Leveraging Model Compression Techniques for Efficient Deployment of Deep Learning in Automatic Modulation Recognition

Mobin Vaziri
Department of Computer and Software Engineering, Polytechnique Montréal
mobin.vaziri@polymtl.ca

***Abstract*—In this study, we explore the use of Deep Learning (DL) models for efficient modulation classification in wireless communication. We aim to improve the efficiency and hardware compatibility of these models by employing model compression techniques such as pruning and quantization. Our study focuses on reducing computational and memory costs during inference by systematically removing unimportant parameters from neural networks (pruning) and reducing the precision of weights and activations (quantization). By combining these techniques, we create more efficient DL models suitable for deployment on resource-constrained hardware platforms. We utilize TensorFlow Lite for pruning and quantization and NVIDIA TensorRT for benchmarking on GPUs specialized for DNN inference. Experimental results demonstrate significant reductions in model size and computational complexity while maintaining accuracy, making the proposed DL models suitable for real-time applications on edge devices. The source code is available on GitHub[1].**

***Index Terms*—Deep Learning, Modulation Classification, Model Compression, Edge Computing**

I. Introduction

Deep Learning (DL) models have become increasingly popular due to their ability to solve complex problems. They are being used in various fields, such as Computer Vision tasks, including image recognition [1], object detection [2], image segmentation [3], and Natural Language Processing tasks, such as speech recognition and text classification [4].

In wireless communication, DL models are proving to be useful in solving problems where traditional analytical or model-based solutions fail. They have been particularly effective in the end-to-end (physical) layer and in spectrum situation awareness [5]. Many Deep Neural Network (DNN)-based approaches have been explored in this area. In the conventional communication chain, which comprises a Transmitter (TX), a Receiver (RX), and a channel, deep autoencoders have been employed to model the TX and RX. This approach allows for a more efficient and accurate representation of the communication process. For channel estimation and modeling, Generative Adversarial Networks (GANs) have been utilized in conjunction with the physical channel in the communication chain. The GANs are trained to learn the underlying patterns or distributions of data, enabling them to generate new, similar data. This approach allows for a more accurate approximation of the channel, leading to improved communication performance.

Another significant problem addressed by DL is Automatic Modulation Recognition (AMR) [6]. AMR is important in providing crucial modulation information of incoming radio signals, especially non-cooperative radio signals. It plays a vital role in various scenarios including cognitive radio, spectrum sensing, signal surveillance, and interference identification. The objective of AMR is to detect the modulation scheme of wireless communications signals automatically, without any prior information. Research on AMR is typically split into two categories: Likelihood Theory-based AMR (LB-AMR) and Feature-based AMR (FB-AMR). LB-AMR methods often achieve optimal recognition accuracy in Bayesian estimation but are computationally complex. FB-AMR methods focus on learning features from training samples and classifying incoming signals with trained models.

With the rise of machine learning, models like Artificial Neural Networks (ANNs), are becoming increasingly common for classification tasks. Although FB-AMR methods offer sub-optimal solutions, they have the advantages of low computational complexity and the ability to identify multiple modulations. ANNs, especially those with stacked multilayer architectures, have shown powerful feature extraction capabilities and spurred extensive research into modulation detection. Some pioneering DL-based methods have been proposed, often outperforming traditional LB-AMR and FB-AMR methods.

Previous research on DL-AMR has mainly focused on Single-Input-Single-Output (SISO) systems, but there has been recent coverage of DL-AMR for Multiple-Input-Multiple-Output (MIMO) systems. For these problems, Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs) networks, or hybrid models combining these architectures are commonly used. CNN models have demonstrated exceptional capability in processing data with spatial characteristics. In AMR research, CNNs have been introduced to detect signal modulation schemes by leveraging their spatial feature extraction power [7]. Depending on the input data types, existing CNN-based AMR approaches can be divided into two categories: CNN models with raw I/Q inputs and CNN models with pre-processed inputs.

Wireless communication signals also carry temporal cor-

[1]https://github.com/smartward3n/TEL-358-Machine-Learning-for-Wireless-Communication

relation features, which can be learned by machine learning models such as RNNs. Several novel model architectures based on RNN have been proposed recently, achieving state-of-the-art performance in AMR [8]. Pure CNN or RNN models focus on either the spatial or temporal features of AMR signals, so using only one type may not yield optimal performance. Researchers have proposed combining the characteristics of both types of neural network layers to build hybrid models for AMR. For example, a CNN-Long-Short-Term Memory DNN model, consisting of one Long short-term memory (LSTM) and three CNN layers, was proposed. This model includes a skip connection before the LSTM, bypassing two CNN layers and providing a longer time context for extracted features [9].

A crucial factor in DL-AMR is the Signal-to-Noise Ratio (SNR). For modulation classification, the DL-based model should be capable of classifying the modulation type across a broad spectrum of SNRs. This requirement often results in the proposal of deep and computationally intensive models. However, this becomes a bottleneck when we aim to perform these classification or prediction tasks in real-time on Edge devices, which are resource-constrained hardware platforms.

In this study, our objective is to explore DL models proposed for modulation classification and enhance their efficiency and hardware compatibility by implementing model compression techniques, namely pruning and quantization [10], [11]. Pruning and quantization are techniques aimed at reducing the computational and memory costs of model inference. Pruning involves systematically removing unimportant parameters, such as weights and biases, from the neural network, thereby reducing its complexity. This process helps in reducing the model's memory footprint and accelerates inference by decreasing the number of computations required. On the other hand, quantization reduces the precision of weights and activations from high-precision floating-point numbers, such as 32-bit, to lower-precision representations, like 8-bit integers. This technique further reduces the memory footprint of the model and speeds up inference by simplifying arithmetic operations. By combining pruning and quantization, we aim to create more efficient DL models that are suitable for deployment on resource-constrained hardware platforms.

II. Model Compression Techniques

A. Quantization

Quantization is a technique that can significantly reduce the computational time and energy consumption of Neural Networks (NNs). This method involves storing the weights and activation tensors in lower bit precision than the standard 32-bit precision used during training. When transitioning from 32 to 8 bits, the memory overhead of storing tensors decreases by a factor of 4, while the computational cost for matrix multiplication decreases quadratically by a factor of 16. It has been demonstrated that neural networks can be effectively quantized to lower bit-widths with only a minor impact on the network's accuracy, which means that quantization is a robust technique [10]. The NN accelerator consists of two essential elements: the processing elements $C_{n,m}$ and the accumulator A_n. The computation begins by initializing the accumulators with the bias value b_n. Subsequently, the weight values $W_{n,m}$ and the input values x_m are loaded into the array, and their product is calculated concurrently in the corresponding processing elements $C_{n,m} = W_{n,m} \times x_m$ within a single cycle. The outcomes are then accumulated in the accumulator:

$$A_n = b_n + \sum_m C_{n,m} \tag{1}$$

The described operation is commonly known as Multiply-Accumulate (MAC). This process is iterated multiple times for larger matrix-vector multiplications. Once all cycles are executed, the values stored in the accumulators are transferred back to memory for utilization in the subsequent NN layer. Typically, NNs are trained using FP32 weights and activations. However, performing inference in FP32 necessitates that both the processing elements and the accumulator support floating-point logic, and require transferring 32-bit data from memory to the processing units. The MAC operations and data transfer are the primary contributors to the energy consumption during neural network inference. Hence, substantial advantages can be attained by employing a lower-bit fixed-point or quantized representation for these quantities. Fixed-point representations, such as INT8, not only reduce data transfer requirements but also diminish the size and energy consumption of the MAC operation [12]. To move from floating-point to efficient fixed-point operations, it is necessary to have a method to convert the floating-point vector $\boldsymbol{x}$ into a discrete grid of integers. This mapping can be either signed or unsigned, depending on the particular application. The process of transforming each element involves the following steps:

$$x_{int} = \text{clip}\left(\lfloor \frac{x}{s} \rceil + z; L, U\right) \tag{2}$$

where s is the scaling factor, z is the zero point, and $\lfloor . \rceil$ is the round-to-nearest integer operator.

For unsigned quantization on b bits, $L = 0$ and $U = 2^b - 1$, whereas for signed quantization we use $L = -2^{b-1}$, and $U = 2^{b-1} - 1$. The function $clip(x; L, U)$ is defined as:

$$\text{clip}(x; L, U) = \begin{cases} L, & x < L, \\ x, & L \leq x \leq U, \\ U, & x > U. \end{cases} \tag{3}$$

This function ensures that the quantized value of $\boldsymbol{x}$ lies within the quantization range L, U. To recover the original real-valued vector $\boldsymbol{x}$ from its quantized representation x_{int}, we apply the following de-quantization step:

$$\hat{x} = s \times (x_{int} - z) \tag{4}$$

Quantization, comprising Post-Training Quantization (PTQ) and Quantization-Aware Training (QAT), offers efficient methods for reducing energy consumption and memory usage in NNs. PTQ, a lightweight approach, achieves quantization without re-training or labeled data by converting high-precision model weights, such as FP32, to lower precision like INT8. QAT, on the other hand, requires fine-tuning with access

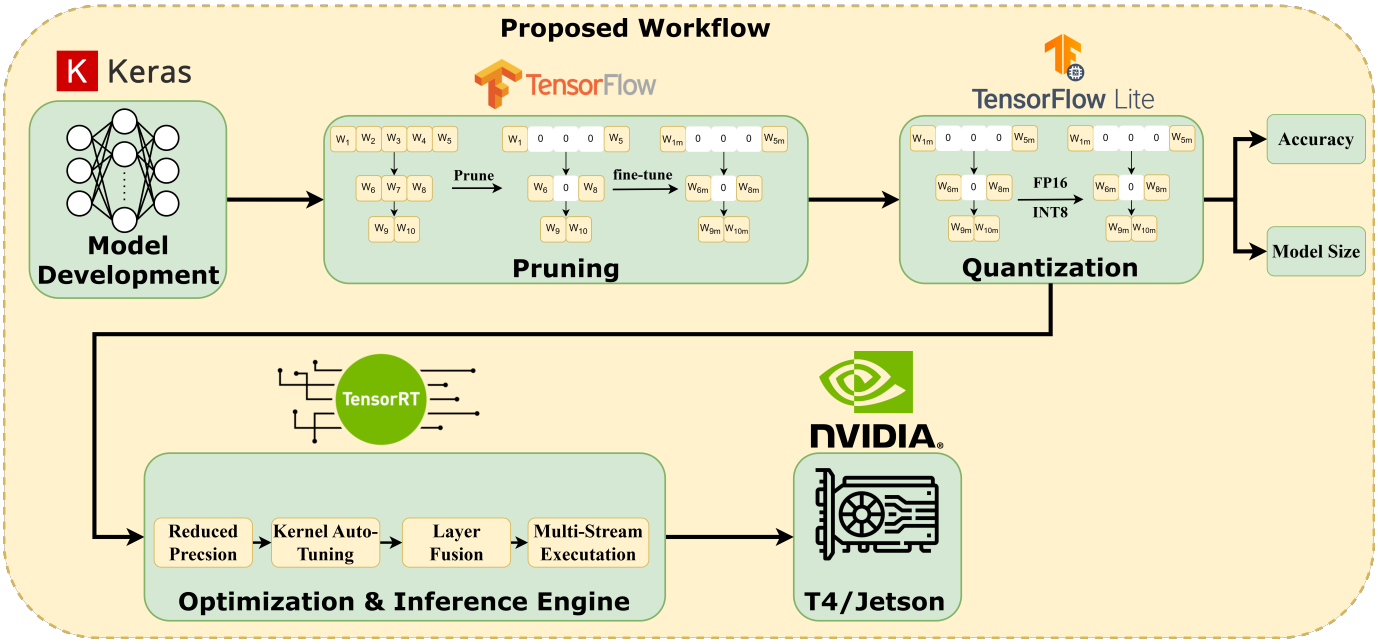

Fig. 1. The Outline of Proposed Workflow.

to labeled training data and models quantization errors during training using fake-quantization modules. It ensures high accuracy by considering quantization effects in both forward and backward passes. Compared to PTQ, QAT generally yields superior accuracy.

B. Pruning

Pruning is a technique that is essential to simplify and optimize neural networks, especially for deploying large and accurate models in computing environments that are resource-constrained or cost-limited [11]. Various pruning methods exist, each offering unique approaches and advantages. Magnitude-based pruning involves removing weights with the smallest absolute values under the assumption that they contribute minimally to the output. Structured Pruning goes further by removing entire neurons, layers, or channels, resulting in significant reductions in model size and computational requirements. This method enables efficient hardware acceleration by capitalizing on structured sparsity patterns. Clustering-based pruning identifies redundancy by clustering similar weights and replacing them with representative ones. Sensitivity Analysis-Based Pruning evaluates the impact of pruning on performance, removing weights with minimal degradation. Pruning benefits hardware efficiency and performance by reducing model size, allowing deployment on devices with limited memory. Fewer parameters and connections facilitate faster inference, crucial for real-time applications. Pruned models consume less power, critical for devices with limited battery life or environmental concerns.

III. Proposed Workflow

The proposed method is outlined in Fig. 1. Our initial goal is to reconstruct the model used for AMC in Keras, which is a high-level API on TensorFlow. In this work, we focus on two CNN models, which are explained in detail in the following sections. Next, we prune the model parameters through structured pruning. The model is fine-tuned so that it becomes aware of pruning. We use TensorFlow Lite (TFLITE) for PTQ and evaluate the model size and accuracy of the pruned and quantized models. To observe the effect of quantization on energy consumption at the circuit level, we estimate the energy consumption for quantized models. In the final step, NVIDIA TensorRT is used to optimize the quantized models for GPUs in different classes, and we benchmark the inference speed. The following subsections provide a detailed explanation of the proposed workflow.

A. Model Development

In this study, we consider working with two CNN models [7], [13]. The model parameters are presented in Table I, and the hyperparameters from model training, inspired by [6], are illustrated in Table II. These models are trained using the RML2016.10a dataset, where the sample dimensions are 2×128, consisting of 220,000 data samples for SNR ranging from -20 to 18 dB and encompassing 11 modulation schemes. Based on our training, the model in [7] with 1,592,383 parameters achieves an accuracy of 56.24%, while the model presented in [13] with 858,123 parameters reaches an accuracy of 56.6% on the test set, averaged across all SNRs. To ensure the correctness of our results, we have validated them by comparing them to the study in [6].

B. Structural Pruning

Structural pruning systematically zeroes out model weights at the beginning of the training process. This technique is applied to regular blocks of weights to accelerate inference

TABLE I
SUMMARY OF THE MODEL PARAMETERS

Model	Layer Type	#Parameters	Description
Model 1 [7]	Conv2D	450	50 filters of size (1, 8)
	Con2D	40050	50 filters of size (2, 8)
	Dense	1549056	Fully connected layer with 256 units
	Dense	2827	Output layer with 11 units
Model 2 [13]	Conv2D	4352	256 filters of size (2, 8)
	Conv2D	524416	128 filters of size (2, 8)
	Conv2D	131136	64 filters of size (2, 8)
	Conv2D	65600	64 filters of size (2, 8)
	Dense	131200	Fully connected layer with 128 units
	Dense	1419	Output layer with 11 units

TABLE II
SUMMARY OF THE TRAINING HYPERPARAMETERS

Hyperparameter	Value
Learning Rate (lr)	0.001
Beta 1 (beta_1)	0.9
Beta 2 (beta_2)	0.999
Loss Function	Categorical Crossentropy
Optimizer	Adam
Epochs	110
Batch Size	256

on supporting hardware. In this study, we experimented with three different percentages of sparsity to measure accuracy using TensorFlow's constant sparsity feature [14]. The process begins by defining the desired percentage of sparsity in each layer for the baseline pre-trained model. Subsequently, through fine-tuning, the model parameters are pruned and made aware of sparsity. Structural pruning can result in acceleration in inference on supported hardware. One of the reasons is that if we can completely remove one or many filters, our specialized hardware will skip zeros for both computation and data movement between different memory levels. The results of pruning both models with different sparsity levels, along with the model parameters, are presented in Table III.

C. Quantization

TFLITE supports both PTQ and QAT, but with limited data types specialized for hardware accelerators like GPUs, DSPs, and EdgeTPU. For PTQ, it offers three different methods: float16 quantization, dynamic range quantization, and integer quantization. Unlike fully integer quantization, which quantizes activations (input and output feature maps) to INT8, the other methods do not require representative data for calibration and do not result in significant accuracy degradation. In dynamic range quantization and float16 quantization, the model parameters are quantized while the activations remain in float32. Since these methods do not require labeled data and are faster compared to QAT methods, we choose PTQ methods for this study, as represented in Fig. 1. The baseline accuracy and model size, as well as their quantized versions, are presented in Table IV.

TABLE III
PRUNED MODELS WITH DIFFERENT SPARSITY LEVELS

	Model 1		Model 2	
	Accuracy	#Parameters	Accuracy	#Parameters
Sparsity 25%	56.0090 %	1,531,258	56.3613 %	553,992
Sparsity 50%	55.8636 %	1,016,217	56.0568 %	429,062
Sparsity 75%	54.0590 %	400,092	54.6272 %	183,431

TABLE IV
COMPARISON BETWEEN BASE MODEL AND QUANTIZED VERSIONS

	Model 1		Model 2	
	Accuracy	Model Size	Accuracy	Model Size
FP32	56.24 %	5.83 MB	56.60 %	3.31 MB
FP16	55.86 %	2.91 MB	56.05 %	1.64 MB
INT8	55.97 %	1.46 MB	54.62 %	0.83 MB

D. Energy Estimation

In this study, for energy estimation, we utilized an extension of the Keras framework called Qkeras, developed by Google. Qkeras employs Energy Tables presented in [12] for energy estimation. The technology considered for energy estimation is CMOS 45 nm. For the custom accelerator, we opt to fetch activations from DRAM, while the model parameters are stored on on-chip SRAM. Although the model considered for energy estimation is very high-level, it provides a good estimation for the impact of data types on the energy consumption of the chip. Their energy model can be described as:

$$E(Layer_i) = E(\text{Input}) + E(\text{parameters}) + E(\text{MAC}) + E(\text{Output}) \quad (5)$$

The energy terms in Eq. 5 can be described as reading input and parameters from memory, performing MAC operations, and writing back the result to memory. For Model 1, the energy consumption is 49.46 μJ, 21.31 μJ, and 11.91 μJ for FP32, FP16, and INT8 respectively. Similarly, Model 2 consumes 393.37 μJ, 133.66 μJ, and 49.95 μJ for FP32, FP16, and INT8 respectively. On average for both models, quantizing to FP16 precision saves approximately 2.63 times power, and for INT8 it's 6.01 times, highlighting the importance of precision for hardware implementation.

In the following section, we present the experimental results for deploying the optimized models on specific hardware that have special cores for computation of quantized data types.

IV. EXPERIMENTAL RESULTS

This section will primarily discuss the obtained results for the deployment of trained and optimized models in the inference phase. For training the base models and acquiring

TABLE V
IMPLEMENTATION RESULTS AND COMPARISON

	FLOPS (M)	Sparsity (%)	Datatype	Accuracy (%)	Model Size (MB)	Throughput (Image/sec)
Model 1	6.5	50	FP32	55.8636	3.43	89,966
			FP16	55.8614	1.89	91,264
			INT8	55.8386	0.84	91,578
Model 2	78.86	50	FP32	56.0568	1.87	24,829
			FP16	56.0386	1.04	59,844
			INT8	55.9341	0.49	61,247

the baseline accuracy, we utilize the TPU available on Google Colab. Subsequently, we employ TFLITE for both pruning and quantization of the models using different configurations. To benchmark the models and demonstrate their real performance on GPUs with specialized cores for DNN inference, we utilize NVIDIA TensorRT and NVIDIA® T4 GPU for benchmarks. TensorRT can optimize and deploy models for data centers down to embedded systems like the NVIDIA Jetson family. In Table V, we outline the main results of this study. Firstly, we measure the computational complexity of models using a criteria called Floating-point operations per second (FLOPs). This criteria gives us a good perspective on the relationship between throughput and model parameters. For the final results, we maintain the sparsity at 50%, as we restrict ourselves to accept a 1% tolerance in accuracy reduction for pruning. Combining quantization with this level of pruning shows that for FP16 and INT8 data types, both models can be compressed by 1.8 times and 3.92 times on average compared to the FP32 datatype, while still meeting the accuracy reduction criteria. Regarding throughput, as expected, the second model achieves superior throughput when quantized down to FP16 and INT8 data types, but the difference in throughput for the first model is negligible. However, Model 1 encounters a problem wherein some DNN hyperparameters should be specific numbers, such as powers of 2, that can be highly optimized and mapped for GPU computation, as seen in Model 2.

REFERENCES

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[2] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[3] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello, "Enet: A deep neural network architecture for real-time semantic segmentation," *arXiv preprint arXiv:1606.02147*, 2016.

[4] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[5] T. Erpek, T. J. O'Shea, Y. E. Sagduyu, Y. Shi, and T. C. Clancy, "Deep learning for wireless communications," *Development and Analysis of Deep Learning Architectures*, pp. 223–266, 2020.

[6] F. Zhang, C. Luo, J. Xu, Y. Luo, and F.-C. Zheng, "Deep learning based automatic modulation recognition: Models, datasets, and challenges," *Digital Signal Processing*, vol. 129, p. 103650, 2022.

[7] T. J. O'Shea, J. Corgan, and T. C. Clancy, "Convolutional radio modulation recognition networks," in *Engineering Applications of Neural Networks: 17th International Conference, EANN 2016, Aberdeen, UK, September 2-5, 2016, Proceedings 17*. Springer, 2016, pp. 213–226.

[8] S. Rajendran, W. Meert, D. Giustiniano, V. Lenders, and S. Pollin, "Deep learning models for wireless signal classification with distributed low-cost spectrum sensors," *IEEE Transactions on Cognitive Communications and Networking*, vol. 4, no. 3, pp. 433–445, 2018.

[9] N. E. West and T. O'shea, "Deep architectures for modulation recognition," in *2017 IEEE international symposium on dynamic spectrum access networks (DySPAN)*. IEEE, 2017, pp. 1–6.

[10] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. Van Baalen, and T. Blankevoort, "A white paper on neural network quantization," *arXiv preprint arXiv:2106.08295*, 2021.

[11] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," *Advances in neural information processing systems*, vol. 28, 2015.

[12] M. Horowitz, "1.1 computing's energy problem (and what we can do about it)," in *2014 IEEE international solid-state circuits conference digest of technical papers (ISSCC)*. IEEE, 2014, pp. 10–14.

[13] K. Tekbıyık, A. R. Ekti, A. Görçin, G. K. Kurt, and C. Keçeci, "Robust and fast automatic modulation classification with cnn under multipath fading channels," in *2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)*, 2020, pp. 1–6.

[14] "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/