

# A data-aware dictionary-learning based technique for the acceleration of deep convolutional networks

Erion-Vasilis Pikoulis<sup>1,2</sup>, Christos Mavrokefalidis<sup>1,2</sup>, Aris S. Lalos<sup>1</sup>

<sup>1</sup>Industrial Systems Institute, Athena Research Center, Patras Science Park, Greece

<sup>2</sup>Computer Engineering and Informatics Dept., University of Patras, Greece

Emails: {pikoulis,maurokef}@ceid.upatras.gr, lalos@isi.gr

**Abstract**—The deployment of high performing deep learning models on platforms of limited resources is currently an active area of research. Among the main directions followed so far, pre-trained neural networks are accelerated and compressed by appropriately modifying their structure and / or parameters. Capitalizing on a recently proposed codebook of a special structure that can be utilized in the frame of the so-called weight sharing methods, this paper describes a “data-driven” technique for designing such a codebook. The performance of the technique, in terms of the observed representation error and classification accuracy versus the achieved acceleration ratio, is demonstrated by considering the VGG16 and the ResNet18 models, pre-trained on the ILSVRC2012 dataset.

**Index Terms**—Model Compression and Acceleration, Weight Sharing, Dictionary Learning

## I. INTRODUCTION

Nowadays, there is an increasing interest in deploying Deep Neural Network (DNN) models on (mobile) platforms of limited resources [1]. As high-end DNN models consist of tens, or even hundreds of millions of parameters [2], the design of smaller/faster, still high-performing, DNN models has been recently considered a key objective. To address this, two main lines of research can be identified [2]. In the first line, new compact, smaller DNN models are designed or searched for in a design space for the applications at hand (e.g., SqueezeNet [3], MobileNets [4], and EfficientNet [5]). In the second line, existing pre-trained, highly performing DNN models (e.g., Visual Geometry Group (VGG) [6], Residual Net (ResNet) [7], etc.) are transformed into new smaller models by utilizing Model Compression and Acceleration (MCA) techniques. The latter line of research enables high-performing, pre-trained DNN models to be utilized also as back-bone modules in DNN models designed for different (but similar in nature) applications. For example, the convolutional layers of VGG (trained initially for image classification), constitute the core modules of the Fast R-CNN object detector [8].

The MCA-related literature has been increasing in recent years and there are numerous surveys that provide a comprehensive overview of the area ([2], [9], [10], [11]). To understand, roughly, the relevant literature, some works propose the pruning (removal) of unimportant parameters which are not considered during the inference (e.g., filters [12], [13]). Other works utilize weight sharing by reducing the bit-width of the involved parameters or by increasing their common

representations (e.g., in a scalar, [14], or vector/product [15], [16], [17], [18], [19] quantization fashion). These common representations construct a so-called codebook with a twofold aim. First, the network can be compressed as less parameters need to be stored. Second, it can be accelerated as the involved calculations are performed using only the codebook and not the original (considerably more) parameters. Finally, other works employ tensor / matrix decompositions on the involved quantities into factors by utilizing, for instance, low-rankness [20].

In this paper, we capitalize on a recently proposed codebook of a special structure that can be utilized in the frame of the so-called weight sharing methods [19]. In [19], the problem was treated from a “data-agnostic” perspective, namely by directly approximating the original kernel parameters. Here, we revisit the problem of designing this codebook by introducing a “data-aware” technique, i.e., a technique that utilizes part of the available dataset in order to accelerate / compress the targeted DNN model. For this, the proposed technique minimizes the representation error at the output of a single convolutional layer, instead of the quantization error considered in [19]. The performance of the new technique, in terms of representation error and classification accuracy, is assessed when applied on the VGG16 and ResNet18 DNN models, pre-trained on the ILSVRC2012 dataset, and compared with [16] and [19]. The results demonstrate an improvement on the representation error at the output of the convolutional layers which ultimately results in a smaller accuracy loss for the accelerated models.

## II. PRELIMINARIES

Let us first define the linear operation performed at a convolutional layer between input volume  $\mathbf{X} \in \mathbb{R}^{m \times m \times N}$  and the  $k$ -th kernel volume  $\mathbf{W}_k \in \mathbb{R}^{p \times p \times N}$ ,  $k = 1, \dots, M$ . To simplify notation and without loss of generality, we assume square-shaped input feature maps and kernel filters, as well as a stride of 1. We also adopt a contiguous numbering format for the spatial coordinates  $(n, l)$ 's of the involved tensors, following any numbering convention (e.g., in a row- or column- major order). Using this convention, we can write the convolution operation in terms of dot-products between

input and kernel vectors, as follows:

$$\mathbf{U}_k[i] = \sum_{j=1}^{p^2} \mathbf{x}_{i,j}^T \mathbf{w}_{k,j}, \quad i = 1, \dots, m^2 \quad (1)$$

where  $\mathbf{U}_k[i]$  denotes the  $i$ -th element (residing at the  $i$ -th spatial coordinate) of the  $k$ -th convolutional output,  $\mathbf{x}_{i,j} \in \mathbb{R}^{N \times 1}$  is the  $j$ -th (depth-wise) input vector around the  $i$ -th spatial position of the input, while  $\mathbf{w}_{k,j} \in \mathbb{R}^{N \times 1}$  denotes the  $j$ -th (depth-wise) vector of the  $k$ -th kernel, respectively, using the same numbering convention. It is stressed here that the involved vectors are defined along the  $N$  channels of the input volume and the kernel volumes.

By adopting the product quantization framework [15], the  $N$ -Dimensional ( $\mathbf{D}$ ) vector space is partitioned into  $S$ ,  $N'$ -D subspaces with  $N' = N/S$ , so that the  $s$ -th subspace spans dimensions  $[(s-1)N' + 1, \dots, sN']$ ,  $s = 1, \dots, S$ . Then, by partitioning vectors  $\mathbf{x}_{i,j}$  and  $\mathbf{w}_{k,j}$  defined in Eq. (1) as  $\mathbf{x}_{i,j} = [(\mathbf{x}_{i,j}^1)^T, \dots, (\mathbf{x}_{i,j}^S)^T]^T$ ,  $\mathbf{w}_{k,j} = [(\mathbf{w}_{k,j}^1)^T, \dots, (\mathbf{w}_{k,j}^S)^T]^T$ , respectively, where each of the sub-vectors lies in an  $N'$ -D space, (1) can be rewritten as

$$\mathbf{U}_k[i] = \sum_{s=1}^S \sum_{j=1}^{p^2} (\mathbf{x}_{i,j}^s)^T \mathbf{w}_{k,j}^s, \quad (2)$$

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

For subspace  $s$ , the goal of product quantization is to perform vector quantization to the  $Mp^2$  kernel sub-vectors lying in this subspace, i.e., cluster them into  $K_s \ll Mp^2$  clusters and represent each sub-vector with the centroid (or representative) of the cluster it belongs to. This way, the original dot-products, between the input and the  $Mp^2$  kernel sub-vectors, are approximated by the ones between the input and the  $K_s$  representatives.

### III. ADOPTED CODEBOOK STRUCTURE

In this section, we describe the adopted codebook structure (as proposed in [19]), which can be viewed as a special case of the general Dictionary Learning (DL) problem [21], and compare it against the conventional approach (referred to as Vector Quantization (VQ) in the following).

Specifically, the conventional VQ, and proposed DL - based kernel approximations can be defined as follows:

$$\text{VQ: } \mathbf{W} \approx \mathbf{C}\mathbf{\Gamma}, \quad \text{DL: } \mathbf{W} \approx \mathbf{D}\mathbf{\Lambda}\mathbf{\Gamma} \quad (3)$$

where the columns of  $\mathbf{W} \in \mathbb{R}^{N' \times p^2 M}$  and  $\mathbf{\Gamma} \in \mathbb{R}^{K_{vq} \times p^2 M}$  contain the sub-vectors of all kernel volumes (of a particular subspace) and assignment vectors, respectively. Matrix  $\mathbf{C} \in \mathbb{R}^{N' \times K_{vq}}$  denotes the representatives (or cluster centroids) in the VQ approximation whereas  $\mathbf{D} \in \mathbb{R}^{N' \times L_{dl}}$  and  $\mathbf{\Lambda} \in \mathbb{R}^{L_{dl} \times K_{dl}}$  denote the dictionary and the matrix of sparse coefficients, respectively, for the DL approximation. 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 the codebook  $\mathbf{C}$  in the VQ case and one column of the codebook  $\mathbf{D}\mathbf{\Lambda}$  in the DL case.

Thus, in the conventional case, the  $Mp^2$  original sub-vectors are approximated by  $K_{vq} \ll p^2 M$  representatives, using the codebook  $\mathbf{C}$ , while in the DL case, they are approximated via  $K_{dl}$  representatives contained in the codebook  $\mathbf{D}\mathbf{\Lambda}$ , which, in turn, are obtained as linear combinations of at most  $\alpha$  atoms from a dictionary of size  $L_{dl}$ , with  $L_{dl} < K_{dl} \ll p^2 M$ .

It should be noted that, 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 “dense” operations. This enables the proposed technique to achieve significantly better approximation of the desired convolutional output, for the same target accelerations, as it is going to be demonstrate by our experimental results.

#### A. Computational complexity

Let us now examine the computational complexity of the original convolutional layer and its two approximate versions based on the VQ and DL approaches, respectively, and define the acceleration gains induced by the approximations. Specifically, by arranging the input sub-vectors of a particular subspace in the columns of a matrix  $\mathbf{X} \in \mathbb{R}^{N' \times m^2}$ , the dot-products entailed by the original convolution operation can be obtained as the following matrix product:

$$\mathbf{Y} = \mathbf{X}^T \mathbf{W}, \quad (4)$$

where  $\mathbf{W}$  contains the kernel sub-vectors (as defined in (3)). In the VQ case, the original  $\mathbf{W}$  is approximated via the codebook  $\mathbf{C}$ , meaning that the approximate  $\mathbf{Y}$  is obtained as:

$$\mathbf{Y} \approx (\mathbf{X}^T \mathbf{C}) \mathbf{\Gamma}, \quad (5)$$

namely, it involves calculating the dot-products between input and codewords, and then substituting the results appropriately, as indicated by the columns of  $\mathbf{\Gamma}$ . Accordingly, for the DL-based approximation scheme, we can write:

$$\mathbf{Y} \approx ((\mathbf{X}^T \mathbf{D}) \mathbf{\Lambda}) \mathbf{\Gamma}, \quad (6)$$

meaning that in this case, the approximate dot-products are obtained via a two-stage operation, namely, it involves calculating the dot-products between the input and the dictionary atoms, and subsequently, obtaining their linear combinations according to the columns of  $\mathbf{\Lambda}$ .

Following (4), (5), (6), its not difficult to show that the computational complexities in terms of Multiply and Accumulate (MAC) operations, for the original and the approximate versions of a convolutional layer, using VQ and DL respectively, can be expressed as follows ([16],[19]):

$$\mathcal{T}_o = m^2 p^2 M N. \quad (7)$$

$$\mathcal{T}_{vq} = m^2 N K_{vq}. \quad (8)$$

$$\mathcal{T}_{dl} = m^2 (N L_{dl} + \alpha S K_{dl}). \quad (9)$$

The achieved acceleration ratio is defined as the ratio of original over accelerated computational complexities and for the two rival parameter-sharing approaches examined in this paper, takes the following form:

$$\rho_{vq} \equiv \frac{\mathcal{T}_o}{\mathcal{T}_{vq}} = \frac{p^2 M}{K_{vq}} \quad (10)$$

$$\rho_{dl} \equiv \frac{\mathcal{T}_o}{\mathcal{T}_{dl}} = \frac{p^2 M}{L_{dl} + \frac{\alpha}{N'} K_{dl}}. \quad (11)$$

Finally, of particular interest is also the relative complexity between the proposed DL-based and the VQ approach, which acts also as a guideline for selecting the free parameters of the proposed technique. To this end, it is not difficult to, that the VQ- and DL-based approximate layers entail the same computational complexity (i.e., yield the same acceleration ratio) when the respective parameters satisfy the following equality:

$$L_{dl} = K_{vq} \left(1 - \frac{\rho c}{N'}\right), \quad (12)$$

where  $c > 1$  is a size coefficient between the DL-based and the VQ-based codebooks, i.e.  $K_{dl} = c K_{vq}$  holds.

#### IV. THE PROPOSED DATA-AWARE SOLUTION

Let us collect the  $j$ -th vector from all kernels in the columns of a matrix  $\mathbf{W}_j \in \mathbb{R}^{N \times M}$  as  $\mathbf{W}_j = [\mathbf{w}_{1,j} \cdots \mathbf{w}_{M,j}]$ ,  $j = 1, \dots, p^2$ , and let us define:

$$\mathbf{u}(i) = \sum_{j=1}^{p^2} \mathbf{x}_{i,j}^T \mathbf{W}_j, \quad (13)$$

where  $\mathbf{u}(i) = [\mathbf{U}_1[i], \dots, \mathbf{U}_M[i]]$  contains the outputs from all kernels, at the  $i$ -th spatial location. By sampling  $I$  spatial locations (from the same or different output volumes), we obtain  $I$  input sub-volumes of dimension  $p \times p \times N$ . Let us, also, collect the  $j$ -th vector from all input sub-volumes in the columns of a matrix  $\mathbf{X}_j \in \mathbb{R}^{N \times I}$ , i.e.,  $\mathbf{X}_j = [\mathbf{x}_{1,j} \cdots \mathbf{x}_{I,j}]$ ,  $j = 1, \dots, p^2$ . Furthermore, by defining the  $I \times M$  matrix  $\mathbf{U} = [\mathbf{u}^T(1) \cdots \mathbf{u}^T(I)]^T$ , we have:

$$\mathbf{U} = \sum_{s=1} \sum_{j=1}^{p^2} (\mathbf{X}_j^s)^T \mathbf{W}_j^s, \quad (14)$$

where  $\mathbf{W}^s$  and  $\mathbf{X}^s$  contain the rows lying in the  $s$ -th subspace of  $\mathbf{W}$  and  $\mathbf{X}$ , respectively.

Under the DL approximation defined in (3), we are seeking the factorization of each  $\mathbf{W}_j^s$  in (14) as  $\mathbf{W}_j^s \approx \mathbf{D}^s \mathbf{\Lambda}^s \mathbf{\Gamma}_j^s$ . In a data-aware fashion, this can be achieved via the following minimization problem.

$$\begin{aligned} \min_{\{\mathbf{D}^s, \mathbf{\Lambda}^s, \mathbf{\Gamma}_j^s\}} & \left\| \mathbf{U} - \sum_{s=1}^S \sum_{j=1}^{p^2} (\mathbf{X}_j^s)^T \mathbf{D}^s \mathbf{\Lambda}^s \mathbf{\Gamma}_j^s \right\|_F^2 \\ \text{s.t.} & \quad \|\mathbf{d}_i^s\|_2^2 = 1, i = 1, \dots, L_{dl}, \forall s \\ & \quad \|\boldsymbol{\lambda}_i^s\|_0 \leq \alpha, i = 1, \dots, K_{dl}, \forall s \\ & \quad \|\boldsymbol{\gamma}_{j,i}^s\|_0 = 1, \mathbf{1}^T \boldsymbol{\gamma}_{j,i}^s = 1, i = 1 \dots Mp^2, \forall s \end{aligned} \quad (15)$$

where  $\|\cdot\|_F$ ,  $\|\cdot\|_2$  and  $\|\cdot\|_0$  denote the Frobenius,  $l_2$  and  $l_0$  norms, respectively. Under the constraints stated in (15),

the dictionary atoms in  $\{\mathbf{D}^s\}$  have unit norm, the columns of the sparse representations in  $\{\mathbf{\Lambda}^s\}$  have at most  $\alpha$  non-zero elements and the assignment vectors in (the columns of)  $\{\mathbf{\Gamma}_j^s\}$  have a single non-zero element equal to one.

The minimization of (15) is carried out in a coordinate-descend fashion, by optimizing one set of parameters at a time, while keeping all others fixed. In the following subsections, we briefly describe this strategy, as well as the proposed solutions to the sub-problems entailed by it.

##### A. Sparse coding

To minimize (15), first, the optimization for the  $\mathbf{\Lambda}^s$ 's in an alternating fashion among  $s$ , is carried out by keeping all other unknowns fixed. Focusing on a particular sub-space  $t$ , the cost function of (15) can be written as

$$\left\| \mathbf{Y}^t - \sum_{j=1}^{p^2} (\mathbf{X}_j^t)^T \mathbf{D}^t \mathbf{\Lambda}^t \mathbf{\Gamma}_j^t \right\|_F^2 = \left\| \mathbf{y}^t - \sum_{i=1}^{K_{dl}} \mathbf{G}_i^t \boldsymbol{\lambda}_i^t \right\|_2^2, \quad (16)$$

where  $\mathbf{Y}^t$  gathers all terms of (15) apart from the ones related to the subspace  $t$ ,  $\mathbf{y}^t = \text{vec}(\mathbf{Y}^t) \in \mathbb{R}^{MI \times 1}$ ,  $\mathbf{G}_i^t \in \mathbb{R}^{MI \times L_{dl}}$  and  $\boldsymbol{\lambda}_i \in \mathbb{R}^{L_{dl} \times 1}$ . The summation terms in (16) are connected via the identity  $\text{vec}(\mathbf{AXB}) = (\mathbf{B}^T \otimes \mathbf{A}) \text{vec}(\mathbf{X})$ , where  $\otimes$  denotes the Kronecker product. Using this identity,  $\mathbf{G}_i^t = \sum_{j=1}^{p^2} (\tilde{\boldsymbol{\gamma}}_{j,i}^t)^T \otimes (\mathbf{X}_j^t)^T \mathbf{D}^t$ , where  $\tilde{\boldsymbol{\gamma}}_{j,i}^t \in \mathbb{R}^{1 \times K_{dl}}$  denotes the  $i$ -th row of  $\mathbf{\Gamma}_j^t$ . The cost-function in (16) along with the relevant constraint can be solved using the standard OMP algorithm by alternating among the  $\boldsymbol{\lambda}_i^t$ 's,  $\forall i$ .

##### B. Dictionary update

In this sub-problem, the optimization is carried out similarly for each  $\mathbf{D}^t$ . The cost-function, in this case, can be written as

$$\left\| \mathbf{y}^t - \left( \sum_{j=1}^{p^2} (\mathbf{\Gamma}_j^t)^T (\mathbf{\Lambda}^t)^T \otimes (\mathbf{X}_j^t)^T \right) \mathbf{d}^t \right\|_2^2 = \|\mathbf{y} - \mathbf{H} \mathbf{d}\|_2^2,$$

where  $\mathbf{y} \in \mathbb{R}^{MI \times 1}$ ,  $\mathbf{H} \in \mathbb{R}^{MI \times N' L_{dl}}$  and  $\mathbf{d} \in \mathbb{R}^{N' L_{dl} \times 1}$ . Additionally,  $\mathbf{d} = [\mathbf{d}_1^T, \dots, \mathbf{d}_{L_{dl}}^T]^T$ , where  $\mathbf{d}_i \in \mathbb{R}^{N' \times 1}$  denotes the  $i$ -th dictionary atom. It is noted that the superscript  $t$  was dropped for simplicity.

Thus, the dictionary-update step of the proposed technique, takes the form of the following Quadratically Constrained Quadratic Program (QCQP):

$$\min_{\mathbf{d}} \|\mathbf{y} - \mathbf{H} \mathbf{d}\|_2^2 \quad \text{s.t.} \quad \|\mathbf{d}_i\|_2^2 = 1, i = 1, \dots, L_{dl} \quad (17)$$

Though (17) constitutes a non-convex, NP-hard problem in general, it can be tackled via the technique of SemiDefinite Relaxation (SDR) [22], [23], which translates a QCQP with an unknown vector  $\mathbf{x}$  into a convex trace minimization problem, where the unknown is a matrix  $\mathbf{X}$ . Such SemiDefinite Programs (SDP) can be solved using standard software packages like CVX [24]. Finally, it is noted that, depending on the input size, the target acceleration and the selection of parameters,  $\mathbf{H}$  can become quite large. Fortunately, in our case,  $\mathbf{H}$  is, also, extremely sparse, which can be exploited for the speed-up of the involved computations.

DNN	VGG16						ResNet18					
Layer	conv1-2		conv2-1		conv3-3		res2b-br2b		res3a-br2a		res3b-br2b	
Ratio ( $\times$ )	10	20	10	20	10	20	10	20	10	20	10	20
VQ-AG [16]	1.54	3.88	6.15	10.1	4.76	8.38	8.68	20.7	9.77	18.1	7.14	14.9
DL-AG [19]	0.75	1.75	3.85	7.33	2.91	5.43	4.71	12.9	4.44	12.8	3.72	10.3
VQ-AW [16]	0.14	0.31	0.61	1.21	0.71	1.17	2.51	6.07	1.74	4.13	2.57	5.62
DL-AW (prop.)	0.10	0.21	0.49	0.85	0.56	0.95	1.98	4.84	1.33	3.44	1.67	4.48

TABLE I: Relative representation error (%). The data-AGnostic (AG) and data-AWare (AW) cases for VQ and DL, are shown.

### C. Assignment update

Let us first write the cost function in (16) as follows:

$$\mathcal{E} = \sum_{k=1}^M \left\| \mathbf{y}_k^t - \sum_{j=1}^{p^2} (\mathbf{X}_j^t)^T \mathbf{D}^t \mathbf{\Lambda}^t \gamma_{j,k}^t \right\|_2^2, \quad (18)$$

where  $\mathbf{y}_k^t \in \mathbb{R}^{I \times 1}$  denotes the  $k$ -th column of  $\mathbf{Y}^t$ , while  $\gamma_{j,k}^t \in \mathbb{R}^{K_{dl} \times 1}$  is the  $k$ -th column of  $\mathbf{\Gamma}_j^t$ , which holds the assignment vector for the  $j$ -th sub-vector lying in the  $t$ -th subspace of the  $k$ -th kernel. Observing (18), the assignment vectors can be updated for each kernel separately, leading to minimization problems of the following form:

$$\min_{\{\zeta_j\}} \left\| \mathbf{y}_k^t - \sum_{j=1}^{p^2} \mathbf{F}_j^t \zeta_j \right\|_2^2, \quad (19)$$

where  $\mathbf{F}_j^t = (\mathbf{X}_j^t)^T \mathbf{D}^t \mathbf{\Lambda}^t \in \mathbb{R}^{I \times K_{dl}}$ , under the constraint that each  $\zeta_j$  has a single non-zero element equal to one. Taking into account this constraint, the goal of the assignment-update problem is to select one column  $\mathbf{f}_{j,i}^t$  from each  $\mathbf{F}_j^t$ ,  $j = 1, \dots, K_{dl}$  (thereby determining the position of the non-zero unity element in each  $\gamma_{j,k}^t$ ), so that the sum of all  $\mathbf{f}_{j,i}^t$ 's best approximates  $\mathbf{y}_k^t$ . This is a problem of combinatorial complexity, which can be tackled via a greedy (OMP-like) algorithm, similar to the one used in the sparse-coding step.

## V. EXPERIMENTAL RESULTS

In this section, the performance of the proposed technique is evaluated and compared against the data-agnostic version presented in [19], as well as both versions of the conventional VQ approach used in [16]. Our experiments are based on pre-trained versions of two state-of-the-art DNNs for image classification, namely, VGG-16 [6] and ResNet18 [7], using the training and validation datasets of ILSVRC2012 [25], for obtaining the original conv-layer responses, as well as for fine-tuning and accuracy evaluation purposes.

### A. Experiment I. Representation error

In the first experiment, we evaluate the performance of the rivals with respect to the achieved representation error for individual conv-layers, namely the error between the original layer output and the approximate output obtained using the kernel approximations defined in (3), for a range of target accelerations. To this end, we obtain the original layer output  $\mathbf{U}$ , and the corresponding input sub-volumes  $\mathbf{X}$  (as defined in (14)), by random sampling of the layer's output and input,

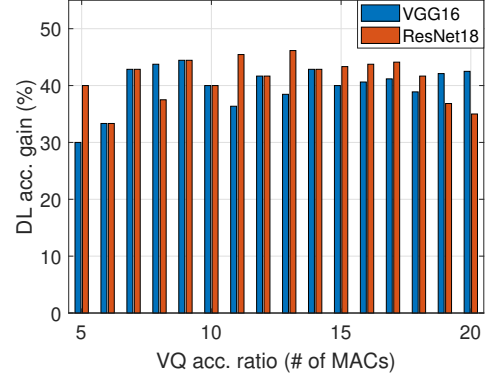


Fig. 1: Average acceleration gain of DL-AW over VQ-AW for the conv-layers of VGG16 and ResNet18. Here, we calculate the acceleration ratios achieved by the rivals, for the same representation error, using the results of Experiment I.

respectively, for  $I = 1000$  images from the ILSVRC2012 training dataset.

For the DL-based approach, the (proposed) data-aware approximation was obtained via the solution of (15), while the data-agnostic variant was obtained as in [19]. The latter variant was also used as an initial solution  $\mathbf{D}_0, \mathbf{\Lambda}_0, \mathbf{\Gamma}_0$ , to the proposed data-aware technique. Moreover, the four free parameters of the technique, namely, the subspace dimension  $N'$ , the number of representatives  $K_{dl}$ , the size of the dictionary  $L_{dl}$ , and the sparsity level  $\alpha$  are obtained according to the procedure outlined in [19]. Specifically in this experiment, the subspace dimension was set to  $N' = 8$ , which is a typical value used in the relevant bibliography, while the parameter values  $c = 2, \alpha = 3$  were selected after some experimentation. Finally, the procedures presented in [16] were followed for the VQ variants.

An instance of the results obtained in Experiment I for three selected conv-layers from the used models, namely, conv1-2 ( $3 \times 3 \times 64 \times 64$ ), conv2-1 ( $3 \times 3 \times 128 \times 128$ ), and conv3-3 ( $3 \times 3 \times 256 \times 256$ ), of VGG16, as well as res2b-branch2b ( $3 \times 3 \times 64 \times 64$ ), res3a-branch2a ( $3 \times 3 \times 128 \times 128$ ), and res3b-branch2b ( $3 \times 3 \times 128 \times 128$ ) from ResNet18, are summarized in Table I. As it is clear from the presented results (and was apparent throughout this experiment), although both data-aware variants are significantly better than their data-agnostic counterparts, the proposed technique clearly outperforms all its rivals with respect to representation error, achieving a better approximation of the original layer's response, for the same acceleration. Equivalently, this superior performance is

translated into a significant acceleration gain (of around 40%), as quantified by the plots shown in Fig. 1.

### B. Experiment II. Accuracy loss

In this experiment, we apply the proposed technique for the “full-model” acceleration of ResNet18. To this end, we followed a stage-wise acceleration approach (as described in [16], [19]) 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. For fine-tuning and performance assessment, we use the training and validation datasets, respectively, from ILSVRC2012. In order to expedite the process, we divided the initial training dataset into smaller subsets (of 100 images per class) for fine-tuning purposes. By following this procedure, we accelerated ResNet18 by  $10\times$  and  $20\times$  with an increase of top5 classification error by only 0.9% and 2.1%, respectively. These preliminary results are inline with the outcome of Experiment I and act as a further confirmation of the quality of the proposed technique.

## VI. CONCLUSIONS

A new data-aware, weight-approximation technique was proposed in this paper. The original problem was treated in a coordinate descent fashion and solutions for the entailed sparse coding, dictionary update, and assignment sub-problems, were presented. The preliminary results on two state-of-the-art pre-trained DNN models, reveal the superior performance of the technique against its data-agnostic variant as well as both VQ counterparts.

## ACKNOWLEDGMENT

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] 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.
- [2] 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.
- [3] F. N. Iandola et al., “Squeezenet: Alexnet-level accuracy with 50x fewer parameters and 0.5 mb model size,” *arXiv:1602.07360*, 2016.
- [4] 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.
- [5] M. Tan and Q. V. Le, “Efficientnet: Rethinking model scaling for convolutional neural networks,” *arXiv:1905.11946*, 2019.
- [6] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv:1409.1556*, 2014.
- [7] 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.
- [8] R. Girshick, “Fast r-cnn,” in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448.

- [9] 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.
- [10] 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.
- [11] 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.
- [12] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, “Pruning filters for efficient convnets,” *arXiv:1608.08710*, 2016.
- [13] 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.
- [14] 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.
- [15] Y. Gong, L. Liu, M. Yang, and L. Bourdev, “Compressing deep convolutional networks using vector quantization,” *arXiv:1412.6115*, 2014.
- [16] 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.
- [17] 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.
- [18] 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.
- [19] E.-V. Piskoulis, C. Mavrokefalidis, and A. S. Lalos, “A new clustering-based technique for the acceleration of deep convolutional networks,” in *Proceedings of the IEEE International Conference on Machine Learning and Applications*, 2020.
- [20] 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.
- [21] B. Dumitrescu and P. Irofti, *Dictionary Learning Algorithms and Applications*. Springer, 2018.
- [22] Y. S. Nesterov, “Semidefinite relaxation and nonconvex quadratic optimization,” *Optim. Methods Softw.*, vol. 9, no. 1-3, pp. 141–160, 1998.
- [23] Z. Luo, W. Ma, A. M. So, Y. Ye, and S. Zhang, “Semidefinite relaxation of quadratic optimization problems,” *IEEE Signal Processing Magazine*, vol. 27, no. 3, pp. 20–34, 2010.
- [24] M. Grant and S. Boyd, “CVX: Matlab software for disciplined convex programming, version 2.1,” <http://cvxr.com/cvx>, Mar. 2014.
- [25] O. Russakovsky et al., “Imagenet large scale visual recognition challenge,” *IJCV*, vol. 115, no. 3, pp. 211–252, 2015.