Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai

Single-shot pruning and quantization for hardware-friendly neural network acceleration

Bofeng Jiang[1], Jun Chen[1,*], Yong Liu[*]

Institute of Cyber-Systems and Control, Zhejiang University, China

ARTICLE INFO

Keywords:
Pruning
Quantization
Hardware-friendly
CNN acceleration

ABSTRACT

Applying CNN on embedded systems is challenging due to model size limitations. Pruning and quantization can help, but are time-consuming to apply separately. Our Single-Shot Pruning and Quantization strategy addresses these issues by quantizing and pruning in a single process. We evaluated our method on CIFAR-10 and CIFAR-100 datasets for image classification. Our model is 69.4% smaller with little accuracy loss, and runs 6–8 times faster on NVIDIA Xavier NX hardware.

1. Introduction

Convolutional Neural Networks (CNNs) are currently widely used in computer vision. The widespread use of CNNs is due to the outstanding computational performance and the growth of large-scale datasets. The performance of CNNs is increasing, and the recognition error rate is decreasing, but at the same time, their spatial and temporal complexity is increasing; the number of parameters and the number of computational operations during network training are also increasing. For example, VGG-16 has up to 138 million parameters, and its overall model size is over 500 M (Simonyan and Zisserman, 2014). It requires 15.5 billion floating-point operations to classify a single image. Moreover, the introduction of ResNet (He et al., 2016) solved the problem of degradation that occurs when the model depth is increased, thus raising the parameter and computation levels of the model to unprecedented heights. Furthermore, as the number of parameters increases, the cost of both CNN training and inference is rising. For high-performance inference devices such as GPUs, this is not a difficult task, but for inference platforms with limited resources, high computational costs and high performance requirements make performing visual tasks difficult. To address this issue, one possible solution is to reduce the cost of training and inference. By implementing techniques that can lower the cost of these processes, we can improve the efficiency and effectiveness of CNNs, which is good to energy structure on carbon emissions and energy storage (Zhang et al., 2023; Licheng and Wang, 2022; Yu et al., 2023).

At the same time, the Internet of Things (IoT) development allows small models to extend deep learning to a more extensive application space. The need for image classification, target detection and OCR (optical character recognition) text recognition algorithms in robots, drones, and other mobile embedded devices is constantly on the rise, necessitating algorithms that are highly accurate and have minimal latency. Researchers strive to meet these demands by continuously delving into cutting-edge technologies and exploring novel methodologies. Their aim is to develop more sophisticated algorithms that allow mobile devices to operate more intelligently and serve people better. Moreover, the compression of neural networks holds a crucial significance in enhancing the cognitive (Li et al., 2022) as well as perceptual facets (Li et al., 2023) of computer vision. How to compress and employ CNN models on embedded devices without compromising accuracy has become a frontier hotspot for network structure optimization.

Quantization and pruning are commonly used to reduce the number of model parameters and computation operations. Quantization is to convert parameters from 32-bit floating point numbers to 16-bit floating point numbers or 8-bit integer to reduce memory occupied by parameters and operation run-time (Jacob et al., 2018). Pruning is adding a judging mechanism to the network training process, eliminating unimportant connections, filters, and layers, to achieve the purpose of streamlining the network structure (Han et al., 2015a)

Current pruning methods are divided into weight pruning, channel pruning, and inter-layer pruning according to granularity (Guo et al., 2016). Weight pruning is a kind of sparse pruning. The convolutional kernel obtained from pruning has unstructured characteristics. However, this unstructured convolutional kernel requires a particular hardware configuration to hit the acceleration effect. Inter-layer pruning reduces the network's depth, effectively reducing the number of network parameters, but the performance degradation is very problematic (Anwar and Sung, 2016). The performance of the model and

* Corresponding authors.
E-mail addresses: 22160068@zju.edu.cn (B. Jiang), junc@zju.edu.cn (J. Chen), yongliu@iipc.zju.edu.cn (Y. Liu).
[1] Bofeng Jiang and Jun Chen contributed equally to this work.

https://doi.org/10.1016/j.engappai.2023.106816
Received 4 January 2023; Received in revised form 3 June 2023; Accepted 13 July 2023
Available online 27 August 2023

the number of parameters are well-balanced through filter pruning. Additionally, filter pruning is also known as structured pruning, which is easy to use on embedded systems (Wen et al., 2016).

As model compression techniques, pruning and quantization are now explored separately. Some people combine them (Han et al., 2015a). However, when they are done individually, pruning and quantization result in a considerable loss in model accuracy since the model is incapable of learning from and correcting any quantization errors promptly. Likewise, quantization and pruning are carried out independently, which is difficult and consumes much training time. Our goal is to appropriately combine pruning and quantization so that they may be accomplished in a single training phase and that the gradient update can be used to lessen the accuracy loss brought on by quantization error and pruning during the iterative process.

We present Single-Shot Pruning and Quantization, which can accomplish linear quantization and two kinds of pruning methods, norm pruning and centroid pruning, in one training procedure. For the quantization and pruning errors to be updated simultaneously when the parameters are updated, the training procedure employs the "soft pruning" (He et al., 2018) method, in which the pruned channels are set to zero. Additionally, we carried out tests and successfully applied the concept to an embedded system. In this work, we make the following contributions:

- We propose the Single-Shot Pruning and Quantization method, which can quantize and prune the model in one training process. Our method successfully enables us to take into account the quantization error and pruning mistake during the deep learning network training and update the weights under these flaws.
- We construct a model, prune all the zeroed filters, and then modify it to be hardware-adaptive. In terms of training time, we conduct a comparative analysis between our approach and the quantized after pruning methods, and find that our technique achieved a remarkable 20%–25% reduction in training time.
- We train our MobileNetV2 in CIFAR-10 and CIFAR-100 from scratch and test the mean inference time and Multiply–Accumulate Operations (MACs)to evaluate the performance of the compressed model. We accelerate MobileNetV2 on NVIDIA Xavier NX by six to eight times with about 2% relative accuracy drop on CIFAR-10 and CIFAR-100.

2. Related work

Nowadays, the most popular deep neural network compression techniques can be divided into four types: quantization (Hubara et al., 2016; Zhao et al., 2017; Zhou et al., 2017; Chen et al., 2023a,b, 2021), network pruning (Chen et al., 2015; Han et al., 2015b; Mao et al., 2017), knowledge distillation (Ba and Caruana, 2014; Hinton et al., 2015; Liu et al., 2023) and low-rank factorization (Denton et al., 2014; Sainath et al., 2013). Low-rank factorization is a powerful technique that exploits matrix decomposition to accurately ascertain the parameters contained in convolution filters, thereby decreasing the computational requirements of deep neutral networks and liberating memory resources (Sainath et al., 2013). Knowledge distillation is a network structure optimization strategy that involves transferring knowledge from the teacher's network to the student's network to guide the training of the student's network and compress and speed the network (Hinton et al., 2015). Quantization is a process that decreases the width of the weight in the filters (Fiesler et al., 1990). It maps the float values with 32-bit to 16-bit float numbers or 8-bit or fewer width integers, meaning that the output consists of a smaller range of values than the input, meanwhile without too much accuracy drop in the process. In order to simplify the network structure, network pruning is a judgment mechanism that is applied to the network training process to remove unnecessary connections, nodes, and even convolutional kernels (LeCun et al., 1989)

Low-rank factorization Low-rank factorization is the process of breaking down an original tensor into some low-rank tensors, which facilitates the reduction of convolution operations and facilitates network operating speed. The convolution kernel can be regarded as a 3-D tensor, and the fully connected layer can be regarded as a 2-D tensor. If X_{mn} is a numerical matrix, rank(X_{mn}) is the rank of X. If rank(X_{mn}) is much smaller than m or n, then we call X_{mn} a low-rank matrix. Each row or column of the low-rank matrix can be expressed as a linear combination of other rows or columns, indicating that it contains a large amount of redundant information. By projecting the matrix onto a lower-dimensional linear subspace, it is possible to represent it with only a few low-rank vectors and decrease computational workload. To accelerate a simple CNN model with a straightforward structure, several low-rank approximation and clustering methods proposed by Denton et al. (2014) achieved a 2x acceleration on a single convolutional layer, while suffering 1% accuracy drop in classification tasks. The low-rank approximation is applied layer by layer, and fine-tuning is performed based on the approximation error once each layer has been approximated. Utilizing non-linear least squares, Lebedev et al. (2014) computes the Canonical Polyadic decomposition for the kernel tensors. Additionally, Batch Normalization is used by Tai et al. (2015) to transform activations in CNNs' hidden layers, allowing for the network to be trained from scratch and further enhancing the overall model performance.Low-rank factorization needs more re-training and is computationally expensive compared to the original model.

Quantization. The concept of quantizing neural networks was first proposed by Balzer et al. (1991) and Fiesler et al. (1990) to facilitate the hardware implementation of neural networks. The neural network can be quantized before, during, or after training. For practical inference, the floating-point Neural Network model is often built and subsequently quantized (Zhou et al., 2017). After training, quantization is used to speed up inference and conserve energy. Nevertheless, quantization is used in training to lower the network size and increase the accuracy. By employing a hash function to randomly group connection weights, HashedNets (Chen et al., 2015) minimize model sizes by giving all connections in the same hash bucket the same parameter value. DoReFa-Net was suggested by Zhou et al. (2016) to train CNN using low bit-width weights and activations with low bit-width gradients. Gradients must be quantized stochastically, but weights and activations can be quantized deterministically in the DoReFa-Net. Choi et al. (2018) developed the PArameterized Clipping acTivation (PACT) function, a new activation method for determining the best quantization scale during training, in which a new activation clipping parameter α is introduced and optimized during training. Hubara et al. (2016) presented quantized Neural Network for quantifying weights and activations during training and inference.

Network Pruning. LeCun proposed Optimal Brain Damage (OBD) (LeCun et al., 1989), which substantially speeds up the network's training process while achieving the ideal balance between network complexity and training error by deleting irrelevant connections from the network. The Hessian matrix in the loss function of Hassibi et al.'s proposed "Optimal Brain Surgeon" (OBS) (Hassibi et al., 1993)is unconstrained, which makes OBS more generalizable than OBD in other networks. In order to transfer deep convolutional networks to mobile devices, Han et al. (2015a) suggested Deep compression, which includes pruning, quantization, and coding algorithms and can compress the network by 35–49 times without impacting accuracy. According to Guo et al. (2016), reinstating pruned essential connections is crucial for enhancing network performance since the relevance of parameters changes as the network is trained. Dynamic Network Surgery suggests combines a repair operation with the pruning process to reactivate the pruned connections when they become crucial. By performing these two operations alternately after each training session, they significantly increase the effectiveness of the network learning process. Magnitude-based pruning does not work well in the convolutional layer, according to Li et al.'s research (Li et al., 2016), but it works well in the fully

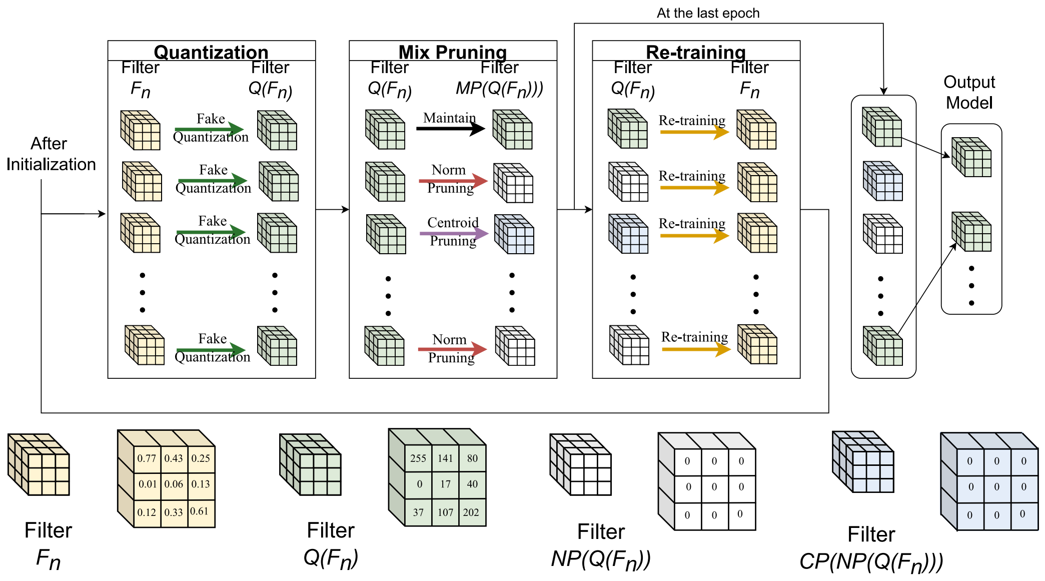

Fig. 1. The method of Single-Shot Pruning and Quantization. The trained filters are marked as yellow, while the quantified filters are marked as green. The filters pruned by norms are marked as white, and those pruned by centroid are marked as blue. The values of the parameters are indicated by the numbers in the filters. The filters are trained in a loop, which includes quantization, mix pruning, and retraining. At the last training epoch, all zeroized filters are pruned, and the small model is produced as output.

connected layer. By directly deleting the convolutional kernels and the accompanying feature maps, which have less of an impact on the output accuracy, in a non-sparse linked manner, they cut the computational complexity by 30% .

Discussion. As far as we know, few works focus on pruning neural networks with quantization; most researchers divide quantization and pruning into two parts in their compression process, such as Han et al. (2015a). The errors caused by pruning and the quantization errors are trained separately, where the parameters cannot be updated by the two kinds of errors simultaneously. Our work can prune and quantize CNNs in a Single-Shot process and modify the weights by pruning and quantization errors. We use two pruning criteria in a "soft" manner to give a better environment for the quantization part and increase the model's performance. Besides, we successfully implemented our compressed model on the embedded system, NVIDIA Xavier NX. We achieved a pipeline from software to hardware.

3. Single-shot pruning and quantization

Our method contains four steps, as illustrated in Fig. 1. First, initialize all the filters for better performance in the weight-updating progress. Second, we quantify all the filters with asymmetric quantization. After that, we set a ratio, which determines the number of channels pruned by norms, and prunes every convolutional layer. Third, based on what has been pruned, we continue to prune the rest channels near the centroid by another ratio. After pruning, we retrain all the weights and turn to the next epoch. The entire algorithm is listed in Algorithm 1:

Algorithm 1 Single-Shot Pruning with Quantization

```
Require:
  norm pruning ratio $R_n\%$
  centroid pruning ratio $R_c\%$
  quantization width $m$
  layer number $C$
  filter number per-layer $N = \{N_i, 0 \le i \le C\}$
  training epoch number $E$
Ensure:
  model parameter $\mathcal{W} = \{\mathcal{W}_i, 0 \le i \le C\}$
  for $epoch = 1, epoch \le E; epoch = epoch + 1$ do
    Update the parameters $\mathcal{W}_i$ by training data
    Fake-quantize the weight with $m$ in $\mathcal{W}_i$ by Equation 1
    for $j = 1, j \le C; j = j + 1$ do
      Zero the $R_n\%N_i$ filters by $l_2 - norm$
      Zero the $R_c\%N_i$ filters near centroid by Equation 7
    end for
  end for
  Get the rest none-zero filters $\mathcal{W}^*$
  Transform the $\mathcal{W}^*$ to hardware-friendly model $\hat{\mathcal{W}}$
  return $\hat{\mathcal{W}}$
```

3.1. *Initialization*

Before applying the quantization method to our model, we need to initialize all the weights in all the layers to avoid imbalanced weight distributions. The Lottery Ticket Hypothesis (Frankle and Carbin, 2018) suggests that in a randomly initialized neural network, there exists a sub-network initialized with specific weights that can match the testing accuracy of the original network after training. The SynFlow method (Tanaka et al., 2020) proposed directly addresses the "winning ticket" issue during network weight initialization, preventing layer collapse and premature pruning. A stronger hypothesis suggests that for any network with sufficient redundant parameters, there exists a sub-network initialized with random weights that can achieve similar accuracy as the original network without any training. However, this relies on many assumptions. Orseau et al. (2020) removes most of these assumptions and relies solely on the constraint of a logarithmic factor coefficient to the sub-network with constrained parameter redundancy. Our training conditions satisfy the above hypothesis so we adopt random initialization which ensures stable convergence of our model. Related work such as He et al. (2018) and He et al. (2019b) also demonstrate that random initialization does not affect stability.

Here we use cross-layer equalization (Nagel et al., 2021) to improve depth-wise separable layers. This initialization method comes from

Nagel, which makes quantization more robust by setting a diagonal matrix with a scale factor for a neuron in a particular way.

3.2. Quantization

The quantization algorithm quantizes the weights of the network matrix into a lower number of bits to represent them, which in turn reduces the storage space and computational complexity occupied by the network. We use asymmetric quantization per channel to quantify the trained parameters. Firstly, we collect the distribution statistics of the parameters to calculate the scale factor **s** and zero-point **z** for each layer. Secondly, we set the zero-point **z** and use the scale factor **s** to map the floating point value to the integer grid. We apply quantization operation as follows:

$$\mathbf{x}_{\mathrm{q}} = \mathbf{Q}(x; s, z, B) = \mathbf{Clamp}\left(\left\lfloor \frac{\mathbf{x}}{\mathrm{s}} \right\rceil + z; 0, 2^B - 1\right) \tag{1}$$

in which **s** stands for the scale factor, which specifies the step-size of the quantifier. **x** represents original activations or weights, and x_q represents quantified activations or weights. **B** is the bit-width. $\lfloor\cdot\rceil$ represents rounding-to-nearest-integer operation. The **Clamp** function is defined as:

$$\mathbf{Clamp}(x; min, max) = \begin{cases} min, & x < min, \\ x, & min \le x \le max, \\ max, & x > max \end{cases} \tag{2}$$

It is noted that we use simulated quantized weights and activations for both forward and backward calculations, but update weights in floating point with gradient updates for gradual rather than abrupt changes. These modified weights are then quantized for further calculations. We use de-quantization function as follows:

$$\hat{\mathbf{x}}_{\mathrm{q}} = \mathbf{D}(\mathbf{x}_{\mathrm{q}}; s, z) = (\mathbf{x}_{\mathrm{q}} - z) \times s \tag{3}$$

So in the end, the simulated quantized parameters could be expressed as:

$$\begin{aligned} \hat{\mathbf{x}}_{\mathrm{q}} &= \mathbf{D}(\mathbf{Q}(x; s, z, B); s, z) \\ &= s \times \left[\mathbf{Clamp}\left(\left\lfloor \frac{\mathbf{x}}{s} \right\rceil + z; 0, 2^B - 1\right) - z\right] \end{aligned} \tag{4}$$

The quantization function **Clamp** is a step function by nature, and the output is a discretized constant. There is a gradient disappearance problem for the discrete constant weights in the quantized network, which brings trouble during gradient-based training. To address that, Hubara et al. (2016) propose straight-through estimator (STE), which approximates the gradient of the rounding operator as 1:

$$\frac{\partial \lfloor Loss \rceil}{\partial w} = 1 \tag{5}$$

In this way, we can calculate the gradients of the quantized parameters. Here we choose to use 16-bit floating point and 8-bit integer as our quantization width.

3.3. Pruning filters with small norms

After quantization, we need to focus on the most "significant" filters, so we use the "smaller-norms-less-important" principle. We use SFP(Soft Filter Pruning) method (He et al., 2018), which suggests that the filter with l2-norm in the lower R_n% of the ranking may be less important. Instead of pruning them, the parameters of the filter are set to zero, allowing these filters to participate in subsequent training processes so that they can be re-trained and join the next training epoch to avoid downgrading the robustness and performance of the model. To determine the number of filters that should be zeroed out, we established a hyper-parameter called the pruning rate, denoted as R_n%, for norm-based pruning. Precisely, during every training epoch, the model is updated by training data. After that, we calculate all the filters' l2-norms, sort all the channels by l2-norms, and zero the filters for the smallest R_n%. Then the partial zeroized model joins the centroid pruning procedure.

3.4. Pruning filters near centroid

In mathematics and physics, the centroid, also known as geometric center or center of figure, of a plane figure or solid figure is the arithmetic mean position of all the points in the surface of the figure. The same definition extends to any object in n-dimensional Euclidean space. There is always a geometric center of a finite number of points, which can be reached by calculating the arithmetic mean of each coordinate component of these points. This center is the unique minimum of the sum of squares of distances from one point in space to these finite points: Given a finite set of points $x_1, x_2, \dots, x_k \in \mathbb{R}^n$, their centers are defined by $X_{centroid}$:

$$X_{centroid} = \frac{x_1 + x_2 + \cdots + x_k}{k} \tag{6}$$

where $x_k = (x_{k1}, x_{k2}, x_{k3}, \dots \dots, x_{kn}) \in \mathbb{R}^n$. From Eq. (5) we can find a centroid of filters in the same layer as:

$$F_{centroid} = \frac{F_1 + F_2 + \cdots + F_c}{c} \tag{7}$$

From Fletcher et al. (2008) we know that centroid is a robust estimation of centrality for data in Euclidean spaces. When a point x_k is closest to the centroid point $X_{centroid}$ in high-dimensional space and removing it has the least impact on the overall distribution of the data, it can be regarded as having the most common information (He et al., 2019b), and can be represented by a linear combination of other points. Similarly, when a filter F_i in the ith layer is closest to the centroid filter $F_{centroid}$, we can assume that removing it has the least impact on the network. Therefore, we set a pruning rate $R_c\%$, select the $R_c\%N_i$ filters closest to the centroid in the ith layer, remove them, and the removed $R_c\%N_i$ filters can be represented by the linear combination of other $(1 - R_c\%)N_i$ filters. We apply similar operations to all convolutional layers. As a result of the parameter updating process involved in backpropagation, it becomes feasible to reconstruct the layer without drop on performance. Thus, it is feasible to expedite our inference by minimizing computational tasks and conserving memory space.

$$\mathcal{F} = \{F_{i,n}; F_{i,n} = \underset{n=1,2,\dots\dots,R_c\%N_i}{argmin} \sqrt{(F_{i,n} - F_{centroid})^2}\} \tag{8}$$

Here m is the number of input channels, n is the number of output channels, R_c% is the ratio or centroid pruning, $F_{centroid}$ means the centroid of filters, and $argmin_{n=1,2,\dots\dots,R_c\%N_i}$ means find the $R_c\%N_i$ channels with the smallest l_2 norm among the output channels in the ith layer.

3.5. Re-training

After pruning filters with small norms and near centroid, we need to retrain all the filters and update the weights by the backpropagation algorithm to eliminate the quantified and pruning errors. Notice that all the pruned filters are set to zero instead of pruning, so the channel numbers remain the same, and most of the optimizer could be used to update the filters. The refresh parameters are quantified and pruned again until the training epochs are done.

3.6. Hardware-friendly model generation

Finally, we need to transfer the pruned model to a hardware-friendly mode. There are three steps: First, prune all the channels set to zeros at the last training epoch. Here we cut the number of channels to the pruning ratio. Second, we rearrange the numbers of each layer so that it can be divided by 8, which is suitable for hardware acceleration. Third, we export the model to the ONNX model so that the trained model can be employed better on embedded systems, such as Raspberry Pi and NVIDIA NX Xavier.

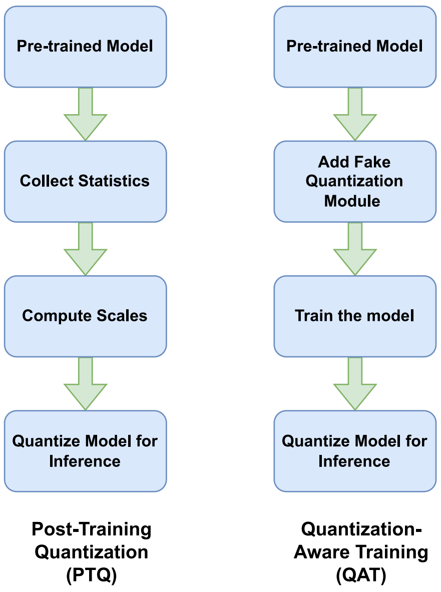
Pre-trained Model
Collect Statistics
Compute Scales
Quantize Model for Inference
Post-Training Quantization (PTQ)
Pre-trained Model
Add Fake Quantization Module
Train the model
Quantize Model for Inference
Quantization-Aware Training (QAT)

Fig. 2. PTQ and QAT workflow. The PTQ method involves calculating the scales by collecting the statistics of parameters and activations in each layer of the pre-trained model, before mapping the layer's parameters to a corresponding low-bit value range. QAT incorporates a "fake-quantization" module during training to introduce quantization error into the model training process, thereby achieving a fine-tuning effect to improve accuracy. Both workflows require quantizing the model to match the specific inference machine.

4. Experiments

We aim to employ a classification model on the embedded system, so we choose MobileNetV2 as our experiment network due to its low cost of memory and energy and high accuracy. We have performed quantization and pruning on MobileNetV2 with CIFAR-10 and CIFAR-100 datasets. Then we pruned and exported a small model to the ONNX model to evaluate its inference time and memory cost. To illustrate our quantization and pruning method's advantages, we have carried ablation study for further comparison.

4.1. Settings

4.1.1. Quantization setting

We choose MobileNetV2 as our training model, and we use 16-bit float (FP16) and 8-bit signed integer (INT8) as our quantization width. There are two kinds of workflow in CNN linear quantization: Post-Training Quantization (PTQ) and Quantization-Aware Training (QAT) (Krishnamoorthi, 2018). PTQ is a quantization method that trains the model with full precision and converts the model to a specific width, such as 16-bit or 8-bit. QAT uses fake quantization operators to train the model in a "quantified" way and convert the model to a specific width in the end. Fig. 2 shows the two quantization workflows. Here we have trained both full-precision models from scratch for further PTQ tests; 8-bit and 16-bit models used QAT during training.

4.1.2. Pruning setting

We use two kinds of criterion to prune the filters: norm pruning and centroid pruning. We choose l2-norm for filter selection and use l2-norm for distance calculation from Eq. (8). We prune all the convolutional and batch normalization layers with the same norm pruning and centroid rates in each training epoch.

Table 1
MobileNetV2 on CIFAR-10.

Mode	NP (%)	CP (%)	TP (%)	Top1 Acc. (%)	Top5 Acc. (%)
Baseline	0	0	0	86.67	99.85
Q	0	0	0	84.16	99.61
Q + P	0.1	0.2	0.3	84.11	99.42
	0.1	0.3	0.4	**85.08**	99.33
	0.1	0.4	0.5	**84.21**	99.25
	0.1	0.5	0.6	81.69	99.10
	0.2	0.2	0.4	**85.35**	99.40
	0.2	0.3	0.5	**84.2**	99.31
	0.2	0.4	0.6	81.97	99.27
	0.2	0.5	0.7	77.5	98.78

"NP" represents "Norm Pruning", which stands for the rate of norm pruning. "CP" represents "Centroid Pruning", which stands for the rate of centroid pruning. "TP" represents "Total Pruning", which stands for the total rate of filter pruning. "Top1 Acc." represents the top1 classification accuracy, and "Top5 Acc." represents the top5 classification accuracy.

Table 2
MobileNetV2 on CIFAR-100.

Mode	NP (%)	CP (%)	TP (%)	Top1 Acc. (%)	Top5 Acc. (%)
Baseline	0	0	0	59.94	85.51
Q	0	0	0	53.6	84.35
Q + P	0.1	0.1	0.2	56.78	84.10
	0.1	0.2	0.3	58.21	83.82
	0.1	0.3	0.4	**56.65**	82.38
	0.1	0.4	0.5	50.03	80.73
	0.2	0.1	0.3	56.68	84.28
	0.2	0.2	0.4	54.65	82.53
	0.2	0.3	0.5	50.81	80.28
	0.2	0.4	0.6	42.92	74.59

"NP" represents "Norm Pruning", which stands for the rate of norm pruning. "CP" represents "Centroid Pruning", which stands for the rate of centroid pruning. "TP" represents "Total Pruning", which stands for the total rate of filter pruning. "Top1 Acc." represents the top1 classification accuracy, and Top5 "Acc." represents the top5 classification accuracy.

4.2. MobileNetV2 on CIFAR-10, CIFAR-100

We have trained MobileNetV2 on image classification tasks with CIFAR-10 and CIFAR-100. The CIFAR-10 dataset consists of 60,000 32×32 color images divided into ten different classes, of which 50,000 are used for training and 10,000 for testing. Similar to the CIFAR-10, the CIFAR-100 offers 100 classes with a total of 600 photos. There are 100 assessment photos and 500 training images per class. The CIFAR-100's 100 classes are divided into 20 superclasses. Each image has a "fine" and a "coarse" designation, indicating the class to which it belongs (the superclass to which it belongs). The parameter settings are the same as He et al. (2016), and the training schedule is defined as Zagoruyko and Komodakis (2016). From Tables 1 and 2, we can see that we have achieved 50% reduction in the number of channels in the model with a 2% drop in accuracy on the CIFAR-10 dataset and 40% reduction in the number of channels in the model with a 3% drop in accuracy on the CIFAR-100.

4.3. Comparison of different compressed MobileNetV2 on CIFAR10 and CIFAR100

We conducted a comprehensive comparison between various compressed methods for the Top1 accuracy on CIFAR10 and CIFAR100, Multiply–Accumulate Operations (MACs), parameter count and the size of the model, which can be found in Table 3. One MAC equals one multiply or divides operation and one add or minus operation. Here we use 40% as our pruning rate on PFEC (Li et al., 2017), FPGM (He et al., 2022) and our method, and choose 8-bit as bit-width for DTQ (Liu et al., 2018) and for our method. From Table 3, we can see that our method achieved great advantages using MobileNetV2 on both CIFAR10 and

Table 3
Comparison of compressed MobileNetV2 on CIFAR10 and CIFAR100.

Method	CIFAR10 Top1 Acc.	CIFAR100 Top1 Acc.	Parameters (M)	MACs (M)	Size (MB)
DTQ (8 bit) (Liu et al., 2018)	83.21	55.72	2.139	6.427	4.278
PFEC (40%) (Li et al., 2017)	83.46	53.10	1.327	3.821	5.308
FPGM (40%) (He et al., 2022)	84.20	53.28	1.112	3.132	4.448
Ours (40%, 8 bit)	84.35	56.65	1.063	3.007	2.216

"CIFAR10/CIFAR100 Top1 Acc." means the top1 accuracy of the method on CIFAR10/CIFAR100, "Parameters" means the number of the model generated by the method, "MACs" means the model's number of multiply–accumulate operations. "Size" means the total size of the model in MB (Million Byte).

Table 4
Average inference time on hardware of CIFAR-10, CIFAR-100.

Data-set	STATE	FP32 (ms)	TRT + FP32 (ms)	TRT + FP16 (ms)	TRT + INT8
CIFAR-10	Original (PTQ)	121.28	75.56	29.17	19.21
	Original (QAT)	118.89	73.84	23.75	17.89
	Pruned 50% (QAT)	63.20	35.48	**15.15**	**11.98**
CIFAR-100	Original (PTQ)	121.64	77.69	24.22	12.39
	Original (QAT)	119.28	75.14	24.05	13.77
	Pruned 40% (QAT)	73.88	41.58	**16.65**	**8.70**

This table shows the average inference time for 1024 RGB images on NVIDIA Xavier NX.
"TRT" stands for "TensorRT".

CIFAR100 datasets. Due to the use of 8-bit integers for inference, we greatly reduced the size of the model while maintaining comparable parameter and computation requirements to PFEC (Li et al., 2017) and FPGM (He et al., 2022). As a result, the memory demand for deployment on embedded systems has significantly decreased, and the computation has correspondingly reduced, leading to an increase in the inference speed. Additionally, the utilization of structured pruning guarantees a seamless integration into embedded systems, augmenting overall usability.

4.4. Hardware test

We have tested the mean inference time of compressed models. We randomly choose 1024 images from the dataset and measure the mean time when the images are imported to the original model and compressed model trained by CIFAR-10 and CIFAR-100. To compare the contribution of quantization and pruning for speedup, we also employ PTQ and QAT to the original model and test the inference time at the same condition. The result is in Table 4. From the table, we can see that regardless of the effect of hardware acceleration from TensorRT, we improved the classification inference time from 75.56 ms to 11.98 ms for 1024 images for the model trained on CIFAR-10, which brings 6.26 times performance improvement.

4.5. Ablation study

4.5.1. Quantization workflow

We choose NVIDIA TensorRT as our hardware transform SDK for deep learning inference. TensorRT can deliver low latency and high throughput for inference. Hence, we separate the Original model in floating point 32, the TensorRT model in floating point 32, and the TensorRT model in floating point 16 to show separate the acceleration of TensorRT, quantization, and pruning. To better illustrate and compare the contribution of quantization and pruning methods to model inference acceleration, we chose two quantization methods for the original model: PTQ and QAT. From Table 4, we can see that PTQ and QAT perform likely on inference acceleration. Pruning speed up the procedure of inference. This is because fewer channels bring up fewer parameters and operations.

Table 5
Comparison of norm pruning, centroid pruning and mixed pruning.

Data-set	PR (%)	NP Acc. (%)	CP Acc. (%)	MP Acc. (%)
CIFAR-10	0.3	81.94	85.03	84.41
	0.4	78.94	84.29	**85.35**
	0.5	75.88	83.31	**84.21**
	0.6	71.51	80.06	81.97
CIFAR-100	0.2	57.6	56.68	59.88
	0.3	56.23	56.33	**58.21**
	0.4	54.94	53.69	**56.65**
	0.5	52.36	50.20	50.08

"PR" represents "Pruning Rate", which stands for the rate of filter pruning. "NP Acc." represents "Norm Pruning Accuracy", which stands for the classification accuracy of norm pruning. "CP Acc." represents "Centroid Pruning Accuracy", which stands for the classification accuracy of centroid pruning. "MP Acc." represents classification accuracy of the mixed pruning.

4.5.2. Comparison of norm pruning, centroid pruning and mixed pruning

We also compare the effect of norm pruning, centroid pruning, and mixed pruning. We use norm pruning, centroid pruning, and mix pruning separately on CIFAR10 and CIFAR100 at the same pruning rate and compare their accuracy to find the best performance. The result is in Table 5. From the table, we can see that on CIFAR-10, when the pruning rate gets higher, mix pruning performs better than singly norm pruning or singly centroid pruning. Besides, centroid pruning performs better than norm pruning from 0.3 to 0.6 pruning rate. On CIFAR-100, norm pruning performs better than centroid pruning while they increase pruning accuracy. When the pruning rate increases, the accuracy falls faster due to an inadequate number of parameters.

The main idea of norm pruning is that filters with small l2 norms are considered unimportant, so these filters with small norms are prioritized for removal during pruning. However, as the pruning rate increases, some important filters may also be removed because they are ranked low in the norm sorting, leading to a decrease in accuracy. However, not all filters with small norms are unimportant. If the distribution variance of filters in the layer is small and the minimum norm value is relatively large, network pruning based on the "less-norm-less-important" principle would remove filters that have a small norm but contribute significantly to the output.

The main idea of centroid pruning is to measure the similarity and "replaceability" of each filter, and remove the most replaceable filter. When the pruned filters are updated to non-zero by back-propagation,

Table 6
MobileNetV2 on CIFAR-10 with post-training.

Dataset	B	Q	TP (%)	PTP Top1 Acc. (%)	PAT Top1 Acc. (%)
CIFAR-10	86.67	84.13	0.3	80.71	84.11
			0.4	78.01	85.35
			0.5	65.85	84.21
CIFAR-100	59.94	53.60	0.3	38.92	58.21
			0.4	35.24	56.65
			0.5	12.90	50.81

"PR" represents "Pruning Rate", which stands for the rate of filter pruning. "B" stands represent "Baseline", which means that the classification accuracy of model without any compression technique. "Q" represents "Quantization", which stands for the classification accuracy of model after training with Quantization. "TP" represents "Total Pruning Rate". "PTP Acc." represents "Post-Training Pruning Accuracy", which stands for the classification accuracy of model pruned after training. "PAT Acc." represents "Pruning-Aware Training Accuracy", which stands for the classification accuracy of model pruned during training.

other filters can be used to represent the replaced filter. Therefore, under the same pruning rate on CIFAR10, the representation ability of the centroid pruned model is better than that of the norm pruning model, because it has more linearly independent tensor in high-dimensional space. But when training MobileNetV2 on CIFAR100, the variance of the parameter distribution is smaller and the smallest norm filter is also relatively small. So at the same pruning rate, the benefits of removing smaller filters are greater than the benefits of keeping them.

Mix pruning is a balance between norm pruning and centroid pruning within the same pruning rate. It removes filters with small norms that have little impact on the output, while retaining filters with small norms but good representational capacity. This combination also reduces computational complexity, as centroid pruning requires more computational operations than norm pruning.

4.5.3. Comparison of pruning during training and pruning after training

We conducted a comprehensive analysis of the impact of PTQ and QAT workflows on model performance, as well as the performance of models with Post-training pruning and pruning-aware training. Table 6 indicate that the accuracy of post-training pruning on a QAT model is inferior to that of pruning-aware training. This can be attributed to the fact that each filter in a trained model corresponds to a feature map of the input image. Without fine-tuning the model, pruning alone leads to a reduction in the number of channels and consequent underfitting of the model's output.

4.5.4. Comparison of training time of quantization after pruning and single-shot pruning and quantization

We conducted a comprehensive analysis of the training and exporting process for models using our methodology versus quantization post pruning on NVIDIA GeForce GTX 1080 Ti. Specifically, we assessed the impact of norm pruning rate = 0.1 and centroid pruning rate = 0.3 on overall time efficiency. Results indicate that when using Pruning + QAT, norm and centroid pruning are the training components, in addition to QAT (INT8) and the model's adaptation to NVIDIA TensorRT, resulting in prolonged training time. Conversely, our Single-Shot approach incorporates quantization modules during pruning, eliminating the need for QAT. Nonetheless, the output model remains in FP32 form, requiring additional time for conversion to TensorRT. From Table 7, we can see that on CIFAR10, we saved 54.5 min and improved efficiency by 19.4%; on CIFAR100, we saved 88.5 min and improved efficiency by 24.7%, and our method reduced the number of operations and lowered operation complexity.

5. Conclusion

In this paper, we first list compression approaches for neural networks and highlight the complexity of the quantization and pruning division implementation process, which suffers from the inability to reduce both quantization and pruning errors during training. In order to achieve this, we propose a technique that can both prune and quantize neural networks. We successfully test the deployed trained and compressed model on a hardware platform.

Our method outperforms the traditional approach of pruning and quantization, as we introduce quantization into the training process. This allows the network to take into account both quantization error and pruning error during parameter updating, leading to an impressive maintenance of accuracy while dramatically reducing training time by 20% to 25%. Our approach stands out among other model compression methods by significantly reducing the computational cost (MACs) and size of the model while maintaining similar accuracy. Using a combination of pruning and quantization, we are able to effectively decrease the number of model parameters and bit-width of parameters, thereby decreasing memory usage and power consumption during model inference.

The compressed model effectively attained an 8-fold speedup while being compressed to 30% of the original model with a 2% drop in recognition accuracy.

6. Future work

In the future, we intend to combine pruning method with more quantitative techniques and test it on tiny embedded systems like MCU.

Filters at different depths have varying sensitivity to feature maps of different depths. Uniformly applying the same pruning rate to all convolutional layers can result in inadequate pruning for some layers with redundant parameters and excessive pruning for other layers sensitive to activations, leading to reduced performance. In the future, we will set different pruning rates between different layers to achieve better compression rates while maintaining high accuracy. Furthermore, we plan to enhance the proficiency by integrating structured pruning techniques alongside diverse quantization methodologies like LSQ (Esser et al., 2019), LSQ+ (Bhalgat et al., 2020) and AdaRound (Nagel et al., 2020) in our upcoming server-based CNNs training regimen. Subsequently, we aim to apply our approach by conducting inference evaluations on less powerful embedded devices such as NVIDIA Jetson Nano and Raspberry Pi 4.

CRediT authorship contribution statement

Bofeng Jiang: Conceptualization, Methodology, Software, Data curation, Writing – original draft, Visualization, Investigation. **Jun Chen:** Validation, Writing – review & editing. **Yong Liu:** Supervision.

Table 7
Comparison of training time of quantization after pruning and single-shot pruning and quantization.

Dataset	Mode	Pruning (min)	QAT (min)	Export (min)	Total (min)
CIFAR10	Pruning + QAT	157.4	97.2	25.4	280.0
	Single-Shot	198.7	0	26.8	225.5
CIFAR100	Pruning + QAT	192.6	127.5	38.3	358.4
	Single-Shot	232.3	0	37.6	269.9

"Pruning" refers to the time required for norm pruning and centroid pruning. "QAT" indicates the time necessary for quantization-aware training. "Export" denotes the time needed to transform a PyTorch model into a NVIDIA TensorRT engine. "Total" is the overall time taken to compress the model and convert it to a hardware-friendly format.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgment

This work is funded in part by the Key Research and Development Project of Zhejiang Province under Grant 2021C01035.

References

Anwar, S., Sung, W., 2016. Coarse pruning of convolutional neural networks with random masks.

Ba, J., Caruana, R., 2014. Do deep nets really need to be deep? Adv. Neural Inf. Process. Syst. 27.

Balzer, W., Takahashi, M., Ohta, J., Kyuma, K., 1991. Weight quantization in Boltzmann machines. Neural Netw. 4 (3), 405–409.

Bhalgat, Y., Lee, J., Nagel, M., Blankevoort, T., Kwak, N., 2020. LSQ+: Improving low-bit quantization through learnable offsets and better initialization. CoRR, abs/2004.09576.

Chen, J., Bai, S., Huang, T., Wang, M., Tian, G., Liu, Y., 2023a. Data-free quantization via mixed-precision compensation without fine-tuning. In: Pattern Recognition, Vol. 109780. Elsevier.

Chen, J., Chen, H., Wang, M., Dai, G., Tsang, I.W., Liu, Y., 2023b. Learning discretized neural networks under Ricci flow. arXiv preprint arXiv:2302.03390.

Chen, J., Liu, L., Liu, Y., Zeng, X., 2021. A learning framework for n-bit quantized neural networks toward FPGAs. IEEE Trans. Neural Netw. Learn. Syst. 32 (3), 1067–1081. http://dx.doi.org/10.1109/TNNLS.2020.2980041.

Chen, W., Wilson, J.T., Tyree, S., Weinberger, K.Q., Chen, Y., 2015. Compressing neural networks with the hashing trick.

Choi, J., Wang, Z., Venkataramani, S., Chuang, P.I.-J., Srinivasan, V., Gopalakrishnan, K., 2018. Pact: Parameterized clipping activation for quantized neural networks. arXiv preprint arXiv:1805.06085.

Denton, E.L., Zaremba, W., Bruna, J., LeCun, Y., Fergus, R., 2014. Exploiting linear structure within convolutional networks for efficient evaluation. Adv. Neural Inf. Process. Syst. 27.

Esser, S.K., McKinstry, J.L., Bablani, D., Appuswamy, R., Modha, D.S., 2019. Learned step size quantization. CoRR, abs/1902.08153.

Fiesler, E., Choudry, A., Caulfield, H.J., 1990. Weight discretization paradigm for optical neural networks. In: Optical Interconnections and Networks, Vol. 1281. SPIE, pp. 164–173.

Fletcher, P.T., Venkatasubramanian, S., Joshi, S., 2008. Robust statistics on Riemannian manifolds via the geometric median. In: 2008 IEEE Conference on Computer Vision and Pattern Recognition. IEEE, pp. 1–8.

Frankle, J., Carbin, M., 2018. The lottery ticket hypothesis: Finding sparse, trainable neural networks. arXiv preprint arXiv:1803.03635.

Guo, Y., Yao, A., Chen, Y., 2016. Dynamic Network Surgery for Efficient DNNs, p. 9.

Han, S., Mao, H., Dally, W.J., 2015a. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. Tech. Rep.

Han, S., Pool, J., Tran, J., Dally, W.J., 2015b. Learning both weights and connections for efficient neural networks.

Hassibi, B., Stork, D.G., Wolff, G.J., 1993. Optimal brain surgeon and general network pruning. In: IEEE International Conference on Neural Networks. IEEE, pp. 293–299.

He, Y., Kang, G., Dong, X., Fu, Y., Yang, Y., 2018. Soft filter pruning for accelerating deep convolutional neural networks.

He, Y., Liu, P., Wang, Z., Hu, Z., Yang, Y., 2019b. Filter pruning via geometric median for deep convolutional neural networks acceleration.

He, Y., Liu, P., Zhu, L., Yang, Y., 2022. Filter pruning by switching to neighboring CNNs with good attributes. IEEE Trans. Neural Netw. Learn. Syst. 1–13.

He, K., Zhang, X., Ren, S., Sun, J., 2016. Identity mappings in deep residual networks. In: European Conference on Computer Vision. Springer, pp. 630–645.

Hinton, G., Vinyals, O., Dean, J., et al., 2015. Distilling the knowledge in a neural network, vol. 2 (7). arXiv preprint arXiv:1503.02531.

Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., Bengio, Y., 2016. Quantized neural networks: Training neural networks with low precision weights and activations.

Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., Kalenichenko, D., 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2704–2713.

Krishnamoorthi, R., 2018. Quantizing deep convolutional networks for efficient inference: A whitepaper.

Lebedev, V., Ganin, Y., Rakhuba, M., Oseledets, I., Lempitsky, V., 2014. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. arXiv preprint arXiv:1412.6553.

LeCun, Y., Denker, J., Solla, S., 1989. Optimal brain damage. Adv. Neural Inf. Process. Syst. 2.

Li, H., Jin, P., Cheng, Z., Zhang, S., Chen, K., Wang, Z., Liu, C., Chen, J., 2023. TG-VQA: Ternary game of video question answering. arXiv preprint arXiv:2305.10049.

Li, H., Kadav, A., Durdanovic, I., Samet, H., Graf, H.P., 2016. Pruning filters for efficient convnets. arXiv preprint arXiv:1608.08710.

Li, H., Kadav, A., Durdanovic, I., Samet, H., Graf, H.P., 2017. Pruning filters for efficient ConvNets. arXiv:1608.08710[cs.CV].

Li, H., Li, X., Karimi, B., Chen, J., Sun, M., 2022. Joint learning of object graph and relation graph for visual question answering. arXiv preprint arXiv:2205.04188.

Licheng, W., Wang, K., 2022. Online estimation of SOH for lithium-ion battery based on SSA-Elman neural network. Prot. Control Mod. Power Syst. 7, http://dx.doi.org/10.1186/s41601-022-00261-y.

Liu, B., Cao, Y., Long, M., Wang, J., Wang, J., 2018. Deep Triplet Quantization. ACM, MM.

Liu, Y., Chen, J., Liu, Y., 2023. DCCD: Reducing neural network redundancy via distillation. IEEE Trans. Neural Netw. Learn. Syst. 1 (1), 1–12. http://dx.doi.org/10.1109/TNNLS.2023.3238337.

Mao, H., Han, S., Pool, J., Li, W., Liu, X., Wang, Y., Dally, W.J., 2017. Exploring the Granularity ofSparsity in Convolutional Neural Networks, p. 8.

Nagel, M., Amjad, R.A., van Baalen, M., Louizos, C., Blankevoort, T., 2020. Up or down? Adaptive rounding for post-training quantization. CoRR, abs/2004.10568.

Nagel, M., Fournarakis, M., Amjad, R.A., Bondarenko, Y., van Baalen, M., Blankevoort, T., 2021. A white paper on neural network quantization.

Orseau, L., Hutter, M., Rivasplata, O., 2020. Logarithmic pruning is all you need. Adv. Neural Inf. Process. Syst. 33, 2925–2934.

Sainath, T.N., Kingsbury, B., Sindhwani, V., Arisoy, E., Ramabhadran, B., 2013. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In: 2013 IEEE International Conference on Acoustics, Speech and Signal Processing. IEEE, pp. 6655–6659.

Simonyan, K., Zisserman, A., 2014. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

Tai, C., Xiao, T., Zhang, Y., Wang, X., et al., 2015. Convolutional neural networks with low-rank regularization. arXiv preprint arXiv:1511.06067.

Tanaka, H., Kunin, D., Yamins, D.L., Ganguli, S., 2020. Pruning neural networks without any data by iteratively conserving synaptic flow. Adv. Neural Inf. Process. Syst. 33, 6377–6389.

Wen, W., Wu, C., Wang, Y., Chen, Y., Li, H., 2016. Learning structured sparsity in deep neural networks. Adv. Neural Inf. Process. Syst. 29.

Yu, X., Ma, N., Zheng, L., Licheng, W., Wang, K., 2023. Developments and applications of artificial intelligence in music education. Technologies 11 (2), 42. http://dx.doi.org/10.3390/technologies11020042.

Zagoruyko, S., Komodakis, N., 2016. Wide residual networks. arXiv preprint arXiv:1605.07146.

Zhang, M., Yang, D., Du, J., Sun, H., Liwei, L., Wang, L., Wang, K., 2023. A review of SOH prediction of Li-Ion batteries based on data-driven algorithms. Energies 16 (7), 3167.

Zhao, R., Song, W., Zhang, W., Xing, T., Lin, J.-H., Srivastava, M., Gupta, R., Zhang, Z., 2017. Accelerating binarized convolutional neural networks with software-programmable FPGAs. In: Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. ACM, Monterey California USA, pp. 15–24.

Zhou, S., Wu, Y., Ni, Z., Zhou, X., Wen, H., Zou, Y., 2016. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. arXiv preprint arXiv:1606.06160.

Zhou, A., Yao, A., Guo, Y., Xu, L., Chen, Y., 2017. Incremental network quantization: Towards lossless CNNs with low-precision weights.