INTEGRATION, the VLSI journal 82 (2022) 147–154

Contents lists available at ScienceDirect

Integration, the VLSI Journal

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

DP-Nets: Dynamic programming assisted quantization schemes for DNN compression and acceleration ☆

Dingcheng Yang [a], Wenjian Yu [a,*], Xiangyun Ding [a], Ao Zhou [b], Xiaoyi Wang [b,*]

[a] Dept. Computer Science & Tech., BNRist, Tsinghua University, Beijing, China
[b] Beijing Engineering Research Center for IoT Software and Systems, Beijing University of Technology, Beijing, China

ARTICLE INFO

Keywords:
Dynamic programming
Neural network compression
Robust model
Weight quantization

ABSTRACT

In this work, we present effective quantization schemes called DP-Nets for the compression and acceleration of deep neural networks (DNNs). A key ingredient is a novel dynamic programming (DP) based algorithm to obtain the optimal solution of scalar K-means clustering. Based on the approaches with regularization and quantization function, two weight quantization approaches called DPR and DPQ for compressing normal DNNs are proposed respectively. Accordingly, a technique based on DP-Nets for inference acceleration is presented. Experiments show that DP-Nets produce models with higher inference accuracy than recently proposed counterparts while achieving same or larger compression. They are also extended for compressing robust DNNs, and the relevant experiments show 16X compression of the robust ResNet-18 model with less than 3% accuracy drop on both natural and adversarial examples. The experiments with FPGA demonstrate that the technique for inference acceleration brings over 5X speedup on matrix–vector multiplication.

1. Introduction

Deep neural networks (DNNs) have been demonstrated to be successful on many tasks. However, the size of DNN model has continuously increased while it achieves better performance. As a result, the storage space of DNN becomes a major concern if we deploy it on resource-constrained devices, especially in the edge-computing and AI-of-things applications.

In recent years, there are a lot of work on compressing DNN models. The proposed techniques consist of pruning [1,2], knowledge distillation [3], quantization [4–9] , low-rank approximation [10–13], etc. Among them, quantization based methods represent the network weights with very low precision, thus yielding highly compact DNN models compared to their floating-point counterparts. Weight sharing [4,5,7,8] is a kind of quantization method, which applies clustering on the weights, so as to achieve compression by only recording cluster centers and weight assignment indices. Other quantization methods can be regarded as variants of scalar weight sharing, which restrict the weights to floating-point numbers satisfying certain constraints [6,9]. The parameter space with them is a subspace of the parameter space for the DNN applying the weight sharing with same bit length. Therefore, the weight sharing approach could provide better performance of compression, while the other quantization schemes may be more friendly to inference acceleration.

DNNs are vulnerable to adversarial examples, which can be crafted by adding visually impercetible perturbations on images. Several approaches for training robust DNN models were recently proposed [14,15], to defense the adversarial examples. However, there is few work devoted to the compression of robust DNN model, and existing ones only employ the pruning and/or simple quantization technique [16,17].

In various quantization approaches, the K-means clustering problem is often involved. It is always solved with the Lloyd's algorithm [18] in existing work, resulting in a solution which is non-optimal, and sensitive to the initial guess as revealed by experiments in [8]. In this work, we consider the scalar K-means clustering without using the Lloyd's algorithm, and explore better weight quantization approaches for the compression of normal and robust DNNs. One of our key contributions is a dynamic programming (DP) based algorithm producing the optimal solution of scalar clustering problem. It has $O(N^2K)$ time complexity, where N and K are the numbers of scalars and clusters respectively. The algorithm is collaborated with the weight quantization approaches to become dynamic programming assisted quantization schemes (called

☆ This work was supported by the National Key Research and Development Plan of China (2020AAA0103502), and Beijing National Research Center for Information Science and Technology, China (BNR2019ZS01001).

* Corresponding authors.
E-mail addresses: ydc19@mails.tsinghua.edu.cn (D. Yang), yu-wj@tsinghua.edu.cn (W. Yu), ding-xy16@tsinghua.org.cn (X. Ding), S201861539@emails.bjut.edu.cn (A. Zhou), wxy@bjut.edu.cn (X. Wang).

https://doi.org/10.1016/j.vlsi.2021.10.002
Received 11 April 2021; Received in revised form 14 August 2021; Accepted 18 October 2021
Available online 2 November 2021

DP-Nets), which improve the compression of normal and robust DNN models, and bring inference acceleration as well.

The major contributions of this work are as follows.

- A dynamic programming (DP) based algorithm is proposed to obtain the optimal solution of the K-means clustering problem with scalar data.
- Two DP assisted quantization schemes (called DP-Nets) are proposed for DNN compression and acceleration. They are the DP assisted approach with regularization (called DPR) and the DP assisted approach with quantization function (called DPQ). DPR trains a clustering-friendly network and then compresses it with weight clustering. DPQ trains the network with a formulation including quantization function and employs the DP based algorithm to obtain better clustering.
- The DP-Nets are then extended to schemes called DPR$^+$ and DPQ$^+$, through collaborating with the recently proposed TRADES (TRadeoff-inspired Adversarial DEfense via Surrogate-loss minimization) model [15], for compressing robust DNNs.
- A technique based on DP-Nets for inference acceleration is presented and validated with experiments on FPGA.

Experiments on normal DNNs have shown the advantages of DPR and DPQ over other compression approaches like Deep K-Means [19] and LQ-Net [9]. For GoogLeNet on ImageNet dataset, DPQ results in a model which is 2.5X smaller and with 1% higher inference accuracy than that produced by Deep K-means. For ResNet-18, the models obtained with DPQ show 0.5% higher accuracy than those by LQ-Net with same compression ratio. Besides, up to 77X compression of Wide ResNet is achieved (with $< 3\%$ accuracy drop) by a combinatorial scheme including DPR, the pruning and Huffman coding techniques. Furthermore, experiments on robust DNNs have validated the effectiveness of the proposed DPR$^+$ and DPQ$^+$ approaches. With 2-bit quantization, DPR$^+$ produces a compressed robust ResNet-18 model which exhibits less than 3% accuracy drop on both natural and adversarial examples.

To show the inference acceleration brought by the DP-Nets, we have designed a custom accelerator with specific implementation of matrix–vector multiplication with the compressed model. The experiments on FPGA reveal that at least 5X speedup can be achieved for inference computations.

2. Background

2.1. Weight sharing and quantization

The scalar weight sharing introduced by [8] is the first quantization approach. Regard the weights of DNN as a set of vectors $W = \{W_1, \dots, W_m\}$, where $W_i \in \mathbb{R}^{n_i}$. Accordingly, the result of scalar weight sharing can be expressed as $C = [C_1, \dots, C_m] \in \mathbb{R}^{K\times m}$, where K is the number of clusters and vector C_i contains the K cluster centers. The uncompressed DNN needs $32\sum_{i=1}^{m} n_i$ bits for storing the weights, as each weight is expressed as a 32-bit floating-point number. With the weight sharing, each element of W_i is represented by an element in $C_i \in \mathbb{R}^K$. So, we just need $\log_2 K \cdot \sum_{i=1}^{m} n_i$ bits to encode the index and $32mK$ bits to store the cluster centers. Fig. 1 provides an example of the storage formats. This scalar clustering leads to the compression ratio:

$$r = \frac{32\sum_{i=1}^{m} n_i}{\log_2 K \cdot \sum_{i=1}^{m} n_i + 32mK}, \tag{1}$$

which approximates $32/\log_2 K$.

A naive approach for weight sharing regards the problem as the K-means clustering of the trained weights. However, the weights often follow the Gaussian distribution, which is consistent with the phenomenon observed by [5] that the weights of learned convolutional filters are typically smooth. This is shown in Fig. 2(a) as an example, and means the learned weights may be unsuitable for clustering or

normal	Bits	Value	Value	Value	Value	Value	Value	Value	Value	Value
	32	3.5	3.5	7.2	7.2	7.2	3.5	3.5	3.5	7.2
quantized	Bits	Value	Value	Value	Value	Value	Value	Value	Value	Value
	1	0	0	1	1	1	0	0	0	1
	32	3.5	7.2							

Fig. 1. An example of storage formats. The top table represents the normal storage format, we need 9 32-bit floating-point numbers. The bottom table represents the quantized storage format. Since the value can only be 3.5 or 7.2, we just need 9 bits to indicate which value each number is, and then store 2 32-bit floating-point numbers (3.5 and 7.2).

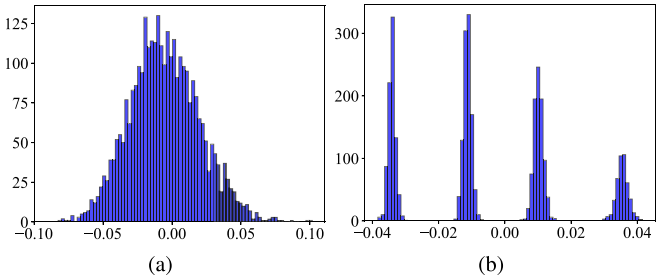

Fig. 2. The histograms of weights in the FreshNet model's first convolution layer obtained from (a) a normal training, and (b) a clustering-friendly training.

quantization. Thus, the network has to be retrained to compensate for the accuracy loss caused by weight sharing [8]. Because for each weight, the cluster it belongs to is fixed during the retraining, this is the same as what is done for HashedNet [4], i.e randomly grouping each weight and then training, and could not produce a good DNN model [20].

Training DNN and performing weight sharing can be formulated as a single optimization problem. If the loss function of training DNN is $L(W)$, the optimization problem can be expressed as:

$$\begin{aligned} &\min_{W,C} L(W), \\ &\text{s. t. } C = [C_1, C_2, \dots, C_m] \in \mathbb{R}^{K\times m}, \\ &W = \{W_1, W_2, \dots, W_m\}, \text{ and } \forall i, j,\ W_{i,j} \in C_i . \end{aligned} \tag{2}$$

The constraint means every element in vector W_i appears in the vector C_i containing cluster centers. This formulation is also suitable for a general quantization method. For a non-weight-sharing approach (e.g. low-bit representation), a problem with more constraints added to (2) is solved.

If the weights are regarded as vectors for clustering, it becomes the vector weight sharing problem, which was investigated in [19]. Vector weight sharing produces larger compression ratio than (1) (even > 32), but it may induce sacrifice on accuracy as compared with the scalar weight sharing. A non-weight-sharing quantization approach can be regarded as a variant of scalar weight sharing. It trades off smaller parameter space for faster inference computation.

There are mainly two kinds of approaches for the quantization problem. One coverts (2) to a formulation with regularization item. The other takes the constraints into account with quantization function. They are briefly introduced in the following two subsections.

2.2. The approach with regularization

The optimization problem (2) is difficult to solve. One can convert the constraints to a regularization item in the loss function. Then, solving the new formulation results in a clustering-friendly model.

For simplicity, we just consider the problem with $m = 1$ where the weight vector $W' \in \mathbb{R}^n$ is clustered to K centers $c_1, c_2, \dots, c_K$. For (2), the constraints can be converted to a regularization item

$\lambda \sum_{i=1}^{n} \min_{1\le k\le K}(W_i' - c_k)^2$ in the loss function, where λ is a Lagrange multiplier. The regularization item corresponds to the loss of K-means clustering problem, which is generally NP-hard for vector data. An approximate algorithm for K-means clustering was proposed in [21], which relaxed the problem to minimizing $Tr(W'^T W') - Tr(F^T W'^T W' F)$ with constraint $F^T F = I$. Here, Tr denotes the matrix trace, $F \in \mathbb{R}^{n\times K}$, and I is the identity matrix. A singular value decomposition (SVD) based algorithm was proposed in [21] to obtain the closed-form solution of the relaxed problem. Based on this, an approach called Deep K-means was proposed for the weight-sharing compression of DNN, which solves [19]:

$$\min_{W',F} \{ L(W') + \lambda[Tr(W'^T W') - Tr(F^T W'^T W' F)]\} \quad \text{s. t.} \quad F \in \mathbb{R}^{n\times K}, \quad F^T F = I. \tag{3}$$

An iterative procedure was proposed in [19] to solve (3), which updates W' and F alternatively. W' is updated with the stochastic gradient descent (SGD) approach at each iteration, while F is updated by computing K-truncated SVD after training every t epochs.

2.3. The approach with quantization function

The quantization function is also used to model the effect of clustering in (2). Suppose we have a quantization function Q_i for each W_i and C_i, where $Q_i(x) = \text{argmin}_{c\in C_i}|x - c|$. Then, the quantization functions can be plugged into the neural network directly. The optimization problem for quantization (2) is converted to:

$$\begin{aligned} &\min_{W,C} L(Q_1(W_1), Q_2(W_2), \dots, Q_m(W_m)), \\ &\text{s. t.} \quad C = [C_1, C_2, \dots, C_m] \in \mathbb{R}^{K\times m}, \\ &\forall i, j, \ Q_i(W_{i,j}) = \text{argmin}_{c\in C_i}|W_{i,j} - c| \ . \end{aligned} \tag{4}$$

Once the cluster centers C is given, W can be optimized by SGD. However, the $\frac{\partial Q_i(W_{i,j})}{\partial W_{i,j}}$ is zero almost everywhere, which makes $\frac{\partial L}{\partial W_{i,j}} = 0$ and training neural network is infeasible. A common solution to this is a so-called straight-through estimator (STE) technique [6], which approximates $\frac{\partial L}{\partial W_{i,j}}$ with $\frac{\partial L}{\partial Q_i(W_{i,j})}$.

The remaining problem is how to choose C. Let L be the bit length for quantization, i.e. $K = 2^L$. With LQ-Net [9], which is an effective DNN quantization scheme with quantization function, a quantizer basis $v_i \in \mathbb{R}^L$ and an encoding matrix $B_i \in \{-1, 1\}^{n_i\times L}$ for each vector W_i are found at each iteration to minimize $\|B_i v_i - W_i\|_2^2$. Then, for each quantization function Q_i vector $C_i = B^* v_i$, where matrix $B^* \in \mathbb{R}^{K\times L}$ contains all vectors from set $\{-1, 1\}^L$.

3. DP-Nets for DNN compression and acceleration

In this section, we first propose the dynamic programming based algorithm for the scalar clustering problem. Then, we present its applications to improve the aforementioned two approaches for DNN quantization. At last, a custom accelerator is proposed to speed up the neural network compressed with the DP-Nets.

3.1. A DP based algorithm for scalar clustering

As we know, the K-means clustering problem is NP-hard for general vector data. Therefore, the global optimum for the vector quantization cannot be found in reasonable time. Nevertheless, we find out that the optimal solution of the scalar K-means clustering can be obtained in polynomial time, based on the following Theorem 1.

Theorem 1. *Let $x_1 \le x_2 \le \cdots \le x_N$ be N scalars which need to be clustered into K classes. The clustering result is expressed as an integer index set $p = \{p_1, p_2, \dots, p_N\}$, $1 \le p_i \le K$, which means x_i belongs to the p_ith cluster. If the K-means clustering is to minimize*

$$g(p, c) = \sum_{i=1}^{N} (x_i - c_{p_i})^2 \ , \tag{5}$$

s.t. $c = \{c_1, c_2, \dots, c_K\}$, $c_1 < c_2 < \cdots < c_K$,

an optimal solution p satisfies: $1 = p_1 \le p_2 \le \cdots \le p_N = K$.

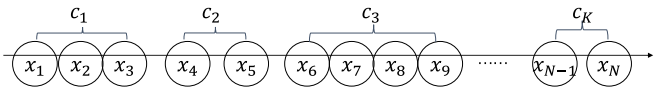

Fig. 3. Illustration of the optimal solution of clustering N scalars into K classes.

Proof. Suppose the ascending array $c = \{c_i\}$ stands for the cluster centers for the optimal solution. Let $c_0' = -\infty$, $c_1' = (c_1 + c_2)/2$, $c_2' = (c_2 + c_3)/2$, $\cdots$, $c_{K-1}' = (c_{K-1} + c_K)/2$, $c_K' = \infty$. We can construct a clustering solution by setting $p_i = j$, for any x_i satisfying $c_{j-1}' \le x_i < c_j'$ (meaning c_j is the closest cluster center for x_i). Therefore, this solution minimizes $g(p, c)$ and is an optimal solution satisfying $1 = p_1 \le p_2 \le \cdots \le p_N = K$. □

Theorem 1 infers that by suitable interval partition we can get the optimal K-means clustering of the weights (see Fig. 3). Let $x_1 \le \cdots \le x_N$ be the sorted weights for quantization, and $G_{n,k}$ be the minimum loss for clustering the first n weights into k clusters. Based on Theorem 1, we have

$$G_{n,k} = \min_{k-1\le i<n} \{G_{i,k-1} + h(i+1, n)\}, \ 1 < k \le K, \ k \le n \le N, \tag{6}$$

where $h(l, q) = \min_c \sum_{i=l}^{q} (x_i - c)^2$, meaning the minimum clustering error (loss) for clustering $x_l, x_{l+1}, \dots, x_q$ to one class. For the trivial situation with $k = 1$, $G_{n,1} = h(1, n)$. Now, we consider how to obtain the optimal clustering corresponding to $G_{n,k}$ for the situations with $k > 1$. Following (6), we need to enumerate all possible i which represents the largest index of scalar not belonging to the kth cluster. The minimum loss for clustering (i.e. quantization error) includes two parts: the minimum loss for clustering $x_1, \dots, x_i$ into $k - 1$ clusters, i.e. $G_{i,k-1}$, and the minimum quantization error that quantizing $x_{i+1}, \dots, x_n$ into a single value, i.e. $h(i + 1, n)$. The latter part can be easily calculated, and the mean of scalars should be the cluster center (quantized value). So,

$$\begin{aligned} h(l, q) &= \sum_{i=l}^{q} (x_i - \frac{1}{q - l + 1} \sum_{j=l}^{q} x_j)^2 \\ &= \sum_{i=l}^{q} x_i^2 - \frac{1}{q - l + 1} (\sum_{i=l}^{q} x_i)^2. \end{aligned} \tag{7}$$

For clustering, we need to know the optimal solution $\{c_i\}$, instead of the clustering error. We can use an auxiliary array z to depict the optimal clustering obtained by solving (6):

$$z_{n,k} = \text{argmin}_{k-1\le i<n} \{G_{i,k-1} + h(i + 1, n)\} + 1. \tag{8}$$

$z_{n,k}$ is the index of the first scalar in the last class, when the first n scalars in $\{x_i\}$ are clustered into k classes optimally. During the recursive procedure of solving (6) we can get the $z_{n,k}$ values. And, for example, with $z_{N,K}$ the last cluster center c_K can be obtained.

The core idea of dynamic programming is breaking a complicated problem down into simpler sub-problems in a recursive manner [22]. From the above discussion, we see that (6) reflects the optimal substructure for solving the scalar K-means clustering problem, and (8) guides us to find an optimal solution. Along with (7) and other dynamic programming skills, we derive Algorithm 1 for optimally solving the scalar quantization problem.

Based on Theorem 1, the related derivation (6) through (8), and the principle of dynamic programming, we can prove the optimality of the DP based algorithm for scalar quantization, i.e. Theorem 2.

Algorithm 1 DP based scalar quantization

Input: N scalars $x_1 \le x_2 \cdots \le x_N$, number of clusters K.
Output: The K cluster centers in the optimal solution.

```
Define two N × K arrays G and z.
for i ← 1 to N do
    Pre-compute h(j, i) for 1 ≤ j ≤ i based on (7).
    G_{i,1} ← h(1, i),   z_{i,1} ← 1.
    Calculate G_{i,k} and z_{i,k} based on (6) and (8) for 1 < k ≤ K.
end for
n ← N
for i ← K downto 1 do
    c_i ← (Σ_{j=z_{n,i}}^{n} x_j)/(n − z_{n,i}).
    n ← z_{n,i} − 1.
end for
return c_1, c_2, ⋯ , c_K.
```

Theorem 2. *The DP based algorithm for scalar quantization (Algorithm 1) obtains the optimal solution for the K-means clustering problem of scalar data.*

For Step 3 of Algorithm 1, we just need enumerate j from i to 1, and use two variables to store $\sum_{k=j}^{i} x_k^2$ and $\sum_{k=j}^{i} x_k$. Then, with (7) we can obtain $h(j,i)$ for all $j < i$, with a time complexity of $O(i)$ for a given i. The time complexity of Step 5 is $O(iK)$ for a given i, since (6) needs $O(i)$ time for calculating each $G_{i,k}$ and $z_{i,k}$ with the pre-computed $h(j,i)$. This derives that the time complexity of Algorithm 1 is $O(N^2K)$, where N is the numbers of scalars. We will make sure that N is not very large to save computation, when applying it to the weight quantization. It should be pointed out that, Algorithm 1 can be easily extended to other kinds of clustering problem, such as that if the L_2 norm in the loss function (5) is replaced with L_1 norm.

3.2. DP assisted approach with regularization

Inspired by Deep K-means [19], we propose a DP assisted approach with regularization (called DPR) for DNN compression, which directly optimizes the Lagrangian function of (2) during the training process with the help of the proposed DP based algorithm. The problem is an unconstrained optimization:

$$\min_{W,C}\{ L(W) + \lambda \sum_{i=1}^{m}\sum_{j=1}^{n_i} \min_{1\le k\le K} (W_{i,j} - C_{i,k})^2\}, \tag{9}$$

where λ is the Lagrange multiplier. After solving it, we can obtain a clustering-friendly network (an example is shown in Fig. 2(b)).

The problem is solved through alternatively optimizing W and C. After every t epochs of SGD based optimization of W, we optimize C by solving a scalar K-means clustering with the DP based algorithm (Algorithm 1). In practice, for an fully-connected (FC) layer with $n_{fc} \times m_{fc}$ weights, we divide them into n_{fc} parts (each contains $n = m_{fc}$ weights). We cluster the weights row by row. For a convolutional layer with $n_{conv} \times m_{conv} \times h_{conv} \times w_{conv}$ weights, we divide the weights into n_{conv} parts and each part contains $n = m_{conv} \times h_{conv} \times w_{conv}$ weights. In this way, the number of weights on which we do clustering will be no more than 10000 for most mainstream DNNs. So, the computational time for the DP based algorithm is affordable. This fine-grained scalar clustering improves the accuracy if compared with clustering all weights as a whole, and just induces negligible drop of compression ratio. Compared with Deep K-means [19], we do not relax the original optimization problem and thus may achieve better accuracy.

The DPR is also applicable to the robust DNN models. We just need to add the clustering error, i.e. the minimum loss $G_{N,K}$, as a regularization term to the loss function optimized during the training process. We consider the state-of-the-art TRADES (TRadeoff-inspired Adversarial DEfense via Surrogate-loss minimization) model for robust DNN [15]. The new formulation for training becomes:

$$\begin{aligned}\min_{W,C}\{\ & L(f(X;W),Y) + \\ & \max_{X'\in\mathbb{B}(X,\epsilon)} \gamma L(f(X;W), f(X';W)) + \\ & \lambda \sum_{i=1}^{m}\sum_{j=1}^{n_i} \min_{1\le k\le K} (W_{i,j} - C_{i,k})^2\ \}\quad ,\end{aligned} \tag{10}$$

where $f(X;W)$ is the output vector of learning model given parameters W, L denotes the cross-entropy loss function, $\mathbb{B}(x,\epsilon)$ represents a neighborhood of x: $\{x' : \|x' - x\|_2 \le \epsilon\}$, and γ is a regularization parameter to trade off between accuracy and robustness. With this formulation, we can train a clustering-friendly robust model.

3.3. DP assisted approach with quantization function

Based on the formulation (4), we propose a DP assisted approach with quantization function (called DPQ). The core idea is performing DP based algorithm to find an accurate quantizer, i.e. obtaining the optimal C_i to minimize $\|Q_i(W_i) - W_i\|_2^2$. This is the standard scalar K-means clustering problem. Then, we solve (4) directly.

We do not consider the constraints for quantizer in LQ-Net [9], and thus search in a larger parameter space. During the solution of (4), we do not perform the proposed DP based algorithm at each iteration to save computation. Instead, it is executed every t epochs while for other iteration we perform the Lloyd's algorithm for clustering. Thanks to the small perturbation of weights incurred by SGD, the cluster centers C in last iteration is a good initial guess of Lloyd's algorithm. The fine-grained scalar clustering strategy in last subsection is also used to improve the runtime efficiency.

This approach is also applicable to the robust DNN model. We just need to plug the quantization function into the loss function for robust model (such as TRADES) to derive the optimization problem:

$$\begin{aligned}\min_{W,C}\{ & L(f(X;Q(W)),Y) \\ & + \max_{X'\in\mathbb{B}(X,\epsilon)} \gamma L(f(X;Q(W)), f(X';Q(W)))\}.\end{aligned} \tag{11}$$

And, the STE technique is used during the training, for either the normal model or robust model.

3.4. Inference acceleration with DP-Nets

There are existing work on accelerating neural network on FPGA [23] and CPU [24]. In [23], the DNN model is compressed by scalar quantization so that it becomes small enough to be stored in SRAM, thus increasing the speed of memory access. In addition to this benefit, specific technique can be developed for more inference acceleration.

In the inference stage the major computation can be decomposed as matrix–vector multiplications, for the FC layer or the convolutional layer. Each matrix–vector multiplication consists of many vector dot products. For simplicity, we just consider accelerating the vector dot product.

For two n-dimensional vectors a, b, normally we need to perform n additions and n multiplications to make the dot product. With weight sharing, one vector, say b, is presented as $[c_{p_1}, c_{p_2}, \ldots, c_{p_n}]^T$, where c is a K-dimensional vector and p is the n-dimensional index vector ($1 \le p_i \le K$). Then, we have $a^T b = \sum_{k=1}^{K} c_k(\sum_{i\in S_k} a_i)$, where S_i is the set of indices j for which $p_j = i$, and we only need to perform K multiplications by distributive property. Suppose an addition and a multiplication costs t_a and t_m time, respectively. This brings a speedup ratio of $\frac{n(t_a+t_m)}{nt_a+Kt_m}$. Notice n is usually much larger than 1000, while K is no more than 16 in our experiments (i.e. $n \gg K$), and multiplication is executed slower than addition ($t_m > t_a$). Thus, significant speedup can be expected if the DNN model compressed with DP-Nets is used for inference.

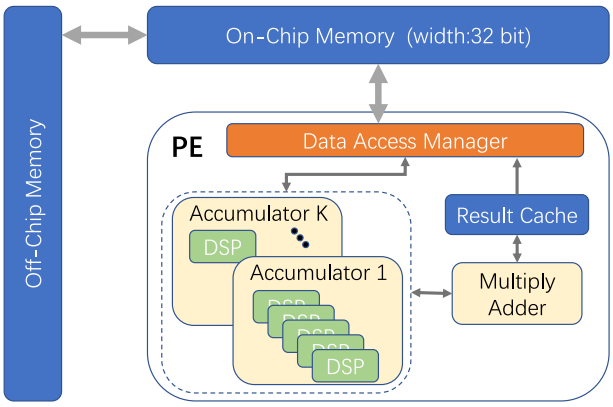

Fig. 4. The design of matrix–vector multiplication accelerator on FPGA for verifying the performance improvement brought by DP-Nets.

We design a simple matrix–vector multiplication accelerator with FPGA to verify the performance improvement brought by DP-Nets, as shown in Fig. 4. The Processing Element (PE) is composed of a Data Access Manager, an Accumulator Set, a Multiply-Adder, and a Result Cache. The On-Chip Memory is a 32-bit wide RAM for storing matrix and vector data. The execution process of the PE is to accumulate the vector elements corresponding to the elements with the same value in each row of the parameter matrix, and finally perform the floating-point multiplication operation on these accumulation results to complete the row vector multiplication column vector operation. To make better use of the parallel computing power of matrix–vector multiplication, PE is composed of K accumulators, and each accumulator performs the accumulation operation in the calculation process of different row vectors of the matrix. DP-Nets improve the computational performance of the model by greatly reducing the multiplication operations in the vector dot product process. Therefore, after the accumulator completes the accumulation operation of each row, only a small amount of multiplication and addition operations are needed to obtain the final matrix–vector multiplication result.

4. Experimental results

In this section, we present the experimental results to demonstrate the effectiveness of the proposed DPR and DPQ approaches for DNN compression. We first conduct experiments with CIFAR-10 dataset [25] for compressing the normal DNN models in [5,11], and the Wide ResNet [26]. The results show the benefit of using the proposed DP based algorithm to replace the Lloyd's algorithm, and the advantage of the proposed approaches over the counterparts. Then, the experiments are carried out with the ImageNet dataset, for compressing GoogLeNet [27], ResNet-18 [28] and MobileNet-V2 [29]. Then, the experiments on compressing robust DNN models are presented. Finally, we show the result of the custom accelerator for DP-Nets on FPGA.

The proposed approaches are implemented with Python 3.6. The DPR approach has two hyperparameters: λ, the regularizer factor in (9); t, the clustering frequency during training. We choose $\lambda = 100$ for all experiments. The value of t varies for different datasets because it affects the number of training epochs to reach convergence. The DPQ approach has only one hyperparameter t, which is set to 5 for all experiments. The training and inference are conducted on PyTorch. In all experiments, the compression ratio (CR) of the proposed approaches is obtained with (1) and then rounded to an integer [30].

4.1. Compressing normal models on CIFAR-10

The TT-Conv model [11] contains six convolutional layers and one fully-connected (FC) layer. The authors of TT-Conv used tensor train decomposition to compress the convolutional layer by 4X which makes the inference accuracy decreases by 2%. In [5], a FreshNet approach is proposed to quantize the weights of a DNN model's weights on the frequency domain, based on the observation that the learned convolutional weights are smooth and low-frequency. It achieves a CR of 16, with a 6.51% drop of inference accuracy. In [19], Deep K-means is used to compress the TT-Conv model and FreshNet model, which achieves less accuracy loss with same compression ratio.

Table 1
The results of compressing the models in [11] and [5]. Δ means the change of inference accuracy compared to the pretrained model. The inference accuracy of the pretrained models we obtained for [11] and [5] are 91.45% and 87.51%, respectively.

TT-Conv model [11]			FreshNet model [5]		
Approach	CR	Δ(%)	Approach	CR	Δ(%)
TT Decomposition [11]	4	−2.00	Hashed Net [4]	16	−9.79
Deep K-means [19]	2	+0.05	FreshNet [5]	16	−6.51
Deep K-means [19]	4	−0.04	Deep K-means [19]	16	−1.30
LR (3 bits)	10	−0.87	LR (2 bits)	16	−0.76
DPR (3 bits)	10	+0.31	DPR (2 bits)	16	**−0.57**
DPQ (3 bits)	10	+0.22	DPQ (2 bits)	16	−1.56

We first test the proposed DPR and DPQ with these two models. We choose $t = 20$ for DPR and use SGD with cosine annealing for training. The learning rate reduces from 0.05 to 0.01 during the training. The inference accuracy of the pretrained models we obtained for TT-Conv and FreshNet model are 91.45% and 87.51%, respectively. They are higher than those reported in [19]. For the TT-Conv model, we train 300 epochs. Because the FreshNet model is prone to overfitting [5], we just train 150 epochs and obtain a better result than training 300 epochs. The number of training epochs used for our DP assisted approaches is consistent with the pretrained model. We also use a variant of DPR called LR as a baseline, which employs Lloyd's algorithm instead of DP to do clustering during training. The other details of LR are consistent with DPR. The experimental results of DPR, DPQ, and the baselines are listed in Table 1.

The results in Table 1 demonstrate two important phenomena. Firstly, the inference accuracy of the model compressed by DPR is always higher than that by LR, which means that the proposed DP based algorithm is helpful to improve the accuracy of compressed model. Secondly, the models compressed with our DPR performs the best, and even can exhibit better accuracy than the pretrained uncompressed model. The possible reason is that the constraint of quantization may suppress overfitting, like what a L_2-norm regularized factor often behaves. Therefore, we believe that performing a suitable weight quantization cannot only reduces the size of model, but also improve the performance of DNN.

Compared with Deep K-means [19], the model compressed by DPR exhibits much higher accuracy with same or larger CR. For the proposed DPQ, it performs the same as DPR for the TT-Conv model, but worse for the FreshNet model. This is possibly caused by the network architecture of FreshNet and the small size of dataset. FreshNet has very simple structure, which makes the overfitting easy to happen. We will show the experiments with more popular networks (GoogLeNet, ResNet-18, and MobileNet-V2) and the larger dataset (ImageNet) in the following subsections.

Compared to DPQ, DPR has an additional hyperparameter λ. We conducted an experiment to study the sensitivity of DPR to it, with the experimental results plotted in Fig. 5. From the figure we see that DPR is not very sensitive to the λ. The phenomenons mentioned previously still exist. DPR always produces higher accuracy than LR under the same configurations in all experiments. And, most DPR models with 3-bit representation performs better than their pretrained models. This shows that a clustering-friendly network with appropriate number of clusters may have better generalization than normal network, which make our method meaningful even when compression is not needed.

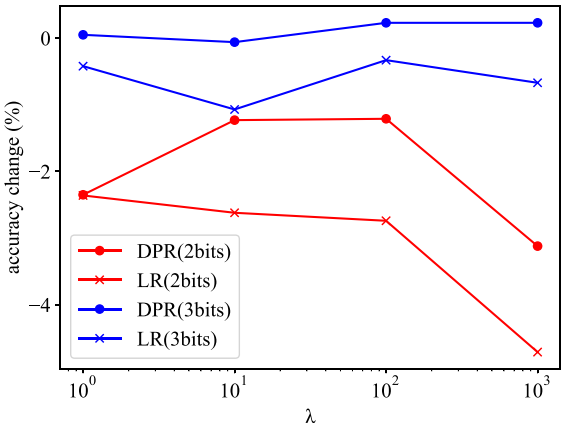

Fig. 5. The results of compressing TT-Conv with DPR. The accuracy change for varied λ with $t = 100$.

Table 2
The results of compressing GoogLeNet. Δ-top1 and Δ-top5 are the changes of top1-accuracy and top5-accuracy compared to the pretrained model, respectively. The top-1 accuracy and top-5 accuracy of the pretrained model are 69.78% and 89.53%..

Approach	CR	Δ-top1 (%)	Δ-top5 (%)
Deep K-means [19]	4	−1.95	−1.14
DPR (3 bits)	10	−1.56	−0.88
DPR (4 bits)	7	+0.30	+0.20
DPQ (3 bits)	10	+0.04	+0.05
DPQ (4 bits)	7	+1.85	+1.02

An additional experiment is carried out, following that was done in [19]. We combine the proposed approach with the pruning and Huffman coding to compress the Wide ResNet [26] on CIFAR-10. The sparsity for each layer and the pruning method are the same as those in [19], where a total CR=47 is achieved with 2.23% drop of inference accuracy and CR=50 is achieved with 4.49% drop of accuracy. We use DPR to replace Deep K-means for 2-bit quantization, resulting in CR of **77** with only 2.94% accuracy drop.

4.2. Compressing normal models on ImageNet

The proposed approaches are first tested on GoogLeNet trained on the ImageNet ILSVRC2012 dataset. We use the PyTorch official model as a pretrained model, whose top-1 accuracy is 69.78% and top-5 accuracy is 89.53%. We quantize the all convolutional layer and fully-connected layer. Our experiment settings are consistent with the PyTorch example[1] of ImageNet except the number of training epochs. We just train 30 epochs and make the learning rate decay 10X at the 20th and 25th epoch respectively, because the pretrained model provides a good initial solution. The learning rate is initialized at 0.001. We choose $t = 3$ for DPR because three epochs are enough for SGD to converge when the cluster centers are fixed. The results are listed in Table 2. It shows that the proposed approaches result in better accuracy than Deep K-means with **2.5X** larger compression as well. Similar to those for TT-Conv, the models compressed by our DPR and DPQ even exhibit better inference accuracy than the pretrained uncompressed model. While comparing DPR and DPQ, we see that the latter performs remarkably better than the former.

Then, we conduct the experiment of compressing ResNet-18 with ImageNet dataset. The proposed approaches are compared with LQ-Net and other approaches, whose results are obtained from [9]. For fairness, we quantize all the convolutional layer and fully-connected layer except the first and last layers and do not employ a pretrained model, the same as in [9]. The results are listed in Table 3. From it we see that DPQ surpasses the others when compressing ResNet-18 to more than 2 bits. For 2-bit quantization of ResNet-18, the accuracy of the model compressed with DPQ is comparable to that compressed with LQ-Net. And for the quantization with 4 bits, the accuracy of the model compressed with DPR is also better than that with LQ-Net. This shows that the proposed approach, especially DPQ, has superior or comparable performance of compression to the recent LQ-Net approach.

Table 3
The results of compressing ResNet-18. acc-top1 and acc-top5 are the top1-accuracy and top5-accuracy on ImageNet, respectively. The top-1 accuracy and top-5 accuracy of the pretrained model are 70.01% and 89.18%, respectively.

Approach	CR	acc-top1 (%)	acc-top5 (%)
TTQ (2 bits) [31]	16	66.6	87.2
ADMM (2 bits) [32]	16	67.0	87.5
LQ-Net (2 bits) [9]	16	68.0	88.0
LQ-Net (3 bits) [9]	10	69.3	88.8
LQ-Net (4 bits) [9]	7	70.0	89.1
DPR (2 bits)	16	63.0	84.5
DPR (3 bits)	10	69.2	88.6
DPR (4 bits)	7	70.3	89.5
DPQ (2 bits)	16	67.7	87.9
DPQ (3 bits)	10	**69.8**	**89.3**
DPQ (4 bits)	7	**70.5**	**89.6**

Finally, we also test the performance of proposed approach for compressing MobileNet-V2. The top-1 accuracy of the pretrained model is 71.88%. By applying DPQ, the top-1 accuracy drop of a 4-bit MobileNet is only 0.46%, while the model is compressed by about 6X.

4.3. Compressing robust models

The proposed DPR and DPQ are extended to compress the robust DNN models. They are denoted by DPR$^+$ and DPQ$^+$ respectively, and train the compressed model through solving (10) and (11) based on the TRADES technique [15]. The training settings and evaluating settings are consistent with the public codes[2] of TRADES . We choose $t = 5$ for DPR$^+$ because their training epochs are small. We first test a small network proposed by [33], which consists of four convolutional layers and three FC layers. We called it SmallCNN and train it on the MNIST dataset. The pretained model achieves 99.54% accuracy on normal testing data and 96.91% accuracy under a powerful attack algorithm named PGD (projected gradient descent) [34]. Then, we train a robust ResNet-18 model on CIFAR-10, which achieves 92.44% accuracy on normal testing data and 46.74% accuracy under the PGD attack.

Because there is few work on compressing a robust model with quantization approach, we consider a baseline called DP$_0$ for comparison. DP$_0$ directly quantizes the weights of the pretrained model with the proposed DP based algorithm to realize compression. Table 4 shows the accuracy drop of different compressed robust models obtained with DP$_0$, DPR$^+$ and DPQ$^+$. From the table, we see that DP$_0$ employing the simple weight-clustering quantization cannot preserve the accuracy of the robust model. It results in the accuracy drop on natural example and adversarial example up to 22.6%. With the proposed DPR$^+$ and DPQ$^+$ approaches, the accuracy drop caused by the compressed model is remarkably reduced (no more than 4.81%). This is due to the proposed formulations (10) and (11) for training the compressed robust model. On the other hand, we find out that DPR$^+$ performs better than DPQ$^+$. Although they have comparable accuracy drop on the natural examples, DPR$^+$ produces the model with 2% less accuracy drop than DPQ$^+$ on the adversarial example. With 2-bit quantization, DPR$^+$ obtains a compressed robust ResNet-18 model which exhibits **less than 3**% accuracy drop on both natural and adversarial examples.

[1] https://github.com/pytorch/examples/tree/master/imagenet

[2] https://github.com/yaodongyu/TRADES

Table 4
The results of compressing the robust SmallCNN (on MNIST) and ResNet-18 (on CIFAR-10). Δ_{nat} and Δ_{adv} are the accuracy changes for natural images and adversarial images compared to the pretrained model, respectively. The pretrained SmallCNN achieves 99.54% accuracy on natural images and 96.91% accuracy on adversarial images. The pretrained ResNet-18 achieves 92.44% accuracy on natural images and 46.74% accuracy on adversarial images.

Approach	CR	Model	$\Delta_{nat}(\%)$	$\Delta_{adv}(\%)$
DP_0 (2 bits)	14	SmallCNN	−0.44	−5.90
DPR^+ (2 bits)	14	SmallCNN	**−0.11**	**−2.38**
DPQ^+ (2 bits)	14	SmallCNN	−0.37	−4.44
DP_0 (2 bits)	16	ResNet-18	−22.48	−22.60
DPR^+ (2 bits)	16	ResNet-18	−0.98	**−2.77**
DPQ^+ (2 bits)	16	ResNet-18	+0.40	−4.81

Table 5
The time of a matrix–vector multiplication on FPGA.

Matrix source	Matrix size	Algorithm	Time (ms)	Speedup
FC layer	1000 × 1024	Baseline	61.5	–
		Ours	11.9	**5.1X**
Conv layer	384 × 1728	Baseline	39.8	–
		Ours	7.60	**5.2X**

4.4. Custom accelerator

To demonstrate the specific technique proposed in Section 3.4, we test the runtime of DNN inference with the DP-Nets deployed on FPGA.

We choose the ZC706 evaluation board as the deployment platform, which works at a clock frequency of 100 MHz. ZC706 contains a wealth of computing and storage resources, including 218600 Look-Up Tables (LUTs), 437200 Flip-Flops (FFs), 545 unit of 36k Block RAM (BRAM), 900 DSP48E units. Our accelerator is implemented in RTL language and compiled and deployed with the help of Vivado tools. Because our method greatly reduces the multiplication operations and greatly compresses the matrix size, only about 10% of the DSP resources and 38% of the LUT resources are used.

The experiments involve an FC layer and a convolutional layer from GoogleNet compressed by weight sharing with 4-bit representation, and they are multiplied by random vectors. For the convolutional layer with $n_{conv} \times m_{conv} \times h_{conv} \times w_{conv}$ weights, we first reshape it to an $n_{conv} \times m_{conv} h_{conv} w_{conv}$ matrix. The baseline approach for matrix–vector multiplication takes in the matrix represented by 32-bit floating-point numbers stored in BRAM and/or distributed RAM, and the random vector stored in distributed RAM (ROM) to facilitate fast access. It adopts a pipelined design technique, and performs floating-point multiply-add operation by rows. The operation is implemented with the Floating-point (7.1) IP Core provided by Xilinx, based on DSP Slice Full Usage. For our approaches based on DP-Nets, the quantized matrix and compressed matrix is small enough to be stored in the distributed RAM synthesized by LUTs. And, the technique in Section 3.4 is implemented to perform the matrix–vector multiplication. The experimental results are listed in Table 5. It shows that the custom accelerator is at least 5X faster than the baseline approach without compression. In this experiment, 4-bits representation means that 16 clusters are used for preserving the accuracy of GoogleNet. For other DNN models, the weights can be clustered into fewer classes, which in turn would lead to larger compression ratio and more inference acceleration on FPGA.

5. Conclusions

A dynamical programming based algorithm for scalar clustering and two DNN compression schemes, DPR and DPQ, are proposed. DPR includes training a clustering-friendly network with a formulation including the regularization item and the DP based algorithm obtaining the optimal solution of scalar weight clustering. DPQ includes using the DP based algorithm to find a better quantizer and training/compressing the DNN with a formulation with quantization function. DPR and DPQ are also extended to compress robust DNNs, through a combination with the TRADES approach [15]. Lastly, a technique for inference acceleration is proposed.

Exhaustive experiments have been carried out to show the advantages of the proposed approaches over existing counterparts for compressing normal and robust DNNs. Experimental results also show, the DPR approach performs better for robust models while the DPQ approach is more suitable for large DNN models. The experiment on FPGA shows that it brings 5X speedup to the computation associated with FC layers and convolutional layers.

CRediT authorship contribution statement

Dingcheng Yang: Writing – Original Draft, Methodology, Software. **Wenjian Yu:** Writing - review & editing, Supervision, Funding acquisition, Resources. **Xiangyun Ding:** Software, Validation. **Ao Zhou:** Software, Methodology, Writing - review & editing. **Xiaoyi Wang:** Funding acquisition, Supervision.

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.

References

[1] S. Han, J. Pool, J. Tran, W. Dally, Learning both weights and connections for efficient neural network, in: Proc. NIPS, 2015, pp. 1135–1143.
[2] Y. LeCun, J. Denker, S. Solla, Optimal brain damage, in: Proc. NIPS, 1990, pp. 598–605.
[3] G. Hinton, O. Vinyals, J. Dean, Distilling the knowledge in a neural network, 2015, arXiv preprint arXiv:1503.02531.
[4] W. Chen, J. Wilson, S. Tyree, K. Weinberger, Y. Chen, Compressing neural networks with the hashing trick, in: Proc. ICML, 2015, pp. 2285–2294.
[5] W. Chen, J. Wilson, S. Tyree, K. Weinberger, Y. Chen, Compressing convolutional neural networks in the frequency domain, in: Proc. SIGKDD, 2016, pp. 1475–1484.
[6] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, Y. Bengio, Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1, 2016, arXiv preprint arXiv:1602.02830.
[7] Y. Gong, L. Liu, M. Yang, L. Bourdev, Compressing deep convolutional networks using vector quantization, 2014, arXiv preprint arXiv:1412.6115.
[8] S. Han, H. Mao, W. Dally, Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding, 2015, arXiv preprint arXiv:1510.00149.
[9] D. Zhang, J. Yang, D. Ye, G. Hua, LQ-Nets: Learned quantization for highly accurate and compact deep neural networks, in: Proc. ECCV, 2018, pp. 365–382.
[10] R. Dai, L. Li, W. Yu, Fast training and model compression of gated RNNs via singular value decomposition, in: Proc. IJCNN, 2018, pp. 1–7.
[11] T. Garipov, D. Podoprikhin, A. Novikov, D. Vetrov, Ultimate tensorization: Compressing convolutional and FC layers alike, 2016, arXiv preprint arXiv:1611.03214.
[12] Y. Ma, R. Chen, W. Li, F. Shang, W. Yu, M. Cho, B. Yu, A unified approximation framework for compressing and accelerating deep neural networks, in: Proc. ICTAI, 2019, pp. 376–383.
[13] W. Yu, Y. Gu, Y. Li, Efficient randomized algorithms for the fixed-precision low-rank matrix approximation, SIAM J. Matrix Anal. Appl. 39 (3) (2018) 1339–1359.
[14] I. Goodfellow, J. Shlens, C. Szegedy, Explaining and harnessing adversarial examples, 2014, arXiv preprint arXiv:1412.6572.
[15] H. Zhang, Y. Yu, J. Jiao, E. Xing, L. El Ghaoui, M. Jordan, Theoretically principled trade-off between robustness and accuracy, in: Proc. ICML, 2019, pp. 7472–7482.
[16] S. Gui, H. Wang, H. Yang, C. Yu, Z. Wang, J. Liu, Model compression with adversarial robustness: A unified optimization framework, in: Proc. NIPS, 2019, pp. 1285–1296.
[17] T. Hu, T. Chen, H. Wang, Z. Wang, Triple wins: Boosting accuracy, robustness and efficiency together by enabling input-adaptive inference, in: Proc. ICLR, 2019.
[18] S. Lloyd, Least squares quantization in PCM, IEEE Trans. Inform. Theory 28 (2) (1982) 129–137.

[19] J. Wu, Y. Wang, Z. Wu, Z. Wang, A. Veeraraghavan, Y. Lin, Deep *k*-Means: Re-training and parameter sharing with harder cluster assignments for compressing deep convolutions, in: Proc. ICML, 2018, pp. 5363–5372.
[20] Z. Liu, M. Sun, T. Zhou, G. Huang, T. Darrell, Rethinking the value of network pruning, in: Proc. ICLR, 2019.
[21] H. Zha, X. He, C. Ding, M. Gu, H. Simon, Spectral relaxation for k-means clustering, in: Proc. NIPS, 2002, pp. 1057–1064.
[22] S. Dreyfus, A. Law, Art and Theory of Dynamic Programming, Academic Press, Inc., 1977.
[23] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M.A. Horowitz, W.J. Dally, EIE: Efficient inference engine on compressed deep neural network, in: Proc. ISCA, 2016, pp. 243–254.
[24] M. Sotoudeh, S.S. Baghsorkhi, C3-Flow: Compute compression co-design flow for deep neural networks, in: Proc. DAC, 2019, p. 86.
[25] A. Krizhevsky, G. Hinton, et al., Learning Multiple Layers of Features from Tiny Images, Technical Report, Citeseer, 2009.
[26] S. Zagoruyko, N. Komodakis, Wide residual networks, in: Proc. BMVC, 2016.
[27] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, in: Proc. CVPR, 2015, pp. 1–9.
[28] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proc. CVPR, 2016, pp. 770–778.
[29] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, L.-C. Chen, Mobilenetv2: Inverted residuals and linear bottlenecks, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 4510–4520.
[30] D. Yang, W. Yu, H. Mu, G. Yao, Dynamic programming assisted quantization approaches for compressing Normal and robust DNN models, in: Proceedings of the 26th Asia and South Pacific Design Automation Conference, 2021, pp. 351–357.
[31] C. Zhu, S. Han, H. Mao, W. Dally, Trained ternary quantization, in: Proc. ICLR, 2017.
[32] C. Leng, Z. Dou, H. Li, S. Zhu, R. Jin, Extremely low bit neural network: Squeeze the last bit out with ADMM, in: Proc. AAAI, 2018, pp. 3466–3473.
[33] N. Carlini, D. Wagner, Towards evaluating the robustness of neural networks, in: Proc. IEEE Symposium on Security and Privacy, 2017, pp. 39–57.
[34] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, A. Vladu, Towards deep learning models resistant to adversarial attacks, in: Proc. ICLR, 2018.