Mohammadnia Qaraei, Mohammadreza; Abbaasi, Saeid; Ghiasi-Shirazi, Kamaledin

Randomized non-linear PCA networks

Published in:
Information Sciences (Elsevier)

DOI:
10.1016/j.ins.2020.08.005

Published: 04/02/2021

Document Version
Peer-reviewed accepted author manuscript, also known as Final accepted manuscript or Post-print

Randomized Non-linear PCA Networks

Mohammadreza Qaraei[a], Saeid Abbaasi[b], Kamaledin Ghiasi-Shirazi[b,*]

[a] *Aalto University, Helsinki, Finland*
[b] *Department of Computer Engineering, Ferdowsi University of Mashhad, Mashhad, Iran*

Abstract

PCANet is an unsupervised Convolutional Neural Network (CNN), which uses Principal Component Analysis (PCA) to learn convolutional filters. One drawback of PCANet is that linear PCA cannot capture nonlinear structures within data. To address this problem, a straightforward approach is utilizing kernel methods by equipping the PCA method in PCANet with a kernel function. However, this practice leads to a network having cubic complexity with respect to the number of training image patches. In this paper, we propose a network called Randomized Nonlinear PCANet (RNPCANet), which uses an explicit kernel PCA to learn the convolutional filters. Although RNPCANet utilizes kernel methods for nonlinear processing of data, using kernel approximation to define an explicit feature space in each stage, we theoretically show that the complexity of this model is not much higher than PCANet. We also show that our method links PCANets to Convolutional Kernel Networks (CKNs) as the proposed model maps the patches to a kernel feature space similar to CKNs. We evaluate our model on image recognition tasks including Coil-20, Coil-100, ETH-80, Caltech-101, MNIST, and C-Cube datasets. The experimental results show that the proposed method has superiority over PCANet and CKNs in terms of recognition accuracy.

Keywords: Convolutional Neural Networks, PCANet, Random Fourier

*Corresponding author
Email addresses: mohammadreza.mohammadniaqaraei@aalto.fi (Mohammadreza Qaraei), s.abbaasi@mail.um.ac.ir (Saeid Abbaasi), k.ghiasi@um.ac.ir (Kamaledin Ghiasi-Shirazi)

1. Introduction

Convolutional Neural Networks (CNNs) [1] are a type of multilayer feedforward neural network, which have achieved state-of-the-art results on visual recognition tasks in recent years. By a multi-stage structure, CNNs are capable of extracting robust high-level image features, which is a challenging problem in machine vision tasks [2]. The core layers of CNNs are convolutional layers, nonlinearity layers, and pooling layers. In convolutional layers, several learnable filters are convolved with the input of the layer in order to extract features from data. In nonlinearity layers, a nonlinear transformation is applied to the input of the layer so that the network can deal with nonlinear data. A pooling layer aggregates and downsamples the features extracted in the preceding layer to make the network robust to translation of the input data.

Training of filters in ordinary CNNs is performed by Stochastic Gradient Descent (SGD) with the backpropagation algorithm. Although SGD with backpropagation can achieve state-of-the-art results, this method needs a lot of computations. To overcome this drawback, some researchers proposed simpler methods for training CNNs. These methods usually perform training in a feedforward layer-by-layer manner. Two examples are ScatNet, which is a learning-free CNN that uses wavelet operators as convolutional filters [3], and Convolutional Kernel Networks (CKNs), in which the filters are optimized in order to approximate a convolution kernel [4].

Another layer-wise trained CNN is PCANet that simply uses Principal Component Analysis (PCA) to learn filters [5]. More specifically, in PCANet, the principal components of the patches extracted from the input of a layer are employed to assign the filters of that layer. After several Convolutional layers, the features are encoded by a binary hashing followed by block-wise histograms, which can be interpreted as nonlinearity and pooling layers. After this step, the extracted features may be used for training a classifier like a Support Vector

Machine (SVM).

The PCA algorithm used in PCANet is a classical feature extraction method. PCA seeks for a linear subspace in which the variance of the data is maximum. One limitation of PCA (and consequently PCANet) is that PCA is a linear method and cannot reveal nonlinear relationships between the features. A straightforward approach to make a nonlinear PCA is to use the kernel trick, which leads to Kernel Principal Component Analysis (KPCA) [6]. In KPCA, by using a kernel function, the data is implicitly mapped to a feature space (in which the data has possibly a linear structure) and the PCA algorithm is performed in that feature space. In some problems, it has been shown that a kernel-based algorithm can perform at least as well as the linear version of that algorithm [7]. Moreover, kernel-based models have higher complexity than linear models which makes them superior to linear models in dealing with real-world data [8]. However, KPCA suffers from a high computational complexity and needs a lot of memory as it engages in computing principal values of the kernel matrix. One way to make the kernel methods scalable is to approximate a kernel function by explicit feature maps. Random Fourier Features (RFF) is one of the popular methods for approximating kernel functions [9]. In RFF, explicit feature maps of a shift-invariant kernel are approximated using random projections followed by a sinusoidal nonlinearity. The effectiveness of using the RFF method in the nonlinear principal analysis has been discussed in [10, 11, 12]. Specifically, [10] showed that the approximated Gram matrix of a Gaussian kernel obtained by RFF converges polynomially in spectral norm to the true Gram matrix and [11, 12] showed that $O(\sqrt{n})$ random features are sufficient for the approximation of Kernel PCA using RFF to achieve comparable performance with the exact KPCA, where n is the number of data.

Motivated by the performance of RFF, in this paper, we propose a Randomized Nonlinear PCA Network (RNPCANet). Similar to PCANet, RNPCANet is a multi-stage convolutional neural network. In every stage of RNPCANet, each patch is mapped to an approximated explicit kernel feature space using RFF. Then for patches in the feature space, a linear subspace is obtained using

PCA and each patch is projected on it. At the end of all the stages, we perform a binary hashing followed by block-wise histograms to encode the features. In contrast to PCANet that uses linear PCA, RNPCANet can capture nonlinear relationships between the features of patches in each stage of the network. As we use the efficient RFF method for nonlinear processing of data, the computational complexity of RNPCANet is not much higher than PCANet, while it has a notable efficiency compared to the use of the exact kernel function.

We evaluate RNPCANet on image classification tasks including character and object recognition. The experiments show that the proposed method outperforms PCANet in terms of recognition accuracy.

The organization of the paper is as follows. In Section 2, we review the related works on improving PCANet. In Section 3, we discuss kernel approximation using the RFF method and show how this method is applied to approximate kernelized machine learning models. In Section 4, we introduce RNPCANet and discuss the computational complexity of this model as well as the its relation to convolutional kernel networks. In Section 5, we evaluate RNPCANet on image recognition tasks and compare it with PCANet. Finally, in Section 6, we conclude the paper and give some suggestions for future works.

2. Related Work

In this section, we review the related works on enhancing the performance of PCANet by replacing PCA with other learning methods, or by modifying the core structure of PCANet, or embedding nonlinear processing of data in each stage of PCANet. We also review the related works on using kernel approximation techniques in neural networks.

There are several works in the literature which used other learning methods in place of PCA in a PCANet architecture in order to improve accuracy or make the training simpler. For example, Ng and Teoh [13] proposed DCTNet, in which no learning is required and the filters are the coefficients of a 2D Discrete Cosine Transform. They also proposed Histogram Tied Rank Normal-

ization to regulate the final feature vectors obtained by block-wise histograms in order to increase robustness. Feng et al. [14] proposed DLANet, in which the filters of every layer are learned by Discriminant Locality Alignment algorithm to map the patches to a space where the inter-class patches have a large margin and the distance between intra-class patches is minimized. Zeng et al. [15] proposed MLDNet for multidimensional object classification, in which the filters are learned by Multilinear Discriminant Analysis. Jia et al. [16] introduced 2DPCANet that uses two-dimensional PCA instead of ordinary PCA in order to retain the 2D structure information in images. Zeng et al. [17] proposed QPCANet that employs Quaternion PCA for learning the filters leading to a better recognition accuracy on color images compared to PCANet. Yang et al. [18] proposed CCANet to process two different views of images by Canonical Correlation Analysis in a convolutional architecture. CCANet may lead to better recognition accuracy compared to PCANet that only uses a single view of images. In [19], the K-means clustering algorithm is employed for training a Compact Unsupervised Network (CUNet). In CUNet, a sequence of a Rectified Linear unit and a weighted pooling operation is applied after each convolutional layer for better processing of nonlinear data and making the model robust to local translations of the input images.

Some methods tried to enhance the performance of PCANet by modifying the core structure of PCANet. In [20, 21], pooling operations were added in PCANet to provide more robustness. Tian et al. [22] removed binary hashing and block-wise histograms and proposed second-order pooling by patch-wise covariance matrices in order to reduce the dimension of the final feature vectors and preserve the information provided by the floating-point format.

Similar to our method, some researchers proposed nonlinear algorithms to learn the convolutional filters. In SPCANet, binary hashing and block-wise histograms are computed after each PCA layer to embed nonlinearity in every stage [23]. Two methods employed Extreme Learning Machine-Autoencoder (ELM-AE) in place of PCA to learn convolutional filters [24, 25]. In ELM-AE networks, first, each patch is mapped to a feature space by a nonlinear

randomized transformation. Then a linear transformation is applied to the patches in the feature space in a way that the distance between the projected and the input patches is minimized. Our proposed model and ELM-AE network differ in the objective functions used for training the filters. The former employs Kernel Principal Component Analysis with explicit feature maps and the latter employs Extreme Learning Machine-Autoencoder [26].

Some methods utilized kernel trick in PCANet for nonlinear processing of image patches [27, 28, 29]. However, all these methods suffer from high computational complexity and the need for a lot of memory since they explicitly construct the kernel matrix and compute its eigenvectors.

In recent years, there have been some researches on investigating the connection between kernel methods and neural networks through kernel approximation techniques. These methods usually make a stack of kernel feature spaces using explicit feature maps, which can be interpreted as a multi-layer neural network. For example, in Convolutional Kernel Networks (CKNs) it has been shown that the feature map of a convolution kernel is similar to the repeating module of CNNs [4, 30]. Also in kernel deep convex networks, the RFF method is used for approximating the kernel ridge regression problem in each stage of this model [31]. Although, similar to RNPCANet, these methods train a multi-layer model in a feedforward manner using kernel approximation techniques, RNPCANet differs from these models as it is a novel method for training convolutional networks by using the simple PCA method equipped with a kernel function.

3. Kernel Approximation using Random Fourier Features

In this section, first, we define the concept of the kernel trick and Bochner's Theorem for shift-invariant kernels. Then we describe the RFF method that uses Bochner's Theorem to approximate shift-invariant kernels to reduce the computations in kernel methods. We start with the definition of the kernel trick.

Definition 1- Kernel Trick. It is known that corresponding to every pos-

itive definite kernel function $k : X \times X \to \mathbb{R}$ there exists a potenftially infinite-dimensional Hilbert space $\mathcal{H}$ and a mapping $\phi : X \to \mathcal{H}$ from the input space to the feature space such that for every pair $x, z \in X$ we have $k(x, z) = \langle \phi(x), \phi(z) \rangle_{\mathcal{H}}$ [6]. If the operations in a linear machine learning algorithm are solely expressed in terms of the inner-product between samples, one can substitute a kernel function for the inner-product to implicitly perform the machine learning algorithm in the feature space. This turns the linear machine learning algorithm into a nonlinear one.

As we mentioned in Section 1, the major limitation of kernel methods is that these methods are time-consuming. One way to reduce computations in kernel methods is approximating the kernel functions with explicit feature maps. In the following, we describe Bochner's theorem and the RFF method that uses Bochner's theorem to approximate shift-invariant kernels.

Theorem 1- Bochner's Theorem [32]. A continuous shift-invariant kernel $K(x, y) = K(x - y)$ is positive definite if and only if K is the inverse Fourier transform of a non-negative measure:

$$K(x - y) = \int_{\mathbb{R}^d} P(w) e^{jw^T (x-y)} dw \tag{1}$$

To approximate Equation 1, RFF defines the mapping $z : \mathbb{R}^d \to \mathbb{R}^D$ as follows:

$$z(x) = \sqrt{\frac{2}{D}} [\cos(w_l^T x + b_l)]_{l=1}^D, \tag{2}$$

where w_l is drawn from $P(w)$ distribution and b_l is uniformly sampled from $[0, 2\pi]$. Considering Equations 1 and 2, one can prove that the mapping z in Equation 2 is an approximated explicit feature map of the shift invariant kernel K [9], i.e.:

$$K(x - y) \approx < z(x), z(y) > \tag{3}$$

One of the popular shift-invariant kernels is the Gaussian kernel, which has the following form:

$$K(x, y) = \exp(-\frac{1}{\sigma^2} ||x - y||_2^2), \tag{4}$$

where σ is the Gaussian kernel width. The Fourier transform of the Gaussian kernel has a normal distribution $p(w) = \mathcal{N}(0, \frac{1}{\sigma^2} I)$.

Consider a linear machine learning model A and its equivalent kernelized version, denoted by KA, employing a shift-invariant kernel K. As stated above, the mapping $z : X \to F$ from input space X to feature space F obtained by RFF is the approximated feature map of k. Therefore, performing A in the feature space F is approximately equivalent to performing KA in the input space:

$$KA(x) \approx A(z(x)), \tag{5}$$

where $x \in X$.

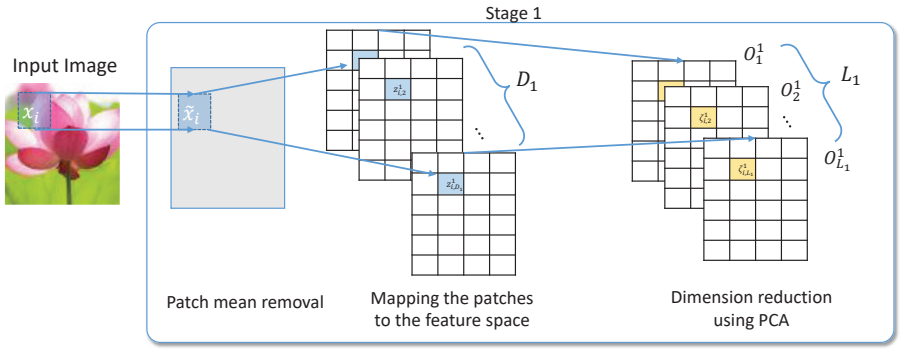
Using explicit feature maps instead of an exact kernel function significantly reduces computations in kernel methods [9]. The effectiveness of using explicit feature maps for the approximation of kernel PCA has been discussed in [10].

4. The Proposed Model

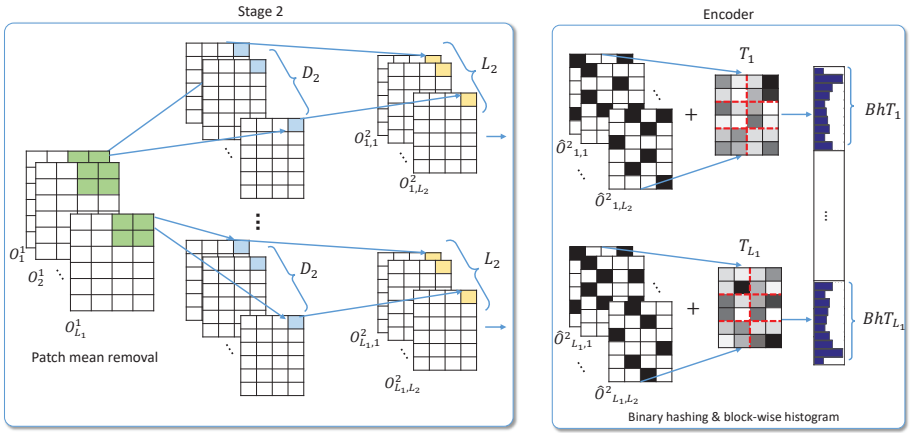
In this section, first, we describe the structure of RNPCANet, which is very similar to the structure of PCANet with an extra convolutional layer followed by sinusoidal nonlinearity in each stage. Second, we discuss the computational complexity of RNPCANet. Finally, we discuss the relationship between RNPCANet and Convolutional Kernel Networks (CKN). We show that the proposed RNPCANet is a type of CKN, in which each convolutional kernel map is followed by a PCA layer.

4.1. The Structure of RNPCANet

RNPCANet is a kernelized version of PCANet which uses an approximation of KPCA by explicit kernel feature maps for learning the filters. RNPCANets consist of multiple stages of successive convolutional and PCA layers, which are finally followed by an encoding step. The training is done layer by layer in a feedforward manner. In the following subsections, we describe RNPCANet architecture and the operations in each stage of RNPCANet required for training and testing this model.



(a)



(b) (c)

Figure 1: The structure of a 2-stage RNPCANet. (a) stage 1. (b) stage 2. (c) encoder.

4.1.1. First Stage

For training the first stage of RNPCANet (Figure 1(a)), assume $X^1 = [x_1^1, ..., x_n^1] \in \mathcal{R}^{k_1^1 k_2^1 \times n_1}$ contains n_1 vectorized patches of size $k_1^1 \times k_2^1$ extracted from training images around each pixel. In the first step, we subtract the mean of every patch to obtain $\tilde{X}^1 = [\tilde{x}_1^1, ..., \tilde{x}_{n_1}^1]$. Then we map every zero-meaned patch to a D_1-dimensional explicit kernel feature space using the RFF method described in Section 3. The matrix $Z^1 = [z_1^1, ..., z_n^1] \in \mathcal{R}^{D_1 \times n_1}$ contains patches mapped to the feature space, where z_i^1 is obtained as follows:

$$z_i^1 = \sqrt{\frac{2}{D_1}}[\cos({w_l^1}^T \tilde{x}_i^1 + b_l^1)]_{l=1}^{D_1}, \tag{6}$$

and $w_l^1 \in \mathcal{R}^{k_1^1 k_2^1 \times 1}$ is drawn from $P_1(w)$ that is the Fourier transform of a predefined kernel K_1 in the current layer. Also, b_l^1 is drawn uniformly from $[0, 2\pi]$. We call w_l^1 and b_l^1 the parameters of the lth filter of the first layer of RNPCANet.

The next step is to reduce the dimension of the patches mapped to the feature space to L_1 ($L_1 < D_1$). In order to do this, the same procedure with PCANet is performed by learning a projection matrix called $V_{D_1 \times L_1}^1$ using the PCA objective function as follows:

$$\min_{V^1} ||Z^1 - V^1 {V^1}^T Z^1||_2, s.t. {V^1}^T V^1 = I_{L_1}, \tag{7}$$

where I_{L_1} is the identity matrix of size L_1. The optimal value of V_1 in Equation 7 is obtained by the eigen-decomposition of $\frac{1}{n_1} Z^1 {Z^1}^T$. More precisely, the columns of V^1 are the eigenvectors of $\frac{1}{n_1} Z^1 {Z^1}^T$ corresponding to the L_1 largest eigenvalues.

After training the first stage of RNPCANet, for each patch x_t extracted from an arbitrary input image, the output of the first stage of RNPCANet is computed by 1) subtracting the mean of the patch resulting in $\tilde{x}_t$, 2) mapping the zero-meaned patch $\tilde{x}_t$ to the feature space by Equation 6 to obtain z_t^1, and 3) reducing the dimension of z_t^1 using the following equation:

$$\zeta_t^1 = {V^1}^T z_t^1 \tag{8}$$

As it is illustrated in Figure 1(a), $\zeta_t^1 \in \mathcal{R}^{L_1}$, where elements of the vector ζ_t^1 are placed in separate output planes $O_1^1, ..., O_{L_1}^1$ of stage 1.

4.1.2. Second Stage

The filters and the projection matrices in other stages of RNPCANet are obtained in the same way as the first stage by using the patches extracted from the output of their previous stage.

One difference between RNPCANet and ordinary CNNs is that each filter of layer 2 or deeper layers operates on a plane. It means that if the projection matrix of the first stage (V^1) has L_1 columns and the projection matrix of the second stage (V^2) has L_2 columns, then the output of stage 2 will have $L_1 \times L_2$ planes (see Figure 1(b)). More precisely, for training the second stage of RNPCANet, let's assume $O_1^1, ..., O_{L_1}^1$ are the output planes of the first stage of RNPCANet and n_2 patches of size $k_1^2 \times k_2^2$ are extracted from each plane. Matrix $X^2 \in \mathcal{R}^{k_1^2 k_2^2 \times n_2 L_1}$ contains all the patches extracted from each plane $O_1^1, ..., O_{L_1}^1$ and $\tilde{X}^2 \in \mathcal{R}^{k_1^2 k_2^2 \times n_2 L_1}$ contains zero-meaned patches. The parameters $[w_l^2]_{l=1}^{D_2}$ are drawn from $P_2(w)$ which is the Fourier transform of the predefined kernel K_2 in the second stage and the parameters $[b_l^2]_{l=1}^{D_2}$ are drawn uniformly from $[0, 2\pi]$. Similar to Equation 6, these parameters are used to map the zero-meaned patches to the feature space to form the matrix $Z^2 \in \mathcal{R}^{D_2 \times n_2}$. After mapping each zero-meaned patch to the feature space, the projection matrix $V^2 \in \mathcal{R}^{D_2 \times L_2}$ is formed by L_2 eigenvectors of $\frac{1}{n_2} Z^2 Z^{2^T}$.

After the training is done, the output of stage 2 for each patch extracted from separate planes $O_1^1, ..., O_{L_1}^1$ is computed similar to the stage 1 by using the parameters $[w_l^2]_{l=1}^{D_2}$ and $[b_l^2]_{l=1}^{D_2}$ and the projection matrix V^2.

4.1.3. Encoding step

After several predetermined stages (including convolutional, nonlinearity, and PCA layers), in the final step, we perform a binary hashing and compute block-wise histograms to encode features (Figure 1(c)). Consider a two-stage RNPCANet followed by an encoder with L_1 planes in the output of the first

stage and $L_1 \times L_2$ planes in the output of the second stage. To perform the binary hashing and compute block-wise histograms, first, we binarize the output of the second stage using a Heaviside step function. Then we divide the binarized planes into L_1 groups, each of them corresponding to a plane in the previous stage, and compute the weighted sum of all the planes in each group as follows:

$$T_j = \sum_{l=1}^{L_2} 2^{l-1} \hat{O}_{j,l}^2, \tag{9}$$

where $\hat{O}_{j,l}^2$ represents the binarized lth plane in the jth group of the output of layer 2. After the binary hashing step, we compute block-wise histograms. In order to do this, each T_j , $j = 1, ..., L_1$ is partitioned into B blocks, which may have overlaps, and the histogram in each block is computed using 2^{L_2} bins. Furthermore, the histograms of T_j are concatenated into a vector represented by BhT_j. Finally, $BhT_1, ..., BhT_{L_1}$ are concatenated to form the final feature vector of the input image as follows:

$$f = [BhT_1, \ldots, BhT_{L_1}] \tag{10}$$

Eventually, we may use the features extracted by RNPCANet to train any classifier, e.g. SVM.

4.2. Computational Complexity

In this subsection, we compute the computational complexity of a two-stage RNPCANet. Extending this computational complexity to T stages is straightforward. We show that, although RNPCANet utilizes a kernel feature space for nonlinear processing of input data of each stage, the computational complexity of this model is only slightly higher than that of PCANet. Assume that the number of patches extracted from an input plane at each stage is m (clearly, the input plane of stage one is the original image). Herein, we use the following notations, which are similar to Section 4. The number of filters in stage i is denoted by D_i and the size of each patch is $k_1 \times k_2$. Also, the number of principal components in the PCA layer of stage i is denoted by L_i.

In stage i, removing the mean of the patches needs $\mathcal{O}(k_1k_2+k_1k_2m)$ computations. Since the convolution filters are assigned randomly, the cost of initializing the filters is $\mathcal{O}(D_ik_1k_2)$, which is negligible compared to the $\mathcal{O}(D_ik_1k_2m)$ cost of the convolution and nonlinearity operations. The complexity of generating the covariance matrix ZZ^T is $\mathcal{O}(2D_i^2m)$ and the cost of eigen-decomposition of this matrix is $\mathcal{O}(D_i^3)$. Reducing the dimension of the patches mapped to the feature space in the PCA layer requires $\mathcal{O}(L_iD_im)$ computations.

In the final stage, the complexity of binary hashing and block-wise histograms are $\mathcal{O}(2L_2m)$ and $\mathcal{O}(mBL_2\log 2)$ respectively, where B is the number of partitioning blocks. Assuming $m >> \max(k_1, k_2, L_1, L_2, B)$, the overall complexity of RNPCANet is derived as follows:

$$\mathcal{O}((D_1L_1 + D_2L_2)m + (D_1^2 + D_2^2)m + (D_1 + D_2)k_1k_2m) \tag{11}$$

Assuming $D_1 > k_1k_2$ and $D_2 > k_1k_2$, the third component in Equation 11 has the upper-bound $(D_1^2+D_2^2)m$. Therefore, the following computational complexity can be derived from Equation 11:

$$\mathcal{O}((D_1L_1 + D_2L_2)m + (D_1^2 + D_2^2)m) \tag{12}$$

Furthermore, since $D_i \geq L_i$, the term $(D_1L_1 + D_2L_2)m$ has the extreme value of $(D_1^2 + D_2^2)m$ and the computational complexity of RNPCANet can be summarized as follows:

$$\mathcal{O}((D_1^2 + D_2^2)m) \tag{13}$$

In practice we set $D = D_1 = D_2$ which leads our model to have a complexity of $\mathcal{O}(D^2m)$. According to [5], the computational complexity of a two-stage PCANet is $\mathcal{O}((k_1k_2)^2m)$. Since we set D only slightly larger than k_1k_2, it turns out that the computational complexity of RNPCANet is a bit higher than that of PCANet.

For computing the computational complexity of a T-stage RNPCANet we can modify Equation 11 as follows:

$$\mathcal{O}\Big(\sum_{i=1}^{T} D_iL_im + \sum_{i=1}^{T} D_i^2m + \sum_{i=1}^{T} D_ik_1k_2m\Big). \tag{14}$$

The same procedure as we did for a 2-stage RNPCANet can be applied to the above equation which results in $\mathcal{O}(TD^2m)$ computational complexity for RNPCANet. Furthermore, the computational complexity of a T-stage PCANet is $\mathcal{O}(T(k_1k_2)^2m)$ which is a bit lower than that of RNPCANet.

We should note that the computational complexity of using the exact kernel is cubic in terms of the number of patches, which is far higher than the computational complexity of RNPCANet.

4.3. Relationship to Convolutional Kernel Networks

Convolutional Kernel Networks (CKNs) are a type of CNN, which are trained with the goal of approximating a convolution kernel [4]. This convolution kernel has the following form:

$$K(I, I^{'}) = \sum_{z\in\Omega}\sum_{z'\in\Omega} e^{-\frac{1}{2\beta^2}\|z-z'\|^2} e^{-\frac{1}{2\sigma^2}\|P_z - P'_{z'}\|^2}, \tag{15}$$

where I and I' represent two input images and Ω contains the pixel coordinates of image patch centers. Furthermore, P_z and $P'_{z'}$ are two patches centered at z and z' which are extracted from I and I' respectively. Approximating the convolution kernel in Equation 15 by a convex combination of cosine kernels as described in [30], leads to the following explicit mapping:

$$\phi(z) = \sum_{u\in\Omega} e^{-\frac{1}{2\beta^2}\|z-u\|^2} \sqrt{1/D}[\cos(w_l^T P_u + b_l)]_{l=1}^{D}, \tag{16}$$

where $[w_l]_{l=1}^{D}$ and $[b_l]_{l=1}^{D}$ represent filters and biases respectively. In the limit, as β approaches zero, the mapping of CKN will be as follows:

$$\phi(z) = \sqrt{2/D}[\cos(w_l^T P_z + b_l)]_{l=1}^{D} \tag{17}$$

In this case, the mapping of CKN is exactly the same as the convolution and the nonlinearity operations of RNPCANet in Equation 6. Two differences between RNPCANet and CKNs are 1) convolution and nonlinearity operations in RNPCANet is followed by a PCA layer, and 2) there is an encoder in the last stage of RNPCANet based on binary hashing and block-wise histograms.

5. Experiments

In this section, we evaluate RNPCANet on several image classification datasets and compare this model with PCANet and three variants of CKN models, which are all unsupervised convolutional models. [4, 30]. The PCANet and RNPCANet models that are used in this section have two stages. In these models, the number of planes in the output of the first stage and the second stage are 8 and 64 respectively. Also, the number of filters in both stages is 50 and the size of filters is 7×7. The size of the blocks in the encoder is set to 7×7 and the overlap ratio of blocks is 0.5. We use a Gaussian kernel in order to compute explicit feature maps in each stage of RNPCANet. We use the same heuristic as [4] for computing the kernel width in each stage of RNPCANet. To this end, we randomly extract some patches from the input planes and set the kernel width to the q-quantile of the distances between the patches, where q depends on the number of classes and how much the classes are balanced. We use linear SVM [33] to classify the extracted feature vectors. The regularization parameter of SVM is set to 1 for RNPCANet and PCANet and is searched from $[2^{-15}, ..., 2^{15}]$ in CKN models. For those datasets without a predetermined training/test split, we use random sub-sampling for splitting the training and test data. The percentage of data used for training in each dataset is the same with other methods on these datasets and is specified in the corresponding subsection. All results are averaged over 10 runs.

Our experiments consist of two parts. First, we run evaluations on object classification datasets [34] Coil-20, Coil-100 [34], ETH-80 [35], Caltech-101 [36]. We also evaluate RNPCANet on handwritten character recognition datasets MNIST [1] and C-Cube [37]. Second, we evaluate the impact of the Gaussian kernel width on accuracy.

5.1. Classification Results on Coil-20 and Coil-100

In this subsection, we evaluate RNPCANet on object classification datasets Coil-20 and Coil-100 [34], which both consist of controlled-environment images.

Every image is taken after rotating the object by 5° in a 360° turnable environment producing 72 images per object. Coil-20 consists of $1,440$ grayscale images of size 128×128 of 20 objects. Figure 2 illustrates several samples of Coil-20. We resized images to 32×32 and randomly selected $\frac{1}{6}$ of images as training

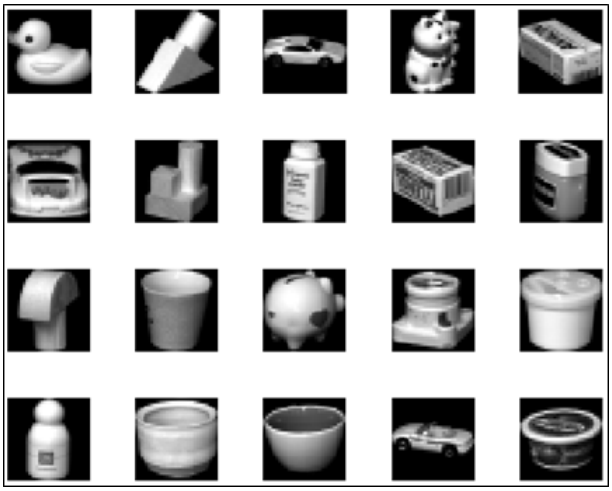


Figure 2: Some examples of resized images of the Coil-20 dataset.

and the rest as testing part. Table 1 shows the comparison of RNPCANet with PCANet and CKNs in terms of recognition accuracy and training time on Coil-20. The classification results show that the recognition accuracy of RNPCANet is 2.09% higher than PCANet.

Table 1: Classification results of RNPCANet, PCANet, and CKN models on Coil-20

Models	Accuracy (%)	Time (h)
Gaussian-CKN [4]	94.32 ± 1.40	55.23
Cos-CKN-RF [30]	94.25 ± 1.92	0.81
Cos-CKN-OP [30]	94.08 ± 1.53	47.88
PCANet	89.98 ± 1.30	0.87
RNPCANet	92.07 ± 1.89	1.25

We performed the same experiments on Coil-100 dataset. This dataset contains $7,200$ color images of size 128×128 of 100 different objects. Some samples of Coil-100 are depicted in Figure 3. We divided Coil-100 images in the same

manner as we did on Coil-20.

Table 2: Classification results of RNPCANet, PCANet, and CKN models on Coil-100

Models	Accuracy (%)	Time (h)
Gaussian-CKN [4]	90.26 ± 0.95	28.93
Cos-CKN-RF [30]	90.02 ± 1.35	7.93
Cos-CKN-OP [30]	90.15 ± 1.12	30.33
PCANet	88.23 ± 0.76	8.55
RNPCANet	90.35 ± 1.15	12.27

The classification results are shown in Table 2. The results show that the recognition accuracy of RNPCANet is 2.12% higher than PCANet and superior to the CKN models.

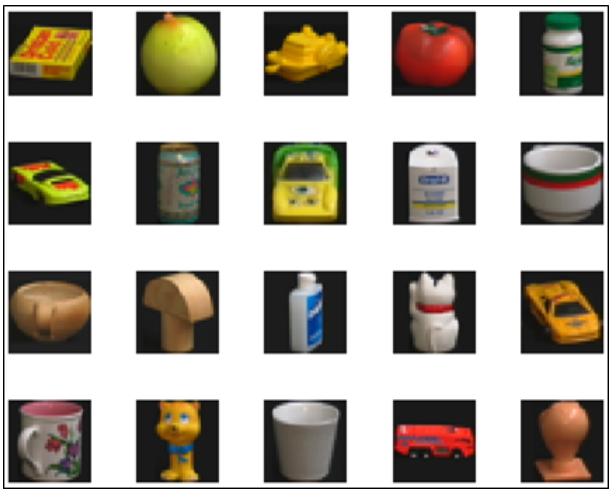


Figure 3: Some examples of resized images of the Coil-100 dataset.

5.2. Classification Results on Caltech-101

Caltech-101 contains $8,677$ color images of 101 different objects. The number of images in each category varies from 31 to 800. Some samples of Caltech-101 dataset are depicted in Figure 4. We resized all images to 128×128. For each run of the models, we randomly select 30 images from each class as training data

and the rest as testing set. The comparison of RNPCANet with other models on Caltech-101 is shown in Table 3.

Figure 4: Some examples of resized images of the Caltech-101 dataset.

As it is presented in the table, two CKN models, Gaussian-CKN and Cos-CKN-OP achieve higher accuracies on Caltech-101 than PCANet and RNPCANet. However, RNPCANet still outperforms PCANet on this dataset. We should note that those CKN models with higher recognition accuracies use a time-consuming unsupervised method for approximating the kernel function in each layer of the models while RNPCANet uses a randomized one.

Table 3: Classification results of RNPCANet, PCANet, and CKN models on Caltech-101

Models	Accuracy (%)	Time (h)
Gaussian-CKN [4]	72.92 ± 1.05	38.94
Cos-CKN-RF [30]	70.40 ± 1.15	15.20
Cos-CKN-OP [30]	72.62 ± 1.15	39.94
PCANet	71.78 ± 0.98	1.42
RNPCANet	72.27 ± 1.02	1.56

5.3. Classification Results on ETH-80

ETH-80 consists of eight super-categories each of them with ten subcategories of color images. Each subcategory has 41 different views of an object. We resized all images to 128×128. Some samples of ETH-80 is illustrated in Figure 5. For each super-category, we randomly select two subcategories as

Figure 5: Some examples of resized images of the ETH-80 dataset.

training and the rest eight subcategories as test data. A comparison of RNPCANet with PCANet and CKN models is shown in Table 4. The results show that the RNPCANet model surpasses the other models on ETH-80.

Table 4: Classification results of RNPCANet, PCANet, and CKN models on ETH-80

Models	Accuracy (%)	Time (h)
Gaussian-CKN [4]	65.40 ± 5.83	6.95
Cos-CKN-RF [30]	64.03 ± 7.03	0.37
Cos-CKN-OP [30]	67.31 ± 6.76	15.62
PCANet	72.01 ± 10.34	0.10
RNPCANet	74.96 ± 9.18	0.11

5.4. Handwritten Character Classification

In this subsection, we perform experiments on two handwritten character recognition datasets MNIST [1] and C-Cube [37].

The MNIST dataset consists of $60,000$ training and $10,000$ testing handwritten images of $0-9$ digits. All images are grayscale with the size of 28×28. Some samples of MNIST images are depicted in Figure 6.

The classification results on MNIST images with PCANet, RNPCANet, and CKN models are shown in Table 5. As the results show, there is no significant difference between the performance of all the models on MNIST. However,

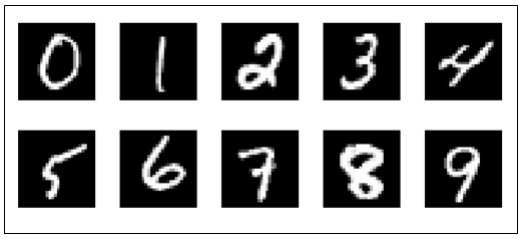


Figure 6: Some examples of MNIST digits.

RNPCANet performs slightly better than PCANet on this dataset.

Table 5: Classification results of RNPCANet, PCANet, and CKN models on MNIST

Models	Accuracy (%)	Time (h)
Gaussian-CKN [4]	99.42 ± 0.06	2.90
Cos-CKN-RF [30]	99.40 ± 0.07	0.25
Cos-CKN-OP [30]	99.44 ± 0.06	2.13
PCANet	99.40 ± 0.04	14.03
RNPCANet	99.42 ± 0.09	15.83

The C-Cube dataset consists of handwritten cursive character images of the English alphabet from 'a' to 'z' and from 'A' to 'Z'. This dataset contains $38,160$ training and $19,133$ testing binary images. Several samples of C-Cube images are depicted in Figure 7. We resized all images to 32×32. The results of classifying C-Cube images using RNPCANet, PCANet, and CKN models are illustrated in Table 6. Although RNPCANet performs marginally better than PCANet, the results show that the recognition accuracy of CKN models is superior to the other models on C-Cube.

Table 6: Classification results of RNPCANet, PCANet, and CKN models on C-Cube

Models	Accuracy (%)	Time (h)
Gaussian-CKN [4]	83.36 ± 1.12	3.02
Cos-CKN-RF [30]	84.03 ± 1.32	0.26
Cos-CKN-OP [30]	84.07 ± 0.95	2.18
PCANet	81.58 ± 0.61	13.92
RNPCANet	81.79 ± 0.82	16.30

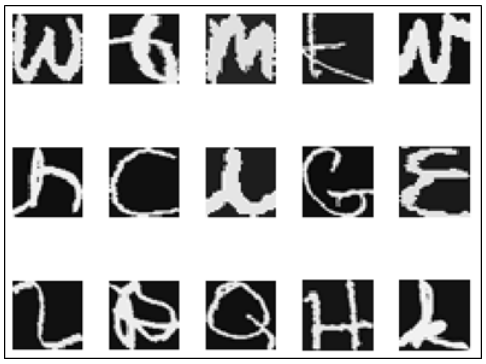


Figure 7: Some examples of C-Cube characters.

5.5. Impact of Gaussian Kernel Width

As we mentioned in Section 4, RNPCANet utilizes explicit feature maps of a shift-invariant kernel (Gaussian kernel in our experiments) in each stage to map the patches into a feature space. In this subsection, we evaluate the impact of the Gaussian kernel width of RNPCANet on two datasets, Coil-20 and Coil-100. The results are illustrated in Figure 8. The Gaussian kernel width of both layers are the same and ranged from 0.2 to 2. The results show that by properly choosing the kernel width, the recognition accuracy can increase significantly. However, the recognition accuracies of RNPCANet are almost higher than PCANet even in the worst cases of RNPCANet.

5.6. Statistical comparison of RNPCANet and PCANet

For clarifying the difference between the performance of RNPCANet and PCANet, we summarize the experimental results obtained by these two models in Table 7.

The number of datasets in our experiments is not enough to ensure that the shape of the differences between the performances of the two models follows a normal distribution. Therefore, we use the Wilcoxon signed-ranks test which is a non-parametric alternative to the paired t-test [38] to see whether the performances of the two models are significantly different on the aforementioned datasets. Using the table of critical values for the Wilcoxon test, one can

Table 7: Summary of the results for PCANet and RNPCANet

Dataset	PCANet (%)	RNPCANet (%)
Coil-20	89.98	92.07
Coil-100	88.23	90.35
Caltech-101	71.78	72.27
ETH-80	72.01	74.96
MNIST	99.40	99.42
C-Cube	81.58	81.79

determine how much the performances of the two models are statistically significantly different. Since RNPCANet is superior to PCANet in the five datasets in Table 7, RNPCANet performs better than PCANet with 95% confidence in one-sided Wilcoxon test.

6. Conclusion and future work

In this paper, we proposed a nonlinear PCA network called RNPCANet, which is a multi-stage convolutional neural network that simply uses a kernelized PCA to learn the weights of the network. RNPCANet extends the training algorithm of PCANet to a nonlinear case by using kernel functions while the simplicity of the training method is preserved. We showed that by using a randomized kernel approximation technique instead of an exact kernel, the computational complexity of RNPCANet is comparable with that of PCANet. Furthermore, we discussed the relationship between RNPCANet and convolutional kernel networks. We showed that RNPCANet is a convolutional kernel network that has a PCA layer after each convolutional layer. We evaluated RNPCANet on several image recognition datasets. The results demonstrated that one can gain higher recognition accuracies by using nonlinear processing of data in a PCANet architecture. We plan to investigate the effectiveness of employing kernel approximation techniques in other convolutional neural networks similar to PCANet, such as MLDNet and CCANet. Another direction for future work is to make RNPCANet more robust to data irregularities in real-world data.

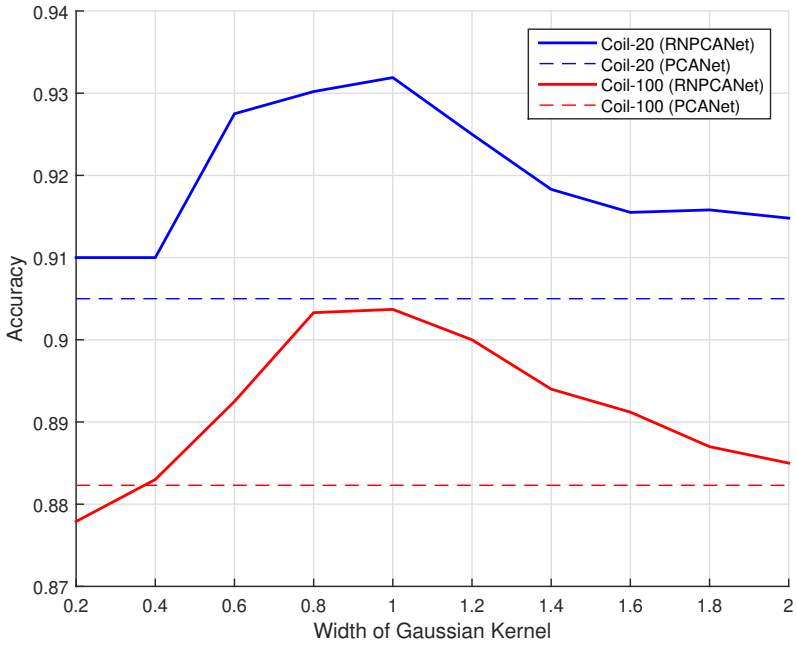


Figure 8: Impact of Gaussian kernel width on the accuracy of RNPCANet on Coil-20 and Coil-100 datasets. To check whether the performance of RNPCANet is significantly better than PCANet, we used a paired t-test for comparing the results obtained for ten different kernel widths. The two-tailed p-value is less than 0.0002 on Coil-20 and less than 0.0093 on Coil-100, which is considered to be very statistically significant.

Various data Irregularities may exist in real-world datasets such as class imbalance and skewed class-distribution [39]. In the case of class imbalance, a balanced sample set can be used in the (kernel) PCA method by over-sampling the minority classes and/or under-sampling the majority classes as it is done in [40]. Also, the problem of class distribution skew can be mitigated by using classifiers that pay more attention to the local structure of data.

References

[1] LeCun Y, Bottou L, Bengio Y, Haffner P. Gradient-based learning applied to document recognition. Proceedings of the IEEE 1998;86(11):2278–324.

[2] Zeiler MD, Fergus R. Visualizing and understanding convolutional networks. In: European conference on computer vision. Springer; 2014, p. 818–33.

[3] Bruna J, Mallat S. Invariant scattering convolution networks. IEEE transactions on pattern analysis and machine intelligence 2013;35(8):1872–86.

[4] Mairal J, Koniusz P, Harchaoui Z, Schmid C. Convolutional kernel networks. In: Advances in neural information processing systems. 2014, p. 2627–35.

[5] Chan TH, Jia K, Gao S, Lu J, Zeng Z, Ma Y. Pcanet: A simple deep learning baseline for image classification? IEEE Transactions on Image Processing 2015;24(12):5017–32.

[6] Schölkopf B, Smola A, Müller KR. Nonlinear component analysis as a kernel eigenvalue problem. Neural computation 1998;10(5):1299–319.

[7] Keerthi SS, Lin CJ. Asymptotic behaviors of support vector machines with gaussian kernel. Neural computation 2003;15(7):1667–89.

[8] Burges CJ. A tutorial on support vector machines for pattern recognition. Data mining and knowledge discovery 1998;2(2):121–67.

[9] Rahimi A, Recht B. Random features for large-scale kernel machines. In: Advances in neural information processing systems. 2008, p. 1177–84.

[10] Lopez-Paz D, Sra S, Smola A, Ghahramani Z, Schölkopf B. Randomized nonlinear component analysis. In: International Conference on Machine Learning. 2014, p. 1359–67.

[11] Ullah E, Mianjy P, Marinov TV, Arora R. Streaming kernel pca with $\tilde{O}(\sqrt{n})$ random features. In: Advances in Neural Information Processing Systems. 2018, p. 7311–21.

[12] Sriperumbudur B, Sterge N. Approximate kernel pca using random features: Computational vs. statistical trade-off. arXiv preprint arXiv:170606296 2017;.

[13] Ng CJ, Teoh ABJ. Dctnet: a simple learning-free approach for face recognition. In: Signal and Information Processing Association Annual Summit and Conference (APSIPA), 2015 Asia-Pacific. IEEE; 2015, p. 761–8.

[14] Feng Z, Jin L, Tao D, Huang S. Dlanet: a manifold-learning-based discriminative feature learning network for scene classification. Neurocomputing 2015;157:11–21.

[15] Zeng R, Wu J, Senhadji L, Shu H. Tensor object classification via multilinear discriminant analysis network. In: Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on. IEEE; 2015, p. 1971–5.

[16] Jia Z, Han B, Gao X. 2dpcanet: Dayside aurora classification based on deep learning. In: CCF Chinese Conference on Computer Vision. Springer; 2015, p. 323–34.

[17] Zeng R, Wu J, Shao Z, Chen Y, Chen B, Senhadji L, et al. Color image classification via quaternion principal component analysis network. Neurocomputing 2016;216:416–28.

[18] Yang X, Liu W, Tao D, Cheng J. Canonical correlation analysis networks for two-view image recognition. Information Sciences 2017;385:338–52.

[19] Izquierdo E. Cunet: A compact unsupervised network for image classification. IEEE Transactions on Multimedia 2018;20(8).

[20] Liu Y, Zhao S, Wang Q, Gao Q. Learning more distinctive representation by enhanced pca network. Neurocomputing 2018;275:924–31.

[21] Low CY, Teoh ABJ, Toh KA. Stacking pcanet+: an overly simplified convnets baseline for face recognition. IEEE Signal Process Lett 2017;24:1581–5.

[22] Tian L, Hong X, Zhao G, Fan C, Ming Y, Pietikäinen M. Pcanet-ii: When pcanet meets the second order pooling. arXiv preprint arXiv:171000166 2017;.

[23] Tian L, Fan C, Ming Y, Jin Y. Stacked pca network (spcanet): an effective deep learning for face recognition. In: Digital Signal Processing (DSP), 2015 IEEE International Conference on. IEEE; 2015, p. 1039–43.

[24] Zhu W, Miao J, Qing L, Huang GB. Hierarchical extreme learning machine for unsupervised representation learning. In: Neural Networks (IJCNN), 2015 International Joint Conference on. IEEE; 2015, p. 1–8.

[25] Cui D, Zhang G, Han W, Kasun LLC, Hu K, Huang GB. Compact feature representation for image classification using elms. In: Computer Vision Workshop (ICCVW), 2017 IEEE International Conference on. IEEE; 2017, p. 1015–22.

[26] Kasun LLC, Zhou H, Huang GB, Vong CM. Representational learning with extreme learning machine for big data. IEEE intelligent systems 2013;28(6):31–4.

[27] Wu D, Wu J, Zeng R, Jiang L, Senhadji L, Shu H. Kernel principal component analysis network for image classification. arXiv preprint arXiv:151206337 2015;.

[28] Pan B, Shi Z, Zhang N, Xie S. Hyperspectral image classification based on nonlinear spectral–spatial network. IEEE Geoscience and Remote Sensing Letters 2016;13(12):1782–6.

[29] Ding C, Bao T, Karmoshi S, Zhu M. Single sample per person face recognition with kpcanet and a weighted voting scheme. Signal, Image and Video Processing 2017;11(7):1213–20.

[30] Mohammadnia-Qaraei MR, Monsefi R, Ghiasi-Shirazi K. Convolutional kernel networks based on a convex combination of cosine kernels. Pattern Recognition Letters 2018;116:127–34.

[31] Huang PS, Deng L, Hasegawa-Johnson M, He X. Random features for kernel deep convex network. In: 2013 IEEE International Conference on Acoustics, Speech and Signal Processing. IEEE; 2013, p. 3143–7.

[32] Bochner S. Monotone funktionen, stieltjessche integrale und harmonische analyse. Mathematische Annalen 1933;108(1):378–410.

[33] Fan RE, Chang KW, Hsieh CJ, Wang XR, Lin CJ. Liblinear: A library for large linear classification. Journal of machine learning research 2008;9(Aug):1871–4.

[34] Nene SA, Nayar SK, Murase H, et al. Columbia object image library (coil-20) 1996;.

[35] Leibe B, Schiele B. Analyzing appearance and contour based methods for object categorization. In: 2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.; vol. 2. IEEE; 2003, p. II–409.

[36] Fei-Fei L, Fergus R, Perona P. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. In: 2004 conference on computer vision and pattern recognition workshop. IEEE; 2004, p. 178–.

[37] Camastra F, Spinetti M, Vinciarelli A. Offline cursive character challenge: a new benchmark for machine learning and pattern recognition algorithms. In: Pattern Recognition, 2006. ICPR 2006. 18th International Conference on; vol. 2. IEEE; 2006, p. 913–6.

[38] Demšar J. Statistical comparisons of classifiers over multiple data sets. Journal of Machine learning research 2006;7(Jan):1–30.

[39] Das S, Datta S, Chaudhuri BB. Handling data irregularities in classification: Foundations, trends, and future challenges. Pattern Recognition 2018;81:674–93.

[40] Zhang X, Su H, Zhang C, Atkinson PM, Tan X, Zeng X, et al. A robust imbalanced sar image change detection approach based on deep difference image and pcanet. arXiv preprint arXiv:200301768 2020;.