Linearized Kernel Dictionary Learning

Alona Golts and Michael Elad, *IEEE Fellow*

***Abstract*—In this paper we present a new approach of incorporating kernels into dictionary learning. The kernel K-SVD algorithm (KKSVD), which has been introduced recently, shows an improvement in classification performance, with relation to its linear counterpart K-SVD. However, this algorithm requires the storage and handling of a very large kernel matrix, which leads to high computational cost, while also limiting its use to setups with small number of training examples. We address these problems by combining two ideas: first we approximate the kernel matrix using a cleverly sampled subset of its columns using the Nyström method; secondly, as we wish to avoid using this matrix altogether, we decompose it by SVD to form new "virtual samples", on which any linear dictionary learning can be employed. Our method, termed "Linearized Kernel Dictionary Learning" (LKDL) can be seamlessly applied as a pre-processing stage on top of any efficient off-the-shelf dictionary learning scheme, effectively "kernelizing" it. We demonstrate the effectiveness of our method on several tasks of both supervised and unsupervised classification and show the efficiency of the proposed scheme, its easy integration and performance boosting properties.**

***Index Terms*—Dictionary Learning, Supervised Dictionary Learning, Kernel Dictionary Learning, Kernels, KSVD.**

I. Introduction

THE field of sparse representations has witnessed great success in a variety of applications in signal and image processing. The basic operation in sparse representations is called "sparse coding", which involves the reconstruction of the signals of interest using a sparse set of building blocks, referred to as "atoms". The atoms are gathered in a structure called the "dictionary", which can be manually crafted to contain mathematical functions that are proven successful in representing signals and images, such as wavelets [1], curvelets [2] and contourlets [3]. Alternatively, it can be learned adaptively from input examples, a task referred to as "dictionary learning" (DL). The latter approach has provided state-of-the-art results in classic image processing applications, such as denoising [4], inpainting [5], demosaicing [6], compression [7], [8] and more. Popular algorithms for DL are the MOD [9] and the K-SVD [10], which generalizes K-means clustering and learns an overcomplete dictionary that best sparsifies the input data.

Although successful in signal processing applications, the K-SVD algorithm "as-is" may not be suited for machine learning tasks such as classification or regression, as its primary goal is to achieve the best reconstruction of the input data, ignoring any discriminative information such as labels or annotations. Many suggestions have been made to extend DL to deal with labeled data. The SRC method by Wright *et al.* [11] achieved impressive results in face recognition by sparse coding each test sample over a dictionary containing the train samples from all classes, and choosing the class that presents the best reconstruction error. In [12], [13] Mairal *et al.* added a discriminative term to the DL model, and later incorporated the learning of the classifier parameters within the optimization of DL. The work reported in [14] by Zhang *et al.* was the first to incorporate the learning of the classifier parameters within the framework of the K-SVD algorithm. A similar extension has been made in [15], [16] by Jiang *et al.*, where in addition to the classifier parameters, another discriminative term for the sparse codes was added and optimized using the regular K-SVD. In [17] Yang *et al.* created an optimization function which forces both the learned dictionary and the resulting sparse coefficients to be discriminative. In [18], Cai *et al.* extended the work in [17] and proposed assigning different weights for each pair of sparse representation vectors, which are optimized during the DL process. These algorithms and others that relate to them have been shown to be competitive with the best available learning algorithms, leading often times to state-of-the-art results.

In machine learning, kernels have provided a straightforward way of extending a given algorithm to deal with nonlinearities. Prominent examples of such algorithms include kernel-SVM [19], kernel-PCA (KPCA) [20] and Kernel Fisher Discriminant (KFD) [21]. Suppose the original data can be mapped to a higher dimensional "feature space", where tasks such as classification and regression are far easier. Under the proper conditions, the "kernel trick" allows one to train a learning algorithm in the higher-dimensional feature space, without using explicitly the exact mapping. This can be done by posing the entire algorithm in terms of inner products between the input signals, and later replacing these inner-products with kernels. One fundamental problem when using the kernel trick is that one is forced to access only the inner products of signals in feature space, instead of the signals themselves. A direct consequence of this is the need to store and manipulate a large kernel matrix $\mathbf{K}$ of dimension $N \times N$ (N being the size of the training set), which contains the modified inner products of all pairs of input examples.

In recent years, kernels have also been incorporated in the field of sparse representations, both in tasks of sparse coding [22]–[28] and DL [24], [29]–[34]. The starting point of this paper is the kernel DL method termed "Kernel K-SVD" (KKSVD) by Nguyen *et al.* The novelty in [29] is in the ability to fully pose the entire DL scheme in terms of kernels, using a unique-structured dictionary which is a multiplication of two parts. The first, a constant matrix called the "base-dictionary", contains all of the mapped signals in feature space, and the second, called the "coefficient-dictionary", which is actually updated during the learning process. The KKSVD suffers from the same issues arising when applying the kernel trick in general. Specifically, in large-scale datasets, where the number of input samples is of the order of thousands and beyond, the

This research is funded under European Unions Seventh Framework Program, ERC Grant agreement no. 320649.

KKSVD quickly becomes impractical, both due to runtime and in the required storage space.

While kernel sparse representation is becoming more common, the existing algorithms are still challenging as they suffer from problems mentioned above. The arena of linear DL on the other hand, has a vast selection of existing tools that are implemented efficiently, enabling learning a dictionary quite rapidly in various settings and even if the number of examples to train on goes to the Millions. Indeed, in such extreme cases, online learning becomes appealing [35], [36].

As we show hereafter, our proposed method, "Linearized Kernel Dictionary Learning" (LKDL), enjoys the benefits of both worlds. LKDL is composed of two stages: kernel matrix approximation, followed by a linearization of the training process by the creation of "virtual samples" [37]. In the first stage, we apply the Nyström method to approximate the kernel matrix $\mathbf{K}$, using a sub-sampled set of its columns. We explore and compare several such sub-sampling strategies, including core-sets, k-means, uniform, column-norm and diagonal sampling. Rather than using $\mathbf{K}$ (or its approximation), we proceed with the assumption that it originates from a linear kernel, i.e. $\mathbf{K} = \mathbf{F}^T\mathbf{F}$, and thus, instead of referring to $\mathbf{K}$, we calculate the virtual samples $\mathbf{F}$, using standard eigen-decomposition. After obtaining these virtual training and test sets, we apply an efficient off-the-shelf version of linear DL and continue with a standard classification scheme. This process essentially "linearizes" the kernel matrix and combines the nonlinear kernel information within that of the virtual samples.

We evaluate the performance of LKDL in four aspects: (1) first, we assure that the added nonlinearity in the form of the virtual datasets indeed improves classification results (with relation to linear DL) and performs comparably well as the exact kernelization performed in KKSVD; (2) we demonstrate the differences in runtime between the two methods; (3) we compare our method with other recent kernelized features [38], [39], and (4) we show the easiness of integration of LKDL with ***any*** existing DL algorithm, including supervised DL.

This paper is organized as follows: section II provides background to classical reconstructive DL with emphasis on the K-SVD and two methods of supervised DL, all of which are used later in the experimental part as the linear foundations over which our scheme is employed. Section III discusses Nguyen's KKSVD algorithm for kernel DL and discusses its complexity. Section IV presents the details of our proposed algorithm, LKDL, for kernel DL. This section also builds a wider picture of this field, by surveying the relevant literature of incorporating kernels into sparse coding and DL. Section V shows results corroborating the effectiveness of our method, and finally, section VI concludes this paper and proposes future research directions.

II. Linear Dictionary Learning

This section provides background on classic reconstructive DL, as well as two examples of discriminative, supervised DL. The purpose of this section is to recall several key algorithms, the MOD and K-SVD, the FDDL, and the LC-KSVD, which we will kernelize in later sections.

A. Background

In sparse representations, given an input signal $\mathbf{x} \in \mathbb{R}^p$ and a "dictionary" $\mathbf{D} \in \mathbb{R}^{p\times m}$, one wishes to find a "sparse representation" vector, $\boldsymbol{\gamma} \in \mathbb{R}^m$ such that $\mathbf{x} \approx \tilde{\mathbf{x}} = \mathbf{D}\boldsymbol{\gamma}$. The dictionary $\mathbf{D} = [\mathbf{d}_1, \dots, \mathbf{d}_m]$ consists of "atoms" which faithfully represent the set of signals $\mathbf{x} \in \mathcal{X}$. The task of finding a signal's sparse representation is termed "sparse coding"[1] or "atom decomposition" and can be solved using the following optimization problem:

$$\boldsymbol{\gamma} = \underset{\boldsymbol{\gamma}}{\text{argmin}} \|\mathbf{x} - \mathbf{D}\boldsymbol{\gamma}\|_2^2 \quad s.t. \quad \|\boldsymbol{\gamma}\|_0 \leq q, \tag{1}$$

where q is the number of nonzero coefficients in $\boldsymbol{\gamma}$, often referred to as the "cardinality" of the representation, and the term $\|\boldsymbol{\gamma}\|_0$ is the l_0-norm which counts the number of non-zeros in $\boldsymbol{\gamma}$. This problem is known to be NP-hard in general, implying that even for moderate m (number of atoms), the amount of required computations becomes prohibitive. The group of algorithms which attempt to find an approximated solution to this problem are termed "pursuit algorithms", and they can be roughly divided into two main approaches. The first are relaxation-based methods, such as the "basis-pursuit" [40], which relaxes the norm to be l_1 instead of l_0. The l_1-norm still promotes sparsity while making the optimization problem solvable with polynomial-time methods. The second family of algorithms used to approximate the solution of (1) are the greedy methods, such as the "matching-pursuit" [41], which find an approximation one atom at a time. In this paper we shall mostly address the latter group of pursuit algorithms, and more specifically, the Orthogonal Matching Pursuit (OMP) [42] algorithm, which is efficient and easy to implement.

B. Classic Dictionary Learning

In "dictionary learning" (DL), one attempts to compute the dictionary $\mathbf{D} \in \mathbb{R}^{p\times m}$ that best sparsifies a set of examples, serving as the input data $\mathbf{X} \in \mathbb{R}^{p\times N}$. A commonly used formulation for DL is the following optimization problem:

$$\underset{\mathbf{D},\boldsymbol{\Gamma}}{\text{argmin}} \|\mathbf{X} - \mathbf{D}\boldsymbol{\Gamma}\|_F^2 \quad s.t. \quad 1 \leq i \leq N \quad \|\boldsymbol{\gamma}_i\|_0 \leq q, \tag{2}$$

where $||\cdot||_F$ is the Frobenius norm and $\boldsymbol{\Gamma} = [\boldsymbol{\gamma}_1, \dots, \boldsymbol{\gamma}_N] \in \mathbb{R}^{m\times N}$ is a matrix containing the sparse coefficient vectors of all the input signals. The problem of DL can be solved iteratively using a Block Coordinate Descent (BCR) approach, of alternating between the sparse coding and dictionary update stages. Two such popular methods for DL are the MOD [9] and K-SVD [10].

In MOD [9], once the sparse coefficients in iteration t, $\boldsymbol{\Gamma}_t$, are calculated using a standard pursuit algorithm, the optimization problem becomes:

$$\mathbf{D}_t = \underset{\mathbf{D}}{\text{argmin}} \|\mathbf{X} - \mathbf{D}\boldsymbol{\Gamma}_t\|_F^2. \tag{3}$$

This convex sub-problem leads to the analytical batch update of the dictionary using Least-Squares:

$$\mathbf{D}_t = \mathbf{X}\boldsymbol{\Gamma}_t^T(\boldsymbol{\Gamma}_t\boldsymbol{\Gamma}_t^T)^{-1} = \mathbf{X}\boldsymbol{\Gamma}_t^{\dagger}. \tag{4}$$

[1]The term "Sparse Coding" implies the quest for the sparse solution for an approximate linear system, as opposed to the terminology used in machine learning, where this refers to what we call "Dictionary Learning".

The problem with MOD is the need to compute the pseudo-inverse of the often very-large $\mathbf{\Gamma}$. The K-SVD algorithm by Aharon *et al.* [10] proposed alleviating this and speeding up the overall convergence by updating the dictionary one atom at a time. This amounts to the use of the standard SVD decomposition of rank-1 for the update of each atom.

C. Fisher Discriminant Dictionary Learning (FDDL)

The work reported in [17] proposes an elegant way of performing discriminative DL for the purpose of classification between L classes by modifying and extending the objective function posed in (2). A fundamental feature of this method is the assumption that the dictionary is divided into L disjoint parts, each serving a different class.

Let $\mathbf{X} = [\mathbf{X}_1, \dots, \mathbf{X}_L] \in \mathbb{R}^{p\times N}$ be the input examples of the L classes, where $\mathbf{X}_i \in \mathbb{R}^{p\times n_i}$ are the examples of class i. Denote $\mathbf{D} = [\mathbf{D}_1, \dots, \mathbf{D}_L] \in \mathbb{R}^{p\times M}$ and $\mathbf{\Gamma} = [\mathbf{\Gamma}_1, \dots, \mathbf{\Gamma}_L] \in \mathbb{R}^{M\times N}$ the dictionary and the corresponding sparse representations. The part $\mathbf{\Gamma}_i \in \mathbb{R}^{M\times n_i}$ can be further decomposed as follows: $\mathbf{\Gamma}_i = [(\mathbf{\Gamma}_i^1)^T, \dots, (\mathbf{\Gamma}_i^j)^T, \dots, (\mathbf{\Gamma}_i^L)^T]^T$, where $\mathbf{\Gamma}_i^j \in \mathbb{R}^{m_j\times n_i}$ are the coefficients of the samples $\mathbf{X}_i \in \mathbb{R}^{p\times n_i}$ over the dictionary $\mathbf{D}_j \in \mathbb{R}^{p\times m_j}$. Armed with the above notations, we now turn to describe the objective function proposed in [14] for the discriminative DL task. This objective is composed of two parts. The first is based on the following expression:

$$r\left(\mathbf{X}_i, \mathbf{D}, \mathbf{\Gamma}_i\right) = \\ \|\mathbf{X}_i - \mathbf{D}\mathbf{\Gamma}_i\|_F^2 + \|\mathbf{X}_i - \mathbf{D}_i\mathbf{\Gamma}_i^i\|_F^2 + \sum_{\substack{j=1\\ j\neq i}}^{L} \|\mathbf{D}_j\mathbf{\Gamma}_i^j\|_F^2 \tag{5}$$

The first term demands a good representation of the i-th class samples using the whole dictionary, and the second term further demands a good representation for these examples using their own class' sub-dictionary. The third term is of different nature, forcing the i-th class examples to minimize their reliance on the other sub-dictionaries. Naturally, the overall penalty function will sum the expression in (5) for all the classes i.

We now turn to describe the second part in the objective function, which relies on the Fisher Discriminant Criterion [43]. We define two scatter expressions, both applied to the representations. The first, $S_W(\mathbf{\Gamma})$ computes the within class spread, while the second, $S_B(\mathbf{\Gamma})$ computes the scatter between the classes:

$$\begin{aligned} S_W(\mathbf{\Gamma}) &= \sum\nolimits_{i=1}^{L} \sum\nolimits_{\boldsymbol{\gamma}_k\in\Gamma_i} (\boldsymbol{\gamma}_k - \boldsymbol{\mu}_i)(\boldsymbol{\gamma}_k - \boldsymbol{\mu}_i)^T \\ S_B(\mathbf{\Gamma}) &= \sum\nolimits_{i=1}^{L} n_i(\boldsymbol{\mu}_i - \boldsymbol{\mu})(\boldsymbol{\mu}_i - \boldsymbol{\mu})^T, \end{aligned} \tag{6}$$

and $\boldsymbol{\mu}, \boldsymbol{\mu}_i \in \mathbb{R}^{M\times 1}$ are the mean vectors of the learned sparse coefficient vectors, $\mathbf{\Gamma}$ and $\mathbf{\Gamma}_i$ correspondingly. Naturally, we aim to minimize the first while maximizing the second.

The final FDDL model is defined by the following optimization expression:

$$\begin{aligned} J_{(\mathbf{D},\mathbf{\Gamma})} = \underset{(\mathbf{D},\mathbf{\Gamma})}{\text{argmin}} \Big\{ \sum\nolimits_{i=1}^{L} r(\mathbf{X}_i, \mathbf{D}, \mathbf{\Gamma}_i) + \lambda_1\|\mathbf{\Gamma}\|_1 + \\ \lambda_2 \left[\text{tr}\left(S_W(\mathbf{\Gamma}) - S_B(\mathbf{\Gamma})\right) + \eta\|\mathbf{\Gamma}\|_F^2\right] \Big\}. \end{aligned} \tag{7}$$

The term $\|\mathbf{\Gamma}\|_F^2$ serves as a regularization that ensures the convexity of (6). The detailed optimization scheme of this rather complex expression is described in [17], along with two classification schemes, a global and a local one, depending on the size of the input dataset.

D. Label Consistent KSVD (LC-KSVD)

In [15], [16], an alternative discriminative DL approach is introduced, in which the learning of the dictionary, along with the parameters of the classifier itself, is performed simultaneously, leading to the scheme termed "Label-Consistent K-SVD" (LC-KSVD). These elements are combined in one optimization objective, which is handled using the standard K-SVD algorithm. In order to improve the performance of a linear classifier, an extra term is added to the reconstructive DL optimization function:

$$\underset{\mathbf{D},\mathbf{T},\mathbf{\Gamma}}{\text{argmin}} \|\mathbf{X} - \mathbf{D}\mathbf{\Gamma}\|_F^2 + \alpha\|\mathbf{Q} - \mathbf{T}\mathbf{\Gamma}\|_F^2 \quad s.t\ \forall i,\ \|\boldsymbol{\gamma}_i\|_0 \leq q. \tag{8}$$

The second term encourages the sparse coefficients to be discriminative. More specifically, the matrix $\mathbf{Q} = [\mathbf{q}_1, \dots, \mathbf{q}_N] \in \mathbb{R}^{m\times N}$ stands for the "ideal" sparse-coefficient matrix for discrimination, where the coefficients $\mathbf{q}_{i,j}$ are '1' if the classes of the atom $\mathbf{d}_i$ and the input signal $\mathbf{x}_j$ match, and '0' otherwise. The matrix $\mathbf{T} \in \mathbb{R}^{m\times m}$ transforms the sparse codes $\mathbf{\Gamma}$ to their idealized versions in $\mathbf{Q}$. This term thus promotes identical sparse codes for input signals from the same class and orthogonal sparse codes for signals from different classes.

In addition to the discriminative term added above, the authors in [15] propose learning the linear classifier within the framework of the DL. A linear predictive classifier is used of the form: $f(\boldsymbol{\gamma}, \mathbf{\Theta}) = \mathbf{\Theta}\boldsymbol{\gamma}$, where $\mathbf{\Theta} \in \mathbb{R}^{L\times m}$. The overall objective function suggested is:

$$\begin{aligned} &\underset{\mathbf{D},\mathbf{\Theta},\mathbf{T},\mathbf{\Gamma}}{\text{argmin}} \Big\{ \|\mathbf{X} - \mathbf{D}\mathbf{\Gamma}\|_F^2 + \alpha\|\mathbf{Q} - \mathbf{T}\mathbf{\Gamma}\|_F^2 \\ &\quad + \beta\|\mathbf{H} - \mathbf{\Theta}\mathbf{\Gamma}\|_F^2 \Big\}, \quad s.t. \quad \forall i,\ \|\boldsymbol{\gamma}_i\|_0 \leq q, \end{aligned} \tag{9}$$

where the classification error is represented by the term $\|\mathbf{H} - \mathbf{\Theta}\mathbf{\Gamma}\|_2^2$, $\mathbf{\Theta}$ contains the classifier parameters, $\mathbf{H} = [\mathbf{h}_1, \dots, \mathbf{h}_N] \in \mathbb{R}^{L\times N}$ is the label matrix of all input examples, in which the vector $\mathbf{h}_i = [0, 0, \dots, 0, 1, 0, \dots, 0]^T$ contains only zeros apart from the index corresponding to the class of the example. The optimization function in (9) can also be written as follows:

$$\underset{\mathbf{D}_{new},\mathbf{\Gamma}}{\text{argmin}} \|\mathbf{X}_{new} - \mathbf{D}_{new}\mathbf{\Gamma}\|_F^2, \quad s.t. \quad \forall i, \quad \|\boldsymbol{\gamma}_i\|_0 \leq q, \tag{10}$$

where $\mathbf{X}_{new} = \left(\mathbf{X}^T, \sqrt{\alpha}\mathbf{Q}^T, \sqrt{\beta}\mathbf{H}^T\right)^T \in \mathbb{R}^{(p+m+L)\times N}$ and $\mathbf{D}_{new} = \left(\mathbf{D}^T, \sqrt{\alpha}\mathbf{T}^T, \sqrt{\beta}\mathbf{\Theta}^T\right)^T \in \mathbb{R}^{(p+m+L)\times m}$. The unified columns in $\mathbf{D}_{new}$ are all normalized to unit l_2 norm. The optimization objective in (10) can be solved using standard DL algorithms, such as K-SVD.

The authors propose two cases of LC-KSVD: LC-KSVD2, in which the parameters of the classifier are learned along with the dictionary, as shown in (9) and the second, LC-KSVD1, in which they are calculated separately by: $\mathbf{\Theta} =$

$\left(\mathbf{\Gamma}\mathbf{\Gamma}^T + \tau_2\mathbf{I}\right)^{-1}\mathbf{\Gamma}\mathbf{H}^T$. More details on these expressions and the numerical scheme for minimizing the objective function can be found in [15], [16]. A new sample $\mathbf{x}$ is classified by first sparse coding over the dictionary $\hat{\mathbf{D}}$, and then, applying the classifier $\hat{\mathbf{\Theta}}$ to estimate the label j.

III. Kernel Dictionary Learning

This section focuses on kernel sparse representations, with emphasis of the kernel-KSVD method by Nguyen *et al.*, which we will compare with later on this paper.

A. Kernels - The Basics

In machine learning, it is well-known that a non-linear mapping of the signal of interest to higher dimension may improve its discriminability in tasks such as classification. Let $\mathbf{x} \in \mathcal{X}$ be a signal in input space, which is embedded to a higher dimensional space $\mathcal{F}$ using the mapping $\Phi, \mathbf{x} \in \mathbb{R}^p \rightarrow \Phi(\mathbf{x}) \in \mathbb{R}^P$ ($P \gg p$ and it might even be infinite). The space in which this new signal $\Phi(\mathbf{x})$ lies is called the "feature space". The next step in machine learning algorithms, in particular in classification, would be learning a classifier based on the mapped input signals and labels. This task can be prohibitive if tackled directly. A way around this hurdle is the "kernel trick" [44], [45], which allows computing inner products between pairs of signals in the feature space, using a simple nonlinear function operating on the two signals in input space:

$$\kappa\left(\mathbf{x}, \mathbf{x}'\right) = \langle \Phi(\mathbf{x}), \Phi(\mathbf{x}') \rangle = \Phi(\mathbf{x})^T \Phi(\mathbf{x}'), \qquad (11)$$

where κ is the "kernel". This relation holds true for positive-semi-definite (PSD) kernels, which according to Mercer's theorem, generate a RKHS (Reproducing Kernel Hilbert Space) [19]. Thus, suppose that the learning algorithm can be fully posed in terms of inner products. In such a case, one can achieve a "kernelized" version by swapping the inner products with the kernel function, without ever operating in the feature space. In case there are N input signals $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_N] \in \mathbb{R}^{p \times N}$, the "kernel matrix" $\mathbf{K} \in \mathbb{R}^{N \times N}$ holds the kernel values of all pairs of input signals:

$$\mathbf{K}_{i,j} = \kappa(\mathbf{x}_i, \mathbf{x}_j) = \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle, \quad \forall i, j = 1..N. \qquad (12)$$

An inherent constraint in kernel algorithms is the fact that the solution vectors, for example the principal components in KPCA, are expansions of the mapped signals in feature space:

$$\mathbf{v} = \sum_{i=1}^{N} \alpha_i \Phi(\mathbf{x}_i). \qquad (13)$$

The subspace in which the possible solutions lie, can be viewed as an N dimensional surface residing in $\mathcal{F}$ [46]. Motivated by the inability to directly approach the mapped signals in feature space, researchers have suggested embedding the N dimensional surface to a finite Euclidean subspace, where all geometrical properties, such as distances and angles between pairs of $\Phi(\mathbf{x}_i)'s$, are preserved [47]. The embedding is called the "kernel empirical map" and the resulting subspace is referred to as the "empirical feature subspace". One way to embed a given signal $\mathbf{x}$ to the empirical feature space is by calculating kernel values originating from inner products with all input training examples: $\mathbf{x} \rightarrow [\kappa(\mathbf{x}, \mathbf{x}_1), \ldots, \kappa(\mathbf{x}, \mathbf{x}_N)]^T$. As we shall see hereafter, we will propose a better tuned variation of this approach.

B. Kernel Dictionary Learning

A straightforward way to kernelize DL would be exchanging all signals (and dictionary atoms) with their respective representations in feature space: $\mathbf{x} \rightarrow \Phi(\mathbf{x}), \mathbf{d} \rightarrow \Phi(\mathbf{d})$ and rephrasing the algorithm such that it contains solely inner products between pairs of these ingredients. A difficulty with this approach is that during the learning process, the dictionary atoms are in feature space. As there is no exact reverse mapping from the updated inner products to their corresponding signals in input space, there is no direct way of accessing the updated atoms, as practiced in linear DL.

In order to solve this problem, the authors in [29] suggest decomposing the dictionary in feature space into: $\Phi(\mathbf{D}) = \Phi(\mathbf{X})\mathbf{A}$, where $\Phi(\mathbf{X})$ is the constant part, called the "base-dictionary", which consists of all mapped input signals, and $\mathbf{A}$ is the only part updated during the learning, called the "coefficient-dictionary". The kernel DL can now be formulated as the following optimization problem:

$$\underset{\mathbf{A},\mathbf{\Gamma}}{\text{argmin}} \|\Phi(\mathbf{X}) - \Phi(\mathbf{X})\mathbf{A}\mathbf{\Gamma}\|_F^2 \quad s.t. \quad \forall i = 1..N \quad \|\boldsymbol{\gamma}_i\|_0 \leq q. \qquad (14)$$

Similarly to linear DL, this optimization problem can be solved iteratively by first performing sparse coding with a fixed dictionary $\mathbf{A}$, then updating the dictionary according to the computed sparse representations $\mathbf{\Gamma}$, and so on, until convergence is reached. The kernelized equivalent of sparse coding is given by:

$$\underset{\boldsymbol{\gamma}}{\text{argmin}} \|\Phi(\mathbf{z}) - \Phi(\mathbf{X})\mathbf{A}\boldsymbol{\gamma}\|_2^2 \quad s.t. \quad \|\boldsymbol{\gamma}\|_0 \leq q, \qquad (15)$$

where $\mathbf{z}$ is the input signal. As mentioned earlier, the sparse coding algorithm we focus on in this paper, as well as in Nguyen's KKSVD [29], is the OMP [42] and its kernel version, KOMP [29]. Table I presents two of the main stages in the OMP algorithm, which are the Atom-Selection (AS) and Least-Squares (LS) stages, and their kernelized version. As can be seen, these stages can be completely represented using the coefficient dictionary $\mathbf{A}$, the sparse representation vector $\boldsymbol{\gamma}$ and the kernel functions $\mathbf{K}(\mathbf{X}, \mathbf{X}) \in \mathbb{R}^{N \times N}$ and $\mathbf{K}(\mathbf{z}, \mathbf{X}) = [\kappa(\mathbf{z}, \mathbf{x}_1), \ldots, \kappa(\mathbf{z}, \mathbf{x}_N)] \in \mathbb{R}^{1 \times N}$.

The dictionary update stage, can also be kernelized. In the MOD algorithm [9], the update of $\mathbf{A}$ in iteration $t+1$ is given by: $\mathbf{A}_{t+1} = \mathbf{\Gamma}_t^T(\mathbf{\Gamma}_t\mathbf{\Gamma}_t^T)^{-1} = \mathbf{\Gamma}_t^\dagger$, being the solution to: $\underset{\mathbf{A}}{\text{argmin}} \|\Phi(\mathbf{X}) - \Phi(\mathbf{X})\mathbf{A}\mathbf{\Gamma}\|_F^2$. A similar update can be derived for the K-SVD algorithm, as described in depth in [29], [31].

C. Difficulties in KDL

There are a few difficulties that arise when dealing with kernels, and specifically in kernel DL. In the input space, a signal $\mathbf{x} \in \mathbb{R}^p$ can be described using its own p features, while in feature space it is described by its relationship with *all of the other N input signals*. The runtime and memory complexity of

TABLE I
COMPLEXITY OF THE ATOM SELECTION (AS) AND THE LEAST SQUARE (LS) STAGES IN LINEAR AND KERNEL-OMP. $\mathbf{I}_S$ IS THE SUPPORT VECTOR IN ITERATION t, STATING THE CHOSEN DICTIONARY ATOMS AND $|\mathbf{I}_S|$ ITS LENGTH, EQUAL TO t. $\mathbf{D}_S$, $\mathbf{A}_S$ AND $\boldsymbol{\gamma}_S$ ARE SUB-MATRICES OF $\mathbf{D}$, $\mathbf{A}$, AND $\boldsymbol{\gamma}$, RESPECTIVELY, CORRESPONDING TO $\mathbf{I}_S$. $\mathbf{r}_t$ IS THE RESIDUAL.

	Term	**Complexity**
OMP-AS	$\langle \mathbf{r}_t, \mathbf{d}_j \rangle = \langle \mathbf{z} - \mathbf{D}_S\boldsymbol{\gamma}_S, \mathbf{d}_j \rangle = \mathbf{z}^T\mathbf{d}_j - \boldsymbol{\gamma}_S^T\mathbf{D}_S^T\mathbf{d}_j$	$O\left(p\lvert\mathbf{I}_S\rvert + p\right)$
KOMP-AS [29]	$\mathbf{K}(\mathbf{z},\mathbf{X})\mathbf{a}_j - \boldsymbol{\gamma}_S^T\mathbf{A}_S^T\mathbf{K}(\mathbf{X},\mathbf{X})\mathbf{a}_j$	$O\left(N^2 + \lvert\mathbf{I}_S\rvert N + N\right)$
OMP-LS	$\boldsymbol{\gamma}_S = \left(\mathbf{D}_S^T\mathbf{D}_S\right)^{-1}\mathbf{D}_S^T\mathbf{z}$	$O\left(p\lvert\mathbf{I}_S\rvert^2 + p\lvert\mathbf{I}_S\rvert + \lvert\mathbf{I}_S\rvert^3\right)$
KOMP-LS [29]	$\boldsymbol{\gamma}_S = \left[\mathbf{A}_S^T\mathbf{K}(\mathbf{X},\mathbf{X})\mathbf{A}_S\right]^{-1}\left(\mathbf{K}(\mathbf{z},\mathbf{X})\mathbf{A}_S\right)^T$	$O\left(N^2\lvert\mathbf{I}_S\rvert + N\lvert\mathbf{I}_S\rvert + \lvert\mathbf{I}_S\rvert^3\right)$

a kernel learning algorithm changes accordingly and depends on the number of input signals, instead of on the dimension of the signals. This observation is also true for Nguyen's KDL where the kernel matrix $\mathbf{K}$ is used during the sparse coding and dictionary update stages, and must be stored in full. In applications where the number of input samples is large, this dependency on the kernel matrix becomes prohibitive. In table I, one can see the complexity of the main stages in the KOMP algorithm and compare it to the linear OMP version. It is clear that both the atom-selection and the least-squares stages are governed quadratically on the size of the input dataset.

Another inherent difficulty in kernel methods in general, is the need to tailor each algorithm such that it is formulated solely through inner products. This constraint creates complex and cumbersome expressions and is not always possible, as some steps in the algorithm may contain a mixture of the signals and their mapped version, as in [24].

IV. The Proposed Algorithm

Section II and III gave some background to the task we address in this paper. We saw that kernelization of the DL task can be beneficial, but unfortunately, we also identified key difficulties this process is accompanied by. In this work we aim to propose a systematic and simple path for kernelizing existing DL algorithms, in a way that will avoid the problems mentioned above. More specifically, we desire to be able to kernelize any existing DL algorithm, be it unsupervised or supervised, and do so while being able to work on massive training sets without the need to compute, store, or manipulate the kernel matrix K. In this section we outline such a solution, by carefully describing its key ingredients.

A. Kernel matrix approximation

Let $\mathbf{X} \in \mathbb{R}^{p\times N}$ be the input signals and $\mathbf{K} \in \mathbb{R}^{N\times N}$ their corresponding kernel matrix. As long as the kernel satisfies Mercer's conditions of positive-semi-definiteness it can be written as an inner product between mapped signals in feature space: $\mathbf{K}_{i,j} = \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j)\rangle$. Assume, for the sake of the discussion here, that the kernel function applies a simple inner product, i.e.: $\mathbf{K}_{i,j} = \langle \mathbf{f}_i, \mathbf{f}_j\rangle = \mathbf{f}_i^T\mathbf{f}_j$, where $\mathbf{f}_i, \mathbf{f}_j$ are the feature vectors of dimension k, corresponding to $\mathbf{x}_i$ and $\mathbf{x}_j$, respectively. Thus, the kernel matrix would have the form: $\mathbf{K} = \mathbf{F}^T\mathbf{F} = \Phi(\mathbf{X})^T\Phi(\mathbf{X})$, where $\mathbf{F} \in \mathbb{R}^{k\times N}$ and k ($\leq rank(\mathbf{K}) \leq N$) is the desired dimension of the approximated feature space. One can refer to the vectors $\{\mathbf{f}_i\}_{i=1}^N$ in $\mathbf{F}$ as "Virtual Samples" [37]. This way, instead of learning using the kernel matrix $\mathbf{K}$, one could work on these virtual samples directly using a linear learning algorithm, leading to the same outcome. In the following, we will leverage on this insight.

The kernel matrix is generally PSD, and as such can be approximated using eigen-decomposition as follows: $\mathbf{K} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^T$, where $\boldsymbol{\Lambda} \in \mathbb{R}^{k\times k}$ is a diagonal matrix containing the top-k eigenvalues of $\mathbf{K}$ in descending order and $\mathbf{U} \in \mathbb{R}^{N\times k}$ contains the matching orthonormal eigenvectors. The virtual samples can be achieved by:

$$\mathbf{F} = \boldsymbol{\Lambda}^{1/2}\mathbf{U}^T = \boldsymbol{\Lambda}^{-1/2}\mathbf{U}^T\mathbf{K}. \tag{16}$$

The virtual samples can be viewed as a mapping of the original input signals to a k-dimensional empirical feature space.

$$\mathbf{x} \rightarrow \boldsymbol{\Lambda}^{-1/2}\mathbf{U}^T\left(\kappa(\mathbf{x},\mathbf{x}_1), \kappa(\mathbf{x},\mathbf{x}_2), \ldots, \kappa(\mathbf{x},\mathbf{x}_N)\right)^T. \tag{17}$$

This "linearization" is the mediator between kernel DL which is obligated to store and manipulate the kernel matrix $\mathbf{K}$, and linear DL that can deal with very large datasets. The decomposition of the matrix $\mathbf{K}$ to its eigenvalues and eigenvectors is a demanding task in itself, both in time $O(N^2k)$ and in space $O(N^2)$. Next we will show how a good approximation of the matrix $\mathbf{K}$ can be constructed with only a subset of its columns, using the popular Nyström method.

B. Nyström method

A common necessity in many algorithms in signal processing and machine learning is deriving a relatively accurate and efficient approximation of a large matrix. An attractive method that has gained popularity in recent years is the Nyström method [48], which generates a low-rank approximation using a subset of the input data. The original Nyström method, first introduced by Williams and Seeger [48], proposed using uniform sampling without replacement.

Let $\mathbf{K} \in \mathbb{R}^{N\times N}$ be a symmetric PSD matrix, and in particular for the discussion here, a kernel matrix. Suppose $c \leq N$ columns from the matrix $\mathbf{K}$ are sampled uniformly without replacement to form the reduced matrix $\mathbf{C} \in \mathbb{R}^{N\times c}$. Without loss of generality, the matrices $\mathbf{C}$ and $\mathbf{K}$ can be decomposed as follows:

$$\mathbf{C} = \begin{bmatrix}\mathbf{W}\\ \mathbf{S}\end{bmatrix} \quad and \quad \mathbf{K} = \begin{bmatrix}\mathbf{W} & \mathbf{S}^T\\ \mathbf{S} & \mathbf{B}\end{bmatrix}, \tag{18}$$

where $\mathbf{W} \in \mathbb{R}^{c\times c}$ is the kernel matrix of the intersection of the chosen c columns with c rows, $\mathbf{B} \in \mathbb{R}^{(N-c)\times(N-c)}$ is the kernel matrix composed of the $N-c$ remaining rows and columns, and $\mathbf{S} \in \mathbb{R}^{(N-c)\times c}$, is a mixture of both. The Nyström method uses both $\mathbf{C}$ and $\mathbf{W}$ to construct an approximation of the matrix $\mathbf{K}$ as follows [48]:

$$\mathbf{K} \approx \mathbf{C}\mathbf{W}^{\dagger}\mathbf{C}^{T}, \tag{19}$$

where $(\cdot)^{\dagger}$ denotes the pseudo-inverse. The symmetric matrix $\mathbf{W}$ can also be posed in terms of eigenvalues and eigenvectors: $\mathbf{W} = \mathbf{V}\mathbf{\Sigma}\mathbf{V}^{T}$, where $\mathbf{\Sigma}$ is a diagonal matrix containing the eigenvalues of $\mathbf{W}$ in descending order and $\mathbf{V}$ contains the matching orthonormal eigenvectors. The pseudo-inverse of $\mathbf{W}$ is given by $\mathbf{W}^{\dagger} = \mathbf{V}\mathbf{\Sigma}^{\dagger}\mathbf{V}^{T}$. The expression of $(\mathbf{W}^{\dagger})^{1/2}$ can be similarly derived: $(\mathbf{W}^{\dagger})^{1/2} = (\mathbf{\Sigma}^{\dagger})^{1/2}\mathbf{V}^{T}$.

We can represent $\mathbf{K}$ as before, using linear inner-products of the virtual samples, and plug in Nyström's approximation:

$$\mathbf{K} = \mathbf{F}^{T}\mathbf{F} = \mathbf{C}\mathbf{W}^{\dagger}\mathbf{C}^{T} = \mathbf{C}\mathbf{V}\mathbf{\Sigma}^{\dagger}\mathbf{V}^{T}\mathbf{C}^{T}, \tag{20}$$

and derive the final k-dimensional ($k \le c$) expression of the virtual samples by:

$$\mathbf{F}_{k} = \left(\mathbf{\Sigma}_{k}^{\dagger}\right)^{1/2}\mathbf{V}_{k}^{T}\mathbf{C}^{T}, \tag{21}$$

where $\mathbf{\Sigma}_{\boldsymbol{k}} = \text{diag}(\sigma_1, \dots, \sigma_k) \in \mathbb{R}^{k\times k}$ contains the k largest eigenvalues of $\mathbf{W}$ and $\mathbf{V}_k \in \mathbb{R}^{c\times k}$, the corresponding orthonormal eigenvectors.

After performing the Nyström approximation, the space complexity of kernel DL reduces from $O(N^2)$ to $O(Nc)$, the size of the matrix $\mathbf{C}$, which is used during the computation of the virtual samples. The time complexity of the Nyström method is $O(Nck + c^2k)$, where $O(Nck)$ represents the multiplication of $\mathbf{V}_k^T\mathbf{C}^T$ and $O(c^2k)$ stands for the eigenvalue decomposition (and inversion) of the reduced matrix $\mathbf{W}_k$.

Note that the process of computing the virtual samples may seem inefficient, but it is performed only once, after which the complexity of the DL is dictated by the chosen algorithm, and not by the "kernelization". In addition, in scenarios where the number of input examples is very large, the ratio c/N in Nyström's method can be reduced greatly, i.e. $c \ll N$, making the approximation even less dominant in terms of runtime and memory, while retaining almost the same accuracy.

C. Sampling Techniques

Since the Nyström method creates an approximation of a large symmetric matrix based on a subset of its columns, the chosen sampling scheme plays an important part. The basic method proposed originally by Williams and Seeger was uniform sampling without replacement [48]. The columns of the Gram matrix can be alternatively sampled from a nonuniform distribution. Two such examples of nonuniform sampling include "column-norm sampling" [49], where the weight of the i-th column $\mathbf{k}^i$ is its l_2 norm: $p_i = \|\mathbf{k}^i\|^2/\|\mathbf{K}\|_F^2$, and "diagonal sampling" [50] where the weight is proportional to the corresponding diagonal element[2]: $p_i = \mathbf{K}_{ii}^2/\sum_{i=1}^{N}\mathbf{K}_{ii}^2$. These methods can be made more sophisticated but require additional complexity: $O(N)$ in time and space for diagonal sampling and $O(N^2)$ for column-norm sampling. A comprehensive theoretical and empirical comparison of these three methods is provided in [51].

[2]This is ineffective for constant diagonal values as in RBF.

In [52], Zhang *et al.* suggested an alternative approach of selecting a few "representative" columns in $\mathbf{K}$ by first performing K-means clustering, then computing the reduced matrix $\mathbf{C}$ based on these so-called "cluster centers". Denote by $\mathbf{X}_R$ the resulting c cluster centers, created from the original data $\mathbf{X}$. The computation of the kernel matrices $\mathbf{C}$ and $\mathbf{W}$ would be: $\mathbf{C} = \mathbf{K}(\mathbf{X}, \mathbf{X}_R)$ and $\mathbf{W} = \mathbf{K}(\mathbf{X}_R, \mathbf{X}_R)$. Zhang *et al.* also show that the combination of k-means clustering with the Nyström method minimizes the approximation error.

Another appealing sampling technique has been suggested in the context of coresets [53]. The idea is to sample the given data by emphasizing unique samples that are ill-represented by the others. In the context of our problem, we sample c signals from $\mathbf{X}$ according to the following distribution: $p_i = err(\mathbf{x}_i, \boldsymbol{\mu})/\sum_{\mathbf{x}_i\in\mathbf{X}} err(\mathbf{x}_i, \boldsymbol{\mu})$, where $err(\mathbf{x}_i, \boldsymbol{\mu}) = ||\mathbf{x}_i - \boldsymbol{\mu}\gamma||_2^2$ is the representation error of the signal $\mathbf{x}_i$, corresponding to the mean of all training signals $\boldsymbol{\mu} = (1/N)\sum_{i=1}^{N}\mathbf{x}_i$ and γ is a scalar.

D. Linearized Kernel Dictionary Learning (LKDL)

Let $\{\mathbf{x}_i, y_i\}_{i=1}^{N}$ be a labeled training set, arranged as a structure in L categories: $\mathbf{X}_{train} = [\mathbf{X}_1, \dots, \mathbf{X}_L] \in \mathbb{R}^{p\times N}$, where $\mathbf{X}_i$ contains the training samples that belong to the i-th class and $N = \sum_{i=1}^{L} n_i$. Our process of kernel DL is divided in two parts: the first, a pre-processing stage that creates new virtual training and test samples, followed by a second stage of applying a standard DL. This whole process is termed "Linearized Kernel Dictionary Learning" (LDKL).

The pre-processing stage is shown in algorithm 1. First, the initial training set $\mathbf{X}_{train}$ is sampled using one of the techniques mentioned in section IV-C, creating the reduced set $\mathbf{X}_R = [\mathbf{x}_{R_1}, \dots, \mathbf{x}_{R_c}] \in \mathbb{R}^{p\times c}$. Then the matrix $\mathbf{C} \in \mathbb{R}^{N\times c}$ in Nyström's method is calculated by simply applying the chosen kernel on each and every pair of columns in $\mathbf{X}_{train}$ and $\mathbf{X}_R$. Next, the reduced matrix $\mathbf{W} \in \mathbb{R}^{c\times c}$ is both calculated and later on inverted using rank-k eigen-decomposition. Finally the virtual training samples $\mathbf{F}_{train} \in \mathbb{R}^{k\times N}$ are calculated using equation (21). The Nyström method permits approximating a new test vector $\mathbf{f}_{test}$ using equation (17), by using the mapping already calculated based on the training set, and multiplying by the joint kernel vector of the sampled set $\mathbf{X}_R$ and the current test sample: $\mathbf{K}(\mathbf{X}_R, \mathbf{x}_{test})$:

$$\boldsymbol{f}_{test} = \left(\mathbf{\Sigma}_{k}^{\dagger}\right)^{1/2}\mathbf{V}_{k}^{T}\left[\kappa(\mathbf{x}_{R_1}, \mathbf{x}_{test}), \dots, \kappa(\mathbf{x}_{R_c}, \mathbf{x}_{test}))\right]^{T}. \tag{22}$$

Once the training and test sets are represented as virtual samples: $\mathbf{F}_{train}$ and $\mathbf{F}_{test}$, any linear DL-based classification method can be implemented. In the context of classification we follow Nguyen's "distributive" approach [31] of learning L separate dictionaries $[\mathbf{D}_1, \dots, \mathbf{D}_L]$ per each class, then classifying each test sample by first computing its sparse coefficient vector over each of the dictionaries $\{\mathbf{D}_i\}_{i=1}^{L}$, and finally choosing the class corresponding to the smallest reconstruction error: $r_i = \|\boldsymbol{f}_{test} - \mathbf{D}_i\boldsymbol{\gamma}_i\|^2, \quad \forall i = 1..L$.

Algorithm 1 LKDL Pre-Processing

```
1: Input: $\mathbf{X}_{train} = [\mathbf{X}_1, \dots, \mathbf{X}_L]$, $\mathbf{X}_{test}$, the kernel $\kappa$, $smp\_method$, $c, k$
2: $\mathbf{X}_R = sub\_sample(\mathbf{X}_{train}, smp\_method, c)$
3: Compute $\mathbf{C}_{train} = \mathbf{K}(\mathbf{X}_{train}, \mathbf{X}_R)$
4: Compute $\mathbf{W} = \mathbf{K}(\mathbf{X}_R, \mathbf{X}_R)$
5: Approximate $\mathbf{W}_k$ using $k$ largest eigenvalues and eigenvectors $\mathbf{W}_k = \mathbf{V}_k \mathbf{\Sigma}_k \mathbf{V}_k^T$
6: Compute virtual train set $\mathbf{F}_{train} = \left(\mathbf{\Sigma}_k^{\dagger}\right)^{1/2} \mathbf{V}_k^T \mathbf{C}_{train}^T$
7: Compute $\mathbf{C}_{test} = \mathbf{K}(\mathbf{X}_{test}, \mathbf{X}_R)$
8: Compute virtual test set $\mathbf{F}_{test} = \left(\mathbf{\Sigma}_k^{\dagger}\right)^{1/2} \mathbf{V}_k^T \mathbf{C}_{test}^T$
9: Output: $\mathbf{F}_{train} = [\mathbf{F}_1, \dots, \mathbf{F}_L]$, $\mathbf{F}_{test}$
```

E. Limitations and Improvements

The largest limitation of Nyström-based kernelized features is the need to perform pre-training on part of the input data to calculate the matrices $\mathbf{C}$ and $\mathbf{W}$ and eigen-decompose them. The need to calculate the embedding before the actual DL requires the storage of a rather large matrix $\mathbf{C} \in \mathbb{R}^{N \times c}$. For this purpose and the case where input training or test examples do not fit into memory, we propose "mini-batch LKDL".

Let the N input training samples be separately stored in disk (labels can be scrambled) in n_B equally sized mini-batches: $\mathbf{X} = [\mathbf{X}_1, ..., \mathbf{X}_{n_B}]$, where $\mathbf{X}_i \in \mathbb{R}^{p \times N/n_B}$. Each mini-batch is read into memory, sampled using one of the sampling strategies shown above, and stored in a matrix $\mathbf{X}_R = [\mathbf{X}_{R_1}, ..., \mathbf{X}_{R_{n_B}}] \in \mathbb{R}^{p \times c}$, where $\mathbf{X}_{R_i} \in \mathbb{R}^{p \times c/n_B}\ \forall i = [1..n_B]$. After collecting samples from all mini-batches (one mini-batch at a time), we compute the matrix $\mathbf{W} = \mathbf{K}(\mathbf{X}_R, \mathbf{X}_R) \in \mathbb{R}^{c \times c}$ and decompose it using eigen-decomposition $\mathbf{W}_k = \mathbf{V}_k \mathbf{\Sigma}_k \mathbf{V}_k^T$, exactly as in steps 4-5 in algorithm [1]. The virtual training set $\mathbf{F}_{train} = [\mathbf{F}_1, ..., \mathbf{F}_{n_B}] \in \mathbb{R}^{k \times N}$ is computed, one mini-batch at a time, using a subset of the matrix $\mathbf{C} = [\mathbf{K}(\mathbf{X}_1, \mathbf{X}_R), ..., \mathbf{K}(\mathbf{X}_{n_B}, \mathbf{X}_R)]^T \in \mathbb{R}^{N \times c}$, and an already computed decomposition of $\mathbf{W}$:

$$\mathbf{F}_i = \left(\mathbf{\Sigma}_k^{\dagger}\right)^{1/2} \mathbf{V}_k^T \mathbf{C}_i^T \quad \forall i = 1..n_B, \tag{23}$$

where $\mathbf{C}_i \in \mathbb{R}^{N/n_B \times c}$ is the i-th portion of the matrix $\mathbf{C}$. The virtual test samples can similarly be stored in mini-batches or computed one sample at a time using equation 22. Once we have the virtual training and test examples we can deploy any mini-batch or online DL algorithm, e.g. [36].

After dealing with the matrix $\mathbf{C}$, mini-batch LKDL still has scaling issues in the form of $\mathbf{W} \in \mathbb{R}^{c \times c}$. For example, suppose our dataset has 10 million training examples and we sample only 1%. The corresponding matrix $\mathbf{W}$ will be of size $100,000 \times 100,000$, which cannot be stored in memory, nor can we compute the eigen decomposition of such a matrix. As one last point we add the following: in some problems/datasets, the Nyström approximation may not deliver a sufficient performance due to the inability to effectively subsample the data.

F. Data-independent kernel empirical maps

The field of kernel empirical maps can be divided to two categories: The first refers to data dependent features where the kernel matrix is approximated using a subset of the data. The Nyström method (on which our virtual samples are based), is a classic example of such an approach. The second category suggests data independent features where these are created randomly, without prior training, with the goal of mimicking the numerical values of the kernel function. In this section we describe two related methods pertaining to the latter category. Later on in the results section we shall provide comparisons between these two and our scheme.

In 2007, Rahimi and Recht [38] proposed a revolutionary idea of creating randomized kernel features, without any prior training, termed "Random Fourier Features" or generally referred to as "Random Kitchen Sinks". This method relied strongly on Bochner's theorem which states: *"A continuous kernel $\kappa(\mathbf{x}, \mathbf{x}') = \kappa(\mathbf{x} - \mathbf{x}')$ on $\mathcal{R}^p$ is positive definite if and only if $\kappa(\delta)$ is the Fourier transform of a non-negative measure"*. The immediate result from this, is that the Fourier transform of a PSD shift-invariant kernel (which can be expressed as $\kappa(\mathbf{x}, \mathbf{x}') = \kappa(\mathbf{x} - \mathbf{x}')$), $\kappa(\delta)$, can be treated as a probability distribution function. One can thus create k-dimensional feature vectors, whose inner product approximates the kernel's numerical value as follows:

$$\begin{aligned} \kappa(\mathbf{x} - \mathbf{x}') &= \int_{\mathcal{R}^p} p_\kappa(\boldsymbol{\omega}) e^{j\boldsymbol{\omega}^T(\mathbf{x}-\mathbf{x}')} d(\boldsymbol{\omega}) \approx \\ &\approx \frac{1}{k}\sum_{i=1}^{k} e^{j\boldsymbol{\omega}_i{}^T(\mathbf{x}-\mathbf{x}')} = \mathbf{z}(\mathbf{x})^T \mathbf{z}(\mathbf{x}') \quad \boldsymbol{\omega}_i \sim p_\kappa(\boldsymbol{\omega})\ iid \end{aligned} \tag{24}$$

where the samples $\boldsymbol{\omega}_i \in \mathbb{R}^p$ are drawn i.i.d. from the distribution function $p_\kappa(\boldsymbol{\omega})$, which is essentially the Fourier transform of the shift invariant kernel. The vectors $\mathbf{z}(\mathbf{x})$ and $\mathbf{z}(\mathbf{x}')$ are the low dimensional kernelized features, p is the original signal's dimension and k is the desired new dimension of the approximated signal. Instead of using k complex features, one can create a $2k$-dimensional embedding using $\sin(\cdot)$ and $\cos(\cdot)$ functions. The full scheme is given in algorithm 2.

Algorithm 2 Random Fourier Features [38]

```
1: Input: A positive definite shift invariant kernel $\kappa(\mathbf{x}, \mathbf{x}') = \kappa(\mathbf{x} - \mathbf{x}')$.
2: Output: A randomized feature map $\mathbf{z}(\mathbf{x}) : \mathcal{R}^p \to \mathcal{R}^{2k}$ so that $\mathbf{z}(\mathbf{x})^T \mathbf{z}(\mathbf{x}') \approx \kappa(\mathbf{x} - \mathbf{x}')$
3: Compute the Fourier transform of the kernel: $p_\kappa(\omega) = \frac{1}{2\Pi} \int e^{-j\omega\Delta} \kappa(\Delta) d\Delta$.
4: Draw $k$ iid samples $\boldsymbol{\omega}_1, ..., \boldsymbol{\omega}_k \in \mathcal{R}^p$ from $p_\kappa(\omega)$.
5: $\mathbf{z}(\mathbf{x}) \equiv \sqrt{\frac{1}{k}} \left[\cos(\boldsymbol{\omega}_1^T \mathbf{x}) ... \cos(\boldsymbol{\omega}_k^T \mathbf{x}) \sin(\boldsymbol{\omega}_1^T \mathbf{x}) ... \sin(\boldsymbol{\omega}_k^T \mathbf{x})\right]$.
```

The Fourier transform of the commonly used RBF kernel $\kappa(\Delta) = \exp^{-\Delta^2/2\sigma^2}$ is very conveniently the normal probability distribution: $p_\kappa(\omega) \sim \mathcal{N}(0, \frac{1}{\sigma^2})$. Furthermore, if $\sigma = 1$, the computation of the features amounts to drawing a random matrix $\mathbf{\Omega} \in \mathbb{R}^{k \times p}$, where $\omega_{i,j} \sim \mathcal{N}(0, 1)$, multiplying it by the signal $\mathbf{x}$ for which we want to compute the feature and computing the $\sin(\cdot), \cos(\cdot)$ functions of each component.

The Fastfood algorithm by Le. *et al.* [39] goes one step further. Instead of storing the Gaussian matrix, $\mathbf{\Omega} \in \mathbb{R}^{k \times p}$, and multiplying by it each time we compute a new feature vector, one could decompose it into a multiplication of Hadamard and Gaussian scaling matrices. The authors show that the result of this multiplication approximates a natural Gaussian distribution matrix. Each matrix in the decomposition is either diagonal or can be generated efficiently in FFT-like fashion. This decomposition reduces runtime and storage complexity from $O(kp)$ to $O(k \log p)$.

At first sight, these two techniques of random kernelized feature creation can pose an alternative to our Nyström-based features. However, both these methods focus on optimizing the training and evaluation process of SVM and its variants, while we target the use of such features in tasks of DL. While the motivation may appear the same, the above-mentioned algorithms are essentially different. For example, the classification process in linear SVM consists of evaluating a decision function that applies only one inner product: $f(\mathbf{x}) = \boldsymbol{\theta}^T \mathbf{x}$ ($\boldsymbol{\theta}$ being the classifier parameters). This evaluation becomes at least N times slower in the case of kernel SVM: $f(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i \kappa(\mathbf{x}, \mathbf{x}_i)$, where N is the number of support vectors (which is correlated with the size of the training set). Great efforts have been involved in optimizing each component of the described evaluation process, including creating efficient matrices for faster multiplication as in Fastfood. In contrast, in DL, the classification process consists of sparse coding over a trained dictionary, where the dimension of the signals plays a considerable role in the complexity of the algorithm. While in SVM one can afford to use larger sized features (since evaluation amounts to a simple inner product between two vectors), in DL and sparse coding-based classification, this becomes a critical matter.

Furthermore, a recent analysis [54] of Nyström features versus Random Fourier ones shows that Nyström approximation, being data-dependent, captures the spectrum of the actual data distribution faster (smaller approximation dimension) than randomized feature approaches, especially in natural signals, where the spectrum of the kernel matrix decays rapidly. As a final note we add that the mentioned randomized features suit shift-invariant kernels only, while our method has complete freedom in choosing the kernel function.

G. Relation to Past Work

The existing works on kernel sparse representations can be roughly divided to two categories. The first corresponds to 'analytical' methods that operate solely in the feature domain and use the kernel trick to find an analytical solution, be it sparse coding or dictionary update [24], [25], [29], [32]. The other category refers to 'empirical' or 'approximal' methods that operate in the input space, while making some approximation or assumption regarding the mapped signals in feature space, in order to alleviate some of the constraints when working with kernels [22], [23], [26]. Naturally, our work belongs to the second group of contributions.

In 2002, Vincent and Bengio [22] kernelized the matching pursuit algorithm by using the kernel empirical map of the input training examples as dictionary atoms. By referring to the kernel empirical map Φ_e instead of the actual mapped signals in $\mathcal{F}$, the authors could perform standard linear matching pursuit without having to rewrite the algorithm in terms of inner products. In this case, the constraint of a PSD kernel was no longer mandatory. In 2005 [23], a similar concept of embedding the signals to a kernel empirical map was used to kernelize the basis pursuit algorithm. This approach of working in the input domain with an approximation of the kernel feature space is very similar to ours and can be described by the following embedding, evaluated over the entire training dataset $\{\mathbf{x}_i\}_{i=1}^{N}$:

$$\mathbf{x} \rightarrow \Phi_e(\mathbf{x}) = \left[\kappa(\mathbf{x}_1, \mathbf{x}), \ldots, \kappa(\mathbf{x}_N, \mathbf{x})\right]^T . \tag{25}$$

The case in our algorithm, where all the training signals are involved in the approximation of the kernel matrix ($c = N, \mathbf{C} = \mathbf{W} = \mathbf{K}$), results in a similar expression for the virtual samples (where we have used $\mathbf{K}^{\dagger} = \mathbf{V}\mathbf{\Sigma}^{\dagger}\mathbf{V}^T$):

$$\mathbf{F} = (\mathbf{\Sigma}^{1/2})^{\dagger}\mathbf{V}^T\mathbf{C}^T = (\mathbf{\Sigma}^{1/2})^{\dagger}\mathbf{V}^T\mathbf{K}^T, \tag{26}$$

where $\mathbf{\Sigma}$ and $\mathbf{V}$ are the eigenvalues and eigenvectors of the matrix $\mathbf{K}$. The embedding in this case is thus

$$\Phi_e(\mathbf{x}) = (\mathbf{\Sigma}^{1/2})^{\dagger}\mathbf{V}^T \left[\kappa(\mathbf{x}_1, \mathbf{x}), \ldots, \kappa(\mathbf{x}_N, \mathbf{x})\right]^T . \tag{27}$$

Contrary to [22], [23], our embedding preserves the similarities in the high-dimensional feature space, represented by the inner products, i.e,

$$\Phi_e(\mathbf{x})^T \Phi_e(\mathbf{x}') \approx \kappa(\mathbf{x}, \mathbf{x}') = \Phi(\mathbf{x})^T \Phi(\mathbf{x}'), \tag{28}$$

In addition, both [22] and [23] focus on sparse coding only and do not address the accuracy of the kernel empirical map, nor its dimension, which can be highly restrictive in large-scale datasets.

Both Gao *et al.* in 2010 [24] and Li *et al.* in 2011 [25], proposed an analytical approach of kernelizing the basis pursuit and orthogonal matching pursuit algorithms. Contrary to [22] and [23], the authors replaced all the inner products by kernels and worked entirely in the feature domain. Classification of faces and objects were achieved in [24] using a similar approach as in the SRC algorithm [11]. Aside from kernelizing the SRC algorithm, [24] also suggested updating the dictionary one atom at a time. By zeroing the derivative of the optimization function with respect to each atom, the authors acquired in the same term, a mixture of both the atom itself and its kernel with the input examples. As the resulting equation could not be solved analytically, an iterative fixed point update was implemented.

In 2012 Zhang *et al.* [26] provided an alternate approach of kernelizing the SRC algorithm. Instead of working with the implicit mapped signals in the feature space $\Phi(\mathbf{x})$, the authors performed dimensionality reduction first, using the KPCA algorithm, then fed the resulting nonlinear features to a linear l_1 basis pursuit solver. It can be shown that kernel PCA eventually entails the eigendecomposition of the kernel matrix (more accurately, the centered kernel matrix), as does our algorithm. The difference is that our method, apart from providing an accurate kernel mapping which aims to preserve similarities in feature space, also avoids dealing with the kernel matrix altogether in the training stage, making it possible to work with large datasets.

Instead of using a constant dimensionality reduction (DR) matrix, such as KPCA, another approach is learning the matrix as part of a sparse representation learning algorithm [55], [56]. In [55], the DR matrix is incorporated in the optimization of the SRC algorithm, leading to comparable accuracy with existing features, with smaller dimension. A kernelized version of [55], called "Sparse Embedding", is proposed in [56]. The authors learn a kernel DR matrix/embedding in conjunction with sparse representations and DL. The resulting features are then fed to a linear KSVD and provide improved classification accuracies, competitive with KDL.

This approach is similar to ours, since after performing dimensionality reduction, the authors are no longer restricted by the number of training samples N, in the sparse coding process, i.e. they perform regular linear OMP and KSVD. Nonetheless, the embedding depends on the dictionary and sparse representations, thus requires update in each iteration of DL. This update entails the storage of the sometimes *huge* kernel matrix $\mathbf{K}$, in addition to the eigen-decomposition of an exact sized matrix. Our method circumvents the use in the kernel matrix all-together by computing the virtual samples. In addition, the work in [56] provides a tailored embedding for the specific case of reconstructive DL (i.e. MOD or KSVD). Our method provides a broader, general solution for utilizing the nonlinearities in the input data. Finally this method cannot be scaled to deal with hundreds of thousands of input examples, and since the embedding requires the computation of the eigen-decomposition of a full rank $N \times N$ matrix, it cannot be performed online or in mini-batches.

V. Experimental Results

The following section highlights the three main benefits of incorporating LKDL with existing DL: (1) improvement in discriminability, which results in better classification (2) a small added computational effort by LKDL in comparison with typical kernel methods and (3) the ability to incorporate LKDL seamlessly in virtually any linear DL algorithm.

In these series of experimentations, all parameters were chosen using a combination of educated guessing and random or grid search. The choice of kernel parameters was fairly straightforward. Most datasets (pre-processed by l_2 unit normalization) reacted well to the Gaussian kernel with σ values of $[0.5, 1, 2]$ or Polynomial kernel of degree 2-4. The exact hyper-parameter of the kernel was determined using cross-validation. As for the Nyström approximation parameters, k was usually chosen as the signal's original dimension. This choice is rather arbitrary and suits to compare the performance of linear DL and LKDL. Finally the ratio of sampled signals in Nyström approximation, c/N, was chosen between $10-20\%$ in larger databases, such as USPS, MNIST and $50-100\%$ in smaller databases such as YaleB and AR-Face. We have seen that for a fixed value of k, c does not have an effect on the overall classification result (see Fig. 1b), thus choosing a smaller number of samples: $1-10\%$ is also possible (when there are sufficient training examples).

A. Unsupervised Dictionary Learning

In this part we demonstrate the performance of our algorithm in digit classification on the USPS and MNIST databases. Our method of classification consists of first preprocessing the training and test data using LKDL, then performing standard DL, using existing tools and finally deploying the classification scheme in section IV-D. For sparse coding and DL, we use the OMP and KSVD implementations from the latest OMP-Box (v10) and KSVD-Box (v13) libraries[3] [57]. During all experiments we use the KKSVD algorithm explained in section III-B [29], [31] as our reference, in addition to linear KSVD. We use the code of Nguyen's KKSVD[4]. A fair comparison in accuracy and runtime, between LKDL and KKSVD can be made, as KKSVD uses the same functions from the OMP and KSVD libraries mentioned earlier. The k-means[5] and coreset[6] sampling techniques were also adopted from existing code. All of the tests were performed on a 64-Bit Windows7 Intel(R) Core(TM) i7-4790K CPU with 16GB memory. The initial dictionary is a random subset of m columns from the training set in each class.

A note about over-completeness of the dictionary: In the following experiments, most of the dictionaries with which we represent the virtual samples are "compact", i.e. not over-complete, even with respect to the finite dimensions of the features we produce. Yet, this should pose no problem, because the goal of these dictionaries is not to fully represent the signal in reconstructive tasks, but rather serve classification needs.

1) USPS dataset: The USPS dataset consists of 7,291 training and 2,007 test images of digits of size 16×16. All images are stacked as vectors of dimension $p = 256$, mean extracted and normalized to unit l_2 norm. Following the experiment in [31], we choose the following parameters: 300 dictionary atoms per class, cardinality of 5 and 5 iterations of DL. The chosen kernel is polynomial of order 2, i.e. $\kappa(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T\mathbf{x}')^2$. The approximation parameters were set to: $c = 20\%$ of N training samples and $k = 256$, the original dimension of the digits. The displayed results are an average of 10 repeated iterations with different initialization of the sub-dictionaries and different sampled columns $\mathbf{X}_R$ in Nyström's method.

First we evaluate the quality of the representation of the kernel matrix using Nyström's method. We randomly choose 2,000 samples and approximate the resulting kernel matrix. In order to isolate the effect of column sub-sampling, we do not perform additional dimensionality reduction using eigen-decomposition and thus choose $k = 256$. Five sampling techniques were examined: uniform [48], diagonal [50], column-norm [49], k-means [52] and coreset [53]. We also added the ideal reconstruction using rank-c SVD decomposition, which is optimal with respect to minimizing the approximation error, but takes much longer time to compute. We perform the

[3]Found in http://www.cs.technion.ac.il/~ronrubin/software.html

[4]Found in http://www.umiacs.umd.edu/~hien/KKSVD.zip

[5]K-means - http://www.mathworks.com/matlabcentral/fileexchange/31274-fast-k-means/content/fkmeans.m

[6]Coreset - http://web.media.mit.edu/~michaf/index.html

comparison using the normalized approximation error:

$$err = \frac{\|\mathbf{K} - \widetilde{\mathbf{K}}\|_F}{\|\mathbf{K}\|_F}, \tag{29}$$

where $\mathbf{K}$ is the original kernel matrix and $\widetilde{\mathbf{K}}$ its Nyström approximation. Fig. 1a shows the quality of the approximation versus the c/N ratio, the percent of samples chosen for the Nyström approximation. As expected, SVD performs the best, as it is meant exactly for the purpose of providing the ideal rank-c approximation of $\mathbf{K}$. The second best approximation is obtained by k-means, which provides 97% accuracy in terms of the normalized approximation error, with only 10% of the samples. All other methods perform roughly the same. The differences in approximation quality reduce as the percent of chosen samples grows to half of the input dataset.

Next we examine the effect of sub-sampling on the classification accuracy of the entire database of USPS. Fig. 1b shows the classification accuracy as a function of c/N, along with the constant results of linear KSVD and KKSVD. There is a gap of 0.5% between the results of linear KSVD and its kernel variants, which suggests that kernelization improves the discriminability of the input signals. It can be seen that most sampling strategies give roughly the same results, competitive with KKSVD, with only a fraction of the samples. In general, the percent of samples in Nyström approximation does not have much impact on the final classification accuracy (apart from small fluctuations that arise from the randomness of each run). This can be explained by the simplicity of the digit images and the relatively large number of training examples.

Following Nguyen's setup in [29] and [31], we inspect the effect of corrupting the test images with white Gaussian noise and missing pixels. We use the same parameters as before and repeat the experiment 10 times with random corruptions. The results of classification accuracy versus the standard deviation of the noise and the percent of missing pixels are given in Fig. 2a and 2b. It is evident that adding the kernel improves the robustness of the database to both noise and missing pixels. The performance of LKDL follows that of KKSVD with a only 20% of the training samples. The trend shown in our results is similar to that in [31], although the results are slightly lower. This can be explained by the fact that in [31], the authors did not use the traditional partitioning of training and test data of the USPS dataset. The only sampling method shown is "coresets", due to the fact that it performed best.

2) MNIST dataset: Next we demonstrate the differences in runtime between our method and KKSVD using the larger-scale digit database of MNIST, which consists of 60,000 training and 10,000 test images of digits of size 28×28. Same as before, the digits were stacked in vectors of dimension $p = 784$ and normalized to unit l_2 norm. We examine the influence of gradually increasing the training set on the classification accuracy and training time of the input data. In this simulation, the entire training set of 60,000 examples is reduced by randomly choosing a defined fraction of the samples, while maintaining the test set untouched. The training runtime measured in LKDL includes the time needed to prepare the virtual train samples, along with training the entire input dataset using linear KSVD. The test runtime includes the time needed to compute the virtual test samples and the actual evaluation of all 10,000 test examples. As for KKSVD, the runtime includes the preparation of the kernel sub-matrices for each class and the kernel DL using KKSVD. Parameters in the simulation were: 2 DL iterations[7], cardinality of 11, 700 atoms per digit, polynomial kernel of order 2, $c = 15\%$ and $k = 784$. The results were averaged over 5 runs.

TABLE II
CLASSIFICATION PERFORMANCE OF RLS-DLA WITH AND WITHOUT LKDL ON THE EXTENDED MNIST. TIME MEASURED IN SECONDS.

Algorithm	Accuracy	Training	Test	Virt. Samples
RLS-DLA	97.49	837.63	6.42	
RLS-DLA+LKDL	98.4	846.59	6.47	703.13

The results can be seen in Fig. 3a-3c. Again, the coreset sampling method was chosen, as it provided the best results. The accuracy of LKDL is competitive with KKSVD and at times even better. Moreover, LKDL is 35-times faster in training, and 445-times faster in evaluating the entire database of MNIST. The training and test runtimes of LKDL follow the ones of KSVD, along with a component of calculating the virtual datasets. This is expected since our method "piggy-backs" on KSVD's performance and complexity. KKSVD's performance however, is dependent quadratically on the number of input samples in each class. When the database is large, the calculation of the virtual datasets (which is performed only once), is negligible versus the alternative of performing kernel sparse coding thousands of times during the DL process.

Next we show that our algorithm can work with an even larger input dataset. We use an enlarged version of MNIST where each digit's image is translated by one pixel in each direction (including diagonal). The result is a training set of 540,000 images and the original test set of 10,000 images. We randomly permute the train set and divide it to 9 batches of size 60,000 examples each. Now we perform RLS-DLA [36][8] with and without batch-LKDL, with the same parameters of the original MNIST experiment. The results in Table II show an almost 1% improvement in accuracy over linear RLS-DLA.

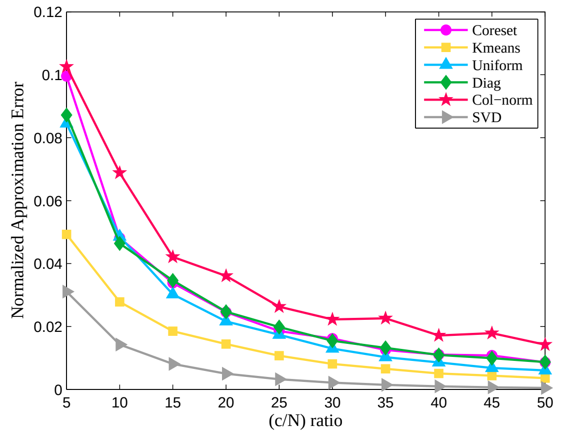
B. Comparison with random kernel empirical maps

In this section we test the effect of extracting different kernelized features. We compare our data-dependent method (LKDL) with two data-independent randomized kernel features: Random Fourier Features [38] (RFF) and Fastfood [39]. The datasets we use are the USPS and MNIST, combined with the Gaussian kernel with $\sigma = 1$. This kernel choice[9] is necessary since the RFF and Fastfood methods rely on the kernel being shift invariant. In this experiment we record the accuracy versus the dimension of the kernelized feature. Once the features are created they are combined with standard KSVD DL, exactly as in the previous sections (including

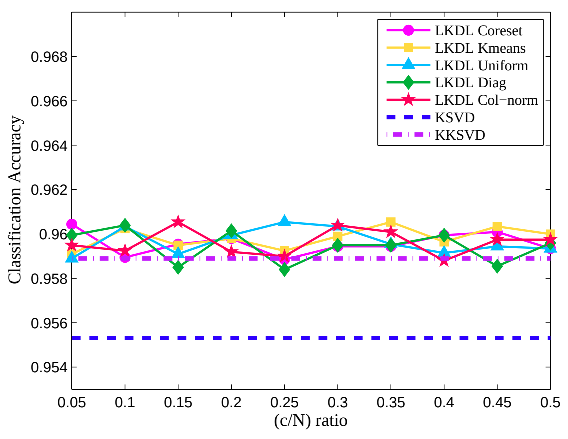
[7]Note that we chose a relatively small number of DL iterations in order to reduce the already-long computation time of KKSVD. A larger number of DL iterations will lead to an even greater difference in runtime between KKSVD and LKDL.

[8]Code can be found in: http://www.ux.uis.no/~karlsk/dle/index.html

[9]Note that this is not the optimal choice for LKDL, but nonetheless we use it, due to its shift-invariant structure, in order to maintain a fair comparison.

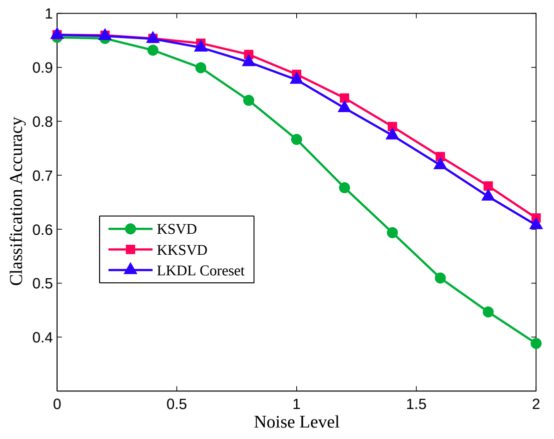


(a)

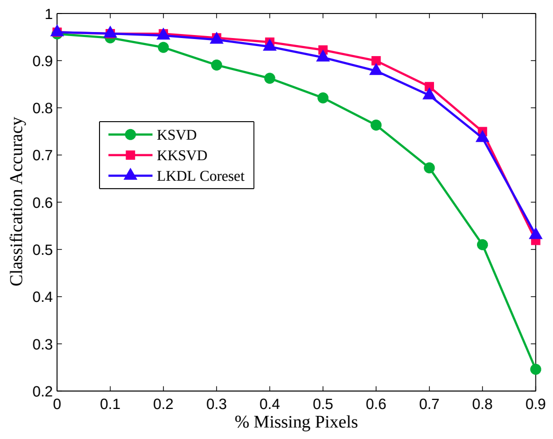


(b)

Fig. 1. Approximation error (a) and classification accuracy (b) as a function of c/N, percent of samples used in Nyström method.

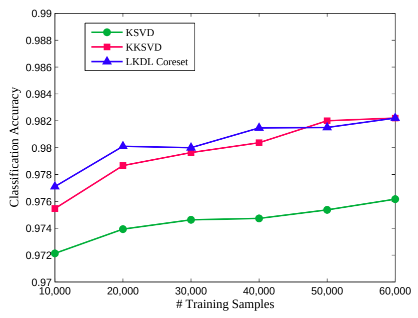


(a)

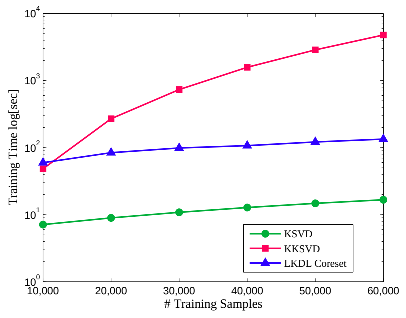


(b)

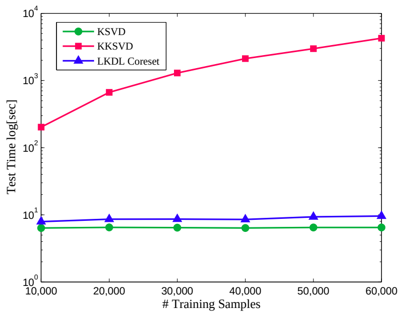
Fig. 2. Classification accuracy in the presence of Gaussian noise (a) and missing pixels (b).



(a)



(b)



(c)

Fig. 3. Accuracy (a) training time (b) and test time (c) versus the number of input training examples in MNIST. Runtime is shown in logarithmic scale.

the DL parameters: number of atoms, sparsity and number of iterations). We use the original code by the creators of RFF[10] and Fastfood[11]. In case of LKDL, we fix the number of examples in Nyström's approximation to be 20% (of the

[10]Found in: https://keysduplicated.com/~ali/random-features/

[11]Found in: http://www.mathworks.com/matlabcentral/fileexchange/49142-fastfood-kernel-expansions

entire number of available training data) in USPS and 3% in MNIST, and change the final dimension of the virtual samples, portrayed by the parameter k. The sampling method we use is K-means sampling. In RFF we sample $\frac{k}{2}$, p-dimensional random vectors $\boldsymbol{\omega}_i$, in order to create k-dimensional features. In Fastfood, we first create a larger sized, power of 2, random matrix[12], then extract $\frac{k}{2}$ rows from it at random, from which the k-dimensional final features are created exactly as in RFF. Finally we add for reference the results of linear KSVD (without kernelized features) and KKSVD. The shown accuracies are the average result of 20 independent runs.

As can be seen in Fig. 4a and 4b, LKDL features are more accurate than the randomized ones. While randomized features require larger dimensional vectors for higher accuracy, our method can manage with a smaller dimension. This is especially important for the task of DL. However, it can be seen that for larger signal dimensions, all features provide similar results. This suggests that randomized features are of important value in some applications and their relation to DL is definitely worth further investigation.

C. *Supervised Dictionary Learning*

In the following set of experiments we demonstrate the easiness of combining our pre-processing stage with any DL algorithm, in particular the LC-KSVD [15] and FDDL [17], both of which are supervised DL techniques that were mentioned earlier. We do so using the original code of LC-KSVD[13] and FDDL[14]. Throughout all tests, the training and test sets were pre-processed using LKDL to produce virtual training and test sets, which were later on fed as input to the DL and classification stages of each method. In all experiments, no code has been modified, except for exterior parameters which can be tuned to provide better results. The point in this setup is using an existing technique of supervised DL and showing the improvement that our method can provide.

1) Evaluation on the USPS Database: We start with comparing the classification accuracy of USPS, same as before. First we perform regular FDDL with the following parameters: 5 DL iterations, 300 dictionary atoms per class, where the dictionary is first initialized using K-means clustering of the training examples. The scalars controlling the tradeoff in the DL and optimization expressions remained the same as in [17]: $\lambda_1 = 0.1, \lambda_2 = 0.001$ and $g_1 = 0.1, g_2 = 0.005$ (in [17], these are referred to as γ_1, γ_2). As for LKDL preprocessing, the chosen parameters were: Polynomial kernel of degree 2, K-means based sub-sampling of 20% $(c/N = 0.2)$ and $k = 256$. All results were averaged over 10 iterations with different initializations. Table III shows classification results with and without LKDL. The results clearly improve when adding LKDL as pre-processing. However the obtained results in this experiment are lower than those reported in [17]. This can be explained by the fact that we used the original database

TABLE III
CLASSIFICATION ACCURACY OF FDDL ON THE USPS DIGIT DATABASE, WITH AND WITHOUT LKDL PRE-PROCESSING

Algorithm	Accuracy
FDDL	95.79
FDDL + LKDL	**96.049**

TABLE IV
CLASSIFICATION ACCURACY OF LC-KSVD1 AND LC-KSVD2 ON THE YALEB AND AR-FACE DATABASES, WITH AND WITHOUT LKDL

Algorithm	Yale-B	AR-Face
LC-KSVD1	94.49	92.5
LC-KSVD1 + LKDL	**96.08**	**94.8**
LC-KSVD2	94.99	93.7
LC-KSVD2 + LKDL	**96.58**	**94.8**

of USPS, while the provided code had a demo intended for an extended translation-invariant version of USPS. In addition, the exterior parameters $\lambda_1, \lambda_2, g_1, g_2$ were tweaked especially for the extended USPS, thus may have provided worse results in our case.

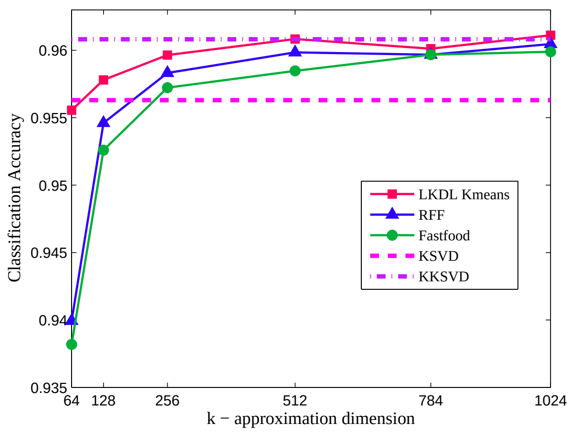
2) Evaluation on the Extended YaleB Database: Next, we show the benefit of combining our method with LC-KSVD on the "Extended YaleB" face recognition database, which consists of 2,414 frontal images that were taken under varying lighting conditions. There are 38 classes in YaleB and each class roughly contains 64 images, which are split in half to training and test sets, following the experiment described in [15]. The original 192×168 images are projected to 504-dimensional vectors using a randomly generated constant matrix from a zero-mean normal distribution. We use a dictionary size of 570 (in average 15 images per class) and sparsity factor of 30, same as in [15]. The kernel chosen for LKDL was Gaussian of the form: $\kappa(\mathbf{x}, \mathbf{x}') = \exp\left(-\|\mathbf{x} - \mathbf{x}'\|_2^2/2\sigma^2\right)$, where $\sigma = 2$. Due to the small size of the dataset, no subsampling was performed and c was set to be the entire size of the training set. The value of the parameter k was set to 504, the initial dimension of the signals. In order to use the Gaussian kernel, the samples in the training and test sets were l_2 normalized, thus the original parameters in [15], [16] of $\sqrt{\alpha}$ and $\sqrt{\beta}$ in expression (9) had to be changed. The final parameters: $\sqrt{\alpha} = 1/200$ in case of LCKSVD1, and $\sqrt{\alpha} = 1/600, \sqrt{\beta} = 1/900$ in case of LCKSVD2, were chosen using a coarse-to-fine grid search and provided the best classification results. We use the original classification scheme in [15], [16]. Table IV shows classification results of LC-KSVD1 and LC-KSVD2, with and without LKDL. It is clear that the addition of the nonlinearity increases the discriminability of the input samples and improves classification results by up to 1.6% in both LC-KSVD1 and LC-KSVD2.

3) Evaluation on the AR Face Database: The AR Face database consists of 4,000 color images of faces belonging to 126 classes. Each class consists of images taken over two sessions, containing different lighting conditions, facial variations and facial disguises. Following the experiment in [16], 2,600 images were chosen, first 50 classes of males and first 50 classes of females. Out of 26 images in each class, 20 were chosen for training and the rest for evaluation.

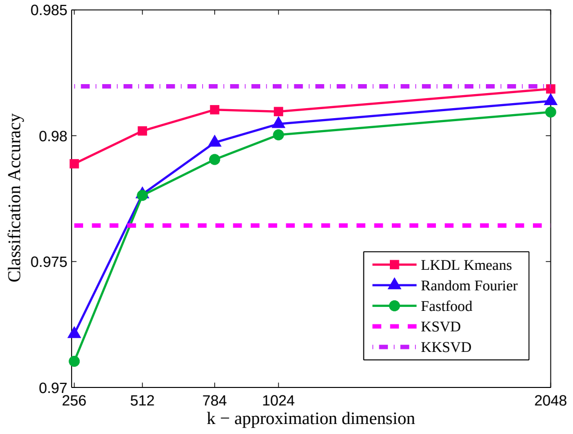
[12]Fastfood relies on Hadamard matrices which are of size that divides by the power of 2, thus to get a random sized feature, we truncate the final Gaussian random matrix by taking a subset of its rows

[13]Found in http://www.umiacs.umd.edu/~zhuolin/LCKSVD/

[14]Found in http://www.vision.ee.ethz.ch/~yangme/database_mat/FDDL.zip



(a)



(b)

Fig. 4. Classification accuracy versus feature dimension of USPS (a) and MNIST (b) databases.

We use the already-processed dataset[15] in [16], where the original images of size 165×120 pixels were reduced to 540-dimensional vectors using random projection as in Extended YaleB. The cardinality is same as before set to 30 and the number of atoms in DL is set to 500 (5 atoms per class). As before, we normalized all the signals to unit l_2-norm. The parameters σ of the Gaussian kernel, $\sqrt{\alpha}$ and $\sqrt{\beta}$ have been determined using a coarse-to-fine random search strategy. We chose $\sigma \in [0.25, 0.5, 0.75, 1, 1.25]$ and $\alpha, \beta \in 1/[1..100]$ and ran the search 100 times. Eventually a local maxima has been found at: $\sigma = 0.5$. The optimal parameter for LCKSVD1 was $\sqrt{\alpha} = 1/14$, whereas the ones for LCKSVD2 were: $\sqrt{\alpha} = 1/15$ and $\sqrt{\beta} = 1/17$. The parameter c was set to the size of the entire database, i.e. no sub-sampling was performed and k was set to the original dimension of the data, 540. In table IV we compare the classification results of LC-KSVD1 and LC-KSVD2, with and without LKDL pre-processing. As can be seen our method improves the performance of LC-KSVD1 by 2.3% and LC-KSVD2 by 1.1%.

VI. Conclusion

In this paper we have discussed some of the problems arising when trying to incorporate kernels in DL, and payed special attention to the kernel-KSVD algorithm by Nguyen *et al.* [29], [31]. We proposed a novel kernel DL scheme, called "LKDL", which acts as a kernelizing pre-processing stage, before performing standard DL. We used the concept of virtual training and test sets and described the different aspects of calculating these signals. We demonstrated in several experiments on different datasets the benefits of combining our LKDL pre-processing stage, both in accuracy of classification and in runtime. Lastly, we have shown the easiness of integrating our method with existing supervised and unsupervised DL algorithms. It is our hope that the proposed methodology will encourage users to consider kernel DL for their tasks, knowing that the extra-effort involved in incorporating the kernel layer is near-trivial. We intend to freely share the code that reproduces all the results shown in this paper.

Our future research includes combining LKDL with complicated signals that do not adhere to Euclidean distances, for example region covariance matrices. We would also like to examine the benefit of applying LKDL to the sparse coefficients instead of the input signals and maybe combining both options. Lastly, our goal is improving the sampling ratio and the size of the matrix $\mathbf{W}$, using advanced sampling techniques, and maybe combining Nyström data-dependent features with randomized ones, in order to enjoy both worlds.

[15]Found in http://www.umiacs.umd.edu/~zhuolin/LCKSVD/

References

[1] S. Mallat, *A wavelet tour of signal processing*. Academic Press, 1999.

[2] E. J. Candes and D. L. Donoho, "Recovering edges in ill-posed inverse problems: Optimality of curvelet frames," *Ann. Statist.*, vol. 30, no. 3, pp. 784–842, Jun. 2002.

[3] M. N. Do and M. Vetterli, "Contourlets: a directional multiresolution image representation," *Proc. IEEE Int. Conf. Image Process. (ICIP)*, 2002.

[4] M. Elad and M. Aharon, "Image denoising via sparse and redundant representations over learned dictionaries," *IEEE Trans. Image Process.*, vol. 15, no. 12, pp. 3736–3745, Dec. 2006.

[5] J. M. Fadili, J. L. Starck, and F. Murtagh, "Inpainting and zooming using sparse representations," *J. Comput.*, vol. 52, no. 1, pp. 64–79, 2007.

[6] J. Mairal, M. Elad, and G. Sapiro, "Sparse representation for color image restoration," *IEEE Trans. Image Process.*, vol. 17, no. 1, pp. 53–69, Jan. 2008.

[7] O. Bryt and M. Elad, "Compression of facial images using the K-SVD algorithm," *J. Visual Commun. Image Representation*, vol. 19, no. 4, pp. 270–282, May 2008.

[8] J. Zepeda, C. Guillemot, and E. Kijak, "Image compression using sparse representations and the iteration-tuned and aligned dictionary," *IEEE J. Sel. Top. Signal Process.*, vol. 5, no. 5, pp. 1061–1073, Sep. 2011.

[9] K. Engan, S. Aase, O. Hakon, and J. Husoy, "Method of optimal directions for frame design," *IEEE Int. Conf. Acoustics, Speech, and Signal Process. (ICASSP)*, vol. 5, pp. 2443–2446, 1999.

[10] M. Aharon, M. Elad, and A. Bruckstein, "K-SVD an algorithm for designing overcomplete dictionaries for sparse representations," *IEEE Trans. Signal Process.*, vol. 54, no. 11, pp. 4311–4322, Nov. 2006.

[11] J. Wright, A. Y. Yang, A. Ganesh, S. S. Sastry, and Y. Ma, "Robust face recognition via sparse representation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 31, no. 2, pp. 210–227, Feb. 2009.

[12] J. Mairal, J. Ponce, G. Sapiro, A. Zisserman, and F. R. Bach, "Supervised dictionary learning," *Advanc. Neural Inform. Process. Syst. (NIPS)*, pp. 1033–1040, 2009.

[13] J. Mairal, F. R. Bach, and J. Ponce, "Task driven dictionary learning," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 4, pp. 791–804, Apr. 2012.

[14] Q. Zhang and B. Li, "Discriminative K-SVD for dictionary learning in face recognition," *IEEE conf. Comput. Vision Pattern Recog. (CVPR)*, pp. 2691–2698, 2010.

[15] Z. Jiang, Z. Lin, and L. S. Davis, "Learning a discriminative dictionary for sparse coding via label consistentlabel consistent K-SVD," *IEEE Conf. Comput. Vision Pattern Recog. (CVPR)*, pp. 1697–1704, 2011.

[16] ——, "Label consistent K-SVD: Learning a discriminative dictionary for recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 11, pp. 2651–2664, 2013.

[17] M. Yang, L. Zhang, X. Feng, and D. Zhang, "Fisher discrimination dictionary kearning for sparse representation," *IEEE Int. Conf. Comput. Vision (ICCV)*, pp. 543–550, 2011.

[18] S. Cai, W. Zuo, L. Zhang, X. Feng, and P. Wang, "Support vector guided dictionary learning," pp. 624–639, 2014.

[19] V. Vapnik, *The nature of statistical learning theory*. Springer, 2000.

[20] B. Scholköpf, A. Smola, and K. R. Müller, "Kernel principal component analysis," *Artific. Neural Networks ICANN*, pp. 583–588, 1997.

[21] B. Scholköpf and K. R. Müller, "Fisher discriminant analysis with kernels," *Proc. IEEE Signal Process. Soc. Workshop Neural Networks for Signal Process.*, pp. 23–25, 1999.

[22] P. Vincent and Y. Bengio, "Kernel matching pursuit," *Mach. Learn.*, vol. 48, no. 1–3, pp. 165–187, 2002.

[23] V. Guigue, A. Rakotomamonjy, and S. Canu, "Kernel basis pursuit," *Mach. Learn. (ECML) 2005*, pp. 146–157, 2005.

[24] S. Gao, I. W. H. Tsang, and L. T. Chia, "Kernel sparse representation for image classification and face recognition," *Comput. Vision – ECCV*, pp. 1–14, 2010.

[25] H. Li, Y. Gao, and J. Sun, "Fast kernel sparse representation," *IEEE Int. Conf. Digital Image Comput. Techniques and Applicat. (DICTA)*, pp. 72–77, 2011.

[26] L. Zhang, W. D. Zhou, P. C. Chang, J. Liu, T. Wang, and F. Z. Li, "Kernel sparse representation-based classifier," *IEEE Trans. Signal Process.*, vol. 60, no. 4, pp. 1684–1695, Apr. 2012.

[27] M. Jian and C. Jung, "Class-discriminative kernel sparse representation-based classification using multi-objective optimization," *IEEE Trans. Signal Process.*, vol. 61, no. 18, pp. 4416–4427, Sep. 2013.

[28] Y. Zhou, K. Liu, R. E. Carrillo, K. E. Barner, and F. Kiamilev, "Kernel-based sparse representation for gesture recognition," *Pattern Recog.*, vol. 46, no. 12, pp. 3208–3222, Dec. 2013.

[29] H. V. Nguyen, V. M. Patel, N. M. Nasrabadi, and R. Chellappa, "Kernel dictionary learning," *IEEE Int. Conf. Acoustics, Speech, and Signal Process. (ICASSP)*, pp. 2021–2024, 2012.

[30] M. T. Harandi, C. Sanderson, R. Hartley, and B. C. Lovell, "Sparse coding and dictionary learning for symmetric positive definite matrices: A kernel approach," *Comput. Vision ECCV 2012*, pp. 216–229, 2012.

[31] H. V. Nguyen, V. M. Patel, N. M. Nasrabadi, and R. Chellappa, "Design of non-linear kernel dictionaries for object recognition," *IEEE Trans. Image Process.*, vol. 22, no. 12, pp. 5123–5135, Dec. 2013.

[32] M. J. Gangeh, A. Ghodsi, and M. S. Kamel, "Kernelized supervised dictionary learning," *IEEE Trans. Signal Process.*, vol. 61, no. 19, pp. 4753–4767, Oct. 2013.

[33] A. Shrivastava, H. V. Nguyen, V. M. Patel, and R. Chellappa, "Design of non-linear discriminative dictionaries for image classification," *Comput. Vision-ACCV 2012*, pp. 660–674, 2012.

[34] Z. Chen, W. Zuo, Q. Hu, and L. Lin, "Kernel sparse representation for time series classification," *Inform. Sci.*, vol. 292, pp. 15–26, 2015.

[35] J. Mairal, F. Bach, J. Ponce, and G. Sapiro, "Online dictionary learning for sparse coding," *Proc. Ann. Int. Conf. Mach. Learn. (ICML)*, pp. 689–696, 2009.

[36] K. Skretting and K. Engan, "Recursive least squares dictionary learning algorithm," *IEEE Trans. Signal Process.*, vol. 58, no. 4, pp. 2121–2130, Apr. 2010.

[37] K. Zhang, L. Lan, Z. Wang, and F. Moerchen, "Scaling up kernel SVM on limited resources: A low-rank linearization approach," *Int. Conf. Artificial Intell. and Stat.*, pp. 1425–1434, 2012.

[38] A. Rahimi and B. Recht, "Random features for large-scale kernel machines," pp. 1177–1184, 2007.

[39] Q. Le, T. Sarlos, and A. Smola, "Fastfood-approximating kernel expansions in loglinear time," 2013.

[40] S. S. Chen, D. L. Donoho, and M. A. Saunders, "Atomic decomposition by basis pursuit," *SIAM J. Sci. Comput.*, vol. 20, no. 1, pp. 33–61, 1998.

[41] S. G. Mallat and Z. Zhang, "Matching pursuits with time-frequency dictionaries," *IEEE Trans. Signal Process.*, vol. 41, no. 12, pp. 3397–3415, 1993.

[42] Y. C. Pati, R. Rezaiifar, and P. S. Krishnaprasad, "Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition," *IEEE Comput. Soc. Press*, pp. 40–44, 1993.

[43] R. Duda, P. Hart, and D. Stork, *Pattern Classification*. Wiley-Interscience, 2000.

[44] N. Aronszajn, "Theory of reproducing kernels," *Trans. Amer. Math. Soc.*, vol. 68, pp. 337–404, 1950.

[45] J. Mercer, "Functions of positive and negative type and their connection with the theory of integral equations," *Philos. Trans. Roy. Soc. London*, pp. 415–446, 1909.

[46] B. Scholköpf, S. Mika, C. J. Knirsch, K. R. Müller, G. Ratsch, and A. J. Smola, "Input space versus feature space in kernel-based methods," *IEEE Trans. Neural Netw.*, vol. 10, no. 5, pp. 1000–1017, 1999.

[47] H. Xiong, M. N. S. Swamy, and M. O. Ahmad, "Optimizing the kernel in the empirical feature space," *IEEE Trans. Neural Networks*, vol. 16, no. 2, pp. 460–474, Mar. 2005.

[48] C. Williams and M. Seeger, "Using the Nyström method to speed up kernel machines," *Proc. Annu. Conf. Neural Inform. Process. Syst. (NIPS)*, pp. 682–688, 2002.

[49] P. Drineas, R. Kannan, and M. W. Mahoney, "Fast monte carlo algorithms for matrices i: Approximating matrix multiplication," *SIAM J. Computing*, vol. 36, no. 1, pp. 132–157, Jan. 2006.

[50] P. Drineas and M. W. Mahoney, "On the Nyström method for approximating a gram matrix for improved kernel-based learning," *J. Mach. Learn. Research*, no. 6, pp. 2153–2175, 2005.

[51] S. Kumar, M. Mohri, and A. Talwalkar, "Sampling techniques for the Nyström method," *Int. Conf. Artific. Intell. and Stat.*, pp. 304–311, 2009.

[52] K. Zhang, I. W. Tsang, and J. T. Kwok, "Improved Nyström low-rank approximation and error analysis," *Proc. Int. Conf. Mach. Learn. (ACM)*, pp. 1232–1239, 2008.

[53] D. Feldman, M. Feigin, and N. Sochen, "Learning big (image) data via coresets for dictionaries," *J. Math. Imaging and Vision*, vol. 46, no. 3, pp. 276–291, Jul. 2013.

[54] T. Yang, Y.-F. Li, M. Mahdavi, R. Jin, and Z.-H. Zhou, "Nyström method vs random fourier features: A theoretical and empirical comparison," pp. 476–484, 2012.

[55] L. Zhang, M. Yang, Z. Feng, and D. Zhang, "On the dimensionality reduction for sparse representation based face recognition," *Int. Conf. Pattern Recog. (ICPR)*, pp. 1237–1240, 2010.

[56] H. V. Nguyen, V. M. Patel, N. M. Nasrabadi, and R. Chellappa, "Sparse embedding: A framework for sparsity promoting dimensionality reduction," pp. 414–427, 2012.

[57] R. Rubinstein, M. Zibulevsky, and M. Elad, "Efficient implementation of the K-SVD algorithm using batch orthogonal matching pursuit," *CS TECHNION*, vol. 40, 2008.

Alona Golts received her B.Sc. (2009) in Electrical Engineering and Physics, from the department of Electrical Engineering at the Technion, Israel, where she is currently pursuing her M.Sc degree. Alona has served in the Israeli Air Force from 2009 to 2015, under the reserve excellence program "Psagot".

Michael Elad received his B.Sc. (1986), M.Sc. (1988) and D.Sc. (1997) from the department of Electrical engineering at the Technion, Israel. Since 2003 he is a faculty member at the ComputerScience department at the Technion, and since 2010 he holds a full-professorship position. Michael Elad works in the field of signal and image processing, specializing in particular on inverse problems, sparse representations and superresolution. Michael received the Technions best lecturer award six times, he is the recipient of the 2007 Solomon Simon Mani award for excellence in teaching, the 2008 Henri Taub Prize for academic excellence, and the 2010 Hershel-Rich prize for innovation. Michael is an IEEE Fellow since 2012. He is serving as an associate editor for SIAM SIIMS, and ACHA. Michael is also serving as a senior editor for IEEE SPL.