Neural Processing Letters
https://doi.org/10.1007/s11063-019-10076-y

Methodologies of Compressing a Stable Performance Convolutional Neural Networks in Image Classification

Mo'taz Al-Hami[1] · Marcin Pietron[2] · Raul Casas[3] · Maciej Wielgosz[2]

Abstract
Deep learning has made a real revolution in the embedded computing environment. Convolutional neural network (CNN) revealed itself as a reliable fit to many emerging problems. The next step, is to enhance the CNN role in the embedded devices including both implementation details and performance. Resources needs of storage and computational ability are limited and constrained, resulting in key issues we have to consider in embedded devices. Compressing (i.e., quantizing) the CNN network is a valuable solution. In this paper, Our main goals are: memory compression and complexity reduction (both operations and cycles reduction) of CNNs, using methods (including quantization and pruning) that don't require retraining (i.e., allowing us to exploit them in mobile system, or robots). Also, exploring further quantization techniques for further complexity reduction. To achieve these goals, we compress a CNN model layers (i.e., parameters and outputs) into suitable precision formats using several quantization methodologies. The methodologies are: First, we describe a pruning approach, which allows us to reduce the required storage and computation cycles in embedded devices. Such enhancement can drastically reduce the consumed power and the required resources. Second, a hybrid quantization approach with automatic tuning for the network compression. Third, a K-means quantization approach. With a minor degradation relative to the floating-point performance, the presented pruning and quantization methods are able to produce a stable performance fixed-point reduced networks. A precise fixed-point calculations for coefficients, input/output signals and accumulators are considered in the quantization process.

✉ Mo'taz Al-Hami
motaz@hu.edu.jo

Marcin Pietron
pietron@agh.edu.pl

Raul Casas
rcasas@cadence.com

Maciej Wielgosz
wielgosz@agh.edu.pl

[1] Department of Computer Information System, The Hashemite University, Zarqa 13115, Jordan

[2] Department of Computer Science and Electrical Engineering, AGH University, Cracow, Poland

[3] Cadence Design Systems, San Jose, CA, USA

Published online: 20 July 2019

Keywords Convolutional neural network (CNN) · Fixed-point · Quantization · Pruning · Clustering · K-means · Hybrid quantization · Incremental pruning · Partial quantization · Histogram

1 Introduction

CNN paradigm architecture has shown a state-of-art performance in the field of image recognition benchmarks [1,2]. Studying other different aspects of this network would enhance the network use and implementation in embedded devices. The underlying architecture of the CNN might vary from one network to another. Recent architectures contain a large number of layers. Some architectures might include hundreds of layers, millions of coefficients and even require billions of operations [3].

CNN has different types of layers. Convolutional layers are the ones that consume the majority of computations. These layers have a large number of coefficients and because of that, multiply-accumulate operations (MACs) are performed extensively. Hence, convlutional layers lead to a large number of consumed MACs cycles.

While typical applications use parallel GPUs and perform floating-point data format, the embedded systems and specially the real-time ones require a fixed-point format [4]. In response to this demand, Nvidia has launched new graphical units architectures with reduced representation formats (16-bits, mini-float, and 8-bits integer arithmetic units) [5]. Cadence Design Systems launched a Vision DSP, and Xilinx FPGA. It is worth noting that due to its arbitrary representation and configuration, FPGAs are well suited for deep learning based embedded applications. They enable using arbitrary precision of data representation in every stage of the processing flow. Consequently, the biggest vendors: Intel and Xilinx introduced their own design flows intended for researchers and engineers to facilitate incorporation of deep learning technology into FPGAs [6].

A key property of CNN is its resistance to noise, while keeping the performance stable. Hence, treating any degradation resulted from quantization process as a new source of noise, makes CNN a preferable choice to do this job. Nowadays, CNN has a reliable sloutions to many robotics and vision problems like [7–10].

The quantization process can be applied to both inference and training phases. Training deep neural networks is done by applying many small changes to the network coefficients. These small adjustments utilize higher precision format which accumulate slowly over time and converge to optimal values. There are research efforts to use the quantized representation for the training phase, but in general more bits are required for training phase than for inference phase [11–16]. Our work focuses on quantizing only the inference phase as most real-time embedded systems are currently deployed with this mode of operations [4]. In this article, we focus on quantizing the CNN for both coefficients and (input/ output) data with a reduced precision format (8-bits and partially 4-bits). This article introduces a novel hybrid quantization (HQ) approach. (HQ) quantizes a given network layer data points (i.e., both coefficients and data) at different levels. Hence, the network performance stability might be guaranteed using a specific HQ level. Additionally, k-means clustering approach is implemented for further complexity reduction. The quantization was implemented without any retraining steps and modifications in the network architecture.

In short, the motivation is: Enhancing the CNN capability to be implemented for embedded devices by reducing the required storage for the network, and the performed computations on these low-powered devices. Such enhancement should keep the CNN performance stable for

the inference task. Our main stability criterion is to achieve drop in accuracy less than 1% in all compressed netowrks. Our contributions are: (1) We run a search algorithm discovering the optimal network pruning on a pre-trained network and pruning it with retraining ability.(2) We apply a hybrid quantization HQ approach. (3) We use a k-means quantization approach and other statistical based quantization which reduces the network computation complexity. (4) We generate a stable CNN network with a reduced precision format (8-bits and partially 4-bits) for both coefficients and (input/ output) data.

The paper is organized as follows. The related works are described in Sect. 2. Section 3 explains the CNN architecture and how complexity of a CNN is measured. Pruning and incremental pruning approaches for memory optimization are discussed in Sects. 4 and 5. The floating-point and fixed-point representation, mapping between them, complexity reduction and other quantization systems are discussed in Sect. 6. Section 7 reviews the hybrid quantization approach. In Sect. 8, we present a new automatic quantization approach. K-means quantization approach is described in Sect. 9. In Sect. 10, we show a histogram based quantization technique. Finally conclusions and remarks appear in Sect. 11.

2 Related Works

Many works have shown CNN quantization [12,17,18]. Common strategies are to quantize all coefficients in a single layer with specified number of bits to represent the integer and fractional parts [17,18] based on the range of values of the coefficients set. The other strategy is to represent coefficients and data by integer numbers with appropriate scaling factors. In [19], an 8-bits CNN implementation is compared to an implementation with 32-bits floating-point for speech recognition to speed-up throughput. In [20], a fixed-point network with ternary weights and 3-bits activations approach was presented.

Several methods attempt to binarize weights and the activations in neural networks [14,21–23]. Majority of them resulted in a significant drop in accuracy relative to the floating-point performance. The most efficient approach among them is the XNOR-Net [23] architecture, which achieves comparable results relative to the floating-point format using only binary weights. Complexity reduction of CNN can be achieved by several other techniques (e.g. SVD reduction, reducing filter size) [24–26]. SqueezeNet [27] is an example of compressed version of AlexNet [28], reducing 60 M parameters size to only 0.5 M while preserving its accuracy. There are also other new approaches for image object recognition like hierarchical temporal memory based on neocortex architecture [29–32] which is suitable for embedded systems.

Recently, many tools have been developed for CNN. Many of them take into account the quantization of the networks. Ristretto tool quantizes CNNs using two phases. In the first phase, conventional quantization is used. In the second phase, there is a fine tuning process where the network is retrained using the reduced precision formats of the first phase [18,33]. The TensorFlow [34] and Pytorch deep learning frameworks provide facilities for quantization, where floating-point numbers in a certain range are quantized/dequantized (to/ from) 8-bits asymmetric integer precision format. A K-means approach for CNN compression has been used in [25]. A key difference between this compression method and our work is the dimensionality of the clusters. In [25] the clustered points are single coefficients which makes it difficult to achieve efficiency in an embedded processor, especially with vector processors that can handle hundreds of coefficients and data samples in parallel with a single instruction.

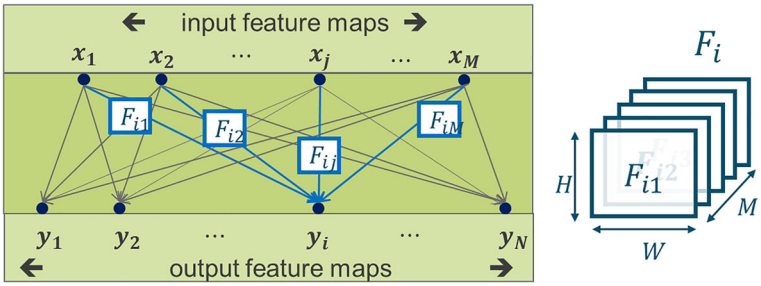

Fig. 1 Convolutional layer structure, and the order of a layer kernels in the convolving process

3 Convolutional Layer Complexity

CNNs are composed of neurons that have learnable weights and biases. Each neuron receives inputs structured as multi-dimensional matrices (tensor), convolves them with its multi-dimensional learnable weights and biases. Typically, the layers of a CNN network have neurons that generate outputs feature maps y_i, $i = 1 \ldots N$ as follows [35]:

$$y_i = b_i + \sum_{j=1}^{M} (F_{ij} \circ x_j) \tag{1}$$

where F_{ij} is a single two-dimensional ($2D$) convolutional kernel with dimensions $H \times W$. x_j represents an input feature maps. '$\circ$' represents the convolving operation and b_i is the bias vector. Figure 1 illustrates one three-dimensional ($3D$) kernel $\bar{F}_i = (F_{i1}, \ldots, F_{iM})$ with dimensions $H \times W \times M$. Altogether, the N three-dimensional kernels form one four dimensional ($4D$) set of filter coefficients related to N output feature maps.

The number of multiply-accumulate (MAC) operations and cycles spent during the execution of Fig. 1 in a practical implementation is often used as the metric for complexity of a CNN [36]. Assuming each output feature map y_i has P output pixels (i.e., image data) the total number of MAC calculations for a convolutional filtering operation is $\text{MACs} = (P \cdot H \cdot W \cdot M \cdot N)$.

4 Pruning

After training a neural model, we acquire a set of weights for each trainable layer. These weights are not evenly distributed over the range of possible values for a selected data format. Majority of weights are concentrated around 0 or very close to it. Therefore, their impact on the resulting activation values is not significant. The pruning mechanism removes weights which values are below a certain threshold level. Authors of [37] examined how pruning affects convolutional neural networks for image classification. In this work, pruning is applied to all convolutional layers according to Algorithm 1. Pruning focuses on weights that are close to 0. For negative values purposes, we use an absolute threshold value.

Depending on the used network implementation specifications, storing weights may require a significant amounts of memory. Applying pruning process to remove some weights has a direct impact on lowering storage requirements. The algorithm computes histograms for a network layers, and divides each one into a specified number of buckets. In each itera-

Table 1 Pruning results of ResNet-50 (drop in Top-1 Error $< 1\%$)

Layer	Coefficients pruned (%)
conv1	36.3
fc1000	56.6
block2	30.8
block3	37.9
block4	34.8
block5	33.8
Average	35.0

tion, it chooses one layer to be pruned. The chosen layer is based on a tournament strategy focuses on layers complexity. If a layer is more complex than others, it has more chance to be chosen.

In each iteration, one layer is pruned with coefficients from the first single bucket which haven't been already pruned (line 7). If drop in accuracy is higher than a given threshold, reverse pruning process is applied (i.e., coefficients from a bucket are set back to the original values (line 9)). Fitness function is the overall memory capacity of the weighted sparsity of current solution (line 11). Solution is represented as a simple genotype, where each layer is a genome and its value is the number of buckets that were pruned in this layer.

Additionally, algorithm stores k-best solutions already found in a ranked list. If algorithm stuck at some point it can return back to a better solution stored in the ranked list. The other applied feature is sensitivity metric which defines how pruning the layer affects the accuracy. If more sensitive layer is chosen, the granularity of buckets is increased for this layer.

The order in which quantization (described later) and pruning are applied affects the quantization process and the storage status, especially for higher precisions. If quantization is applied first, then pruning (if applied for a certain quantization level) will remove the whole level contents. If pruning is applied first, then it is able to cut out weights more precisely. Results of pruning are presented in Tables 1, 2, 3 and 4. The results show that it is possible to achieve between 20 and 50% saving in convolutional layers. Algorithm is also able to prune fully connected layers with saving between 52 and 72%.

The comparison between Algorithm 1 and the approach described in [38] which uses reinforcement learning is presented in Table 5. The used dataset for SegNet is CamVid dataset, while in VGG-16 we used CIFAR-10 dataset. For both SegNet and VGG-16, Algorithm 1 approach achieved worse results than in [38]. In case of VGG-16 in both cases the drop is 1%. In case of SegNet algorithm 1 drops 0.8%, while in [38] the accuracy increases because of retraining process. The baseline accuracies were SegNet 84.01%, VGG-16 92.64%. In our approach, and in [38] were 86.50% and 92.77% respectively. In case of VGG-16, we achieved better results than the approach presented in [39]. In [39] there is up to 64% pruned coefficients and about 6% drop in accuracy. Therefore, the presented approach is significantly faster and can be done on a smaller dataset. It worth noting that in case of FLOPs reduction our approach achieved significantly better results with VGG-16 than in [38], as shown in Table. 6. In this case it guarantees better speedup with higher memory capacity.

The pruning process was reapplied while the drop in `Top-1 Error` is less than 2%. Figures 2 and 3 show the sparsity for each layer for SegNet and VGG-16 respectively.

Algorithm 1 Pruning algorithm.

```
Input: number_of_buckets_to_which_hist_is_divided
Input:
drop_in_accuracy_threshold, baseline_error
compute_hist_coeff_of_all_layers
layer = random_choose_layer_for_pruning()
for number_of_iterations do
    Top-1 Error = compute_accuracy()
    if
    ((Top-1 Error)- baseline) < drop_in_accuracy_thresh then
        prune_next_bucket_from_hist()
    else
        reverse_pruning_bucket_from_hist()
    end if
    fitness = compute_new_fitness()
    if fitness < best_fitness then
        next_solution = current_solution
        best_solution = current_solution
    else
        next_solution = best_solution
    end if
    layer_sensitivity_update
end for
```

Table 2 Pruning results of ResNet-101 (drop in Top-1 Error $< 1\%$)

Layer	Coefficients pruned (%)
conv1	22.7
fc1000	72.9
block2	28.5
block3	31.0
block4	25.6
block5	22.7
Average	26.7

Table 3 Pruning results of VGG-19 (drop in Top-1 Error $< 1\%$)

Layer	Coefficients pruned (%)
conv layers	27.7
fc	51.7
Average	31.5

5 Incremental Pruning

Incremental pruning involves retraining a CNN model after the pruning operation. The process may be considered as incremental shaping of coefficients distribution. Usually, at the very beginning of the process severe degradation of performance occurs. The accuracy of the model improves with more epochs of training. There are several factors affecting the process of the incremental pruning such as: (1) Percentage of coefficients to be pruned. (2) Number of epochs in the retraining, and (3) Degree to which an original model was trained.

It may be sometimes more effective to start with the model which has not been completely trained and prune it gradually with the training process.

Table 4 Pruning results of AlexNet (drop in Top-1 Error $< 1\%$)

Layer	Coefficients pruned (%)
conv1	19.3
conv2	38.6
conv3	42.8
conv4	35.0
conv5	25.3
fc6	58.7
fc7	64.3
fc8	54.0
Average	42.2

Table 5 Compression using the pruning approach. Values represent the percentage of sparsity

Network	Algorithm 1 (%)	Approach in [38] (%)
SegNet	40	56.9
VGG-16	73	81

Table 6 Compression Comparison of FLOPs saved

Network	Algorithm 1 (%)	Approach in [38] (%)
SegNet	40	63.9
VGG-16	71.6	55.2

The incremental pruning experiments involve: (1) Training the model for 8 epochs. (2) Pruning is done after every single epoch. (3) Both coefficients and activation (i.e., output) are quantized to 8-bits, and (4) 40% of each layer coefficients are pruned.

Figures 4 and 5 present results of incremental pruning with retraining. It is worthy to mention that the model of Inception_v3 is not completely trained. The authors trained the model for 11 more epochs and reached accuracy of `Top-1 Error` and `Top-5 Error`: 77.222%, 93.526%, respectively. Going beyond 11 epochs does not lead to a better performance. We repeated the simulations for many different models including ResNet-101, ResNet-50, and VGG-19.

6 Network Quantization

Quantization is the process of constraining values from a continuous set or more dense domain to a relatively discrete set (e.g. integers), and more sparse domain in which the quantized input values will be represented. In our case, the domain is the floating-point representation. A floating-point number can be represented as

$$\text{flp} = m \cdot b^e \tag{2}$$

$\mathbf{m} \in \mathcal{Z}$ is the mantissa, $\mathbf{b} = \mathbf{2}$ is the base, and $\mathbf{e} \in \mathcal{Z}$ is the exponent. In case of single precision floating-point format according to the IEEE-754 standard, the mantissa is assigned 23-bits, the exponent is assigned 8-bits, and 1-bit is assigned for a sign indicator. Therefore,

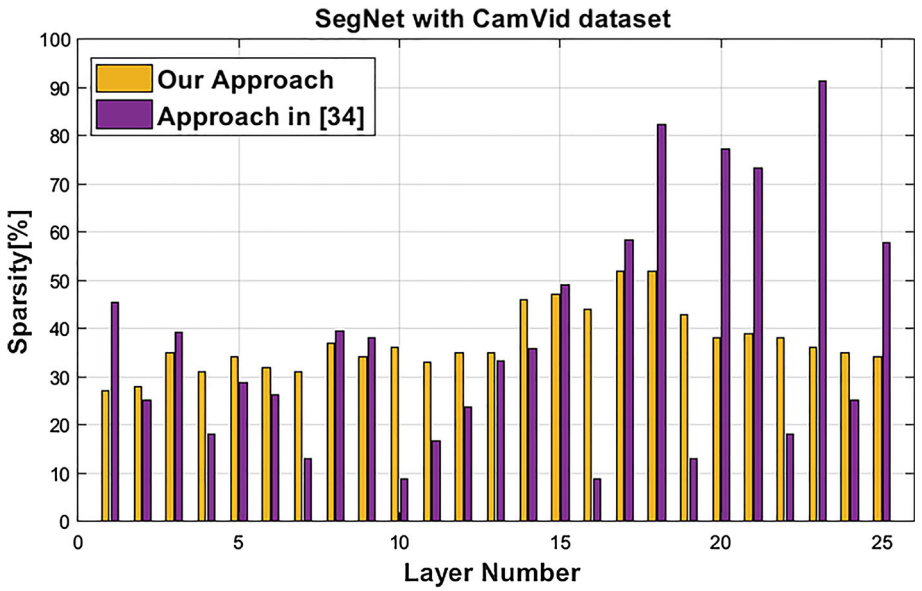

Fig. 2 Layers sparsity in SegNet model. The used dataset is CamVid

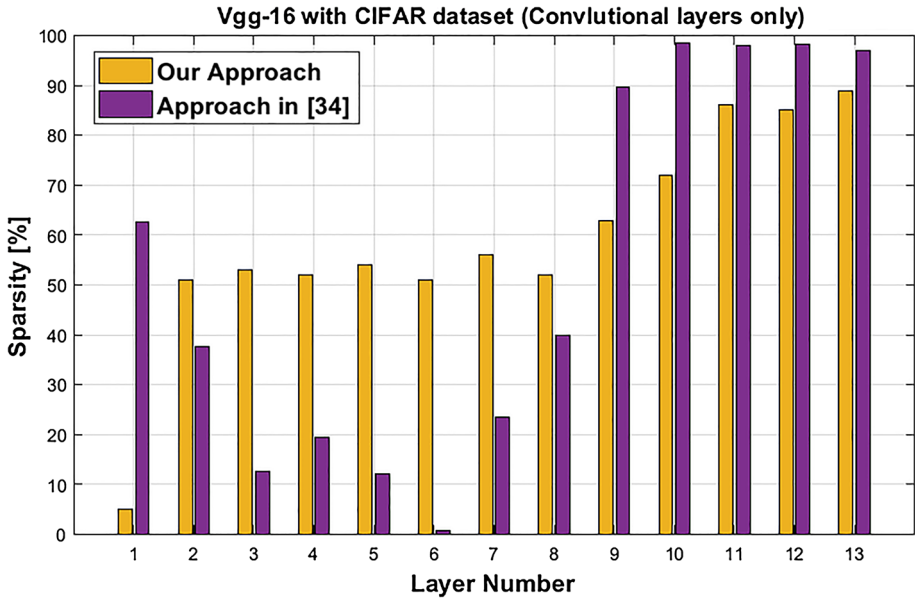

Fig. 3 Layers sparsity in VGG-16 model. The used dataset is CIFAR-10

the set of values that can be defined by this format is described by:

$$\text{flp}_{single} : \{\pm 2^{-126}, \ldots, \pm(2 - 2^{-23}) \times 2^{127}\}. \tag{3}$$

Similarly, a reduced precision IEEE-754 mini-float format assigns 10-bits of mantissa, 5-bits of exponent and a sign bit.

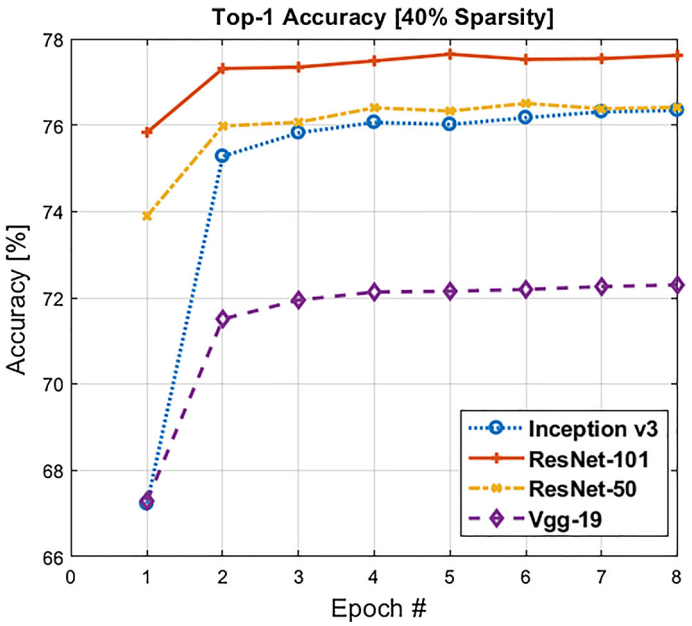

Fig. 4 Top1 accuracy (y-axis) after running 8 epochs (x-axis) of pruning with 40% sparsity in each layer

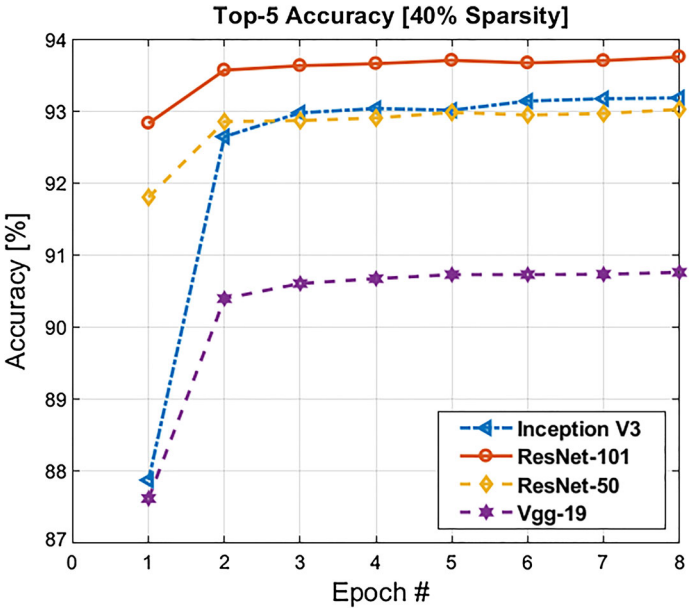

Fig. 5 Top5 accuracy (y-axis) after running 8 epochs (x-axis) of pruning with 40% sparsity in each layer

Currently, GPUs with parallel processing being applied to machine learning operate mainly using the single precision format. In contrast, embedded DSPs and the latest GPUs operate with 8-bits integer and 16-bit fixed-point processing that restricts numbers to the range:

$$\text{fxp} : 2^{-\textbf{frac\_bits}} \cdot \{-2^{\textbf{total\_bits}-1}, \ldots, 2^{\textbf{total\_bits}-1} - 1\} \tag{4}$$

where **total_bits** $= 8, 16$ are conventional bit-widths. The **frac_bits** or $(\mathcal{F})$ is a shift down (or up) that determines fractional length, or number of fractional bits. The signed integer length **int_bits** or $(\mathcal{I}) =$ (**total_bits**) $-$ (**frac_bits**) $- 1$. This format is also referred to as *dynamic fixed-point* [40].

It is possible to define a general mapping from a set of floating-point data $x \in \mathcal{S}$ to a fixed-point q as follows (assuming signed representation)

$$q_{\text{fxp}} = \mathcal{Q}(x_{\text{flp}}) = \mu + \sigma \cdot \text{round}(\sigma^{-1} \cdot (x - \mu)). \tag{5}$$

In our case $\mu = 0$ and $\sigma = 2^{-\textbf{frac\_bits}}$ where:

$$\textbf{int\_bits} = \text{ceil}(\log_2(\max_{x \in \mathcal{S}} |x|)) \tag{6}$$

frac_bits $=$ (**total_bits**) $-$ (**int_bits**) $- 1$. The scaling factor σ is essentially just a shift up or down. A drawback is that a great deal of precision may be lost if the distribution of the dataset $\mathcal{S}$ is skewed by a large mean.

Yet, another approach can define the number of integer and fractional bits to represent regions of a distribution that will represent a large percentage of the range. In this case, there will be saturation of a small percentage of the data such as outliers, through the quantization procedure which may or may not affect the accuracy in a significant way. To determine the effects of saturation, one can experiment with different saturation levels. Therefore, histogram analysis is used to analyze outliers and assign the best levels of saturation.

Another approach is quantization that maps floating-point values to integers:

$$q_{\text{int}} = \mathcal{Q}(x_{\text{flp}}) = ceil((x - \mu)/((max(X) - min(X)) \cdot \sigma^{-1})) \tag{7}$$

The parameter μ can be set to $min(X)$ (i.e., X is an input set of values to be quantized) or can be zero value. In the first case, it is known as a asymmetric integer quantization (e.g. used in Tensorflow framework). In the second case, it is called symmetric. The compression system presented in this paper is based on the symmetric quantization and dynamic fixed point.

All the aforementioned approaches are examples of linear quantization. It is also possible to use nonlinear version to minimize quantization loss, however its hardware implementation is more sophisticated and difficult to achieve a reasonable gain in the used hardware resources.

Yet, another approach can define $\mathcal{I}$ and $\mathcal{F}$ based on data points distribution histogram. Assuming a given percentage (i.e., a large portion of the histogram data points like 99.9%) to be covered in the quantization assigned $\mathcal{I}$ and $\mathcal{F}$. In this case, there will be a small percentage of the data points considered as outliers. To determine the effect of saturation (i.e., outliers), one can experiment the performance with different saturation levels.

Remark 1 Complexity may be interpreted in terms of the size of the hardware blocks required to implement the MAC operations. Also, it can be interpreted in terms of the total or average number of cycles required by a processor to perform these operations. In order to compare the reduction in complexity across quantization, we adopt the MAC complexity of 8-bits operations with notation $< \text{coeff, data} > = < 8b, 8b >$ as a reference unit. Other quantization formats can be compared relative to the complexity of $< 8b, 8b >$ operations. For instance,

Table 7 The parameters of 4th AlexNet convolutional layer and its computational complexity in various quantization formats

Format $< \text{coeff, data} >$	Relative complexity [M MACs]
$< 8b, 8b >$	$224 = HPWMN$
$< 4b, 8b >$	$112 = \frac{1}{2} < 8b, 8b >$
$< 4b, 4b >$	$56 = \frac{1}{4} < 8b, 8b >$
$< \text{flp, flp} >$	$\geq 2016 = 9 < 8b, 8b >$
$N = 256, P = 27^2, M = 48, H = W = 5$	

if a 4-bits representation is used for both coefficients and data, then $< 4b, 4b >$ notation is used. The relative complexity of $< 4b, 4b >$ is approximately $1/4$ that of $< 8b, 8b >$. Similarly, floating-point operations $< \text{flp, flp} >$ will have approximately $9\times$ to $10\times$ of the complexity of $< 8b, 8b >$ since there is $3\times$ of the number of bits to represent the mantissas plus the remaining work required to handle exponents. Table 7 calculates the complexity for the 4th convolutional layer of AlexNet for various quantization formats.

7 Hybrid Quantization

This section explores Hybrid Quantization (HQ) of a CNN network. In HQ, coefficients are represented in a hybrid fashion by assigning different fixed-point formats to different partitions of convolutional layers and feature maps. For instance, different precisions (i.e., numbers of bits) and different formats (i.e., $\mathcal{I}$, $\mathcal{F}$) can be assigned to each 2D filter or 3D kernel of a convolutional layer. Fully connected layers are only treated as 4D kernels. Similarly, different precisions and formats can be assigned to different partitions of the input and output feature maps. Defining precisions and formats in a hybrid fashion adds additional overhead for data representation both in terms of storage space and facilities to decode it. Nevertheless, it can bring about a significant improvement in CNN accuracy when compared to a homogeneous quantization (i.e., 4D). Varying formats for intermediate accumulated values can be specified in the same hybrid fashion. There are three different formatting schemes are summarized as follows:

- **Layer-based or 4D-HQ**: Entire layer kernels are quantized using the same precision format. In Fig. 6a all coefficients in the 4D convolutional kernels F_{ij} are quantized by the same $\mathcal{F}, \mathcal{I}$ precision format.
- **Kernel-based or 3D-HQ**: Coefficients in each 3D kernel in a layer are quantized independently [41]. In Fig. 6b each 3D kernel with filters $\bar{F}_i = (F_{i1}, \ldots, F_{iM})$ is quantized independently with precision format $\mathcal{F}[i], \mathcal{I}[i]$ for each i.
- **Filter-based or 2D-HQ**: Each 2D filter (each channel in a kernel) in each kernel in a layer is quantized independently. In Fig. 6c each filter F_{ij} is quantized independently with precision format $\mathcal{F}[i, j], \mathcal{I}[i, j]$ for each pair i, j.

For a large data precision, homogeneous quantization (i.e., 4D) is sufficient enough to quantize the data points. The problem appears with low precision format, the assigned quantized levels are not sufficient enough to represent all the data points in all layer kernels with satisfied accuracy. In 3D quantization, the maximum absolute value in each kernel controls the kernel quantization format. Hence, it reduces the gap between the data points and the used maximum absolute value. When using 2D quantization, the quantization produces more

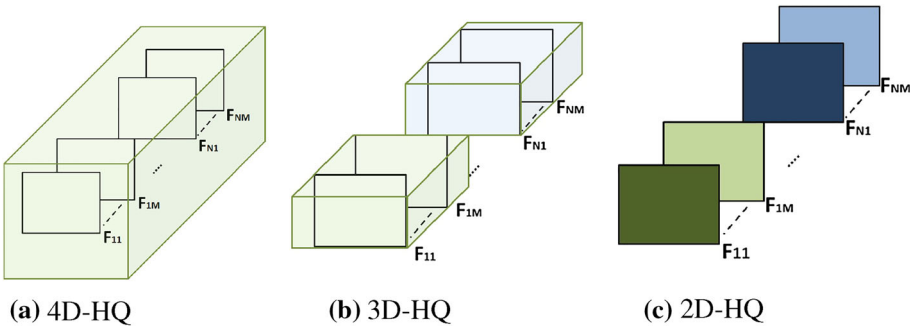

Fig. 6 Different quantization schemes

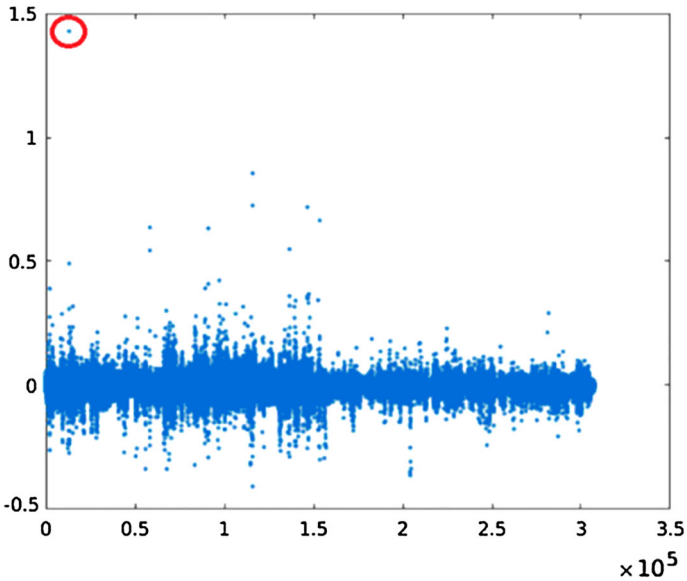

Fig. 7 4D quantization problem when outliers appears in the data. x dimension represents the data points in the 4th convolutional layer in AlexNet. The y dimension represents the magnitude for each data point

efficient quantization, where we restrict the maximum absolute value to each 2D filter independently. This effect is described in Fig. 7. The data point in the red circle (which might seem as an outlier) affects the layer quantization format using the homogeneous quantization (4D). However, using 3D or 2D quantization this effect can be drastically reduced. Therefore, kernel quantization (3D) is performed in case of $< 8b, 8b >$ AlexNet quantization to achieve 1% loss in accuracy relative to the floating-point representation.

Next, we evaluate the performance of various HQ modes on several CNN models. The results are presented in Table 8. The system is also ported to a fixed-point C-model in which the intermediate accumulator values at the output of a convolutional filter can be also quantized. The ported C code supports collecting internal statistics about filters that can be used for a deeper quantization.

Table 8 Comparison of hybrid quantization for several CNN networks models

	Top-5 Error (%)		Top-1 Error (%)		FXP Configuration	
	FLP	FXP	FLP	FXP	Coeff.	Data
Caffe-AlexNet	20.33	20.86	43.35	44.01	8b, 4D	8b, 4D
	20.33	20.74	43.35	43.92	8b, 3D	8b, 4D
MatConvNet-AlexNet	19.93	35.64	42.36	59.66	8b, 4D	8b, 4D
	19.93	20.26	42.36	42.87	8b, 3D	8b, 4D
GoogleNet	13.54	13.68	34.89	35.23	8b, 4D	8b, 4D
	13.54	13.66	34.89	35.23	8b, 3D	8b, 4D
VGG-VeryDeep-19	10.64	10.73	29.48	29.93	8b, 4D	8b, 4D
	10.64	10.83	29.48	30.21	8b, 3D	8b, 4D
ResNet-50	8.29	9.18	25.11	27.00	8b, 4D	8b, 4D
	8.29	8.86	25.11	26.32	8b, 3D	8b, 4D
ResNet-101	8.61	18.33	23.96	40.40	8b, 4D	8b, 4D
	8.61	16.53	23.96	38.30	8b, 3D	8b, 4D
ResNet-152	7.23	91.29	23.51	97.06	8b, 4D	8b, 4D
	7.23	91.08	23.51	96.82	8b, 3D	8b, 4D

7.1 Performance Analysis for Different Networks

Several networks have been tested against our proposed quantization strategy. The goal is to quantize the networks while keeping the performance within the range less than 1% performance degradation relative to the floating-point performance. We applied $< 8b, 8b >$ quantization precision to all networks. Hybrid quantization 4D and 3D are used in the quantization. Table 8 shows the effect of using hybrid quantization using different network models. The results compare between the floating-point performance and the fixed-point performance for both `Top-5 Error`, and `Top-1 Error`. Coefficients and data configurations are described for each network model. Several network models can be quantized using 3D-HQ with less than 1% drop in the accuracy. In some cases for ResNet (i.e., ResNet-101, and ResNet-152) we failed to achieve stable results, however we were able to make them stable with the help of histogram based quantization (explained later).

8 Automatic Advanced Quantization

Investigating the performance of mixing 8-bits and 4-bits precision format on different network layers has a lot of insights, specially for embedded devices. Utilizing some statistics gathered during the evaluation of the CNN, we can decide which parts of the network should use (8-bits/4-bits) precision format. In this quantization scheme, the energy contribution E_{ij} of each 2D filter F_{ij} is accumulated over a set of probe images as follows:

$$E_{ij} = \sum_{\substack{\text{probe} \\ \text{images}}} \sum_{\substack{\text{pixels} \\ p}} |y_{ij}(p)|^2 \tag{8}$$

where $y_{ij} = F_{ij} \circ x_j$ represents partial feature maps. Essentially, the quantization scheme assigns baseline precision (i.e., 8-bits) to 2D filters with high energy contribution and

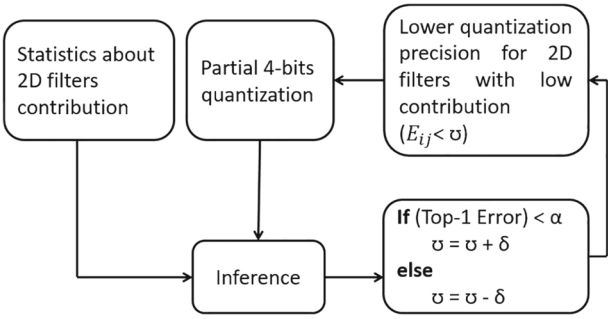

Fig. 8 Automatic tuning. α is the reference Top-1 Error. $\mho$ is the energy threshold. δ is a hyper parameter for energy adjustment amount

Table 9 Hybrid quantization of the 4th layer in AlexNet (40% of coefficients are in 4-bits format), all biases are in 8-bits

Experiment	Layers: 1, 18	4	7, 9, 11, 14, 16	Top-1 Error (%)	Savings (%)
Conservative Baseline	$< 8b, 8b >$	$< 8b, 8b >$	$< 8b, 8b >$	42.7	0
Moderate	$< 8b, 8b >$	$< 4b, 8b >$ (40%)	$< 4b, 8b >$	43.1	30
Aggressive	$< 8b, 8b >$	$< 4b, 8b >$ (100%)	$< 4b, 8b >$	64.1	44

Table 10 VGG-19 4-bits partial quantization

	Fxp Format
Layers 1–15	8-bits (3D-HQ)
Layer 16	4-bits (3D-HQ)
FC (17, 18, 19)	4-bits (3D-HQ)
Top-1 Error	30.8 (%)
FLP	29.48 (%)

lower precision (i.e., 4-bits) to 2D filters with low energy contribution based on some energy threshold $\mho$. An automatic threshold tuning and quantization procedure was developed to minimize complexity through quantization while maintaining a reference level of `Top-1 Error` $= a\alpha$. The procedure is illustrated in Fig. 8.

Table 9 shows the results of this quantization scheme. The *Conservative* experiment utilizes 8-bits 2D-HQ quantization throughout all layers. In the *Moderate* experiment, the layers (1, 18) are quantized to $< 8b, 8b >$. The most computationally expensive Layer 4 is quantized with 40% of its 2D filters in $< 4b, 8b >$ format and all other layers use $< 4b, 8b >$ quantization. The *Moderate* experiment suffers $< 1\%$ loss in accuracy relative to the floating-point but results in 30% savings in complexity relative to the *Conservative* experiment. The *Aggressive* experiment completely quantizes Layer 4 to $< 4b, 8b >$ resulting in a drastic loss in accuracy.

A similar strategy to Algorithm 1, we developed another one for advance quantization and tested it on VGG-19. The approach is based on searching for the optimal partial 4-bits quantization. Algorithm 2 tries to find the configuration with the smallest overall network memory capacity while drop in the accuracy is less than a given threshold (line 10). This

approach prioritizes complex layers for a deeper quantization (line 4). The results are presented in Table 10. The presented solution achieves using 3D-HQ 50% saving in memory capacity and 50% in MAC operations.

Algorithm 2 4-bits partial quantization.

```
1: Input: network layers with theirs complexity
2: Input: drop_in_accuracy_threshold
3: for number_of_iterations do
4:     layer = layer_selection_based_on_complexity()
5:     accuracy_4D = quantize_the_layer_to_4-bits_4D(layer)
6:     accuracy_3D = quantize_the_layer_to_4-bits_3D(layer)
7:     accuracy = max(accuracy_3D, accuracy_4D)
8:     sensitivity = check_sensitivity_of_the_layer(accuracy)
9:     fitness = compute_fitness_function()
10:    if (flp_accuracy - accuracy) < drop_in_accuracy_threshold then
11:        if fitness < best_fitness then
12:            best_fitness = fitness
13:        end if
14:    end if
15: end for
```

9 K-Means Quantization

K-means approach splits a layer coefficients kernels into two parts, basis filters $G_{\ell(i)}$ (i.e., approximations) and residual filters $\tilde{F}_i$ (i.e., differences) leading to $F_i = G_{\ell(i)} + \tilde{F}_i$. The K-means clustering algorithm splits a layer kernels into k centroids kernels serve as approximate kernels for their related cluster groups. The residual kernels are the differences between the original kernels and their corresponding approximation centroids.

The base and residual filters are assigned different quantization formats in order to reduce the complexity while maintaining the accuracy. In this work, we apply K-means clustering to derive a set of $K < N$ basis 3D kernels. These kernels are formatted with $< 8b, 8b >$ precision. The residual kernels are assigned a lower precision (i.e., $< 4b, 8b >$ or $< 4b, 4b >$). The final y_i can be described as:

$$y_i = b_i + \sum_{j=1}^{M} \underbrace{G_{\ell(i),j} \circ x_j}_{<8b,8b>} + \underbrace{\tilde{F}_{ij} \circ x_j}_{<4b,8b>,<4b,4b>} \tag{9}$$

where $G_{\ell(i),j}$ are the basis filters assigned to each 3D kernel via some mapping $\ell(\cdot) : [1..N] \mapsto [1..K]$. The residual filters are given by $\tilde{F}_{ij} = F_{ij} - G_{\ell(i),j}$. Assuming the base kernels use $< 8b, 8b >$ precision and the residuals use $< 4b, 8b >$, the effective complexity savings of this approach is a factor of $(N - K)/2N$. Figure 9 illustrates the implementation of K-means quantization. In the first step, the input is convolved with K basis kernels and generate K temporary feature maps. Next, the output of the residual filters are added to their corresponding base filter temporary feature maps yielding the final output of the convolutional layer. A disadvantage of this approach is that the temporary feature maps have to be stored.

Table 11 shows the performance of the K-means approach for the 4th convolutional layer in AlexNet, the layer with the largest complexity (all other layers are kept in floating-point format). While 2D-HQ with $< 4b, 8b >$ format results in 50% reduction in complexity of

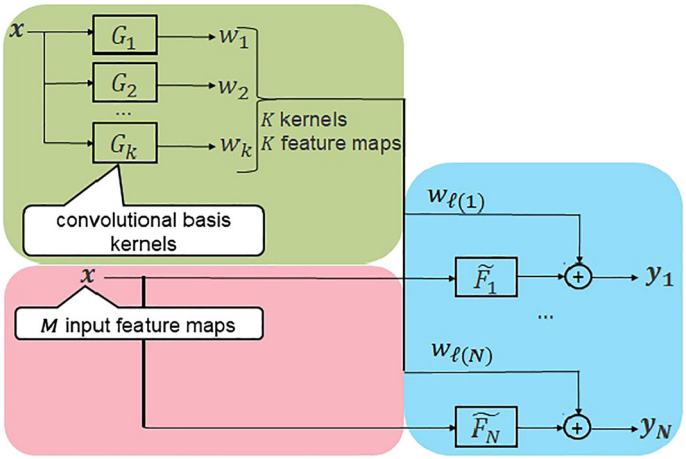

Fig. 9 Quantization using K-means clustering

Table 11 K-means quantization of the 4th Convolutional Layer in AlexNet (all other layers are kept in $< \text{flp}, \text{flp} >$)

Quantization method	Base format	Residuals format	Top-1 Error (%)	Complexity [M Macs]	Layer savings (%)	Network savings (%)
2D-HQ Baseline		$< 8b, 8b >$	42.7	224		
2D-HQ		$< 4b, 8b >$	65.9	112	50	15
K-means	$< 8b, 8b >$	$< 8b, 8b >$	42.2	227	−2	−0.6
K-means	$< 8b, 8b >$	$< 4b, 8b >$	42.7	117	48	15

the layer, the loss in accuracy is drastic. K-means quantization to $< 8b, 8b >$ base format and $< 8b, 8b >$ residuals format with $K = 4$ is attempted as a baseline (i.e., K is chosen to be 4 based on empirical experiments), yielding a slight increase of 2% in complexity and no significant change in performance. Next, K-means quantization to $< 8b, 8b >$ base format and $< 4b, 8b >$ residuals format leads to 48% complexity reduction in the layer MACs and no degradation in accuracy relative to $< 8b, 8b >$.

In Table 12 we applied K-means quantization to $< 4b, 8b >$ residuals format to all layers of the network except layer number 1, which is sensitive to perturbations. The net savings for the network is more than 40% with a degradation in accuracy less than 2%. In this experiment, we chose different k values for different layers (i.e., based on empirical experiments). In Tables 13 and 14, the results of using k-means approach in some parts of ResNet-50 and VGG-19 networks are presented. The results show that further quantization can be done on VGG-19 using k-means without significant drop in the accuracy. ResNet-50 is more sensitive to 4-bits quantization than AlexNet and VGG-19 because of the higher drop in the accuracy.

10 Histogram-Based Quantization

Histogram-based quantization is a combination of 3D-HQ, and 4D-HQ (i.e., first approach) or can be done only as 4D-HQ (i.e., second approach). The motivation beyond this methodology

Table 12 Hybrid clustered quantization of AlexNet

FLP	42.36%
Base	$< 8b, 8b >$
Residuals	Layer 1: $< 8b, 8b >$
	All others: $< 4b, 8b >$
Top-1 Error	44.1%
k	$k = 2$ for layers 1, 4, 11, 14, 16
	$k = 3$ for layer 7
	$k = 6$ for layer 9
	$k = 4$ for layer 18
Complexity [M MACs]	422
Network Savings	41.6%

Table 13 Hybrid clustered quantization of ResNet-50

FLP	25.11%
Base	$< 8b, 8b >$
Residuals	Block 2, 3, 4, 5: $< 8b, 8b >$
	Block 1: $< 4b, 8b >$
Top-1 Error	29.2%
k	$k = 4$
Network Savings	19.8%

Table 14 Hybrid clustered quantization of VGG-19

FLP	29.48%
Base	$< 8b, 8b >$
Residuals	conv layers from 3 to 12 and FCs: $< 8b, 8b >$
	All others: $< 4b, 8b >$
Top-1 Error	30.5%
k	$k = 4$
Network Savings	20.2%

is to separate the data into two parts. One part, is quantized using 4D-HQ, and the other one is using 3D-HQ. The first approach approach uses the histogram of the input, which gives indication on how data is distributed in the histogram curve. Usually a low portion of the data (i.e., less than 1%) can affect the whole process. The tradeoff in this methodology is the computation cost. Quantizing the whole input using 3D-HQ might be a computationally expensive choice, however 4D-HQ might degrade the quantization process accuracy. To deal with such a situation, the majority of the input which is located within a predefined upper and lower bounds in the histogram is quantized using the 4D-HQ, which is a cheap choice of quantization. The rest of the input which is located beyond the predefined upper and lower bounds (i.e., less than 1%) is quantized using 3D-HQ, which is an expensive choice.

Algorithm 3 shows how the histograms are created as a preparation step for quantization. A convolutional layer output (i.e., feature maps) is preprocessed for creating its related histogram. Feature maps (i.e., assuming we are working on feature maps quantization) `data` is extracted from that layer in the preprocessing (line 2). The bounds of the required histogram are specified using `max_data` and `min_data` (lines 3, 4) which represent the maximum

Algorithm 3 Histogram Creation for feature maps.

1: **Input:** Layer output feature maps values
2: `data` = Layer output feature maps values
3: `max_data` = max (`data(:)`)
4: `min_data` = min (`data(:)`)
5: `nbins` = 1000
6: `edges` = linspace (`min_data`, `max_data`, `nbins` + 1)
7: `hist_data_values` = histogram (`data`, `edges`)
8: `cp` = `edges` + (`edges(2) - edges(1)`) / 2

and minimum values in the `data` tensor. For histogram building, we fixed the number of bins to 1000 (line 5), and estimating each bin boundary in the `edges` (line 6). Next, histogram is constructed based on the specified edges and the input feature maps values `data` (line 7). Finally, center point (`cp`) for each histogram bin is estimated (line 8). For coefficients quantization, we follow the same procedure, however this time the input for the histogram is the layer coefficients.

Algorithm 4 Histogram Based Quantization.

1: **Input:** `hist_data`, `cp`
2: **Input:** `data_prob_thresh`
3: `totalValues` = sum (hist_data_values(:))
4: `prob` = 100.
5: `max_int_bits` = ceil (log2 (max (abs (`data(:)`))))
6: `int_bits` = `max_int_bits`
7: **while** `prob` >= `data_prob_thresh` **do**
8: `optimal_prob` = `prob`
9: `optimal_int_bits` = `int_bits`
10: `int_bits` = `int_bits` - 1
11: I = find (abs (`cp(:)`) <= $2^{(\texttt{int\_bits})}$)
12: `linValues` = sum (`hist_data_values`(I));
13: `prob` = 100*sum (`linValues`) / `totalValues`
14: **end while**
15: `lower_bound` = $-2^{\texttt{optimal\_int\_bits}}$
16: `upper_bound` = $2^{\texttt{optimal\_int\_bits}}$
17: `idx_4D` = not(abs (sign (sign (`lower_bound - data`) + sign (`upper_bound - data`))))
18: `idx_3D` = not(not(abs (sign (sign (`lower_bound - data`) + sign (`upper_bound - data`)))))

Algorithm 4 describes the quantization process using histogram based technique. Starting with `data_prob_thresh` which is the used threshold for specifying the mixed quantization percentage. For example, 99 means 99% of a layer output feature maps values are quantized using 4D-HQ, and only one percent is quantized using 3D-HQ. The question now, is how to determine the feature maps values that will share the 4D-HQ and the ones that will be restricted to 3D-HQ. The algorithm focuses on finding an optimal int-bit `optimal_int_bits` which is suitable to quantize a large potion of the data points. This `optimal_int_bits` is estimated through an iterative procedure (lines 7–14). Starting from an initial int-bits `max_int_bits` which covers the whole data points quantization range (line 5), each iteration reduces the number of `optimal_int_bits` by one (line 9–10). This new `optimal_int_bits` makes some data points in the histogram out of quantization range (lines 11–13). This iterative process continues until the new `optimal_int_bits` is suitable for the pre-defined threshold which is `data_prob_thresh`. The algorithm separates data points into two regions, the

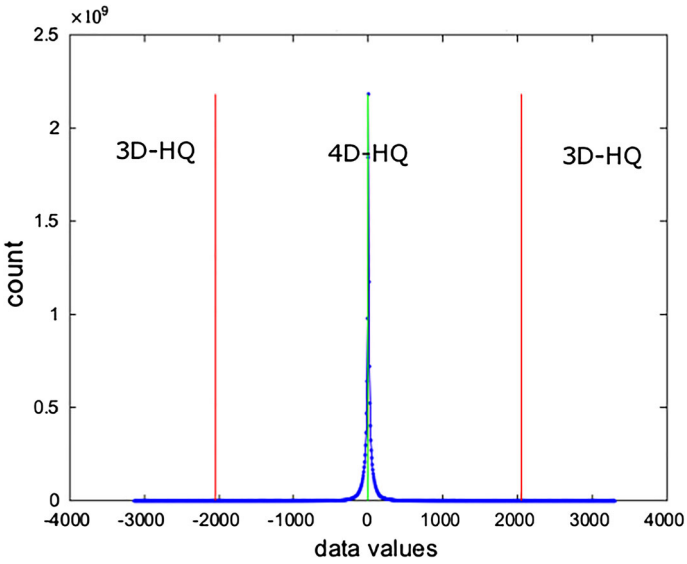

Fig. 10 Histogram based quantization. Histogram of output data for layer 1 for AlexNet model `opt_int_bit` = 11 (`data_prob_thresh` = 99.9988%; `max_int_bit` = 12)

Table 15 Effect of using histogram based quantization with hybrid quantization on several CNN networks models. `data_prob_thresh` = 99.9988%

	Top-5 Error (%)		Top-1 Error (%)		FXP Configuration	
	FLP	FXP	FLP	FXP	Coeff.	Data
Caffe-AlexNet	20.33	20.51	43.35	43.64	8b, 4D	8b, Histogram (second approach)
	20.33	20.43	43.35	43.52	8b, 3D	8b, Histogram (second approach)
MatConvNet-AlexNet	19.93	34.28	42.36	58.83	8b, 4D	8b, Histogram (second approach)
	19.93	20.13	42.36	42.53	8b, 3D	8b, Histogram (second approach)
GoogleNet	13.54	13.60	34.89	34.90	8b, 4D	8b, Histogram (second approach)
	13.54	13.58	34.89	35.09	8b, 3D	8b, Histogram (second approach)
VGG-VeryDeep-19	10.64	10.72	29.48	29.93	8b, 4D	8b, Histogram (second approach)
	10.64	10.61	29.48	29.62	8b, 3D	8b, Histogram (second approach)
ResNet-50	8.29	8.80	25.11	26.25	8b, 4D	8b, Histogram (second approach)
	8.29	8.63	25.11	25.78	8b, 3D	8b, Histogram (second approach)
ResNet-101	8.61	8.36	23.96	25.69	8b, 4D	8b, Histogram (first approach)
	8.61	7.91	23.96	24.59	8b, 3D	8b, Histogram (first approach)
ResNet-152	7.23	8.11	23.51	24.96	8b, 4D	8b, Histogram (first approach)
	7.23	8.07	23.51	25.05	8b, 3D	8b, Histogram (first approach)

one between the red bars (see Fig. 10) is quantized using 4D-HQ. The rest of data points (i.e., the ones outside the red bars in Fig. 10) are quantized using 3D-HQ. In case of the second approach, all points are quantize using 4D-HQ and points that are outliers (1%) are saturated by the maximum value in the computed range (i.e., range in which 99% of points belong to).

Table 15 shows the effect of using histogram based quantization for several CNN networks models. The results show some improvements for some models compared with the results

Table 16 Top-1 Error Comparison with Ristretto quantization tool (8-bits quantization)

Network	Ristretto (8-bits) (%) FXP(FLP)	Ours (8-bits) (%) FXP(FLP)
Caffe-AlexNet	44 (43.1)	43.52 (43.35)
GoogleNet	33.4 (31.1)	34.9 (34.89)

Table 17 Memory Compression Ratio

Network	Ours	Approach in [43]
Caffe-AlexNet	14	27
VGG-16	10	31

in Table 8. For netwok models ResNet-101, and ResNet-152, the histogram based approach helped a lot in making the quantized networks stable. Using additional optimization by using mixed 4D-HQ and 3D-HQ, histogram quantization was able to shrink the drop in accuracy below 1% in ResNet-152 which wasn't achieved in case of using only 4D-HQ histogram based quantization.

A similar approach for dynamic fixed-point quantization included in Ristretto system [42]. We compare our histogram based approach with Ristretto system. The comparison is described in Table 16. In both cases, AlexNet and GoogleNet, the proposed histogram based quantization outperforms Ristretto system [42]. In case of 8-bits quantization of Googlenet the difference is significant. In our case no drop observed (0.01%), in Ristretto more than 2%. Our conclusion is that the described histogram approach for feature maps quantization can boost performance when many outliers appear.

An Additional comparison to the work presented in [43] has been made. The goal is to compare the histogram based quantization approach compression ratio with their approach. It is worthy to mention that, our histogram based quantization does not use retraining phase at all, while in [43] retraining phase is used extensively. Table 17 shows the compression ratio results. The results reveal a big difference between our histogram based quantization and their approach. We can conclude the main reason is the used retraining phase, which allows them to gain a higher sparsity. On the other hand, their approach is quite slow, while our approach requires running about 200 iterations of the algorithm 4 which takes less than half an hour without making any additional retraining phase. Also, their approach uses full dataset in the retraining phase, while our approach uses only random subset of the test dataset (10k images) for creating the histograms.

Another approach in the literature [44] uses dimensionality reduction to compress networks (Q-CNN). We compare our histogram based quantization approach against their dimensionality reduction approach. Table 18 shows the drop in `Top-1 Error` amount using our approach and Q-CNN approach. In Table 19, the numbers represent the compression ratio in our approach using quantization and pruning against the Q-CNN approach. The Q-CNN approach has a better compression ratio for both AlexNet and VGG-16, however our approach has better speedup as shown in Table 20.

11 Conclusions and Further Work

In this work, we propose a novel quantization methods for Convolutional Neural Networks (CNNs). The system reduces memory requirements, number of MACs, and overall power

Table 18 Comparison of drop in accuracy (drop in `Top-1 Error`) between our approach and Q-CNN

Layer	Ours (%)	Q-CNN (%)
Caffe-AlexNet	1.17	1.46
VGG-16	1.21	1.35

Table 19 CNN compression systems (compression ratio with pruning)

Layer	Ours (%)	Q-CNN (%)
AlexNet	14	18.76
VGG-16	10	20.34

Table 20 Comparison of speedup between our approach and Q-CNN

Layer	Ours (%)	Q-CNN (%)
AlexNet	20	4.15
VGG-16	16	4.06

consumption for hardware accelerators. Our research shows that several image recognition networks can be quantized using 8-bits data precision for both coefficients and data outputs with minor degradation in the accuracy compared to the floating-point accuracy. The performed simulations show that memory capacity of well known image recognition networks can be reduced efficiently. Hybrid quantization and K-means quantization allow us to use the advanced quantization (e.g. $< 4b, 8b >$) without any significant degradation in the performance. Our system is implemented in `Pytorch`, `MatConvnet` and ported to a C-model. We consider adding new features in the future such as: exploiting other statistics about a network in the quantization process, further optimizing the K-means approach, network (coefficientss/outputs) binarization [23,45], and applying fine-tuning and advanced incremental quantization. Research will also focus on improving algorithm for finding best compressed network using genetic algorithms and deep reinforcement learning techniques. These additional features will help in reducing the bit-width even further.

References

1. Ciresan D, Meier U, Masci J, Gambardella L, Schmidhuber J (2011) Flexible, high performance convolutional neural networks for image classification. In: Proceedings of the twenty-second international joint conference on artificial intelligence, pp 1237–1242
2. Ciresan D, Meier U, Schmidhuber J (2012) Multi-column deep neural networks for image classification. In: IEEE conference on computer vision and pattern recognition (CVPR), pp 3642–3649
3. He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 770–778
4. Dipert B, Bier J, Rowen C, Dashwood J, Laroche D, Ors A, Thompson M (2016) Deep learning for object recognition: DSP and specialized processor optimizations. http://www.embedded-vision.com/platinum-members/embedded-vision-alliance/embedded-vision-training/documents/pages/cnn-dsps. Aaccessed 30 Dec 2018
5. NVIDIA, Nvidia pascal architecture. http://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf (2016). Accessed 30 Dec 2018
6. Delaye E, Sirasao A, Dudha C, Das S (2017) Deep learning challenges and solutions with xilinx fpgas. In: IEEE/ACM international conference on computer-aided design (ICCAD), pp 908–913

7. Al-Hami M, Lakaemper R (2014) Sitting pose generation using genetic algorithm for nao humanoid robots. In: IEEE international workshop on advanced robotics and its social impacts, pp 137–142
8. Al-Hami M, Lakaemper R (2015) Towards human pose semantic synthesis in 3D based on query keywords. VISAPP (3)
9. Al-Hami M, Lakaemper R (2017) Reconstructing 3D human poses from keyword based image database query. In: International conference on 3D vision (3DV), pp 440–448
10. Al-Hami M, Lakaemper R, Rawashdeh M, Hossain MS (2019) Camera localization for a human-pose in 3D space using a single 2D human-pose image with landmarks: a multimedia social network emerging demand. Multimed Tools Appl 78(3):3587–3608
11. Hubara I, Courbariaux M, Soudry D, El-Yaniv R, Bengio Y (2017) Quantized neural networks: training neural networks with low precision weights and activations. J Mach Learn Res 18:6869–6898
12. Lin D, Talathi S, Annapureddy V (2016) Fixed point quantization of deep convolutional networks. In: International conference on machine learning (ICML), pp 2849–2858
13. Courbariaux M, David J-P, Bengio Y (2014) Training deep neural networks with low precision multiplications. arXiv preprint arXiv:1412.7024
14. Courbariaux M, Hubara I, Soudry D, El-Yaniv R, Bengio Y (2016) Binarized neural networks: training neural networks with weights and activations constrained to $+1$ or -1. arXiv preprint arXiv:1602.02830
15. Gupta S, Agrawal A, Gopalakrishnan K, Narayanan P (2015) Deep learning with limited numerical precision. In: Proceedings of the 32nd international conference on machine learning, pp 1737–1746
16. Esser S, Appuswamy R, Merolla P, Arthur J, Modha D (2015) Backpropagation for energy-efficient neuromorphic computing. In: Advances in neural information processing systems, vol 435, pp 1117–1125
17. Anwar S, Hwang K, Sung W (2015) Fixed point optimization of deep convolutional neural networks for object recognition. In: IEEE international conference on acoustics, speech and signal processing (ICASSP), pp 1131–1135
18. Gysel P, Motamedi M, Ghiasi S (2016) Hardware-oriented approximation of convolutional neural networks. ArXiv e-prints, arXiv:1604.03168
19. Vanhoucke V, Senior A, Mao M (2011) Improving the speed of neural networks on cpus. In: Proceedings of the deep learning and unsupervised feature learning NIPS workshop
20. Hwang K, Sung W (2014) Fixed-point feedforward deep neural network design using weights +1, 0, and -1. In: IEEE workshop on signal processing systems (SiPS)
21. Courbariaux M, Bengio Y, David J (2015) Binaryconnect: training deep neural networks with binary weights during propagations. In: Advances in neural information processing systems, pp 3123–3131
22. Soudry D, Hubara I, Meir R (2014) Expectation backpropagation: parameter-free training of multilayer neural networks with continuous or discrete weights. In: Advances in neural information processing systems (NIPS), pp 963–971
23. Rastegari M, Ordonez V, Redmon J, Farhadi A (2016) Xnor-net: Imagenet classification using binary convolutional neural networks. ArXiv e-prints, arXiv:1603.05279
24. Han S, Liu X, Mao H, Pu J, Pedram A, Horowitz M A, Dally W J (2016) Eie: Efficient inference engine on compressed deep neural network. arXiv preprint arXiv:1602.01528
25. Han S, Mao H, Dally W J (2016) Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1602.01528
26. Hinton GE, Salakhutdinov RR (2006) Reducing the dimensionality of data with neural networks. Science 313(5786):504–507
27. Iandola F N, Moskewicz M W, Ashraf K, Han S, Dally W J, Keutzer K (2016) Squeezenet: Alexnet-level accuracy with 50x fewer parameters and 0.5mb model 465 size. arXiv preprint arXiv:1602.07360
28. Krizhevsky A, Sutskever I, Hinton G E (2012) Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems, pp 1097–1105
29. Wielgosz M, Pietron M (2017) Using spatial pooler of hierarchical temporal memory to classify noisy videos with predefined complexity. J Neurocomput 240:84–97
30. Wielgosz M, Pietron M, Wiatr K (2016) Opencl-accelerated object classification in video streams using spatial pooler of hierarchical temporal memory. arXiv preprint arXiv:1608.01966
31. Pietron M, Wielgosz M, Wiatr K (2016) Formal analysis of htm spatial pooler performance under predefined operation condition. In: International joint conference on rough sets, pp 396–405
32. Pietron M, Wielgosz M, Wiatr K (2016) Parallel implementation of spatial pooler in hierarchical temporal memory. In: International conference on agents and artificial intelligence (ICAART), pp 346–353
33. Ristretto quantization system, http://lepsucd.com/?page_id=621 (2016). Accessed 31 Dec 2018
34. Google, Tensorflow, https://www.tensorflow.org (2016). Accessed 22 Feb 2017

35. Al-Hami M, Pietron M, Casas R, Hijazi S, Kaul P (2018) Towards a stable quantized convolutional neural networks: an embedded perspective. In: 10th International conference on agents and artificial intelligence (ICAART), pp 573–580
36. Courbariaux M, Bengio Y, Jean-Pierre D (2014) Training deep neural networks with low precision multiplications. arXiv preprint arXiv:1412.7024
37. Han S, Pool J, Tran J, Dally W (2015) Learning both weights and connections for efficient neural network. In: Advances in neural information processing systems (NIPS), pp 1135–1143
38. Huang Q, Zhou K, You S, Neumann U (2018) Learning to prune filters in convolutional neural networks, arXiv:1801.07365
39. Li H, Kadav A, Durdanovic I, Samet H, Graf H P (2016) Pruning filters for efficient convnets arXiv preprint arXiv:1608.08710,
40. Gysel P (2016) Ristretto: Hardware-oriented approximation of convolutional neural networks. arXiv preprint arXiv:1605.06402
41. Al-Hami M, Pietron M, Kumar R, Casas R, Hijazi S, Rowen C (2018) Method for hybrid precision convolutional neural network representation. arXiv preprint arXiv:1807.09760
42. Gysel P, Motamedi M, Ghiasi S (2016) Hardware-oriented approximation of convolutional neural networks. arXiv preprint arXiv:1604.03168
43. Han S, Mao H, Dally W J (2015) Deep compression: compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149
44. Wu J, Leng C, Wang Y, Hu Q, Cheng J (2016) Quantized convolutional neural networks for mobile devices. In: IEEE conference on computer vision and pattern recognition (CVPR)
45. Zhang L, Liu B (2016) Ternary weight networks. arXiv:1605.04711