IMPLEMENTATION OF A DINAMIC QUANTIZED CNN FOR DETECTION OF ATRIAL FIBRILLATION IN AN 8-BIT MICROCONTROLLER

IMPLEMENTACIÓN DE UNA CNN CUANTIZADA DINÁMICAMENTE PARA LA DETECCIÓN DE FIBRILACIÓN AURICULAR EN UN MICROCONTROLADOR DE 8 BITS

Laura C. Martinez Cruz, Mauricio Bautista Porras, Jeyson Castillo and Carlos A. Fajardo

Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones
Universidad Industrial de Santander
Bucaramanga, Santander, Colombia.

Abstract: Atrial fibrillation (AF) is a silent disease that is difficult to diagnose because its symptoms are sporadic. This disease has a high mortality rate in the world when it is diagnosed late. Currently, convolutional neural networks (CNN) are an important tool used for the diagnosis of diseases as AF, breast cancer, among others. However, CNNs are both computationally and memory intensive, making them difficult to deploy in devices with low computational resources. Quantized networks are a solution to reduce the amount of computing and memory resources. We aim to implement the inference process of a CNN into an 8-bit microcontroller (ATMEGA2560) by using quantization strategies. Several 8-bits quantization techniques were tested before implement the CNN into the microcontroller. The final implementation was done by a proposed heuristic method that we called Dynamic Layer Quantization. This method allows us to achieve an effective way to reduce the computational complexity of CNN and its memory requirements. Our results show an accuracy of 89.48% on the test data. This work is the first stage of a project that aims to build a portable device for the detection of AF.

Keywords: Atrial fibrillation, Heuristic method, Microcontroller, Convolutional Neural Network, Quantization.

Resumen: La fibrilación auricular (FA) es una enfermedad de difícil diagnóstico porque sus síntomas son esporádicos y tiene una alta tasa de mortalidad en el mundo cuando se diagnostica tarde. Actualmente, las redes neuronales convolucionales (CNN) son una herramienta importante utilizada para el diagnóstico de enfermedades como fibrilación auricular, cáncer de mama, entre otras. Sin embargo, las CNN tiene una alta demanda computacional y de memoria, lo que dificulta su implementación en dispositivos con bajos recursos computacionales como los microcontroladores de 8 bits. Un tema muy activo en la investigación son las redes neuronales cuantizadas ya que son una solución para reducir la cantidad de recursos informáticos y de memoria. En este trabajo se implementó el proceso de inferencia de una CNN en un microcontrolador de 8 bits (ATMEGA2560) mediante el uso de estrategias de cuantización. Se probaron varias técnicas de cuantización de 8 bits antes de implementar la CNN en el microcontrolador. La implementación final se realizó mediante un método heurístico que llamamos Cuantificación Dinámica por Capa. Este método nos permite lograr una forma efectiva de reducir la complejidad computacional de la CNN y sus requerimientos de memoria. Nuestros resultados muestran una precisión del 89,48 % en los datos de prueba. Este trabajo es la primera etapa de un macroproyecto que tiene como objetivo construir un dispositivo portátil para la detección de fibrilación auricular.

Palabras clave: Cuantización**.,** Fibrilación Auricular, Método Heurístico, Microcontrolador, Red neuronal Convolucional.

1. INTRODUCTION

Atrial fibrillation (AF) is a type of cardiac arrhythmia, characterized by very rapid and uncoordinated atrial activity. This disorder of the electrical signals of the heart has important clinical implications. AF patients have a high risk of stroke and thromboembolism. Furthermore, this disease can be paroxysmal and asymptomatic, making its early diagnosis difficult (Tse, H.-F., & Lane, D. A, 2016). Several studies have proposed the convolutional neural networks (CNN) for the detection of atrial fibrillation achieving high levels of accuracy (Yao, Z., Zhu, Z., & Chen, Y, 2017)(Xia, Y., Wulan, N., & Zhang, H, 2018)(Pourbabaee, B., Roshtkhari, M. J., & Khorasani, K, 2018). By achieving high performance, CNN-based methods demand high amount computation and memory resources. The demand of resources become challenging for the inference of CNN in integrated circuit applications as microcontrollers.

Quantization is an effective strategy to solve computing demand and memory resources. In the past, various CNN quantization methodologies have been studied to perform hardware implementation (Tang, C. Z., & Kwan, H. K, 1993). CNN quantization requires a set of methodologies to reduce the size of the architecture and the weights of the neural network (Lee, C., & Kang, I, 2018) (Seo, S., & Kim, J, 2019) (Athar, A, 2018). Quantization seeks to maintain the original accuracy as much as possible (Ruminski, J., & Rad, P, 2019). In this study, we aim to quantize the inference process of CNN Castillo-Granados (Castillo, J. A., Granados, Y. C., & Fajardo, C. A, 2020) and subsequently implement it in a microcontroller. We tested four different quantization methods. A heuristic method, that we called Dynamic Layer Quantization, was chosen because it provides the least loss of precision. With this method, we assign a different quantization factor to each layer. The Dynamic Layer Quantization method allow us to quantize CNN to 8-bit integers with an accuracy of of 89,48%. The implementation was carried out on the Atmega 2560 microcontroller using only 10% and 54% of the FLASH memory and SRAM, respectively.

Contributions:
Our main contribution in this work is the Dynamic Layer Quantization method, which offers a quantization scheme to implement DNNs on microcontrollers of 8 bits. Our results suggest that the proposed method offers better accuracy than the commercial tool TensorFlow Lite.

The rest of this paper is structured as follows: Section II summarizes the CNN model CastilloGranados (Castillo, J. A., Granados, Y. C., & Fajardo, C. A, 2020). Section III describes each of the quantization methods tested. Section IV describes the implementation in the Atmega 2560 microcontroller. In Section V we present the most important results. Finally, this paper is closed with the conclusions in Section VI.

2. CONVOLUTIONAL NEURAL NETWORKS

The artificial neural networks have an input layer, intermediate layers, and an output layer. The convolutional neural networks (CNN) have three types of layers between their intermediate layers. Those layers are the convolutional layers that extract information by filters, the grouping layers that reduce the size of the input, and the fully connected layers that classify information extracted from previous layers. In this work, we implemented the Castillo-Granados CNN (Castillo, J. A., Granados, Y. C., & Fajardo, C. A, 2020), which detects atrial fibrillation (AF) by ECG signals. The network was trained using the MIT-BIH database (Goldberger, A. L, 2000). The ECG signals were stored in vectors of 500 samples at 250 [samples / s]. In (Castillo, J. A., Granados, Y. C., & Fajardo, C. A, 2020) the authors achieve an accuracy of 97,44 % by using the traditional 64-bit double-precision floating-point format. The Castillo-Granados CNN is made up of 12 layers. An input layer, four convolutional layers with max-pooling layers and three fully connected layers. The last full connected layer has a single output data. The CNN has a total of 9.385 parameters. The Table 1 summarizes the characteristics of the layers.

3. QUANTIZATION STRATEGY

Four different techniques were tested to quantize the CNN to 8 bits before implementing it in the microcontroller. First, we used two techniques contained in the TensorFlow backend (Seo, S., & Kim, J, 2019), and then we used two heuristic methods based on Fake Quantization (Gupta, K., & Narayanan, P, 2015),(Nagel, Louizos, C., & Blankevoort, T, 2020).

Table 1: convolutional neural network architecture

NEURAL NETWORK ARCHITECTURE		
LAYER TYPE	OUTPUT DIMENSION	PARAMETERS
INPUT	(500x1)	0
CONVOLUTION	(474x3)	84
MAX-POOLING	(237x3)	0
CONVOLUTION	(224x10)	430
MAX-POOLING	(112X10)	0
CONVOLUTION	(110x10)	310
MAX-POOLING	(55x10)	0
CONVOLUTION	(52x10)	410
MAX-POOLING	(26x10)	0
FLATTEN	260	0
FULLY-CONECTED	30	7830
FULLY-CONECTED	10	310
FULLY-CONECTED	1	11

3.1 TensorFlow Quantization:

The TensorFlow library offers a framework for deep learning called TensorFlow Lite (Seo, S., & Kim, J, 2019). This tool converts the model into a special storage format (Flatbuffer) with reduced computational and memory resources. TensorFlow Lite has different optimization methods available to quantize the parameters that are not quantized in the conversion. We apply Quantization for Size and Integer Quantization. The Quantization for Size method optimizes the network depending on the weight of the operations of the model (Google, 2020). Once the Castillo-Granados CNN was quantized, we used the software Netron (Roeder, L, 2020) for the extraction and review of the quantized weights. This method converts all parameters to 32-bit floating-point, except for the first dense layer, that was converted to an 8-bit integer. Integer quantization takes all weights and network activation functions to 8-bit integers (Google, 2020). This method applied in the Castillo-Granados CNN converted all parameters of all layers to 8-bit integers. Table 2 summarizes the results regarding the accuracy by using Quantization for Size and Integer Quantization.

Table 2: accuracy of tensorflow methods

TENSORFLOW	
METHOD	ACCURACY
QUANTIZATION FOR SIZE	92,77 %
INTEGER QUANTIZATION	63,58 %

Source: Authors own creation

3.2 Heuristic Quantization

We tested two heuristic methods, which are an adaption of the Stochastic Rounding Method proposed in (Google, 2020). The first method is the static layer quantization, and the second method is the dynamic layer quantization. The Static Layer Quantization (SLQ) applies the same quantization factor (2^Q) to all layers in the model, according the following equation,

$$B * 2Q = A \quad (1)$$

Where B is the floating-point number to be quantized, Q is the number of bits after the point that will shift to the left. 2^Q is the quantization factor. Finally, A is a version of B shifted Q − bits to the left.

To find the appropriate value of Q, we generate the histograms of the inputs and the activations of the network layers for the MIT signals (See section 1). Using the histogram, we analyze the dynamic range of the input data and intermediate values generated through the network. Figure 1 is an example of the histograms in this analysis, in this case, is showing the distribution of the parameters in the FC1 layer. Note that the values are concentrated in the range of 0 to 6. Table 3 shows the dynamic range for each layer in the network. In this table, CONVi and FCi refer to convolutional and to full connected layers, respectively. The data was analyzed after going through the ReLu activation function (Agarap, A. F, 2018) except for the output layer. The output layer uses the Sigmoid activation function (Zhang, C., & Woodland, P. C, 2015) to generate a probability distribution. For this reason, we analyze the data in the output layer before the Sigmoid function. Then, we use the equation 2 to calculate the number of bits needed to represent the integer part.

Table 3: Layer output range on CNN

LAYER	LAYER OUTPUT RANGE	
CONV1	0	0,761
CONV2	0	1,103
CONV3	0	0,884
CONV4	0	2,196
FC1	0	7,691
FC2	0	11,357
FC3	-28,9	8,486

Source: Authors own creation

$$C = [log_2(|OL_{max}|)] \quad (2)$$

Where OL_{max} is the maximum value of the entire neural network, and C is the number of bits to represent the integer part of the number. The equation 3 is used to calculate the number of bits needed to represent the decimal part.

$$Q = N_D - C \quad (3)$$

Where ND is the number of bits to represent the data, in our case ND is 8-bit. And Q is the number of bits to represent the decimal part of the number. In Table 3, the absolute maximum value is found in the FC3 layer. Substituting this number in the equation 2 we obtain C = 5, then with the equation 3 we obtain Q = 3. We tested this quantization factor on the network and the results showed a significantly reduction on the accuracy. Using Q = 3, we take the entire range of available values, including those that are far from the highest concentration of data (see Figure 1). To improve the accuracy, we focus our analysis in the range to the highest data concentration intervals. Next, we apply equations 2 and 3 to calculate the value of the new Q. We repeat this process until we find the Q value for the maximum accuracy of the quantized CNN. Table 4 shows the results when we choose the highest data concentration to calculate quantization factor.

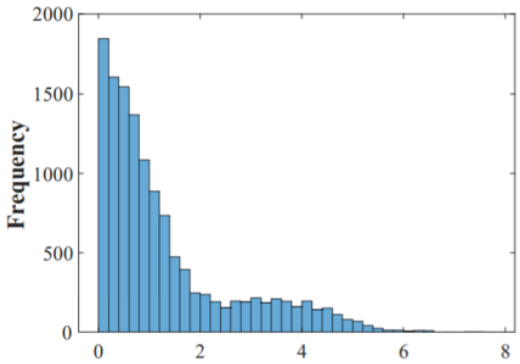

Fig. 1. Histogram of the layer's output data Fully Connected 1
Source: Authors own creation

On the other hand, the quantization process generates an unquantized decimal part. We use truncation and rounding methods to remove the unquantized decimal part (Sripad, A., & Snyder, D, 1977) (Schwarz, E. M, 1996). The rounding error is much less than the truncation error. In the Table 4, we show the results when applying the rounding method to eliminate the unquantized decimal part with all the Q found. Dynamic Layer Quantization (DLQ) is an improvement to the Static Layer Quantization technique. The difference is that we include the dynamic range analysis of the values of the weights per layer as shown in Table 5. We analyze each layer individually and apply a different quantization factor to each one. To find the appropriate Q value in each of the layers, we analyzed the dynamic range of the parameters and the outputs for all layers, as shown in Table 5. We use equation 4 to calculate the number of bits needed to represent the integer part.

Table 4: Neural Network accuracy for each value of Q

QUANTIZATION FACTOR Q	ROUNDING METHOD ACCURACY
Q = 3	49,532 %
Q = 4	54,281 %
Q = 5	73,510 %
Q = 6	70,337 %
Q = 7	65,921 %

Source: Authors own creation

$$C_i = [log_2(Max\{|OL_i|, |P_i|\})] \quad (4)$$

Where Ci is the number of bits to represent the integer part of layer ii, OL is the maxi- mum value of the activations of layer ii and P is the maximum value of the parameters of layer II.

Then using equation 5, we calculate the number of remaining bits to represent the decimal part. The quantization factor Q represents the available decimal bits.

$$Q_i = 8 - C_i \quad (5)$$

Where Qi is the number of bits to represent the decimal part of layer i.

Table 5 shows the dynamic range of the parameters and the outputs of the layers. The highlighted boxes represent the maximum values used to calculate the value of Qi. The Q values shown in this table generated the best accuracy when quantizing the CNN when to using the DLQ method.

Table 5: Outputs and parameters of each layer with the appropriate Q.

LAYER	LAYER OUTPUT RANGE		PARAMETERS RANGE		Q. FACTOR
	0	0,76	-0,453	0,94	7
CONV 2	0	1,10	-1,569	0,78 8	7
CONV 3	0	0,88	- 2,116	3	6
CONV 4	0	2,19	-2,77	1,13 6	6
FC1	0	7,69	- 2,072	0,99	5
FC2	0	11,35	- 1,295	1,32 1	4
FC3	-28,9	8,48 6	-24	0,84 7	3

Source: Authors own creation

4. IMPLEMENTATION IN THE MICROCONTROLLER

The implementation in the microcontroller was done, describing the quantized CNN inference and verifying the accuracy of the network by sending the data from an SD module to the ATMEGA2560 microcontroller (Atmel, 2014). CNN is made up of thirteen layers, including the input layer as shown in Table 1. The outputs of these thirteen layers must be stored in SRAM memory because their value is constantly changing. To reduce the amount of SRAM memory required, we merge the convolution and max-pooling layers. Figure 2 shows how was reduced the number of outputs from 13 to 9. We calculate all the output values of the convolutional layers, but we only store the outputs of the max-pooling layers.

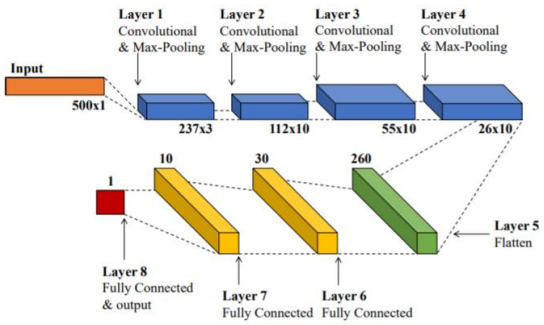

Fig. 2. Convolutional Neural Network with fusion of some layers
Source: Authors own creation

At the output layer, negative values correspond to a non-fibrillated signal and positive values to fibrillated signals. We use this fact to change the activation function Sigmoid (Zhang, C., & Woodland, P. C, 2015), in the FC3 layer, by the function Hard-Limit (MATLAB hardlim, 2020). The Hard-Limit function assigns 0 to negative values and 1 to positive values, so this function converts the CNN output into a binary output. This change reduces the number of bits needed to store CNN output data. Table 6 shows the memory usage in the ATMEGA2560 microcontroller. The network parameters were stored in FLASH memory because the numeric value does not change after stored. To store the network parameters, we use the PROGMEM function (PROGMEM, 2020). The intermediate values are stored in SRAM memory because these values change with each inference. EEPROM memory was not used because it is small and has few write cycles. The implemented network has an inference time of 677[ms] using an oscillator of 12[MHz]. This inference time is enough for this application because the intervals of the ECG signal are collected each two seconds.

Table 6: Memory used on the MCU implementation

ATMEGA 2560			
MEMORY	AVAILABLE	USED	% USE
FLASH	256 KB	25,56 KB	10 %
SRAM	8 KB	4,44 KB	54 %

Source: Authors own creation

We developed a code in MATLAB to emulate the inference process carried out in the microcontroller. We use this code to validate the accuracy of the network for the quantized methods tested in this work.

5. RESULTS

Table 7 summarizes the results obtained in this work. In this table, we compare the accuracy of the quantization methods against the CNN at 64-bits double-precision floating point format (Unquantized). We also show the percentage of accuracy loss regarding unquantized network. Finally, we indicate the possibility of implementing each quantized model in an-8-bit microcontroller.

Table 7: Accuracy, loss accuracy and supported for each quantization methods

METHOD	ACCURACY	% LOSS	SUPPORTED 8-BIT MCU
UNQUANTIZED	97,44 %	—	N
TF LITE QUANTIZATION FOR SIZE	92,77 %	4,67 %	N
TF LITE INTEGER QUANTIZATION	63,58 %	33,86 %	Y
SLQ	73,51 %	23,93 %	Y
DLQ	89,48 %	7,96 %	Y

Source: Authors own creation

The results show that the network quantized with the Dynamic Layer Quantization method has the best accuracy.

6. CONCLUSIONS

In this work, four quantization methods were tested to implement the Castillo-Granados CNN into an 8-bit microcontroller (ATMEGA2560). First, we used the TensorFlow backend to quantize the network. Our result showed that TensorFlow quantization significantly reduces the precision of the network. The best results regarding accuracy were obtained by a method that we called Dynamic Layer Quantization. Our results showed that the implementation achieves an accuracy of 89.48%, and only uses the 10% in the FLASH memory and 54% in the SRAM memory. A future job will focus on using post quantization strategies, which have proved to improve the accuracy of fake-quantized CNN (Song Han, 2017). This work is part of a project that seeks the development of a low-cost portable device for the early diagnosis of AF.

7. REFERENCES

Roeder, L. (2020). Netron. In PyPI. Retrieved from https://pypi.org/project/netron/

Google. (2020). Post-training Quantization for Size | TensorFlow Lite. In TensorFlow. Retrieved from https://www.tensorflow.org/lite/performance/post_training_quant

Yao, Z., Zhu, Z., & Chen, Y. (2017). Atrial fibrillation detection by multi-scale convolutional neural networks. *2017 20th International Conference on Information Fusion (Fusion)*, 1–6.

Kwasniewska, A., Szankin, M., Ozga, M., Wolfe, J., Das, A., Zajac, A., Ruminski, J., & Rad, P. (2019). Deep Learning Optimization for Edge Devices: Analysis of Training Quantization Parameters. *IECON 2019-45th Annual Conference of the IEEE Industrial Electronics Society*, *1*, 96–101.

Goldberger, A. L., Amaral, L. A. N., Glass, L., Hausdorff, J. M., Ivanov, P. C., Mark, R. G., Mietus, J. E., Moody, G. B., Peng, C., & Stanley, H. E. (2000). *The MIT-BIH Atrial Fibrillation Database*. Retrieved from http://physionet.incor.usp.br/physiobank/database/afdb/

Seo, S., & Kim, J. (2019). Efficient Weights Quantization of Convolutional Neural Networks Using Kernel Density Estimation based Non-uniform Quantizer. *Applied Sciences*, *9*(12), 2559.

Zhang, C., & Woodland, P. C. (2015). Parameterised sigmoid and ReLU hidden activation functions for DNN acoustic modelling. *Sixteenth Annual Conference of the International Speech Communication Association*.

Xia, Y., Wulan, N., Wang, K., & Zhang, H. (2018). Detecting atrial fibrillation by deep convolutional neural networks. *Computers in Biology and Medicine*, *93*, 84–92.

Arduino Reference - PROGMEM. (2020). In Arduino.cc. Retrieved from https://www.arduino.cc/reference/tr/language/variables/utilities/progmem/

Song Han. (2017). *EFFICIENT METHODS AND HARDWARE FOR DEEP LEARNING* (Issue September). STANFORD UNIVERSITY.

Castillo, J. A., Granados, Y. C., & Fajardo, C. A. (2020). Patient-Specific Detection of Atrial Fibrillation in Segments of ECG Signals using Deep Neural Networks. *Ciencia E Ingenieria Neogranadina*, *30*(1). doi: https://doi.org/10.18359/rcin.4156

Google. (2020). TensorFlow Lite guide. In TensorFlow. Retrieved from https://www.tensorflow.org/lite/guide.

Gupta, S., Agrawal, A., Gopalakrishnan, K., & Narayanan, P. (2015). *Deep Learning with Limited Numerical Precision*. *37*. doi: 10.1109/72.80206

Atmel. (2014). *ATmega 640/V-1280/V-1281/V-2560/V-2561/V - Datasheet*. 435. Retrieved from https://ww1.microchip.com/downloads/en/devicedoc/atmel-2549-8-bit-avr-microcontroller-atmega640-1280-1281-2560-2561_datasheet.pdf

Agarap, A. F. (2018). Deep learning using rectified linear units (relu). *ArXiv Preprint ArXiv:1803.08375*.

Pourbabaee, B., Roshtkhari, M. J., & Khorasani, K. (2018). Deep convolutional neural

networks and learning ECG features for screening paroxysmal atrial fibrillation patients. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, *48*(12), 2095–2104.

Lip, G. Y. H., Fauchier, L., Freedman, S. B., Van Gelder, I., Natale, A., Gianni, C., Nattel, S., Potpara, T., Rienstra, M., Tse, H.-F., & Lane, D. A. (2016). Atrial fibrillation. *Nature Reviews Disease Primers*, *2*(1), 16016. doi: 10.1038/nrdp.2016.16

Krishnamoorthi, R. (2018). *Quantizing deep convolutional networks for efficient inference: A whitepaper*. 7–8. Retrieved from http://arxiv.org/abs/1806.08342

Nagel, M., Amjad, R., Baalen, M., Louizos, C., & Blankevoort, T. (2020). Up or Down ? Adaptive Rounding for Post-Training Quantization. *ArXiv Preprint*.

Sripad, A., & Snyder, D. (1977). A necessary and sufficient condition for quantization errors to be uniform and white. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, *25*(5), 442–448.

Hard-limit transfer function - MATLAB hardlim. (2020). In MathWorks América Latina. Retrieved from https://la.mathworks.com/help/deeplearning/ref/hardlim.html

Kim, D., Yim, H. Y., Ha, S., Lee, C., & Kang, I. (2018). Convolutional Neural Network Quantization using Generalized Gamma Distribution. *ArXiv Preprint ArXiv:1810.13329*.

Athar, A. (2018). An Overview of Datatype Quantization Techniques for Convolutional Neural Networks. *ArXiv Preprint ArXiv:1808.07530*.

Tang, C. Z., & Kwan, H. K. (1993). Multilayer feedforward neural networks with single powers-of-two weights. *IEEE Transactions on Signal Processing*, *41*(8), 2724–2727.

Schwarz, E. M. (1996). Rounding for quadratically converging algorithms for division and square root. *Conference Record of The Twenty-Ninth Asilomar Conference on Signals, Systems and Computers*, *1*, 600–603.