Compression Technique for an Automatic Modulation Recognition CNN model

AliMohammad Hakimi
Institut National de la Recherche Scientifique - Énergie, Matériaux et Télécommunications (INRS-EMT) 800 de la Gauchetière Ouest, Suite 6900, H5A 1K6, Montréal (Québec), Canada

***Abstract*—In this investigation, we delve into the utilization of Deep Learning (DL) models to enhance modulation recognition efficiency within wireless communication systems. Our primary objective is to enhance model efficiency and ensure compatibility with hardware by implementing compression techniques such as quantization and classic optimization. Our focus lies in diminishing computational and memory burdens during inference by systematically lowering the precision of weights and activations (quantization) and changing the hyperparameters of an existing model. Through the combination of these methods, we developed a DL model which is more adept for deployment on hardware platforms with limited resources. Experimental outcomes exhibit notable reductions in model size and computational complexity without sacrificing accuracy, rendering the proposed DL models suitable for real-time applications on edge devices. The source code is available on GitHub.**

Keywords— AMR, DL-based, CNN, Optimization, hyper parameters, quantization.

I. INTRODUCTION

The advances in analog-to-digital converter (ADC) and digital signal processing (DSP) technologies have paved the way for software-defined radio (SDR). With the digitalized components approaching closer to antennas, dynamic reconfigurable capability becomes technically achievable for an SDR terminal. Both the commercial 4G and 5G cellular and military radio applications are driving the demand for adaptive modulation to enable robust and spectrally efficient transmission over a time-varying channel. For multimedia applications, Internet is also expected to provide a broad range of services transmitted at variable data rates while maintaining desired quality of service (QoS). All these requirements motivate SDR to select modulation schemes dynamically. In prior arts, a known pilot symbol is usually embedded in the data frame to indicate the modulation scheme for appropriate demodulation at the receiver. To make the channel bandwidth more efficient, automatic modulation recognition (AMR) attracts more interests in recognizing the modulation scheme without costing precious bandwidth. With the concept of cognitive wireless communication network, automatic modulation classification is not only critical in recognizing the modulations blindly but also finding its way to identify users and nodes in dynamic spectrum access applications [1].

Previously, many AMR methods have deployed probabilistic frameworks in likelihood-based approaches and traditional machine learning frameworks in feature-based

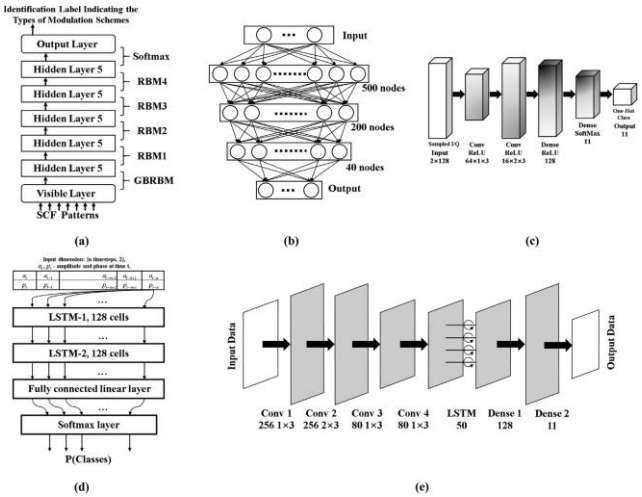

Fig. 1: Representative DL-AMR models for each neural network type. (a) Unsupervised, (b) DNN, (c) CNN, (d) RNN, (e) Hybrid. [1]

approaches. Generally, the likelihood-based approaches apply probability theories and hypothesis models to solve modulation identification problems under the conditions of known and unknown channel information. Although the likelihood-based approaches can reach the optimal classification accuracy with the perfect knowledge of signal model and channel model (which cannot be obtained in the real world), they require high computation complexity to estimate model parameters. By following a regular machine learning (ML) framework for classification task, the feature-based approaches are more favorable to deploy in practical systems compared to the likelihood-based approaches, thanks to their relatively easy implementation and low complexity.

Recently, inspired by the great explosion of Deep Learning (DL) with unprecedented success in different fields, several modern approaches have exploited different DL architectures, such as deep neural networks (DNNs) [2], recurrent neural networks (RNNs) [3], long short-term memory networks (LSTMs) [4], and convolutional neural networks (CNNs) [5] to improve the performance of AMR. Compared with conventional machine learning, DL has presented the essential advantages of automatic feature extraction and high learning capacity, thus increasing the accuracy of higher order modulation formats under a synthetic channel deterioration. Besides that, effectively processing big data enables DL to be deployed for modulation recognition in Internet-of-Things

(IoT) systems, where a vast amount of data is generated by edge devices.

An essential element in DL-AMR revolves around the Signal-to-Noise Ratio (SNR). To effectively perform modulation classification, the DL-based model needs to accurately classify modulation types across a wide range of SNRs. Meeting this criterion frequently leads to the suggestion of complex and computationally demanding models. For DL-based AMRs proposed by previous works, achieving higher accuracy has been done mostly based on stacking more layers and deepening the model. While such strategies may lead to feasible models in many offline tasks, AMR often requires online processing and will encounter excessive delays if a model is too complex. The high computational complexity also prevents their application in resource constrained devices such as many Internet-of-thing (IoT) devices that have limited memory, computing power, and energy, while efficient DL-based AMR models with low complexity and lightweight property can meet the requirement of next generation mobile communication systems, such as B5G systems, which have greater demand for extremely reliable and low-latency. In the machine learning community, model compression and simplification methods such as pruning, knowledge distillation, and weight quantization have been shown to successfully decrease model complexity significantly while maintaining the same level of performance, hence could be exploited in DL-AMR. In [6], Average Percentage of Zeros (APoZ) criterion is used to prune a CNN-based AMR, and the results showed that simplified CNN could use only 1.5% ~ 5% parameter and 33% ~ 35% time without losing accuracy more than 1.2%. Besides, in [7], for a CNN-based AMR, a new filter-level pruning technique based on activation maximization (AM) that omits the less important convolutional filter has been presented. Results demonstrated a higher classification accuracy in the RadioML2016.10a dataset compared to other network pruning techniques. And finally, in [8], an effective LightAMR method using CNN and compressive sensing under varying noise regimes is presented which resulted in fewer device memories and faster computational speed under the limited performance loss.

In this work we aim to investigate model compression for a CNN-based AMR model presented in [5]. Our objective is to enhance the model effectiveness and adaptability to hardware by utilizing model compression methods, particularly classic optimization and quantization. These techniques are employed to reduce the computational and memory demands of model inference. Decreasing the architectural hyperparameters of a CNN model, known as classic optimization, can result in significant changes in the model complexity. However, this adjustment typically has a detrimental impact on the model accuracy. Therefore, any alterations to the hyperparameters should aim to minimize variations in accuracy levels. Similarly, quantization involves reducing the precision of weights and activations from high-precision floating-point numbers (e.g., 32-bit) to lower-precision representations, such as 8-bit integers. This approach further decreases the model memory requirements and speeds up inference by simplifying arithmetic operations. By integrating these methods, the objective is to develop a more efficient DL model suitable for deployment on hardware platforms with limited resources, while ensuring an acceptable level of accuracy is maintained.

II. Compression Technique Fundamentals

A. Modfication of Artichecural Hyperparametrs

The hyperparameters of a DL model play a crucial role in determining the complexity the model. Mostly, DL models designed for specific applications are not optimized efficiently, resulting in unnecessarily large and complex models that require significant resources to run. In this study, we began by studying the original model described in [5]. This model is consisted of 6 CNN layers. Having analyzed the model, we reduced the filter size of each CNN layer and added one layer, resulting in 7 layers in total. This adjustment allowed us to decrease the complexity by 70% while maintaining an acceptable level of accuracy compared to the original model. In order to make a comparison, Table I shows the architecture of original and optimized model. It can be observed that the number of learnable parameters for each layer has decreased, this will further discuss in the experimental result part.

Table I
Architecture of the Original and Optimized Model

Model	Layer Type	Learnable Parameters	Hyperparameter
Original Model	Conv2D	272	16 filters of size [1 4]
	Conv2D	3096	24 filters of size [1 4]
	Conv2D	6176	32 filters of size [1 4]
	Conv2D	12336	48 filters of size [1 4]
	Conv2D	24640	64 filters of size [1 4]
	Conv2D	49248	96 filters of size [1 4]
Optimized Model	Conv2D	144	16 filters of size [1 4]
	Conv2D	792	24 filters of size [1 2]
	Conv2D	1568	32 filters of size [1 2]
	Conv2D	3120	48 filters of size [1 2]
	Conv2D	6208	64 filters of size [1 2]
	Conv2D	12384	96 filters of size [1 2]
	Conv2D	24704	128 filters of size [1 2]

B. Quantaztion Basics

Quantization refers to the process of reducing the precision of numerical values used to represent parameters (such as weights and activations) in a DL model. Typically, DL models use high-precision floating-point numbers, such as 32-bit floating-point values, to represent these parameters. However, for deployment on resource-constrained hardware platforms, it is beneficial to reduce the precision of these parameters to lower-precision representations, such as 8-bit integers. This results in smaller memory footprint and reduced computational requirements during inference, since lower-precision representations require less memory and entail simpler arithmetic operations. Research works have shown that neural networks can be successfully quantized to lower bit-widths with minimal effect on the model accuracy, and this suggests that quantization is a reliable technique [9].

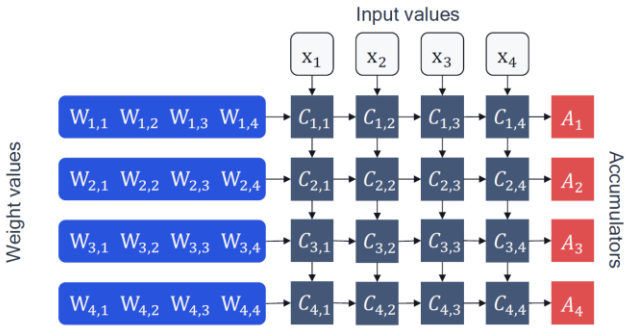

Fig. 2. A schematic overview of matrix-multiply logic in neural network accelerator hardware [9].

Fig. 2. shows a basic outline of how a crucial operation in neural network (NN) accelerators, matrix-vector multiplication ($y = Wx + b$) can be done. This operation forms the basis for more complex calculations in neural networks. The hardware components involved aim to make NN inference more efficient by doing many calculations simultaneously. These components include processing elements ($C_{n,m}$) and accumulators (A_n). In the example of Fig. 2, there are 16 processing elements in a grid and 4 accumulators. The process starts by loading accumulators with bias values (b_n). Then, weight values ($W_{n,m}$) and input values (x_m) are loaded, and their products are computed in processing elements ($C_{n,m} = W_{n,m}\, x_m$) in one cycle. The results are summed in the accumulators using the formula:

$$\hat{A}_n = \hat{b}_n + \sum C_{n,m} \qquad (1)$$

This operation is known as Multiply-Accumulate (MAC). The process is repeated for larger calculations. Once done, accumulator values are sent back to memory for use in the next neural network layer. Neural networks are usually trained using FP32 weights and activations. But performing inference in FP32 requires supporting floating-point logic and transferring 32-bit data, which consumes a lot of energy. Using lower-bit fixed-point or quantized representations like INT8 can significantly reduce data transfer and energy consumption during MAC operations. This is because the expense of digital arithmetic generally increases proportionally to quadratically with the quantity of bits employed, and fixed-point addition is more effective than its floating-point equivalent. In order to transition from floating-point to efficient fixed-point operations, a method for converting floating-point vectors into integers is required. Essentially, a floating-point vector x can be approximated as a scalar multiplied by an integer vector:

$$\hat{x} = s_x . x_{int} \approx x \qquad (2)$$

In this context, S_x represents a floating-point scale factor, and x_{int} stands for an integer vector, such as INT8. This transformed version of the vector is labeled as $\hat{x}$. By quantizing the weights and activations, express the transformed version of the accumulation equation can be expressed as:

$$A_n = b_n + s_w s_x \sum W_{n,m}^{int}\, x_m^{int} \qquad (3)$$

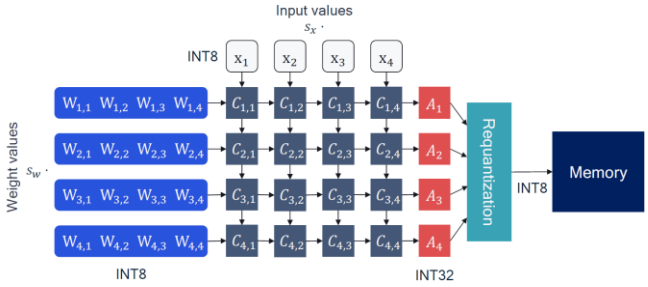

Fig. 3. A schematic of matrix-multiply logic in a neural network accelerator for quantized inference [9].

Different scale factors for weights s_w and activations s_x to be more flexible and to reduce quantization errors is used. This helps simplify calculations, allowing to perform operations in fixed-point format. Based on [9], bias quantization should be ignored, because the bias is normally stored in higher bit-width (32-bits) and its scale factor depends on that of the weights and activations. Fig. 3. illustrates the alterations in the neural network accelerator with the introduction of quantization. In Fig. 3 demonstration, INT8 arithmetic is utilized, although any quantization format could apply for this purpose. It is crucial to retain a wider bit-width for the accumulators, typically 32-bits wide. This precaution helps prevent loss from overflow as more products are accumulated during computation. Quantization is characterized by three parameters: the scale factor (s), the zero-point (z), and the bit-width (b). These parameters are employed to convert a floating-point value to an integer grid, the dimensions of which are determined by the bit-width. The scale factor, usually depicted as a floating-point number, defines the quantizer step-size. Meanwhile, the zero-point, an integer, guarantees accurate quantization of real zero, which is crucial for preserving the integrity of common operations like zero padding or ReLU without introducing quantization errors.

Once the three quantization parameters are established, we can move forward with the quantization process. Initially, from a real-valued vector x, we map it to the unsigned integer grid $\{0, \ldots, 2^b - 1\}$.

$$x_{int} = clamp\left(\left\lfloor \frac{x}{s} \right\rceil + z; 0, 2^b - 1\right) \qquad (4)$$

where $\lfloor . \rceil$ is the round-to-nearest operator and clamping is defined as:

$$clamp\,(x; a, c) = \begin{cases} a, & x < a \\ x, & a \le x \le c \\ c, & x > c \end{cases} \qquad (5)$$

To approximate the real-valued input x a de-quantization step is preformed:

$$x \approx \hat{x} = s(x_{int} - z) \qquad (6)$$

Combining the two steps above, a general definition for the quantization function, q(.), can be provided as:

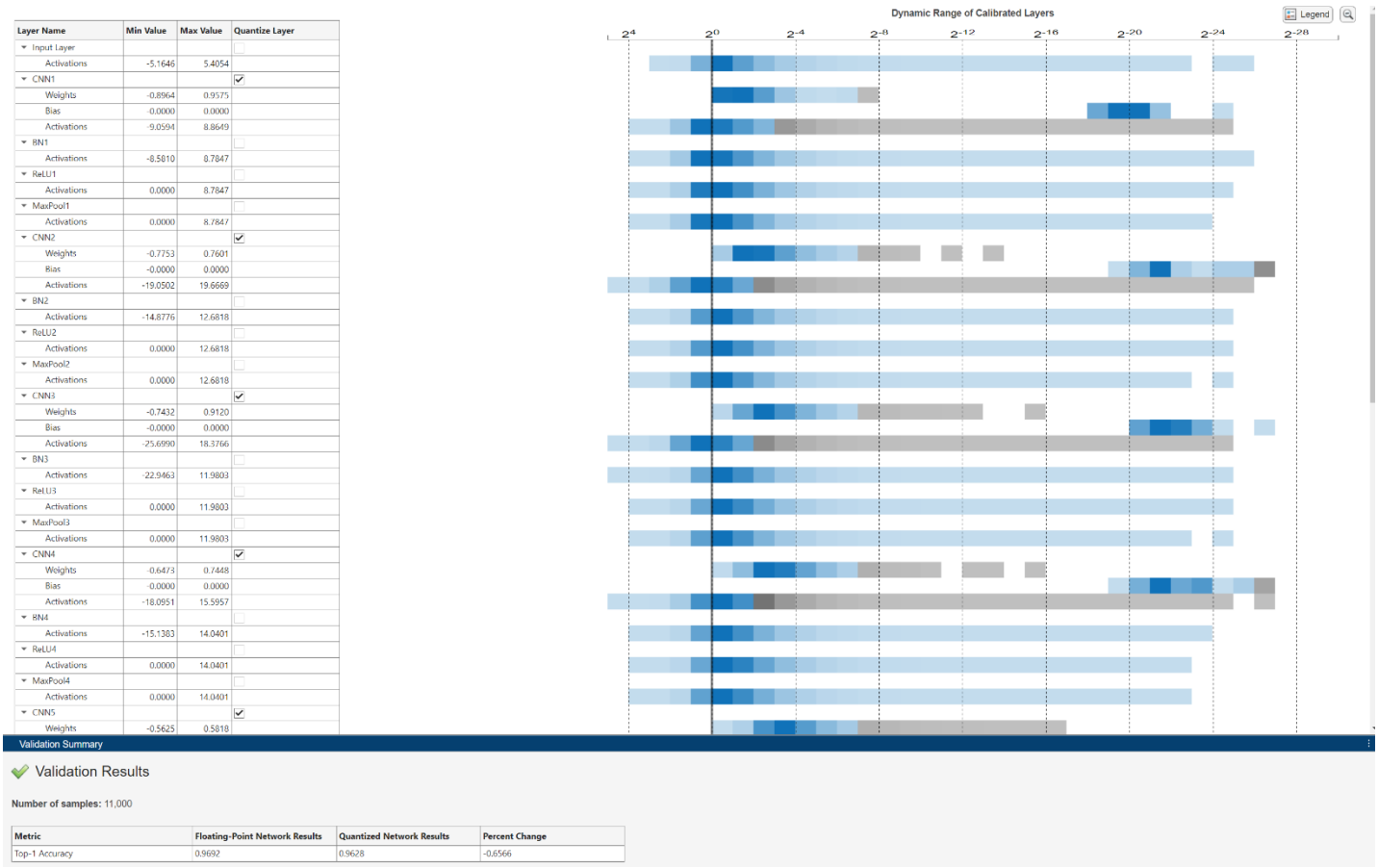

Fig. 4. Picture of MATLAB quantizer APP for result of quantizing the optimized model, dynamic range of calibrated layers, top-1 accuracy and the quantized layers of CNN

$$\hat{x} = \mathrm{q}(x; s, z, b) = s\left[clamp\left(\left\lfloor\frac{x}{s}\right\rfloor + z; 0, 2^b - 1\right) - z\right] \qquad (7)$$

Quantization, which includes Post-Training Quantization (PTQ) and Quantization-Aware Training (QAT), provides efficient techniques for reducing energy consumption and memory usage in neural networks (NNs). PTQ is a lightweight method that achieves quantization without re-training or labeled data by converting high-precision model weights (e.g., FP32) to lower precision formats like INT8. On the other hand, QAT involves fine-tuning with access to labeled training data and addresses quantization errors during training using fake-quantization modules. It ensures high accuracy by accounting for quantization effects in both forward and backward passes. In comparison to PTQ, QAT typically delivers superior accuracy.

C. Quantization Precedure

After finishing the classic optimization described in section II. A, the resulted model is used to proceed for quantization. MATLAB R2023b Deep Learning toolbox for quantization of the model is utilized. In the procedure, we first created an augmented image datastore object to use for calibration and validation. The calibration data is used to collect the dynamic ranges of the weights and biases in the convolution and fully connected layers of the network and the dynamic ranges of the activations in all layers of the network. For the best quantization results, the calibration data must be representative of inputs to the network. The validation data is used to test the network after quantization to understand the effects of the limited range and precision of the quantized convolution layers in the network.

Table II
The quantized values for learnable parameters in the quantized model

Layer	Parameter	Value
CNN1	Weights	4-D int8
CNN1	Bias	1x1x16 int32
CNN2	Weights	4-D int8
CNN2	Bias	1x1x24 int32
CNN3	Weights	4-D int8
CNN3	Bias	1x1x32 int32
CNN4	Weights	4-D int8
CNN4	Bias	1x1x48 int32
CNN5	Weights	4-D int8
CNN5	Bias	1x1x64 int32
CNN6	Weights	4-D int8
CNN6	Bias	1x1x96 int32
CNN7	Weights	4-D int8
CNN7	Bias	1x1x128 int32

Having defined the validation and calibration data, now we use dlquantizer object to reduce the memory requirement of the pretrained model by quantizing weights and activations to 8-bit

scaled integer data types. It is possible to create and verify the behavior of a quantized network for GPU, FPGA, CPU deployment, or explore the quantized network in MATLAB. In our work, we explored the quantized network in MATLAB.

Table III
Comparison between original model and quantized optimized models.

Model	Accuracy	MAC	Parameter Memory (MB)
Original	97.7%	8127520	0.3693999
Optimized	97.43%	2491798	0.19202802
Optimized and Quantized	97.05%	2491798	0.05206704

After definition of dlquantizer object, the calibrate function is used to exercise the network with sample inputs and collect range information. This function exercises the network and collects the dynamic ranges of the weights and biases in the convolution and fully connected layers of the network and the dynamic ranges of the activations in all layers of the network. The function returns a table, each row of which contains range information for learnable parameters of the optimized model. In this step, we can quantize the object using quantize function. This function quantizes a deep neural network using a calibrated dlquantizer object. The quantized neural network object enables visibility of the quantized layers, weights, and biases of the network, as well as simulatable quantized inference behavior. Fig. 4. shows the dynamic range of calibrated layers for the trained model, and it also shows that which layers are quantized. It is possible to select the layers we required to be quantized. We use the quantizationDetails function to see if the network is now quantized. This function returns a 1-by-1 structure array containing the quantization details for the neural network. The data is returned as a structure with the fields:

- IsQuantized — Returns 1 (true) if the network is quantized; otherwise, returns 0 (false).
- TargetLibrary — Target library for code generation.
- QuantizedLayerNames — List of quantized layers.
- QuantizedLearnables — Quantized network learnable parameters.

It is possible to go further and investigate the other details of the quantized network using this function. Table II shows the quantized values for learnable parameters in the quantized model. It can be observed that as mentioned before the bias quantization is ignored.

D. Experimental Results

This section will mainly discuss the obtained results for the deployment of optimized and quantized model in the inference phase. Since we had not access to the target hardware, we created a target agnostic, simulatable quantized deep neural network in MATLAB. All trainings, tests, optimizations, and quantization are performed in MATLAB R2023b. Here are the PC hardware specs of used system for MATLAB simulations:

- Processor: 12th Gen Intel(R) Core(TM) i5-12600KF 3.70 GHz
- Installed RAM: 32.0 GB (31.9 GB usable)
- GPU: Amd Radeon R7 200 series

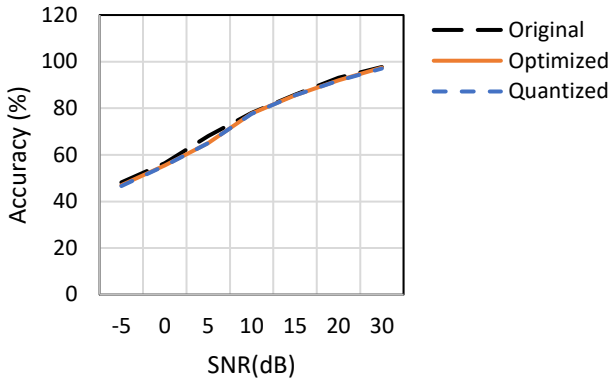

Fig. 5. Top-1 accuracy results of the quantized model for different SNRs

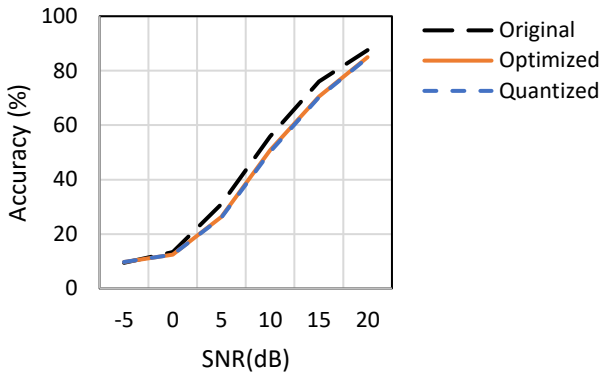

Fig. 6. Top-1 accuracy results of the quantized model for different SNRs

Table III outlines the main result of this investigation. The accuracy, Number of MACs, and Parameter Memory are measured for three different trained model with dataset of 10,000 frames and SNR=30 dB, including original model [5]; optimized model from section A; and quantized as well as optimized model. Parameter memory is a metric to measure the memory used to store the weights and biases parameters of a DL model. The results are measured in MATLAB using estimateNetworkMetrics function. The results imply that the optimized quantized model has the best performance in terms of complexity and memory, while maintaining a good level of accuracy compared to original model. For all three models, top-1 accuracy is almost 97% while Parameter Memory and Number of MACs decreased by 85% and 70%.

In order to investigate the generality of model for other SNRs, results of accuracy for other SNRs are demonstrated in Fig. 5. Additionally, the quantized trained model for SNR=30 and

dataset 10000 frames has been tested for different SNRs and a comparison with the same results of original model is made in Fig. 6. It can be observed that for different SNRs the accuracy results of optimized and quantized models are in good agreement with the original model.

III. Conclusion

In this study, we used classic optimization method and quantization to compress and lower the complexity of an existing modulation recognition model presented in [5]. The source code is in MATLAB, and we used MATLAB DL toolbox to quantize the model. The results suggest that the quantized model outperforms original model regarding complexity and memory usage, while still retaining a acceptable level of accuracy when compared to the original model. For all model, the top-1 accuracy remains close to 97%, while the Parameter Memory and Number of MACs have notably decreased by 85% and 70%, respectively.

References

[1] Zhang F, Luo C, Xu J, Luo Y, Zheng FC. Deep learning based automatic modulation recognition: Models, datasets, and challenges. Digital Signal Processing. 2022 Sep 1;129:103650.

[2] W. Shi, D. Liu, X. Cheng, Y. Li, Y. Zhao, Particle swarm optimization-based deep neural network for digital modulation recognition, IEEE Access 7 (2019) 104591–104600.

[3] D. Hong, Z. Zhang, X. Xu, Automatic modulation classification using recurrent neural networks, in: Proc. IEEE Int. Conf. Comput. Commun., 2017, pp.695–700.

[4] S. Rajendran, W. Meert, D. Giustiniano, V. Lenders, S. Pollin, Deep learning models for wireless signal classification with distributed low-cost spectrum sensors, IEEE Trans. Cogn. Commun. Netw. 4 (2018) 433–445.

[5] O'Shea, T. J., J. Corgan, and T. C. Clancy. "Convolutional Radio Modulation Recognition Networks." Preprint submitted June 10, 2016. https://arxiv.org/abs/1602.04105.

[6] Y. Tu and Y. Lin, "Deep Neural Network Compression Technique Towards Efficient Digital Signal Modulation Recognition in Edge Device," in IEEE Access, vol. 7, pp. 58113-58119, 2019, doi: 10.1109/ACCESS.2019.2913945.

[7] Y. Lin, Y. Tu and Z. Dou, "An Improved Neural Network Pruning Technology for Automatic Modulation Classification in Edge Devices," in IEEE Transactions on Vehicular Technology, vol. 69, no. 5, pp. 5703-5706, May 2020, doi: 10.1109/TVT.2020.2983143.

[8] Y. Wang, J. Yang, M. Liu and G. Gui, "LightAMC: Lightweight Automatic Modulation Classification via Deep Learning and Compressive Sensing," in IEEE Transactions on Vehicular Technology, vol. 69, no. 3, pp. 3491-3495, March 2020, doi: 10.1109/TVT.2020.2971001.

[9] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. Van Baalen, and T. Blankevoort, “A whit paper on neural network quantization,” arXiv preprint arXiv:2106.08295, 2021