Journal of Visual Communication and Image Representation 50 (2018) 237–246

Contents lists available at ScienceDirect

Journal of Visual Communication and Image Representation

journal homepage: www.elsevier.com/locate/jvci

On data-driven Saak transform☆

C.-C. Jay Kuo*, Yueru Chen

Ming-Hsieh Department of Electrical Engineering, University of Southern California, Los Angeles, CA 90089-2564, USA

ARTICLE INFO

Keywords:
Data-driven transform
RECOS transform
Saak transform
The Karhunen-Loéve transform (KLT)
Linear subspace approximation
Principal component analysis

ABSTRACT

Being motivated by the multilayer RECOS (REctified-COrrelations on a Sphere) transform, we develop a data-driven Saak (**S**ubspace **a**pproximation with **a**ugmented **k**ernels) transform in this work. The Saak transform consists of three steps: (1) building the optimal linear subspace approximation with orthonormal bases using the second-order statistics of input vectors, (2) augmenting each transform kernel with its negative, (3) applying the rectified linear unit (ReLU) to the transform output. The Karhunen-Loéve transform (KLT) is used in the first step. The integration of Steps 2 and 3 is powerful since they resolve the sign confusion problem, remove the rectification loss and allow a straightforward implementation of the inverse Saak transform at the same time. Multiple Saak transforms are cascaded to transform images of a larger size. All Saak transform kernels are derived from the second-order statistics of input random vectors in a one-pass feedforward manner. Neither data labels nor backpropagation is used in kernel determination. Multi-stage Saak transforms offer a family of joint spatial-spectral representations between two extremes; namely, the full spatial-domain representation and the full spectral-domain representation. We select Saak coefficients of higher discriminant power to form a feature vector for pattern recognition, and use the MNIST dataset classification problem as an illustrative example.

1. Introduction

Signal transforms provide a way to convert signals from one representation to another. For example, the Fourier transform maps a time-domain function into a set of Fourier coefficients. The latter representation indicates the projection of the time-domain function onto a set of orthonormal sinusoidal basis functions. The orthonormal basis facilitates the inverse transform. The original function can be synthesized by summing up all Fourier basis functions weighted by their Fourier coefficients. The basis functions (or transform kernels) are typically selected by humans. One exception is the Karhunen-Loéve transform (KLT) [1]. The KLT kernels are the unit eigenvectors of the covariance matrix of sampled data. It is the optimal transform in terms of energy compaction. That is, to obtain an approximation to an input signal class, we can truncate part of KLT basis functions associated with the smallest eigenvalues. The truncated KLT provides the optimal approximation to the input with the smallest mean-squared-error (MSE).

We develop new data-driven forward and inverse transforms in this work. For a set of images of size $N \times N$, the total number of variables in these images is N^2 and their covariance matrix is of dimension N^4. It is not practical to conduct the KLT on the full image for a large N. Instead, we may decompose images into smaller blocks and conduct the KLT on each block. To give an example, the Discrete Cosine Transform (DCT) [2] provides a good approximation to the KLT for image data, and the block DCT is widely used in the image/video compression standards. One question of interest is whether it is possible to generalize the KLT so that it can be applied to images of a larger size in a hierarchical fashion? Our second research motivation comes from the resurgent interest on convolutional neural networks (CNNs) [3,4]. The superior performance of CNNs has been demonstrated in many applications such as image classification, detection and processing. To offer an explanation, Kuo [5,6] modeled the convolutional operation at each CNN layer with the RECOS (REctified-COrrelations on a Sphere) transform, and interpreted the whole CNN as a multi-layer RECOS transform.

By following this line of thought, it would be a meaningful task to define the inverse RECOS transform and analyze its properties. The analysis of the forward/inverse RECOS transform will be conducted in Section 2. Being similar to the forward and inverse Fourier transforms, the forward and inverse RECOS transforms offer tools for signal analysis and synthesis, respectively. However, unlike the data-independent Fourier transform, the RECOS transform is derived from labeled training data, and its transform kernels (or filter weights) are optimized by backpropagation. The analysis of forward/inverse RECOS transforms is challenging due to nonlinearity. We will show that the RECOS transform has two loss terms: the approximation loss and the rectification loss. The approximation loss is caused by the use of a limited

☆ This paper has been recommended for acceptance by Zicheng Liu.

* Corresponding author.
E-mail address: cckuo@sipi.usc.edu (C.-C. Jay Kuo).

https://doi.org/10.1016/j.jvcir.2017.11.023
Received 12 October 2017; Accepted 30 November 2017
Available online 05 December 2017
1047-3203/ © 2017 Elsevier Inc. All rights reserved.

number of transform kernels. This error can be reduced by increasing the number of filters at the cost of higher computational complexity and higher storage memory. The rectification loss is due to nonlinear activation. Furthermore, since the filters in the RECOS transform are not orthogonal to each other, the inverse RECOS transform demands the solution of a linear system of equations.

It is stimulating to develop a new data-driven transform that has neither approximation loss nor the rectification loss as the RECOS transform. Besides, it has a set of orthonormal transform kernels so that its inverse transform can be performed in a straightforward manner. To achieve these objectives, we propose the Saak (**S**ubspace **a**pproximation with **a**ugmented **k**ernels) transform. As indicated by its name, the Saak transform has two main ingredients: (1) subspace approximation and (2) kernel augmentation. To seek the optimal subspace approximation to a set of random vectors, we analyze their second-order statistics and select orthonormal eigenvectors of the covariance matrix as transform kernels. This is the well-known KLT. When the dimension of the input space is very large (say, in the order of thousands or millions), it is difficult to conduct one-stage KLT. Then, we may decompose a high-dimensional vector into multiple lower-dimensional sub-vectors. This process can be repeated recursively to form a hierarchical representation. For example, we can decompose one image into four non-overlapping quadrants recursively to build a quad-tree whose leaf node is a small patch of size 2×2. Then, a KLT can be defined at each level of the quad-tree.

If two or more transforms are cascaded directly, there is a "sign confusion" problem [5,6]. To resolve it, we insert the Rectified Linear Unit (ReLU) activation function in between. The ReLU inevitably brings up the rectification loss, and a novel idea called kernel augmentation is proposed to eliminate this loss. That is, we augment each transform kernel with its negative vector, and use both original and augmented kernels in the Saak transform. When an input vector is projected onto the positive/negative kernel pair, one will go through the ReLU operation while the other will be blocked. This scheme greatly simplifies the signal representation problem in face of ReLU nonlinear activation. It also facilitates the inverse Saak transform. The integration of kernel augmentation and ReLU is equivalent to the sign-to-position (S/P) format conversion of the Saak transform outputs, which are called the Saak coefficients. By converting the KLT to the Saak transform stage by stage, we can cascade multiple Saak transforms to transform images of a large size. The multi-stage Saak transforms offer a family of joint spatial-spectral representations between two extremes – the full spatial-domain representation and the full spectral-domain representation. The Saak and multi-stage Saak transforms will be elaborated in Sections 3 and 4, respectively.

Although both CNNs and multi-stage Saak transforms adopt the ReLU, it is important to emphasize one fundamental difference in their filter weights (or transform kernels) determination. CNN's filter weights are determined by the training data and their associated labels. After initialization, these weights are updated via backpropagation. A CNN determines its optimal filter weights by optimizing a cost function via backpropagation iteratively. The iteration number is typically huge. The multi-stage Saak transforms adopt another fully automatic method in determining their transform kernels. They are selected based on the second-order statistics of input vectors at each stage. It is a one-pass feedforward process from the leaf to the root. Neither data labels nor backpropagation is needed for transform kernel determination.

The Saak coefficients in intermediate stages indicate the spectral component values in the corresponding covered spatial regions. The Saak coefficients in the last stage represent spectral component values of the entire input vector (i.e., the whole image). We use the MNIST dataset as an example to illustrate the distribution of Saak coefficients of each object class. Based on the ANalysis Of VAriance (ANOVA), we select Saak coefficients of higher discriminant power by computing their F-test score. The larger the F-test score, the higher the discriminant power. Finally, we compare the classification accuracy with the support vector machine (SVM) and the K-Nearest-Neighbors (KNN) classifiers.

The rest of this paper is organized as follows. The forward and inverse RECOS transforms are studied in Section 2. The forward and inverse Saak transforms are proposed in Section 3. The multi-stage Saak transforms are presented in Section 4. The application of multi-stage Saak transforms to image classification for the MNIST dataset is described in Section 5. The CNN approach and the Saak-transform-based machine learning methodology are compared in Section 6. Finally, concluding remarks are given and future research directions are pointed out in Section 7.

2. RECOS transform

The RECOS transform is a data-driven transform proposed in [5,6] to model the convolutional operations in a CNN. In the context of image processing, the forward RECOS transform defines a mapping from a real-valued function defined on a three-dimensional (3D) cuboid to a one-dimensional (1D) rectified spectral vector. The forward and inverse RECOS transforms will be studied in this section.

2.1. Forward RECOS transform

As illustrated in Fig. 1, a spatial-spectral cuboid, denoted by $C(i_p,j_p,L_i,L_j,L_k)$, consists of 3D grid points with indices (i,j,k)

- along the horizontal dimension: $i \in \{i_p, i_p+1,\ldots,i_p+L_i-1\}$,
- along the vertical dimension: $j \in \{j_p, j_p+1,\ldots,j_p+L_j-1\}$,
- along the spectral dimension: $k \in \{0,1,2,\ldots L_k-1\}$,

where $(i,j) = (i_p,j_p)$ is its spatial pivot and L_i,L_j and L_k are its width, height and depth, respectively. For a real-valued function defined on cuboid $C(i_p,j_p,L_i,L_j,L_k)$, one can flatten these values into a one-dimensional (1D) vector,

$$\mathbf{f} \in R^N, \quad \text{where} \quad N = L_i \times L_j \times L_k, \tag{1}$$

by scanning the 3D grid points with a fixed order. All vectors in R^N are generated by the same flattening rule.

Consider an anchor vector set [5,6] that contains L_k vectors of unit length,

$$A = \{\mathbf{a}_0,\mathbf{a}_1,\ldots,\mathbf{a}_k,\ldots,\mathbf{a}_K\}, \quad \|\mathbf{a}_k\| = 1 \text{ and } K = L_k-1, \tag{2}$$

where $\mathbf{a}_k$ is defined on $C(i_p,j_p,L_i,L_j,L_k)$ and flattened to a vector in R^N. We divide anchor vectors into two types. The vector

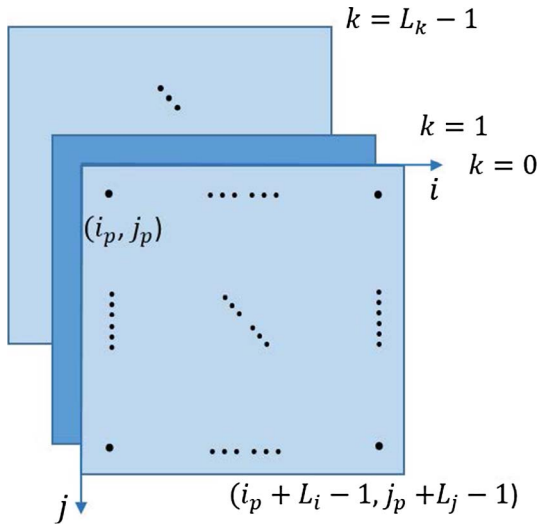


Fig. 1. Illustration of a cuboid with its pivot at $(i,j) = (i_p,j_p)$.

$$\mathbf{a}_0 = \frac{1}{\sqrt{N}}(1,1,...,1)^T \tag{3}$$

is the DC anchor vector while the remaining ones, $\mathbf{a}_1,...,\mathbf{a}_K$, are the AC anchor vectors. An effective RECOS transform depends on careful selection of AC anchor vectors. A common way to select anchor vectors is to train the CNN with labeled data via backpropagation. Given anchor vector set A in Eq. (2), the forward RECOS transform can be summarized as follows.

Forward RECOS Transform. The forward RECOS transform consists of two steps.

1. Project input $\mathbf{f} \in R^N$ onto $K+1$ anchor vectors to get $K+1$ projection values[1]:

$$p_k = \mathbf{a}_k^T \mathbf{f}, \quad k = 0,1,...K. \tag{4}$$

Since anchor vector $\mathbf{a}_k$ plays the transform kernel role as shown in Eq. (4), the two terms "anchor vectors" and "transform kernels" are used interchangeably. We call

$$\mathbf{p} = (p_0,p_1,...,p_K)^T \in R^{K+1} \tag{5}$$

the projection vector of input $\mathbf{f}$.

2. Preserve the DC projection (i.e., p_0) and pass AC projections (i.e., $p_1,...,p_K$) through the ReLU activation function. The output $\mathbf{g} \in R^{K+1}$ of the RECOS transform can be written as

$$\mathbf{g} = (g_0,g_1,...,g_K)^T, \tag{6}$$

where

$$g_0 = p_0, \tag{7}$$

and

$$g_k = \begin{cases} p_k, & \text{if } p_k > 0, \\ 0, & \text{if } p_k \leqslant 0. \end{cases} k = 1,2,...,K. \tag{8}$$

In above, we allow g_0 to be either positive or negative while g_k, $k = 1,...,K$, is non-negative. The DC projection contributes to the bias term. The forward RECOS transform defined above offers a mathematical description of convolutional operations in both convolutional and fully connection layers of CNNs.

2.2. Inverse RECOS transform

The inverse RECOS transform reconstructs input $\mathbf{f} \in R^N$ as closely as possible from its output $\mathbf{g} \in R^{K+1}$ of the forward RECOS transform. To proceed, we first examine the reconstruction of the input from the unrectified projection vector, $\mathbf{p}$, as given in Eq. (5). When $K+1 < N$, we cannot reconstruct input $\mathbf{f} \in R^N$ exactly but its approximation in the linear space spanned by vectors in A, i.e.

$$\mathbf{f} \approx \hat{\mathbf{f}} = \sum_{k=0}^{K} \alpha_k \mathbf{a}_k. \tag{9}$$

Because of Eqs. (4) and (9), we get

$$p_k \approx \mathbf{a}_k^T \hat{\mathbf{f}} = \mathbf{a}_k^T \left(\sum_{k'=0}^{K} \alpha_{k'} \mathbf{a}_{k'} \right), \quad k = 0,1,...,K.$$

If the approximation error $\|\hat{\mathbf{f}} - \mathbf{f}\|$ is negligible, we can set up a linear system of equations in form of

$$p_k = \sum_{k'=0}^{K} \mathbf{a}_k^T \mathbf{a}_{k'} \alpha_{k'}, \quad k = 0,1,...,K,$$

with unknowns . If the $(K+1) \times (K+1)$ coefficient matrix has the full rank, we can solve for uniquely.

Inverse RECOS Transform. All AC outputs of the RECOS transform are either positive or zero as given in Eq. (8). We can use a permutation matrix to re-arrange $g_1,...,g_K$ so that all zero elements are moved to the end of the re-ordered vector. Suppose that there are $1 \leqslant Q \leqslant K$ nonzero elements in $g_1,...,g_K$. After re-arrangement, we can express it as

$$\mathbf{g}' = (g_0,g_1',...,g_Q',0,...,0)^T, \tag{10}$$

which has $K-Q$ zero elements at the end. The re-arrangement of $g_1,...,g_K$ is the same as re-arranging AC anchor vectors, $\mathbf{a}_1,...,\mathbf{a}_K$. We use $\mathbf{a}_1',...,\mathbf{a}_Q'$ to denote anchor vectors associated with projections $p_1',...,p_Q'$. Then, we obtain

$$\mathbf{f}' = \sum_{q=0}^{Q} \beta_q \mathbf{a}_q', \tag{11}$$

where $\mathbf{a}_0' = \mathbf{a}_0$. Eqs. (9) and (11) are the reconstruction formulas of $\mathbf{f} \in R^N$ using unrectified and rectified projection vectors, respectively.

The inverse RECOS transform is a mapping from a $(Q+1)$-dimensional rectified vector $(p_0,p_1',...,p_Q')^T$ to an approximation of $\mathbf{f}$ in the $(Q+1)$-dimensional subspace as defined in Eq. (11). By following the same procedure given above, we have

$$p_q \approx \mathbf{a}_q^T \left(\sum_{q'=0}^{Q} \beta_{q'} \mathbf{a}_{q'}' \right), \quad q = 0,1,...,Q,$$

due to Eq. (4), Eq. (11) and $\mathbf{f} \approx \mathbf{f}'$. Besides the approximation loss, there is an additional loss caused by the rectification of the ReLU unit in form of

$$E_r(\hat{\mathbf{f}},\mathbf{f}') = \|\hat{\mathbf{f}} - \mathbf{f}'\|^2, \tag{12}$$

which is called the rectification loss.

Let $\Psi,\hat{\Psi}$ and Ψ' denote the space of $\mathbf{f} \in R^N$, the linear space spanned by A and the linear space spanned by

$$A' = \{\mathbf{a}_0,\mathbf{a}_1',...,\mathbf{a}_Q'\}. \tag{13}$$

It is clear from Eqs. (9) and (11) that

$$\Psi' \subset \hat{\Psi} \subset \Psi.$$

The loss between $\mathbf{f}$ and $\mathbf{f}'$ can be computed as

$$E(\mathbf{f},\mathbf{f}') = \|\mathbf{f} - \mathbf{f}'\|^2 = \|\mathbf{f} - \hat{\mathbf{f}}\|^2 + \|\hat{\mathbf{f}} - \mathbf{f}'\|^2 + 2(\mathbf{f} - \hat{\mathbf{f}})^T(\hat{\mathbf{f}} - \mathbf{f}'). \tag{14}$$

Recall that $\|\mathbf{f} - \hat{\mathbf{f}}\|^2 = E_a(\mathbf{f},\hat{\mathbf{f}})$ is the approximation loss and $\|\hat{\mathbf{f}} - \mathbf{f}'\|^2 = E_r(\hat{\mathbf{f}},\mathbf{f}')$ is the rectification loss. The term $2(\mathbf{f} - \hat{\mathbf{f}})^T(\hat{\mathbf{f}} - \mathbf{f}')$ becomes zero if $(\mathbf{f} - \hat{\mathbf{f}})$ and $(\hat{\mathbf{f}} - \mathbf{f}')$ are orthogonal to each other.

If $\|\mathbf{f} - \mathbf{f}'\|^2$ is negligible, we can set up a linear system of $(Q+1)$ equations in form of

$$p_q = \sum_{q'=0}^{Q} \mathbf{a}_q^T \mathbf{a}_{q'} \alpha_{q'} \quad q = 0,1,...,Q,$$

with $(Q+1)$ unknowns $\beta_q, q = 0,1,...,Q$. Again, if the $(Q+1) \times (Q+1)$ coefficient matrix is of full rank, we can solve $\alpha_{q'}$ uniquely. It is worthwhile to point out that the inverse RECOS transform is well defined once all anchor vectors are specified.

3. Saak transform

3.1. Forward Saak transform

The forward/inverse RECOS transforms have several limitations.

1. It is desired to have orthonormal transform kernels to facilitate forward and inverse transforms. In other words, anchor vectors should satisfy the following condition:

[1] The input $\mathbf{f}$ was assumed to be a mean-removed vector of unit length in [5,6]. This assumption is no more needed here.

$$\mathbf{a}_i^T \mathbf{a}_j = \langle \mathbf{a}_i, \mathbf{a}_j \rangle = \delta_{i,j}, \tag{15}$$

where $0 \leqslant i,j \leqslant K$ and $\delta_{i,j}$ is the Kroneckor delta function. Under this condition, the reconstruction formulas in Eqs. (9) and (11) can be greatly simplified as

$$\hat{\mathbf{f}} = \sum_{k=0}^{K} p_k \mathbf{a}_k, \quad \text{and} \quad \mathbf{f}' = \sum_{q=0}^{Q} p_q \mathbf{a}'_q, \tag{16}$$

respectively. Furthermore, the loss between $\mathbf{f}$ and $\mathbf{f}'$ in Eq. (14) can be computed exactly as

$$E(\mathbf{f},\mathbf{f}') = E_a(\mathbf{f},\hat{\mathbf{f}}) + E_r(\hat{\mathbf{f}},\mathbf{f}'), \tag{17}$$

since $(\mathbf{f}-\hat{\mathbf{f}})$ and $(\hat{\mathbf{f}}-\mathbf{f}')$ are orthogonal to each other. Unfortunately, anchor vectors in the RECOS transform do not satisfy the condition in Eq. (15).

2. Once the network architecture is fixed, the RECOS transform cannot make the approximation loss, $E_a(\mathbf{f},\hat{\mathbf{f}})$, arbitrarily small. The approximation error has a certain floor.
3. The rectification operation makes signal representation less efficient since the representation power of some anchor vectors is lost. It is desired to recover the representation capability of these rectified anchor vectors.

To address the first two problems, we consider the Karhunen-Loéve transform (KLT) on a set of mean-removed input vectors. Since the projection of input $\mathbf{f} \in R^N$ on the DC anchor vector gives its mean, the residual

$$\tilde{\mathbf{f}} = \mathbf{f} - p_0 \mathbf{a}_0 \tag{18}$$

is a zero-mean random vector. We compute the correlation matrix of $\tilde{\mathbf{f}}$, $\mathbf{R} = E[\tilde{\mathbf{f}}\tilde{\mathbf{f}}^T] \in R^{N \times N}$. Matrix $\mathbf{R}$ has $(N-1)$ positive eigenvalues and one zero eigenvalue. The eigenvector associated with the zero eigenvalue is the constant vector. The remaining $(N-1)$ unit-length eigenvectors define the Karhunen-Loéve (KL) basis functions, which are KLT's kernel vectors. It is well known that the KLT basis functions are orthonormal. Besides, we can order them according to their eigenvalues from the largest to the smallest, and the first M (with $M \leqslant N$) basis functions provide the optimal linear approximation subspace R^M in R^N. Consequently, we can choose these M orthonormal kernels to yield the minimum approximation error. The above-mentioned procedure is usually known as the Principal Component Analysis (PCA) or the truncated KLT.

To address the third problem, we propose to augment each kernel vector with its negative vector. That is, if $\mathbf{a}_k$ is a kernel, we choose $-\mathbf{a}_k$ to be another kernel. An example is shown in Fig. 2. In this figure, we show the projection of input $\mathbf{f}$ onto two AC anchor vectors $\mathbf{a}_1$ and $\mathbf{a}_2$ with $p_1 > 0$ and $p_2 < 0$ as their respective projection values. The ReLU preserves p_1 but clips to 0. By augmenting them with two more anchor vectors, $-\mathbf{a}_1$ and $-\mathbf{a}_2$, we obtain projection values $-p_1 < 0$ and $-p_2 > 0$. The ReLU clips $-p_1$ to 0 and preserves $-p_2$. The kernel augmentation idea is powerful in two aspects: 1) eliminating the rectification loss, and 2) annihilating the nonlinear effect of ReLU to greatly simplify the analysis. By following the notations in Section 2, we summarize the forward Saak transforms below.

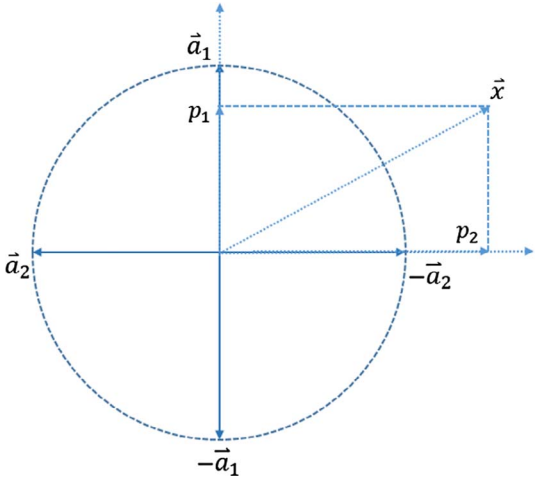


Fig. 2. Illustration of the kernel augmentation idea.

Forward Saak Transform.

1. Kernel Selection and Augmentation
   Collect a representative set of input samples $\mathbf{f} \in R^N$ and determine its KLT basis functions, which are denoted by $\mathbf{b}_k, k = 1,...,N$. The DC kernel $\mathbf{a}_0$ is defined by Eq. (3). The remaining $2(N-1)$ AC kernels are obtained using the augmentation rule:
   $$\mathbf{a}_{2k-1} = \mathbf{b}_k, \quad \mathbf{a}_{2k} = -\mathbf{b}_k, \quad k = 1,...,N-1. \tag{19}$$
2. Projection onto the Augmented Kernel Set
   Project input $\mathbf{f}$ on the augmented kernel set from Step 1:
   $$p_k = \mathbf{a}_k^T \mathbf{f}, \tag{20}$$
   and
   $$\mathbf{p} = (p_0, p_1, ..., p_{2(N-1)})^T \in R^{2N-1} \tag{21}$$
   is the projection vector of input $\mathbf{f}$.
3. Apply the ReLU to the projection vector except the first element to yield the final output:
   $$\mathbf{g} = (g_0, g_1, ..., g_{2(N-1)})^T \in R^{2N-1}, \tag{22}$$
   where $g_0 = p_0$ and
   $$g_{2k-1} = p_{2k-1} \text{ and } g_{2k} = 0, \quad \text{if } p_{2k-1} > 0, \tag{23}$$
   $$g_{2k-1} = 0 \text{ and } g_{2k} = p_{2k}, \quad \text{if } p_{2k} > 0. \tag{24}$$
   for $k = 1,...,N-1$.

The kernel augmentation scheme was described above in a way to motivate the Saak transform. In practical implementation, we can offer another view to the cascade of kernel augmentation and ReLU. As shown in Fig. 2, projection values p_k on KLT basis functions $\mathbf{b}_k, k = 0,...,N-1$ can be positive or negative. It is called the sign format of the projection output and denoted by

$$\mathbf{g}_s = (g_{s,0}, g_{s,1}, ..., g_{s,N-1})^T \in R^N, \tag{25}$$

where

$$g_{s,k} = \mathbf{b}_k^T \mathbf{f}, \quad k = 0,1,...,N-1. \tag{26}$$

In contrast, the position of each AC element in Eq. (22) is split into two consecutive positions. Its magnitude is recorded in the first and second positions, respectively, depending on whether it is positive or negative. This is called the position format of the projected output. The cascade of kernel augmentation and ReLU is equivalent to the sign-to-position (S/P) format conversion as shown in Fig. 3. To simplify the presentation below, we apply the S/P format conversion to the DC component as well. Thus, the dimension of position format $\mathbf{g}_p$ is twice of that of sign format $\mathbf{g}_s$. To give an example, the position format of $(5,-3)^T$ is $(5,0,0,3)^T$, and vice versa. Note that we demand the position-to-sign (P/S) format conversion for the inverse Saak transform.

3.2. Inverse Saak transform

The inverse Saak transform can be written as

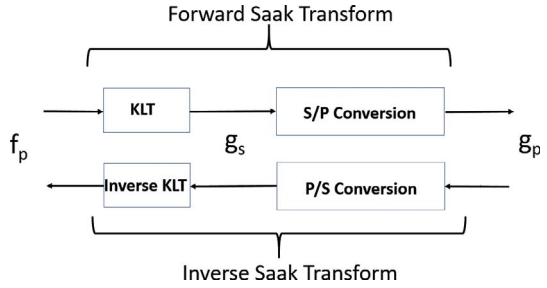


Fig. 3. The block diagram of forward and inverse Saak transforms, where $\mathbf{f}_p$,$\mathbf{g}_s$ and $\mathbf{g}_p$ are the input in position format, the output in sign format and the output in position format, respectively.

$$\mathbf{f}_p = \sum_{k=0}^{N-1} g_{s,k}\mathbf{b}_k, \tag{27}$$

where $\mathbf{b}_k$ is a basis function. Given position format $\mathbf{g}_p$, we need to perform the position-to-sign (P/S) format conversion to get sign format $\mathbf{g}_s$ and then feed it to the inverse KLT. This is illustrated in the lower branch of Fig. 3.

It is straightforward to verify the following.

- The input and the output of the forward/inverse KLT have the same l_2 distance.
- The input and the output of the S/P format conversion and the P/S format conversion have the same l_1 distance.

Besides, the l_1 distance is always no less than the l_2 distance. Let $\mathbf{g}_{p,1}$ and $\mathbf{g}_{p,2}$ be two output vectors of input vectors $\mathbf{f}_{p,1}$ and $\mathbf{f}_{p,2}$, respectively. It is easy to verify that

$$\|\mathbf{g}_{p,1}-\mathbf{g}_{p,2}\|_2 \leqslant \|\mathbf{g}_{s,1}-\mathbf{g}_{s,2}\|_2 = \|\mathbf{f}_{p,1}-\mathbf{f}_{p,2}\|_2. \tag{28}$$

As derived above, the l_2-distance of any two outputs of the forward Saak transform is bounded above by the l_2-distance of their corresponding inputs. This bounded-output property is important when we use the outputs of the Saak transform (i.e. Saak coefficients) as the features. Furthermore, we have

$$\|\mathbf{f}_{p,1}-\mathbf{f}_{p,2}\|_2 = \|\mathbf{g}_{s,1}-\mathbf{g}_{s,2}\|_2 \leqslant \|\mathbf{g}_{s,1}-\mathbf{g}_{s,2}\|_1 = \|\mathbf{g}_{p,1}-\mathbf{g}_{p,2}\|_1, \tag{29}$$

which is important for the inverse Saak transform since the roles of inputs and outputs are reversed. In this context, the l_2-distance of any two outputs of the inverse Saak transform is bounded above by the l_1-distance of their corresponding inputs.

There are similarities and differences between the RECOS and the Saak transforms. For similarities, both project the input onto each kernel vector in the kernel set and adopt the ReLU to avoid sign confusion when multiple linear transforms are in cascade. For differences, the RECOS transform does not specify the kernel number, which is determined heuristically. Its kernel weights are optimized by back-propagation. In developing the Saak transform, we are concerned with efficient forward and inverse transform simultaneously so that kernel orthonormality is demanded. We argument transform kernels to annihilate the loss caused by the ReLU. The kernel number for a lossless Saak transform is uniquely specified. The approximation loss and the rectification loss of the RECOS transform are completely eliminated in the lossless Saak transform.

4. Multi-stage Saak transforms

To transform images of a larger size, we decompose an image into four quadrants recursively to form a quad-tree structure with its root being the full image and its leaf being a small patch of size 2×2. The first-stage Saak transform is applied at the leaf node. Then, multi-stage Saak transforms are applied from all leaf nodes to their parents stage by stage until the root node is reached. This process can be elaborated below.

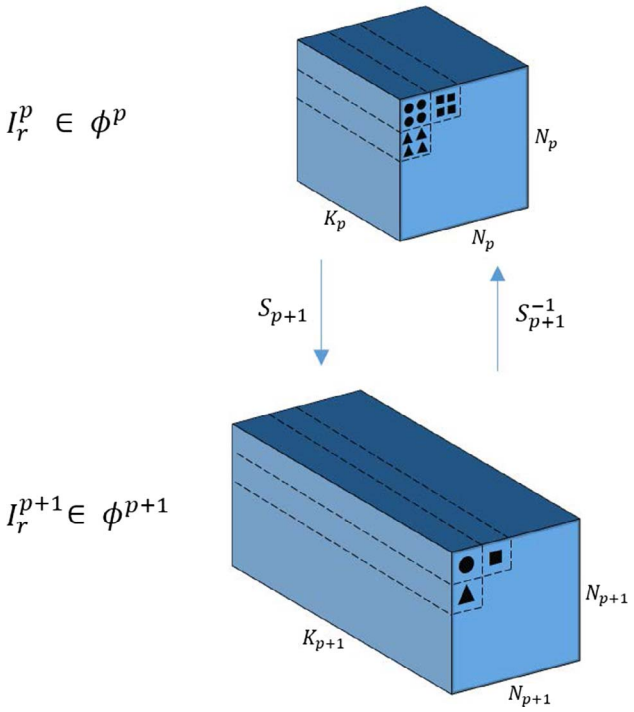


Fig. 4. The relationship between local cuboids (LCs) and global cuboids (GCs) in stages p and $p+1$, where three representative LCs are shown with solid dots, triangles and squares, respectively. The same Saak transform is applied to all LCs and each of them yields one output spectral vector. Sets Φ^p and $\Phi^{(p+1)}$ are collections of all GCs in Stages p and $(p+1)$ and images I_r^p and $I_r^{(p+1)}$ are samples in Φ^p and $\Phi^{(p+1)}$, respectively.

- Stage 1:
  Conduct the KLT on function values defined on non-overlapping local cuboids (LCs) of size $2 \times 2 \times K_0$, where $K_0 = 1$ for a monochrome image and $K_0 = 3$ for a color image, each of which yields a set of signed KLT coefficients. The spatial dimension of the entire input or the global cuboid (GC) is reduced by one half in both horizontal and vertical directions.
- Stage $p = 2,3,\ldots,$
  - Step (a): The signed KLT coefficients are converted to the positioned KLT coefficients. The spectral dimension is doubled if DC and AC components are treated in the same manner.
  - Step (b): Conduct the KLT on positioned KLT coefficients defined on non-overlapping LCs of dimension $(2 \times 2) \times 2K_{p-1}$, each of which yields a vector of signed KLT coefficients of dimension $K_p = 8 \times K_{p-1}$ as shown in Fig. 4. Since the spatial dimension of the input LC is 2×2 and the spatial dimension of the output LC is 1×1, the spatial dimension of the GC is reduced by one half in both horizontal and vertical directions. There is no explicit spatial pooling in multi-stage Saak transforms since non-overlapping cuboids are used here.
- Stopping Criterion
  The whole process is stopped when we reach the last stage that has one set of signed KLT coefficients of dimension $1 \times 1 \times K_f$. If the image size is of $2^P \times 2^P$, we have $K_f = 2^{3P}$.

The signed KLT coefficients in the final stage are called the last-stage Saak coefficients. The process is illustrated in Fig. 5.

The Saak transform is a mapping from a real-valued function defined on cuboid $C(i_p,j_p,L_i,L_j,L_k)$ to an output vector. In practice, its spatial dimensions L_i and L_j should be relatively small so that the total number of grid points is under control. Here, we set $L_i = L_j = 2$. Hence,

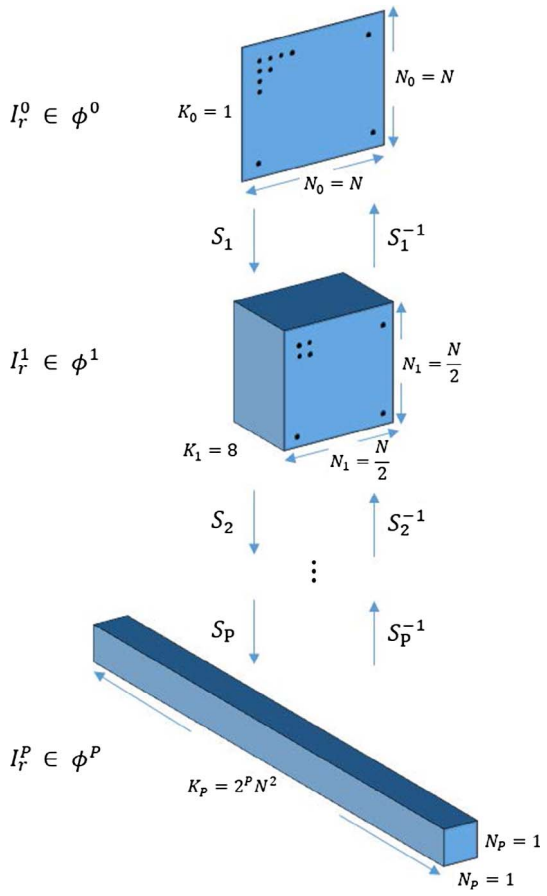


Fig. 5. The conversion between an input image and its P spatial-spectral representations via forward and inverse multi-stage Saak transforms, where set Φ^p contains all GCs in the pth stage, and S_p and S_p^{-1} are the forward and inverse Saak transforms between stages $(p-1)$ and p, respectively.

the Saak transform handles a cuboid of spatial dimension 2×2, which is called the local cuboid (LC). In contrast, the cuboid in set Φ^p as shown in Fig. 5 is the GC in the pth stage and denoted by GC_p. A GC can be decomposed into non-overlapping LCs. The relationship between the LC_p and the GC_p is illustrated in Fig. 4. The spatial dimension, $N_p \times N_p$, of the input GC in Step (a) of Stage $(p+1)$, can be recursively computed as

$$N_p = N_{p-1}2^{-1}, \quad p = 1,2,... \text{ and } N_0 = N. \tag{30}$$

It implies that $N_p = N \times 2^{-p}$. Since $N = 2^P$, we have $N_P = 1$.

The GC_p and LCs have the same number of spectral components in each stage. By augmenting DC and AC transform kernels equally, the spatial-spectral dimension of LC_p is $2 \times 2 \times K_p$, where K_p can be recursively computed as

$$K_p = 2 \times 4 \times K_{(p-1)}, \quad K_0 = 1, \quad p = 1,2,... \tag{31}$$

The right-hand-side (RHS) of the above equation is the product of three terms. The first one is due to the S/P conversion. The second and third terms are because that the degree of freedom of the input cuboid and the output vector should be preserved through the KLT.

With this expression, we can give a physical meaning to the Saak transform from the input LC of dimension $2 \times 2 \times K_{p-1}$ in position format to the output LC of dimension $1 \times 1 \times K_p$ in sign format (i.e. before kernel augmentation). The forward Saak transform merges the spectra of 4 child nodes (corresponding to 4 spatial sub-regions) into the spectrum of their parent node (corresponding to the union of the 4 spatial sub-regions). The inverse Saak transform splits the spectrum of a parent node into the spectra of its 4 child nodes. In other words, the forward Saak transform is a process that converts spatial variation to spectral variation while the inverse Saak transform is a process that converts spectral variation to spatial variation. Multi-stage Saak transforms provide a family of spatial-spectral representations through recursion. The spatial resolution is the highest in the source image and the spectral resolution is the highest in the output of the last stage Saak transform. The intermediate stages provide different spatial-spectral trade offs.

It is worthwhile to point out that we can adopt a different decomposition to make the hierarchical tree shallower, say, splitting a parent node into $m \times m$ (m = 3,4,... child nodes. Then, the tree has a depth of $\log_m N$. As one traverses the tree from the leaf to the root, the coverage (or the receptive field) is larger, the spatial resolution of the GC is lower, and the spectral component number is larger. As a consequence of Eq. (31), we have $K_p = 8^p$, which grows very quickly. In practice, we should leverage the energy compaction property of the KLT and adopt the lossy Saak transform by replacing the KLT with the truncated KLT (or PCA). We focus on the lossless Saak transform due to space limitation in this work, and will discuss the topic of lossy Saak transforms systematically and separately in the future.

Based on the above discussion, Saak transform kernels actually do not depend on the geometrical layout of input elements but their statistical correlation property. Thus, *if we can decompose a set of high-dimensional random vectors into lower-dimensional sub-vectors hierarchically, we can apply the Saak transform to these sub-vectors stage by stage from bottom to top until the full random vector is reached*. We can generalize multi-stage Saak transforms to other types of signals, say, communication signals received by receiver arrays of an irregular geometry that contain both spatial and temporal data as input elements.

5. MNIST image reconstruction, feature extraction and classification

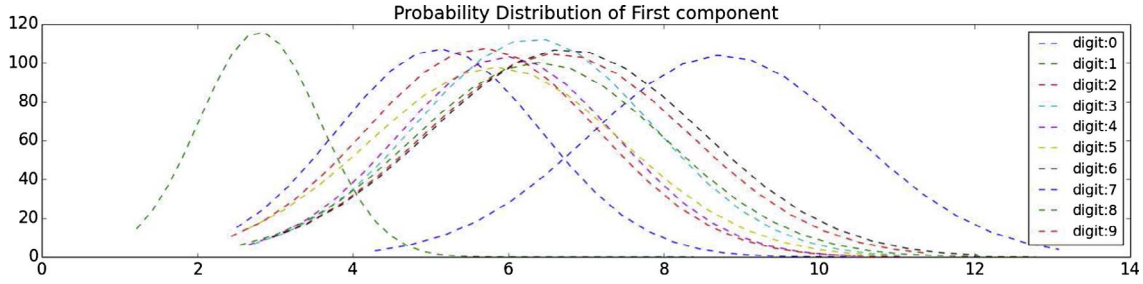
We use the MNIST dataset[2] as an example to illustrate the image reconstruction, feature selection and classification processes. Since we are only concerned with the signed Saak coefficients in this section, we simply call them Saak coefficients for convenience.

5.1. Distributions of Saak coefficients

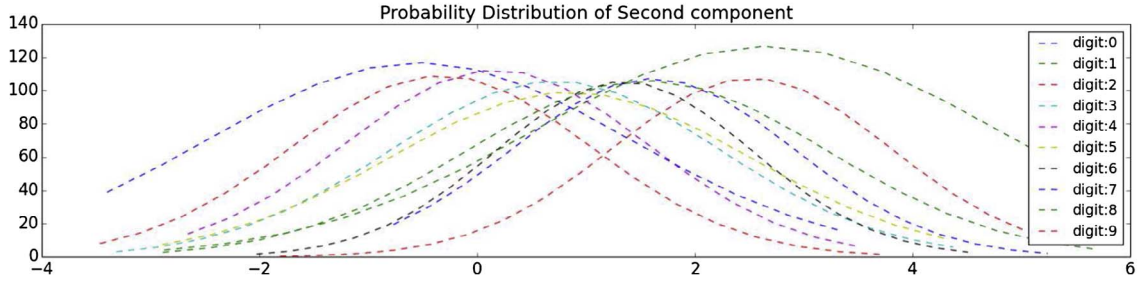
There are 10 handwritten digit classes ranging from 0 to 9 in the MNIST dataset. For image samples of the same digit, we expect their Saak coefficients to have a clustering structure. We plot the distributions of the first three last-stage Saak coefficients in Fig. 6. Each plot has ten curves corresponding to the smoothed histograms of ten digits obtained by 1000 image samples for each digit. By inspection, each of them looks like a normal distribution. To verify normality, we show the percentages of the leading 256 last-stage Saak coefficients for each digit class that pass the normality test with 100 and 1000 samples per class in Table 1. We adopt the Jarque-Bera test [7] for the normality test and the Grubbs' test [8] for outlier removal in this table.

We see from this table that, when each class has 100 samples, 90% or higher of these 256 last-stage Saak coefficients pass the normality test after outlier removal. However, the percentages drop when the sample number increases from 100 to 1000. This sheds light on the difference between small and big datasets. It is a common perception that the Gaussian mixture model (GMM) often provides a good signal model. Even it is valid when the sample number is small, there is no guarantee that it is still an adequate model when the sample number becomes very large.

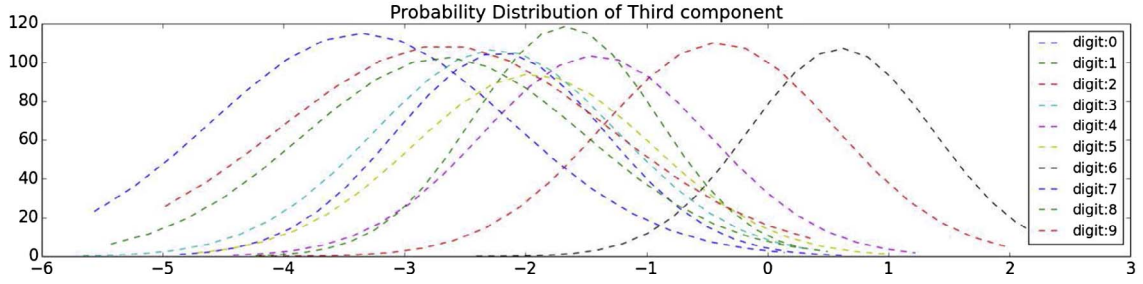
[2] http://yann.lecun.com/exdb/mnist/.



(a) The first spectral component



(b) The second spectral component



(c) The third spectral component

Fig. 6. The distributions of Saak coefficients for the (a) first, (b) second and (c) third components for 10 digit classes.

Table 1
The percentages of the leading 256 last-stage Saak coefficients that pass the normality test for each digit class before and after outlier removal with 100 and 1000 samples per digit class, where S denotes the number of samples.

Digit Class	0	1	2	3	4	5	6	7	8	9
S = 100, raw	91	73	96	90	87	89	88	88	90	86
S = 100, outlier removed	97	89	98	97	97	96	96	93	98	96
S = 1000, raw	63	30	74	68	56	66	53	46	61	53
S = 1000, outlier removed	77	52	87	85	80	84	72	75	85	74

5.2. Image synthesis via inverse Saak transform

We can synthesize an input image with multi-stage inverse Saak transforms with a subset of its last-stage Saak Coefficients. An example is shown in Fig. 7, where six input images are shown in the first row and their reconstructed images using leading 100, 500, 1000, 2000 last-stage Saak coefficients are shown in the 2nd, 3rd, 4th and 5th rows, respectively. The last row gives reconstructed images with all Saak coefficients (a total of 16,000 coefficients). Images in the first and the last rows are actually identical. It proves that multi-stage forward and inverse Saak transforms are exactly the inverse of each other numerically. The quality of reconstructed images with 1000 Saak coefficients is

Fig. 7. Illustration of image synthesis with multi-stage inverse Saak transforms (from top to bottom): six input images are shown in the first row and reconstructed images based on the leading 100, 500, 1000 and 2000 last-stage Saak coefficients are shown from the second to the fifth rows. Finally, reconstructed images using all last-stage Saak coefficients are shown in the last row. Images of the first and the last rows are identical, indicating lossless full reconstruction.

already very good. It allows people to see them clearly without confusion. Thus, we expect to get high classification accuracy with these coefficients, which will be confirmed in Section 5.4. Furthermore, it indicates that we can consider lossy Saak transforms in all stages to reduce the storage and the computational complexities.

5.3. Feature selection

The total number of Saak coefficients in lossless multi-stage Saak transforms is large. To use Saak coefficients as image features for classification, we can select a small subset of coefficients by taking their discriminant capability into account. To search Saak coefficients of higher discriminant power, we adopt the F-test statistic (or score) in the ANalysis Of VAriance (ANOVA), which is in form of

$$F = \frac{\text{between-group variability(BGV)}}{\text{within-group variability(WGV)}}. \tag{32}$$

The between-group variability (BGV) and the within-group variability (WGV) can be written, respectively, as

$$\text{BGV} = \sum_{c=1}^{C} n_c(\overline{S}_c - \overline{S})^2/(C-1), \tag{33}$$

and

$$\text{WGV} = \sum_{c=1}^{C} \sum_{d=1}^{n_c} (S_{c,d} - \overline{S}_c)^2/(T-C), \tag{34}$$

where C is the class number, n_c is the number of Saak coefficients of the cth class, $\overline{S}_c$ is the sample mean of the cth class, $\overline{S}$ is the mean of all samples, $S_{c,d}$ is the dth sample of the cth class, and T is the total number of samples. If 6000 samples per class is used in the training, we have $C = 10, n_c = 6000$ and $T = 60{,}000$. We select Saak coefficients of higher discriminant power by computing their F-test scores. The larger the F-test score, the higher the discriminant power. These selected Saak coefficients form the raw feature vector. Then, we apply the dimension reduction technique further to obtain the reduced-dimension feature vector.

5.4. MNIST image classification results

We use 60,000 image samples (namely, 6000 training images for each digit) to compute the F-test score of all Saak coefficients. Then, we test with the following three settings.

- Setting No. 1: Selecting the leading 2000 last-stage Saak coefficients (without considering their F-test scores).
- Setting No. 2: Selecting 2000 last-stage Saak coefficients with the highest F-test scores.
- Setting No. 3: Selecting 2000 Saak coefficients with the highest F-test scores from all stages.

Furthermore, we conduct the PCA on these 2000 selected Saak coefficients (called the raw feature vector) to reduce the feature dimension from 2000 to 64, 128 or 256 (called the reduced-dimension feature vector). Finally, we apply the SVM classifier to the reduced-dimension feature vectors of each test image. The classification accuracy is given in Table 2. It is no surprise that Setting NO. 1 is the worst while Setting No. 3 is the best. However, it is worthwhile to emphasize that the performance gap between Setting No. 1 and No. 2 is very small. It shows the energy compaction power of the Saak transform. If we select features only from the last-stage Saak coefficients, it appears to be fine to focus on leading coefficients. However, to boost the classification performance more significantly, we need to consider Saak coefficients from earlier stages since these Saak coefficients take both spatial and spectral information into account.

Table 2
The classification accuracy using the SVM classifier, where each column indicates the reduced feature dimension from 2000 Saak coefficients. Each row indicates a particular setting in selecting the Saak coefficients.

	64	128	256
Setting No. 1	97.22	97.08	96.82
Setting No. 2	97.23	97.10	96.88
Setting No. 3	98.46	98.50	98.31

Table 3
The classification accuracy using the SVM classifier under Setting No. 3, where each column indicates the raw feature dimension and each row indicates the reduced feature dimension.

	1000	2000	3000	4000	5000
64	98.49	98.46	98.42	98.43	98.41
128	98.46	98.50	98.52	98.51	98.52
256	98.20	98.31	98.27	98.29	98.25

Table 4
The classification accuracy using the KNN classifier under Setting No. 3, where each column indicates the raw feature dimension and each row indicates the reduced feature dimension.

	1000	2000	3000	4000	5000
64	97.53	97.52	97.50	97.52	97.45
128	97.49	97.45	97.46	97.37	97.39
256	97.50	97.42	97.39	97.41	97.39

Since Setting No. 3 gives the best performance, we choose this setting and vary the dimension of raw and reduced-dimension feature vectors. The classification accuracy results are shown in Table 3. We see that these results do not vary much. The accuracy is all between 98% and 99%. The best result in this table is 98.52%. Furthermore, we still choose setting No. 3 but adopt a different classifier, which is the KNN classifier with $K = 5$. The classification accuracy results are shown in Table 4. Again, we see that these results do not vary much. The accuracy is all between 97% and 98%. The best result in this table is 97.53%. The SVM classifier is better than the KNN classifier in this example.

6. CNN versus Saak transform: comparative study

CNNs and multi-stage Saak transforms share some similarities in overall system design. For example, both adopt the convolution (or transform) operations to generate responses which are followed by the ReLU operation before being fed into the next layer (or stage). However, there are major differences between them. We compare them in four aspects below.

6.1. End-to-end optimization versus modular design

A CNN is built upon the choice of an end-to-end network architecture and an optimization cost function. The filter weights are optimized through backpropagation which is an iterative optimization procedure. The Saak transform solution follows the traditional pattern recognition framework that by partitioning the whole recognition task into two separate modules; namely, the feature extraction module and the classification module.

If the number of object classes is fixed, the training and testing data are reasonably correlated, and the cost function is suitably defined, the end-to-end optimization approach taken by CNNs offers the state-of-the-art performance. However, this end-to-end optimization methodology has its own weaknesses: (1) robustness against perturbation, (2) scalability against the class number, (3) portability among different

datasets. The first one has been reported in several papers, e.g. [9–11]. For the second one, if we increase or decrease the object class number by one for the ImageNet, all filter weights of the whole network, which consists of the feature extraction subnet and the decision subnet, have to be retrained. It is understandable that the increase or decrease of object classes will affect the decision subnet. However, it is against intuition that filter weights of the feature extraction subnet are also affected by the object class number. For the third one, it explains the need of a large amount of research for domain adaption [12,13]. The end-to-end objective optimization of CNNs offers the best performance for a specific task under a specific setting at the cost of robustness, flexibility and generalizability.

In contrast with CNN's end-to-end optimization methodology, the modular design and the Saak transform approach is expected to be more robust against perturbation and less sensitive to the variation of object classes. The small perturbation will not affect leading last-stage Saak coefficients much due to the use of PCA. Kernels in earlier Saak transform stages should not change much if their covariance matrices do not change much. If this is the case, we can use the same network to generate different Saak features for new object classes. More studies along this line will be reported.

6.2. Generative model versus inverse transform

No inverse transform has been studied for CNNs except the inverse RECOS presented in Section 2 of this work. On the other hand, two approaches have been examined to generate images based on CNNs. One is the Generative Adversarial Network (GAN) [14] and the other is the Variational AutoEncoder (VAE) [15]. One has to train a generative network and a discriminative network for the GAN solution, and an encoder network and a decoder network for the VAE solution. Once the whole network system is trained, one can use a latent vector to generate images through the generative network and the decoder for GANs and VAEs, respectively.

Here, we follow the traditional signal analysis and synthesis framework. The kernels of the Saak transform are orthonormal, and the inverse Saak transform can be easily defined and conducted conveniently. For an arbitrary input, the cascade of the forward and inverse multi-stage Saak transforms results in an identity operator. This is a trivial case. It is however possible to consider non-trivial applications. To take the single image super-resolution problem as an example, we can build two forward/inverse Saak transform pipes – one for the low resolution images and the other for the high resolution images. We may try the following high-level idea. That is, one can build bridges between these two pipes and, then, feed the low-resolution images into the low-resolution forward transform pipe and get the high-resolution images from the high-resolution inverse transform pipe. This is an interesting research topic for further exploration.

6.3. Theoretical foundation

Significant efforts have been made to shed light on the superior performance of CNNs. The early work of Cybenko [16] and Hornik et al. [17] interpreted the multi-layer perceptron (MLP) as a universal approximator. Recent theoretical studies on CNNs include scattering networks [18–20], tensor analysis [21], generative modeling [22], relevance propagation [23], Taylor decomposition [24], the multi-layer convolutional sparse coding (ML-CSC) model [25], over-parameterized shallow neural network optimization [26], etc. Another popular research topic is the visualization of filter responses at various layers [27–29]. Despite intuition was given in the above-mentioned work, complete CNN theory is lacking. As the complexity of recent CNN architectures goes higher, the behavior of these networks is mathematically intractable. In contrast, The Saak transform is fully built upon linear algebra and statistics. It can be easily and fully explained mathematically.

6.4. Filter weight determination, feature selection and others

There is a fundamental difference in filter weight determination between CNNs and Saak-transform-based machine learning methodology. Filter weights in CNNs are determined by data and their associated labels. They are updated by backpropagation using such information pair provided at two ends of the network. A CNN learns its best parameters by optimizing an objective function iteratively. The iteration number is typically very large. We propose another fully automatic filter weight selection scheme. The Saak transform select its kernel functions (or filter weights) using the KLT stage by stage. It is built upon the second-order statistics of the input data. Neither data labels nor backpropagation is needed. Data labels are only needed in the decision module, which is decoupled from the Saak transform module.

Before the resurgence of CNNs, feature extraction was probably the most important ingredient in pattern recognition. Different applications demanded different features, and they were extracted heuristically. These features are called the "handcrafted" features nowadays. One of the key characteristics of the CNN approach is that features can be "learned" automatically from the data and their labels through backpropagation. These iteratively optimized features are called the "deep" or the "learned" features. Here, we provide a third approach to feature extraction based on multi-stage Saak transforms. They are the Saak features. To obtain Saak features, we need data labels but use them in a different manner. Multi-stage Saak coefficients of samples from the same class are computed to build feature vectors of that class. We should collect those Saak coefficients that have higher discriminant power among object classes to reduce the feature dimension.

There are differences in implementation details. CNNs conduct convolutional operations on overlapping blocks/cuboids and adopt spatial pooling to reduce the spatial dimension of responses before they are fed into the next layer. The Saak transform is applied to non-overlapping blocks/cuboids and no spatially pooling is needed. It achieves spatial dimension reduction with no additional effort. The horizontal and vertical spatial dimensions of CNN filters typically take odd numbers, say 3×3,5×5,11×11, etc. The spatial dimension of the Saak transform in each stage can be even or odd. The CNN parameter setting mostly follows prior art as a convention. The parameters of the Saak transform can be determined with theoretical support.

7. Conclusion and future work

Data-driven forward and inverse Saak transforms were proposed. By applying multi-stage Saak transforms to a set of images, we can derive multiple representations of these images ranging from the pure spatial domain to the pure spectral domain as the two extremes. There is a family of joint spatial-spectral representations with different spatial-spectral trade offs between them. The Saak transform offers a new angle to look at the image representation problem and provides powerful spatial-spectral Saak features for pattern recognition. The MNIST dataset was used as an example to demonstrate this application.

There are several possible extensions. First, we focused on the lossless Saak transform in this work. It is desired to study the lossy Saak transform by removing spectral components of less significance in a systematic fashion. Second, we used the F-test score to select Saak coefficients of higher discriminant power. There may be other more effective methods to extract discriminant features from a huge number of Saak coefficients to achieve better classification performance. Third, the Saak transform can be applied to any high dimensional random vectors as long as they can be decomposed into lower dimensional subvectors hierarchically. Along this line, we may generalize the Saak-transform-based feature extraction method to other types of signals, say, communication signals. Fourth, the inverse Saak transform can be used for image synthesis/generation. It is interesting to investigate the Saak-transform-based image generation approach and compare it with other approaches such as VAEs and GANs.

Acknowledgment

This material is partially based on research sponsored by DARPA and Air Force Research Laboratory (AFRL) under agreement number FA8750-16-2-0173. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA and Air Force Research Laboratory (AFRL) or the U.S. Government. This project was also partially supported by the National Heart Lung Institute (R01HL129727) (T.K.H.).

References

[1] H. Stark, J.W. Woods, Probability, Random Processes, and Estimation Theory for Engineers, Prentice Hall, Englewood Cliffs, 1986.
[2] N. Ahmed, T. Natarajan, K.R. Rao, Discrete cosine transform, IEEE Trans. Comput. 100 (1) (1974) 90–93.
[3] Y. LeCun, Y. Bengio, G.E. Hinton, Deep learning, Nature 521 (2015) 436–444.
[4] B.H. Juang, Deep neural networks–a developmental perspective, APSIPA Trans. Signal Inform. Process. 5 (2016) e7.
[5] C.-C.J. Kuo, Understanding convolutional neural networks with a mathematical model, J. Vis. Commun. Image Represent. 41 (2016) 406–413.
[6] C.-C.J. Kuo, The CNN as a guided multilayer RECOS transform [lecture notes], IEEE Signal Process. Mag. 34 (3) (2017) 81–89.
[7] C.M. Jarque, A.K. Bera, A test for normality of observations and regression residuals, Int. Stat. Rev./Revue Internationale de Statistique (1987) 163–172.
[8] F.E. Grubbs, Sample criteria for testing outlying observations, Ann. Math. Stat. (1950) 27–58.
[9] A. Nguyen, J. Yosinski, J. Clune, Deep neural networks are easily fooled: high confidence predictions for unrecognizable images, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 427–436.
[10] S.-M. Moosavi-Dezfooli, A. Fawzi, P. Frossard, Deepfool: a simple and accurate method to fool deep neural networks, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 2574–2582.
[11] A. Fawzi, S.M. Moosavi Dezfooli, P. Frossard, A Geometric Perspective on the Robustness of Deep Networks, Tech. Rep., Institute of Electrical and Electronics Engineers, 2017.
[12] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, T. Darrell, Decaf: a deep convolutional activation feature for generic visual recognition, in: International Conference on Machine Learning, 2014, pp. 647–655.
[13] Y. Ganin, V. Lempitsky, Unsupervised domain adaptation by backpropagation, in: International Conference on Machine Learning, 2015, pp. 1180–1189.
[14] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative adversarial nets, in: Advances in Neural Information Processing Systems, 2014, pp. 2672–2680.
[15] D.P. Kingma, M. Welling, Auto-encoding Variational Bayes. Available from: < 1312.6114 > .
[16] G. Cybenko, Approximation by superpositions of a sigmoidal function, Math. Control Signals Syst. 2 (4) (1989) 303–314.
[17] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural Netw. 2 (5) (1989) 359–366.
[18] S. Mallat, Group invariant scattering, Commun. Pure Appl. Math. 65 (10) (2012) 1331–1398.
[19] J. Bruna, S. Mallat, Invariant scattering convolution networks, IEEE Trans. Pattern Anal. Mach. Intell. 35 (8) (2013) 1872–1886.
[20] T. Wiatowski, H. Bölcskei, A Mathematical Theory of Deep Convolutional Neural Networks for Feature Extraction. Available from: < 1512.06293 > .
[21] N. Cohen, O. Sharir, A. Shashua, On the Expressive Power of Deep Learning: A Tensor Analysis. Available from: < 1509.05009 > (556).
[22] J. Dai, Y. Lu, Y.-N. Wu, Generative Modeling of Convolutional Neural Networks. Available from: < 1412.6296 > .
[23] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, W. Samek, On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation, PloS One 10 (7) (2015) e0130140.
[24] G. Montavon, S. Bach, A. Binder, W. Samek, K.-R. Müller, Explaining Nonlinear Classification Decisions with Deep Taylor Decomposition. Available from: < 1512.02479 > .
[25] J. Sulam, V. Papyan, Y. Romano, M. Elad, Multi-layer Convolutional Sparse Modeling: Pursuit and Dictionary Learning. Available from: < 1708.08705 > .
[26] M. Soltanolkotabi, A. Javanmard, J.D. Lee, Theoretical Insights Into the Optimization Landscape of Over-parameterized Shallow Neural Networks. Available from: < 1707.04926 > .
[27] K. Simonyan, A. Vedaldi, A. Zisserman, Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps. Available from: < 1312.6034 > .
[28] M.D. Zeiler, R. Fergus, Visualizing and understanding convolutional networks, in: European Conference on Computer Vision, Springer, 2014, pp. 818–833.
[29] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, A. Torralba, Object Detectors Emerge in Deep Scene CNNs. Available from: < 1412.6856 > .