A New Clustering-Based Technique for the Acceleration of Deep Convolutional Networks

Erion Vasilis Pikoulis[1,2], Christos Mavrokefalidis[1,2], Aris S. Lalos[1]
[1]Industrial Systems Institute, Patras Science Park, Greece
[2]Computer Engineering and Informatics Dept., University of Patras, Greece
Emails: {pikoulis,maurokef}@ceid.upatras.gr, lalos@isi.gr

***Abstract*—Deep learning and especially the use of Deep Neural Networks (DNNs) provides impressive results in various regression and classification tasks. However, to achieve these results, there is a high demand for computing and storing resources. This becomes problematic when, for instance, real-time, mobile applications are considered, in which the involved (embedded) devices have limited resources. A common way of addressing this problem is to transform the original large pre-trained networks into new smaller models, by utilizing Model Compression and Acceleration (MCA) techniques. Within the MCA framework, we propose a clustering-based approach that is able to increase the number of employed centroids/representatives, while at the same time, have an acceleration gain compared to conventional, k-means based approaches. This is achieved by imposing a special structure to the employed representatives, which is enabled by the particularities of the problem at hand. Moreover, the theoretical acceleration gains are presented and the key system hyper-parameters that affect that gain, are identified. Extensive evaluation studies carried out using various state-of-the-art DNN models trained in image classification, validate the superiority of the proposed method as compared for its use in MCA tasks.**

I. Introduction

Deep Neural Networks (DNN) [1] have emerged recently as a central ingredient in many modern artificial intelligence applications [2], [3], [4], [5], [6]. However, the impressive performance that has been reported in the literature, is closely related to the size of the DNNs, which, for state-of-the-art models can reach to tens, or even hundreds of millions of parameters (e.g., 138 millions of parameters are used by the Visual Geometry Group (VGG) DNN [7]). This leads to vast computing and storage requirements during both the training phase of the DNNs and (most importantly) the operational (or inference) phase, i.e., when the DNNs are actually employed. In real applications, these requirements are usually tackled via high-performance computing platforms [8] that include graphics processing units (GPUs).

Nowadays, there is an increasing interest in blending deep learning and mobile computing in which platforms of limited resources are employed [9], including smart devices (such as phones, watches, and embedded sensors). In order for these platforms to exploit the DNN gains, especially, during the inference phase, two main lines of research can be identified. In the first one, new compact, smaller DNN models [10] are designed or searched for in a design space for the applications at hand (e.g., SqueezeNet [11], MobileNets [12], and EfficientNet [13]). In the second line, existing pre-trained, highly performing DNN models (e.g., AlexNet [14], VGG [7], Residual Net (ResNet) [15], and many more) are transformed into new smaller models by utilizing Model Compression and Acceleration (MCA) techniques [10], [16], [17], [18]. The importance of this line of research stems from the fact that, apart from their standalone use, state-of-the-art, pre-trained DNN models can also be utilized as back-bone modules in models designed for different (but similar in nature) applications. For example, the convolutional layers of AlexNet and VGG (that are originally trained for image classification tasks), constitute the core modules of the R-CNN [19] and Fast R-CNN [20], respectively, object detectors.

The MCA-related literature has been increasing in recent years and there are numerous surveys that provide a comprehensive overview of the area ([10], [16], [17], [18]). Roughly speaking (and by no means being exhaustive), some of the earliest works proposed parameter pruning, in which, unimportant parameters (e.g., filters [21], [22]) are removed and, hence, not considered during the inference phase of the DNN deployment. Other works focus on limiting the representation of the involved parameter by reducing their bit-width or increasing common representations via the design of codebooks (e.g., scalar [23], vector and product quantization [24]). Finally, several works employ tensor / matrix decompositions on the involved quantities (e.g., filters) into factors by utilizing, for instance, low-rankness [25].

In this paper, a new MCA technique for pre-trained DNNs is described and evaluated. The highlights of the paper are outlined as follows:

- We propose a novel codebook design procedure that, for the same target acceleration, leads to larger codebooks than the typically used k-means-based approaches, thus improving considerably the quantization error.
- This is achieved by imposing a special structure to the learned codewords based on a Dictionary Learning framework.
- Theoretical analysis is provided for determining the parameters that dictate the structure of the codebook.
- The efficacy of the proposed MCA technique is assessed on three state-of-the-art DNN models (VGG, ResNet, SqueezeNet) on the demanding ILSVRC2012 dataset [14], achieving up to 100% (or $2\times$) acceleration gain over the conventional approach, for the same quantization error.

- The application of the proposed technique for the selected DNNs results in significantly accelerated models with limited performance degradation.

In the following, first, the relevant bibliography is presented and the positioning of this paper is described. Then, in Sec. III, the problem is formulated, while, in Sec. IV, the proposed technique is explained. Simulation results are presented in Sec. V. Finally, Sec. VI concludes the paper.

II. Relevant bibliography

The work in this paper is related to MCA techniques that utilize a codebook for quantizing network parameters. The codebook contains representative quantities (called codewords, centroids, etc.) for the parameters to be approximated (e.g., vectors of weights). As multiple network parameters are mapped to a single representative quantity, such approaches are also called parameter sharing techniques.

Towards this end, [24] proposed vector quantization methods (using the k-means algorithm) for estimating the desired codebook with the aim to compress the weights of fully connected layers. In [26], a three stage method was presented in which parameter pruning was followed by a codebook design for parameter quantization and concluded with Huffman coding. In [27], a new codebook design was devised that improved upon or generalized works like [24], [26]. In [27], the proposed approach could be applied to both convolutional and fully connected layers, while two cost-functions for estimating the involved codebooks, were devised that minimized the quantization error of the weights and the representation error of the layers' output (namely, the error between the outputs of the original and the accelerated layers for a given input), respectively. In [28], the proposed technique operates on scaled versions of the 2D kernels for estimating the desired centroids using the k-means algorithm. Then, during fine-tuning, both the scales and the centroids are considered free variables to be updated. In [29], [30] and [31], proper regularization terms are introduced and via re-training procedures the resulted weights can be more easily clustered using the k-means algorithm. The work in [32] adopts product quantization and focuses on the representation error of the layer outputs. Finally, in [33], a methodology is devised for determining the size of the codebooks by introducing a sensitivity analysis per layer in order to assess the impact of compression on the accuracy performance.

The proposed MCA technique is mostly related to works like [24], [26], and [27]. Here, however, by exploiting the special structure of the weights to be quantized, improved quantization error is achieved. For assessing the performance, [27] is selected as a baseline. Moreover, although, here, a new MCA technique is outlined, the proposed dictionary-learning approach for designing the codebook, can be actually utilized by all the works mentioned above (i.e., instead of the commonly used k-means algorithm).

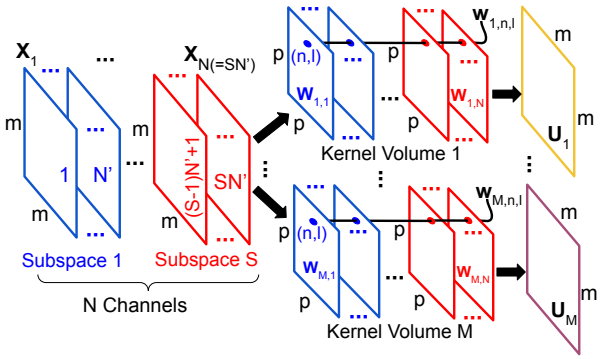


Fig. 1: The linear operation of a single convolutional layer.

III. Problem Formulation

The core operation performed by a convolutional layer and the involved quantities, are depicted in Fig. 1. In particular, the input volume consists of N channels $\mathbf{X}_i$, $i = 1, 2, \ldots, N$. Also, there are M kernel volumes and the k-th kernel volume has N filters $\mathbf{W}_{k,i}$, $k = 1, 2, \ldots, M$, $i = 1, 2, \ldots, N$. For simplicity, it is assumed that the dimensions of the $\mathbf{X}_i$'s, $\mathbf{W}_{k,i}$'s and $\mathbf{U}_k$'s are $m \times m$, $p \times p$, and $m \times m$, respectively.

The convolution of the input volume with the k-th kernel volume is given by

$$\mathbf{U}_k = \sum_{i=1}^{N} \mathbf{X}_i \star \mathbf{W}_{k,i}, \tag{1}$$

where $\star$ denotes the 2D convolution operation.

In order to proceed and describe the entities to be clustered (i.e., coded by the codebook that will be designed), (1) is rewritten in order to describe the (i,j)-th element $\mathbf{U}_k[i,j]$ as

$$\mathbf{U}_k[i,j] = \sum_{n,l \in \mathcal{R}_{i,j}} \mathbf{x}_{n,l}^{\mathrm{T}} \mathbf{w}_{k,i-n,j-l}, \tag{2}$$

where $\mathbf{x}_{i,j} = [\mathbf{X}_1[i,j], \ldots, \mathbf{X}_N[i,j]]^{\mathrm{T}}$ contains the samples at the (i,j)-th position of all input channels. Also, $\mathbf{w}_{k,u,v} = [\mathbf{W}_{k,1}[u,v], \ldots, \mathbf{W}_{k,N}[u,v]]^{\mathrm{T}}$ contains the filter weights at the (u,v) position of all channels in the k-th kernel volume. Finally, the set $\mathcal{R}_{i,j}$ contains p^2 indices around the position (i,j).

In the product quantization framework, the N-dimensional vector space is partitioned into S, N'-dimensional subspaces with $N' = N/S$, so that the s-th subspace spans dimensions $[(s-1)N'+1, \ldots, sN']$, $s = 1, \ldots, S$. Let us now partition vectors $\mathbf{x}_{i,j}$, $\mathbf{w}_{k,u,v}$ defined in Eq. (2), accordingly, as

$$\mathbf{x}_{i,j} = [(\mathbf{x}_{i,j}^1)^{\mathrm{T}}, \ldots, (\mathbf{x}_{i,j}^S)^{\mathrm{T}}]^{\mathrm{T}}, \tag{3}$$

$$\mathbf{w}_{k,u,v} = [(\mathbf{w}_{k,u,v}^1)^{\mathrm{T}}, \ldots, (\mathbf{w}_{k,u,v}^S)^{\mathrm{T}}]^{\mathrm{T}}, \tag{4}$$

where each of the sub-vectors lies in N'-D space. Then, (2) can be rewritten as

$$\mathbf{U}_k[i,j] = \sum_{s=1}^{S} \sum_{n,l \in \mathcal{R}_{i,j}} (\mathbf{x}_{n,l}^s)^{\mathrm{T}} \mathbf{w}_{k,i-n,j-l}^s, \tag{5}$$

where the inner sum denotes the contribution of the s-th subspace to the k-th convolutional output, at position (i, j).

For each subspace, the goal of product quantization is to perform vector quantization to the Mp^2 kernel sub-vectors lying in s-th subspace, and cluster them into $K_s \ll Mp^2$ clusters. This way, each sub-vector is represented by the centroid of the cluster it belongs to, reducing accordingly the number of required dot-products. To be more specific, the acceleration occurs because the original dot-products between the input and the Mp^2 kernel sub-vectors, are approximated by the ones between the input and the K_s centroids/representatives.

IV. Dictionary-learning-based weight clustering

In this section, first, the proposed codebook structure for approximating the kernel sub-vectors, is described and discussed in comparison with the conventional codebook structure that appears in current literature. This discussion is also extended towards the gains that are achieved through a computational complexity analysis. Then, the proposed codebook design is approached as a Dictionary Learning (DL) problem, which actually treats the k-means-based conventional codebook design as a special case. The latter will be referred to as Vector Quantization (VQ) in the following. Finally, some implementation details are described concerning the initialization of the involved parameters when applying the proposed DL solution.

A. Proposed approximation

Let us first define the kernel approximation scheme incurred by the conventional codebook structure, as follows:

$$\mathbf{W} \approx \mathbf{C}\mathbf{\Gamma}, \tag{6}$$

where the columns of $\mathbf{W} \in \mathbb{R}^{N' \times p^2 M}$, $\mathbf{C} \in \mathbb{R}^{N' \times K_{vq}}$, and $\mathbf{\Gamma} \in \mathbb{R}^{K_{vq} \times p^2 M}$, contain the kernel sub-vectors (of a particular subspace), the representatives (or cluster centroids), and assignment vectors, respectively. Specifically, each column of $\mathbf{\Gamma}$ has exactly one non-zero element, equal to 1, meaning that each column of $\mathbf{W}$ is approximated by one column of $\mathbf{C}$. Thus, in the conventional case, the Mp^2 sub-vectors are approximated by $K_{vq} \ll p^2 M$ representatives, using the codebook $\mathbf{C}$.

Instead, in this paper, the following approximation is proposed:

$$\mathbf{W} \approx \mathbf{D}\mathbf{\Lambda}\mathbf{\Gamma}, \tag{7}$$

where $\mathbf{W} \in \mathbb{R}^{N' \times p^2 M}$ and $\mathbf{\Gamma} \in \mathbb{R}^{K_{dl} \times p^2 M}$ are defined as in (6), while $\mathbf{D} \in \mathbb{R}^{N' \times L_{dl}}$ and $\mathbf{\Lambda} \in \mathbb{R}^{L_{dl} \times K_{dl}}$ denote the dictionary and the matrix of sparse coefficients, respectively. Specifically, the columns of $\mathbf{D}$ (called dictionary atoms), are normalized, while $\mathbf{\Lambda}$ is a sparse matrix in the sense that each of its columns contains at most α non-zero elements, with α being the sparsity level. Thus, under the proposed scheme, the $p^2 M$ sub-vectors are approximated via K_{dl} representatives contained in the codebook $\mathbf{D}\mathbf{\Lambda}$. In turn, these representatives are obtained as linear combinations of at most α atoms from a dictionary of size L_{dl}, with $L_{dl} < K_{dl} \ll p^2 M$. Note that the matrix approximation defined in (7) can be viewed as a special case of the general Dictionary Learning (DL) problem [34], which is why we call our acceleration technique as a DL-based one.

It should be noted that, in the general case, the proposed approximation requires more representatives than the conventional approach (i.e., $K_{dl} > K_{vq}$), for achieving the same quantization (i.e. approximation) error. This is by definition since the conventional codebook $\mathbf{C}$ is obtained in an unconstrained fashion, while the proposed codebook $\mathbf{D}\mathbf{\Lambda}$ follows a specific structure. Although it seems counter intuitive (in the sense that the proposed approximation is less efficient than the conventional one, in the general case), due to the particularities of the problem at hand, namely, due to the fact that the "data points" in $\mathbf{W}$ are in fact filters used in convolution operations, the proposed approximation results actually in significantly higher acceleration ratios for the same quantization error, as it is going to be demonstrated. This is because, due to the linearity of the operations performed in the convolutional layer, the sparse coefficients in $\mathbf{\Lambda}$ need only be applied to the convolution between the input and the dictionary atoms in $\mathbf{D}$, instead of the atoms themselves. This endows the proposed approximation scheme with the flexibility to use a number of representatives K_{dl} that is several times larger than K_{vq}, while restricting the size of the dictionary (so that $L_{dl} \ll K_{vq}$) thus reducing the number of "heavy" convolutions, as it will become clearer in the following subsection.

B. Computational complexity analysis

Since the core operations of a DNN are ultimately translated into dot-products between input and kernel vectors, the computational complexity of a DNN is usually measured in terms of the number of Multiply and Accumulate (MAC) operations. A MAC is dominated by the involved multiplication (MUL), which is a significantly "heavier" computation than the involved addition. As such, in the subsequent analysis, in order to compare the techniques on a common ground, MAC and MUL operations are going to be used interchangeably, i.e, a MAC will be considered equivalent to one MUL, so that the computational cost is measured as the number of MULs.

Let us first begin by examining the computational complexity of the original layer, where, by arranging the m^2 input sub-vectors of the s-th subspace in the columns of a matrix $\mathbf{X} \in \mathbb{R}^{N' \times m^2}$, we see that the convolution operation involves the calculation of a matrix product of the form:

$$\mathbf{Y} = \mathbf{X}^{\mathrm{T}}\mathbf{W}, \tag{8}$$

where $\mathbf{W}$ contains the kernel sub-vectors (as defined in (6)), followed by the appropriate summation of the dot-products according to (5). Since calculating $\mathbf{Y}$ requires $m^2 p^2 M N'$ multiplications, the overall (i.e. for all S subspaces) computational complexity of the original convolutional layer, measured in MULs, is obtained as:

$$\mathcal{T}_o = m^2 p^2 M (S N') = m^2 p^2 M N. \tag{9}$$

In the VQ case (described by the approximation in (6)), the approximate $\mathbf{Y}$ is obtained as:

$$\mathbf{Y} \approx (\mathbf{X}^{\mathrm{T}}\mathbf{C})\mathbf{\Gamma}, \tag{10}$$

namely, it involves calculating the dot-products between input sub-vectors and representatives and then "plugging" the results appropriately, according to the columns of $\mathbf{\Gamma}$. Calculating $\mathbf{X}^{\mathrm{T}}\mathbf{C}$ requires only $m^2N'K_{vq}$ MULs (as $k_{vq} \ll Mp^2$), meaning that the overall computational complexity for the approximate convolutional output is reduced to:

$$\mathcal{T}_{vq}(K_{vq}) = m^2(SN')K_{vq} = m^2NK_{vq}. \tag{11}$$

Finally, for the DL-based approximation scheme, we can write:

$$\mathbf{Y} \approx \left((\mathbf{X}^{\mathrm{T}}\mathbf{D})\mathbf{\Lambda}\right)\mathbf{\Gamma}, \tag{12}$$

meaning that, in this case, calculating the approximate $\mathbf{Y}$ is a two-stage operation. First, we calculate $\mathbf{X}^{\mathrm{T}}\mathbf{D}$, i.e., the dot-products between the input and the dictionary atoms, which requires $m^2N'L_{dl}$ MULs, where L_{dl} denotes the dictionary size. Subsequently, the results are combined according to the columns of $\mathbf{\Lambda}$, which requires $\alpha\, m^2K_{dl}$ additional MULs Thus, the overall computational complexity for the approximate convolutional output in the DL case, is obtained as:

$$\mathcal{T}_{dl}(K_{dl}, L_{dl}, \alpha) = m^2(NL_{dl} + \alpha SK_{dl}). \tag{13}$$

Accordingly, the acceleration ratio (namely, the ratio of original vs accelerated computational complexities) achieved by the two rival approaches, can be written as follows:

$$\rho_{vq} \equiv \frac{\mathcal{T}_o}{\mathcal{T}_{vq}} = \frac{p^2M}{K_{vq}} \tag{14}$$

$$\rho_{dl} \equiv \frac{\mathcal{T}_o}{\mathcal{T}_{dl}} = \frac{p^2M}{L_{dl} + \frac{\alpha}{N'}K_{dl}}. \tag{15}$$

Of great interest is also the relative acceleration between the proposed DL-based and the VQ approach, which will also provide rules for selecting the free parameters of the proposed technique. First, in order to have a better representation error, we set the number of representatives used by the proposed technique as a multiple of the representatives used by the VQ approach, i.e., $K_{dl} = c\,K_{vq}$, $c > 1$. Then, using (11), (13), we can see that for the DL-based approximation to achieve at least the same acceleration with the VQ technique, i.e., for $\mathcal{T}_{dl}(K_{dl}, L_{dl}, \alpha) \leq \mathcal{T}_{vq}(K_{vq})$ to hold, the following inequality should hold regarding the size of the used dictionary:

$$L_{dl} \leq K_{vq}\left(1 - \frac{\alpha\, c}{N'}\right). \tag{16}$$

As we are going to demonstrate in our experimental results for various combinations of the coefficient c and sparsity level α, and for (16) holding with equality (i.e., for the two rivals achieving the same acceleration ratio), the proposed technique leads to a significantly better approximation of the original weights, which ultimately translates into better classification accuracy for the accelerated DNNs.

C. Proposed algorithm

For deriving the matrix factorizations described by the proposed weight factorization in (7), the quantization error between the original $\mathbf{W}$ and its approximate version is minimized. In particular, the following minimization problem is defined.

$$\begin{aligned} \min_{\mathbf{D},\mathbf{\Lambda},\mathbf{\Gamma}} \quad & ||\mathbf{W} - \mathbf{D\Lambda\Gamma}||_F^2 \\ \text{s.t.} \quad & ||\mathbf{d}_i||_2^2 = 1,\ i = 1, \ldots, L_{dl}, \\ & ||\boldsymbol{\lambda}_i||_0 \leq \alpha,\ i = 1, \ldots, K_{dl}, \\ & ||\boldsymbol{\gamma}_i||_0 = 1,\ \mathbf{1}^{\mathrm{T}}\boldsymbol{\gamma}_i = 1, \quad i = 1 \ldots Mp^2, \end{aligned} \tag{17}$$

where $||\cdot||_F$, $||\cdot||_2$, $||\cdot||_0$ denote the Frobenius, l_2, and l_0 norms, respectively, while the last constraint ensures that the elements of $\mathbf{\Gamma}$ take values in $\{0, 1\}$ and each of its columns has exactly one non-zero element.

In order to solve (17), we follow a strategy of alternating optimizations over each set of parameters, leading to the following three sub-problems:

a) Sparse coding: With $\mathbf{D}$, $\mathbf{\Gamma}$ fixed, the loss function in (17) can be rewritten as follows:

$$\left|\left|\mathbf{W} - \sum_{i=1}^{K_{dl}} (\mathbf{D}\boldsymbol{\lambda}_i)\widetilde{\boldsymbol{\gamma}}_i\right|\right|_F = \sum_{i=1}^{K_{dl}} \left|\left|\mathbf{W}_{I_i} - (\mathbf{D}\boldsymbol{\lambda}_i)\,\mathbf{1}^{\mathrm{T}}\right|\right|_F, \tag{18}$$

where $\mathbf{y}_i$, $\widetilde{\mathbf{y}}_i$ is used to denote the i-th column and row of matrix $\mathbf{Y}$, respectively, $I_i = \{j | \gamma_{ij} = 1\}$ is the set of indices of the non-zero elements of $\widetilde{\boldsymbol{\gamma}}_i$, $\mathbf{W}_{I_i}$ is the submatrix formed by the columns of $\mathbf{W}$ indexed by I_i, while $\mathbf{1}$ denotes the all-ones vector of dimension $|I_i|$.

Observing (18), due to the l_0-norm constraints on $\mathbf{\Gamma}$, I_i's, $i = 1, \ldots, K_{dl}$, are a partition of $\{1, 2, \ldots, p^2M\}$, meaning that the minimization of (18) over $\mathbf{\Lambda}$ is translated into K_{dl} separate sub-problems, one for each of $\mathbf{\Lambda}$'s columns:

$$\begin{aligned} \min_{\boldsymbol{\zeta}} \quad & ||\mathbf{W}_{I_i} - (\mathbf{D}\boldsymbol{\zeta})\,\mathbf{1}^{\mathrm{T}}||_F^2 \\ \text{s.t.} \quad & ||\boldsymbol{\zeta}||_0 \leq \alpha. \end{aligned} \tag{19}$$

In order to solve (19), we follow an Orthogonal Matching Pursuit (OMP) approach, which builds the support of the sparse representation (non-zero elements of $\boldsymbol{\lambda}_i$), by adding one dictionary atom at a time, up to α atoms [34].

This sparse coding sub-problem is outlined in lines 5-12 of Table I. There, $\mathcal{S}$ denotes a set of non-zero indices, while $\mathbf{D}_{\mathcal{S}}$ and $\boldsymbol{\zeta}_{\mathcal{S}}$ contain the columns of $\mathbf{D}$ and the elements of $\boldsymbol{\zeta}$ indexed by $\mathcal{S}$, respectively.

b) Dictionary update: With $\mathbf{\Lambda}$, $\mathbf{\Gamma}$ fixed, we write the loss function in (17) as follows:

$$\mathcal{E} = ||\mathbf{W} - \mathbf{DG}||_F^2 = \left|\left|\mathbf{W} - \sum_{i=1}^{L_{dl}} \mathbf{d}_i\widetilde{\mathbf{g}}_i\right|\right|_F^2, \tag{20}$$

where $\widetilde{\mathbf{g}}_i$ denotes the i-th row of $\mathbf{G} = \mathbf{\Lambda\Gamma}$. Thus, the dictionary update step translates to minimizing $\mathcal{E}$ under the l_2-norm constraint for the dictionary atoms. In order to solve this problem, we follow the coordinate-descent-based approach

1: **procedure** DL-based sub-space clustering
2: **Input**: original sub-vectors $\mathbf{W}$, # of representatives K_{dl} dictionary size L_{dl}, sparsity level α.
3: Obtain initial solution $\{\mathbf{D}_0, \mathbf{\Lambda}_0, \mathbf{\Gamma}_0\}$
4: **repeat**
5: **for** $i = 1 : K_{dl}$ *//Sparse Coding*
6: **Initialize**: $\mathbf{E} = \mathbf{W}_{I_i}$, $\mathcal{S} = \emptyset$
7: **for** $j = 1 : \alpha$
8: Build new support: $\mathcal{S} \leftarrow \mathcal{S} \cup \{k\}$, where $k = \arg\max_{j \notin \mathcal{S}} |\mathbf{1}^{\mathrm{T}} \mathbf{E}^{\mathrm{T}} \mathbf{d}_j|$.
9: Find new solution: $\boldsymbol{\zeta}_{\mathcal{S}}$ by solving $\min_{\boldsymbol{\xi}} ||\mathbf{W}_{I_i} - \mathbf{D}_{\mathcal{S}}\, \boldsymbol{\xi}\, \mathbf{1}^{\mathrm{T}}||_F^2$.
10: Update residual: $\mathbf{E} = \mathbf{W}_{I_i} - \mathbf{D}_{\mathcal{S}}\, \boldsymbol{\zeta}_{\mathcal{S}}\, \mathbf{1}^{\mathrm{T}}$.
11: **end**
12: **end**
13: **Initialize**: $\mathbf{E} = \mathbf{W} - \mathbf{D}\mathbf{\Lambda}\mathbf{\Gamma}$ *//Dictionary Update*
14: **for** $i = 1 : L_{dl}$
15: Modify error: $\mathbf{F} = \mathbf{E}_{I_i} + \mathbf{d}_i\, \widetilde{\mathbf{g}}_{i,I_i}$.
16: Update i-th atom: $\mathbf{d}_i = \frac{\mathbf{F}\,(\widetilde{\mathbf{g}}_{i,I_i})^{\mathrm{T}}}{||\mathbf{F}\,(\widetilde{\mathbf{g}}_{i,I_i})^{\mathrm{T}}||}$.
17: Re-compute error: $\mathbf{E}_{I_i} = \mathbf{F} - \mathbf{d}_i\, \widetilde{\mathbf{g}}_{i,I_i}$.
18: **end**
19: **for** $i = 1 : p^2 M$ *//Assignment Update*
20: Update $\boldsymbol{\gamma}_i$ solving $j_i = \arg\min_{j \in \{1,\ldots,K_{dl}\}} ||\mathbf{w}_i - \widetilde{\mathbf{c}}_j||_2$.
21: **end**
22:**Until**: a maximum number of iterations is met.
23:**Return**: $\mathbf{D}$, $\mathbf{\Lambda}$, $\mathbf{\Gamma}$.
24:**end procedure**

TABLE I: Proposed algorithm for solving (17)

outlined in Algorithm 3.5 of [34]. This sub-problem is described in lines 13-18 of Table I.

c) Assignment update: With $\mathbf{D}$, $\mathbf{\Lambda}$ fixed, the loss function in (17) takes the following form:

$$\mathcal{E} = ||\mathbf{W} - \widetilde{\mathbf{C}}\mathbf{\Gamma}||_F^2 = \sum_{i=1}^{p^2 M} ||\mathbf{w}_i - \widetilde{\mathbf{C}}\boldsymbol{\gamma}_i||_2^2 \qquad (21)$$

where $\widetilde{\mathbf{C}} = \mathbf{D}\mathbf{\Lambda}$ is the $N' \times K_{dl}$ matrix of representatives. Taking into account the special structure of $\mathbf{\Gamma}$, updating $\boldsymbol{\gamma}_i$ is equivalent to determining the position j_i of its non-zero (unity) element, which simply assigns $\mathbf{w}_i$ to its closest representative. This sub-problem is described in lines 19-21 of Table I.

D. Initial Solution and Parameter Selection

In order to provide an initial solution $\mathbf{D}_0$, $\mathbf{\Lambda}_0$, $\mathbf{\Gamma}_0$, to the proposed acceleration technique, we work as follows:

1) We obtain a clustering of the original kernel sub-vectors into K_{dl} clusters by minimizing $||\mathbf{W} - \mathbf{C}\mathbf{\Gamma}||_F$ under the constraints on the assignment matrix $\mathbf{\Gamma}$ stated in (17). This problem can be solved by using the k-means algorithm.
2) We then obtain a sparse representation of the cluster centroids in $\mathbf{C}$ as $\mathbf{C} \approx \mathbf{D}_0\mathbf{\Lambda}_0$, by using a dictionary of size L_{dl} and target sparsity α. This problem can be solved with standard DL techniques such as the ones described earlier.
3) Finally, we obtain the initial assignment matrix $\mathbf{\Gamma}_0$ by assigning each of the sub-vectors in $\mathbf{W}$ to its closest representative in $\widetilde{\mathbf{C}} = \mathbf{D}_0\mathbf{\Lambda}_0$.

There are four free parameters in the proposed technique, namely, the subspace dimension N', the number of representatives K_{dl}, the size of the dictionary L_{dl}, and the sparsity level α. For a target acceleration ρ, we first determine the number of representatives K_{vq} required by the VQ technique in order to achieve ρ, by using (14). This provides a lower bound for the number of representatives K_{dl} required by the proposed technique. We set $K_{dl} = c\,K_{vq}$, $c > 1$. Then, for a target sparsity α, we use (16) with equality in order to determine the dictionary size required to achieve ρ. Typical ranges for the parameter values are $c = 2, \ldots, 5$, $\alpha = 1, 2, 3$, and $N' = 4, \ldots, 8$.

V. Experimental Results

In this section, the performance of the proposed technique is evaluated and compared against the conventional VQ approach defined in (6) (and used in [27]). A two-fold performance evaluation is presented here. Specifically, in Experiment I, we evaluate the representation power of the rivals by means of the achieved quantization error, namely the error between $\mathbf{W}$ and its approximations defined in (6) and (7), respectively, for a range of target accelerations. This experiment is performed on the basis of individual-layer kernel approximations.

As expected, the less the per-layer quantization error, the less the anticipated accuracy loss of the accelerated model. Measuring this loss is the topic of Experiment II, where we perform full-range acceleration for selected modern DNNs, and compare the achieved accuracy of the accelerated models. In this case, the acceleration is limited to conv layers which are responsible for the vast majority of the DNN's computational complexity.

Our experiments are based on pre-trained versions of three state-of-the-art DNNs for image classification, namely, VGG-16 [7], SqueezeNet [11], and ResNet18 [15], using the training and validation datasets of ILSVRC2012 [35], for fine-tuning and accuracy evaluation purposes, respectively.

A. Experiment I. Quantization error in individual layers

In the first experiment, we evaluate the quantization error of the proposed technique for a range of target accelerations and compare the results against the conventional, k-means-based, VQ technique. To this end, we approximate the kernels of individual convolutional layers from the VGG16, SqueezeNet, and ResNet18 networks, using (6) and (7), and measure, in each case, the mean error between the original and approximated weights. For the target acceleration ratios, the number of representatives K_{vq} required by the VQ technique was calculated via (11). Then, by setting the number of DL representatives as $K_{dl} = c\,K_{vq}$, $c > 1$, the dictionary size L_{dl} was obtained so that (16) holds with equality. We then calculated the quantization error versus the achieved acceleration for various selections of the coefficient c and the sparsity level α. The subspace dimension was set to $N' = 8$, which is a typical value used in the relevant bibliography.

A representative instance of Experiment I involving three selected conv layers from the used models, namely, (a) conv4$-$1 of VGG16 (512 kernels of size $3\times3\times256$), (b) res4a-branch2b of ResNet18 (256 kernels of size $3 \times 3 \times 256$), and (c) fire8-expand3x3 of SqueezeNet (256 kernels of size $3\times3\times64$),

is shown in Fig. 2 (top row). As shown in Fig. 2, (and will become more apparent in the Experiment II), the proposed technique clearly outperforms its rival with respect to quantization error, achieving a better approximation of the original weights, for the same acceleration. Equivalently, this superior performance is translated into a significant acceleration gain for the same quantization error, as quantified by the respective plots on the bottom row of Fig. 2.

B. Experiment II. Accuracy loss

In this experiment, we apply the rival techniques to the three DNN models in a "full-model" acceleration scenario. It involves accelerating multiple (or all) convolutional layers of the original models and measuring the achieved classification accuracy of the accelerated networks.

It is stressed here that, although full-range acceleration depends heavily on the performance of the technique used for the acceleration of each layer, it also involves experimentation over the strategy used for accelerating the layers and the involved fine-tuning (re-training) of the accelerated model. Here, we follow a stage-wise acceleration approach (as proposed in [27]) with each stage involving accelerating (and fixing) one or more layers of the network, and subsequently, fine-tuning (i.e., re-training) the remaining original layers. The starting point for each stage is the accelerated and fine-tuned version of the previous stage. The process begins with the original network, and it is repeated until all target layers are accelerated. For fine-tuning and performance assessment we use the training and validation datasets, respectively, from ILSVRC2012. Since, in each stage, only a small fraction of the network is affected and in order to expedite the process, we divided the initial training dataset into smaller subsets and used these smaller sets for fine-tuning purposes.

a) Accelerating VGG16: VGG16 consists of 13 3×3 convolutional and 3 fully-connected (fc) layers and it is (by far) the most computationally intensive network of the three used in our experiments. Of the total 15.5×10^9 MACs/MULs required for inference, over 99% are consumed by the conv layers, meaning that the acceleration of the conv layers is practically equivalent to the acceleration of the entire network. The conv layers of VGG16 are organized in 5 groups of consecutive conv layers, which is why we accelerate VGG16 in 5 stages, with the i-th group being accelerated at stage i. The stage-by-stage results of our VGG16 acceleration experiment, using the procedure describe in the previous paragraph for three different acceleration ratios, namely $\rho = 20$, $\rho = 30$, and $\rho = 40$, are shown in the top row of Fig. 3.

b) Accelerating ResNet18: ResNet18 is based on the concept of residual learning and follows the architecture of other bigger ResNet variants (e.g. ResNet34, ResNet50, etc.). Its building block comprises of two consecutive 3×3 conv layers, with the block's output being summed to its input using a "bypass" connection (hence, the block is required to only learn the residual representation). ResNet18 consists of 8 such blocks (plus an input conv layer and an fc layer), that, similarly to the VGG16 case, are responsible for roughly 99% of the total 1.8×10^9 MACs/MULs required by the network. In our experiments with ResNet18 we accelerated its building blocks in a one-block-per-stage fashion leading to 8 total acceleration stages. The acceleration results using ratios $\rho = 10$, $\rho = 20$, and $\rho = 30$, are shown in the middle row of Fig. 3.

c) Accelerating SqueezeNet: SqueezeNet is a fully convolutional CNN that employs a special architecture managing to drastically reduce its size while still remaining within the state-of-the-art performance territory. Its building block is the "fire" module that consists of a "squeeze" 1×1 conv layer with the purpose of reducing the number of input channels, followed by 1×1 and 3×3 "expand" conv layers that are connected in parallel to the "squeezed" output. SqueezeNet consists of 8 such modules, connected in series. Since SqueezeNet constitutes an already "streamlined" network, in our acceleration experiments we followed a moderate acceleration strategy only targeting the 3×3 "expand" layers that are responsible for roughly 53% of the total 3.9×10^8 MACs/MULs required by the network. Acceleration was performed in a one-module-per-stage fashion for a total of 8 acceleration stages. The results for acceleration ratios $\rho = 10$, $\rho = 15$, and $\rho = 20$, corresponding to a total acceleration of the network by 91%, 98%, and 101%, respectively, are shown in the bottom row of Fig. 3.

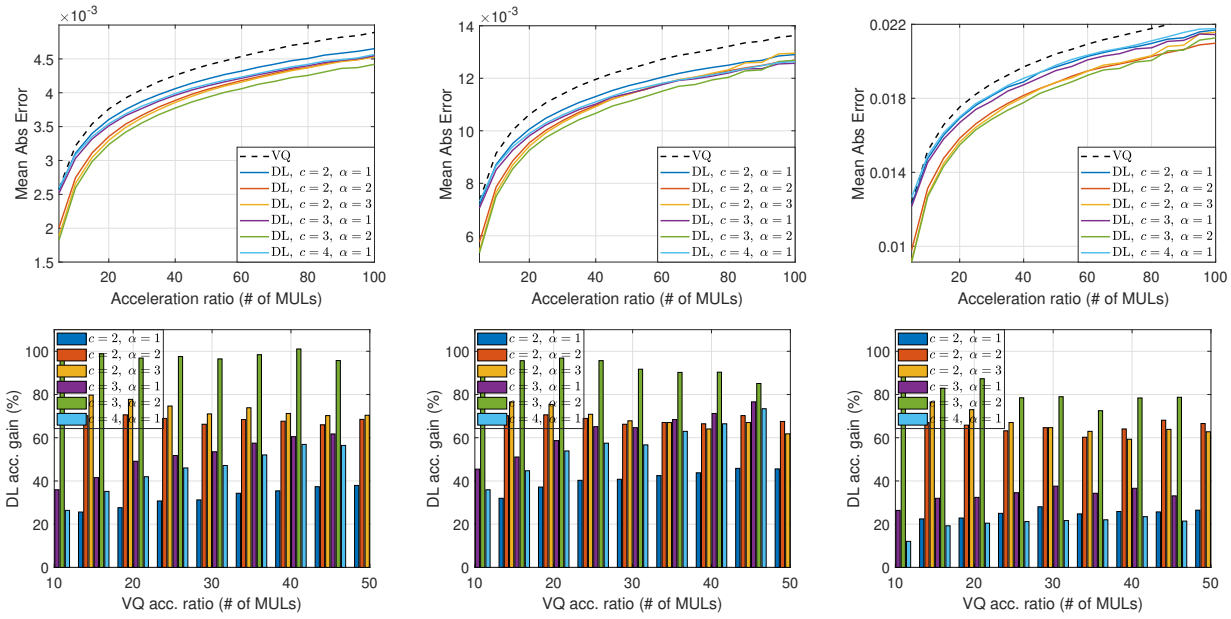
As a general comment regarding the result presented in Fig. 3, it could be stated that the "before fine-tuning" error values (shown in bars) at each stage, reveal the sensitivity of the network with respect to accelerating/approximating the kernels of the layers involved at that particular stage, but also, the performance of the technique used to achieve the acceleration. As such, it is evident by the shown plots, that the proposed technique achieves a universally superior performance compared to its rival.

On the other hand, the corresponding "after fine-tuning" error values reflect the capacity of the remaining (original) layers to "adapt" to the newly accelerated part. Here we see that, by offering a better starting point to the fine-tuning process, the proposed technique still manages to outperform its rival by a safe margin in all cases, although, as it is to be expected, fine-tuning compresses the difference between the compared techniques to a great extent.

In summary, both the comparative analysis of the results shown in Fig. 3, and also, the final Top5 error figures achieved by the accelerated networks, reveal a very promising performance by the proposed technique, whose application results in significantly accelerated CNNs, with limited loss of their classification power. It should be finally noted that the shown results could be further improved by following a more targeted acceleration strategy (e.g. experimentation over the acceleration sequence, the acceleration ratio per layer, using a more extensive fine-tuning process, etc.), which acts as further confirmation of our conclusion.

VI. Conclusions

A new clustering-based weight-approximation technique for the acceleration of DNNs was proposed in this paper. The technique exploits the particularities of the problem at hand



(a) (b) (c)

Fig. 2: Mean quantization error (top row) and acceleration gain (bottom row) of DL vs VQ techniques as a function of the acceleration ratio for layers: (a) layer conv$4-1$ of VGG16, (b) layer res4a-branch2b of ResNet18, and (c) layer fire8-expand3x3 of SqueezeNet. In all cases, the subspace dimension was $N' = 8$.

in order to increase the number of used centroids for the same target acceleration, as compared to the conventional k-means technique. This is achieved using a Dictionary Learning framework, by imposing a special structure to the centroids that reduces the overall computational complexity of the accelerated layer. The superior performance of the technique was validated via a number of experiments on three well-known state-of-the-art pre-trained DNN models.

ACKNOWLEDGEMENT

This paper has received funding from H2020 project CPSoSaware (No 873718) and the DEEP-EVIoT - Deep embedded vision using sparse convolutional neural networks project (MIS 5038640) implemented under the Action for the Strategic Development on the Research and Technological Sector, co-financed by national funds through the Operational programme of Western Greece 2014-2020 and European Union funds (European Regional Development Fund).

REFERENCES

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[2] M. M. Najafabadi, F. Villanustre, T. M. Khoshgoftaar, N. Seliya, R. Wald, and E. Muharemagic, "Deep learning applications and challenges in big data analytics," *Journal of Big Data*, vol. 2, no. 1, 2015.

[3] C. Cao, F. Liu, H. Tan, D. Song, W. Shu, W. Li, Y. Zhou, X. Bo, and Z. Xie, "Deep learning and its applications in biomedicine," *Genomics, proteomics & bioinformatics*, vol. 16, no. 1, pp. 17–32, 2018.

[4] J. Wang, Y. Ma, L. Zhang, R. X. Gao, and D. Wu, "Deep learning for smart manufacturing: Methods and applications," *Journal of Manufacturing Systems*, vol. 48, pp. 144–156, 2018.

[5] K. Suzuki, "Overview of deep learning in medical imaging," *Radiological physics and technology*, vol. 10, no. 3, pp. 257–273, 2017.

[6] T. Bolton and L. Zanna, "Applications of deep learning to ocean data inference and subgrid parameterization," *Journal of Advances in Modeling Earth Systems*, vol. 11, no. 1, pp. 376–399, 2019.

[7] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv:1409.1556*, 2014.

[8] W. G. Hatcher and W. Yu, "A survey of deep learning: platforms, applications and emerging research trends," *IEEE Access*, vol. 6, pp. 24 411–24 432, 2018.

[9] J. Tang, D. Sun, S. Liu, and J.-L. Gaudiot, "Enabling deep learning on iot devices," *Computer*, vol. 50, no. 10, pp. 92–96, 2017.

[10] L. Deng, G. Li, S. Han, L. Shi, and Y. Xie, "Model compression and hardware acceleration for neural networks: A comprehensive survey," *Proceedings of the IEEE*, vol. 108, no. 4, pp. 485–532, 2020.

[11] F. N. Iandola et al., "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size," *arXiv:1602.07360*, 2016.

[12] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv:1704.04861*, 2017.

[13] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," *arXiv:1905.11946*, 2019.

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[15] 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.

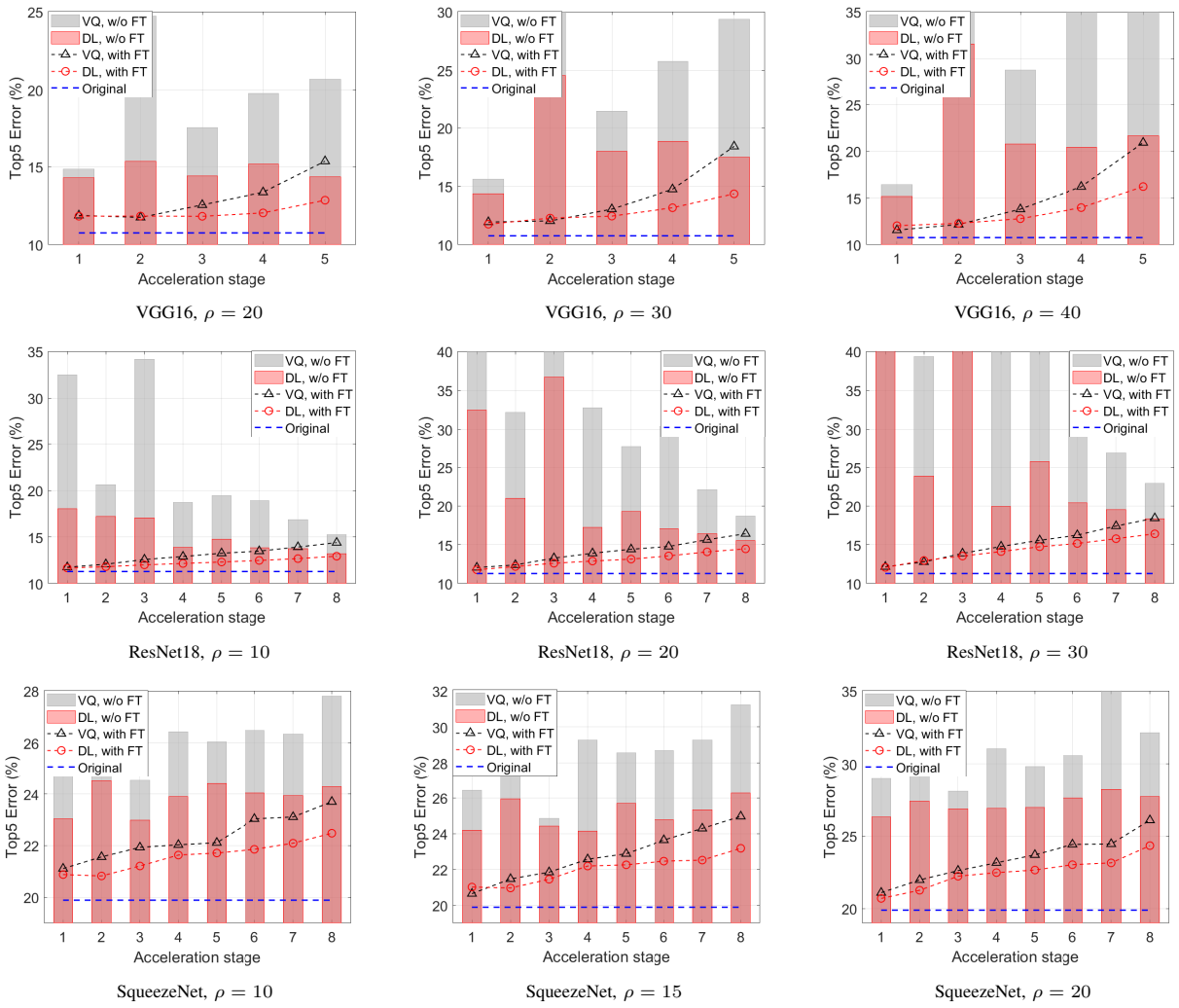
[16] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec. 2017.

[17] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "Model compression and acceleration for deep neural networks: The principles, progress, and challenges," *IEEE Sig. Proc. Mag.*, vol. 35, pp. 126–136, 2018.

[18] Q. Zhang, M. Zhang, T. Chen, Z. Sun, Y. Ma, and B. Yu, "Recent advances in convolutional neural network acceleration," *Neurocomputing*, vol. 323, pp. 37–51, 2019.

[19] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[20] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448.



VGG16, $\rho = 20$ VGG16, $\rho = 30$ VGG16, $\rho = 40$

ResNet18, $\rho = 10$ ResNet18, $\rho = 20$ ResNet18, $\rho = 30$

SqueezeNet, $\rho = 10$ SqueezeNet, $\rho = 15$ SqueezeNet, $\rho = 20$

Fig. 3: Full-model acceleration for the convolutional layers of VGG-16 (top), ResNet18 (middle), and SqueezeNet (bottom). The acceleration is performed sequentially in 5, 8, and 8 stages, respectively. Each stage involves accelerating one or more conv layers and measuring the validation performance of the network before and after fine-tuning. In each case, the starting point for stage i is the corresponding accelerated and fine-tuned network from stage $i-1$. The parameters values were $N' = 8$, $c = 3$, and $\alpha = 2$, while the acceleration ratio ρ was common for all stages, and it is reported on the bottom of the plots.

[21] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," *arXiv:1608.08710*, 2016.

[22] S. Lin, R. Ji, Y. Li, C. Deng, and X. Li, "Toward compact convnets via structure-sparsity regularized filter pruning," *IEEE transactions on neural networks and learning systems*, 2019.

[23] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *International Conference on Machine Learning*, 2015, pp. 1737–1746.

[24] Y. Gong, L. Liu, M. Yang, and L. Bourdev, "Compressing deep convolutional networks using vector quantization," *arXiv:1412.6115*, 2014.

[25] S. Bhattacharya and N. D. Lane, "Sparsification and separation of deep learning layers for constrained resource inference on wearables," in *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM*, 2016, pp. 176–189.

[26] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv:1510.00149*, 2015.

[27] J. Cheng, J. Wu, C. Leng, Y. Wang, and Q. Hu, "Quantized cnn: A unified approach to accelerate and compress convolutional networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 10, pp. 4730–4743, 2018.

[28] S. Son, S. Nah, and K. Mu Lee, "Clustering convolutional kernels to compress deep neural networks," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 216–232.

[29] J. Wu et al., "Deep k-means: Re-training and parameter sharing with harder cluster assignments for compressing deep convolutions," *arXiv:1806.09228*, 2018.

[30] Y. Hu et al., "Cluster regularized quantization for deep networks compression," in *2019 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2019, pp. 914–918.

[31] Z. Ben, X. Long, X. Zeng, and J. Liu, "Model compression for image classification based on low-rank sparse quantization," in *2019 IEEE 4th International Conference on Image, Vision and Computing (ICIVC)*. IEEE, 2019, pp. 412–415.

[32] P. Stock, A. Joulin, R. Gribonval, B. Graham, and H. Jégou, "And the bit goes down: Revisiting the quantization of neural networks," *arXiv:1907.05686*, 2019.

[33] E. Dupuis, D. Novo, I. O'Connor, and A. Bosio, "Sensitivity analysis and compression opportunities in dnns using weight sharing," in *2020 23rd International Symposium on DDECS*. IEEE, 2020, pp. 1–6.

[34] B. Dumitrescu and P. Irofti, *Dictionary Learning Algoithms and Applications*. Springer, 2018.

[35] O. Russakovsky et al., "Imagenet large scale visual recognition challenge," *IJCV*, vol. 115, no. 3, pp. 211–252, 2015.