elektron ámbar

Original Article

Revista elektron, Vol. 6, No. 2, pp. 107-114 (2022)
https://doi.org/10.37537/rev.elektron.6.2.169.2022

B-VGG16: Red Neuronal de Convolución cuantizada binariamente para la clasificación de imágenes

B-VGG16: Binary Quantized Convolutional Neuronal Network for image classification

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: 31/10/22; Aceptado: 07/12/22

***Resumen*—En este trabajo se entrena y evalúa una red neuronal de convolución cuantizada de forma binaria para la clasificación de imágenes. Las redes neuronales binarizadas reducen la cantidad de memoria, y es posible implementarlas con menor hardware que las redes que utilizan variables de valor real (Floating Point 32 bits). Este tipo de redes se pueden implementar en sistemas embebidos, como FPGA. Se realizó una cuantización consciente del entrenamiento, de modo de poder compensar los errores provocados por la pérdida de precisión de los parámetros. El modelo obtuvo una precisión de evaluación de un 88% con el conjunto de evaluación de CIFAR-10.**

Palabras clave: Redes Neuronales de Convolución; Clasificación; Cuantización.

***Abstract*— In this work, a Binary Quantized Convolution neural network for image classification is trained and evaluated. Binarized neural networks reduce the amount of memory, and it is possible to implement them with less hardware than those that use real value variables (Floating Point 32 bits). This type of network can be implemented in embedded systems, such as FPGA. A quantization-aware training was performed, to compensate for the errors caused by the loss of precision of the parameters. The model obtained an evaluation accuracy of 88% with the CIFAR-10 evaluation set.**

Keywords: Convolution Neural Network; Classification; Quantization.

I. Introducción

Las Redes Neuronales de Convolución (CNN - Convolutional Neural Network) se utilizan en la actualidad en diversas aplicaciones de reconocimiento de imágenes, ya sea, en la detección de defectos en la industria [1], en la clasificación de obstáculos en la navegación autónoma [2], o en el reconocimiento de objetos en la robótica [3].

Con el fin de clasificar imágenes, las CNN extraen características de las mismas a partir de filtros o kernels [4]. Dichos filtros contienen pesos y sesgos que deben ser entrenados, de modo de obtener características de bajo nivel, como puntos o líneas, y a partir de la sucesión de estos filtros, obtener características de alto nivel, como formas o texturas.

En el aprendizaje profundo (DL - Deep Learning) se utiliza una gran cantidad de capas con este tipo de filtros, a fin de poder reconocer figuras y formas. Dichos filtros se almacenan en variables de valor real, generalmente de punto flotante de 32 bits (FP32 - Floating Point 32 bits). Las arquitecturas actuales de aprendizaje profundo para reconocimiento de imágenes tienen millones de parámetros, por ejemplo, la red VGG16 (Visual Geometry Group 16) [5] que posee 16 capas, tiene más de 138 millones de parámetros, y se necesitan de 552 MB de memoria para almacenarlos, y para realizar la clasificación de una sola imagen se necesitan de 30.8 GFLOPS (FLOPS - Floating Point Operation).

La gran cantidad de memoria necesaria para almacenar los parámetros de estas redes, y la cantidad de operaciones necesarias para realizar una clasificación, hacen que implementar dichos algoritmos en sistemas embebidos sea una tarea compleja, especialmente si se necesitan respuestas en tiempo real. Es por ello, que diversos trabajos de investigación abordan con diferentes enfoques la implementación de estas redes en sistemas embebidos [6].

En la actualidad, existen trabajos que se enfocan en la cuantización de parámetros, este concepto se basa en representar los parámetros con variables de menor precisión, como por ejemplo enteros de 8 bits (INT8 - Integer 8 bits) [7]. Dentro de la rama de redes neuronales de convolución cuantizadas, Courbariux et. al [8] demostró que es posible cuantizarlas con variables binarias, de modo de reducir la cantidad de memoria necesaria, y a su vez, inferirlas de forma eficiente a partir de operaciones binarias en sistemas embebidos como FPGA (Field Programmable Gate Array).

En el presente trabajo se entrena y evalúa una red neuronal de convolución binaria, basada en VGG16, cuantizada de

forma directa a partir de la librería Brevitas [9], lo cual permite exportar la red a diferentes aceleradores de inferencia como por ejemplo, el framework FINN [10].

Así mismo, se comparan de los resultados obtenidos con los métodos de BNN de última generación con el conjunto de datos CIFAR10 [11]. Y, se discute acerca de qué método es más apto para implementar en un dispositivo FPGA. Además, se dispone de un bloc de notas en Google Colab, a partir del cual se pueden realizar evaluaciones del modelo. El mismo se puede acceder desde la referencia [12].

El documento se encuentra organizado de la siguiente manera, en la sección 2 se encuentran los antecedentes de las redes neuronales binarizadas. En la sección 3 se da un panorama de los trabajos de cuantización binaria relacionados. En la sección 4 se detalla la red propuesta. En la sección 5 se informa sobre los procedimientos para la evaluación del modelo. En la sección 6 se presentan los resultados. Y por último, en la sección 7 y 8 se esbozan las conclusiones y se detalla el trabajo a futuro.

II. Antecedentes

A. Cuantización de Redes Neuronales de Convolución

El enfoque de los investigadores actuales se centra en implementar redes neuronales con una cuantización de 8 bits enteros INT8 o 4 bits enteros INT4. Como es el caso de los desarrolladores de Xilinx y Nvidia, quienes ofrecen herramientas de software para implementar una cuantización de enteros de 8 bits, los framework TensorRT [13] y VITIS AI [14] respectivamente, lo cual reduce considerablemente los requerimientos de memoria y el costo computacional. TensorRT es un framework diseñado para ser utilizado en CPU o en placas de desarrollo basadas en GPU como Jetson Nano. VITIS AI ha logrado grandes avances en la implementación de CNN en FPGA, abordan la cuantización INT8 e INT4, pero se necesitan de dispositivos FPGA tipo Zynq UltraScale+, ya que es necesario implementar una DPU (Deep Learning Processing Unit), como es el caso de las placas de desarrollo Kria de Xilinx. Los tipos de cuantización más utilizados en este campo son: cuantificación posterior al entrenamiento y entrenamiento consciente de cuantificación.

1) Cuantización posterior al entrenamiento: La cuantización posterior al entrenamiento (PTQ— Post Training Quantization), es una técnica en donde la red neuronal es entrenada utilizando variables de valor real (FP32 o FP16), y luego es cuantizada. Para realizarlo, se cuantiza directamente los parámetros de la red, estos parámetros no pueden ser cambiados, y no se puede realizar un nuevo entrenamiento, solo inferir. El inconveniente de este tipo de cuantización, es que al cambiar los parámetros por los nuevos cuantizados, aparece un error en la precisión de cada una de las capas de la red que no son compensados.

2) Cuantización consciente del entrenamiento: La cuantización consciente del entrenamiento QAT (Quantized Aware Training) compensa los errores de cuantización durante el mismo entrenamiento. La librería Pytorch [15] ofrece herramientas para implementar cuantizaciones tipo QAT, pero están destinadas a ser utilizadas en CPU Específicos (ARM y x64).

En los últimos años, se han desarrollado diferentes herramientas para llevar a cabo redes QAT [16]. El proyecto de investigación Brevitas [9], es una plataforma que permite implementar dichas redes, y es compatible con Pytorch. Ofrece un conjunto de bloques de diferentes niveles de abstracción para reducir la precisión del modelo en hardware en el entrenamiento. Actualmente, solo ofrece cuantización uniforme.

B. Cuantización Binaria

Si bien la mayoría de las investigaciones sobre CNN se basan en punto flotante, y las cuantizaciones en INT8 e INT4, hay investigaciones que han implementado redes con 1 bit o 2 bits de activación, logrando una alta precisión. Como es el caso de BinaryNet desarrollada por Courbariaux et al. [8], este tipo de redes se denominan BNN (Binary Neural Network).

Se consideran 3 aspectos en la binarización, la binarización de activaciones de entrada, de pesos de las capas, y de las salidas. En el caso de que tanto las entradas, los pesos y las salidas sean binarias, se denomina binarización total, cuando hay 1 o 2 componentes binarizados, se denomina binarización parcial.

En las redes BNN, solo los valores binarizados de los pesos y activaciones se utilizan en todos los cálculos. Como la salida de una capa es la entrada de la siguiente, todas las entradas de las capas son binarias, a excepción de la primera capa. Esto no resulta un inconveniente, ya que en el campo de la visión artificial, la representación de la entrada suele tener muchos menos canales, por ejemplo RGB tiene 3 canales, que las representaciones internas, por ejemplo 512, es por ello que la primera capa de una red de convolución suele ser la más pequeña, tanto en parámetros como en cantidad de cálculos.

Los autores de BinaryNet realizan una BNN, donde los pesos y las activaciones se restringen a -1 y $+1$, el objetivo primordial, es que sean simples de implementar en hardware con compuertas tipo XNOR. Utilizan una binarización determinística basada en la función $sign$:

$$x^b = sign(x) = \begin{cases} +1 & \text{if } \ x \geq 0 \\ -1 & \text{if } \ x < 0 \end{cases} \tag{1}$$

Siendo x el peso o la activación de valor real y x^b binarizado. La cuantización se realiza con la siguiente ecuación:

$$q = sign(r) \tag{2}$$

Siendo r los parámetros de la capa anterior. Se obtiene el estimador g_q del gradiente de la función de pérdidas con la estimación directa (STE- Straight-through gradient estimation) [17]. Tal cual se describre en la siguiente ecuación:

$$g_r = g_q 1_{|r|\leq 1} \tag{3}$$

Courbariaux obtiene resultados competitivos respecto a los modelos de precisión completa en CNN y redes totalmente conectadas y con capas de normalización de lote en conjuntos de datos como MNIST, SVHN y CIFAR10.

C. Aceleradores de BNN en hardware

En lo que respecta a la implementación en Hardware, estas redes presentan muchas ventajas frente a las CNN tradicionales, ya que se pueden inferir con menos recursos computacionales, son de bajo consumo energético, y a su vez necesitan de menor cantidad de memoria. Es por ello, que se busca ejecutarlas en hardwares limitados computacionalmente, como pueden ser los ASIC, las FPGA o las CPU. Si bien un diseño de un dispositivo ASIC, presenta ventajas a la hora de la eficiencia energética, las FPGA son los dispositivos más versátiles, es por ello que diversos autores se centran en la implementación en este tipo de dispositivos.

Los autores de XNOR-NET [18], han implementado su modelo en una Raspberry PI zero obteniendo resultados favorables. A su vez, los desarroladores de JDAI, han creado una librería denominada daBNN [19] con el fin de realizar inferencias en dispositivos móviles con procesadores ARM.

Desde la empresa Xilinx han desarrollado FINN [10] un framework capaz de implementar BNN en sus FPGA a partir de una arquitectura de transmisión de múltiple, utilizando conteo de bits, límites y operadores tipo OR para mapear dichas redes. Logrando inferir el modelo BNN-PYNQ [20] basado en CNV, en un chip Zynq 7020.

III. Trabajos Relacionados

En los últimos años, se han propuesto diversas redes neuronales binarizadas, basadas en diferentes técnicas. A grandes rasgos se las puede clasificar cómo, redes neuronales binarias directas, y redes neuronales basadas en la optimización. Dentro de esta última categoría se distinguen los métodos que minimizan el error de cuantización, aquellos que mejoran la función de pérdida y los métodos que buscan reducir el error de gradiente.

Las redes neuronales binarizadas directas cuantizan los pesos y las activaciones a un 1 bit, utilizando una función de binarización, luego realizan la propagación hacia atrás, optimizando el modelo de la forma clásica. BinaryConnect [21] convierte los pesos FP32 dentro de la red en pesos de 1 bit. En la propagación hacia adelante, un método de binarización estocástico es utilizado para cuantizar los pesos. Luego, en la propagación hacia atrás, una función de recorte es utilizada para limitar el rango de actualización de los pesos de FP32, logrando así que los pesos de valor real no crezcan demasiado sin un impacto en los pesos binarios.

Los mismos autores implementaron la red BNN [8] demostrando que estas redes necesitan 32 veces menos memoria que una red de precisión completa, y pueden inferir en un tiempo 60% menor , a partir de una normalización por lote basada en turnos y contadores de bit tipo XNOR.

El equipo de Xilinx ha desarrollado el repositorio BNN-PYNQ, que es parte del proyecto FINN [10], está basado en BinaryNet, en la cual utilizan un modelo inspirado en VGG11, al que denominan CNV, y obtienen una precisión en el conjunto de datos CIFAR10 del 84.22% con una cuantización de 1 bit de pesos y 1 bit de activación. El modelo consta de 4 pilas de convolución, activación y Maxpooling, se utiliza la binarización binaria denominada bipolar, ya se restringe los valores a +1 y -1.

En el caso de la optimización a partir de la disminución del error de cuantización de los pesos y las activaciones, se emplea una solución similar a la de la binarización directa, pero se introduce un factor de escala para el parámetro binario. De forma tal que, en vez de binarizar entre -1 y $+1$ se binariza entre $-\alpha$ y $+\alpha$, siendo el peso representado como $w \approx \alpha b_w$. Luego se minimiza el error de cuantización para encontrar el factor de escala óptimo y los parámetros binarios.

Este método disminuye el error de cuantización, respecto a usar -1 y $+1$, aumentando la precisión de inferencia de la red. Dentro de estos métodos se pueden considerar XNOR-NET [18] y BWN [18], siendo la primera una red totalmente binarizada y la segunda, una red donde solamente se binarizan los pesos.

Otro enfoque se basa en minimizar el error de cuantización, encontrando una función de pérdidas óptima, que controle el entrenamiento de los parámetros, teniendo en cuenta las restricciones de la binarización. Los métodos descriptos en los párrafos anteriores, solo se centran en mejorar la precisión local, logrando que los parámetros sean los más cercanos posibles a los valores de FP32. En cambio, la red LAB [22], busca minimizar la pérdida total asociada a los pesos binarios utilizando el algoritmo cuasi-Newton. Para ello, emplean la información de la media móvil de segundo orden que es calculada por el optimizador ADAM [23], y de este modo encontrar los pesos óptimos teniendo en cuenta las características de la binarización.

Por otro lado, el método Aprendiz [24] entrena a una red de baja precisión llamada estudiante, usando una red de gran escala entrenada con una red Maestro de las mismas características. La BNN es supervisada por la red Maestro, pero preserva la capacidad de aprendizaje, y obtiene rendimientos cercanos a la red de precisión completa. Este concepto se denomina destilación.

La red Principal/Subsidiaria [25] demuestra que la función de pérdidas, relacionada con el modelo de precisión completa de la red Maestra, ayuda a estabilizar el entrenamiento de los modelos binarios Aprendiz con una gran precisión.

Otros enfoques actuales se basan en reducir el error del gradiente. Ya que, al utilizar el estimador directo STE (Straight-Through Estimator) [17] para los gradientes de la propagación hacia atrás, existe un desajuste entre el gradiente de la función de binarización. Como es el caso de la función $sign$ y el mismo STE que en general, suele ser una función tipo $clip$. Esto puede provocar que la red no sea optimizada correctamente, generando una baja considerable de la precisión.

Este inconveniente puede ser solucionado diseñando una función aproximada de binarización de modo de disminuir el desajuste de la propagación hacia atrás. La red DSQ [26] reemplaza la función de cuantización tradicional, con una función de cuantización de software, que ajusta el valor de corte y la forma de la función de cuantización para acercarse de forma gradual a la función $sign$. La red DSQ rectifica la distribución de datos de forma orientable, logrando una disminución en el desajuste del gradiente y obteniendo una precisión inclusive superior que la del modelo con precisión completa.

IV. Experimentación

Para llevar a cabo este trabajo se empleó la librería Brevitas, la cual ofrece un conjunto de bloques de construcción con diferentes niveles de abstracción con el objetivo de realizar cuantizaciones en modelos de redes neuronales. A su vez, se destaca que Brevitas ofrece la exportación a diferentes plataformas de inferencia de modelos y aceleración de hardware, como FINN [10], onnxruntime [27], TVM [28] y PyXIR [29].

A. Conjunto de datos

Se utilizó el conjunto de datos CIFAR-10 [11] que consiste de 60 mil imágenes color (RGB) de 32x32 píxeles, con 6 mil imágenes por cada clase. Hay 50 mil imágenes para entrenamiento y 10 mil para evaluación. Las clases se encuentran etiquetadas de acuerdo a la Figura 1.

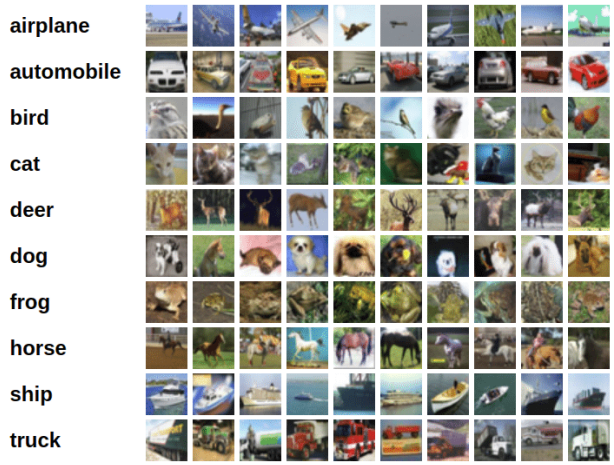

Fig. 1. Clases de conjunto de datos CIFAR-10

B. Arquitectura

En este trabajo se utiliza como modelo a la variación de la red VGG16 propuesta por Lui et. al [30]. La cual consiste de pilas de capas formadas por convolución, activación, y pooling. La primera pila de capas consta de 2 convoluciones de 64 filtros cada una, y un Maxpooling, que reduce el tamaño de la matriz a la mitad, utilizando un kernel de 2x2, y seleccionando el mayor elemento de esa porción de la matriz.

La segunda, consta de 2 convoluciones de 128 filtros cada una con un Maxpooling a la salida. La tercera de 3 convoluciones de 256 filtros con Maxpooling. Y la cuarta y la quinta constan de 3 convoluciones de 512 filtros cada una, en este caso sin Maxpooling. Ya que las dimensiones de las matrices no son divisibles por 2.

Por último, se utilizan dos redes neuronales totalmente conectadas para relacionar los 512 elementos con 1024, y la segunda los 1024 elementos con las 10 clases de salida de la red, por último se utiliza la función Softmax. Se puede observar la arquitectura completa en la Figura 2.

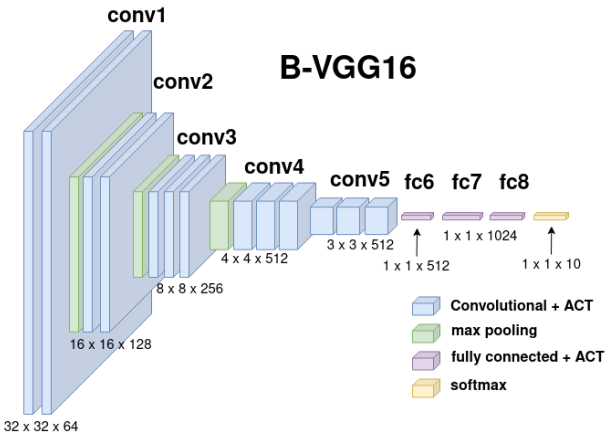

Fig. 2. Arquitectura de la red BVGG16 propuesta

C. Función de pérdidas

La función de pérdida o de coste cuantifica el error del modelo, a través del cálculo de la predicción de la red contra el valor verdadero. Durante el entrenamiento, el optimizador ajusta el modelo para minimizar las pérdidas.

En este trabajo se utiliza una función de entropía cruzada categórica, ya que se trata de una clasificación multi-clase. La misma cuantifica la diferencia entre dos distribuciones de probabilidad, la predicha por el modelo y la real. Se computa a partir de la ecuación 4.

$$Loss = -\sum_{i=1}^{m} y_i \cdot \log \hat{y}_i \tag{4}$$

Donde $\hat{y}_i$ es el i valor escalar de salida del modelo, y_i corresponde al valor real. Cuando se utiliza para distinguir a 2 distribuciones discretas de probabilidad, como el caso de este experimento, y_i representa la probabilidad de la clase i, y la suma de todos los y_i es igual 1. El signo negativo asegura que la probabilidad se vuelva menor cuando las dos distribuciones de probabilidad son cercanas entre sí.

D. Regularización

Al entrenar redes neuronales profundas se corre el riesgo de caer en el sobre ajuste. Por ese motivo existen diversas técnicas para evitarlo, como Dropout [31], Kernel regularizer y BatchNormalization [32].

En este caso, se utiliza BatchNormalization, quién normaliza la activación de la capa anterior de cada lote, aplicando una transformación que mantiene el promedio de activación cercano a 0 y la desviación estándar cercana a 1. Esto direcciona el problema del desplazamiento de la covarianza interna. También actúa como regularizador, en algunos casos eliminando la necesidad de utilizar Dropout. Logra la misma precisión con menores pasos de entrenamiento, por lo tanto, acelera el proceso de entrenamiento.

E. Optimizador

El optimizador se encarga de generar pesos que ajusten de mejor manera el modelo. Calcula el gradiente de la función de pérdidas por cada peso, a través de la derivada parcial. El objetivo es minimizar el error, por lo tanto, los pesos se

modifican en dirección negativa del gradiente.

$$W_t + 1 = W_t - \frac{df(coste)}{dW} * lr \tag{5}$$

El descenso de gradiente estocástico mantiene una tasa de aprendizaje única para actualizar todos los pesos, y la tasa no cambia durante el proceso de entrenamiento. En este trabajo se utiliza el algoritmo de Adam (Adaptive moment estimation) [23], el cual es una combinación de AdaGrad (Algoritmo de gradiente adaptativo) y RMSprop (Propagación cuadrática media). Se tiene un factor de entrenamiento por parámetro. Cada factor de entrenamiento se ve afectado por la media del momentum del gradiente.

F. Cuantización

De acuerdo a lo propuesto por BinaryNet [21], todas las capas utilizan valores de 1 bit para las activaciones de entrada, los pesos y las activaciones de salida, lo que se denomina binarización total, y se representa de forma bipolar con -1 y +1.

La capas de convolución se implementan con la función QuantConv2d de la librería Brevitas [9]. La misma es construida con Conv2d de PyTorch y se instancia a través de una clase de cuantización denomina QuantWBIOL (QuantWeightBiasInputOutputLayer) la cual recibe la entrada, el sesgo y los pesos de la capa de convolución y retorna su versión cuantizada con los parámetros seleccionados. Para instanciar QuantConv2D es necesario definir las dimensiones y los canales de la entrada, como así también, la dimensión del filtro y otros parámetros como el padding y el striding. Además, es necesario definir el tipo de cuantizador para los pesos, el ancho de bits, el rango y si la cuantización es signada.

1) Cuantizador: Brevitas provee varios tipos de cuantizadores, en este trabajo se utilizó QuantType.BINARY, el cual se implementa con el módulo BinaryQuant(). Este módulo entrega la salida cuantizada a un bit, en el formato sin cuantizar, que consta de la escala, el punto cero, y el ancho de bits de la cuantización. Dado el siguiente tensor:

$$A = \begin{pmatrix} 0.0250 & 0.4700 & 2.0575 \\ 0.4244 & -0.7854 & -0.5433 \\ 0.2290 & 0.6359 & 1.0642 \end{pmatrix} \tag{6}$$

En primer lugar, se obtiene el máximo valor, y a partir de este valor, se calcula la escala del siguiente modo:

$$escala = \frac{max|A_{ij}|}{1} \tag{7}$$

Al ser una cuantización binaría, el máximo valor que puede tomar es +1.

En segundo lugar, se calcula el nuevo tensor a partir de la escala y se redondea al valor entero más cercano -1 o +1, ya que el cero se encuentra restringido.

$$A_{quant} = \frac{A}{escala} \approx \begin{pmatrix} +1 & +1 & +1 \\ +1 & -1 & -1. \\ +1 & +1 & -1. \end{pmatrix} \tag{8}$$

2) Capas Cuantizadas: A continuación se detalla como se instanció la función QuantCon2d en la primera capa de convolución de la red planteada en el trabajo.

```
QuantConv2d(kernel_size=3,
            padding=1,
            in_channels=3,
            out_channels=64,
            bias=False,
            weight_quant=CommonWeightQ,
            weight_bit_width=1,
            return_quant_tensor=True)
```

Además, se utilizaron para las capas de pooling la función QuantMaxPool2d, y para las activaciones la función QuantIdentity.

G. Activación

En los experimentos realizados, se utilizó la función identidad como activación. Vale la pena aclarar, que al ser una activación cuantizada, aunque la función sea identidad, la cuantización, que en este caso eso es la función $sign$, hace las veces de función transferencia. En la Fig. 3 se observa la función $sign$.

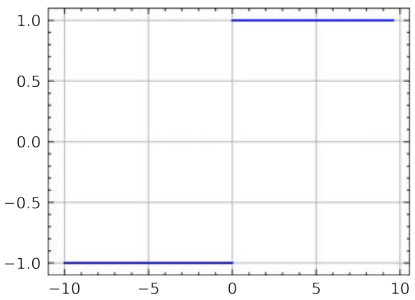

Fig. 3. Función $sign$ utilizada para la cuantización

H. Parámetros e hiperparámetros de entrenamiento

Se configuró la red con los parámetros e hiperparámetros que se observan en la tabla I.

TABLA I
PARÁMETROS E HIPERPARÁMETROS DE ENTRENAMIENTO

Arquitectura	VGG16
Conjunto de Datos	CIFAR-10
Épocas	1000
Tasa de Aprendizaje	0.02
Tamaño de lote	100
Función de pérdidas	Cross Entropy
Función de Activación	Identidad(Cuantizada)
Optimizador	ADAM
Regularizador	BatchNormalization
Bits de Pesos	1 bit
Bits de activaciones	1 bit

V. EVALUACIÓN

A. Entrenamiento

Se entrenó el modelo cuantizado sobre el conjunto de datos CIFAR-10 con una tasa de aprendizaje de 0.02. De las 50 mil imágenes del conjunto de entrenamiento, se separaron 5 mil de forma aleatoria para utilizarlas

como validación. Se utilizó un tamaño de batch de 100, y a las imágenes del conjunto de datos se les aplicó la transformación a tensor de PyTorch, y una función de volteo aleatorio y recorte aleatorio para aumentar los datos. Se puede visualizar un ejemplo del lote en la Fig. 4.

Fig. 4. Ejemplo de batch de 100 imágenes

Se realizó el entrenamiento en una computadora de escritorio con procesador I7 11700k, 16 GB DDR4-2666 MHz de Memoria, y una placa GPU NVIDIA GeForce RTX-3060. Se entrenaron 1000 épocas en casi 5 horas. Vale la pena aclarar, que si bien se obtienen pesos y activaciones binarios, para realizar el entrenamiento se utilizan valores reales que luego son cuantizados en cada capa.

VI. Resultados

Con el modelo propuesto B-VGG16 se obtiene una precisión con el conjunto de datos de evaluación de CIFAR10 de un 88.21%. La misma es calculada como el cociente de las predicciones correctas con el número total de predicciones.

En la Figura 5 se observa la matriz de confusión, la cual representa en el eje x las clases reales, y en el eje y los valores predichos. La diagonal principal representa a las clases que han sido correctamente predichas por el algoritmo.

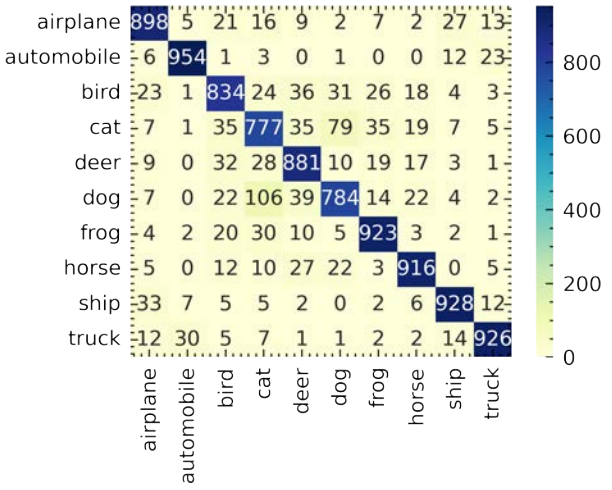

Fig. 5. Matriz de confusión del conjunto de evaluación

En la izquierda de la Figura 6 se observa una imagen de 32x32 píxeles RGB obtenida del conjunto de evaluación de CIFAR10. Luego de ser inferida por el modelo en análisis, se observan las probabilidades predichas para cada clase a la derecha de la Figura 6. La clase auto es la que tiene mayor probabilidad.

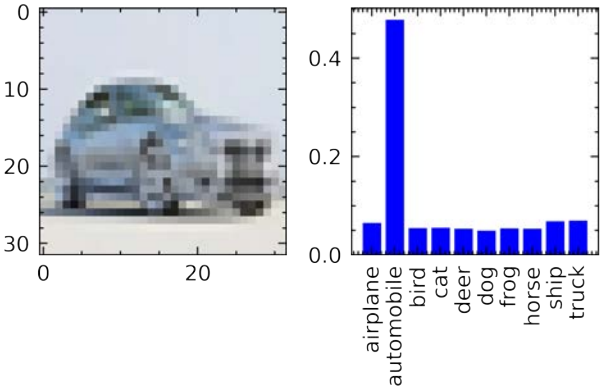

Fig. 6. Imagen del conjunto de datos CIFAR10

Además, de la evaluación con imágenes pertenecientes al conjunto de datos se evaluó con fotografías externas, como es el caso de la imagen de la izquierda de la Figura 7, donde en primer lugar se recortó la imagen para centrar al vehículo, y luego se la redujo a 32x32 píxeles. Se pueden observar las predicciones obtenidas en el gráfico de la derecha de la Figura 7.

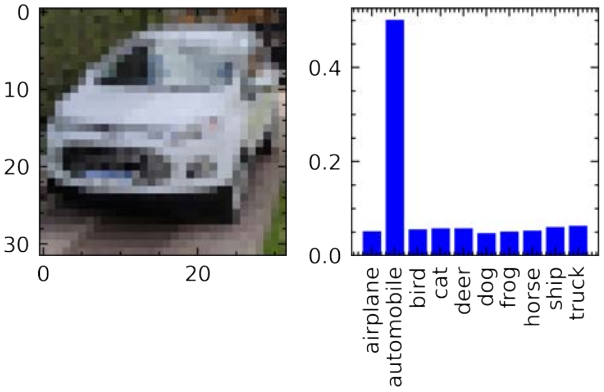

Fig. 7. Imagen reducida a 32x32 píxeles

A. *Comparación con otras redes BNN*

En la tabla II se observa la comparación de la precisión de evaluación de diferentes redes de convolución binarizadas que han surgido en los últimos años. Se hace mención al método utilizado, la arquitectura de la red, la cantidad de bits de los pesos y las activaciones. Todos los métodos están evaluados con el conjunto de datos CIFAR-10.

Para esta tabla fueron seleccionados métodos que realizan cuantización totalmente binaria, de 1 bit tanto para los pesos como las activaciones, como es el caso de las redes BNN-PYNQ [20], XNOR-NET [18], LAB [22], Main/Subsidiary [25] y DSQ [26].

Si bien la red propuesta B-VGG16 no supera a la red de última generación DSQ, la cual reduce el error del gradiente, ni a XNOR-NET que minimiza el error de cuantización. Sí presenta mejoras en aquellas redes que utilizan el enfoque de mejorar la función de pérdidas, como es el caso de la red LAB y Main/Subsidiary.

A su vez, la red obtiene un rendimiento más alto que BNN-PYNQ, esto se debe principalmente a que la red pro-

TABLA II
COMPARACIÓN CON OTRAS REDES BNN EVALUADAS CON EL CONJUNTO DE DATOS CIFAR10

Red	Enfoque	Arquitectura	Bits(W/A)	Acc.[%]
VGG16 [30]	Precisión completa	VGG-16	32/32	90,2
B-VGG16(propia)	Directo	B-VGG16	1/1	88,21
BNN-PYNQ [20]	Directo	CNV	1/1	84,22
XNOR-NET [18]	Minimizar el Error de cuantización	VGG-SMALL	1/1	89,8
LAB [22]	Mejorar la Función de pérdidas	VGG-SMALL	1/1	87,72
Main/Subsidiary [25]	Mejorar la Función de pérdidas	VGG-11	1/1	82
DSQ [26]	Reducir el error de Gradiente	VGG-SMALL	1/1	91,7

puesta utiliza más capas de convolución que la arquitectura CNV.

Vale la pena resaltar que la precisión no es el único criterio a evaluar de las BNN, ya que la versatilidad es de gran importancia para implementar estas redes en aplicaciones prácticas. Ya sea para reentrenar la red con otro conjunto de datos, o para llevarla a hardware.

Por otro lado, los métodos que buscan mejorar la precisión de las BNN a partir de cuantizadores dedicados, se basan en cálculos complejos, y en varias etapas, lo cual dificulta su implementación y reproducibilidad en hardware, en otras palabras, carecen de un sentido práctico. El equilibrio entre la precisión y la velocidad de inferencia es un criterio a tener en cuenta.

En resumen, el método propuesto por los autores B-VGG16 logra una precisión competitiva, superando incluso algunos métodos más complejos a nivel cálculo, pero sin llegar a los resultados obtenidos por los métodos de última generación como DSQ. Sin embargo, tiene la ventaja de ser simple de implementar, debido a que utiliza un método de cuantización directo. A su vez, la red se construye a partir de Brevitas y Pytorch, por lo que permite que sea evaluada en el bloc de notas puesto a disposición en la referencia [12]. También, es importarte resaltar que la librería Brevitas, permite exportar directamente al framework FINN, y esto permitiría en un futuro implementar la red en una FPGA. Por estos motivos, se considera que el modelo propuesto, tiene como ventaja principal su versatilidad, frente a los modelos comparados.

VII. CONCLUSIONES

En el presente trabajo se hace una discusión y análisis de diferentes métodos de cuantización binaria de redes neuronales de convolución y luego, sobre la base de esto, se implementa una red de convolución basada en la arquitectura VGG16 a partir de las herramientas de cuantización tipo QAT que ofrece la librería Brevitas. Se utiliza una cuantización binaria de 1 bit tanto para los pesos como para las activaciones. Se entrena y evalúa con el conjunto de datos CIFAR10. La librería Brevitas presenta la ventaja de ser de fácil utilización, y a su vez, está directamente asociada al framework FINN de inferencia a FPGA.

Si bien, la red no obtiene los resultados más altos, en cuanto a la comparación de la precisión con otros métodos de cuantización binaria de última generación, como DSQ. Si supera a la ya mencionada BBN-PYNQ, y a los métodos basados en mejorar la función de pérdidas.

Respecto a DSQ, el modelo propuesto por este trabajo, tiene la ventaja de ser más versátil, ya que es mucho más simple de implementar, debido principalmente, a que utiliza una cuantización directa y a su vez presenta la ventaja de que al ser codificado con la librería Brevitas se puede exportar directamente al framework FINN, y a partir de la arquitectura de transmisión múltiple con conteo de bits y compuertas OR, se puede lograr una alta velocidad de inferencia a un bajo costo energético.

El modelo propuesto obtiene una mayor precisión de evaluación respecto al modelo CNV [20], esto se debe a que se aumentan la cantidad de pilas de capas de convolución, activación y pooling. Al utilizar un modelo más profundo, con mayor cantidad de activaciones y parámetros, también aumenta la cantidad de memoria en juego. Esto podría implicar un problema a la hora de implementar la inferencia de la red en sistemas tipo FPGA, pero el modelo CNV está implementado para un chip de Xilinx ZYNQ 7020, que se encuentra dentro de la generación anterior de FPGA. Actualmente, la nueva generación de dispositivos de la marca Xilinx, como por ejemplo la Zynq Ultrascale+ es capaz de albergar dichos modelos.

VIII. TRABAJO A FUTURO

A partir del framework experimental FINN [10], se implementará esta red en una FPGA tipo SOC (System On Chip) del proyecto PYNQ. Para ello se deberá optimizar el modelo, a partir de técnicas de pruning, de modo de reducir la cantidad de parámetros y de operaciones.

AGRADECIMIENTOS

Los autores agradecen al “Instituto de Investigaciones Científicas y Técnicas para la Defensa”— CITEDEF dependiente del Ministerio de Defensa, por el apoyo dado; además, al Programa de Investigación y Desarrollo para la Defensa del Ministerio de Defensa por el financiamiento del proyecto PIDDEF 06/17. Se agradece a la Universidad Tecnológica Nacional, por el Programa de Investigación y Desarrollo por el proyecto MSTCAHA0008161TC. Y en particular a la UTN Facultad Regional Haedo, por el apoyo dado.

REFERENCIAS

[1] X. Chen, J. Chen, X. Han, C. Zhao, D. Zhang, K. Zhu, and Y. Su, “A light-weighted cnn model for wafer structural defect detection,” *IEEE Access*, vol. 8, pp. 24 006–24 018, 2020, doi: 10.1109/ACCESS.2020.2970461.

[2] H. Gao, B. Cheng, J. Wang, K. Li, J. Zhao, and D. Li, “Object classification using cnn-based fusion of vision and lidar in autonomous vehicle environment,” *IEEE Transactions on Industrial Informatics*, vol. 14, no. 9, pp. 4224–4231, 2018, doi: 10.1109/TII.2018.2822828.

[3] F. Foroughi, Z. Chen, and J. Wang, “A cnn-based system for mobile robot navigation in indoor environments via visual localization with a small dataset,” *World Electric Vehicle Journal*, vol. 12, no. 3, 2021, doi: 10.3390/wevj12030134. [Online]. Available: https://www.mdpi.com/2032-6653/12/3/134

[4] N. Aloysius and M. Geetha, "A review on deep convolutional neural networks," *Proceedings of the 2017 IEEE International Conference on Communication and Signal Processing, ICCSP 2017*, vol. 2018-Janua, pp. 588–592, 2018, doi: 10.1109/ICCSP.2017.8286426.

[5] 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

[6] 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, 2019, doi: 10.1109/ACCESS.2018.2890150.

[7] 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.

[8] 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

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

[10] 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.

[11] A. Krizhevsky, "Learning multiple layers of features from tiny images," 2009.

[12] N. Urbano Pintos, H. Lacomi, and M. Lavorato, "Bvgg16 - entrenamiento en google colab," 2022. [Online]. Available: https://colab.research.google.com/drive/1irvyEzHj7tAvIfV56bFHP50fCNDCHZsu?usp=sharing

[13] NVIDIA, "Tensorrt open source software," 2022. [Online]. Available: https://github.com/NVIDIA/TensorRT

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

[15] 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.

[16] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A Survey of Quantization Methods for Efficient Neural Network Inference," *Low-Power Computer Vision*, pp. 291–326, 2022, doi: 10.1201/9781003162810-13.

[17] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *arXiv preprint arXiv:1308.3432*, 2013.

[18] 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.

[19] J. Zhang, Y. Pan, T. Yao, H. Zhao, and T. Mei, "dabnn: A super fast inference framework for binary neural networks on arm devices," in *Proceedings of the 27th ACM international conference on multimedia*, 2019, pp. 2272–2275.

[20] 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.

[21] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," *Advances in neural information processing systems*, vol. 28, 2015.

[22] L. Hou, Q. Yao, and J. T. Kwok, "Loss-aware binarization of deep networks," *arXiv preprint arXiv:1611.01600*, 2016.

[23] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," 2014, doi: 10.48550/ARXIV.1412.6980. [Online]. Available: https://arxiv.org/abs/1412.6980

[24] A. Mishra and D. Marr, "Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy," *arXiv preprint arXiv:1711.05852*, 2017.

[25] Y. Xu, X. Dong, Y. Li, and H. Su, "A main/subsidiary network framework for simplifying binary neural networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 7154–7162.

[26] R. Gong, X. Liu, S. Jiang, T. Li, P. Hu, J. Lin, F. Yu, and J. Yan, "Differentiable soft quantization: Bridging full-precision and low-bit neural networks," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 4852–4861.

[27] O. R. developers, "ONNX Runtime," 11 2018. [Online]. Available: https://github.com/microsoft/onnxruntime

[28] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, H. Shen, M. Cowan, L. Wang, Y. Hu, L. Ceze *et al.*, "{TVM}: An automated {End-to-End} optimizing compiler for deep learning," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 2018, pp. 578–594.

[29] Xilinx, "PyXIR," 11 2019. [Online]. Available: https://github.com/Xilinx/pyxir

[30] 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.

[31] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[32] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," 2015, doi: 10.48550/ARXIV.1502.03167. [Online]. Available: https://arxiv.org/abs/1502.03167