN, tales como la poda, cuantización, compresión, destilación, entre otros. El presente trabajo, se enfoca exclusivamente a las dos primeras categorías.

Tabla 3. Tiny ML en procesadores ARMv7 (32 bits).

Dispositivos (32 bits)	**CPU ARMv7 (núcleos)**	**Reloj RAM Flash**	**Consumo energía (Watts)**	**Precio (año)**
Raspberry Pi Pico	Cortex-M0+ (2 núcleos)	133 MHz 264 KB 2 MB	0.1 W	$4 (2021)
Espressif ESP32	Xtensa LX6 (2 núcleos) 0.006 GFLOPS	240 MHz 320 KB 4 MB	0.2 W	$10 (2016)
SparkFun Edge	Cortex-M4F (1 núcleo)	48 MHz 384 KB 1 MB	0.06 W	$15 (2019)
Silicon Labs SLTB004A	Cortex-M4 (1 núcleo)	40 MHz 256 KB 1 MB	0.1 W	$20 (2018)
Arduino Nano 33 BLE Sense Adafruit Circuit Playground	Cortex-M4F nRF 52840 (1 núcleo)	64 MHz 256 KB 1MB	0.2 W	$30 (2019) $20 (2018)
Adafruit Edge/Py Badge Wio Terminal	Cortex-M4F ATSAMD51 (1 núcleo)	120 MHz 192 KB 512 KB	~0.3 W	$35 (2019) $30 (2015)
SMT32F746 Discovery kit	Cortex-M7 STM32F746 (1 núcleo) 0.022 GFLOPS	216 MHz 340 KB 1 MB	~0.9 W	$50 (2015)

Tabla 4. Dispositivos para TinyML con procesadores ARMv8 (64 bits).

Dispositivos (64 bits)	**CPU ARMv8 (núcleos)**	**Reloj RAM Flash**	**Rendi-miento GFLOPs**	**Potencia (watts)**	**Precio**
Raspberry Pi 3B	Cortex-A53 (4 núcleos)	1.2 GHz 1 GB	1-4	2-5 W	$35 (2016)
Raspberry Pi 4	Cortex-A72 (4 núcleos)	1.5 GHz 2/8 GB	5-10	5-8W	$35/75 (2019)

3. Dispositivos para TinyML

En la Tabla 2 se describen las propiedades típicas de los elementos de cómputo para los principales tipos de sistemas con modelos DNN. A diferencia de los predominantes sistemas de cómputo en la nube, para el cómputo en la frontera, interesa principalmente ejecutar modelos DNN en dispositivos móviles y dispositivos IoT o embebidos que cuentan con un hardware muy limitado y de menor costo, conocidos de distintas formas: *Tiny ML, edge AI, smart IoT, on-device inferencing y edge inference* [8-11, 12-13].

Debido a que los modelos DNN poseen muchas capas y millones de parámetros de punto flotante, típicamente float32, existe un campo muy activo de investigación

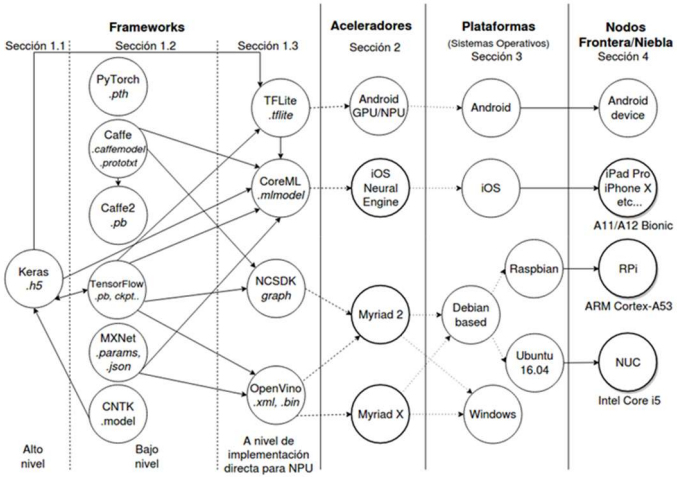

Fig. 1. Flujo de trabajo para transferir modelo DNN a los distintos dispositivos.

enfocado en la reducción y compresión de dichos modelos DNN a escalas y precisiones más acordes para dispositivos pequeños, brindando como beneficio, la posibilidad de procesar grandes cantidades de datos complejos, dinámicos e inciertos, permitiendo así procesar flujos continuos de datos adquiridos de dispositivos sensores que van desde video, micrófonos, acelerómetros, altímetros, termómetros, etc. para identificar patrones, eventos y objetos de interés directamente en la frontera de la red.

Revisión de dispositivos ARMv7 con potencial para TinyML. Como muestra la Tabla 3, a diferencia de los equipos de cómputo en la nube, existe una categoría de dispositivos embebidos (SoC) para TinyML con procesadores de 32 bits (ARMv7) que cuestan menos de $50 dlrs, corren a ~300 MHz, tienen 200-400 KB RAM y consumen miliwatts (usan pilas). A su vez, podemos segmentar esta categoría en dos partes (Tabla 3): a) los dispositivos más pequeños, con muy bajo consumo energético y más baratos (menos de $10 dlrs); b) los dispositivos con precio de $15-50 dlrs, CPU Cortex-M (1 núcleo), que pueden correr modelos DNN muy simplificados y reducidos.

Dispositivos ARMv8 para TinyML. La Tabla 4 muestra equipos de 64 bits de bajo costo (ARMv8), con sistemas operativos tipo Linux o para dispositivos móviles (smartphones), consumo de 2-8 watts, y suficiente memoria (hasta 8 GB RAM) para ejecutar modelos DNN de medianas proporciones. En la Tabla 4 solo se muestran modelos similares a los usados en el trabajo (RPi). Esta categoría está ganando gran auge y nuevos modelos en el mercado (e.g. RPi4), y el surgimiento de la nueva arquitectura ARMv9 para TinyML.

Aceleradores DNN para TinyML. La Tabla 5 muestra aceleradores de hardware para modelos DNN [31], cuya arquitectura interna y conjunto de operaciones que pueden ejecutarse en paralelo y son más específicas para ejecutar modelos DNN, acelerando hasta 10 veces más la ejecución comparada con CPUs de precio similar, consumiendo además menos energía.

Algunas de las tarjetas de desarrollo de la Tabla 5 son más sofisticadas porque incluyen varios procesadores (ARM, GPU, DSP, VPU, TPU multicore) que elevan considerablemente el costo y consumo de potencia.

Tabla 5. Tarjetas de desarrollo TinyML y aceleradores DNN.

Dispositivo	**Unidades de aceleración**	**Arquitectura CPU**	**Rendi-miento GFLOPs (FP16)**	**Consumo de energía**	**Precio (año)**
Intel Movidius (NCS)	VPU 12 cores 4 GB RAM	Ninguno	40	1 W	$75 (2017)
Intel Movidius 2 (NCS2)	VPU 16 cores 4 GB RAM	Ninguno	~100	1.5 W	$75 (2018)
Google Coral Accelerator	TPU	Ninguno	~250	2 W	$75 (2019)
NVIDIA Jetson Nano	GPU 128 cores 4 GB RAM	Cortex-A57 4 cores	500	10 W	$130 (2019)
Google Coral Dev Board	TPU GPU 1GB RAM	Cortex-A53 4 cores	~600	15 W	$150 (2019)
ASUS Tinker Edge R	TPU GPU NPU 6 GB RAM	Cortex-A72+A53 6 cores	~1,200	15W	$250 (2020)
NVIDIA Jetson TX2	GPU 256 cores 8 GB RAM	Cortex-A57 4 cores +ARMv8 2 cores	1,300	15 W	$400 (2021)
NVIDIA Jetson AGX Xavier	GPU 512 cores 32 GB RAM	ARMv8 8 cores	11, 000	50 W	$700 (2021)

4. Metodología

Para realizar exitosamente la comparación de los tiempos de inferencia en modelos DNN en diversos dispositivos fue necesario llevar a cabo un flujo de trabajo (*workflow*) para portar los modelos DNN a los formatos específicos de cada plataforma y hardware de los dispositivos de prueba, lo cual, a su vez permitió derivar una perspectiva práctica para implementar las pruebas en distintos tipos de procesadores, siguiendo para ello, la siguiente metodología:

a. Etapa de investigación: la primera etapa consistió investigar modelos DNN pre-entrenados viables de ser ejecutados en cada dispositivo de prueba, con el objetivo de obtener al menos un modelo DNN capaz de ser ejecutado en todos los dispositivos para asegurar la confiabilidad de la evaluación comparativa.

b. Etapa de implementación: en esta etapa se realizó la conversión e implementación del modelo DNN para cada dispositivo de prueba.

Tabla 6. Artículos publicados entre 2010-2020.

Año	Artículos ACM	Artículos IEEE
2010	7	22
2011	3	22
2012	9	25
2013	9	18
2014	9	26
2015	14	48
2016	44	74
2017	53	152
2018	77	282
2019	109	470
2020	143	594

c. Etapa de evaluación: finalmente, se calculó el tiempo de procesamiento de detección de objetos presentes en imágenes de video, considerando la métrica de cuadros por segundo (*frames per second*, FPS) para luego, comparar y analizar los resultados obtenidos.

5. Desarrollo

A continuación, se presentan los dispositivos de prueba considerados en esta evaluación comparativa, así como sus características principales:

- Computadora Intel NUC 6-i5SYH con i5-6260U 64-bit CPU @ 1.8 GHz, 16 GB RAM (15 watts) con Ubuntu 16.04 LTS (Linux Kernel 4.4) instalado.
- Computadora Raspberry Pi 3B (RPi), quad-core ARMv8 64-bit CPU @ 1.2GHz, 1GB RAM (5W) con Raspbian 2018 (Linux Kernel 4.9) instalado.
- Dos aceleradores DNN intel Movidius Neural Compute Stick (NCS) con VPU Myriad 2 de 1 watt y una NCS2 con VPU Myriad X de 1.5 watts.
- Un dispositivo móvil Apple iPad 6th-Gen 9.7" con iOS 12, quad-core A10 Fusion ARMv8 64-bit CPU @ 2.34 GHz y 2 GB RAM.
- Un teléfono inteligente Apple iPhone 8 con iOS 12, hexa-core A11 Bionic ARMv8 64-bit CPU @ 2.39 GHz y 2 GB RAM.

Etapa de investigación. Se seleccionaron 5 modelos DNN pre-entrenados para la detección de objetos: YOLOv3, TinyYOLOv2, TinyYOLOv3, SSD + MobileNet y SSDLite + MobileNetv2. La Figura 1 sumariza el flujo de trabajo efectuado para intercambiar el formato de un modelo DNN para soportar cada uno de los distintos sistemas operativos, dispositivos de prueba y aceleradores DNN.

Frameworks ML (sección 1). Los modelos DNN son muy dependientes del formato del framework ML en que fueron entrenados. Esto representa un problema al momento de tratar de ejecutarlos en un hardware distinto, donde dependiendo del fabricante, puede que no exista soporte para un determinado formato. Sin embargo, es posible convertir el formato de un modelo DNN, como muestra la sección 1 de la Figura 1, donde las flechas representan las conversiones de formato y los nodos indican el formato soportado por cada framework (segundo renglón de cada nodo). En la sección

Tabla 7. Resultados de rendimiento (FPS) de cada modelo DNN en cada dispositivo.

		YOLOv3	**Tiny YOLOv3**	**Tiny YOLOv2**	**SSD + MobileNet**	**SSDLite + MobileNetv2**
RPi	CPU	-	0.5	0.5	-	-
	1 NCS	0.5	3.7	2.5	9.6	6.0
	2 NCS	-	-	5.8	13.4	-
	1 NCS2	1.0	3.9	3.2	11.9	6.5
NUC	CPU	1.8	14.7	5.5	-	54.0
	1 NCS	0.7	6.5	4.7	11.3	9.0
	2 NCS	1.3	13.0	12.4	19.0	-
	1 NCS2	2.7	30.0	10.2	20.0	10.0
iOS	iPad 6G	2.5	-	5.9	-	6.0
	iPhone 8	3.0	-	27.0	-	9.0

1.1 aparece el framework ML de alto-nivel Keras, que presenta múltiples facilidades para la creación, entrenamiento y modificación de modelos DNN.

Los frameworks de la sección 1.2 son herramientas de bajo nivel con las cuales se realiza el diseño y evaluación de modelos de manera más eficiente y con mayor flexibilidad en comparación con una herramienta de alto nivel como Keras. En este punto, es posible convertir desde Keras a casi cualquier otro formato de bajo nivel pasándolo por ejemplo al formato ONNX, y viceversa, incluyendo también otros como OpenVino y CoreML.

Finalmente, en la sección 1.3 se encuentran los frameworks para instalar el modelo DNN (*deployment*) directamente en algún dispositivo o aceleradores DNN, en otras palabras, en los formatos nativos para cada equipo. Se recomienda convertir de un formato a otro de manera directa, ya que a pesar de que exista el soporte de conversión, en ocasiones los modelos DNN son diseñados con operaciones o capas muy específicas de un determinado framework ML, ocasionando que durante la conversión algunas operaciones sean modificadas y afecten la calidad o el funcionamiento del mismo.

Aceleradores DNN, sistemas operativos y dispositivos (secciones 2-4). En la sección 2 de la Figura 1 se señalan cuáles formatos son compatibles con los aceleradores DNN y cómo puede afectar además el tipo de sistema operativo instalado en el dispositivo de prueba al que se conectan por puerto USB (sección 3).

Si bien una manera de evitar los problemas de incompatibilidad de formatos es creando nuevos modelos DNN para cada caso de aplicación, pero esta opción es poco viable en la mayoría de las ocasiones debido a que si bien la arquitectura de la red neuronal se puede replicar en otro framework, sería necesario entrenar ese modelo, y con ello surge la necesidad de grandes cantidades de datos y capacidades de cómputo para entrenamiento, por ello la importancia de conseguir reutilizar modelos pre-entrenados para convertirlos a otras plataformas.

En este trabajo se realizaron las conversiones de formato indicadas hasta llegar a los dispositivos de prueba de la sección 3, exceptuando la plataforma Android que aparece como posible referencia.

Etapas de implementación. Se utilizó TensorFlow para realizar las conversiones de formatos para los equipos NUC y RPi. Adicionalmente, en estos dos dispositivos se hicieron evaluaciones agregando los aceleradores Intel Movidius, en cuyos casos se empleó OpenVino para convertir del formato de TensorFlow para los modelos DNN

ejecutados en dichos aceleradores, cuyo formato OpenVino consiste de dos archivos, uno para almacenar la información de la estructura de la red neuronal (XML) y otro para almacenar los pesos de la red (formato binario). De manera similar, para los dispositivos iOS, los modelos DNN se convirtieron a formato. MLmodel para ser procesados por medio del framework CoreML de iOS.

Etapa de evaluación. Para evaluar los modelos DNN se desarrollaron distintos scripts en Python para ejecutarlos en las computadoras de prueba (NUC, RPi) y una aplicación móvil de prueba en iOS. Las pruebas consistieron en obtener imágenes de video de una webcam, realizar el preprocesamiento correspondiente y procesar el modelo DNN para obtener las inferencias y desplegar resultados de la detección de objetos sobre la imagen de vídeo de salida en pantalla.

Para comparar resultados de rendimiento se calculó el promedio de cuadros de video (FPS) procesados por segundo para una misma muestra de video de prueba consistente en 200 cuadros de video de entrada. En la Tabla 7 se muestran los resultados obtenidos para dispositivos y modelos DNN de detección de objetos, siendo TinyYOLOv2 el único modelo que pudo ser ejecutado exitosamente en todas las siguientes combinaciones:

- Con un acelerador DNN Movidius NCS (Myriad 2).
- Con un acelerador DNN Movidius NCS2 (Myriad X).
- Con dos aceleradores DNN Movidius NCS.
- En una computadora con CPU 64 bit NUC Intel Core i5.
- En una computadora con CPU 32 bit RPi ARM Cortex-A53.

6. Resultados

Si asumimos como rendimiento mínimo aceptable para soportar una aplicación práctica para detección de objetos en video un rango ubicado entre 2 y 5 FPS [32], tenemos que, si bien, el equipo de prueba con menor capacidad (RPi ARMv8) logró ejecutar 2 de 5 modelos DNN (los modelos más pequeños TinyYOLO), su rendimiento resultó inadecuado en términos prácticos (0.5 FPS).

Sin embargo, al incorporar uno o dos aceleradores DNN en ese mismo equipo, se logró un resultado aceptable en 4/5 casos, destacando incluso que para TinyYOLOv2 (5.8 vs 5.5 FPS) superó al equipo con CPU de mayor capacidad (NUC i5) y se observa que 2 unidades NCS brindan mejores resultados que 1 unidad NCS2 para TinyYOLO.

Desafortunadamente, los modelos DNN más demandantes (YOLOv3 y MobileNet) no se lograron ejecutar en la computadora RPi usando CPU ni tampoco usando dos aceleradores DNN, debido en este último caso al alto consumo de energía que demandaban. Sin embargo, múltiples aceleradores corrieron sin problema en la computadora NUC que, además, al contar con puertos USB3.0, logrando así el mejor rendimiento de dichas unidades de aceleración, destacando que, en 4 de 5 casos, el rendimiento del acelerador NCS2 superó en alrededor de 45% al procesador de la computadora (Core i5).

Finalmente, en dispositivos móviles iOS se ejecutaron 3 modelos DNN, donde destaca que se logró el mayor rendimiento (iPhone 8) para el modelo pequeño TinyYOLOv2 (27 FPS) y en los demás modelos DNN más pesados se logró un rendimiento promedio, que sin embargo, demuestra que los dispositivos móviles más

recientes tienen buenas capacidades para ejecutar este tipo de modelos DNN, donde resulta interesante las capacidades que ofrece el procesador del iPhone 8, A11 Bionic con 6 núcleos (2 núcleos de alto rendimiento), un GPU de 3 núcleos y un nuevo procesador neuronal integrado (*neural engine*), que sin embargo, no fue posible determinar en las prueba realizadas con este dispositivo, en qué tipo de unidad de procesamiento (CPU, GPU, NPU) fueron realizadas las inferencias de los modelos DNN probados.

Por último, es posible observar algunas variaciones no proporcionales de los rendimientos entre los diferentes modelos y dispositivos (e.g. 12.5 FPS de NCS2 en NUC para TinyYOLOv2 y 54 FPS con CPU de NUC para MobileNetv2). En este trabajo no se profundizó en esos casos, aunque es útil señalar que los aceleradores están optimizados para cierto tipo de operaciones, por lo cual a veces un CPU puede ser más eficiente dependiendo de la arquitectura específica de un modelo DNN, y algo similar ocurre entre distintos tipos de CPU.

7. Conclusiones

En este trabajo se logró convertir y ejecutar 5 modelos DNN para detección de objetos en distintas combinaciones de dispositivos y aceleradores DNN. Debido a su reducido tamaño y mayor compatibilidad (tipos de capas/operaciones), el modelo TinyYOLOv2 fue el único modelo ejecutado con éxito en todos los casos de prueba.

Destaca también la importante mejora en el rendimiento que proporcionan los aceleradores DNN, sobre todo en los dispositivos de menos capacidad, brindando así una opción de bajo costo y consumo de energía. También resulta muy notable la capacidad y el nivel de desempeño que brindan los teléfonos inteligentes de alta gama para ejecutar modelos DNN y, por tanto, posibilitan una nueva generación de aplicaciones móviles donde cada vez es más posible aplicar la Inteligencia Artificial en la vida cotidiana.

Se encontró además que para ejecutar modelos DNN en dispositivos pequeños (TinyML), hay dos obstáculos importantes: a) el excesivo tamaño de la mayoría de los modelos DNN supera, por lo general, la capacidad de procesamiento y memoria disponible en dichos dispositivos; b) una de las tareas más complicadas es migrar dichos modelos de su formato original al formato de la plataforma-destino, donde las diferentes capas y operaciones del modelo que cada plataforma/framework soporta, pueden en el proceso de conversión alterar la estructura interna del modelo o terminar siendo incompatible.

Por ello, para aplicaciones que incluyan distintos dispositivos, sean móviles, embebidos o aceleradores DNN, es útil comprender primero el grado de compatibilidad que soportan, donde la Figura 1 sirvió de guía en este trabajo, al recopilar distintas rutas posibles para convertir el formato de un modelo DNN. Por otro lado, los frameworks CoreML (iOS) y OpenVino (aceleradores DNN) soportan un gran número de conversiones de formato y tipos de operaciones, elevando su grado de compatibilidad para soportar más plataformas de hardware.

Como trabajo a futuro, se considera evaluar dispositivos recientes (RPi 4, nuevos procesadores móviles y las nuevas arquitecturas RISC-V y ARMv9). Para la siguiente etapa del proyecto, existe una gran oportunidad de mejora en el rendimiento de modelos

DNN al aplicar técnicas de optimización, para podar, comprimir y acelerar los modelos antes de su despliegue, buscando así, desde otro frente, acortar la brecha entre dispositivos de la frontera y redes neuronales profundas.

Referencias

1. Pepper, R.: Cisco visual networking index: Global mobile data traffic forecast update. Cisco, White Paper, (2017) https://goo.gl/2aVofM
2. WinterGreen Research: Internet of Things (IoT) Market Shares. Strategies, Forecasts, Worldwide, 2017 to 2023, Lexington, Massachusetts, Market Analysis Report (2017) https://goo.gl/bDZDB7
3. Grand View Research: Smart home automation market size. Industry Trend Report 2014-2025, Report USA, Aug (2017) https://goo.gl/FCRteF
4. Bilal, K.: Potentials, trends, and prospects in edge technologies: Fog, cloudlet, mobile edge and micro data centers. Computer Networks, vol. 130, pp. 94–120 (2018) doi: 10.10 16/j.comnet.2017.10.002
5. Satyanarayanan, M.: The emergence of edge computing. Computer, vol. 50, no. 1, pp. 30–39 (2017) doi:10.1109/MC.2017.9
6. Iorga, A.: Fog computing conceptual model. National Institute of Standards and technology. NIST Special Publication U.S. Dept. of Commerce, pp. 500–325 (2018) doi:10.6028/NIST.SP.500-325
7. Mohammadi, M., Al-Fuqaha, A.: Enabling cognitive smart cities using big data and machine learning: Approaches and challenges. IEEE Communications Magazine, vol. 56, no. 2, pp. 94–101 (2018) doi:10.1109/MCOM.2018.1700298
8. Li, H., Ota K., Dong, M.: Learning IoT in edge: Deep learning for the internet of things with edge computing. IEEE Network, vol. 32, no. 1, pp. 96–101 (2018) doi:10.11 09/MNET.2018.1700202
9. Li, Q., Xiao, Q., Liang, Y.: Enabling high performance deep learning networks on embedded systems. In: Proceedings of IECON 2017 - 43rd Annual Conference of the IEEE Industrial Electronics Society, pp. 8405–8410 (2017) doi:10.1109/IECON.2017.8217476
10. Verhelst, M., Moons, B.: Embedded DNN processing: Algorithmic and processor techniques bring deep learning to IoT and edge devices. IEEE Solid-State Circuits Magazine, vol. 9, no. 4, pp. 55–65 (2017)
11. Gershgorn, D.: Forget the space race. The Artificial Intelligence Race is Just Beginning, (2018) http://shamov.info/node/652
12. Georgiev, P.: Low-resource multi-task audio sensing for mobile and embedded devices via shared deep neural network representations. In: Proceedings of ACM Interact. Mob. Wearable Ubiquitous Tech, vol. 1, no. 50, pp. 1–19 (2017) doi: 10.1145/3131895
13. Lane, N. D., Bhattacharya, S., Mathur, A., Georgiev, P., Forlivesi, C., Kawsar, F.: Squeezing deep learning into mobile and embedded devices. IEEE Pervasive Computing, vol. 16, no. 3, pp. 82-88 (2017) doi:10.1109/MPRV.2017.2940968
14. Han, S.: Putting AI on a diet: TinyML and efficient deep learning. TinyML Summit Keynote (2021) doi: 10.1109/VLSI-DAT52063.2021.9427348
15. Pacheco, A., Flores, E., Sánchez, R., Almanza, S.: Smart classrooms aided by deep neural networks inference on mobile devices. In: Proceedings of IEEE International Conference on Electro/Information Technology (EIT), pp. 605–609 (2018) doi:10.1109/EIT.20 18.8500260
16. Pacheco, A., Cano, P., Flores, E., Trujillo E., Marquez, P.: A smart classroom based on deep learning and osmotic IoT computing. In: Proceedings of Congreso Internacional Innovación y Tendencias en Ingeniería, pp. 1–5 (2018) doi:10.1109/CONIITI. 2018.8587095

17. EEMBC. Benchmarks, Embedded Microprocessor Benchmark Consortium (2021) https://www.eembc.org/products/
18. Tang, S.: The AI chips list: A list of ICs and IPs for AI, ML and DL. Github repository, (2021) https://basicmi.github.io/AI-Chip/
19. Allan, A.: Benchmarking edge computing: Comparing Google, Intel and NVIDIA accelerator hardware. Medium blog, (2019) https://aallan.medium.com/benchmarking-edge-computing-ce3f13942245
20. TinyML Foundation. In: TinyML Summit conference and research symposium, online event, California, USA (2021) https://www.tinyml.org/event/summit-2021
21. ACM Search query for machine learning inference research publications from 2010 to 2021. ACM Digital Library, http://tiny.cc/comia1
22. IEEE Search query for machine learning inference research publications from 2010 to 2021. IEEEXplore Digital Library, https://aka.my/i8y
23. Li, H., Ota, K., Dong, M.: Learning IoT in edge: Deep learning for the internet of things with edge computing. IEEE Network, vol. 32, no. 1, pp. 96–101 (2018) doi:10.1109/MNET.2018.1700202
24. Jacob, B.: Quantization and training of neural networks for efficient integer-arithmetic-only inference. In: IEEE/CVF Conference on Computer Vision and Pattern Recognition, USA, pp. 2704–2713 (2018) doi:10.1109/CVPR.2018.00286
25. Wang, S., Tuor, T., Salonidis, T., Leung, K. K., Makaya, C., He, T., Chan, K.: Adaptive federated learning in resource constrained edge computing systems. IEEE J. Sel. Areas in Communications, vol. 37, no. 6, pp. 1205–1221 (2019) doi:10.1109/JSAC.2019.2904348
26. Zhang, C.: Optimizing FPGA-based accelerator design for deep convolutional neural networks. In: Proceedings of ACM/SIGDA International Symposium Field-Programmable Gate Arrays. ACM, NY, USA, pp. 161–170 (2015) doi:10.1145/2684746.2689060
27. Ma, Y.: Optimizing loop operation and dataflow en FPGA acceleration of deep CNNs, In: Proceedings of 2017 ACM/SIGDA International Symposium Field-Programmable Gate Arrays, NY, USA, pp. 45–54 (2017) doi:10.1145/3020078.3021736
28. Shen, Y., Ferdman, Y., Milder, P.: Maximizing CNN accelerator efficiency through resource partitioning. SIGARCH Computer Architectures News, vol. 45, no. 2, pp. 535–547 (2017) doi:10.1145/3140659.3080221
29. Sanchez-Iborra, R., Skarmeta, A. F.: TinyML-enabled frugal smart objects: Challenges and opportunities. IEEE Circuits and Systems Magazine, vol. 20, no. 3, pp. 4–18 (2020) doi:10.1109/MCAS.2020.3005467
30. Dai, X., Spasić, I., Chapman, S., Meyer, B.: The state of the art in implementing machine learning for mobile apps: A survey. SoutheastCon, pp. 1–8 (2020) doi:10.1109/SoutheastCon44009.2020.924965
31. Xie, Y.: A brief guide of xPU for AI accelerators. ACM SIGARCH (2018) https://www.sigarch.org/?p=15093
32. Nikouei, S.: Real-time human detection as an edge service enabled by a lightweight CNN. In: IEEE International Conference on Edge Computing, pp. 125–129 (2018) doi10.1109/EDGE.2018.00025