IEEE Int Joint Conf on Neural Networks
(IJCNN 2021), July 2021

A Novel Training Strategy for Deep Learning Model Compression Applied to Viral Classifications

Marcelo A. C. Fernandes* and H. T. Kung†
*Department of Computer and Automation Engineering, Federal University of Rio Grande do Norte, Natal, RN, Brazil.
†John A. Paulson School of Engineering and Applied Sciences, Harvard University, Cambridge, MA, USA.
Email: *mfernandes@dca.ufrn.br, †kung@harvard.edu

***Abstract*—Deep learning techniques, such as deep neural networks (DNNs), have been used with success in many viral classification problems associated with metagenomics, diagnosis of viral infections, pharmacogenomics, phylogenetic analysis, and others. However, deep learning algorithms require a large number of math operations, and these computations themselves can be a bottleneck for processing the vast number of virus sequences in a short time. Currently, most works in this area use basic DNNs in viral classification, and they are not optimized for computational efficiency. This paper proposes a novel training strategy that simultaneously minimizes both pruning and quantization losses in training compressed models for reducing deep learning computational complexity. In training a compressed convolutional neural network (CNN), the scheme uses weight quantization followed by pruning in each training iteration rather than the pruning followed by quantization. The proposed training strategy scheme has been applied to train compressed models for efficient viral classification of 1600 sequences of four types of viruses associated with three families and one realm. A substantial reduction of DNN weights (77%) and operations (58%) is demonstrated, while maintaining high classification accuracy. These results show that the proposed new training regime of weight quantization followed weight pruning for each training iteration is superior to conventional approaches with weight pruning epochs followed by weight quantization epochs.**

***Index Terms*—Viral classification, deep learning, model compression, model training, quantization, pruning.**

I. Introduction

Recently there have been substantial works in using deep learning (DL) techniques such as DNNs for tertiary analysis in viral classification, viral host classification, and viral segments classification [1]–[14]. Figure 1 depicts a use example of applying DL to genome analysis. Note that the DL can be applied before, during or after the secondary analysis.

Viral genome classification with DL is composed of two cascaded stages, mapping and processing (see Figure 2). In the mapping stage (or encoding stage), the genome sequence is mapped to a feature space [15], [16], and in the processing stage a DNN is used for classfication [1]–[4].

There are several mapping strategies reported in literature. They include word composition [15], [16], number representation [2], [17], and use of digital signal processing techniques such as discrete Fourier transform (DFT) [18], [19], one-hot encoding [1], [3], [5], [7], [9], and others [4], [20], [21]. In some cases, multiple mapping strategies are applied sequentially, as presented in [15], [16], [18]. In the processing stage, DNNs such as convolutional neural networks (CNN) [2]–[6], [8], [9], and long-short term memory (LSTM) neural networks [7], [9]–[12] have been used.

Figure 3 presents a common CNN approach in viral classification [2]. A virus genome sequence of length N is mapped with one-hot encoding, and afterwords processed with a CNN. The CNN is composed by L layers with C convolutional layers, C batch normalization layers, C ReLU layers, C pooling layers, F fully connected layers, $F-1$ dropout layers, and a softmax layer, where $L = 4C + 2F$. CONV1D ($K_k@B_k$) is the k-th convolutional layer with K_k one-dimensional kernels of size B_k, MaxPool1D (S_k) is the max pooling layer with stride S_k, FC (P_k) is k-th fully connected layer of size P_k, and α_k is the k-th dropout probability.

The DL approach for viral classification has a timely application concerning the COVID-19 disease. The SARS-CoV-2 virus has been spreading around the world since the end of 2019. In March 2020, the World Health Organization declared a SARS-Cov-2 pandemic. Controlling outbreaks requires the elucidation of the taxonomic classification and the origin of the virus (SARS-CoV-2) from the genomic sequence, for strategic planning, containment, and treatment of the disease [22]–[24].

DL-based viral classification faces two challenges. First, we need to process a massive number of genome sequences created by next generator sequencing (NGS) [25]. Second, we need to cope with a large number of parameters (weights) and operations associated with DNN models. Prior works for viral classification have not focused on these computational issues.

This work proposes a novel training scheme for compressed DNN models, i.e., quantization followed by pruning for each training iteration. Compared to conventional training methods for compressed models, our proposed approach significantly reduces the number of parameters and their bitwidth while having a minimum impact on model accuracy.

Our model compression approach can enable efficient DL processing in individual laboratories by using local GPUs, thereby reducing the need to use the cloud for inference. This procedure enhances the privacy and security of data and reduces communication costs with the cloud.

II. Related works

In [13], [14], a taxonomic classification for metagenomics applications is proposed. Both works use segments of genome (reads) as DL input (see Figure 1), and the DL output is the number of the classes. In [13], DL models are used to

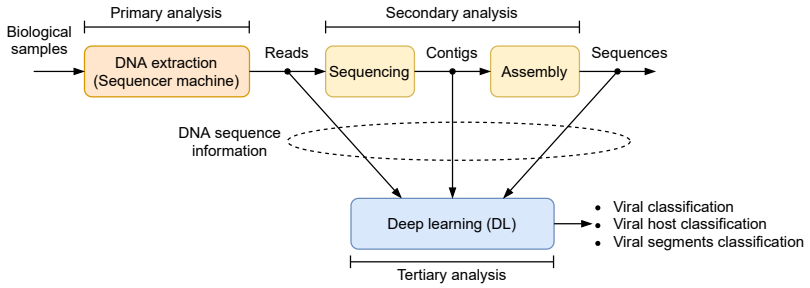


Fig. 1: Deep learning for virus tertiary analysis.

Fig. 2: Use of deep learning in viral genome classification.

classify species and genus of the virus. In [14], a hierarchical taxonomic classification for viral metagenomic data via DL, called CHEER, is proposed. Similar to the work proposed in [13], the CHEER framework classifies groups in the virus taxonomy structure such as genus, family, and order.

Works in [1], [3], [8] use contigs as DL input (see Figure 1) for viral prediction and classification. In [1], [3] a DL virus identification framework is proposed to recognize if the input is a virus or not. The proposal presented in [1], called ViraMiner, has a binary output (0 if is not a virus, and 1 if yes) and the proposal presented in [3], called DeepVirFinder, outputs a score between 0 and 1 for a binary classification between virus and prokaryote. The work presented in [8] differentiates phage, chromosomes, and plasmids, and the output also is a score.

Sub-type virus identification is proposed in [2] and [5]. In [2], a viral genome deep classifier uses a CNN to classify several sub-types virus, including HIV, Dengue, Influenza, and Hepatitis. In [5], a viral classification is proposed to identify a coronavirus sub-type, including SARS-CoV-2, MERS-CoV, SARS-CoV, and other viruses. The architecture of the CNNs used in this work is based in [2], [5].

Table I shows a summary of the viral classification references in terms of DNN structures. Note that DNNs applied to viral classification is a novel topic, and none of work in Table I uses model compression strategies yet in speeding up DNN computation and making it efficient, as proposed in this paper.

III. Materials and Methods

A. Data collection

In this study, a dataset consisting of the 1081 virus sequences was downloaded from the National Center for Biotechnology Information (NCBI) [26]. All downloaded sequences belong to three virus families (Alphasatellitidae, Anelloviridae, and Tolecusatellitidae), and the Riboviria realm. The length of the sequences used was between 1000 and 3500 base pair (bp). Table II shows a resume of the sequences used in this work, and Figure 4 presents the frequency of these sequences as a function of bp. The three virus families were chosen randomly from the Virus-Host DB families (available on August 14, 2020), and the Riboviria realm was chosen because it includes viruses groups of interest to this study such as Coronavirus, Ebola, HIV, Influenza, and the rabies virus.

TABLE I: Summary of the viral classification references in terms of DNN structures in N and L (see N base pairs and L neural network layers in Figure 3).

Reference	Number of sequences $\times 1000$	Input size (N)	Number of layers (L)
[2] (2019)	300	$24700 \times 1 \times 1$	25
[8] (2019)	20	$1600 \times 1 \times 4$	22
[1] (2019)	40	$300 \times 1 \times 5$	16
[5] (2020)	0.384	$32029 \times 1 \times 1$	5
[3] (2020)	1335	$3000 \times 1 \times 4$	6
[13] (2020)	10	$150 \times 100 \times 1$	10
[14] (2020)	60	$248 \times 100 \times 1$	12

TABLE II: Resume of the viruses sequences used in this work.

Name	No. seq.	Taxonomy	Min. seq. length	Max. seq. length
Alphasatellitidae	92	Family	1001	1479
Anelloviridae	81	Family	2002	3371
Riboviria	771	Realm	1010	3498
Tolecusatellitidae	137	Family	1012	1436

In order to balance the number of sequences for each class, 400 sequences from the Riboviria realm were chosen randomly, and for the other classes randomly selected sequences were duplicated until there are 400 sequences. After this procedure, a total of 1600 sequences were available (400 for each virus class).

B. DNN architecture

This work uses a CNN architecture described in Table III called the baseline architecture in this paper. Following proposals in literature [1], [3], [5], [7], [9], this work used the one-hot encoding (mapping stage) associated with CNN (processing stage) as presented in Figure 3.

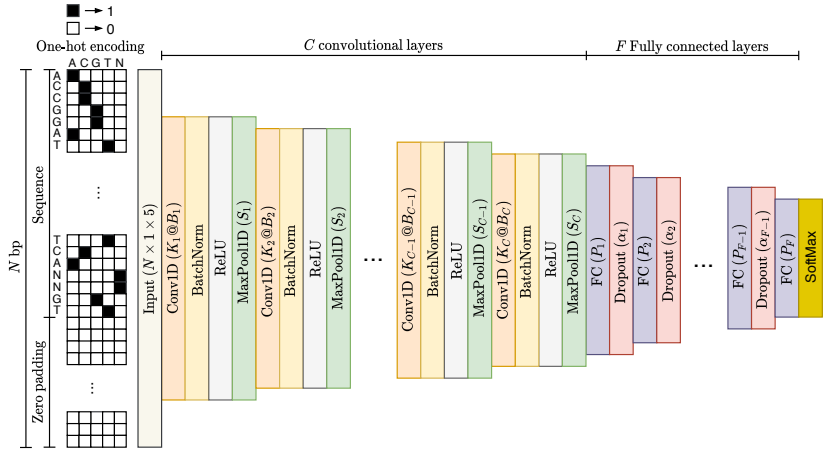


Fig. 3: One-hot encoding and CNN applied to a viral classification problem.

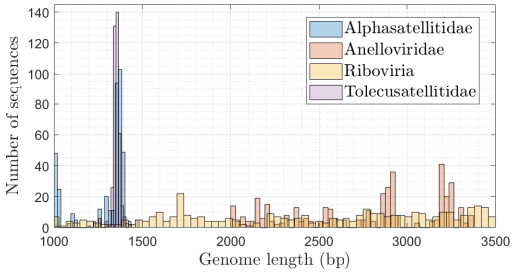


Fig. 4: Histogram of the number of the sequences.

The CNN input length, $N = 3500$, has been chosen to accommodate all input sequences presented in Table II. The zero padding was used for sequences of length less than 3500. The architecture size for single-precision float-point format (32 bits) is presented in Table IV. The number of operations (NOPs) of the CNN architecture, presented in Table III, has about 0.54 Giga Operations (GOPs). However, considering the bitwidth, there are about $0.539 \times 32 \approx 17.25$ Gigabit operations (GbOPs). The total size of weights for the architecture is about 46.62 Mbits. Note that for presentation simplicity the NOPs $\times$ bitwidth is only a linear proxy for the true float-point computation cost, i.e., it can be seen as a lower bound.

IV. Proposed Model Compression Scheme

Accelerating the execution of inference is crucial for bioinformatics applications, including viral classification. We note, however, that direct use of large DNN architectures may achieve only limited throughput due to excessive processing requirements [27], [28]. This limitation is reflected in numerous papers in the bioinformatic area reporting challenges and issues related to the DNN's high computational complexity [2], [8], [11].

Accelerating DL computation has been a recent focus in machine learning, and its literature has been gowning rapidly. The acceleration strategies for inference acceleration can be classified into algorithm-based and system-based. Algorithm-based acceleration approaches include parallel processing (data and model parallelism), model compression (pruning of weights, and quantization of weight and activation values), sparsity exploitation (both value-level and bit-level sparsity), and model reduction via approximation and learning [27], [29]–[35]. System-based acceleration approaches include high-

TABLE III: Architecture of the baseline CNN used in this paper with $C = 4$ convolutional layers and $F = 4$ fully connected layers (see Figure 3).

Layer	Description	Values
1	Input ($N \times 1 \times 5$)	$N = 3500$
2	Conv1D (K_1@B_1)	$K_1 = 256$ and $B_1 = 16$
3	BatchNorm	–
4	ReLU	–
5	MaxPool1D (S_1)	$S_1 = 2$
6	Conv1D (K_2@B_2)	$K_2 = 64$ and $B_2 = 32$
7	BatchNorm	–
8	ReLU	–
9	MaxPool1D (S_2)	$S_2 = 2$
10	Conv1D (K_3@B_3)	$K_3 = 32$ and $B_3 = 64$
11	BatchNorm	–
12	ReLU	–
13	MaxPool1D (S_3)	$S_3 = 2$
14	Conv1D (K_4@B_4)	$K_4 = 32$ and $B_4 = 128$
15	BatchNorm	–
16	ReLU	–
17	MaxPool1D (S_4)	$S_4 = 2$
18	FC (P_1)	$FP_1 = 64$
19	Dropout (α_1)	$\alpha_1 = 0.4$
20	FC (P_2)	$P_2 = 32$
21	Dropout (α_2)	$\alpha_2 = 0.4$
22	FC (P_3)	$P_3 = 16$
23	Dropout (α_3)	$\alpha_3 = 0.4$
24	FC (P_4)	$P_4 = 4$
25	SofMax	4 classes

TABLE IV: Architecture size of the baseline CNN in single-precision float-point format (32 bits).

NOPs (GOPs)	NOPs $\times$ bitwidth (GbOPs)	Weights-size (Mbits)
0.539	17.25	46.62

bandwidth memory systems for minimizing memory access time, hardware accelerators (ASIC and FPGA), distributed computing systems [28], [36], [37], and compiler optimizations such as commons expression elimination [38].

A. Model compression

The algorithm-based strategy using model compression, a focus of this work, is a common way to reduce the number of DNN operations and, at the same time, have a minimum impact to classification accuracy. The approach can improve the DNN performance in bioinformatics analysis for a lab facility that has only modest CPU/GPU computing resources.

Conventional compression approaches prune [29], [30], and quantize [27], [31] model weights, with pruning followed by quantization over training epochs [39], [40]. We denote such a training scheme as "$P \rightarrow Q$ epochs". In this paper, in contrast, we propose a novel model compression scheme that simultaneously minimizes both pruning and quantization losses during training with pruning followed by quantization ($P \rightarrow Q$ iteration) or quantization followed by pruning ($Q \rightarrow P$ iteration) for each training iteration.

B. Weight compression by quantization

Figure 5 depicts the architecture of quantization-aware training [27], [31]. In the schematic, n represents the n-th iteration, z^{-1} is a unitary delay, $\mathbf{W}(n)$ represent the CNN weights of the all layers, $\mathbf{X}(n)$ is the CNN input, $\mathbf{Y}_{ref}(n)$ is the desired output (label), $\mathbf{Y}(n)$ is CNN output, $\mathbf{E}(n)$ is the an error metric (loss) between $\mathbf{Y}(n)$ and $\mathbf{Y}_{ref}(n)$, and $\mathbf{G}(n)$ represents the gradient metric of the all layers. In every mini-batch a copy of weights, $\mathbf{W}(n)$, is quantized and passed to the forward stage through of the variable $\mathbf{C}(n)$.

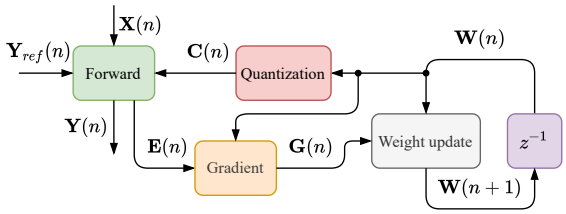


Fig. 5: Quantization-aware training.

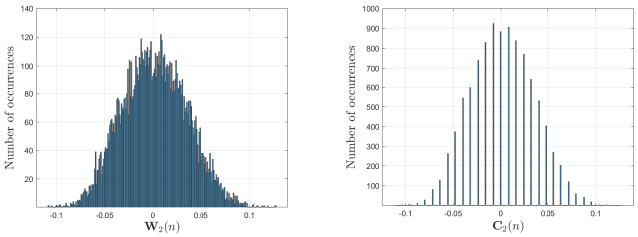
The compression by quantization produces low bitwidth integers for representing weights. This is, $\mathbf{W}(n)$ is represented by one of M discrete possible levels that can be represented by b bits. There are many quantization techniques in literature, as presented in [27], [31], [34], [35]. A common scheme is uniform quantization, with $M = 2^b - 1$. In this case, the variable $\mathbf{C}(n)$ of the each k-th layer is expressed as

$$\mathbf{C}_k(n) = Q\left(\mathbf{W}_k(n), q_k\right) = \left\lceil \frac{\mathbf{W}_k(n)}{q_k} \right\rfloor \times q_k \qquad (1)$$

where $Q(\cdot,\cdot)$ is the quantization function, $\lceil \cdot \rfloor$ is the round operation, and q_k, the scale factor, can be expressed as

$$q_k = \frac{\max\left\{|\mathbf{W}_k(n)|\right\}}{2^{b-1}-1}. \qquad (2)$$

Figures 6a, and 6b illustrate an example of the weights of the second layer, $\mathbf{W}_2(n)$, of the baseline CNN detailed in Table III. The weights are presented before and after quantization with $b = 5$ ($M = 31$), respectively.



(a) Before quantization. (b) After quantization.

Fig. 6: Example of the weights compression by quantization with $b = 5$ ($M = 31$) applied to the second layer, $\mathbf{W}_2(n)$, of the baseline CNN detailed in Table III.

C. Weight compression by pruning

In pruning (see Figure 7), the weights for each k-th layer, $\mathbf{W}_k(n)$, with small values (below a threshold) are reset to zero, that is,

$$\mathbf{C}_k(n) = P\left(\mathbf{W}_k(n), \beta_k\right) = \begin{cases} \mathbf{W}_k(n) & \text{if } |\mathbf{W}_k(n)| \geq \beta_k \\ 0 & \text{if } |\mathbf{W}_k(n)| < \beta_k \end{cases} \qquad (3)$$

where $P(\cdot,\cdot)$ is the pruning function, and β_k is the pruning cut-off threshold of the k-th layer. Figures 8a and 8b illustrate an example of the weights of the second layer (first convolutional layer), $\mathbf{W}_2(n)$, of the baseline CNN detailed in Table III. The weights are presented before and after pruning, respectively, with the pruning cut-off threshold, $\beta_k = 0.5 \times \sigma_k$. The variable σ_k represents the standard deviation of the $\mathbf{W}_2(n)$.

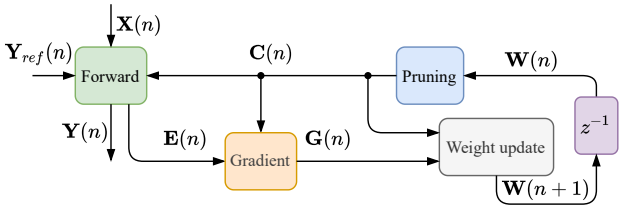


Fig. 7: Pruning-aware training.

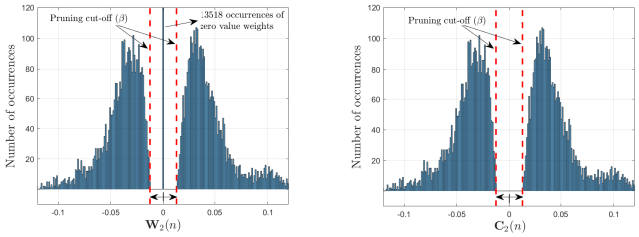
D. Proposed pruning followed by quantization iteration ($P \rightarrow Q$ iteration)

The compression proposed by this work based on pruning followed by quantization iteration, $P \rightarrow Q$ iteration, can be expressed as

$$\mathbf{C}_k(n) = Q\left(P\left(\mathbf{W}_k(n), \beta\right), q'_k\right) \qquad (4)$$

where

$$q'_k = \frac{\max\left\{|\mathbf{W}_k(n)|\right\} - \beta_k}{2^{b-1}-1}. \qquad (5)$$



(a) Weights before pruning. (b) Weights after pruning.

Fig. 8: Example of the weights compression by pruning applied to the second layer, $\mathbf{W}_2(n)$, of the baseline CNN detailed in Table III ($\beta_k = 0.5 \times \sigma_k$).

The diagram presented in Figure 9 illustrates $P \rightarrow Q$ iteration, and Figures 10a, and 10b show an example of the weights of the second layer, $\mathbf{W}_2(n)$, of the baseline CNN detailed in Table III. The weights are presented before and after $P \rightarrow Q$ iteration, respectively.

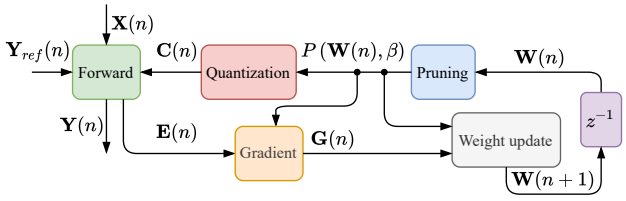
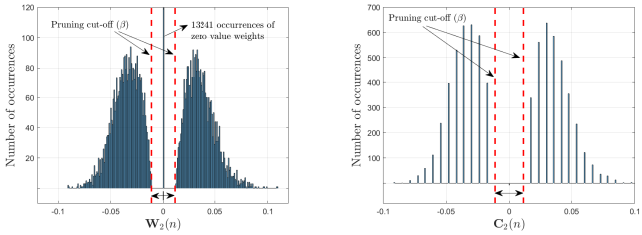


Fig. 9: Pruning followed by quantization iteration, $P \rightarrow Q$ iteration, architecture.



(a) Weights before $P \rightarrow Q$ iteration. (b) Weights after $P \rightarrow Q$ iteration.

Fig. 10: Example of the weights compression by $P \rightarrow Q$ iteration applied to the second layer, $\mathbf{W}_2(n)$, of the baseline CNN detailed in Table III. We assume, $\beta_k = 0.5 \times \sigma_k$ and $b = 5$ ($M = 31$).

E. Proposed quantization followed by pruning iteration ($Q \rightarrow P$ iteration)

Figure 11 shows the quantization followed by pruning iteration, $Q \rightarrow P$ iteration, architecture proposed in this work. In each mini-batch, a copy of weight values is quantized using Equation 1, and this information is passed to the forward stage (see Figure 11). Based on the error metric (loss), $\mathbf{E}'(n)$, the new gradient, $\mathbf{G}'(n)$, is computed and sent to the weight update stage.

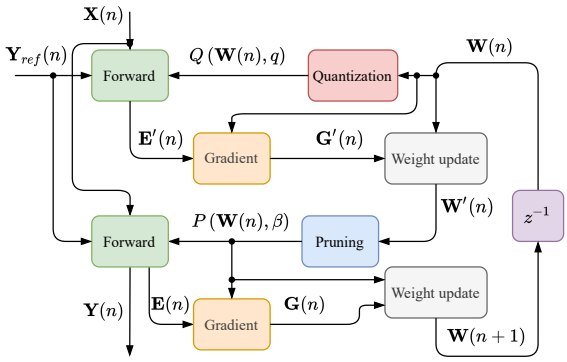


Fig. 11: Quantization followed by pruning iteration, $Q \rightarrow P$ iteration, architecture.

After quantization, the quantized weight values are pruned using Equation 3 and a new gradient, $\mathbf{G}(n)$, is calculated in the same mini-batch. The new gradient, $\mathbf{G}(n)$, is computed using the new error metric (loss), $\mathbf{E}(n)$. The $Q \rightarrow P$ iteration weights values sent to the forward stage for k-th layer can be expressed as

$$\mathbf{C}_k(n) = P\left(Q\left(\mathbf{W}_k(n), q_k\right), \beta_k\right). \tag{6}$$

With the new gradient, $\mathbf{G}(n)$, the weights are updated again in the same mini-batch. Figures 12a and 12b show an example of the weights compression, $Q \rightarrow P$ iteration, of the second layer, $\mathbf{W}_2(n)$, of the baseline CNN detailed in Table III. The weights are presented before and after $Q \rightarrow P$ iteration with pruning cut-off threshold, $\beta_k = 0.5 \times \sigma_k$, and $b = 5$ ($M = 31$).

(a) Weights before $Q \rightarrow P$ iteration. (b) Weights after $Q \rightarrow P$ iteration.

Fig. 12: Example of the weights compression by $Q \rightarrow P$ iteration applied to the second layer, $\mathbf{W}_2(n)$, of the baseline CNN detailed in Table III. We assume, $\beta_k = 0.5 \times \sigma_k$ and $b = 5$ ($M = 31$).

V. Results and Discussion

In order to show the advantages of the quantization followed by pruning in viral classification, we demonstrate several results with $b = 8$ bits quantization ($M = 254$) for different values of cut-off threshold, β_k, where $\beta_k = \gamma \times \sigma_k$ for some parameter γ. Use of a larger γ means more aggressive pruning. The results were collected using the CNN presented in Figure 3 and the training strategies illustrated in Figures 9, and 11 for $P \rightarrow Q$ iteration and $Q \rightarrow P$ iteration, respectively.

These strategies were applied to classify the virus presented in Table II, with 80% of the sequences (1280 sequences) for training and 20% (320 sequences) for validation. The ratio (80% : 20%) was maintained for each class. These training experiments used mini-batches of size 64, and 35 epochs.

Figures 13 and 14 show the density of all CNN weights, $\mathbf{C}(n)$, versus validation accuracy for several values of weights-size (and γ), and NOPs (and γ), respectively. The density was calculated as the number of the non-zero weights (in all layers) over the number of weights (in all layers).

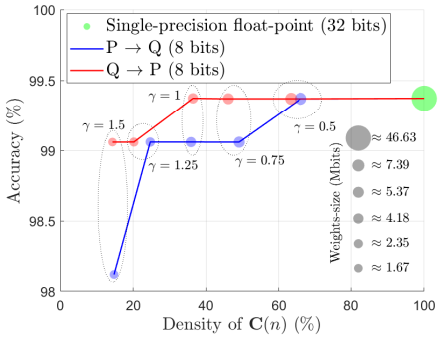


Fig. 13: Overall validation accuracy (for classes presented in Table II) as a function of density of the CNN weights, $\mathbf{C}(n)$, for several values of weights-size (and γ), with the density being the fraction of nonzero weights in $\mathbf{W}(n)$ remaining. Lower densities mean more aggressive model compression. We note that $Q \rightarrow P$ iteration achieves higher accuracy than $P \rightarrow Q$ iteration at similar densities.

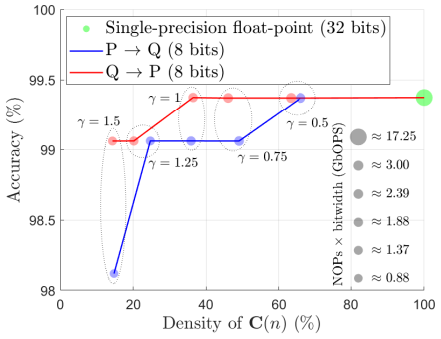


Fig. 14: Overall validation accuracy (for classes presented in Table II) as a function of density of the CNN weights, $\mathbf{C}(n)$, for several values of NOPs (and γ).

As presented in figures 13 and 14, the value of the cut-off threshold β_k (or γ) impacts in the density reduction. Note that $Q \rightarrow P$ iteration compression can maintain the accuracy above 99% to low density values ($< 20\%$), compared to the $P \rightarrow Q$ iteration strategy. As illustrated in the graphs presented in Figures 13 and 14, the $Q \rightarrow P$ iteration strategy, as opposed to $P \rightarrow Q$ iteration, can maintain the accuracy above 99% for a pruning cut-off threshold with $\gamma = 1.5$.

Figures 15 and 16 show the CNN weights-size (in Mbits), and the CNN NOPs $\times$ bitwidth (in GbOPs) versus validation accuracy for several values of CNN weights density $\mathbf{C}(n)$, respectively. The graphs illustrated in Figures 15 and 16 show that the accuracy can be maintained even under severe compression.

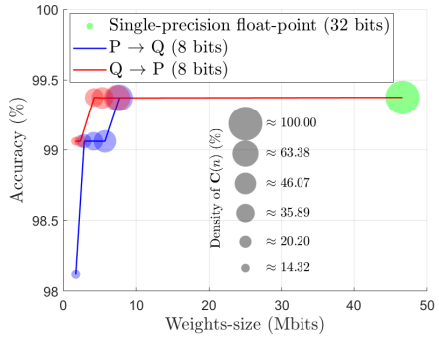


Fig. 15: Overall validation accuracy (for classes presented in Table II) as a function of CNN weights-size (in Mbits), for several values of CNN weights density $\mathbf{C}(n)$.

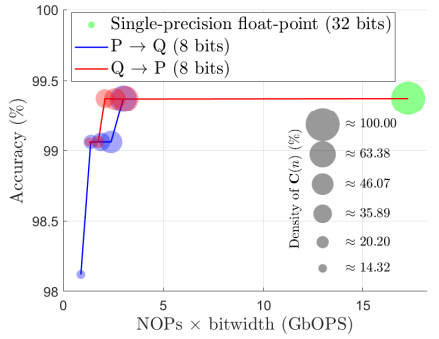


Fig. 16: Overall validation accuracy (for classes presented in Table II) as a function of NOPs $\times$ bitwidth (in GbOPs), for several values of CNN weights density $\mathbf{C}(n)$.

Figures 17, 18 and 19 show the curves of the density (in %), weights-size (in Mbps), and NOPs (in GbOPs) as a function of pruning cut-off threshold ($\beta_k = \gamma \times \sigma_k$), just for accuracy above 99%. $Q \rightarrow P$ iteration archives the higher compression than $P \rightarrow Q$ iteration. Table V presents a summary of the results presented in Figures 17, 18 and 19.

Based on the values presented in Table V, the highest compression for the $P \rightarrow Q$ iteration strategy is about 62.6% for density and weights-size, and 54.3% for NOPs. For the $Q \rightarrow P$ iteration strategy they are about 77.4% for density and weights-size, and 58.4% for NOPs. The difference between weights-size and NOPs reflects the architecture of the DNN (number of convolutional and fully connected layers).

Regarding DL-based viral classifiers, Figure 20 shows the confusion matrix for the results with the highest compression $Q \rightarrow P$ iteration strategy. The result indicates that a significant reduction in DNN parameters (more than 77% for $Q \rightarrow P$

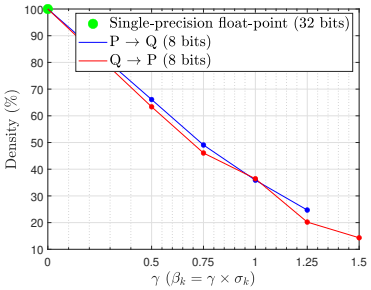


Fig. 17: Density (in %) as a function of pruning cut-off threshold ($\beta_k = \gamma \times \sigma_k$).

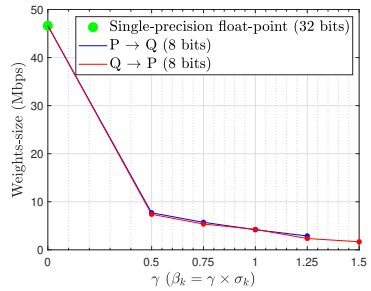


Fig. 18: Weights-size (in Mbps) as a function of pruning cut-off threshold ($\beta_k = \gamma \times \sigma_k$).

iteration) and NOPs (more than 58% for $Q \rightarrow P$ iteration) can be achieved, and, at the same time, keeping up the accuracy.

Riboviria is a realm (set of several families), and this permits a high genetic diversity. Note that differently from the other classes (Alphasatellitidae, Anelloviridae, and Tolecusatellitidae), Riboviria has samples with all sequence length range, i.e., the sequence length between 1010 and 3498 bp (see Table II and Figure 5). This genetic diversity associated with Ribovira explains the false positive in confusion matrix presented in Figure 20.

It can be seen from the results that the strategy proposed here can help significantly in areas associated with the analysis of viral sequences in real-time such as metagenomics, diagnosis of viral infections, pharmacogenomics, and others. Based on works presented in the literature, such as those presented in [2], [5], [9], the structure proposed here can also be adapted for longer viral sequences.

VI. Comparison with Conventional Approach

In this section, we compare our proposed $Q \rightarrow P$ iteration scheme with the conventional $P \rightarrow Q$ epochs scheme [39], [40]. Figure 21 shows the block diagram of the conventional $P \rightarrow Q$ epochs compression approach. Figure 22 shows the training time control, where switches SW_1 and SW_2 are defined in Figure 21. Suppose that there are G epochs and I iterations per epoch. As presented in Figure 22, the pruning (P) and quantization (Q) actions apply to the first and second

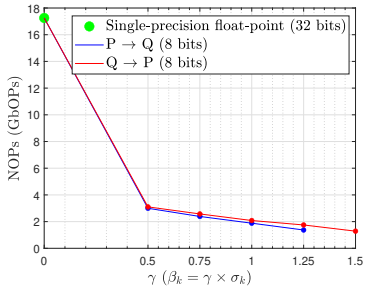


Fig. 19: NOPs (in GbOPs) as a function of pruning cut-off threshold ($\beta_k = \gamma \times \sigma_k$).

TABLE V: Results with $b = 8$ bits quantization ($M = 254$) for different values of cut-off threshold, where $\beta_k = \gamma \times \sigma_k$.

Comp. method	γ	Density (%)	NOPs × bitwidth (GbOPs)	Weights-size (Mbits)
$P \rightarrow Q$	0.50	66.10	3.00	7.71
	0.75	49.07	2.38	5.72
	1.00	35.89	1.88	4.18
	1.25	24.70	1.37	2.88
$Q \rightarrow P$	0.50	63.38	3.10	7.39
	0.75	47.07	2.58	5.37
	1.00	36.44	2.08	4.25
	1.25	20.20	1.75	2.35
	1.50	14.32	1.29	1.67

halves of epochs, respectively:

$$\mathrm{SW}_2 = \begin{cases} \mathrm{A} & \text{for epoch} = 1 \ldots G/2 \\ \mathrm{B} & \text{for epoch} = G/2 + 1 \ldots G. \end{cases} \tag{7}$$

For each epoch, pruning or quantization action is enabled just the first iteration. For example, for the first half of epochs, SW_1 connected to the the pruning blocking has the following settings:

$$\mathrm{SW}_1 = \begin{cases} \mathrm{A} & \text{for iteration} = 1 \\ \mathrm{B} & \text{for iteration} = 2 \ldots I. \end{cases} \tag{8}$$

In contrast, Figure 23 shows the training time control associated with our proposed $P \rightarrow Q$ iteration and $Q \rightarrow P$ iteration training schemes.

Figure 24 illustrates an example of the conventional weights compression scheme, $P \rightarrow Q$ epochs, for the second layer, $\mathbf{W}_2(n)$, of the baseline CNN detailed in Table III. The weights are presented before and after $P \rightarrow Q$ epochs with pruning cut-off threshold, $\beta_k = 0.5 \times \sigma_k$ and quantization settings $b = 5$ ($M = 31$).

We compare the values of the weights distributions before the pruning between the conventional $P \rightarrow Q$ epochs scheme (see Figure 24a) and our proposed $Q \rightarrow P$ iteration scheme (see Figure 12a). We note that the conventional scheme ($P \rightarrow Q$ epochs) has many weights values spread inside the cut-off range. Differently, our proposal ($Q \rightarrow P$ iteration) concentrates all weights values at zero.

Figure 25 shows the confusion matrices for the results related to the conventional $P \rightarrow Q$ epochs scheme with $b = 8$

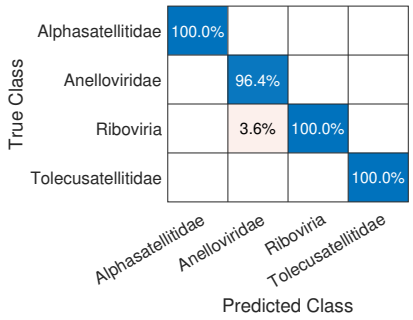


Fig. 20: Confusion matrix associated with the proposed $Q \rightarrow P$ iteration strategy of this paper with $b = 8$ ($M = 254$) and $\gamma = 1.5$, where $\beta_k = \gamma \times \sigma_k$.

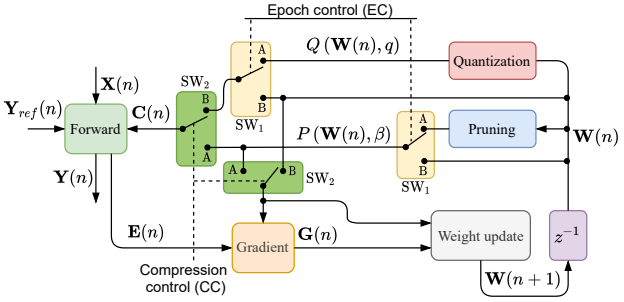


Fig. 21: Conventional scheme of pruning epochs followed by quantization epochs, $P \rightarrow Q$ epochs, architecture.

($M = 254$) and $\gamma = 1.25$, where $\beta_k = \gamma \times \sigma_k$. We compare this result with the result shown in Figure 20, using the same conditions and parameters. We note that the approach proposed in this paper is superior to conventional compression strategy.

After we had designed our proposed compression schemes, $P \rightarrow Q$ iteration and $Q \rightarrow P$ iteration, we became aware of the CLIP-Q training method presented in [33]. The CLIP-Q paper presents a scheme similar to our $P \rightarrow Q$ iteration, but it does not consider schemes similar to our $Q \rightarrow P$ iteration. For the viral classification tasks of this paper, we demonstrate that $Q \rightarrow P$ iteration has superior performance to $P \rightarrow Q$ iteration (see Figures 13, 14, 15, and 16, Table V).

VII. Conclusion

We have proposed the $Q \rightarrow P$ iteration training scheme, where quantization is followed by pruning per training iteration, for training compressed models in accelerating deep learning inference for viral classification. The proposed strategy was applied to 1600 sequences of four groups of viruses associated with three families and one realm. We have empirically demostrated that $Q \rightarrow P$ iteration yields a more efficient compressed model than $P \rightarrow Q$ iteration, per training iteration for the viral classfication tasks considered. Our proposed $Q \rightarrow P$ iteration strategy is superior in prunning performance to conventional $P \rightarrow Q$ epochs strategy, pruning epochs followed by quantization epochs. The results were obtained for an 8-bit quantized DNN compared to a single-precision float-point (32-bit) DNN. As future work, we plan to extend these results to other classification tasks.

Acknowledgment

Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) for their financial support by CAPES-Print-UFRN program and the finance code 001. The work was also supported in part by the Air Force Research Laboratory under award number FA8750-18-1-0112 and by a gift from MediaTek USA.

References

[1] A. Tampuu, Z. Bzhalava, J. Dillner, and R. Vicente, "Viraminer: Deep learning on raw dna sequences for identifying viral genomes in human samples," *PloS one*, vol. 14, no. 9, p. e0222271, 2019.

[2] A. Fabijańska and S. Grabowski, "Viral genome deep classifier," *IEEE Access*, vol. 7, pp. 81 297–81 307, 2019.

[3] J. Ren, K. Song, C. Deng, N. A. Ahlgren, J. A. Fuhrman, Y. Li, X. Xie, R. Poplin, and F. Sun, "Identifying viruses from metagenomic data using deep learning," *Quantitative Biology*, pp. 1–14, 2020.

[4] J. A. Morales, R. Saldaña, M. H. Santana-Castolo, C. E. Torres-Cerna, E. Borrayo, A. P. Mendizabal-Ruiz, H. A. Vélez-Pérez, and G. Mendizabal-Ruiz, "Deep learning for the classification of genomic signals," *Mathematical Problems in Engineering*, vol. 2020, 2020.

[5] A. Lopez-Rincon, A. Tonda, L. Mendoza-Maldonado, E. Claassen, J. Garssen, and A. D. Kraneveld, "Accurate identification of sars-cov-2 from viral genome sequences using deep learning," *bioRxiv*, 2020.

[6] H. Zhu, Q. Guo, M. Li, C. Wang, Z. Fang, P. Wang, J. Tan, S. Wu, and Y. Xiao, "Host and infectivity prediction of wuhan 2019 novel coronavirus using deep learning algorithm," *BioRxiv*, 2020.

[7] F. Mock, A. Viehweger, E. Barth, and M. Marz, "Viral host prediction with deep learning," *bioRxiv*, p. 575571, 2019.

[8] Z. Fang, J. Tan, S. Wu, M. Li, C. Xu, Z. Xie, and H. Zhu, "Ppr-meta: a tool for identifying phages and plasmids from metagenomic fragments using deep learning," *GigaScience*, vol. 8, no. 6, p. giz066, 2019.

[9] J. M. Bartoszewicz, A. Seidel, and B. Y. Renard, "Interpretable detection of novel human viruses from genome sequencing data," *bioRxiv*, 2020.

[10] C. Pian, Z. Li, H. Jiang, L. Kong, Y. Chen, and L. Zhang, "Deep6ma: a deep learning framework for exploring similar patterns in dna n6-methyladenine sites across different species," *bioRxiv*, 2019.

[11] S. Kuang and L. Wang, "Identification and analysis of consensus rna motifs binding to the genome regulator ctcf," *NAR Genomics and Bioinformatics*, vol. 2, no. 2, p. lqaa031, 2020.

[12] Y. Zhang, S. Qiao, S. Ji, and Y. Li, "Deepsite: bidirectional lstm and cnn models for predicting dna–protein binding," *International Journal of Machine Learning and Cybernetics*, pp. 1–11, 2019.

[13] Q. Liang, P. W. Bible, Y. Liu, B. Zou, and L. Wei, "Deepmicrobes: taxonomic classification for metagenomics with deep learning," *NAR Genomics and Bioinformatics*, vol. 2, no. 1, p. lqaa009, 2020.

[14] J. Shang and Y. Sun, "Cheer: hierarchical taxonomic classification for viral metagenomic data via deep learning," *Methods*, 2020.

[15] L. Pinello, G. Lo Bosco, and G.-C. Yuan, "Applications of alignment-free methods in epigenomics," *Briefings in Bioinformatics*, vol. 15, no. 3, pp. 419–430, 11 2013. [Online]. Available: https://doi.org/10.1093/bib/bbt078

[16] S. Vinga and J. Almeida, "Alignment-free sequence comparison—a review," *Bioinformatics*, vol. 19, no. 4, pp. 513–523, 03 2003. [Online]. Available: https://doi.org/10.1093/bioinformatics/btg005

[17] G. Mendizabal-Ruiz, I. Román-Godínez, S. Torres-Ramos, R. A. Salido-Ruiz, and J. A. Morales, "On dna numerical representations for genomic similarity computation," *PloS one*, vol. 12, no. 3, p. e0173288, 2017.

[18] G. S. Randhawa, K. A. Hill, and L. Kari, "Ml-dsp: Machine learning with digital signal processing for ultrafast, accurate, and scalable genome classification at all taxonomic levels," *BMC genomics*, vol. 20, no. 1, p. 267, 2019.

[19] G. S. Randhawa, M. P. Soltysiak, H. El Roz, C. P. de Souza, K. A. Hill, and L. Kari, "Machine learning using intrinsic genomic signatures for rapid classification of novel pathogens: Covid-19 case study," *Plos one*, vol. 15, no. 4, p. e0232391, 2020.

[20] R. de M. Barbosa and M. A. Fernandes, "Chaos game representation dataset of sars-cov-2 genome," *Data in Brief*, vol. 30, p. 105618, 2020.

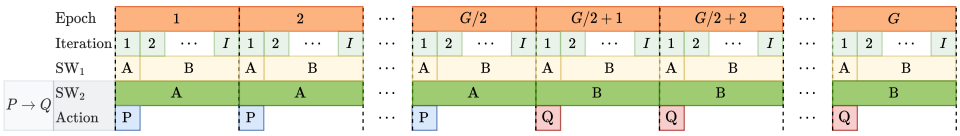


Fig. 22: Conventional $P \rightarrow Q$ epochs training scheme: pruning epochs followed by quantization epochs.

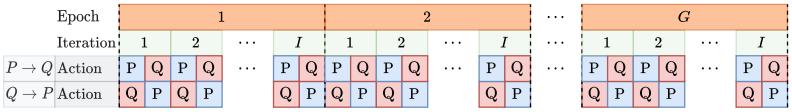
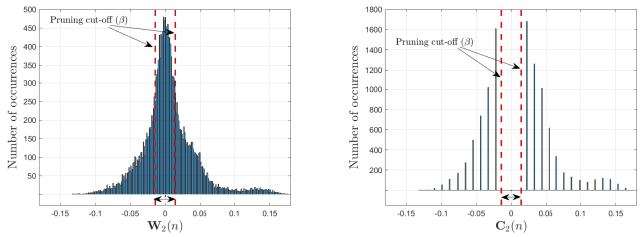


Fig. 23: Proposed $P \rightarrow Q$ and $Q \rightarrow P$ iteration training schemes.



(a) Weights before $P \rightarrow Q$ epochs.

(b) Weights after $P \rightarrow Q$ epochs.

Fig. 24: Example of the weights conventional $P \rightarrow Q$ epochs compression applied to the second layer, $\mathbf{W}_2(n)$, of the baseline CNN detailed in Table III. We use $\beta_k = 0.5 \times \sigma_k$ and $b = 5$ ($M = 31$).

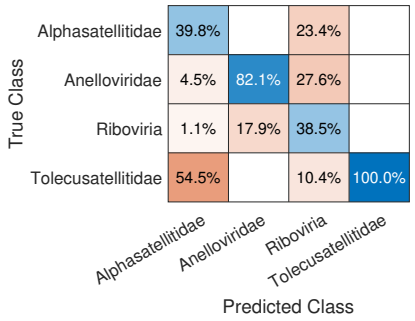


Fig. 25: Confusion matrix associated with conventional $P \rightarrow Q$ epochs strategy with $b = 8$ ($M = 254$) and $\gamma = 1.25$, where $\beta_k = \gamma \times \sigma_k$.

[21] ——, "Data stream dataset of sars-cov-2 genome," *Data in Brief*, p. 105829, 2020.

[22] T. T.-Y. Lam, M. H.-H. Shum, H.-C. Zhu, Y.-G. Tong, X.-B. Ni, Y.-S. Liao, W. Wei, W. Y.-M. Cheung, W.-J. Li, L.-F. Li *et al.*, "Identifying sars-cov-2 related coronaviruses in malayan pangolins," *Nature*, pp. 1–6, 2020.

[23] K. G. Andersen, A. Rambaut, W. I. Lipkin, E. C. Holmes, and R. F. Garry, "The proximal origin of sars-cov-2," *Nature medicine*, vol. 26, no. 4, pp. 450–452, 2020.

[24] R. L. Graham and R. S. Baric, "Sars-cov-2: Combating coronavirus emergence," *Immunity*, 2020.

[25] C. Charre, C. Ginevra, M. Sabatier, H. Regue, G. Destras, S. Brun, G. Burfin, C. Scholtes, F. Morfin, M. Valette, B. Lina, A. Bal, and L. Josset, "Evaluation of NGS-based approaches for SARS-CoV-2 whole genome characterisation," *Virus Evolution*, vol. 6, no. 2, 10 2020.

[26] National Center for Biotechnology Information (NCBI). (2021) Bethesda (md): National library of medicine (us), national center for biotechnology information. [Online]. Available: https://www.ncbi.nlm.nih.gov/

[27] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "Haq: Hardware-aware automated quantization with mixed precision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2019, pp. 8612–8620.

[28] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang, "A survey of fpga-based neural network accelerator," *arXiv preprint arXiv:1712.08934*, 2017.

[29] D. Blalock, J. J. G. Ortiz, J. Frankle, and J. Guttag, "What is the state of neural network pruning?" *arXiv preprint arXiv:2003.03033*, 2020.

[30] Q. Huang, K. Zhou, S. You, and U. Neumann, "Learning to prune filters in convolutional neural networks," in *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 2018, pp. 709–718.

[31] A. S. Rakin, J. Yi, B. Gong, and D. Fan, "Defend deep neural networks against adversarial examples via fixed and dynamic quantized activation functions," *arXiv preprint arXiv:1807.06714*, 2018.

[32] S. Jung, C. Son, S. Lee, J. Son, J.-J. Han, Y. Kwak, S. J. Hwang, and C. Choi, "Learning to quantize deep networks by optimizing quantization intervals with task loss," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4350–4359.

[33] F. Tung and G. Mori, "Deep neural network compression by in-parallel pruning-quantization," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 3, pp. 568–579, 2020.

[34] W. Zhe, J. Lin, V. Chandrasekhar, and B. Girod, "Optimizing the bit allocation for compression of weights and activations of deep neural networks," in *2019 IEEE International Conference on Image Processing (ICIP)*, 2019, pp. 3826–3830.

[35] H. Kung, B. McDanel, and S. Q. Zhang, "Term revealing: Furthering quantization at run time on quantized dnns," *arXiv preprint arXiv:2007.06389*, 2020.

[36] M. Imani, M. Samragh Razlighi, Y. Kim, S. Gupta, F. Koushanfar, and T. Rosing, "Deep learning acceleration with neuron-to-memory transformation," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2020, pp. 1–14.

[37] M. G. F. Coutinho, M. F. Torquato, and M. A. C. Fernandes, "Deep neural network hardware implementation based on stacked sparse autoencoder," *IEEE Access*, vol. 7, pp. 40 674–40 694, 2019.

[38] Y. Kim, Q. Tong, K. Choi, E. Lee, S. Jang, and B. Choi, "System level power reduction for yolo2 sub-modules for object detection of future autonomous vehicles," in *2018 International SoC Design Conference (ISOCC)*, 2018, pp. 151–155.

[39] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[40] S. Jin, S. Di, X. Liang, J. Tian, D. Tao, and F. Cappello, "Deepsz: A novel framework to compress deep neural networks by using error-bounded lossy compression," in *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing*, 2019, pp. 159–170.