applied sciences

Article

Efficient Weights Quantization of Convolutional Neural Networks Using Kernel Density Estimation based Non-uniform Quantizer

Sanghyun Seo and Juntae Kim *

Department of Computer Engineering, Dongguk University, 30, Pildong-ro 1-gil, Jung-gu, Seoul 04620, Korea; shseo@dongguk.edu

* Correspondence: jkim@dongguk.edu; Tel.: +82-2-2260-3712

Received: 30 April 2019; Accepted: 20 June 2019; Published: 23 June 2019

Abstract: Convolutional neural networks (CNN) have achieved excellent results in the field of image recognition that classifies objects in images. A typical CNN consists of a deep architecture that uses a large number of weights and layers to achieve high performance. CNN requires relatively large memory space and computational costs, which not only increase the time to train the model but also limit the real-time application of the trained model. For this reason, various neural network compression methodologies have been studied to efficiently use CNN in small embedded hardware such as mobile and edge devices. In this paper, we propose a kernel density estimation based non-uniform quantization methodology that can perform compression efficiently. The proposed method performs efficient weights quantization using a significantly smaller number of sampled weights than the number of original weights. Four-bit quantization experiments on the classification of the ImageNet dataset with various CNN architectures show that the proposed methodology can perform weights quantization efficiently in terms of computational costs without significant reduction in model performance.

Keywords: weights quantization; kernel density estimation; Lloyd–Max quantizer; K-means clustering; convolutional neural networks

1. Introduction

In the field of image recognition, various types of deep learning models based on CNN have been proposed and achieved excellent results [1–5]. A typical CNN model consists of several convolution layers that extract features of input data and fully-connected layers that perform classification from feature maps. Through such a structure, CNN records superior performance exceeding human classifying ability, but there is trade-off. CNN consists of very deep layers to extract the features of input data, and the number of weights used in each layer is very large. In other words, despite its excellent performance, the deep learning model has high computational costs. For example, AlexNet has about 60 million parameters with 8 layers and VGGNet-16 consists of about 138 million parameters with 16 layers. As there is a lot of computation from input to output in these deep learning models, for a mobile or embedded device with insufficient memory space or power, it is difficult to distribute the deep learning model for various applications.

In order to efficiently train CNN architectures, which are composed of multiple layers, a graphics processing device capable of performing operations with a lot of weights and gradients is indispensably required. In contrast, when only the forward propagation is performed to apply the pre-trained CNN, the hardware restriction is relatively free. Nevertheless, forward propagation of CNN is also problematic for applications in mobile and embedded hardware devices because it requires a large amount of computation between input data and numerous weights. First, the manufacturing of high-performance embedded hardware capable of performing large-scale tasks requires relatively high production cost. Second, the high computational costs of the deep learning model make it difficult to apply the model in the real-time environment, and increase the energy consumption of the device. Therefore, it is important to reduce the computational costs in order to apply the CNN model with high performance in such an environment.

Weights quantization is one of the potential solutions to this situation. In the past, various neural network quantization methodologies have been studied to make digital hardware implementation easier [6]. The recently proposed quantization studies of deep learning model are not only focused on simplifying the hardware implementation but further reducing the computational costs while maintaining the performance of the original model. For example, in FPGA based hardware, it is possible to develop a more efficient deep learning model application device by reducing the number of multipliers required for the operation [7]. By reducing the bit size of the weights from the neurons to the neurons, the number of FPGA based hardware multipliers that actually perform calculations can be reduced. As a result, the power efficiency of the hardware can be increased, and the computational efficiency of real-time applications of the object recognition model in the embedded environment can be increased.

Figure 1 shows the general weights quantization process for the CNN model. In this process, clustering for the pre-trained weights of CNN is performed first and then a codebook of weights is created. The size of the codebook is determined by the target bit-width of quantization. When a codebook is generated through a quantizer, the original weights are mapped to an index of the codebook having the closest value to the original weights. Through this process, we can obtain quantized weights that have smaller bandwidth compared to that of 32-bit float type weights. This process is referred to as weights quantization, and it enables more efficient application of various deep learning models. Recent studies related to weights quantization attempt to restore the performance of the deep learning model in the quantization process, including the retraining process as well as the quantization [8,9].

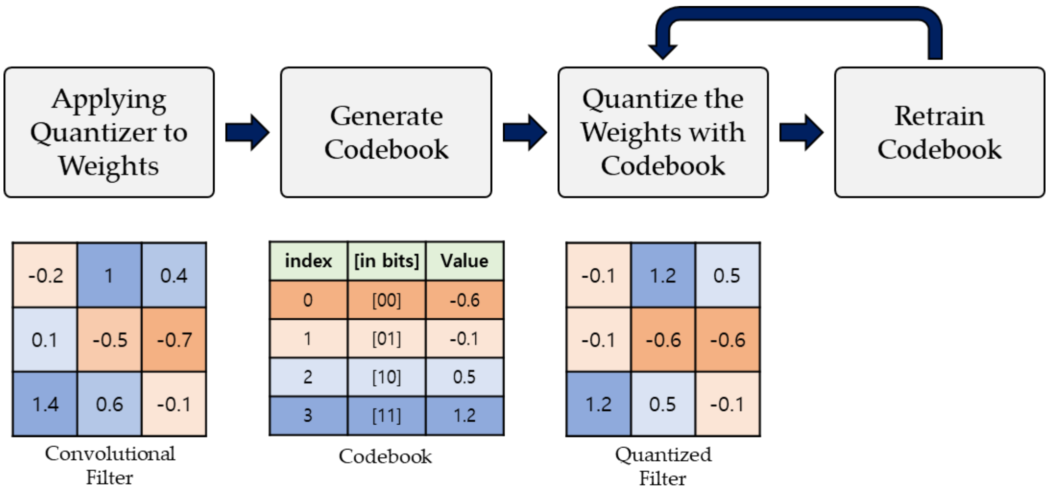

Figure 1. Process of weights quantization in convolutional neural networks.

General weights quantization methodologies with retrain have some disadvantages. First, because codebook generation and retraining are usually performed repeatedly, it takes a lot of time to construct the final quantized model [8]. This problem is a limitation when we want to deploy the quantized model quickly. The existing approaches usually use non-uniform quantization rather than uniform quantization in order to minimize the loss of performance inherent in quantization, even though it involves retrain processes [8–10]. In general, non-uniform quantization methods including k-means are often applied to the weights quantization of neural networks [8–13]. However, the simple k-means have a problem because it takes a lot of time to reconstruct a codebook. Because the range of weights is not wide and weights have similar values, to perform distance calculations for all the weights is not efficient.

In this paper, we utilize kernel density estimation to obtain the probability density function of pre-trained weights of CNN. Then we perform sampling through the previously obtained probability density function, and non-uniform quantization is applied on those sampled data. The contribution of this paper is summarized as follows.

- The existing weights quantization methodology is inefficient in terms of computational costs. We propose a method that performs quantization more efficiently by using kernel density estimation and sampled data from the probability density function.
- A number of recent studies on weights quantization have focused on the recovery of lost performance through retrain phase. The proposed methodology can perform fast quantization suitable for this retrain phase and fast deployments.
- We compare the performance of CNN weights quantization using quantization methodologies such as uniform quantization, Lloyd–Max quantizer, and k-means clustering.
- Experiments on various CNN architectures such as AlexNet, VGGNet, and ResNet show that the proposed methodology can be widely applied to various CNN architectures.

2. Related Works

2.1. Weights Quantization of Deep Learning Model

Compression of the neural network requires a set of methodologies to reduce the size of the architecture and the weights of the neural network, while maintaining the original performance as much as possible. There are CNN architectures designed for use in mobile and embedded devices. First, SqueezeNet has similar accuracy to AlexNet, but the size of the weights is about 1/50 [14]. SqueezeNet proposes a fire module configured to the squeeze layer and the expansion layer utilizing a 1x1 convolution filter to design a tight architecture. Similarly, MobileNet allows forward propagation at a faster speed by applying a method such as a depth-wise separable convolution [15].

There are two main approaches in quantization of neural network weight. The first is to quantize the weights of neural networks in the training phase, the other is to quantize the weights of the trained neural networks. BinaryConnect is a typical way of doing quantization during the training phase [16]. BinaryConnect is a forward and backward propagation that enables the use of binarized weights. This method has the property of using the sign function to binarize the weights in the forward process and to hold the real-valued weights in the backpropagation process. XNOR-Net has attempted to use binary weights of neural networks for classifying large datasets such as ImageNet [17]. XNOR-Net improves the training and inference efficiency by binarizing weights and the input data, along with gradients. Based on these studies, DoReFa-Net uses binarized weights, binarized activation, and binarized gradients up to an arbitrary bit width. This method improved the performance of XNOR-Net using a sophisticated rounding mechanism [18].

On the other hand, deep compression is a typical method of quantizing the connection weights of neural networks learned [8]. Deep compression is a three stage pipeline that can reduce the memory requirements of the network from 35 to 49 times without degrading the accuracy by applying the quantization Huffman coding technique. The method first removes the value of the unaffected connection to zero and performs the quantization. After the retrain, the lost performance is restored and the quantization process is repeated a predetermined number of times. Finally, Huffman coding is applied to connection weights so that it is possible to efficiently store connection weights with a high frequency. Deep compression achieves high compression rates in this way while maintaining good performance. Entropy-constrained scalar quantization (ECSQ) was designed to quantize using the k-means clustering methodology to minimize loss of the neural network as well as the quantization error by applying a hessian matrix [19]. Incremental networks quantization (INQ) efficiently reduces the loss caused by quantization by performing three independent operations of weights partitioning, quantization, and retrain [9]. This methodology combines the benefits of quantization during training with the benefits of quantization after training compared to other quantization methods.

In addition, the quantization methodology to minimize the energy consumption of the model using fixed point weights has been proposed [20]. WAGE trains the model by quantizing all elements such as weights, activation, gradients, and error, which adjust the range or set of weights [21]. In addition, various methods of weights quantization of neural networks, including retraining and so on, are being studied [22–26]. Based on these studies, a quantization codebook has been applied to LCNN [27]. Through LCNN, the quantized weights are then stored in a codebook that can be designed to enable more efficient inference using a lookup table.

2.2. Scalar Quantization

Quantization generally means dividing the data into finite levels by transformations and assigning specific values to each level. Among them, scalar quantization is the most basic form of the quantization methods that maps a data to a specific value in the codebook. This scalar quantization is distinguished by the types of intervals in the codebook—uniform or non-uniform. Uniform quantization is a method using quantization intervals which are the same size within a certain range. For example, there is a method of setting the size of the quantization interval by dividing the minimum and maximum values for specific input data by the quantization level.

$$Q_{step} = (X_{max} - X_{min})/L \tag{1}$$

$$Q_{index}(X) = \lfloor (X - X_{min})/Q_{step} \rfloor \tag{2}$$

$$Q(X) = Q_{index}(X) * Q_{step} + \frac{Q_{step}}{2} + X_{min} \tag{3}$$

Quantization step-size Q_{step} can be calculated using the desired quantization level L and the maximum range of weights as in Equation (1). The quantized index $Q_{index}(X)$ of each weight can be obtained by calculating the number of quantization steps apart from the minimum value X_{min} of weights as shown in Equation (2). Finally, using Equation (3), the quantized vector $Q(X)$ can be obtained as the center value of each quantization interval. The actual quantization is done by replacing X with $Q(X)$. Uniform quantization allows very fast quantization for a given data, but there is a constraint that the distribution of input data must be uniform. In other words, when the distribution of input data is Gaussian distribution, Laplace distribution, or a special distribution such as a gamma distribution, the relative quantization error by using a uniform quantization method can become a loss of performance in CNN.

An alternative to this is non-uniform quantization. In non-uniform quantization the value is shifted in order to reduce the quantization error, and the intervals between quantization values are not uniform. Therefore, more sophisticated quantization with relatively low quantization error compared to uniform quantization can be expected. Representative non-uniform quantization is the Lloyd–Max quantizer and k-means clustering. Both of them try to minimize the mean squared error between the original value and the quantized value.

The Lloyd–Max quantizer adjusts the quantization value so that the area of the probability density function is the same for each quantization step. Optimized through this approach, the Lloyd–Max quantizer has a dense quantization step in areas with high probability density and, conversely, a wide quantization step in areas with low probability density.

$$\begin{aligned} D = \mathbb{E}\left[(x-Q(x))^2\right] &= \int_{-\infty}^{\infty} \mathbb{E}\left[(x-Q(x))^2\right] f(x) d(x) \\ &= \sum_{i=1}^{L} \int_{-b_{i-1}}^{b_i} \mathbb{E}[(x-q_i)^2] f(x) d(x) \end{aligned} \tag{4}$$

$$b_i = \frac{q_i + q_{i+1}}{2} \tag{5}$$

$$q_i = \frac{\int_{b_i}^{b_{i+1}} x f_x(x)dx}{\int_{b_i}^{b_{i+1}} f_x(x)dx} \tag{6}$$

The Lloyd–Max quantizer adjusts the values of boundary b_i and quantized value q_i so that the mean squared error between the original weights and the quantized weights are minimized, as shown in Equation (4). The two values in Equations (5) and (6) can be obtained by applying a Lagrange condition to Equation (4), $\frac{\partial D}{\partial b_i} = 0$, $\frac{\partial D}{\partial q_i} = 0$.

Another representative methodology for non-uniform quantization is the k-means clustering. Quantization through k-means clustering is one of the main methodologies for quantizing weights of CNN. The quantization using k-means sets the number of quantization bits to $\log_2 k$, and based on the result of clustering, the data of each cluster is mapped to the center value of the cluster.

$$\underset{C}{\operatorname{argmin}} \sum_{i=1,\ \mu^i \in M, c^i}^{k} \sum_{w_j^i \in W, c^i} ||\mu^i - w_j^i||^2 \tag{7}$$

Equation (7) is the objective of k-means clustering methodology. This method sets the size of k to 2^q where q is the number of quantization bits, and clusters the weights to minimize the mean square error between the μ^i, which is the centroid of cluster c^i and each weight w_j^i in the same cluster c^i. M is the set of cluster centroids and W is the set of all original weights.

By using non-uniform quantization methods, we can expect a relatively high performance of the quantized CNN model because the quantization is performed in the direction in which the quantization error is reduced. However, because both the Lloyd–Max quantizer and the k-means clustering method described above are based on iteration, there is a problem that the computation time is increased depending on the size of the input data.

2.3. Kernel Density Estimation

From a probabilistic point of view, data is a specification of one of the various possibilities of a particular random variable. As one piece of data is only a part of a random variable, a large number of data is needed to understand the overall appearance of the random variable. In general, the method of estimating the distribution characteristics of the original random variable from the distribution of observed data is called density estimation. Estimating the characteristics of a random variable X is equivalent to estimating the probability density function of that variable. For example, when there is a probability density function $f(X)$ of a random variable X, $f(X = a)$ refers to a relative likelihood that X has the value of a.

Density estimation methods can be divided into parametric and non-parametric methods. The simplest method of non-parametric density estimation is the histogram representation. This is a way of expressing the frequency of a given data divided by a specific bin. However, histograms have a discontinuity at the boundaries of the histogram bin, and this is a disadvantage in high dimensional data where the bins are different according to the size and the starting position.

Parametric density estimation uses the pre-defined model of the probability density function and estimates the parameters of the model from the data. The kernel density estimation is one of the parametric density estimation methods that solves the problem of the histogram method by using the kernel function. There are two general conditions for the kernel function. One is that it should be symmetric about the origin and non-negative, and the other is that the summation of the integral should be 1. For example, Gaussian, Epanechinkv, and Uniform are typical kernel functions.

$$f(x) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x_i - x}{h}\right) \tag{8}$$

In Equation (8), the h which controls the size of the neighborhood around x is the smoothing parameter. The function K is called the kernel which controls the weight given to the observations x_i at each point x based on their proximity. Therefore, it can control the shape of the probability distribution. The kernel function $f(\cdot)$ for the observed data x can make the probability density function by summing all these kernel functions and dividing them by the total number of the data. The estimated probability density function reflects the characteristics of the observed data. By using this function, it is possible to sample new data from a distribution of observed data.

3. Kernel Density Estimation based on Non-Uniform Quantization

3.1. Overview of the Proposed Quantization Process

The non-uniform quantization methodology is suitable for performing weights quantization of CNN because it optimizes to reduce quantization errors. However, one of the key features of deep learning models like CNN is the large number of weights. For k-means, the amount of computations increases proportionally as the number of weights increases. In particular, because the k-means is one of the NP-hard problems, it is difficult to proceed until convergence, and by placing an upper limit on optimization, the results of quantization with relatively low performance can be obtained by early stopping. Similarly, the Lloyd–Max quantizer is affected by the number of data when performing an integrated calculation in the actual computing environment. As the range of weights in the pre-trained deep learning model is relatively small and a lot of weights have similar values, it is inefficient to obtain quantized values for all weights to reflect the characteristics of the weights. This paper proposes a kernel density estimation based non-uniform quantization that can be more efficient in terms of computation time.

Figure 2 shows an overview of the suggested kernel density estimation based non-uniform quantization methodology. First, weights from a pre-trained CNN are used to perform the kernel density estimation to make a probability density function about the original weights. By using this probability density function, we can sample the data reflecting characteristics of original weights. At this point, the number of sampled data is smaller than the number of original weights, enabling more efficient quantization operations. Through sampled data, k-means quantization becomes more efficient in terms of computation. Similarly, the sampled data are used again for kernel density estimation and make a probability density function of data. The Lloyd–Max quantizer using the probability density function of sampled data is more efficient in the integral computation, because the number of sampled data used to calculate the area of the probability distribution function is much smaller than the number of original weights.

K-means use sampled data in the probability density function of the original data, and the Lloyd–Max quantizer performs the kernel density estimation on the sampled data and uses the probability function of the sampled data. The k-means and Lloyd–Max quantizer of the proposed methodology are separately performed. Therefore, each methodology independently generates a codebook to carry out quantization of the originals.

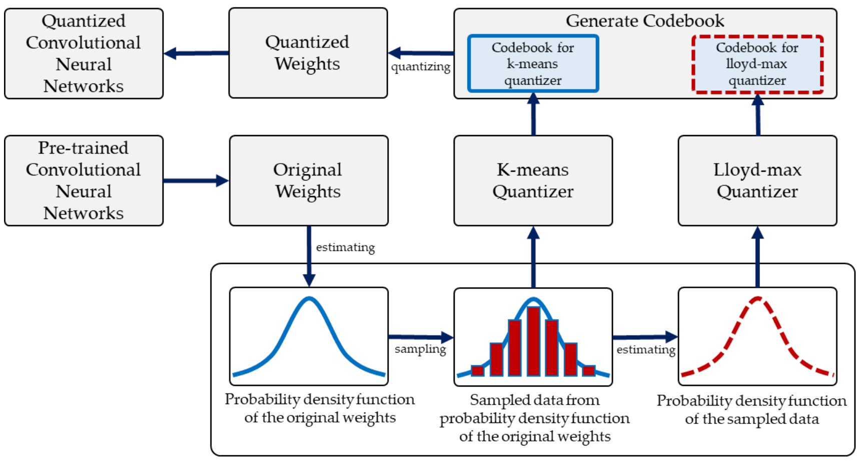
Quantized Convolutional Neural Networks
Quantized Weights
quantizing
Generate Codebook
Codebook for k-means quantizer
Codebook for lloyd-max quantizer
Pre-trained Convolutional Neural Networks
Original Weights
K-means Quantizer
Lloyd-max Quantizer
estimating
sampling
estimating
Probability density function of the original weights
Sampled data from probability density function of the original weights
Probability density function of the sampled data

Figure 2. Overview of weights quantization using the kernel density estimation.

3.2. Kernel Density Estimation based K-means Quantization (KDE-KM)

The proposed kernel density estimation based k-means quantization (KDE-KM) obtains the probability density function by performing the kernel density estimation from the original weights. The KDE-KM is then performed using sampled data from the probability density function that has the same characteristics of the original data, and it is possible to quantize more efficiently because it performs calculations on a much smaller number of data. Below is the pseudocode of the proposed KDE-KM methodology.

Algorithm 1 shows the proposed KDE-KM process. The proposed method estimates the PDF^o from the original weights W^o and then PDF^o samples a set of sampled data D^s by using the number of samples N, which is less than the number of original weights W^o. The proposed method can reduce the time complexity of quantization by using sampled data D^s. In this case, K is set to $2^{q_{bit}}$. After the clustering is finished, it assigns all the original weights W^o to a specific cluster c^i, and assigns μ^i, which is the cluster centroid of c^i as the quantization value to the original weights to get W^q.

Algorithm 1. KDE-KM

```
Input: $W^o, q_{bit}, N, E$
original weights $W^o$, target quantization bit $q_{bit}$, the number of sampling data $N$,
the number of maximum iteration $E$
Output: $W^q$
quantized weights $W^q$
Initialization: $K, C, M$
The number of cluster $K = 2^{q_{bit}}$, clusters $C = \{c^1, c^2, \dots, c^k\}$,
cluster centroids $M = \{\mu^1, \mu^2, \dots, \mu^k\}$
1:  $PDF^o \leftarrow KDE(W^o)$ // kernel density estimation using original weights
2:  $D^s \leftarrow PDF^o(N)$ //sampling $N$ data using probability density function
3:  for e = 1 to $E$ do
4:      $c^j \leftarrow \underset{c^j}{\mathrm{argmin}}\|\mu^j - d_i\|^2$, for $d_i \in D^s$        // assign $D^s$ to clusters in $C$
5:      for j = 1 to $K$ do
6:          $\mu^j \leftarrow \frac{1}{|c^j|}\sum_{i=1}^{|c^j|} d_i^j$, where $d_i^j \in D^s, c^j$     // update cluster centroids $M$
7:  $c^j \leftarrow \underset{c^j}{\mathrm{argmin}}\|\mu^j - w_i^o\|^2$, for all $w_i^o \in W^o$     // assign all original weights $W^o$ to a cluster
8:  for j = 1 to $K$ do
9:      for i = 1 to $|c^k|$ do
10:         $w_i^j \leftarrow \mu^j$, where $w_i^j \in W^o, c^j$        // quantize the original weights $W^o$
11:   $W^q \leftarrow W^o$                                      // set quantized weights $W^q$
12:   return $W^q$
```

3.3. Kernel Density Estimation Based Lloyd–Max Quantizer (KDE-LM)

The proposed kernel density estimation based Lloyd–Max quantizer (KDE-LM) builds the quantizer in a manner similar to the k-means using sampled data. KDE-LM minimizes quantization errors by repeatedly minimizing the difference in area separated by quantization values in the probability density function by adjusting the quantization values. At this point, if the number of weights increases, the computational costs for the correct integral computation also heavily increases. Therefore, by performing a kernel density estimation once again using relatively small number of sampled data, and quantizing using it, quantization can be performed more efficiently in terms of the number of operations. Below is the pseudocode of the proposed KDE-LM methodology.

Algorithm 2 shows the proposed KDE-LM process. Similar to KDE-KM, KDE-LM uses the KDE for original weights W^o. However, KDE-LM has a difference in that it performs KDE once more from sampled data D^s to get the PDF^s. The advantage of using sampled data D^s and the PDF^s is less computation costs when performing integral computation at line 7. The quantization is performed by assigning the quantization value to the original weights through line 9.

Algorithm 2. KDE-LM

```
Input: $W^o, q_{bit}, N, E$
original weights $W^o$, target quantization bit $q_{bit}$, the number of sampling data $N$,
the number of maximum iteration $E$
Output: $W^q$
quantized weights $W^q$
Initialization: $K, Q$
The number of quantization value $K = 2^{q_{bit}}$,
quantization values $Q = \{q_1, q_2, \dots, q_K\}$
1:  $PDF^o \leftarrow KDE(W^o)$ // kernel density estimation using original weights
2:  $D^s \leftarrow PDF^o(N)$   // sampling $N$ data using probability density function
3:  $PDF^s \leftarrow KDE(D^s)$ // kernel density estimation using sampled data
4:  for e = 1 to $E$ do
5:    for i = 1 to $K - 1$ do
6:      $b_i = \frac{q_i + q_{i+1}}{2}$
7:      $q_i = \frac{\int_{b_i}^{b_{i+1}} xPDF^s{}_x(x)dx}{\int_{b_i}^{b_{i+1}} PDF^s{}_x(x)dx}$
8:  for i = 1 to $|W^o|$ do
9:    $w_i^o \leftarrow \underset{q_j}{\mathrm{argmin}} \|w_i^o - q_j\|^2$, where $w_i^o \in W^o$ // quantize the original weights $W^o$
10:  $W^q \leftarrow W^o$                       // set quantized weights $W^q$
11:  return $W^q$
```

4. Experiments

4.1. Experimental Setup

Experiments were conducted without retraining on pre-trained CNN and we measured the accuracy of classifying 1000 classes using the 50,000 validation images of the ImageNet dataset [28]. The weights quantizations using the proposed KDE-KM and KDE-LM were performed on AlexNet, VGGNet, ResNet, which are well-known in the field of image classification [1,2,4]. Then the accuracy of the 4-bit quantized model using Uniform, KDE-KM, and KDE-LM were compared to the accuracy of the reference model using 32-bit float type weights. Additionally, the proposed methodology's sampling ratio was measured indicating how many less weights were used to perform quantization compared to the number of existing original weights. In all experiments, the size of the sampled data was fixed to 10,000. The number of sampled data used in the quantization was divided by the number of weights of all the networks. The reference models used were pre-trained models in the pytorch framework.

4.2. Performance Evaluations on Quantized CNN Architecutres

We quantized the weights of AlexNet, VGGNet-16, and ResNet-18 using a total of three quantizers. The first model uses a uniform quantizer, which sets the uniform quantization step in the range of minimum and maximum values. The second model is a KDE-LM quantizer using probability density function of sampled data. Finally, we tested the performance of KDE-KM quantizer using sampled data from the probability density function of the original weights. The proposed models perform non-uniform quantization by using fewer data to improve efficiency. Tables 1–3 represent bit-width, top-1 and top-5 accuracy, and sampling ratio of each quantized model for each CNN architecture.

Table 1. Performance comparison of AlexNet architecture.

Model	**Bit-Width**	**Top-1 Accuracy (%)**	**Top-5 Accuracy (%)**	**Sampling Ratio** ($\frac{\lvert W_s\rvert}{\lvert W_t\rvert}$)
Reference	32	56.52	79.07	1.0
Uniform	4	6.48	15.99	1.0
KDE-LM (ours)	4	41.25	66.16	0.0013
KDE-KM (ours)	4	47.43	72.1	0.0013

Table 2. Performance comparison of VGGNet-16 architecture.

Model	**Bit-Width**	**Top-1 Accuracy (%)**	**Top-5 Accuracy (%)**	**Sampling Ratio** ($\frac{\lvert W_s\rvert}{\lvert W_t\rvert}$)
Reference	32	71.55	90.33	1.0
Uniform	4	39.46	63.81	1.0
KDE-LM (ours)	4	66.16	86.38	0.0012
KDE-KM (ours)	4	67.76	88.14	0.0012

Table 3. Performance comparison of ResNet-18 architecture.

Model	**Bit-Width**	**Top-1 Accuracy (%)**	**Top-5 Accuracy (%)**	**Sampling Ratio** ($\frac{\lvert W_s\rvert}{\lvert W_t\rvert}$)
Reference	32	69.73	89.04	1.0
Uniform	4	1.05	3.53	1.0
KDE-LM (ours)	4	58.39	81.48	0.0154
KDE-KM (ours)	4	61.82	83.89	0.0154

The reference CNN models in the experiment perform operations with the weights of the 32-bit float type. In comparison, the quantized CNN models in the experiment perform operations with the weights of 4-bits. The experiment was conducted based on 4-bit quantization because existing quantization related studies reported relatively stable quantization up to a 5-bit band-width [26].

First, for the Uniform quantization, it recorded relatively very low accuracy compared to the reference model except for VGGNet-16. It can be seen that the classification is rarely performed correctly, especially for ResNet-18.

In contrast, in the case of the proposed KDE-LM and KDE-KM, they maintain competitive performance despite the 4-bit quantization. Given the performance difference between KDE-LM and KDE-KM, KDE-KM generally performs better with quantization. KDE-LM carries out quantization using the probability density function obtained by conducting the kernel density estimation once more on the sampled data, so it seems that more errors are introduced during sampling and estimation.

The typical k-means method clusters d-dimensional data of length n into k clusters through i iterations. Thus, the time complexity of general k-means is $O(dnki)$. In the weights quantization task, k and d do not have much effect on time complexity. As the number of clusters k is set to 2^q where q is usually 8-bit or less, and each weight is a scalar rather than a vector, the time complexity of quantization using k-means is $O(ni)$. However, since the typical CNN uses a large number of weights, the value of n is huge. Applying KDE-KM can greatly reduce the number of weights n to s ($s \ll n$), so it becomes $O(si)$, and because the number of iterations i are also affected by the number of data, the time for the weights quantization can be reduced more than proportional to the sampling ratio. Likewise, because the Lloyd–Max quantizer needs an integral calculation to obtain the certain area of the probability density function, the amount of computation for the integral also increases as the number of weights increases. Therefore, applying KDE-LM can also reduce the computation costs for weights quantization proportional to the sampling ratio.

The ratio between sampled dataset and original weights set is presented in Tables 1–3 as the sampling ratio. The size of all sampled data was fixed at 10k data, and so the sampling ratio was 0.0013, 0.0012, and 0.0154 in AlexNet, VGGNet-16, and ResNet-18, respectively. This shows that similar accuracy to the reference model can be achieved using only 0.1% to 1% of original weights. These findings enable more efficient quantization when the quick quantization process is required in retraining, or when rapid deployments of a quantized model is required under certain circumstances.

4.3. Visualization of Quantized Convolutional Neural Networks

One of the important features of the proposed methodology is to perform kernel density estimation using the originals weights, and sampling from the resulting probability density function. Therefore, it is very important to perform the kernel density estimation efficiently using the characteristics of the distribution in the originals.

Figure 3 is the result of visualizing the distribution of the original weights and the sampled data together with the distribution of the quantized weights through the proposed KDE-KM, for the first two CNN architecture cases.

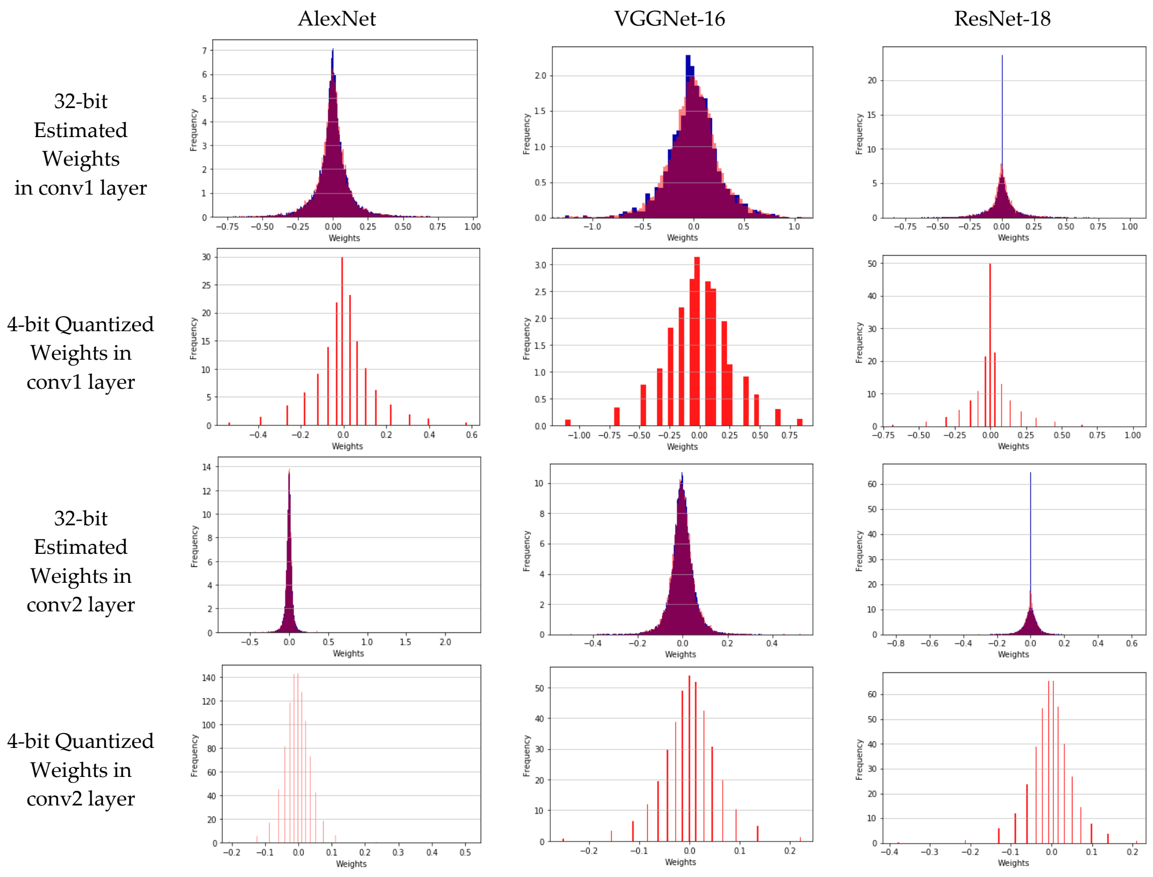

Figure 3. Visualization of estimated weights and quantized weights in the first two convolutional layers.

The first and third rows of Figure 3 are histograms of the weights of the first and second convolution layer, respectively. The distribution of original weights is shown in blue, and the histogram shown in red on top of them indicates the distribution of data sampled from the estimated probabilities density. Although there are a few discrepancies in certain areas, it can be seen that overall the distribution characteristics of the original weights and sampled data are consistent.

Next, the second and fourth rows of Figure 3 are the quantization results of the proposed KDE-KM methodology that are used to generate the codebook. It shows that the spacing of high density areas is narrow and the spacing of low density areas is relatively wide. These visualization results confirm that the proposed methodology is estimates the distribution of the original weights well, while at the same time performing non-uniform quantization successfully.

Figure 3 is the histogram of the entire weights in each convolution layer. So, it is difficult to determine what changes in each weight actually occur when quantization is performed. Figure 4 shows the nine filters of the AlexNet's first convolution layer and the activated feature maps obtained from those filters.

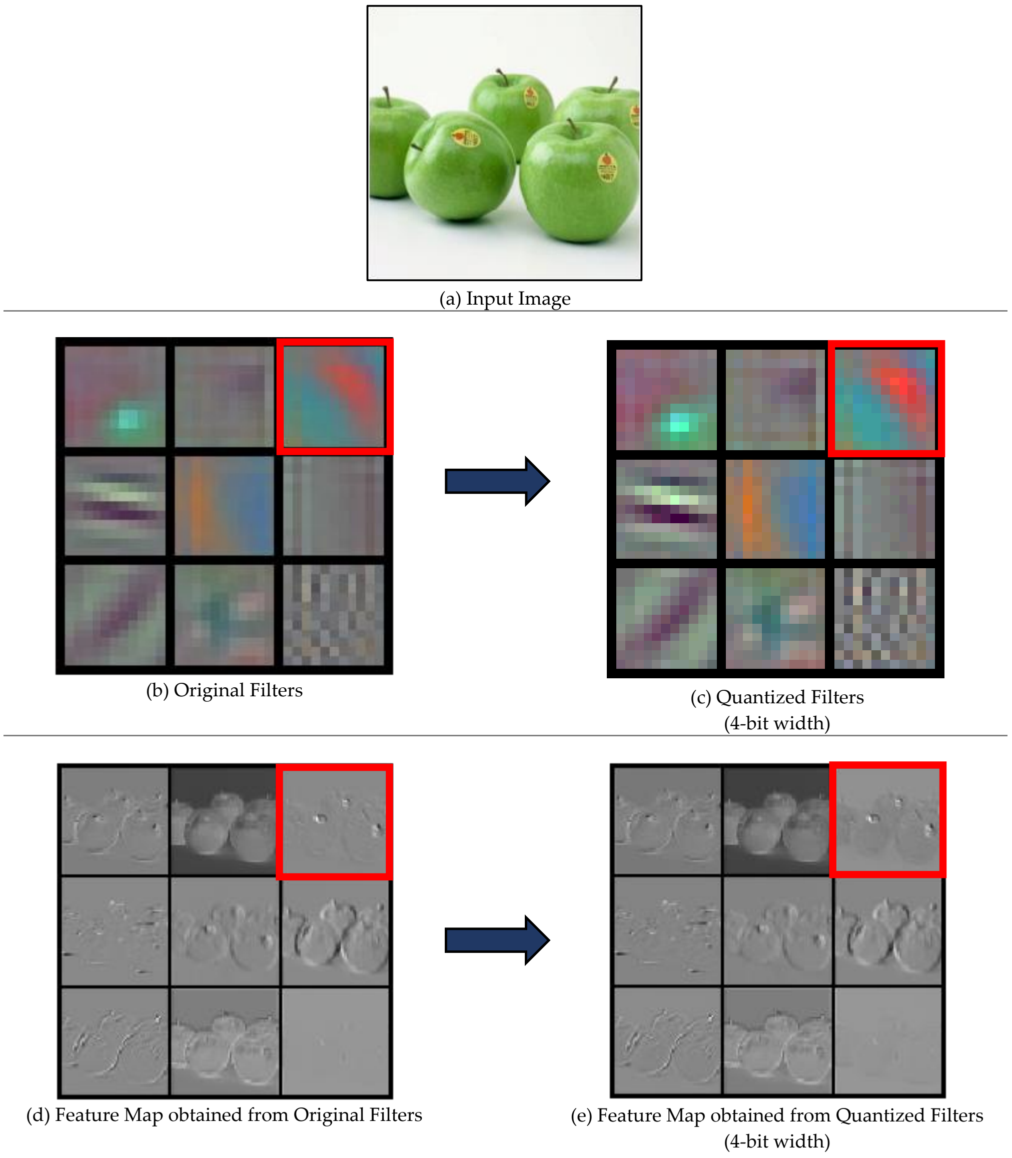
(a) Input Image
(b) Original Filters
(c) Quantized Filters (4-bit width)
(d) Feature Map obtained from Original Filters
(e) Feature Map obtained from Quantized Filters (4-bit width)

Figure 4. Visualization of quantized filters and feature maps.

In Figure 4, (b) is the original filters and (c) is the 4-bit quantized filters using the proposed KDE-KM. Using input image (a) and filters (b) and (c), activated feature maps (d) and (e) are obtained.

The position of filters and feature maps are matched. In the upper right filter emphasized by red boxes in (b) and (c), we can see the difference of the adjacent weights in the quantized filter more clearly in (c). Accordingly, the upper right feature map highlighted by red boxes in (d) and (e) also shows that the contrast is relatively prominent in the feature map in (e), which is from the quantized filter. However, the original filter and the quantized filter show only a slight difference, and the feature maps obtained from each filter also show no significant differences.

5. Conclusions

Deep learning models such as CNN consist of many layers and have a very large number of weights. Due to the computation cost, it is difficult to apply them to a mobile or embedded device with insufficient memory or power, and weights quantization is one of the potential solutions.

Non-uniform quantization methodology such as k-means clustering and the Lloyd–Max quantizer are needed for more accurate quantization. However, recent deep learning models have a very large number of weights, so it is inefficient to perform non-uniform quantization with all the weights in the model. In particular, the speed of the quantization is important when the retraining is performed repeatedly, or when rapid deployment is required. In this paper, it was shown that the kernel density estimation can be used to perform quantization rapidly up to 4-bit without a significant decrease in accuracy, by using a very small number of weight samples of 0.13% to 1.5%.

In future studies, we plan to conduct further studies on the quantization of deep learning modes, including the retraining methodology applied to the quantization to restore the classification accuracy, the efficient quantization method for extreme low bit-width, and the application of quantization to other deep learning models for various tasks such as RNN or LSTM.

Author Contributions: writing—original draft preparation, S.S.; writing—review and editing, J.K.

Acknowledgments: This research was supported by Next-Generation Information Computing Development Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Science, ICT (NRF-2017M3C4A7083279).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. Imagenet Classification with Deep Convolutional Neural Networks. In Proceedings of the Advances in Neural Information Processing Systems, Lake Tahoe, NV, USA, 3–8 December 2012; pp. 1097–1105.
2. Simonyan, K.; Zisserman, A. Very Deep Convolutional Networks for Large-scale Image Recognition. *arXiv* **2014**, arXiv:1409.1556.
3. Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Rabinovich, A. Going Deeper with Convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Boston, MA, USA, 8–10 June 2015; pp. 1–9.
4. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep Residual Learning for Image Recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778.
5. Huang, G.; Liu, Z.; Van Der Maaten, L.; Weinberger, K.Q. Densely Connected Convolutional Networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 4700–4708.
6. Tang, C.Z.; Kwan, H.K. Multilayer Feedforward Neural Networks with Single Powers-of-two Weights. *IEEE Trans. Signal Process.* **1993**, *41*, 2724–2727. [CrossRef]
7. Qiu, J.; Wang, J.; Yao, S.; Guo, K.; Li, B.; Zhou, E.; Wang, Y. Going Deeper with Embedded FPGA Platform for Convolutional Neural Network. In Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, 21–23 February 2016; pp. 26–35.
8. Han, S.; Mao, H.; Dally, W.J. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *arXiv* **2015**, arXiv:1510.00149.

9. Zhou, A.; Yao, A.; Guo, Y.; Xu, L.; Chen, Y. Incremental Network Quantization: Towards Lossless CNNs with Low-precision Weights. *arXiv* **2017**, arXiv:1702.03044.
10. Al-Hami, M.; Pietron, M.; Casas, R.A.; Hijazi, S.L.; Kaul, P. Towards a Stable Quantized Convolutional Neural Networks: An Embedded Perspective. In Proceedings of the International Conference on Agents and Artificial Intelligence, Madeira, Portugal, 16–18 January 2018; pp. 573–580.
11. Park, E.; Ahn, J.; Yoo, S. Weighted Entropy based Quantization for Deep Neural Networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 5456–5464.
12. Cai, Z.; He, X.; Sun, J.; Vasconcelos, N. Deep Learning with Low Precision by Half-wave Gaussian Quantization. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 5918–5926.
13. Zhang, D.; Yang, J.; Ye, D.; Hua, G. Lq-nets: Learned Quantization for Highly Accurate and Compact Deep Neural Networks. In Proceedings of the European Conference on Computer Vision, Munich, Germany, 8–14 September 2018; pp. 365–382.
14. Iandola, F.N.; Han, S.; Moskewicz, M.W.; Ashraf, K.; Dally, W.J.; Keutzer, K. SqueezeNet: AlexNet-level Accuracy with 50x Fewer Parameters and <0.5 MB Model Size. *arXiv* **2016**, arXiv:1602.07360.
15. Howard, A.G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Adam, H. Mobilenets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv* **2017**, arXiv:1704.04861.
16. Courbariaux, M.; Bengio, Y.; David, J.P. Binaryconnect: Training Deep Neural Networks with Binary Weights during Propagations. In Proceedings of the Advances in Neural Information Processing Systems, Montreal, QC, Canada, 7–12 December 2015; pp. 3123–3131.
17. Rastegari, M.; Ordonez, V.; Redmon, J.; Farhadi, A. Xnor-net: Imagenet Classification using Binary Convolutional Neural Networks. In Proceedings of the European Conference on Computer Vision, Amsterdam, The Netherland, 11–14 October 2016; pp. 525–542.
18. Zhou, S.; Wu, Y.; Ni, Z.; Zhou, X.; Wen, H.; Zou, Y. Dorefa-net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients. *arXiv* **2016**, arXiv:1606.06160.
19. Choi, Y.; El-Khamy, M.; Lee, J. Towards the Limit of Network Quantization. *arXiv* **2016**, arXiv:1612.01543.
20. Moons, B.; Goetschalckx, K.; Van Berckelaer, N.; Verhelst, M. Minimum Energy Quantized Neural Networks. In Proceedings of the 2017 51st Asilomar Conference on Signals, Systems, and Computers, Pacific Grove, CA, USA, 29 October–1 November 2017; pp. 1921–1925.
21. Wu, S.; Li, G.; Chen, F.; Shi, L. Training and Inference with Integers in Deep Neural Networks. *arXiv* **2018**, arXiv:1802.04680.
22. Lee, D.; Kim, B. Retraining-based Iterative Weight Quantization for Deep Neural Networks. *arXiv* **2018**, arXiv:1805.11233.
23. Jacob, B.; Kligys, S.; Chen, B.; Zhu, M.; Tang, M.; Howard, A.; Kalenichenko, D. Quantization and Training of Neural Networks for Efficient Integer-arithmetic-only Inference. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–22 June 2018; pp. 2704–2713.
24. Hou, L.; Kwok, J.T. Loss-aware Weight Quantization of Deep Networks. *arXiv* **2018**, arXiv:1802.08635.
25. Geng, X.; Fu, J.; Zhao, B.; Lin, J.; Aly MM, S.; Pal, C.; Chandrasekhar, V. Dataflow-based Joint Quantization of Weights and Activations for Deep Neural Networks. *arXiv* **2019**, arXiv:1901.02064.
26. Seo, S.; Kim, J. Hybrid Approach for Efficient Quantization of Weights in Convolutional Neural Networks. In Proceedings of the 2018 IEEE International Conference on Big Data and Smart Computing (BigComp), Shanghai, Chana, 15–17 January 2018; pp. 638–641.
27. Bagherinezhad, H.; Rastegari, M.; Farhadi, A. Lcnn: Lookup-based Convolutional Neural Network. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 8–12 July 2017; pp. 7120–7129.
28. Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Berg, A.C. Imagenet Large Scale Visual Recognition Challenge. *Int. J. Comput. Vis.* **2015**, *115*, 211–252. [CrossRef]