

Iteratively Training Look-Up Tables for Network Quantization

Fabien Cardinaux, Stefan Uhlich, Kazuki Yoshiyama, Javier Alonso García, Lukas Mauch, Stephen Tiedemann, Thomas Kemp, Akira Nakamura.

Abstract—Operating deep neural networks (DNNs) on devices with limited resources requires the reduction of their memory as well as computational footprint. Popular reduction methods are network quantization or pruning, which either reduce the word length of the network parameters or remove weights from the network if they are not needed. In this article we discuss a general framework for network reduction which we call *Look-Up Table Quantization* (LUT-Q). For each layer, we learn a value dictionary and an assignment matrix to represent the network weights. We propose a special solver which combines gradient descent and a one-step k-means update to learn both the value dictionaries and assignment matrices iteratively. This method is very flexible: by constraining the value dictionary, many different reduction problems such as non-uniform network quantization, training of multiplierless networks, network pruning or simultaneous quantization and pruning can be implemented without changing the solver. This flexibility of the LUT-Q method allows us to use the same method to train networks for different hardware capabilities.

Index Terms—Neural Network Compression, Network Quantization, Look-up Table Quantization, Weight tying, Multiplier-less Networks, Multiplier-less Batch Normalization

I. INTRODUCTION

Deep neural networks (DNNs) are currently used in many machine learning and signal processing applications with great success as their performance often beats the previous state-of-the-art approaches by a large margin, e.g., see [2] for an overview of deep learning. DNN approaches have become standard practice in computer vision, automatic speech recognition and partially in natural language processing. They are also extensively investigated to support other domains like medicine, robotics and finance forecasting.

Recently, there has been a lot of interest in the research community in reducing the memory/computational footprint of neural networks. This interest stems from the desire to operate neural networks on devices with limited resources.

F. Cardinaux, S. Uhlich, K. Yoshiyama, J. Alonso García, L. Mauch, S. Tiedemann and T. Kemp are with Sony European Technology Center, Stuttgart, Germany.

A. Nakamura is with Sony Corporate, Tokyo, Japan

F. Cardinaux (fabien.cardinaux@sony.com), S. Uhlich (stefan.uhlich@sony.com) and K. Yoshiyama (kazuki.yoshiyama@sony.com) are equal contributors.

This work extends the preliminary study that we presented as an extended abstract at the NeurIPS 2018 CDNNRIA Workshop [1]

©2019 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

The most commonly used DNN reduction methods can be categorized in the following groups of techniques:

- *Factorized Layers* DNNs [3]–[6] use bottleneck architectures by factorizing traditional layers. These architecture have typically much fewer parameters than traditional DNNs.
- *Pruning* methods [7]–[9] reduce the number of weights by removing the less important connections in the DNNs.
- *Quantization* methods [10]–[13] discretize the weights and/or activations of DNNs.

All these network reduction methods have in common that they add structure to the weight matrices \mathbf{W} , which can later be used for efficient inference, i.e., to store the weights efficiently or to reduce the number of multiplication and accumulation (MAC) operations. Network pruning for example introduces (structured) sparsity, meaning that some elements, rows or columns of \mathbf{W} are set to zero and therefore can be neglected during inference. Factorization methods introduce a low-rank structure to \mathbf{W} , which can also be used for efficient inference. Quantization methods restrict the elements of \mathbf{W} to be from a restricted finite set, such that they can be encoded with a small number of bits and thus \mathbf{W} can be efficiently stored in memory.

In this article we discuss a general framework for network reduction which we call *Look-Up Table Quantization* (LUT-Q). Primarily, LUT-Q is a non-uniform quantization method for DNNs, which uses learned dictionaries $\mathbf{d} \in \mathbb{R}^K$ and lookup tables $\mathbf{A} \in \{1, \dots, K\}^{O \times I}$ to represent the network weights $\mathbf{W} \in \mathbb{R}^{O \times I}$, i.e., we use $\mathbf{W} \in \{\mathbf{X} : [\mathbf{X}]_{oi} = [\mathbf{d}]_{[\mathbf{A}]_{oi}}, \mathbf{d} \in \mathbb{R}^K, \mathbf{A} \in \{1, \dots, K\}^{O \times I}\}$. In this article, we show that LUT-Q is a very flexible tool which allows for an easy combination of non-uniform quantization with other reduction methods like pruning. With LUT-Q, we can easily train networks with highly structured weight matrices \mathbf{W} , by imposing constraints on the dictionary vector \mathbf{d} or the assignment matrix \mathbf{A} . For example, a dictionary vector \mathbf{d} with K elements results in quantized weights which can be encoded with $\log_2(K) + 32K$ bit. Alternatively, we can constrain the \mathbf{d} to contain only the values $\{-1, 1\}$ and obtain a *Binary Connect Network* [14], or to $\{-1, 0, 1\}$ resulting in a *Ternary Weight Network* [15]. This flexibility of our LUT-Q method allows us to use the same method to train networks for different hardware capabilities. Moreover, we show that LUT-Q benefits from optimized dictionary values, compared to other approaches which use predefined values (e.g. [14]–[17]).

The contributions of this paper are as follows:

- We introduce LUT-Q, a trainable non-uniform quantization method which reduces the size and computational complexity of a DNN.
- We propose an update rule to train DNNs which use LUT-Q. The update rule is a combination of a gradient descent and a k-means update, which can jointly learn the optimal weight dictionary \mathbf{d} and assignment matrix \mathbf{A} .
- We show that popular quantization methods from the literature are special cases of LUT-Q. By imposing specific constraints to the basic LUT-Q training, we can learn such networks.
- We propose a multiplier-less *batch normalization* (BN) that can be combined with LUT-Q to train *fully multiplier-less* networks.

This paper is organized as follows: Sec. II summarizes known techniques for neural network reduction and relates them to our LUT-Q. Sec. III describes our basic LUT-Q training algorithm and some extensions of it. Sec. IV introduces a multiplier-less version of batch normalization which produces *fully multiplier-less* networks when combined with LUT-Q. Furthermore, Sec. V discusses efficient inference with LUT-Q networks and Sec. VI presents our experiments and results. Finally, Sec. VII summarizes our approach and gives some future perspectives of LUT-Q.

We use the following notation throughout this paper: x , \mathbf{x} , \mathbf{X} and \mathcal{X} denote a scalar, a (column) vector, a matrix and a tensor with three or four dimensions, respectively; $\lfloor \cdot \rfloor$ and $\lceil \cdot \rceil$ are the floor and ceiling operators.

II. RELATED WORK

Different *compression* methods were proposed in the past in order to reduce the memory footprint and the computational requirements of DNNs.

Common methods either reduce the number of parameters in the architecture or focus on efficient encoding of the parameters. It has been shown that the amount of parameters of a DNN can be reduced drastically with minimal loss in performance by either designing special networks with factorized layers like in MobileNetV2 [6], or alternatively by pruning an over parametrized network [7], [8]. Quantization methods allow an efficient encoding of the parameters which results in a reduced network size. LUT-Q is primarily a quantization method which can also be used effectively for pruning with only small changes in the basic algorithm.

In general, there are three categories of existing network quantization methods:

- *Soft weight sharing*: These methods train the full precision weights such that they form clusters and therefore can be more efficiently quantized [11], [12], [18]–[20].
- *Fixed quantization*: These methods choose a dictionary of values beforehand to which the weights are quantized. Afterwards, they learn the assignments of each weight to the dictionary entries. Examples are *Binary Neural Networks* [14], *Ternary Weight Networks* [15] and also [16], [17].

- *Trained quantization*: These methods learn a dictionary of values to which weights are quantized during training [10]. In [10], the authors propose to run k -means once after a full precision training of a DNN with float32 weights. As soon as the weight dictionary and assignments are obtained from the float32 network, they propose to fine tune the dictionary, while keeping the assignments fixed.

The LUT-Q approach takes the best of the latter two methods: for each layer, we jointly update both dictionary and weight assignments during training. This approach to compression is similar to *Deep Compression* [10] in the way that we learn a dictionary and assign each weight in a layer to one of the dictionary's values. However, we run k -means iteratively during training and update both the assignments and the dictionary at each mini-batch iteration.

In [21] the authors introduce a quantization method that learns a set of binary weights with multiple projection matrices using backpropagation. They show several results for binary networks.

Recently, [22], [13] proposed to learn the step size of a uniform quantizer by backpropagation of the training loss. While the approaches in [13], [22] add more flexibility to fixed step size quantization, all the quantization values are constrained to be equally spaced. LUT-Q allows more flexibility by arbitrarily choosing the dictionary values.

III. LOOK-UP TABLE QUANTIZATION NETWORKS

We consider training and inference of DNNs with LUT-Q layers, i.e., layers which compute

$$\mathbf{Q} = \text{LUTQ}(\mathbf{W}) \quad (1)$$

$$\mathbf{y} = \Phi(\mathbf{Q}\mathbf{x} + \mathbf{b}), \quad (2)$$

where $\mathbf{x} \in \mathbb{R}^I$ is the input vector, $\mathbf{y} \in \mathbb{R}^O$ is the output vector, $\mathbf{W} \in \mathbb{R}^{O \times I}$ is the unquantized weight matrix, $\mathbf{Q} \in \mathbb{R}^{O \times I}$ is the quantized weight matrix, $\mathbf{b} \in \mathbb{R}^O$ is the bias vector and $\Phi : \mathbb{R}^O \rightarrow \mathbb{R}^O$ is the activation function of the layer.¹ $\text{LUTQ} : \mathbb{R}^{O \times I} \rightarrow \mathbb{R}^{O \times I}$ is the look-up table quantization operation, which computes

$$\text{LUTQ}(\mathbf{W}) = \text{lookup}(\mathbf{d}, \mathbf{A}), \quad (3)$$

where $\text{lookup}(\mathbf{d}, \mathbf{A})$ is the table look-up operation that uses the elements of \mathbf{A} to index into the dictionary \mathbf{d} , i.e.,

$$[\text{lookup}(\mathbf{d}, \mathbf{A})]_{oi} = [\mathbf{d}]_{[\mathbf{A}]_{oi}}. \quad (4)$$

At each forward pass, $\text{LUTQ}(\cdot)$ first computes an optimal dictionary $\mathbf{d} \in \mathbb{R}^K$ and an assignment matrix $\mathbf{A} \in \{1, \dots, K\}^{O \times I}$, which fits best to the current weight matrix \mathbf{W} .

$$\mathbf{d}, \mathbf{A} = \arg \min_{\mathbf{d}', \mathbf{A}'} \frac{1}{2} \|\mathbf{W} - \text{lookup}(\mathbf{d}', \mathbf{A}')\|^2 \quad (5)$$

Then, the layer applies the $\text{lookup}(\mathbf{d}, \mathbf{A})$ to obtain the quantized representation \mathbf{Q} of \mathbf{W} and calculates the activation \mathbf{y} .

Fig. 1 illustrates our LUT-Q training scheme. For each forward pass, we learn an assignment matrix $\mathbf{A} \in \mathbb{R}^{O \times I}$ and a

¹For simplicity, we discuss the case where weights are represented as a matrix \mathbf{W} . However, LUT-Q easily extends to the tensor case \mathcal{W} where \mathcal{A} is now also a tensor of the same size.

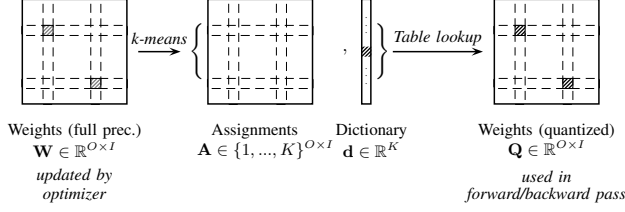


Fig. 1: Proposed look-up table quantization scheme.

dictionary $\mathbf{d} \in \mathbb{R}^K$ and approximate the float weights \mathbf{W} with $\mathbf{Q} = \text{lookup}(\mathbf{d}, \mathbf{A})$. Hence, each weight is quantized to one of the K possible values d_1, \dots, d_K , according to the indices in the assignment matrix \mathbf{A} . We can use the k -means algorithm to learn \mathbf{d} and assignment matrix \mathbf{A} .

A. Training Look-Up Table Quantization Networks

Training quantized neural networks with *stochastic gradient descent* (SGD) is not trivial. One problem is, that running k -means to obtain \mathbf{d} and \mathbf{A} in each forward pass is prohibitively expensive. Furthermore, $\text{LUTQ}(\cdot)$ is not differentiable, meaning that SGD can not be applied directly. Indeed, simply quantizing the weights after each update does not work well since the updates need to be very large (i.e., a high learning rate is required) to change the weights to the next quantization value. Therefore, two methodologies have been used in the literature to train networks with quantized weights:

- *Incrementally quantizing weights*: after each iteration, a subset of the weights is selected and quantized. These weights are fixed for the rest of the training, while the remaining weights are still optimized [23].
- *Gradient accumulation*: the full precision weights are kept and updated during training. For the forward/backward pass the weights are quantized, i.e., gradients are computed with respect to the quantized weights; however, the update of the weights is carried out on the full precision weights. By this, information from gradients over several minibatches is accumulated and the full precision weights act as the accumulator [14], [24], [25].

In our work, we follow the second methodology and simply apply the straight-through gradient estimator whenever we differentiate $\text{LUTQ}(\cdot)$. Please refer to [26] for an analysis of straight-through estimator. The gradient descent step only updates the continuous weights \mathbf{W} , according to

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \nabla_{\mathbf{W}} J(\mathbf{W}), \quad (6)$$

where $J(\mathbf{W})$ is the loss function. We use the straight through gradient estimator (STE) and simply ignore $\text{LUTQ}(\cdot)$ in the backward pass, i.e., we compute

$$\nabla_{\mathbf{W}}(\mathbf{Q}\mathbf{x} + \mathbf{b}) = \nabla_{\mathbf{W}}(\mathbf{W}\mathbf{x} + \mathbf{b}). \quad (7)$$

Furthermore, we unroll the k -means updates over the training iterations, meaning that we just perform one k -means update for each forward pass. This considerably reduces the computational complexity of each forward pass, but is sufficiently

accurate for training if we assume that the continuous weights \mathbf{W} do not change much between iterations (which is always the case if we use a sufficiently small learning rate η).

Algorithm 1 summarizes the LUT-Q update steps for a minibatch $\{\mathbf{X}, \mathbf{T}\}$, where \mathbf{X} denotes the minibatch data and \mathbf{T} the corresponding ground truth. We denote the layer index by l and the total number of layers by L . $K^{(l)}$ is the number of values in the dictionary $\mathbf{d}^{(l)}$. In the forward/backward pass, we use the current quantized weights $\{\mathbf{Q}^{(1)}, \dots, \mathbf{Q}^{(L)}\}$ in order to obtain the cost C and the gradients $\{\mathbf{G}^{(1)}, \dots, \mathbf{G}^{(L)}\}$. These gradients are used to update the full precision weights $\{\mathbf{W}^{(1)}, \dots, \mathbf{W}^{(L)}\}$. Finally, using M steps of k -means after each minibatch, we update the dictionaries $\{\mathbf{d}^{(1)}, \dots, \mathbf{d}^{(L)}\}$ and the assignment matrices $\{\mathbf{A}^{(1)}, \dots, \mathbf{A}^{(L)}\}$. In all our experiments of Sec. VI, we use $M = 1$. k -means ensures that LUT-Q is a good approximation of the full precision weights. Note that in contrast to other approaches [10], we do not fix the assignment matrices but learn them during training. The full precision weights $\mathbf{W}^{(l)}$ can be initialized randomly or using the weights of a previously trained full precision network. After this initialization, for each layer, we run k -means in order to obtain the initial dictionary and assignment matrix².

In “Step 1”, we quantize the full precision weights $\mathbf{W}^{(l)}$ to obtain $\mathbf{Q}^{(l)}$. As described previously, we run just one k -means update step to update the dictionary \mathbf{d} and the assignment matrix \mathbf{A} . In “Step 2”, we calculate the loss gradient with respect to the full precision weights. In “Step 3” of Algorithm 1, we use *stochastic gradient descent* (SGD) to update the full precision weights $\mathbf{W}^{(l)}$, with η being the learning rate. Please note that other optimization strategies can be used, e.g., Adam [27] and Nesterov accelerated gradient descent [28].

B. The flexibility of LUT-Q

While the previous section introduced the basic version of LUT-Q, we will now show that by simple modifications of the clustering (“Step 1” of Algorithm 1), LUT-Q can implement many network compression schemes from the literature.

Weight pruning reduces the network size by setting less important weights to zero [29]. In the LUT-Q scheme, we modify the clustering such that we impose a zero value in the first element of the dictionary, i.e., $\mathbf{d}^T = [0, \dots]$, and force the weights with the smallest magnitudes to be assigned to it such that we achieve a certain pruning ratio. The remaining weights are clustered using a slightly modified k -means where the first cluster centroid is kept to zero. I.e. $d_1^{(l)}$ in Algorithm 1 (Step 1) is set to 0 and not updated. Therefore all weights attracted by $d_1^{(l)}$ will be set to zero, resulting in sparse weight tensors. Han et. al. [10] show that the network size can be reduced by combining pruning with quantization. In [10], however, weights are pruned only once prior to quantization while LUT-Q allows to continuously accumulate the gradient with respect to the pruned weights (through the full precision weights) and may assign them to a non-zero value later in training.

²If one of the two (\mathbf{d} or \mathbf{A}) is initially given, then we can easily compute the other one without running k -means.

```

// Step 1: Compute quantized weights  $LUTQ(\mathbf{W})$ 
// Step 1(A): Update  $\mathbf{d}$  and  $\mathbf{A}$  by  $M$   $k$ -means iterations
for  $l = 1$  to  $L$  do
  for  $m = 1$  to  $M$  do
     $A_{ij}^{(l)} = \arg \min_{k=1, \dots, K^{(l)}} |W_{ij}^{(l)} - d_k^{(l)}|$ 
    for  $k = 1$  to  $K^{(l)}$  do
       $d_k^{(l)} = \frac{1}{\sum_{ij, A_{ij}^{(l)}=k} 1} \sum_{ij, A_{ij}^{(l)}=k} W_{ij}^{(l)}$ 
    end for
  end for
end for
// Step 1(B): Table lookup
for  $l = 1$  to  $L$  do
   $\mathbf{Q}^{(l)} = \mathbf{d}^{(l)}[\mathbf{A}^{(l)}]$ 
end for

// Step 2: Compute current cost and gradients with STE
 $C = \text{Loss}(\mathbf{T}, \text{Forward}(\mathbf{X}, \mathbf{Q}^{(1)}, \dots, \mathbf{Q}^{(L)}))$ 

 $\{\mathbf{G}^{(1)}, \dots, \mathbf{G}^{(L)}\} = \left\{ \frac{\partial C}{\partial \mathbf{Q}^{(1)}}, \dots, \frac{\partial C}{\partial \mathbf{Q}^{(L)}} \right\}$ 
 $\quad = \text{Backward}(\mathbf{X}, \mathbf{T}, \mathbf{Q}^{(1)}, \dots, \mathbf{Q}^{(L)})$ 

// Step 3: Update full precision weights (here: SGD)
for  $l = 1$  to  $L$  do
   $\mathbf{W}^{(l)} = \mathbf{W}^{(l)} - \eta \mathbf{G}^{(l)}$ 
end for

```

Algorithm 1: LUT-Q training algorithm

Binary neural networks are networks with only binary weights [14] and they can be obtained with LUT-Q by choosing a fixed dictionary $\mathbf{d}^T = [-1, +1]$. Furthermore, if we additionally use batch normalization layers, we learn networks which are equivalent to *Binary Weight Networks* [30].

Uniform quantization can be implemented with LUT-Q by choosing and fixing the dictionary; only the weight assignments are learned. Keeping uniform quantization has computational benefits for specific hardware (e.g., fixed-point numbers).

Structured weight matrices/tensors like Toeplitz or circulant as in [31], [32] are an efficient way to learn compact networks. In LUT-Q this can be done by fixing the assignment matrix to encode a specific structure and learning only the dictionary.

Multiplier-less networks can be achieved by either choosing a dictionary \mathbf{d} whose elements d_k are of the form $d_k \in \{\pm 2^{b_k}\}$ for all $k = 1, \dots, K$ with $b_k \in \mathbb{Z}$, or by rounding the output of the k -means algorithm to powers-of-two. In this way we can learn networks whose weights are powers-of-two and can, hence, be implemented without multipliers. Multiplier-less networks will be extensively discussed in the next section.

IV. MULTIPLIER-LESS NETWORKS

A. Introduction

In general, DNN inference involves multiplications as we need to perform matrix-vector products and convolutions. Although multiplications are very optimized in modern computers, inference typically requires millions of them. Therefore,

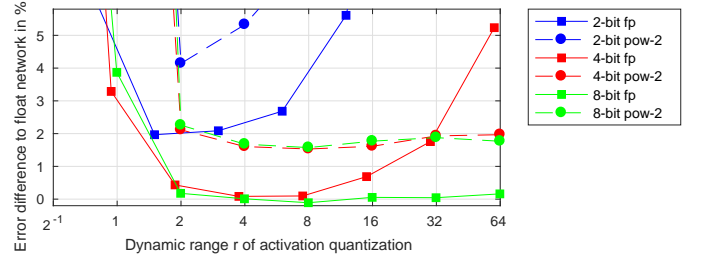


Fig. 2: CIFAR-10: Comparison of activation quantization methods (no weight quantization; y -axis gives validation error difference compared to network trained with float activations).

there is an interest in obtaining networks that do not require multiplications, especially for low power devices.

The idea of training multiplier-less neural networks is not new. Already in the 90's, several works [33]–[35] proposed multiplier-less networks with quantization of the weights to powers-of-two and Simard and Graf [36] created a multiplier-less network by using power-of-two activations. By constraining weights or activations to powers-of-two, we can drastically reduce the computational complexity of multiplications as multiplying a fixed-point number with 2^b becomes a bit shift by b bits and multiplying a floating-point number by 2^b becomes an addition of b to the exponent. More recently, [23], [30], [37], [38] have proposed approaches to train neural networks that avoid multiplications.

Power-of-two weights are easily achieved with our LUT-Q training. The first possibility is to set the dictionary \mathbf{d} to powers-of-two (e.g., $\mathbf{d}^T = [-1, -\frac{1}{2}, \frac{1}{2}, 1]$ for $K = 4$) and fix it during training. A second possibility is to learn the dictionary with an additional rounding of its elements to the closest power-of-two in “Step 1” of the Algorithm 1. In order to quantize $d_k = s \cdot 2^b$ and minimize the quantization error, the quantization threshold should be the arithmetic mean between $2^{\lfloor b \rfloor}$ and $2^{\lceil b \rceil}$. Hence, the quantized value is given by

$$\hat{d}_k = \begin{cases} s \cdot 2^{\lfloor b \rfloor}, & \text{if } b - \lfloor b \rfloor \leq \log_2 1.5 \\ s \cdot 2^{\lceil b \rceil}, & \text{if } b - \lfloor b \rfloor > \log_2 1.5 \end{cases} \quad (8)$$

Both possibilities are compared in Sec. VI-A and we will observe there that the second possibility leads to better performance.

Using power-of-two weights simplifies the calculations in affine/convolution layers. However, traditional *batch normalization* (BN) [39], which has become very popular as it reduces the training time as well as the generalization gap of DNNs, still requires multiplications. Although the number of multiplications in the BN layers is typically small compared to the number of multiplications in the affine/convolution layers, removing all multiplications in a DNN is of interest for specific applications. This is the motivation for our proposed *multiplier-less BN* method, which we will now describe in detail.

B. Multiplier-less Batch Normalization

From [39] we know that the traditional BN at inference time for the o th output is

$$y_o = \gamma_o \frac{x_o - E[x_o]}{\sqrt{\text{VAR}[x_o] + \epsilon}} + \beta_o, \quad (9)$$

where \mathbf{x} and \mathbf{y} are the input and output vectors to the BN layer, γ and β are parameters learned during training, $E[\mathbf{x}]$ and $\text{VAR}[\mathbf{x}]$ are the running mean and variance of the input samples, and ϵ is a small constant to avoid numerical problems. During inference, γ , β , $E[\mathbf{x}]$ and $\text{VAR}[\mathbf{x}]$ are constant and, therefore, the BN function (9) can be written as

$$y_o = a_o \cdot x_o + b_o, \quad (10)$$

where we use the scale $a_o = \gamma_o / \sqrt{\text{VAR}[x_o] + \epsilon}$ and offset $b_o = \beta_o - \gamma_o E[x_o] / \sqrt{\text{VAR}[x_o] + \epsilon}$. In order to obtain a multiplier-less BN, we require \mathbf{a} to be a vector of powers-of-two during inference. This can be achieved by quantizing γ to $\hat{\gamma}$. The quantized $\hat{\gamma}$ is learned with the same idea as for LUT-Q: during the forward pass, we use traditional BN with the quantized $\hat{\gamma} = \hat{\mathbf{a}} / \sqrt{\text{VAR}[\mathbf{x}] + \epsilon}$ where $\hat{\mathbf{a}}$ is obtained from \mathbf{a} by using the power-of-two quantization (8). Then, in the backward pass, we update the full precision γ . Please note that the computations during training time are not multiplier-less but $\hat{\gamma}$ is only learned such that we obtain a multiplier-less BN during inference time. This is different to [37] which proposed a shift-based batch normalization using a different scheme that avoids all multiplications in the batch normalization operation by rounding multiplicands to powers-of-two in each forward pass. Their focus is on speeding up training by avoiding multiplications during training time, while our multiplier-less batch normalization approach avoids multiplications during inference.

C. Naming Convention

For the description of our results in the next sections, we will use the following naming convention:

- *Quasi multiplier-less* networks avoid multiplications in all affine/convolution layers, but they are not completely multiplier-less since they contain standard BN layers, which are not multiplier-less. For example, the networks trained by Zhou et al. [23] are quasi multiplier-less.
- *Fully multiplier-less* networks avoid all multiplications at all. Either they are multiplier-less networks with no BN layers or multiplier-less networks that use multiplier-less BN as explained in the previous section.
- We call all other networks *unconstrained*.

V. INFERENCE EFFICIENCY

In this section, we discuss the efficient implementation for inference of networks trained with LUT-Q. Note that our objective is to reduce the memory footprint, required computations and energy consumption of neural networks at inference time. We start by briefly discussing the reduction that can be achieved in terms of memory footprint and then discuss the reduction in the number of computations.

The memory used for network parameters is dominated by the weights in affine/convolution layers. Using LUT-Q, instead of storing \mathbf{W} , the dictionary \mathbf{d} and the assignment matrix \mathbf{A} are stored. Hence, for an affine/convolution layer with N parameters, the reduction is

$$NB_{\text{float}} \xrightarrow{\text{LUT-Q}} KB_{\text{float}} + N \lceil \log_2 K \rceil, \quad (11)$$

where B_{float} is the number of bits to store a weight (e.g., $B_{\text{float}} = 32$ bit for single precision). In Sec. VI, we experiment with K ranging from $K = 2$ (1-bit quantization) to $K = 256$ (8-bit quantization).

Assuming that the device used for inference performs a forward pass by computing layer operations sequentially, the device typically keeps a buffer to store the input and output activations of the layer being processed. Under this assumption, the buffer memory should be large enough to store the input and output activations of any layer in the network. Table I shows that for full precision networks, the buffer memory is one order of magnitude smaller than the memory needed for the parameters. However, for heavily quantized networks the buffer memory used for the activations dominates the memory footprint and, therefore, it should be quantized as well. Fig. 2 compares “fp” and “pow-2” activation quantization.³ This plot shows that uniformly quantizing the activations to 8-bit with a well chosen dynamic range $[0, r]$ allows to reduce the required buffer memory by a factor of four without loss in accuracy. We will use this 8-bit activation quantization for the remaining experiments presented in this paper except where explicitly stated.

Using LUT-Q or other quantization methods, we also achieve a reduction in the number of computations. Consider the case of computing one output value y_o for a LUT-Q affine layer which is given by

$$y_o = b_o + \sum_{i=1}^I Q_{oi} x_i = b_o + \sum_{k=1}^K d_k \left(\sum_{i=1, A_{oi}=k}^I x_i \right), \quad (12)$$

where I denotes the size of the input vector \mathbf{x} . We reduce the number of multiplications from I to K . Similarly, in the case of a 2-D convolution layer we reduce the multiplications for one output map from $I \cdot S \cdot F$ to $S \cdot K$, where I is now the number of input maps, S is the map size (height \times width) and F is the filter size (height \times width). The efficient hardware implementation in (12) is achieved by K parallel registers that store the sum of activations for each k .

Table I summarizes the memory footprint and computations for the image classification networks that we will use in the next section.

VI. EXPERIMENTS

We conducted extensive experiments with LUT-Q quantization on the CIFAR-10 image classification benchmark [40]. We first confirm the potential of learning a dictionary of quantized values. Then we demonstrate the capability of LUT-Q to train

³Please refer to Sec. VI-A for more details about fp and pow-2 quantization. In contrast to weight quantization, we only need to quantize to $[0, r]$ due to the ReLU nonlinearity, i.e., we do not need to spend a bit on coding the sign.

TABLE I: Memory and computations for ResNet-20 for CIFAR-10 and ResNet-18/-34/-50 for ImageNet. Activations have 32 bit.

Net	Weight Quant.	Param. Memory (MB)	Buffer Memory (MB)	Computations (million of ops)	
				Add.	Mul.
ResNet-20 for CIFAR-10					
Full Prec.	32-bit	1.03	0.13	40.64	40.55
LUT-Q	8-bit	0.28	0.13	40.64	32.56
LUT-Q	4-bit	0.13	0.13	40.64	3.01
LUT-Q	2-bit	0.07	0.13	40.64	0.75
LUT-Q	1-bit	0.04	0.13	40.64	0.38
ResNet-18 for ImageNet					
Full Prec.	32-bit	44.59	3.64	1814.85	1814.07
LUT-Q	8-bit	5.61	3.64	1814.85	39.76
LUT-Q	2 bit	2.83	3.64	1814.85	9.94
ResNet-34 for ImageNet					
Full Prec.	32-bit	83.15	3.64	3665.17	3663.76
LUT-Q	4-bit	10.46	3.64	3665.17	59.83
LUT-Q	2-bit	5.26	3.64	3665.17	14.96
ResNet-50 for ImageNet					
Full Prec.	32-bit	97.49	4.59	4094.80	4089.18
LUT-Q	4-bit	12.37	4.59	4094.80	177.84
LUT-Q	2-bit	6.29	4.59	4094.80	44.46

pruned and multiplier-less networks. Then, we evaluate LUT-Q on large scale tasks, namely the ImageNet ILSVRC-2012 image classification [41], PASCAL VOC object detection [42] and the Wall Street Journal acoustic modeling [43] for *automatic speech recognition* (ASR).

All experiments are implemented, using the *Sony Neural Network Library*⁴. We implemented efficiently the k-means updates in CUDA, using monolithic kernels, vectorized loads operations and shuffle instructions. This reduces drastically the training time overhead due to the k-means updates.

For CIFAR-10, we first trained the full precision *ResNet-20*, which is used to initialize the quantization trainings (seed network). We followed the training procedure used in [44] with data augmentation: we trained for 160 epochs, decreasing the learning rate at epochs 80 and 120. For training the quantized network we started from the seed network and applied quantization training (e.g., LUT-Q) for additional 160 epochs, following the same training scheme. Performance is evaluated as the best validation error without image augmentation on the validation data.

The seed network achieves an error rate of 7.84%. However, for a fair comparison to the quantized networks, we continued training it for additional 160 epochs and achieved the slightly lower error rate of 7.42% (averaged over 10 runs). We use this error rate as our 32-bit full precision network baseline.

A. Fixed Quantization versus LUT-Q

One of the main advantages of LUT-Q is that it jointly learns both dictionary values and assignments. Traditional approaches fix the quantization values in advance and just learn the assignments. Common choices for the quantization to n bits are *fixed-point* or *powers-of-two*:

TABLE II: CIFAR-10: Validation error of ResNet-20 with 4-bit and 2-bit weight quantization. The full precision baseline model achieves an error rate of 7.42%.

Weight quant. method	Batch norm method	Activation quant. bitwidth	Validation error method	4-bit	2-bit
fp	traditional	8-bit	fp	7.60%	12.72%
pow-2 LUT-Q	traditional	8-bit	pow-2	7.61%	8.02%

- *Fixed-point quantization* (“fp”) quantizes a float weight $w = s|w|$ to

$$q = s \cdot \delta \cdot \begin{cases} \left\lfloor \frac{|w|}{\delta} + 0.5 \right\rfloor & |w|/\delta \leq (2^{n-1} - 1) \\ 2^{n-1} - 1 & \text{otherwise} \end{cases},$$

i.e., uniformly quantizes w using the quantization step size δ and, hence, the dynamic range is $r = (2^{n-1} - 1)\delta$. For an efficient hardware implementation, δ needs to be a power-of-two as the multiplication of two fp quantized numbers can then be implemented by an integer multiplication and a bit shift by n bits.

- *Pow-2 quantization* (“pow-2”): Similarly, the pow-2 quantization of the weight $w = s|w|$ is given by

$$q = s \cdot \begin{cases} 0 & |w| \leq 2^{m-2^{n-2}+0.5} \\ 2^{\lfloor \log_2 |w| + 0.5 \rfloor} & 2^{m-2^{n-2}+0.5} < |w| \leq 2^m \\ 2^m & \text{otherwise} \end{cases},$$

where $m \in \mathbb{Z}$ with $2^m = r$ being the dynamic range.

We use the same scheme as [16] and [17] to train these networks. Note that this is equivalent to LUT-Q training but skipping the dictionary update in the initial k -means and in the “Step 1” of Algorithm 1.

For fp and pow-2 quantization, we need to choose the dynamic range $[-r, r]$ for the weight quantization. Preliminary experiments showed that the choice of the dynamic range drastically influences the performance, especially for very small bitwidth quantization. We follow the approach from [23] where the dynamic range is chosen for each layer using $r = 2^{\lceil \log_2 \max_{oi} |W_{oi}| \rceil}$. As explained in Sec. V, full precision activations dominate the memory requirements for very low bitwidth weights. Therefore, we trained the networks with activations quantized to 8-bit using fp quantization.

Table II compares the results of 4-bit and 2-bit quantization of a ResNet-20 with either fixed point (fp) or LUT-Q (constrained to power-of-two). We observe that fp achieves near baseline performances, however, we lose performance when the weights are quantized to 2 bit. LUT-Q achieves similar performance for 4-bit networks and significantly outperformed the fixed quantization methods for 2-bit weights with 8.02% error rate, even if we constrain the values to be powers-of-two.

B. Pruning

As explained in Sec. III-B, LUT-Q can be used to prune and quantize networks. As seen in Sec V, quantization is a very effective way of reducing the memory size and also the number of multiplications of a network. From Table I, we observe that the remaining computations are dominated by

⁴Neural Network Libraries by Sony: <https://nnabla.org/>

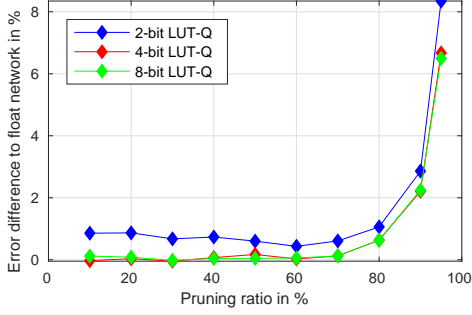


Fig. 3: CIFAR-10: Validation error for LUT-Q with pruning.

the additions. Interestingly, from (12), when using LUT-Q for pruning and quantization, $d_1 = 0$ and all additions for $k = 1$ are avoided. Therefore, the reduction in number of additions in affine/convolution layers is proportional to the pruning ratio.

Figure 3 shows the error rate increase between the baseline full precision ResNet-20 and the pruned and quantized network using full precision activations. Using LUT-Q we can prune and quantize the networks up to 70% without significant loss in performance. With this pruning ratio we reduce the total number of additions from 40.64M to 12.38M.

Thanks to the ability of LUT-Q to simultaneously prune and quantize networks, we have shown that we can drastically reduce the memory footprint, the number of multiplications and the number of additions of deep neural networks at the same time.

C. Multiplier-less Networks

While the number of multiplications can be reduced using standard LUT-Q, remaining multiplications in affine/convolution layers can be avoided by restricting the dictionary to powers-of-two. Sec. IV described how to train such a network with LUT-Q. We refer to this method as *pow-2 LUT-Q*. In our experiments, constraining the dictionary to powers-of-two did not degrade the performance compared to an unconstrained dictionary. Therefore, in the remaining of this paper we will not show unconstrained LUT-Q results and will only focus on *pow-2 LUT-Q*.

Recently, Zhou et al. [23] proposed an *incremental network quantization* (INQ) approach to train quasi multiplier-less⁵ networks. Since Zhou et al. [23] did not conduct experiments on the ResNet-20 dataset, we compare LUT-Q with our own implementation of INQ in Table III.⁶ For training we initialized ResNet-20 with the same full precision model and trained the INQ network for the same number of epochs (i.e., 160 epochs) as for LUT-Q. Every 20 epochs we quantized and fixed half of the weights based on the so-called *pruning-inspired partition* criterion. We found that we obtained best results when we reduced the learning rate twice within each period of 20 epochs

and reset it to the original learning rate after quantizing and fixing more weights. Table III compares the performance of the LUT-Q networks to the performance of quasi multiplier-less networks quantized to $\{8, 4, 2\}$ bits with INQ. We can observe, that LUT-Q systematically outperform INQ for both quasi and fully multiplier-less networks. This can be best observed for networks which use a very small bitwidth. Second, to construct fully multiplier-less networks, it is beneficial to choose a network with power-of-two quantized weights which uses a multiplier-less batch normalization. Such networks consistently outperform multiplier-less networks with power-of-two activations and traditional batch normalization. Moreover, we observe that the best fully multiplier-less network lose around 0.5% to 2% accuracy compared to the quasi multiplier-less networks as usually reported in the literature.

Most of the multiplications are typically in the affine/convolution layers. However, for networks like ResNet, some multiplications remain in the batch normalization layers during inference. To get rid of these multiplications, we can think of three approaches:

- Remove batch normalization layers and train without them.
- Quantize the activations to powers-of-two: In this case, it is not required to enforce power-of-two weights and traditional batch normalization can be used, since the coefficients of the batch normalization can be folded into the weights from a preceding affine/convolution layer during inference.
- Replace traditional batch normalization by a multiplier-less batch normalization as described in Sec. IV-B.

In our experiments, training quantized networks without batch normalization layers is difficult and leads to poor performance. For example, the error rate for 8-bit weight quantization is more than 5% higher than for the full precision baseline with batch normalization. Table III shows the results of quasi multiplier-less against fully multiplier-less networks. We observe some performance loss when constraining to fully multiplier-less networks. However, the models with multiplier-less batch normalization systematically outperform the models with power-of-two activations. Furthermore, the networks trained with LUT-Q outperform those trained with INQ.

For 4-bit power-of-two weights and 8-bit activations, the error rate of LUT-Q quasi multiplier-less ResNet-20 is only 0.19% higher than the baseline full precision model and the fully multiplier-less version has an increased error of 0.65% compared to the baseline.

D. ImageNet Experiments

We also trained quantized and multiplier-less networks on the more challenging ImageNet task. We use *ResNet-18*, *ResNet-34* and *ResNet-50* as reference networks [44]. For training the ResNet models on ImageNet, we first resized all images to 320×320 , then applied data augmentation (aspect ratio, flipping, rotation, etc.) and finally randomly cropped to a 224×224 image. We train the models for 90 epochs with SGD using Nesterov momentum starting with a learning rate of 0.1 and decay it by a factor of 10 every 30 epochs. We

⁵The authors refer to them as multiplier-less but, as explained in Sec. IV-B, multiplications are still required for the batch normalization layers. We call these networks *quasi multiplier-less*, see Sec. IV-C.

⁶Note that INQ networks are trained with the native INQ implementation from *Sony Neural Network Library*

TABLE III: CIFAR-10: Validation error of multiplier-less ResNet-20 with 8-bit/4-bit/2-bit/1-bit weight quantization. The full precision baseline model achieves an error rate of 7.42%.

Weight quant. method	Batch norm method	Activation quant.		Validation error			
		bitwidth	method	8-bit	4-bit	2-bit	1-bit
<i>Quasi multiplier-less</i>							
INQ	traditional	8-bit	fp	8.34%	8.46%	38.84%	-
pow-2 LUT-Q	traditional	8-bit	fp	7.22%	7.61%	8.02%	9.31%
<i>Fully multiplier-less</i>							
INQ	multiplier-less	8-bit	fp	10.61%	10.58%	39.47%	-
LUT-Q	traditional	8-bit	pow-2	8.72%	8.99%	10.37%	13.82%
pow-2 LUT-Q	multiplier-less	8-bit	fp	8.23%	8.07%	8.98%	11.58%

TABLE IV: ImageNet: Multiplier-less networks using LUT-Q. \checkmark : fully multiplier-less. \checkmark : quasi multiplier-less. \times : unconstrained.

Weights		Batch norm	Activations		Multiplier-less	Validation error		
						ResNet-18	ResNet-34	ResNet-50
32-bit		traditional	32-bit		\times	30.96%	28.06%	25.87%
4-bit	LUT-Q	traditional	8-bit	pow-2	\checkmark	34.37%	31.44%	27.50%
4-bit	LUT-Q pow-2	multiplier-less	8-bit	fp	\checkmark	35.06%	30.65%	26.89%
4-bit	LUT-Q pow-2	traditional	8-bit	fp	\checkmark	31.58%	28.10%	25.46%
2-bit	LUT-Q	traditional	8-bit	pow-2	\checkmark	43.16%	37.34%	32.12%
2-bit	LUT-Q pow-2	multiplier-less	8-bit	fp	\checkmark	43.23%	35.20%	29.81%
2-bit	LUT-Q pow-2	traditional	8-bit	fp	\checkmark	35.80%	30.51%	26.92%

followed the same training scheme for the quantized networks and initialized their parameters with the trained full precision parameters. Reported performance are top-1 errors of the center-cropped validation images.

Results of multiplier-less LUT-Q training for ResNet-18, ResNet-34 and ResNet-50 are shown in Table IV. Note that all the quantized networks trained with LUT-Q also use activation quantization to 8 bit, which avoids some of the overhead due to the buffer memory as explained in Sec. V.

For fully multiplier-less networks, the models with multiplier-less batch normalization outperform quantizing the activations to powers-of-two for the two larger ResNets (ResNet-34 and Resnet-50). However, for ResNet-18, quantizing the activations to powers-of-two leads to slightly better results than using multiplier-less batch normalization.

Better performance is achieved at the cost of keeping the multiplications in the batch normalization. Remarkably, ResNet-50 with 2-bit weights and 8-bit activations achieves 26.92% error rate which is only 1.05% worse than baseline. This memory footprint of this network (for parameters and activations) is only 7.35MB compared to 97.5MB for the full precision network. Furthermore, the number of multiplications is reduced by two orders of magnitude and most of them can even be replaced by bit-shifts.

We are not aware of any other published result with both power-of-two weights and quantized activations. In Table V, we compare LUT-Q against the following methods reported in the literature:

- The INQ approach, which also trains networks with power-of-two weights.
- The best results from literature with 4-bit or 2-bit weight

quantization and full precision or 8-bit activation quantization, collected by [17].

- The HAQ approach recently reported in [45] which achieves state-of-the-art performance for ResNet with quantized weights. The reported HAQ results are obtained with a method similar to deep compression [10] and by learning the optimal bitwidth for each layer in order to achieve the same network memory footprint than 2-bit and 4-bit networks. The activations, however, are not quantized for the reported HAQ results.

Note that we cannot directly compare the results from the *apprentice* method proposed in [17] and *LSQ* [22] because they do not quantize the first and last layers of the ResNets. We observe that LUT-Q always achieves better performance than other methods with the same weight and activation bitwidth except for ResNet-18 with 2-bit weight and 8-bit activation quantization. Additionally, LUT-Q networks are superior to the other networks in the sense that they combine activation quantization and power-of-two weights, i.e., most multiplications can be replaced by simpler bit-shifts.

E. Object Detection Experiments

We evaluated the performance of LUT-Q for object detection on the Pascal VOC [42] dataset. We use our implementation of YOLOv2 [46] as baseline. This network has a memory footprint of 200MB and achieves a *mean average precision* (mAP) of 72% on Pascal VOC. We were able to reduce the total memory footprint by a factor of 20 while maintaining the mAP above 70% by carrying out several modifications: replacing the feature extraction network with traditional residual networks

TABLE V: ImageNet: LUT-Q compared to other quantization methods. ✓: quasi multiplier-less. ×: unconstrained. The HAQ [45] results are for networks of the same size as 2-bit and 4-bit quantized networks but with variable bitwidth per layer.

Quantization		Source	Multiplier-less	Validation error		
Weights	Activations			ResNet-18	ResNet-34	ResNet-50
32-bit	32-bit	our implementation	×	31.0%	28.1%	25.9%
5-bit pow-2	32-bit	INQ [23]	✓	31.0%	-	25.2%
4-bit pow-2	32-bit	INQ [23]	✓	31.1%	-	-
~4-bit	32-bit	HAQ [45]	×	-	-	24.9%
4-bit	8-bit	Mishra <i>et. al</i> [17]	×	33.6%	29.7%	28.5%
4-bit pow-2	8-bit	LUT-Q pow-2 (ours)	✓	31.6%	28.1%	25.5%
2-bit pow-2	32-bit	INQ [23]	✓	34.0%	-	-
2-bit	32-bit	Mishra <i>et. al</i> [17]	×	33.4%	28.3%	26.1%
2-bit pow-2	32-bit	LUT-Q pow-2 (ours)	✓	31.8%	-	-
~2-bit	32-bit	HAQ [45]	×	-	-	29.4%
2-bit	8-bit	Mishra <i>et. al</i> [17]	×	33.9%	30.8%	29.2%
2-bit pow-2	8-bit	LUT-Q pow-2 (ours)	✓	35.8%	30.5%	26.9%

[44], replacing the convolution layers by factorized convolutions⁷, and finally applying LUT-Q in order to quantize the weights and activations of the network to 8 bit. Note that the feature extractor with residual and factorized layers was pre-trained on the ImageNet dataset, however, the LUT-Q quantization training was performed directly on the Pascal VOC object detection task.

When we further quantize weights and activations to 4 bit, we are able to reduce the total memory footprint down to just 1.72MB and still achieve a mAP of about 64%. An example of the object detection results for this 4-bit quantized network can be seen on Fig. 5.

F. Automatic Speech Recognition Experiments

In this section, we report our evaluation of LUT-Q on an acoustic model for *automatic speech recognition* (ASR). We use a fully connected network with wide layers, in contrast to convolution neural networks used in the previous image classification tasks, which typically have a smaller number of parameters per layer. Moreover, acoustic models are trained on very large datasets, typically one or two orders of magnitude larger than ImageNet. The ASR dataset used in these experiments has 20 times more samples than the ImageNet dataset.

For these experiments we use features extracted from the *Wall Street Journal* (WSJ) speech corpus [43]; the feature vectors have a size of 440. The task is a large classification task with 3407 different classes. The network output provides the *senone* probabilities that are used later by the *decoder* in order to recognize a speech sequence. The WSJ training dataset contains 26.5 million samples and the WSJ validation set contains 2.65 million samples. The used DNN model has seven wide fully connected layers with ReLU activations, the

TABLE VI: WSJ: Comparison of multiplier-less LUT-Q networks for acoustic modeling.

Method	bitwidth	Param. Memory	Val. Error
Full Precision	32-bit	923.7 Mbit	35.1%
LUT-Q pow2	4-bit	115.9 Mbit	35.6%
LUT-Q pow2	3-bit	87.1 Mbit	36.4%
LUT-Q pow2	2-bit	58.2 Mbit	36.8%
LUT-Q pow2	1-bit (binary)	29.3 Mbit	38.6%

output of the network is a *Softmax* layer. This model does not use Batch Normalization and does not have any Dropout layers.

The full precision network achieves 35.1% validation error and is used to initialize the quantization training. As the large layer size slows down the clustering in “Step 1” of Algorithm 1, we run the update of $\mathbf{A}^{(l)}$ and $\mathbf{d}^{(l)}$ only every 50 minibatches. For this task, activations were not quantized as their memory footprint remains small compared to the memory footprint of the weights for the large affine layers.

Table VI shows the performance of the LUT-Q quantization. These results confirm that LUT-Q can be successfully applied for different architectures – in this example for a network with wide affine layers.

VII. CONCLUSIONS AND FUTURE PERSPECTIVES

We have presented LUT-Q, a novel approach for the reduction of size and computations for deep neural networks. After each minibatch update, the quantization values and assignments are also updated by a clustering step. We show that our LUT-Q approach can be efficiently used for pruning weight matrices and training multiplier-less networks as well. We also introduce a new form of batch normalization that avoids the need for multiplications during inference.

⁷Each convolution is replaced by a sequence of pointwise, depthwise and pointwise convolutions (similarly to MobileNetV2 [6]).

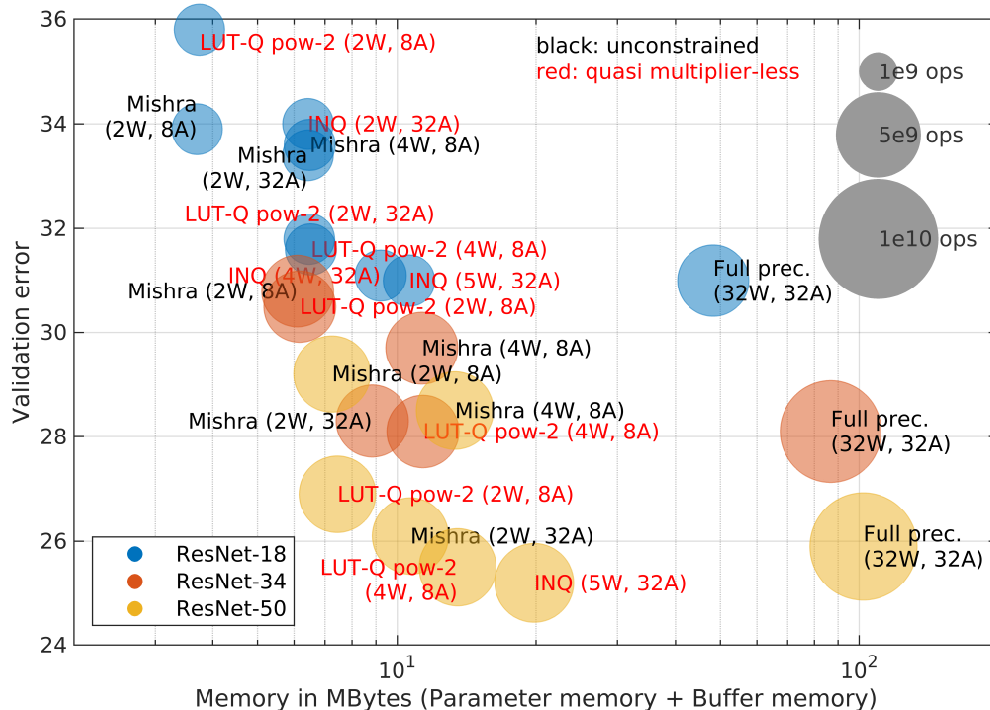


Fig. 4: ImageNet: LUT-Q graphically compared to other quantization methods: the bubble size gives number of operations, i.e., the sum of #additions and #multiplications; “ n_1W, n_2A ” refers to n_1 -bit weights and n_2 -bit activations.

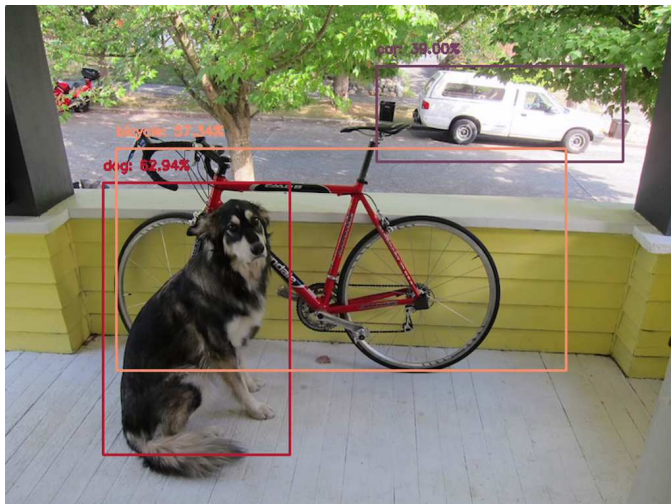


Fig. 5: Example of object detection results with a very low memory footprint DNN. Weights and activations are quantized to 4 bit. The total memory footprint is only 1.72MB (more than 100 times smaller than YOLOv2 [46]).

As argued in this paper, if weights are quantized to very low bitwidth, the activations may dominate the memory footprint of the network during inference. Therefore, we perform our experiments with activations uniformly quantized to 8 bit. We believe that a non-uniform activation quantization, where the quantization values are learned parameters, will help quantize activations to lower precision. This is one of the promising directions for continuing this work.

Recently, several papers have shown the benefits of training

quantized networks using a *distillation* strategy [17], [47]. Distillation is compatible with our training approach and we are planning to investigate LUT-Q training with distillation.

REFERENCES

- [1] F. Cardinaux, S. Uhlich, K. Yoshiyama, J. Alonso García, S. Tiedemann, T. Kemp, and A. Nakamura, “Iteratively training look-up tables for network quantization,” *NIPS 2018 Workshop on Compact Deep Neural Network Representation with Industrial Applications*, *arXiv preprint arXiv:1811.05355*, 2018.
- [2] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.
- [3] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky, “Speeding-up convolutional neural networks using fine-tuned cp-decomposition,” *arXiv preprint arXiv:1412.6553*, 2014.
- [4] C. Yunpeng, J. Xiaojie, K. Bingyi, F. Jiashi, and Y. Shuicheng, “Sharing residual units through collective tensor factorization in deep neural networks,” *arXiv preprint arXiv:1703.02180*, 2017.
- [5] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan *et al.*, “Searching for mobilenetv3,” *arXiv preprint arXiv:1905.02244*, 2019.
- [6] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “Mobilenetv2: Inverted residuals and linear bottlenecks,” in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.
- [7] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel, “Handwritten digit recognition with a back-propagation network,” in *Advances in neural information processing systems*, 1990, pp. 396–404.
- [8] S. Han, J. Pool, J. Tran, and W. Dally, “Learning both weights and connections for efficient neural network,” in *Advances in Neural Information Processing Systems*, 2015, pp. 1135–1143.
- [9] L. Mauch and B. Yang, “Least-squares based layerwise pruning of convolutional neural networks,” in *2018 IEEE Statistical Signal Processing Workshop (SSP)*. IEEE, 2018, pp. 60–64.
- [10] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding,” in *International Conference on Learning Representations (ICLR)*, 2016.

- [11] K. Ullrich, E. Meeds, and M. Welling, "Soft weight-sharing for neural network compression," in *International Conference on Learning Representations (ICLR)*, 2017.
- [12] W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen, "Compressing neural networks with the hashing trick," in *International Conference on Machine Learning (ICML)*, 2015, pp. 2285–2294.
- [13] S. Uhlich, L. Mauch, K. Yoshiyama, F. Cardinaux, J. Alonso García, S. Tiedemann, T. Kemp, and A. Nakamura, "Differentiable quantization of deep neural networks," *arXiv preprint arXiv:1905.11452*, 2019.
- [14] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Advances in Neural Information Processing Systems*, 2015, pp. 3123–3131.
- [15] F. Li, B. Zhang, and B. Liu, "Ternary weight networks," in *NIPS Workshop on Efficient Methods for Deep Neural Networks (EMDNN)*, 2016.
- [16] A. Mishra, E. Nurvitadhi, J. J. Cook, and D. Marr, "WRPN: Wide reduced-precision networks," *arXiv preprint arXiv:1709.01134*, 2017.
- [17] A. Mishra and D. Marr, "Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy," *International Conference on Learning Representations (ICLR)*, 2018.
- [18] S. J. Nowlan and G. E. Hinton, "Simplifying neural networks by soft weight-sharing," *Neural computation*, vol. 4, no. 4, pp. 473–493, 1992.
- [19] C. Louizos, K. Ullrich, and M. Welling, "Bayesian compression for deep learning," *Conference on Neural Information Processing Systems (NIPS)*, 2017.
- [20] J. Achterhold, J. M. Koehler, A. Schmeink, and T. Genewein, "Variational network quantization," *International Conference on Learning Representations (ICLR)*, 2018.
- [21] J. Gu, C. Li, B. Zhang, J. Han, X. Cao, J. Liu, and D. S. Doermann, "Projection convolutional neural networks for 1-bit cnns via discrete back propagation," *CoRR*, vol. abs/1811.12755, 2018.
- [22] S. K. Esser, J. L. McKinstry, D. Bablani, R. Appuswamy, and D. S. Modha, "Learned step size quantization," *CoRR*, vol. abs/1902.08153, 2019. [Online]. Available: <http://arxiv.org/abs/1902.08153>
- [23] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless CNNs with low-precision weights," in *International Conference on Learning Representations (ICLR)*, 2017.
- [24] E. Fiesler, A. Choudry, and H. J. Caulfield, "Weight discretization paradigm for optical neural networks," in *Optical interconnections and networks*, vol. 1281, 1990, pp. 164–174.
- [25] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *arXiv preprint arXiv:1609.07061*, 2016.
- [26] P. Yin, J. Lyu, S. Zhang, S. Osher, Y. Qi, and J. Xin, "Understanding straight-through estimator in training activation quantized neural nets," *arXiv preprint arXiv:1903.05662*, 2019.
- [27] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.
- [28] Y. Nesterov, "A method of solving a convex programming problem with convergence rate $o(1/k^2)$," in *Soviet Mathematics Doklady*, vol. 27, no. 2, 1983, pp. 372–376.
- [29] R. Reed, "Pruning algorithms - a survey," *IEEE Transactions on Neural Networks*, vol. 4, no. 5, pp. 740–747, 1993.
- [30] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: Imagenet classification using binary convolutional neural networks," *arXiv preprint arXiv:1603.05279*, 2016.
- [31] V. Sindhwani, T. Sainath, and S. Kumar, "Structured transforms for small-footprint deep learning," in *Advances in Neural Information Processing Systems*, 2015, pp. 3088–3096.
- [32] M. Moczulski, M. Denil, J. Appleyard, and N. de Freitas, "ACDC: A structured efficient linear layer," *arXiv preprint arXiv:1511.05946*, 2015.
- [33] B. A. White and M. I. Elmasry, "The digi-neocognitron: a digital neocognitron neural network model for VLSI," *IEEE Transactions on Neural Networks*, vol. 3, no. 1, pp. 73–85, 1992.
- [34] H. K. Kwan and C. Tang, "Multiplierless multilayer feedforward neural network design suitable for continuous input-output mapping," *Electronics Letters*, vol. 29, no. 14, pp. 1259–1260, 1993.
- [35] M. Marchesi, G. Orlandi, F. Piazza, and A. Uncini, "Fast neural networks without multipliers," *IEEE Transactions on Neural Networks*, vol. 4, no. 1, pp. 53–62, 1993.
- [36] P. Y. Simard and H. P. Graf, "Backpropagation without multiplication," in *Advances in Neural Information Processing Systems (NIPS)*, 1994, pp. 232–239.
- [37] M. Courbariaux and Y. Bengio, "Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1," *arXiv preprint arXiv:1602.02830*, 2016.
- [38] S. Zhou, Z. Ni, X. Zhou, H. Wen, Y. Wu, and Y. Zou, "DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *arXiv preprint arXiv:1606.06160*, 2016.
- [39] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.
- [40] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," 2009.
- [41] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.
- [42] M. Everingham, L. Van Gool, C. K. I. Williams, J. M. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International Journal of Computer Vision*, vol. 88, pp. 303–338, 2009.
- [43] D. B. Paul and J. M. Baker, "The design for the wall street journal-based csr corpus," in *Proceedings of the workshop on Speech and Natural Language*. Association for Computational Linguistics, 1992, pp. 357–362.
- [44] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.
- [45] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "HAQ: hardware-aware automated quantization," *CoRR*, vol. abs/1811.08886, 2018. [Online]. Available: <http://arxiv.org/abs/1811.08886>
- [46] J. Redmon and A. Farhadi, "Yolo9000: better, faster, stronger," *arXiv preprint*, 2017.
- [47] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," in *NIPS Deep Learning and Representation Learning Workshop*, 2015.