Acceleration Approaches for Big Data Analysis

ACCELERATION APPROACHES FOR BIG DATA ANALYSIS

Anton Muravev[*], *Dat Thanh Tran*[*], *Alexandros Iosifidis*[†], *Serkan Kiranyaz*[‡] *and Moncef Gabbouj*[*]

[*]Department of Signal Processing, Tampere University of Technology, Finland
[†]Department of Engineering, Electrical & Computer Engineering, Aarhus University, Denmark
[‡]Department of Electrical Engineering, Qatar University, Qatar
e-mail: {anton.muravev,dat.tranthanh,moncef.gabbouj}@tut.fi;
alexandros.iosifidis@eng.au.dk; mkiranyaz@qu.edu.qa

ABSTRACT

The massive size of data that needs to be processed by Machine Learning models nowadays sets new challenges related to their computational complexity and memory footprint. These challenges span all processing steps involved in the application of the related models, i.e., from the fundamental processing steps needed to evaluate distances of vectors, to the optimization of large-scale systems, e.g. for non-linear regression using kernels, or the speed up of deep learning models formed by billions of parameters. In order to address these challenges, new approximate solutions have been recently proposed based on matrix/tensor decompositions, randomization and quantization strategies. This paper provides a comprehensive review of the related methodologies and discusses their connections.

Index Terms— Approximate Nearest Neighbor Search, Vector Quantization, Hashing, Approximate kernel-based learning, Low-rank Approximation, Neural Network Acceleration

1. INTRODUCTION

Given a data collection, e.g. a set of N images $\mathcal{I}_i$, $i = 1, \ldots, N$, two of the fundamental processing steps of a Machine Learning model are those of distance calculation and regression. In the first case, a data pre-processing step is initially applied in order to define a vector representation for each image in a D-dimensional feature space, i.e. $\mathbf{x}_i \in \mathbb{R}^D$, $i = 1, \ldots, N$. Given the vectors representing two images $\mathcal{I}_j$ and $\mathcal{I}_k$, their (squared) Euclidean distance is defined as:

$$d_{\mathcal{I}}(\mathcal{I}_j, \mathcal{I}_k) = d(\mathbf{x}_j, \mathbf{x}_k) = \|\mathbf{x}_j - \mathbf{x}_k\|_2^2. \tag{1}$$

In the latter case, each image $\mathcal{I}_i$ is associated with a target $\mathbf{t}_i \in \mathbb{R}^C$, e.g. in the case of an image classification problem formed by C image categories. Thus, a regression model $\mathcal{M}$ should be optimized so that $\mathcal{M}(\mathcal{I}_j) \rightarrow \mathbf{t}_j$. Regression models $\mathcal{M}$ can take the form of (possibly deep) neural networks or kernel methods operating on vector representations, i.e. $\mathcal{M}_\kappa(\mathbf{x}_j) \rightarrow \mathbf{t}_j$.

2. APPROXIMATE DISTANCE COMPUTATION

Pairwise distance calculation is used in a variety of applications, including nearest neighbor search, kernel methods and others. However, as the data dimensionality and dataset size increase, direct distance computations become prohibitively expensive, warranting the development of many approximate solutions. They can be divided in two groups, their theoretical foundation being hashing and quantization, respectively.

2.1. Hashing-based approaches

Hashing-based approaches map the data to binary codes, replacing the original metric with the Hamming distance. This allows extremely efficient storage and fast calculations at the expense of severely reduced discrimination, as d-dimensional binary code permits at most $d + 1$ different distances.

Locality Sensitive Hashing (LSH) is a wide family of methods that construct hash functions to maximize the code collision probability between similar items and minimize it between dissimilar items [1]. Thus, LSH does not attempt to preserve the original space, but limits it to the similarity-dissimilarity relationship defined using a user-provided distance threshold. To improve the representation, several hash functions are typically used to generate the codes, each with an individual hash table. LSH families are distance-specific; for instance, the Euclidean distance can be associated with the following hash function [2]:

$$h_l(\mathbf{x}_i) = \left\lfloor \frac{\mathbf{w}_l^T \mathbf{x}_i + b_l}{r_l} \right\rfloor, \tag{2}$$

where $\mathbf{w}_l \in \mathbb{R}^D$, $l = 1, \ldots, d$ is a Gaussian random vector, b is a uniformly random real number from the interval $[0,\, r]$, and r is a positive real parameter defining the distance threshold for similarity. Using the above, each vector $\mathbf{x}_i \in \mathbb{R}^D$ is mapped to the corresponding hash-vector $\mathbf{h}_i \in \mathbb{R}^d$.The multitude of hash functions for different distances have been derived, including cosine distance [3], Jaccard coefficient [4], χ^2 distance [5], rank similarity [6].

Yet better hash code representations can be obtained by learning data-specific mappings. Spectral hashing [7] formulates extra requirements for the produced binary codes to keep them short: individual bits should be balanced (values 0 and 1 being about equally likely) and uncorrelated (to minimize redundancy). Then, assuming $\mathbf{h}_i,\ i = 1, \dots, N,\ \mathbf{h}_i \in \{-1,\ 1\}^M$ is a set of M-bit codes for N datapoints and $\mathbf{W} \in \mathbb{R}^{N\times N}$ is the data similarity matrix correlating with Euclidean distance, the following optimization problem is put forward:

$$\min_{\mathbf{h}} \sum_{ij} W_{ij} \|\mathbf{h}_i - \mathbf{h}_j\|_2^2, \tag{3}$$

$$s.t. \sum_i \mathbf{h}_i = 0 \quad \text{and} \quad \frac{1}{N}\sum_i \mathbf{h}_i\mathbf{h}_i^T = I$$

This problem is equivalent to balanced graph partitioning and is thus NP-hard, but can be solved via spectral relaxation under the assumption that the data is uniformly distributed. Follow-up approaches use similar objective functions [8, 9].

2.2. Quantization-based approaches

Quantization-based approaches learn data representations in $\mathbb{R}^D$, aiming to preserve pairwise Euclidean distances between the samples. These representations are drawn from a learnt set of data-specific codebooks. The codebook distances can then be pre-computed and stored in a look-up table for faster computation. Such solutions typically offer better distance preservation at the cost of extra computations and storage space.

Product Quantization (PQ) [10] divides the D-dimensional space into M subspaces of $\frac{D}{M}$ dimensions each, then applies clustering on each subspace independently to obtain K centroids in each subspace. Cartesian product of these M codebooks produces K^M centroids. Due to subspace orthogonality the Euclidean distance to any quantized vector $\tilde{\mathbf{x}} = c_1 \times ... \times c_M$ can be calculated by M table lookups:

$$\|\mathbf{x}_j - \tilde{\mathbf{x}}\|_2^2 = \sum_{i=1}^{M} \|\mathbf{x}_j - c_i\|_2^2 - (M-1)\|\mathbf{x}_j\|_2^2 \tag{4}$$

Optimized Product Quantization (OPQ) [11] additionally introduces the adaptive rotation by multiplying the data with an orthogonal matrix before the subspace decomposition. The matrix is updated between the clustering iterations. As rotation does not affect centroid assignments, no other changes to the algorithm are necessary. Further advances include multiple rotation matrices [12]. PQ assignment codes, if considered as binary strings, can form notably efficient hash codes for fast Hamming distance estimation [13].

Additive quantization (AQ) [14, 15] follows a similar approach, but utilizes D-dimensional codebooks. Vector representations are formed not by concatenation, but by addition: $\tilde{\mathbf{x}} = \mathbf{c}_1 + \dots + \mathbf{c}_M$, allowing for much lower distance distortion. Codebook learning still follows the iterative approach akin to Lloyd's algorithm, alternating between solving codebook equations and performing the data assignment to centroids. The latter step, unlike in PQ, is NP-hard, resulting in a number of optimization heuristics being proposed [14]. As the codebooks are no longer orthogonal, the distance estimation receives an additional dot product term:

$$\|\mathbf{x}_j - \tilde{\mathbf{x}}\|_2^2 = \sum_{i=1}^{M} \|\mathbf{x}_j - \mathbf{c}_i\|_2^2 - (M-1)\|\mathbf{x}_j\|_2^2 + \sum_{i\neq k}^{M} \mathbf{c}_i^T\mathbf{c}_k \tag{5}$$

The presence of the dot product term means that the number of lookups goes from M to $(M + M^2)$. Composite quantization (CQ) [16] aims to alleviate this drawback by imposing additional constraints on additive codebooks:

$$\forall\tilde{\mathbf{x}} = \{\mathbf{c}_1 + ... + \mathbf{c}_M\} : \sum_{i=1}^{M}\sum_{j=1}^{M} \mathbf{c}_i^T\mathbf{c}_j = \varepsilon,\ i \neq j \tag{6}$$

Then the inter-centroid dot product is replaced by a scalar ε, which is learned during the quantizer training. As a result, in CQ the distance computation only requires M lookups.

3. APPROXIMATE KERNEL-BASED REGRESSION

Kernel methods use a *kernel function* $\kappa(\cdot,\cdot)$ to express dot-products of D-dimensional data, i.e., $\kappa(\mathbf{x}_i, \mathbf{x}_j) \triangleq \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j)$, where $\phi(\cdot)$ is a nonlinear function, such that

$$\mathbf{x}_i \in \mathbb{R}^D \xrightarrow{\phi(\cdot)} \phi(\mathbf{x}_i) \in \mathcal{F}. \tag{7}$$

$\mathcal{F}$ is the so-called kernel space, and its properties are defined by the kernel function $\kappa(\cdot,\cdot)$. After applying the nonlinear mapping in (7), a linear regression problem $\mathbf{W}_\phi^T\phi(\mathbf{x}_i) \rightarrow \mathbf{t}_i$ is solved, where $\mathbf{W}_\phi \in \mathbb{R}^{|\mathcal{F}|\times C}$ is the regression matrix in $\mathcal{F}$. Further, by expressing $\mathbf{W}_\phi$ as a linear combination of the training samples the regression problem takes the form:

$$\mathcal{J} = \sum_{i=1}^{N} \|\mathbf{A}^T\mathbf{\Phi}^T\phi(\mathbf{x}_i) - \mathbf{t}_i\|_2^2 = \|\mathbf{A}^T\mathbf{K} - \mathbf{T}\|_F^2, \tag{8}$$

where $\mathbf{T} = [\mathbf{t}_1, \dots, \mathbf{t}_N] \in \mathbb{R}^{C\times N}$. $\mathbf{K} \in \mathbb{R}^{N\times N}$ is the (positive semi-definite) *kernel matrix* with elements $[\mathbf{K}]_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$. The solution of (8) is given by:

$$\mathbf{A} = (\mathbf{K} + c\mathbf{I})^{-1}\,\mathbf{T}^T, \tag{9}$$

where $c > 0$ is a regularization parameter used to avoid singularities in the inversion. For large-scale problems, storage and inversion of $\mathbf{K}$ are intractable, since the time and space complexities are in the order of $O(N^3)$ and $O(N^2)$, respectively. Approximate kernel-based regression methods can be divided in two major groups, and they are based on low-rank matrix decomposition or randomization approaches.

3.1. Low-rank approximation of $\mathbf{K}$

Under the assumption that the $\mathbf{K}$ is a low-rank matrix, it can be decomposed as $\mathbf{K} \approx \tilde{\mathbf{K}} = \mathbf{C}\mathbf{C}^T$, where $\mathbf{C} \in \mathbb{R}^{N \times n}$, $n \ll N$, and (9) can be transformed to:

$$\mathbf{A} \simeq \left(\tilde{\mathbf{K}} + \delta\mathbf{I}\right)^{-1} \mathbf{T}^T = \frac{1}{\delta}\left[\mathbf{I} - \mathbf{C}\left(\delta\mathbf{I} + \mathbf{C}^T\mathbf{C}\right)^{-1}\mathbf{C}^T\right]\mathbf{T}^T, \tag{10}$$

requiring the inversion of a $n \times n$ matrix, highly reducing the time (and memory) complexity of the regression solution.

A randomized low-rank matrix approximation approach uses a random (Gaussian) matrix $\mathbf{\Omega} \in \mathbb{R}^{N \times n}$ to map $\mathbf{Y} = \mathbf{K}\mathbf{\Omega}$ [17]. $\mathbf{Y}$ is then decomposed as $\mathbf{Y} = \mathbf{Q}\mathbf{R}$, where $\mathbf{Q} \in \mathbb{R}^{N \times n}$ is an orthogonal matrix used to calculate $\mathbf{B} = \mathbf{Q}^T\mathbf{K}\mathbf{Q} \in \mathbb{R}^{n \times n}$. An eigenanalysis step is then applied for $\mathbf{B} = \hat{\mathbf{U}}\mathbf{D}\hat{\mathbf{U}}^T$. The matrix $\mathbf{C}$ is finally obtained by $\mathbf{C} = \mathbf{Q}\hat{\mathbf{U}}\mathbf{D}^{\frac{1}{2}}$. While this approach leads to reduced computational cost for the subsequent kernel-based regression, it requires the calculation of the entire kernel matrix $\mathbf{K}$.

In order to obtain the matrix $\mathbf{C}$, the eigen-value decomposition of $\mathbf{K} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$ can be used, where $\mathbf{\Lambda} \in \mathbb{R}^{N \times N}$ a (diagonal) matrix containing the eigenvalues and by $\mathbf{U} \in \mathbb{R}^{N \times N}$ matrix formed by corresponding eigenvectors. Then, by discarding the smallest eigenvalues and the corresponding eigenvectors, we obtain $\mathbf{C} = \mathbf{U}_n\mathbf{\Lambda}_n^{\frac{1}{2}}$. However, the above-described process requires the eigenanalysis of $\mathbf{K}$. In order to avoid the eigenvalue decomposition of $\mathbf{K}$, the Nyström method can be used [18, 19].

Let us denote by $\mathbf{K}_{Nn} \in \mathbb{R}^{N \times n}$ a sub-matrix of $\mathbf{K}$ obtained by selecting n of its columns and by $\mathbf{K}_{nn}$ the sub-matrix of $\mathbf{K}$ formed by the same columns and corresponding rows. Several selection processes have been proposed, i.e. uniform sampling [18], probabilistic sampling [20], column-norm sampling [21], adaptive sampling [22], clustering-based sampling [23] and deterministic sampling [24]. The matrix containing the n leading eigenvectors of $\mathbf{K}$ can be approximated by the Nyström extension by $\mathbf{U}_n \approx \mathbf{K}_{Nn}\mathbf{U}_{(n)}\mathbf{\Lambda}_{(n)}^{-1}$, where $\mathbf{U}_{(n)}$ and $\mathbf{\Lambda}_{(n)}$ are the eigenvectors and eigenvalues of $\mathbf{K}_{nn}$. An n-rank approximation of $\mathbf{K}$ is, then, given by exploiting:

$$\mathbf{K} \approx \tilde{\mathbf{K}} = \mathbf{K}_{Nn}\mathbf{K}_{nn}^{-1}\mathbf{K}_{Nn}^T = \mathbf{G}\mathbf{G}^T, \tag{11}$$

where $\mathbf{G} = \mathbf{K}_{Nn}\mathbf{K}_{nn}^{-\frac{1}{2}}$. Extensions of Nyström method include the Density Weight Nyström method [25], the Nyström method by spectral shifting [26], the ensemble Nyström method [27] and the CUR-based Nyström method [28].

Another low-rank decomposition-based method [21] uses a scaled version of $\mathbf{K}_{Nn}$ (using a factor $\frac{1}{\sqrt{np_k}}$, where where p_k corresponds to the k-th column) in order to form a matrix $\mathbf{G} \in \mathbb{R}^{N \times n}$. The eigenvectors and eigenvalues of $\mathbf{K}$ are approximated by the eigenvectors of the matrix $\tilde{\mathbf{G}} = \mathbf{G}^T\mathbf{G}$. Specifically, the eigenanalysis of $\tilde{\mathbf{G}}$ is applied to get the left singular vectors and the singular values of $\mathbf{G}$, i.e. $\tilde{\mathbf{G}} = \mathbf{G}^T\mathbf{G} = \mathbf{V}\mathbf{\Sigma}\mathbf{V}^T$. Then, the left singular vectors of $\mathbf{G}$ (and, thus, the eigenvectors of $\tilde{\mathbf{K}}$) are obtained by $\mathbf{U} = \mathbf{G}\mathbf{V}\mathbf{\Sigma}^{\dagger}$, where $\cdot^{\dagger}$ denotes the Moore-Penrose pseudo-inverse.

Finally, kernel matrix approximation methods based on explicit feature maps have been proposed [29, 30]. In this case the explicit map is selected so that the dot product of the derived data representations will be approximately equal to the dot product of the data representations in the kernel space for shift-invariant kernel function.

3.2. Reduced or randomized kernel space definition

Methods following a reduced kernel space definition exploit express $\mathbf{W}_\phi$ as a linear combination of a set of reference vectors $\mathbf{z}_k \in \mathbb{R}^D$, $k = 1, \dots, K$ in $\mathcal{F}$, i.e. $\mathbf{W}_\phi = \tilde{\mathbf{\Phi}}\tilde{\mathbf{A}}$, where $\tilde{\mathbf{\Phi}} = [\phi(\mathbf{z}_1), \dots, \phi(\mathbf{z}_K)] \in \mathbb{R}^{|\mathcal{F}| \times K}$ and $\tilde{\mathbf{A}} \in \mathbb{R}^{K \times C}$.

In this case, the regression problem takes the form:

$$\mathcal{J} = \sum_{i=1}^{N} \|\tilde{\mathbf{A}}^T\tilde{\mathbf{\Phi}}^T\phi(\mathbf{x}_i) - \mathbf{t}_i\|_2^2 = \|\tilde{\mathbf{A}}^T\tilde{\mathbf{K}} - \mathbf{T}\|_F^2, \tag{12}$$

where $\tilde{\mathbf{K}} \in \mathbb{R}^{K \times N}$ is a reduced kernel matrix having elements equal to $[\tilde{\mathbf{K}}]_{ij} = \kappa(\mathbf{z}_i, \mathbf{x}_j)$, $i = 1, \dots, K$, $j = 1, \dots, N$. $\tilde{\mathbf{A}}$ is then given by:

$$\tilde{\mathbf{A}} = \left(\tilde{\mathbf{K}}\tilde{\mathbf{K}}^T\right)^{-1}\tilde{\mathbf{K}}\mathbf{T}^T, \tag{13}$$

The above solution requires the inversion of a $K \times K$ matrix, highly reducing the time (and memory) complexity of the regression solution, when $K << N$.

Reference vectors $\mathbf{z}_k$, $k = 1, \dots, K$ can be determined as: randomly selected training vectors [31, 32]; prototypes obtained by applying a clustering technique, e.g. K-means, on the training data [33, 34]; random vectors drawn by a multi-dimensional (uniform) distribution $\mathbb{R}^D$ [35, 36]. As has been shown in [37], the use of random prototype vectors leads to an approximation of the original kernel-based regression [38].

4. NEURAL NETWORK COMPRESSION

Modern network architectures comprise of several layers with millions of parameters, requiring billions of floating point operations, which hinders the deployment of such networks to mobile devices with limited memory and computational resources. Therefore, several works have been dedicated to reduce memory and computation requirement during inference while minimizing the performance losses. Existing approaches can be classified into four categories: pruning, quantization and hashing, low-rank approximation and knowledge distillation.

Early works [39] used second-derivative information to prune network parameters during the training stage, leading to both speedup and generalization performance during test time. Later approaches rely on l_1 or l_2 norm of the weights to select pruning candidates. In [40], similar neurons within a layer are detected and pruned, leading to substantial compression rate for the fully-connected layers. Choosing l_1 norm, [41] proposed to compress a pre-trained CNN by removing a set of convolution filters of a layer, and thereby the corresponding channels in the subsequent layer. Similarly, different group-sparsity constraints have been employed during training to achieve a compact structure such as the works in [42, 43].To learn a compact network structure, [44] proposed a three-step training procedure in which essential connectivities are learned and redundant ones are removed.

Expressing network parameters using low bidwidth representation is a straightforward way to compress a pre-trained model. Focusing on the fully-connected layers, [45] explored the vector quantization schemes to this end. PQ was also proposed in [46] to compress both convolution and fully-connected layers. Deep networks training with low precision weights leads to significant reduction in memory usage and computation time [47]. Network parameters discretization was proposed in [47] leading to a 6-bit weight representation. Instead of trying to preserve extremal values, the quantization scheme in [48] was designed to achieve balanced distribution of quantized values, producing 4-bit quantized model. Binarized networks have also been proposed in [49], however achieving much lower accuracy than the baseline networks using 32-bit floating-point numbers.

Low-rank approximation has also been a popular approach to reduce memory footprint and computation in deep networks. It has been shown in [50] that a small subset of bases can be used to predict the remaining weights through kernel ridge regression. Canonical Polyadic (CP) decomposition-based weights approximation has been proposed in [51]. [52] proposed a low-rank expansion of convolution layer by performing two convolution steps that operate on the horizontal and vertical direction. Using the same expansion, [53] showed that the approximation problem has a closed-form, global solution. In addition, the low-rank expansion can be trained from scratch to yield even better accuracy using Batch Normalization. Multilinear filters projection was proposed in [54], while Tucker decomposition with a rank selection scheme was proposed in [55] to approximate convolution layers.

While directly training a small network from scratch might not achieve good performance, knowledge transfer from a large trained network to a smaller one was shown to be effective [56, 57]. This is known as Knowledge Distilation or teacher-student paradigm. It should be noted that the aforementioned approaches can be combined to further reduce the memory and computation requirement. For example, in [58] a compression pipeline called "Deep Compression" which performs pruning, quantization and Huffman coding of the weights.

5. REFERENCES

[1] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," in *Int. J. VLDB*, 1999, vol. 99, pp. 518–529.

[2] M. Datar, N. Immorlica, P. Indyk, and V.S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *Symp. Comput. Geom.*, 2004.

[3] A. Andoni and P. Indyk, "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions," in *IEEE Symp. Found. Comp. Sc.*, 2006.

[4] W.L. Zhao, H. Jégou, and G. Gravier, "Sim-min-hash: An efficient matching technique for linking large image collections," in *ACM Int. Conf. Mult.*, 2013.

[5] D. Gorisse, M. Cordand, and F. Precioso, "Locality-sensitive hashing for chi2 distance," *IEEE Trans. Patt. Anal. Mach. Intell.e*, vol. 34, no. 2, pp. 402–409, 2012.

[6] J. Yagnik, D. Strelow, D.A. Ross, and R. Lin, "The power of comparative reasoning," in *ICCV*, 2011.

[7] Y. Weiss, A. Torralba, and R. Fergus, "Spectral hashing," in *NIPS*. 2009.

[8] J. He, W. Liu, and S.F. Chang, "Scalable similarity search with optimized kernel hashing," in *ACM SIGKDD*, 2010.

[9] E.C. Ozan, S. Kiranyaz, and M. Gabbouj, "M-pca binary embedding for approximate nearest neighbor search," in *IEEE BigDataSE*, 2015.

[10] H. Jégou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search.," *IEEE Trans. Patt. Anal. Mach. Intell.*, vol. 33, no. 1, pp. 117–28, 2011.

[11] M. Norouzi and D.J. Fleet, "Cartesian K-Means," *Computer Vision and Pattern Recognition*, 2013.

[12] E.C. Ozan, S. Kiranyaz, and M. Gabbouj, "K-subspaces quantization for approximate nearest neighbor search," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 7, pp. 1722–1733, 2016.

[13] M. Douze, H. Jégou, and F. Perronnin, "Polysemous codes," *ECCV*, 2016.

[14] A. Babenko and V. Lempitsky, "Additive quantization for extreme vector compression," *CVPR*, 2014.

[15] A. Muravev, E.C. Ozan, A. Iosifidis, and M. Gabbouj, "Pyramid encoding for fast additive quantization," in *EUSIPCO*, 2017.

[16] T. Zhang, C. Du, and J. Wang, "Composite Quantization for Approximate Nearest Neighbor Search," *ICML*, 2014.

[17] N. Halko, P.G. Martinsson, and J.A. Tropp, "Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions," *SIAM Reviews*, vol. 53, pp. 217–288, 2011.

[18] C. Williams and M. Seeger, "Using the nyström method to speed up kernel machines," in *NIPS*, 2001.

[19] A. Iosifidis and M. Gabbouj, "Nyström-based approximate kernel subspace learning," *Patt. Rec.*, vol. 57, pp. 190–197, 2016.

[20] P. Drineas and M.W. Mahoney, "On the nyström method for approximating a gram matrix for improved kernel-based learning," *JMLR*, vol. 6, pp. 2153–2175, 2005.

[21] P. Drineas, R. Kannan, and M.W. Mahoney, "Fast mont carlo algorithms for matrices ii: Computing a low-rank approximation to a matrix," *SIAM J. Comput.*, vol. 36, pp. 158–183, 2011.

[22] S. Vempala A. Deshpante, L. Rademacher and G. Wang, "Matrix approximation and projective clustering via volume sampling," in *ACM-SIAM Symp. Discr. Alg.*, 2006.

[23] K. Zhang, I.W. Tsang, and J.T. Kwok, "Improved nyström low-rank approximation and error analysis," in *ICML*, 2008.

[24] M.A. Belabbas and P.J. Wolf, "Spectral methods in machine learning and new strategies for very large datasets," in *Proc. Nat. Ac. Sciences*, 2006.

[25] K. Zhang and J.T. Kwok, "Density-weighted nyström methods for computing large kernel eigensystems," *Neural Comput.*, vol. 21, pp. 121–146, 2009.

[26] Z. Zhang, "The matrix ridge approximation: algorithms and applications," *Mach. Learn.*, vol. 97, pp. 227–258, 2014.

[27] M. Mohri S. Kumar and A. Talwalkar, "Ensemble nyström method," in *NIPS*, 2009.

[28] S. Wang and Z. Zhang, "Improving cur matrix decomposition and nyström approximation via adaptive sampling," *JMLR*, vol. 14, pp. 2729–2769, 2013.

[29] A. Rahimi and B. Recht, "Random features for large-scale kernel machines," in *NIPS*, 2007.

[30] A. Vedaldi and A. Zisserman, "Ieee trans. pattern analysis and machine intelligence," *SIAM Reviews*, vol. 34, no. 3, pp. 480–492, 2012.

[31] Y. Lee and S. Huang, "Reduced support vector machines: A statistical theory," *IEEE Trans. Neur. Net.*, vol. 18, pp. 1–13, 2007.

[32] A. Iosifidis, A. Tefas, and I. Pitas, "Large-scale nonlinear facial image classification based on approximate kernel extreme learning machine," in *IEEE ICIP*, 2015.

[33] K. Zhang, L. Lan, J. Kwok, S. Vucetic, and B. Parvin, "Scaling up graph-based semisupervised learning via prototype vector machines," *IEEE Trans. Neur. Net. Learn. Syst.*, vol. 26, no. 3, pp. 444–457, 2015.

[34] A Iosifidis and M Gabbouj, "Scaling up class-specific kernel discriminant analysis for large-scale face verification," *IEEE Trans. Inf. Forens. Sec.*, vol. 11, no. 11, pp. 2453–2465, 2016.

[35] G.B. Huang, Zhou H, X. Ding, and R. Zhang, "Extreme learning machine for regression and multiclass classification," *IEEE Trans. Syst., Man, Cyb., Part B: Cyb.*, vol. 42, no. 2, pp. 513–529, 2012.

[36] A. Iosifidis, A. Tefas, and I. Pitas, "Graph embedded extreme learning machine," *IEEE Trans. Cyb.*, vol. 46, no. 10, pp. 311–324, 2016.

[37] A. Iosifidis, A. Tefas, and I. Pitas, "On the kernel extreme learning machine classifier," *PRL*, vol. 54, pp. 11–17, 2015.

[38] C.K.I. Williams, "Computation with infinite neural networks," *Neur. Comput.*, vol. 10, pp. 1203–1216, 1998.

[39] Y. LeCun, J.S. Denker, and S.A. Solla, "Optimal brain damage," in *NIPS*, 1990.

[40] S. Srinivas and R.V. Babu, "Data-free parameter pruning for deep neural networks," *arXiv:1507.06149*, 2015.

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

[42] S. Anwar, K. Hwang, and W. Sung, "Structured pruning of deep convolutional neural networks," *ACM J. Em. Techn. Comp. Syst.*, vol. 13, no. 3, pp. 32, 2017.

[43] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *NIPS*, 2016.

[44] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *NIPS*, 2015.

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

[46] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized convolutional neural networks for mobile devices," in *CVPR*, 2016.

[47] 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:1609.07061*, 2016.

[48] S.C. Zhou, Y.Z. Wang, H. Wen, Q.Y. He, and Y.H. Zou, "Balanced quantization: An effective and efficient approach to quantized neural networks," *J. Comp. Sc. Techn.*, vol. 32, no. 4, pp. 667–682, 2017.

[49] M. Courbariaux, Y. Bengio, and J.P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *NIPS*, 2015.

[50] M. Denil, B. Shakibi, L. Dinh, N. De Freitas, et al., "Predicting parameters in deep learning," in *NIPS*, 2013.

[51] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky, "Speeding-up convolutional neural networks using fine-tuned CP-decomposition," *arXiv:1412.6553*, 2014.

[52] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," *arXiv:1405.3866*, 2014.

[53] C. Tai, T. Xiao, et al., "Convolutional neural networks with low-rank regularization," *arXiv:1511.06067*, 2015.

[54] D.T. Tran, A. Iosifidis, and M. Gabbouj, "Improving Efficiency in Convolutional Neural Network with Multilinear Filters," *arXiv:1709.09902*, 2017.

[55] Y.D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin, "Compression of deep convolutional neural networks for fast and low power mobile applications," *arXiv:1511.06530*, 2015.

[56] C. Bucilua, R. Caruana, and A. Niculescu-Mizil, "Model compression," in *ACM SIGKDD International Conference on Knowledge Discovery and Data mining*, 2006.

[57] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv:1503.02531*, 2015.

[58] 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.