Pattern Recognition 100 (2020) 107176

Contents lists available at ScienceDirect

Pattern Recognition

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

Deep eigen-filters for face recognition: Feature representation via unsupervised multi-structure filter learning

Ming Zhang[a,*], Sheheryar Khan[b], Hong Yan[a]

[a] Department of Electrical Engineering, City University of Hong Kong, Hong Kong, China
[b] Department of Imaging and Interventional Radiology, The Chinese University of Hong Kong, Hong Kong, China

ARTICLE INFO

Article history:
Received 3 September 2019
Revised 28 November 2019
Accepted 15 December 2019
Available online 16 December 2019

Keywords:
Deep eigen-filters
Convolution kernels
Face recognition
Convolutional neural networks
Feature representation

ABSTRACT

Training deep convolutional neural networks (CNNs) often requires high computational cost and a large number of learnable parameters. To overcome this limitation, one solution is computing predefined convolution kernels from training data. In this paper, we propose a novel three-stage approach for filter learning alternatively. It learns filters in multiple structures including standard filters, channel-wise filters and point-wise filters which are inspired from variations of CNNs' convolution operations. By analyzing the linear combination between learned filters and original convolution kernels in pre-trained CNNs, the reconstruction error is minimized to determine the most representative filters from the filter bank. These filters are used to build a network followed by HOG-based feature extraction for feature representation. The proposed approach shows competitive performance on color face recognition compared with other deep CNNs-based methods. Besides, it provides a perspective of interpreting CNNs by introducing the concepts of advanced convolutional layers to unsupervised filter learning.

1. Introduction

With the development of deep learning in recent years, deep neural networks, especially deep convolutional neural networks (CNNs) have achieved state-of-the-art performance in many image-based applications [1], e.g., image classification [2,3], face recognition [4,5], fine-grained image categorization [6,7] and depth estimation [8,9]. Compared with traditional visual recognition methods, CNNs have the advantage of learning both low-level and high-level feature representations automatically instead of designing hand-crafted feature descriptors [10,11]. Due to these powerful features, CNNs have revolutionized the computer vision community and become one of the most popular tools in many visual recognition tasks [7,12,13].

Generally, CNNs are made up of three types of layers, i.e. convolutional layers, pooling layers, and fully-connected layers. The features are extracted by stacking many convolutional layers on top of each other, and backpropagation starts from the loss function and goes back to the input in order to learn the weights and biases contained in the layers. However, how this kind of mechanism works on images remains an open question and yet needs to be explored. Besides, learning powerful feature representations requires a large amount of labeled training data otherwise the performance may deteriorate [14,15], whereas training data in practical applications are often not readily available. To solve these problems, some researchers propose learning convolutional layers alternatively independent of training data. In [16], ScatNet was proposed by using wavelet transforms to represent convolutional filters. These predefined wavelet transforms are cascaded with nonlinear and pooling operations to build a multi-layer convolutional network. Therefore, no learning is needed in computing image representation. Different from ScatNet, researchers in [17] introduced a structured receptive field network that combines the flexible learning property of CNNs and the fixed basis filters. The receptive fields can be expressed as a weighted sum of the fixed basis filters, the coefficient of which can be tuned by the users. Similarly, Takumi in [14] applied the orthonormal steerable filters as base filters for re-parameterizing the convolutional filters in CNNs. The reformulated filters are reported to improve the classification performance and reduce the model size on various architectures of CNNs.

Recently, many generalized and improved modules from conventional CNNs were introduced to build new deep learning frameworks. One existing direction is integrating the information in multiple channels to capture more comprehensive features. In [18], Hong et al. applied multitask learning on face-pose estimation to combine different views of face representations. In [19], a multimodal approach on click prediction was proposed for ranking web images. It constructed multiple hypergraph

* Corresponding author.
E-mail addresses: mzhang367-c@my.cityu.edu.hk (M. Zhang), sheheryarkhan@cuhk.edu.hk (S. Khan), h.yan@cityu.edu.hk (H. Yan).

https://doi.org/10.1016/j.patcog.2019.107176
0031-3203/

Laplacians and preformed sparse coding on integrated features. Another promising direction is learning filter banks in advance from training data and the learned filters are fixed during the test to get feature representations of input data. A typical example was Principal Component Analysis Network (PCANet) [20] where the filter banks appearing as convolutional layers are obtained by principal component analysis (PCA). The nonlinear layers and pooling layers of PCANet are simulated by binarization and block-wise histogram respectively. By stacking two stages of filter banks, it can achieve promising results in many classification tasks. Inspired by PCANet, many generalized versions were presented more recently. For example, Sun et al. [21] proposed combing Fisher Linear Discriminant Analysis (LDA) with PCANet to learn features with more discriminative information. Yu and Wu [22] introduced a two-dimensional PCANet, where 2D-PCA is applied to the image patch matrix to learn the basic filter components. Basically, the unsupervised idea of these variations is analogous to Auto-encoders (AEs) [23] which are the most representative unsupervised deep learning methods. Distinguished from CNNs, AEs aim to learn low-dimensional feature embeddings which can perfectly reconstruct the original data. This mechanism is similar to traditional dimensional reduction methods like PCA.

The other direction from improving CNNs in terms of efficiency and accuracy is convolutional layer designing [1]. The general trend of CNNs in past years has made the networks going deeper with a significant increase in the number of learnable parameters and computation operations. Consequently, the training of CNNs requires a large space of parameter storage. For example, a standard VGG-16 [2] network takes the storage of parameters with more than 500Mb which brings a heavy burden to embedded devices where the computation and storage capability are both limited. Another leading problem is, a deep CNN becomes hard to converge and vulnerable to overfitting [12,15,24]. To solve these problems, variations of CNNs' convolutional layers are developed. The typical examples among them are 1×1 convolution introduced in [25], the depth-wise separable convolution conducted in MobileNets [26] and a series of Inception modules introduced in [24,27,28]. These newly invented convolutional layers not only contribute to develop deeper networks with better generalization ability but also reduce the number of learned parameters significantly.

For CNNs, generally, the first layer is a convolutional layer followed by stacked layers on top of each other. It means all the information has been extracted within the output of first layer while the following layers apply various types of transformations, e.g. convolution, nonlinearity, pooling to map the features further. If the convolution kernels of first layer in CNNs can be obtained alternatively instead of learning by backpropagation, then, on one hand, it provides the understanding of CNNs' behavior from another view. On the other hand, the obtained kernels can be employed for learning feature representation directly. Moreover, one can expect competitive performance by building consecutive stages to learn kernels in multi-layers. In this paper, by analyzing the characteristic of convolutional layers in CNNs, we propose a novel filter leaning approach. Note we only consider the kernels contained in the convolutional layer. Since the number of parameters of biases is much fewer than that of kernels, biases are known to have little influence on a deep CNN which can be eliminated during training.

The main contributions of this study are three-fold: (1) We propose a three-stage approach to learn multi-structure convolution filters alternatively called eigen-filters. Using these eigen-filters, a network with three convolutional layers can be built followed by HOG based feature extraction for image representation. We call it Deep Eigen-filters Net (DEFNet). (2) The origin convolution kernels of CNNs can be re-parameterized as the linear combination of learned eigen-filters. A threshold-based filter selection method is proposed to minimize the reconstruction error and select a minimal number of top filters from the initial filter bank. (3) From face recognition experiments, our proposed DEFNet is highly superior on small size datasets or extreme cases with several images per class. The training of our network is computation-efficient while it performs better than other compared deep learning-based methods on several subsets of public databases [13,29].

The rest of the paper is organized as follows. In Section 2, we first review the standard convolutional layer with their two variations in structure and then introduce some works on learning predefined convolution kernels of CNNs. In Section 3, the methodology of our proposed filter learning approach is described in detail. Then, DEFNet is proposed for feature representation based on learned multi-structure eigen-filters. In Section 4, we first show the intermediate results of filter selection, filter visualization, and face representation. Then face recognition tests on several popular datasets including VGGFace [13], FaceScrub [29], AR [30], color FERET [31] and LFW [32] are presented. In Section 5, we justify our proposed approach theoretically from two aspects. Finally, we conclude this work in Section 6 and discuss important directions for future research.

2. Related work

In this section, we start by reviewing the convolutional layer of CNNs and their variations. Then, we discuss some important studies related to the relationship between the convolution kernels of CNNs and prefixed filters.

2.1. Design of CNNs' convolutional layers

2.1.1. Standard convolutional layer

Given a square 3-dimensional input Z with shape $C \times D_Z \times D_Z$, a conventional convolutional layer will produce a $N \times D_V \times D_V$ feature map V, where D_Z is the spatial height and width of Z, C is the number of input channels, D_V is the spatial height and width of output V and N is the number of output channels. Denote the kernel contained within the convolutional layer as a 4-dimensional tensor W with shape $N \times C \times D_K \times D_K$ and the elements of W as $W_{n,\,c,\,i,\,j}$, where D_K is the spatial size of a square kernel which is odd ($D_K = 2k + 1$, $k \in \mathbb{N}$) and $W_{n,\,c,\,i,\,j}$indicates the weights connecting the channel n of output and channel c of input with the offsets of i rows and j columns between the output receptive field and input receptive field [15].

When using stride $s = 1$ and padding, the relationship between the output size D_V and input size D_Z can be formulated as: $D_V = D_Z - D_K + 2p + 1$, where p is the padding size along each of the axis. Choosing $p = D_K/2$ will result in D_Z and D_V with the same size which is a good property sometimes. In the general case, for no-unit stride s ($s > 1$), the relation between the input size and output size is extended as: $D_V = (D_Z + 2p - D_K)/s + 1$.

In the case of unit-stride and zero-padding, the standard convolution operation [15] can be parameterized as:

$$V_{n,p,q} = \sum_{i,\ j,c} W_{n,c,i,j} \cdot Z_{c,p+i-1,q+j-1} \tag{1}$$

From Eq. (1), the number of multiplications required for a standard convolution is

$$D_K \cdot D_K \cdot C \cdot N \cdot D_V \cdot D_V \tag{2}$$

The number of learnable parameters required within the convolution kernel W is $D_K^2 CN$. The computational cost lies on the product of the square kernel size $D_K \times D_K$, number of input channels C, number of output channels N and the output feature map size $D_V \times D_V$. Since the channel number of activations increases significantly when going deeper in CNNs, the required multiplications in Eq. (2) is computationally expensive.

2.1.2. Convolutional layer of 1 × 1 convolution

The use of 1 × 1 convolutions was first proposed by Lin et. al. in Network in Network [25], the idea of which has been widely adopted in many later proposed CNNs architectures, like Inception [28], GoogLeNet [24], and ResNet [12]. The basic idea of 1 × 1 convolutions is straightforward, the convolution operation is convolved on the unit spatial size of the input along the channel-axis. Given a square input feature map Z with shape $C \times D_Z \times D_Z$, where C is the number of input channels, D_Z is the height and width of the input feature map. A 1 × 1 convolution kernel $W^{\dagger}$ with shape $N \times C \times 1 \times 1$ will produce an output feature map V with shape $N \times D_V \times D_V$ where $D_V = D_Z$ is the spatial size of output feature map and N is the number of output channels. The 1 × 1 convolution can be parameterized as follow:

$$V_{n,p,q} = \sum_{c} W^{\dagger}_{n,c,1,1} \cdot Z_{c,p,q} \tag{3}$$

The motivation behind this 1 × 1 convolution is to reduce (when $N < C$) or increase the dimension (when $N > C$) along the channel-axis. Compared with Eq. (2), the number of multiplications required for a convolution operation is greatly reduced from $D_K \cdot D_K \cdot C \cdot N \cdot D_V \cdot D_V$ to $C \cdot N \cdot D_V \cdot D_V$, which is extremely useful when dealing with large channel number of deep activations. Moreover, the number of learnable parameters contained within the convolution kernel is dropped from $D_K^2 CN$ to CN. Another benefit of 1 × 1 convolution is it helps to enhance the nonlinearity of CNNs without any spatial information loss by cross-channel linear combination meanwhile keeping the spatial height and width of the input unchanged after convolution.

2.1.3. Convolutional layer using depth-wise separable convolution

Depth-wise separable convolution was initially proposed in [33] and subsequently applied in Xception [27] and MobileNets [26]. Generally, the conventional convolutional layers first convolve on each channel of the input feature map, then combine the filtered outputs of each channel simultaneously to produce a new representation. Different from that, the characteristic of depth-wise separable convolution is it factorizes the conventional convolution into two independent steps, i.e. depth-wise convolutions and point-wise convolutions, each step of which deals with the corresponding convolution. Specifically, in depth-wise step, each channel of input feature maps is filtered with a single convolution kernel in horizontal and vertical direction individually, so the dimension of input feature maps in channel-axis remains unchanged in this step. The depth-wise convolution can be formulated as:

$$V_{c,p,q} = \sum_{i,\ j} \hat{W}_{c,i,j} \cdot Z_{c,\ p+i-1,q+j-1} \tag{4}$$

where $\hat{W}$ is the convolution kernel in shape $C \times D_K \times D_K$ and the cth channel of $\hat{W}$ is applied to the cth channel of input feature map Z to generate the cth channel of output feature map V.

While in point-wise step, the 1 × 1 convolution implemented in the same manner as in Eq. (3) is used to produce the linear combination of output feature maps of depth-wise convolution. The total computation cost required for a depth-wise separable convolution is the sum of computation costs of depth-wise and point-wise convolution, which is calculated as:

$$D_K \cdot D_K \cdot C \cdot D_V \cdot D_V + C \cdot N \cdot D_V \cdot D_V \tag{5}$$

The ratio of Eqs. (2) and (5) is equal to $1/N + 1/D_k^2$ which indicates a decrease of computation cost from standard convolution to depth-wise separable convolution. And the ratio of number of learnable parameters between two types of convolution operations is $(D_K^2 C + CN)/D_K^2 CN = 1/N + 1/D_k^2$, which is the same times of reduction as that of computation cost.

2.2. The relationship between convolution kernels of CNNs and prefixed filters

By analyzing the composition of convolution kernels learned from CNNs' training procedure, alternative filters can be designed to replace or simplify the parameterization of original kernels. This section briefly summarizes two main streams of existing works: one is learning predefined filters independent of data [14,16,17], the other is computing training-fixed filter banks from task-dependent datasets extending unsupervised methods like PCA [20,22] or supervised methods like LDA [20,21].

2.2.1. Reconstruct the convolution kernels by prefixed filters independent of data

Within this scope, predefined wavelet filters [16] and Gaussian derivative filters [14,17] have been explored to reconstruct the original convolution kernels. Consider a convolution kernel W in C channels, $W = \{w_1,\ w_2, \ldots w_C\}$. Introduce a set of n learned basis filters $B = \{b_1,\ b_2,\ \ldots,\ b_n\}$, then each channel of the convolution kernel w_c can be expressed as the linear combination of B and a set of basis weights $a_c = \{a_{c,\ 1},\ a_{c,\ 2},\ \ldots,\ a_{c,\ n}\}$. Denote the C-channel input as Z, the convolution over Z based on W can be reformulated as:

$$W * Z = \sum_{c=1}^{C} w_c * z_c = \sum_{c=1}^{C} \sum_{i=1}^{n} a_{c,i} b_i * z_c \tag{6}$$

where z_c is the cth channel of Z and $\overset{n,C}{\underset{i,c=1}{}}\{a_{c,i}\}$ is a set of trainable coefficients of the convolution kernel W.

Since basis filters B are precomputed, the number of learnable parameters is equal to the number of coefficients, i.e. Cn. This approach brings dual advantages. Firstly, it greatly reduces the training cost of convolutional layers and improves the robustness of the network to overfitting. Secondly, it still lefts parameters to be tuned by users. However, the discriminative performance of these predefined filters generated by Gaussian derivative, wavelet transformation [14,16,17] may be worse if the applied dataset is far from the domain of the pre-trained dataset.

2.2.2. Learn the filter bank from training sets based on PCA or LDA

The most popular work belonging to this scope was PCANet [20], and some generalized versions such as 2DPCANet [22] and FPCANet [21] were proposed recently. The basic idea of PCANet is learning two filter banks by PCA in two consecutive stages respectively. Given a set of training images $\{I_1,\ I_2, \ldots I_n\}$ with the same size $w \times h$. The first stage of PCANet is briefly summarized as followings: (1) Split each image I_i into patches with size $p_1 \times p_2$, unit stride and no zero-padding referring to standard convolution operation. The valid region of the input image has the shape of $\tilde{w} \times \tilde{h} = (w - p_1 + 1) \times (h - p_2 + 1)$; (2) Flatten each patch of I_i to a patch vector with dimension $p_1 p_2$, concatenate them to obtain the matrix $X_i = [X_{i,1},\ X_{i,2},\ \ldots, X_{i,\tilde{w}\tilde{h}}] \in \mathbb{R}^{p_1 p_2 \times \tilde{w}\tilde{h}}$, where $X_{i,\ m}$ is the mth patch vector of I_i; (3) Subtract the patch-mean of each patch vector of I_i to obtain the patch-mean removed image matrix $\bar{X}_i$, concatenate all the patch-mean removed $\bar{X}_i$ to form $\bar{X} = [\bar{X}_1, \bar{X}_2, \ldots \bar{X}_n\ ] \in \mathbb{R}^{p_1 p_2 \times n\tilde{w}\tilde{h}}$; (4) Finally, PCANet adopts minimizing the reconstruction error in Frobenius norm as following:

$$\min_{U} \left\| \bar{X} - UU^T \bar{X} \right\|_F^2,\ s.t. U^T U = I \in \mathbb{R}^{l \times l} \tag{7}$$

where $U \in \mathbb{R}^{p_1 p_2 \times L_1}$ is the optimum projection matrix and l is the number of filters in the first stage. The solution of Eq. (7) is a matrix containing column vectors of $\overset{l}{\underset{i=1}{}}\{u_i(\bar{X}\bar{X}^T)\}$ where $u_i(\bar{X}\bar{X}^T)$ is the ith eigenvector of the covariance matrix $\bar{X}\bar{X}^T$. Convert each column vector u_i to the matrix w_i with shape $p_1 \times p_2$, then the obtained filter bank with l filters can be formulated as $W = \{w_1,\ w_2, \ldots w_l\} \in \mathbb{R}^{l \times p_1 \times p_2}$.

The second stage of PCANet is analogous to the first stage, whereas the inputs of the second stage are the output feature maps of the first stage based on l filters. Although these two simple stages of PCANet are reported to obtain satisfactory results on many datasets, it does not utilize advanced structures of current CNNs, and the filter banks are selected mainly based on empirical experiences. Hence, the relationship between CNNs' convolution kernels and computed filter banks is not well established.

3. Deep eigen-filters and DEFNet for feature representation

Our proposed approach consists of three stages for learning filters. We first introduce our filter learning procedure stage by stage. Then, the pipeline of building a network based on the obtained filter banks for feature representations is given. The overall diagram of the filter learning approach is illustrated in Fig. 1.

3.1. Problem definition and preparation

Assuming we are given the training set with N images $_{i=1}^{N}\{I_i\}$. Following the general case of CNNs, I_i is three-channel RGB image with the same shape $3 \times h \times w$, where h and w are the spatial height and spatial width of I_i respectively. The task of proposed three-stage approach is to learn filter banks $_{i=1}^{3}\{W^i\}$ and corresponding statistical moment, i.e. filter-mean $_{i=1}^{3}\{\boldsymbol{\mu_i}\}$ in each stage. To facilitate the description of formulation, the patch size, stride and zero-padding size keep the same in all the stages. Specifically, the patch referred to the convolution kernel in CNNs is fixed with square shape $p \times p$, where p is an odd number $p = 2n + 1,\ n \in \mathbb{N}$. The stride s is applied in patch-unfold with zero-padding size $u = p/2 = n$, which ensures the same size of inputs and outputs when $s = 1$.

3.2. The first stage of filter learning

We start with learning filter bank and filter-mean from the data augmented training set, then the analysis between the initial filter bank and convolution kernels in the first layer of CNNs is provided to select the final l_1 filters. Finally, apply the learned l_1 filters to obtain the output feature maps of the first stage.

3.2.1. Learning standard eigen-filters and filter-mean

Data augmentation is applied first on $_{i=1}^{N}\{I_i\}$ to obtain the augmented data $_{i,\ l=1}^{N,L}\{I_i^l\}$. It is realized by flipping the image in horizontal direction with a 50% probability and cropping the image into a specific size each time it is sampled. In practice, the procedure is conducted online in N epochs to make the number of processed images growing to NL. Generally, it helps to increase the number of training images and one can expect learning more representative filter banks from the augmented data. We divide each image I_i^l into patches with zero-padding size u, patch size p and stride s. This will give a number of $h_1 w_1$ patches, where $h_1 = (h + 2u - p)/s + 1$ and $w_1 = (w + 2u - p)/s + 1$ are the expected output size in vertical and horizontal directions. Flatten each patch image into a $3p^2$-dimensional vector and concatenate them to obtain:

$$M_i^l = \left[m_i^{l,1},\ m_i^{l,2}, \ldots, m_i^{l,h_1 w_1}\right] \in \mathbb{R}^{3p^2 \times h_1 w_1} \tag{8}$$

where $m_i^{l,q}$ is the qth patch vector of image I_i^l. The final patch matrix containing all the images $_{i,\ l=1}^{N,L}\{I_i^l\}$ can be denoted as

$$\boldsymbol{M} = \left[M_1^1, M_1^2, \ldots M_1^L, M_2^1, \ldots, M_2^L, \ldots, M_N^L\right] \in \mathbb{R}^{3p^2 \times h_1 w_1 LN} \tag{9}$$

Then, compute the filter-mean $\boldsymbol{\mu}_1$ of M. $\boldsymbol{\mu}_1$ is the mean along the second axis of $\boldsymbol{M}$, which is distinguished from the mean along the first axis, known as patch-mean in [20]. As the second dimension of matrix $\boldsymbol{M}$ is tremendous, alternatively, the filter-mean vector $\boldsymbol{\mu}_1$ can be calculated as

$$\boldsymbol{\mu}_1 = \frac{1}{h_1 w_1 NL} \sum_{i,l=1}^{N,L} M_i^l 1_{h_1 w_1} \tag{10}$$

where $1_{h_1 w_1} \in \mathbb{R}^{h_1 w_1}$ is a vector of ones. Thus, subtracting the filter-mean from image matrix M_i^l to obtain

$$\bar{M}_i^l = M_i^l - \boldsymbol{\mu}_1 \tag{11}$$

the substituted patch matrix $\bar{\boldsymbol{M}}$ including all the filter-mean removed images is

$$\bar{\boldsymbol{M}} = \left[\bar{M}_1^1, \bar{M}_1^2, \ldots \bar{M}_1^L, \bar{M}_2^1, \ldots, \bar{M}_2^L, \ldots, \bar{M}_N^L\right] \in \mathbb{R}^{3p^2 \times h_1 w_1 LN} \tag{12}$$

Finally, compute the covariance matrix C of $\bar{\boldsymbol{M}}$ and solve the eigenvalues decomposition of C. Similarly, we compute the covariance matrix in an alternative way, which is formulated as

$$C = \frac{1}{h_1 w_1 LN} \sum_{i,l=1}^{N,L} \bar{M}_i^{l^T} \bar{M} \in \mathbb{R}^{3p^2 \times\ 3p^2} \tag{13}$$

The eigendecomposition of C is defined as $C = U \Lambda U^{-1}$, where $U \in \mathbb{R}^{3p^2 \times\ 3p^2}$ is a square matrix. Each column of U denoted as $\boldsymbol{u_i}$

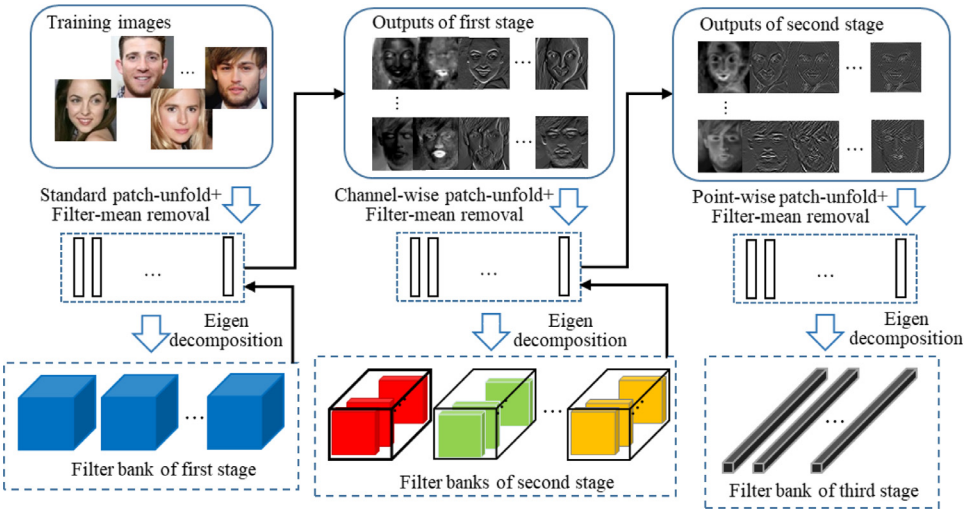


Fig. 1. Overall diagram of proposed multi-structure filter learning approach.

is an eigenvector of C. We sort all the eigenvectors ${}_{i}^{3p^2}\{\boldsymbol{u_i}\}$ and corresponding eigenvalues ${}_{i}^{3p^2}\{\lambda_i\}$ based on the value of λ_i in decreasing order. The initial $3p^2$ eigen-filters ${}_{i}^{3p^2}\{W_i^1\}$ is obtained by

$$W_i^1 = \mathbf{mat}_{3,p,p}\boldsymbol{u_i} \in \mathbb{R}^{3\times p\times p} \tag{14}$$

where operator **mat** means reshaping the vector into a matrix with the corresponding dimension in each axis.

3.2.2. Filter selection and reconstruction towards comparing with CNNs

Consider a pre-trained CNN with the same convolution configurations of kernel size, stride and zero-padding as in Section 3.1. The CNN is trained on the same dataset ${}_{i=1}^{N}\{I_i\}$ with data augmentation. Denote the convolution kernels contained in the first layer of CNN as $V = \{v_1, v_2, \dots v_C\} \in \mathbb{R}^{C\times 3\times p\times p}$, where C is the number of output channels. Assuming for each convolution kernel, it can be described as a linear combination between all of the initial eigen-filters ${}_{i}^{3p^2}\{W_i^1\}$ and coefficients ${}_{c,i=1}^{C,3p^2}\{\alpha_{c,i}\}$, which is formulated as:

$$v_c = \sum_{i=1}^{3p^2}\alpha_{c,i}W_i^1 = \hat{W}\boldsymbol{\alpha_c} \tag{15}$$

where $\hat{W}$ is the matrix each column of which corresponds to the vectorized W_i^1 and $\boldsymbol{\alpha_c} = [\alpha_{c,1}, \alpha_{c,2}, \dots \alpha_{c,3p^2}]^T$ is the coefficient vector of cth kernel of CNN. $\boldsymbol{\alpha_c}$ can be efficiently obtained by solving the linear system equation of Eq. (15). Since $\hat{W}$ is orthonormal, the solution is nontrivial. To select the top l eigen-filters out of total $3p^2$ filters, we define the following criterion to evaluate how well the selected eigen-filters can explain the convolution kernels in the pre-trained CNNs:

$$\epsilon^l(W;V) = 1 - \frac{\sum_c \left\|\mathbf{vec}(\sum_{i=1}^{l}\alpha_{c,i}W_i^1)\right\|_2^2}{\sum_c \left\|\mathbf{vec}(v_c)_2^2\right\|} \tag{16}$$

where the operator **vec** represents vectorizing the matrix (filter) to a row vector. From Eq. (16), $\epsilon^l(W;V)$ is the reconstruction error between the selected l eigen-filters and total filter energy $\sum_c \|\mathrm{vec}(v_c)\|_2^2$ of the convolution kernel V. Thus, the optimum l_* is found by:

$$\underset{l}{\operatorname{argmin}}\, \epsilon^l(W;V) < \varepsilon \tag{17}$$

where ε is a tolerance found from experiment results. It is set to 0.02 in this paper in terms of the tradeoff between the reconstruction error and the minimum number of covered filters.

The top l_* eigen-filters selected from initial filter bank can explain most of the convolution kernels of pre-trained CNN. To make full use of the eigen components, a reconstructed filter W_*^1 is additionally built as a weighted sum of the remaining unselected filters ${}_{i=l_*+1}^{3p^2}\{W_i^1\}$, which is formulated as:

$$W_*^1 = \sum_{i=l_*+1}^{3p^2}\lambda_i^* W_i^1 \tag{18}$$

where $\lambda_i^* = \lambda_i / \sum_{j=l_*+1}^{3p^2}\lambda_j$ are normalized eigenvalues of ${}_{i=l_*+1}^{3p^2}\{\lambda_i\}$. Finally, the eigen-filters of first stage are parameterized as $W^1 = [W_1^1, W_2^1, \dots, W_{l_*}^1, W_*^1]$ with shape $(l_*+1)\times 3\times p\times p$. For simplicity, denote $l_1 = l_* + 1$ as the number of filters in the first stage.

3.2.3. Outputs of the first stage

Note the training images are only augmented for filter learning, and the outputs of first stage are based on the original training set ${}_{i=1}^{N}\{I_i\}$. Provided the filter bank W^1 and filter-mean $\boldsymbol{\mu}_1$ of the first stage, the jth feature map of I_i is formulated as

$$Z_{i,j} = W_j^1 * I_i = \mathbf{mat}_{h_1,w_1}\left[\mathbf{vec}\left(W_j^1\right)\bar{M}_i\right] \in \mathbb{R}^{h_1\times w_1};$$
$$i = 1, 2, \dots N;\ j = 1, 2, \dots l_1 \tag{19}$$

where $\bar{M}_i$ is the filter-mean subtracted patch vector matrix of I_i according to Eqs. (8) and (11). Note the most right-hand side of the Eq. (19) follows the essence of the convolution operation, i.e. unfold-multiplication-fold, which is more practical to use in the implementation. Here, we first fold the image into a patch vectorized matrix and subtract the filter-mean, then it is multiplied by a vectorized filter to produce a row vector. Finally, the result is reshaped to the expected size of the output feature map.

To provide the local translation invariance of the output feature maps as well as reducing the computational cost in training, pooling layers are employed in CNNs [2,3,15]. Inspired by these, in the case of stride $s = 1$, the outputs of the first stage are followed by an average pooling layer with pooling size 2×2 and pooling stride 2. This operation produces a halved output spatial size in both vertical and horizontal directions.

3.3. The second stage of filter learning

Given the output feature maps $Z_i^1 = [Z_{i,1}^1, Z_{i,2}^1, \dots, Z_{i,l_1}^1] \in \mathbb{R}^{l_1\times h_1\times w_1}$ of each training image I_i from the first stage, concatenate the outputs of all the images to obtain $Z^1 = [Z_1^1, Z_2^1, \dots, Z_N^1] \in \mathbb{R}^{N\times l_1\times h_1\times w_1}$. Split Z^1 into l_1 groups ${}_{j=1}^{l_1}\{G_j\}$:

$$G_j = \left\{Z_{1,j}^1, Z_{2,j}^1, \dots, Z_{N,j}^1\right\} \tag{20}$$

where G_j collects the feature maps produced by the same eigen-filter W_j^1 of each training image. The task of the second stage is learning the filter banks ${}_{j=1}^{l_1}\{W_j^2\}$ and filter-mean $\boldsymbol{\mu}_{2,\boldsymbol{j}}$ of each group G_j .

3.3.1. Learning channel-wise filter banks and filter-mean

We first split each two-dimensional feature map $Z_{i,j}^1 \in \mathbb{R}^{h_1\times w_1}$ of G_j into patches following the configurations of patch size p, zero-padding u and stride s. Different from the first stage where the patch image is three-dimensional of $3\times p\times p$, now it becomes two-dimensional of $p\times p$. Denote the shape of expected output feature maps as $h_2\times w_2$. Flatten each patch images into a vector of dimension p^2 and concatenate each vector into a matrix:

$$M_{i,j} = \left[m_{i,j}^1, m_{i,j}^2, \dots, m_{i,j}^{h_2w_2}\right] \in \mathbb{R}^{p^2\times h_2w_2} \tag{21}$$

where $m_{i,j}^q$ is the qth patch vector in the jth feature map produced in the first stage of image I_i.

Then, calculate the filter-mean of M_j with the same approach as in Eq. (10):

$$\boldsymbol{\mu}_{2,\boldsymbol{j}} = \frac{1}{h_2w_2N}\sum_{i=1}^{N}M_{i,j}1_{h_2w_2} \tag{22}$$

where $1_{h_2w_2} \in \mathbb{R}^{h_2w_2}$ is a vector of ones. Analogous to Eq. (11), subtract the filter-mean from each $M_{i,j}$ and concatenate them to obtain the filter-mean removed patch matrix of group G_j:

$$\bar{M}_j = \left[\bar{M}_{1,j}, \bar{M}_{2,j}, \dots \bar{M}_{N,j}\right] \in \mathbb{R}^{p^2\times h_2w_2N} \tag{23}$$

Finally, the covariance matrix C_j of $\bar{M}_j$ is formulated as

$$C_j = \frac{1}{h_2w_2N}\sum_{i=1}^{N}\bar{M}_{i,j}\bar{M}_{i,j}^T \in \mathbb{R}^{p^2\times p^2} \tag{24}$$

Solve the eigenvalues decomposition of C_j and sort all the eigenvectors ${}_{q=1}^{p^2}\{\boldsymbol{u_{j,q}}\}$ regarding the corresponding eigenvalues in

the decreasing order. Denote the number of selected filters of each group as l_2. Then, for each group G_j, the l_2 eigen-filters of the second stage are

$$W_j^2 = {}^{l_2}_{q=1}\left\{W_{j,q}^2\right\} \in \mathbb{R}^{l_2 \times p \times p} \tag{25}$$

where $W_{j,q}^2 = \mathbf{mat}_{p,p}\boldsymbol{u}_{\boldsymbol{j},\boldsymbol{q}} \in \mathbb{R}^{p \times p}$. Note there are l_1 groups of output feature maps in the first stage, repeat the above procedure for all the groups of G, which gives the final filter banks and filter-mean:

$$W^2 = \left\{W_1^2, W_2^2, \ldots, W_{l_1}^2\right\} \in \mathbb{R}^{l_1 \times l_2 \times p \times p};$$
$$\boldsymbol{\mu}_2 = \left\{\boldsymbol{\mu}_{2,1}, \boldsymbol{\mu}_{2,2}, \ldots, \boldsymbol{\mu}_{2,\boldsymbol{l}_1}\right\} \in \mathbb{R}^{p^2 \times l_1} \tag{26}$$

3.3.2. Outputs of second stage based on multi-outputs channel-wise convolution

Given the feature maps Z^1 from the first stage, learned filter banks W_j^2 and filter-mean $\boldsymbol{\mu}_{2,\boldsymbol{j}}$ of each channel-wise group, the output feature map of I_i at the second stage is formulated as:

$$Z_i^2 = {}^{l_1}_{j=1}\left\{W_j^2 * Z_{i,j}^1\right\} \tag{27}$$

The part inside the braces of Eq. (27) can be extended as:

$$W_j^2 * Z_{i,j}^1 = {}^{l_2}_{q=1}\left\{W_{j,q}^2 * Z_{i,j}^1\right\} = {}^{l_2}_{q=1}\left\{\mathbf{mat}_{h_2,w_2}\left[\mathbf{vec}\left(W_{j,q}^2\right)\bar{M}_{i,j}\right]\right\} \tag{28}$$

where $\bar{M}_{i,j} = M_{i,j} - \boldsymbol{\mu}_{2,\boldsymbol{j}}$. From Eqs. (27) and (28), each filter bank W_j^2 is applied to the jth feature map $Z_{i,j}^1$ of image I_i to produce l_2 feature maps in second stage, aggregate all the feature maps produced by l_1 groups of filter banks to obtain the total number of $l_1 l_2$ feature maps for I_i.

Different from the depth-wise convolution used in [26] where each kernel convolved with a single channel of input only generates one feature map. Here, each filter contained in the filter bank will convolve with a single-channel feature map $Z_{i,j}^1$, which produces l_2 feature maps per filter bank. There are l_1 filter banks leading to a $l_1 l_2$-channel output per image. In this paper, we simply choose $l_2 = l_1$. Hence, the outputs of second stage can be represented as a four-dimension tensor Z^2 with shape $N \times l_1^2 \times h_2 \times w_2$.

3.4. The final stage of filter learning

Given the output feature maps $Z^2 = {}^{N}_{i=1}\{Z_i^2\}$ from the second stage with depth l_1^2, the task of the final stage is learning a series of filters ${}^{l_3}_{j=1}\{W_j^3\}$ with shape $l_1^2 \times 1$ and the filter-mean $\boldsymbol{\mu}_3$, where l_3 is the number of filters selected for the final stage. This method is inspired by the 1×1 convolution [25] which is widely employed for reducing the channel dimension of deep activations.

3.4.1. Learning point-wise filters and filter-mean

For each output $Z_i^2 \in \mathbb{R}^{l_1^2 \times h_2 \times w_2}$, unfold Z_i^2 around each pixel along the channel dimension to obtain:

$$M_i = \left\{m_{1,1}, m_{1,2}, \ldots m_{1,w_2}, m_{2,1}, m_{2,2}, \ldots m_{2,w_2}, \ldots, m_{h_2,1}, m_{h_2,2}, \ldots m_{h_2,w_2}\right\} \in \mathbb{R}^{l_1^2 \times h_2 w_2} \tag{29}$$

where $m_{j,k} \in \mathbb{R}^{l_1^2}$ is a vector containing all the pixels located at the jth row and kth column of each channel in Z_i^2. Repeat the same procedure for all the N training images to build $M = [M_1, M_2, \ldots, M_N]$ and calculate the mean of M along the second dimension as following

$$\boldsymbol{\mu}_3 = \frac{1}{h_2 w_2 N}\sum_{i=1}^{N} M_i 1_{h_2 w_2} \in \mathbb{R}^{l_1^2} \tag{30}$$

where $1_{h_2 w_2} \in \mathbb{R}^{h_2 w_2}$ is a vector of ones. Note when applying point-wise patch-unfold, each entry of $\boldsymbol{\mu}_3$ is equal to the mean of the corresponding channel of all the outputs, described as

$$\mu_{3,k} = mean_{i=1}^{N}\left\{Z_{i,k}^2\right\} \tag{31}$$

where $Z_{i,k}^2$ is the kth channel of the second stage output of I_i. Subtract $\boldsymbol{\mu}_3$ from each M_i and concatenate them to obtain $\bar{M} = [\bar{M}_1, \bar{M}_2, \ldots \bar{M}_N] \in \mathbb{R}^{l_1^2 \times h_2 w_2 N}$.

Follow the same approach as in Eq. (24), the covariance matrix $C \in \mathbb{R}^{l_1^2 \times l_1^2}$ is calculated from $\bar{M}$. Solve the eigenvalue decomposition of C to find the sorted eigenvectors ${}^{l_1^2}_{k=1}\{\boldsymbol{u}_{\boldsymbol{k}}\}$ regarding the corresponding eigenvalues in descending order. The obtained eigen-filters of the final stage are

$$W^3 = {}^{l_3}_{k=1}\left\{\mathbf{mat}_{l_1^2,1}\boldsymbol{u}_{\boldsymbol{k}}\right\} \in \mathbb{R}^{l_3 \times l_1^2 \times 1} \tag{32}$$

In our work, l_3 is chosen to be twice of l_1 following the convention of doubling the number of channels in sequential modules of CNNs. In summary, $l_3 = 2l_1 = 2l_2$.

3.4.2. Outputs of final stage based on 1×1 convolution

Provided learned filters $W^3 = {}^{l_3}_{k=1}\{W_k^3\}$, the filter-mean $\boldsymbol{\mu}_3$ and the output Z_i^2 of image I_i from the second stage, the output of the final stage can be parameterized as

$$Z_i^3 = {}^{l_3}_{k=1}\left\{W_k^3 * Z_i^2\right\} = {}^{l_3}_{k=1}\left\{\mathbf{mat}_{h_2,w_2}\left[\mathbf{vec}\left(W_k^3\right)\bar{M}_i\right]\right\} \tag{33}$$

The middle part of Eq. (33) represents the 1×1 convolution, which can be reformulated as the combined operation of unfold-multiplication-fold shown in the most right-hand side of Eq. (33). Thus, each feature map Z_i^2 is convolved with l_3 filters in 1×1 convolution to generate l_3 feature maps. When $l_3 = 2l_1$, the output of the final stage can be denoted as a tensor Z^3 with shape $N \times 2l_1 \times h_2 \times w_2$.

In the case of unit stride, Z^3 is followed by an average pooling layer with pooling size 2×2 and pooling stride 2 which results in halving the height and the width of each feature map. In conclusion, the spatial size of feature maps is reduced twice during the whole three-stage procedure. It is realized either by non-unit stride during the first stage and second stage or average pooling on the outputs of the first stage and final stage.

3.5. Feature representation based on DEFNet

Given the learned filter banks ${}^{3}_{i=1}\{W^i\}$ and corresponding filter-mean ${}^{3}_{i=1}\{\boldsymbol{\mu}_{\boldsymbol{i}}\}$ of each stage, each training sample I_i is processed through the three stages to obtain the final output denoted as the tensor Z_i^3 with shape $l_3 \times h_3 \times w_3$, where h_3 and w_3 are the spatial height and width of Z_i^3 in final stage. We integrate this three-stage procedure into a network called DEFNet which connects all the output steps without filter learning according to Eqs. (19), (28) and (33).

Based on l_3 feature maps of each image, we apply the histogram of oriented gradients (HOG) introduced in [11] to extract features from each feature map. HOG is a popular hand-crafted image descriptor widely employed in human detection and face recognition [11,34]. The advantage of applying HOG in our approach is dual. First, it introduces translation invariance in some degree. Second, it can be computed efficiently due to the twice down-sampling on the spatial size of outputs. Different from conventional HOG where all the cell histograms within a block are normalized and concatenated to form an extended HOG vector, we adopt a modified HOG-based feature extraction. Specifically, all the histograms within each block are aggregated, which leads to one histogram per block with the same dimension as the original histogram. We find this modification is important for improving the performance on face recognition. The detail implementation of our HOG feature extraction is given as followings. Assume the square cell size is $c \times c$, we apply the block size 2×2 which means each block contains 4 cells. The number of bins of each cell histogram is set to 18 with signed gradients, which means the gradient is

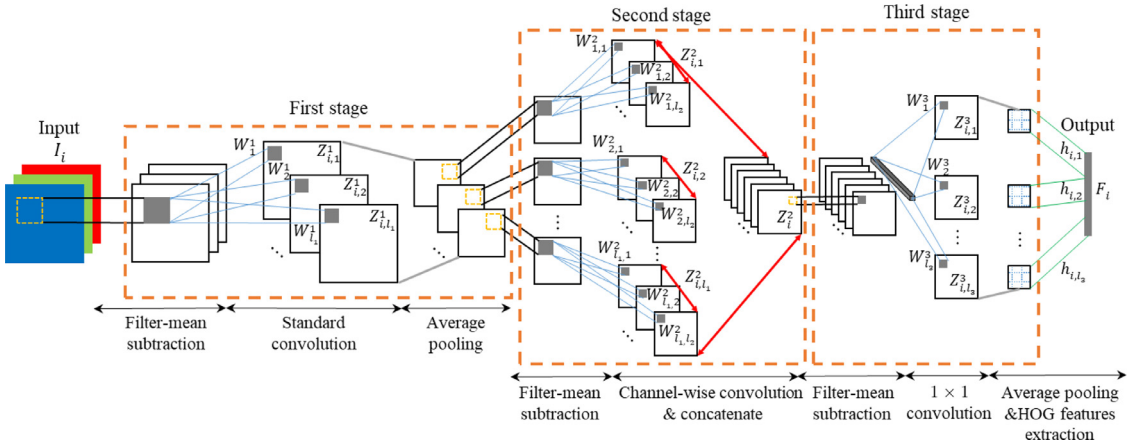


Fig. 2. The architecture of DEFNet based on learned deep-eigen filters.

evaluated on 18 orientations ranging from 0 degrees to 360 degrees. Suppose the number of overlapped blocks generated by HOG in one feature map is B, concatenate all the histograms of a feature map and flatten them into a row vector denoted as $\boldsymbol{h}_{i,k} \in \mathbb{R}^{18B}$, where $\boldsymbol{h}_{i,k}$ represents HOG-based feature vector extracted from the kth feature map of image I_i. Concatenate all the feature vectors $\boldsymbol{h}_{i,k}$ of l_3 feature maps, the final feature representation of I_i is formulated as

$$F_i = \left[\boldsymbol{h}_{i,1}, \boldsymbol{h}_{i,2}, \ldots, \boldsymbol{h}_{i,l_3}\right] \in \mathbb{R}^{18Bl_3} \tag{34}$$

The architecture of the proposed DEFNet is illustrated in Fig. 2 and the overall training procedure of obtaining feature representations is demonstrated in Algorithm 1.

The extracted feature representations ${}_{i}^{N}\{F_i\}$ can be directly fed as inputs to train a linear SVM or k-Nearest Neighbor (k-NN) classifier for the purpose of classification. Since the dimension of F_i maybe too high, it's recommended to use F_i followed by PCA and LDA for dimension reduction as reported in many HOG features-based works [34,44].

4. Experiments and results

In this section, we examine the performance of our proposed deep eigen-filters approach. We first conduct experiments on the procedures of filter learning and feature representation. Then, we test the face recognition performance of our DEFNet on VGGFace [13], FaceScrub [29], AR [30], color FERET [31] and LFW [32] respectively. The experiments on computational cost are provided finally.

4.1. Experiments on filter learning and feature representation

The generic training set for learning filters of our approach is a subset of VGGFace [13] dataset. The images contained in the dataset are all taken under unconstraint conditions. To ensure the purity of the dataset, we manually remove some low-quality and mislabeled images of each identity from the original dataset and eliminate the identities with too fewer images remained (typically less than 100 images). This contributes to a final dataset with around 36000 images of 240 identities, and each identity has around 150 images with one face per image. For each image, face

Algorithm 1
The overall training procedure for feature representation

Input:
Training dataset ${}_{i=1}^{N}\{I_i\}$; Learned filter banks ${}_{i=1}^{3}\{W^i\}$ and filter-mean ${}_{i=1}^{3}\{\boldsymbol{\mu_i}\}$;
Outputs:
Feature representations of training set
for $i = 1 : N$ do
  Patch-unfold of I_i and subtraction of filter-mean μ_1
  Convolution with W^1 to obtain ${}_{j=1}^{l_1}\{Z_{i,j}^1\}$using Eq. (19)
end for (obtain feature maps Z^1 of training set at first stage)
if stride $s == 1$:
  Average pooling on Z^1
for $i = 1 : N$ **do**
  for $j = 1 : l_1$ **do**
  Patch-unfold of $Z_{i,j}^1$ and subtraction of filter-mean $\boldsymbol{\mu}_{2,j}$
  Depth-wise convolution with W_j^2 to obtain ${}_{q=l_2(j-1)+1}^{l_2 j}\{Z_{i,q}^2\}$ using Eq. (28)
end for (obtain each feature map Z_i^2 of I_i)
end for (obtain feature maps Z^2 of training set at second stage)
for $i = 1 : N$ do
  Pointwise patch-unfold of Z_i^2 and subtraction of filter-mean $\boldsymbol{\mu}_3$
  1×1 convolution with W^3 to obtain ${}_{j=1}^{l_3}\{Z_{i,j}^3\}$ using Eq. (33)
end for (obtain feature maps Z^3 of training set at final stage)
if stride $s == 1$:
  Average pooling on Z^3
for $i = 1 : N$ do
  Feature extraction using HOG descriptor on all the feature maps of Z_i^3
end for
  return feature representation of training set ${}_{i}^{N}\{F_i\}$

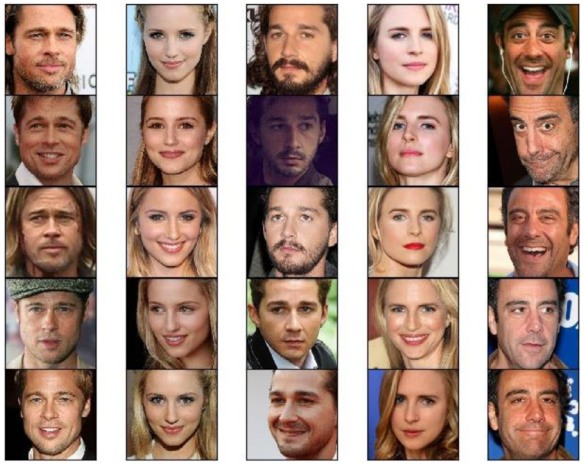


Fig. 3. Example faces from the used training dataset. Each column of the images belongs to the same identity.

Table 1
Summary of three filter learning configurations.

Convolution configuration	l_1	l_2	l_3
Patch size 3 × 3, stride = 1	8	8	16
Patch size 5 × 5, stride = 1	24	24	48
Patch size 7 × 7, stride = 2	34	34	68

detection is applied to obtain a loosely-cropped face image, then it is further processed with 2D alignment. Both the face detection and face alignment are based on the implementation of Dlib [35]. Finally, each face image is resized to the size of 224 × 224. The processed dataset is split into 80% for training, 10% for validation and 10% for testing. The example faces from the employed subset of VGGFace are shown on Fig. 3. We can see the dataset containing face images in variations of pose angle, facial expression, lighting and disguise, which are the common factors considering the real-world applications.

To observe the effects of different patch sizes in our approach, we compare three configurations, i.e. patch size 3 × 3 with unit stride, patch size 5 × 5 with unit stride and patch size 7 × 7 with stride $s = 2$. Denote the number of filters in the first stage as l_1, number of filters in each channel-wise group in second stage as l_2 and number of filters in the third stage as l_3. The summary of three filter-learning configurations is shown in Table 1.

The comparative trained CNN employs the same network architecture and optimization method as in [13], known as VGGFace CNN, except that the last two fully-connected layers are replaced with a global average pooling layer in order to reduce overfitting. The network extensively employs 3 × 3 convolution kernels which is the same with our 3 × 3 filter learning approach. For the 5 × 5 approach, since the functionality of two consecutive 3 × 3 convolution layers is equal to a 5 × 5 layer [28], we train the network replacing the first two 3 × 3 layers with a 5 × 5 layer. For the patch size 7 × 7 with stride 2, we use the network architecture introduced in [12], known as ResNet-50. Specifically, it employs convolution kernel 7 × 7 with stride 2 in the first layer. Note all the employed CNNs have 64 convolution kernels contained in the first layer and they are all trained from scratch on the aforementioned dataset using an NVIDIA RTX 2080Ti GPU.

Fig. 4 illustrates the reconstruction error from Eq. (16) against different numbers of initial filters under three filter learning configurations. Reconstruction error evaluates the fitting performance between selected filters and pretrained CNN. From left to right in Fig. 4, the first indexes along x-axis under the threshold 0.02 are 7, 23 and 33 respectively for three configurations. Hence, we select

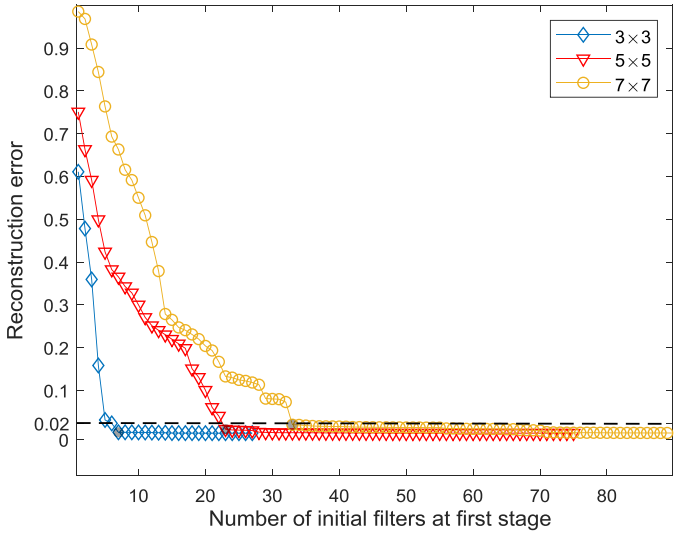


Fig. 4. Reconstruction error between eigen-filters and convolution kernels of the trained CNNs. Indexes of markers shown in filled black are used to establish Table 1.

7 out of 27 filters, 23 out of 75 filters and 33 out of 147 filters for the case of 3 × 3, 5 × 5 and 7 × 7 respectively. Adding the reconstructed filter according to Eq. (18), l_1 is equal to 8, 24 and 34 for three cases respectively. This provides a quantitative analysis of the results established shown in Table 1.

The qualitative analysis of the 3 × 3 approach is achieved by visualizing the coefficient matrix from Eq. (17). Given initial 27 filters of the first stage and 64 convolution kernels contained in the first layer of CNN, we can calculate the coefficient matrix $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \ldots \alpha_{64}]$. The coefficient matrix indicates the importance of each eigen-filter when reconstructing the convolution kernels of CNNs. The visualization of α is illustrated in Fig. 5. Here, each column of α represents a vector of coefficients for a corresponding convolution kernel and each row of α represents all the coefficients from a specific eigen-filter constituting all the convolution kernels. Each entry of α shown in color represents a coefficient ranging from $[-1, 1]$. The darker the color, the larger the absolute value of the entry. From Fig. 5, the first seven eigen-filters take a dominant role in constructing all the convolution kernels of CNN, which corresponds with the leftmost black marker shown in Fig. 4.

The visualization of parts of the convolution kernels and the filter bank of the first stage is shown in Fig. 6(a). Here, the first row illustrates the first eight 3 × 3 convolution kernels contained in the first layer of the trained CNN, and the second row illustrates eight eigen-filters learned in the first stage. For visualization convenience, weights of each kernel/filter are rescaled to the range $[-0.5, 0.5]$ shown in pseudo-color. The eight-channel outputs produced at the first stage, are first given to the average pooling layer to halve the spatial height and width. Then, each channel-wise group of filters in the second stage will convolve with each channel of the feature map, which generates a 64-channel output feature map for each image at the second stage. The channel dimension of outputs is reduced to 16 by propagating to the point-wise filter at final stage. One example image and its 16 output feature maps generated at the final stage are shown in Fig. 6(b) and (c) respectively.

4.2. Experiments on face recognition

4.2.1. Face recognition test on VGGFace database

We learn our proposed DEFNet on the training partition of employed dataset aforementioned in Section 4.1, then evaluate and

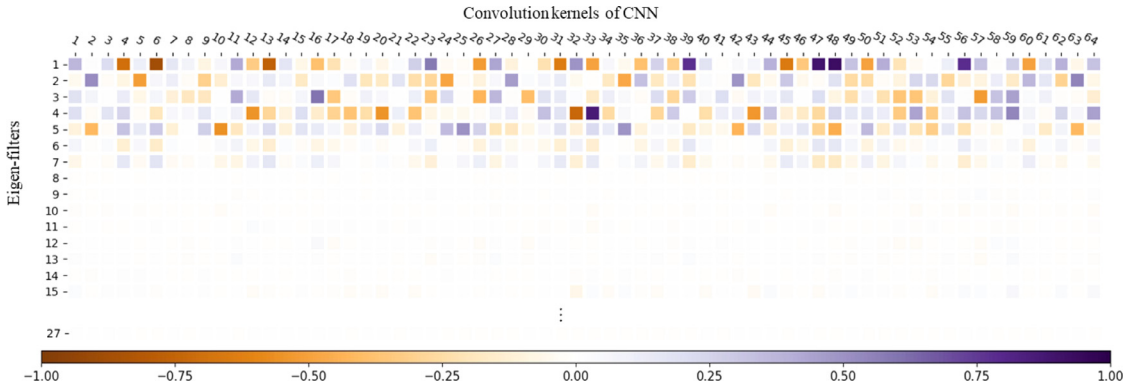


Fig. 5. Visualization of coefficient matrix constituting CNN's convolution kernels

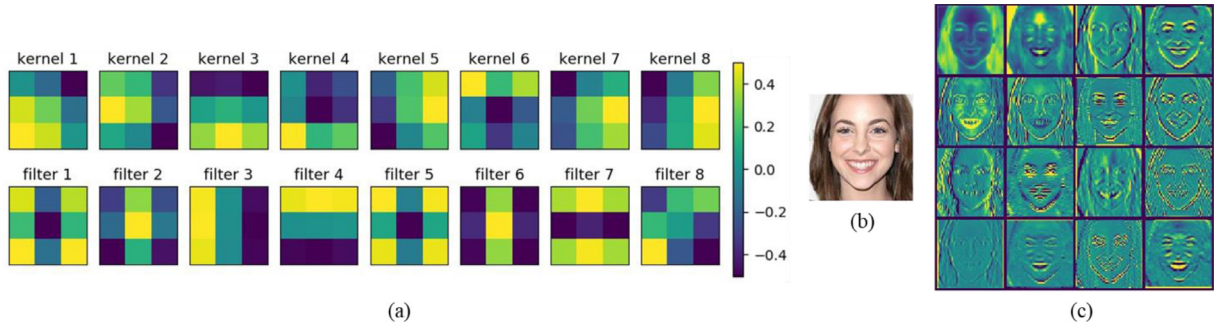


Fig. 6. (a) Visualization of first eight convolution kernels of CNN (first row) and eight eigen-filters (second row). (b) Original image. (c) Output feature maps produced by the final stage. The filter outputs are scaled and mapped to pseudocolors.

test the performance on the other two partitions respectively. For all the comparative methods in the experiment, they are trained using the exact same split dataset. We compare our approach with other six methods, i.e. LBP [10], PCANet [20], VGG-CNN [13], CenterFace [5], SphereFace [36] and CosFace [4]. Among them, LBP is a traditional approach based on the hand-crafted feature descriptor. PCANet is served as a baseline of predefined-filter learning. SphereFace and CosFace are two state-of-the-art deep CNNs with carefully designed loss functions. The implementation details are given as followings: (1) for CNN-based methods, the network of VGG-CNN is the same as described in Section 4.1, and it is trained with classical softmax loss. In CenterFace, we apply the same VGG-CNN architecture as a backbone but changing the softmax loss to the introduced center loss in [5]. For SphereFace and CosFace, we follow the network backbone introduced in [38], which is a modified ResNet [12] with 20 layers. (2) for our approach, we adopt the 5×5 patch configuration. Considering the final output of size 56×56, we use the HOG cell size 8×8 and the block size 2×2. It finally generates 36 blocks per feature map. The concatenated HOG features are further processed by PCA and LDA for dimension reduction. Then, the dimension-reduced feature vector is fed into a k-NN classifier with Euclidean distance measurement for recognition. The comparison of the recognition rates between seven approaches is given in Table 2.

From Table 2, our DEFNet achieves superior average recall to other methods on the dataset where the filters are learned from. Note the minimum recall score is drastically improved to 86.17%. From the validation result, our approach is on par with several state-of-the-art methods like CosFace and SphereFace which utilize well-designed loss function for deep metric learning. One of the reasons why our DEFNet outperforms these compared deep metric learning-based methods may be the relatively small size of our training set. Besides, the proposed filter selection method effectively minimizes the reconstruction error between pre-trained CNN's convolution kernels and our eigen-filters, which promises the performance of DEFNet not inferior to a standard CNN like VGG-CNN. Note our approach is not trained in an end-to-end manner, the validation accuracy could still be enhanced by finetuning some hyper-parameters like the number of components of PCA.

Table 2
Recognition results (%) on the subset of VGGFace dataset.

Methods	Validation	Testing		
		Recall-avg	Recall-min	Recall-max
LBP [10]	92.52	91.83	74.28	95.61
PCANet [20]	94.85	94.43	76.92	100.00
VGG-CNN [13]	96.61	96.47	81.82	100.00
CenterFace [5]	97.79	98.05	84.62	100.00
SphereFace [36]	**98.30**	98.17	81.82	100.00
CosFace [4]	97.67	97.93	84.62	100.00
Proposed DEFNet	98.10	**98.39**	**86.17**	**100.00**

4.2.2. Face recognition test on FaceScrub dataset

To observe the generalization ability of our deep eigen-filters, we directly apply the filters learned from the previous VGGFace dataset on the FaceScrub [29] dataset without filter learning. FaceScrub is similar to VGGFace, which collects color images of celebrities under unconstraint conditions. Here, we use a subset of FaceScrub. We first select 240 subjects including 120 females and 120 males from the original FaceScrub. Then, the same preprocessing as in Section 4.1 is manipulated on the original images to build a dataset with around 150 images per subject. For the fair comparison, we ensure that no subjects used in FaceScrub appear in the employed VGGFace dataset. Follow the same face detection and alignment procedure, each loosely-cropped face image is resized to the size 224×224. The dataset is further split into two partitions, i.e. 80% for training and 20% for test.

Table 3
Recognition results (%) on the subset of FaceScrub database.

Method	Recall-avg	Precision-avg	F1-avg	Recall-min
PCANet [20]	94.28	94.73	94.60	78.57
VGG-CNN [13] + SVM	96.55	96.69	96.57	80.25
CenterFace [5] + SVM	97.67	97.84	97.84	78.57
CosFace [4] finetuning	97.54	97.77	97.57	83.33
SphereFace [36] finetuning	97.88	**98.25**	97.90	84.62
DEFNet + *k*-NN	**98.50**	98.17	**98.17**	**85.71**
DEFNet + SVM	96.84	97.26	97.02	80.25

We compare the generalization performance of our approach with other deep learning-based trained models from the last experiment. The implementation details are given as followings: (1) for VGG-CNN and CenterFace, we directly extract the features from the layer before the last fully-connected layer, which gives a 516-dimensional feature vector for each image. The extracted feature vector is sequentially processed with LDA for dimension reduction and a linear SVM for classification. While for CosFace and SphereFace, to make the most of the advantages of the model, we freeze all the learned parameters before the last block of the network and apply fine-tuning from the last block to the end. (2) for the PCANet, the two-stage filters are prefixed from the previous experiment. (3) for our approach, we follow the same configuration as in the last experiment. Besides, a linear SVM classifier is tested additionally; (4) for all SVM or *k*-NN based methods, 10-fold cross-validation is applied to find the best group of hyperparameters of the classifier.

Table 3 provides the recognition results of average recall, average precision, average f1-score, and minimum recall. It is clear our DEFNet based on prefixed filters generalizes best on the new dataset in terms of average recall, average f1-score, and minimum recall. Compared to PCANet, the better performance may owe to the proposed multi-structure filters in filters learning, which contributes to learning more robust and discriminative representations than simply stacking multi-layer standard filters. One observation from Table 3 is, all pre-trained CNN models deteriorate on the recall scores of new dataset. While for DEFNet, it is slightly improved to 98.50. We can infer since the layers in CNNs are much deeper than that of DEFNet, the features directly extracted from the last layer or fine-tuned from the last several layers may still be too specific to be applied in a new dataset. Another observation from Table 3 is *k*-NN based approach is superior to a linear SVM based approach. Hence, we always select *k*-NN classifier as the first choice in the rest experiments unless special statements.

4.2.3. Face recognition test on AR dataset

To further investigate the face recognition performance of our approach in terms of individual factors, like facial expression, illumination and occlusion, we carry out the experiments on the AR dataset [30]. The dataset contains color images in size 768 × 576 of 70 men and 56 women, with 26 images per subject. Each image of the subject is taken with one of the features including neural expression, non-neural expression, frontal lighting, side illumination, wear scarf/glasses only and wear scarf/glasses with side illumination. Some sample images of AR are shown in Fig. 7. Following the same experimental protocol in [20], we select a subset of the AR dataset consisting of 50 males and 50 females. Each face image is cropped by Dlib [35] face detector and then resized to the size of 224 × 224. For each subject, we choose the images featured with neural expression and frontal illumination as training gallery, while the other images are used for testing. This leads to a dataset with 400 images for training and 2200 images for testing. Test images are further categorized into four subtypes according to their features, i.e. illumination, expressions, occlusion and occlusion with illumination. We compare our approach with RSC [37] and four other pre-trained model-based methods i.e. PCANet, VGG-CNN, CenterFace, and SphereFace. All the compared pre-trained models are trained on the previous VGGFace dataset and they are used to extract features followed by PCA and LDA before fed into an NN classifier with cosine distance measurement. Note we choose to not fine-tune the models for all CNN-based methods because the training set here is too small to train promising CNNs. In proposed DEFNet, specifically, we employ the configuration of patch size 3 × 3 with two types of cell sizes for HOG feature extraction respectively, i.e. 8 × 8 and 14 × 14. We learn the LDA and PCA projections on the feature representations of the training set. The final NN classifier uses a Euclidean distance measurement.

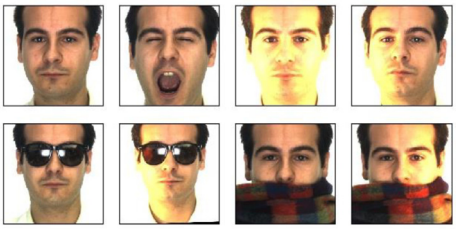


Fig. 7. Example images of AR for a subject.

Table 4
Recognition results (%) of face recognition on AR dataset

Method	Illum.	Exps.	Occlu.	Occlu.&Illum.
RSC [37]	94.00	94.82	95.56	**95.35**
PCANet [20]	98.75	85.41	96.32	94.47
VGG-CNN [13]	99.25	78.83	88.20	87.88
CenterFace [5]	99.50	82.50	91.50	84.13
SphereFace [36]	**100.00**	85.33	91.00	77.13
DEFNet_8 × 8	**100.00**	**96.53**	94.50	94.68
DEFNet_14 × 14	**100.00**	95.20	**96.50**	94.43

The recognition results on AR dataset are summarized in Table 4. For the illumination factor, our approach achieves full scores which is on par with SphereFace. Besides, the recognition rate under variations of facial expressions also outperforms other methods with 96.53%. To the best of our knowledge, we obtain state-of-the-art result on the overall performance on illumination and facial expression. The impressive performance on expression may benefit from the multiple edge-like output feature maps, which greatly improve the robustness to variations of expression. One can infer the prefect score obtained on the illumination factor owes to our filter-mean subtraction in each stage. Under the occlusion case, we also achieve the best recognition rate with 96.50%. While under the case of occlusion with illumination, the recognition rate drops under 95%, one possible reason can be our training set where the filters learned from contains few blocked samples with severe illumination. It's clear all the deep CNN-based models do not generalize well on AR dataset which is much different from the VGGFace in terms of background and image quality. However, the proposed DEFNet is highly adaptive to these changes.

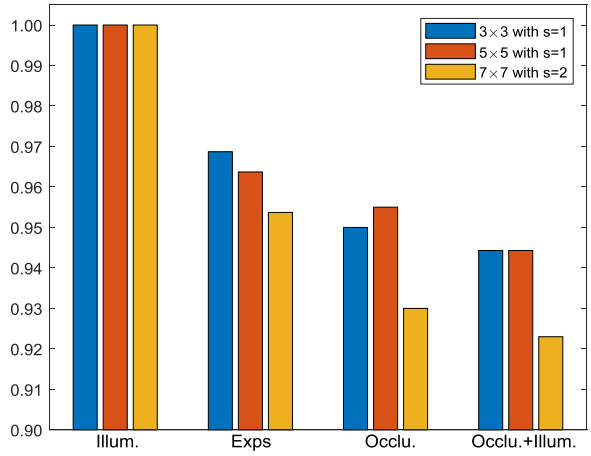


Fig. 8. Comparison on recognition rates under three different patch configurations.

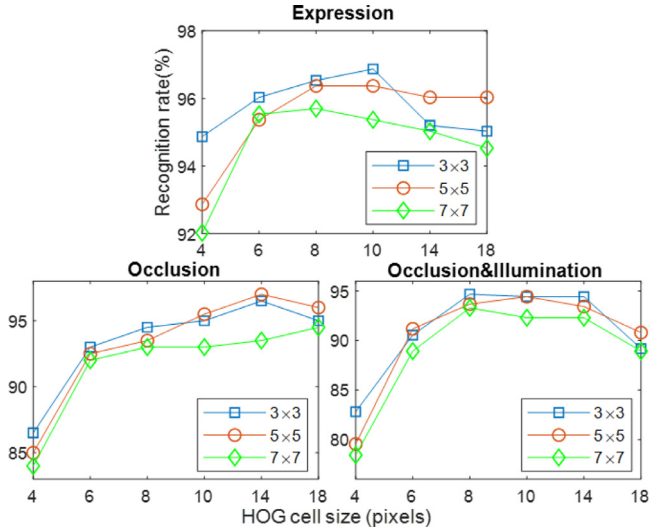


Fig. 9. Results of using different sizes of HOG cells under three convolution configurations.

To observe how different sizes of patches influence the recognition rate, we test three patch configurations listed in Table 1 individually. Here, we apply a fixed 10 × 10 cell size for HOG descriptor. The grouped bar chart summarizing the recognition rates under three configurations is illustrated in Fig. 8. One can see each configuration in our approach achieves 100% recognition accuracy under variations of illumination. For the expression variations, the 3 × 3 approach performs best and the smaller the convolution size, the better the performance. Under occlusion, the 5 × 5 approach performs most robust while the 7 × 7 approach is most sensitive to blocking. In the case of occlusion with illumination, the 3 × 3 and 5 × 5 approach achieve the same accuracy while the 7 × 7 approach is most vulnerable to the adverse impacts.

To further observe how the square cell size of HOG influences the recognition rate, we test six types of cell sizes from 4 to 18 under three patch configurations and plot the curves regarding expression, occlusion only and occlusion with illumination individually in Fig. 9. One can see the influence of HOG cell size is more significant in occlusion than facial expression. In the case of expression, with the increase of the cell size, all three lines first rise then descend. The smaller the convolution size, the larger the HOG cell size when the best performance is achieved. In the case of occlusion only, the results follow the common sense that the bigger the cell size, the more robust it performs. This is because the occluded area is eliminated with features in a larger local region. In the case of occlusion with illumination, all three lines first arise then drop with the increase of the cell size and the best size ranges between 8 to 10.

Table 5
Recognition rates (%) on color FERET dataset

Method	Fb	Dup1	Dup2	Avg.
CID [38]	98.50	88.80	86.40	91.23
RSC [37]	98.75	90.30	89.64	92.90
PCANet [20]	99.20	90.25	90.02	93.16
CenterFace [5]	96.88	77.68	87.72	87.43
SphereFace [36]	98.39	82.16	**90.79**	90.45
DEFNet (origin)	**99.80**	88.79	87.15	91.91
DEFNet (feret)	**99.80**	**91.84**	90.60	**94.08**

4.2.4. Face recognition test on color FERET dataset

We further conduct the experiments on the popular color FERET [31] dataset. We follow the recommendation to examine the standard test subsets which constitute a gallery set **fa** for training and three probe sets **fb, dup1** and **dup2** for testing. All the images of standard testing sets are frontal images. The gallery set contains 994 images with one image per subject and the **fb** set contains 992 images taken a few seconds after the gallery image, which is used for testing the facial expression performance. The **dup1** set consists of all the rest 736 frontal images of the subjects captured with glasses or alternative hairstyle at later sessions. The **dup2** set is a subset of **dup1** which contains 228 images taken at least 540 days later after the gallery image of the subject.

The implementation details are described as followings. We first crop each image into shape 176 × 176 using Dlib face detector. Then we employ the 5 × 5 patch configuration to obtain feature representations for the gallery set and all three probe sets. Note we adopt a combined HOG cell size strategy to extract the HOG features from the outputs. Specifically, for the first half feature maps produced from the final stage, a cell size of 2 × 2 is applied to extract the dense features. On the other half feature maps, we adopt the normal 8 × 8 cell to extract more sparse features. We find this strategy helps to improve the recognition performance with few training samples per subject. The dimension of extracted HOG features is reduced by whiten PCA where the projection matrix is learned from the gallery set and is fixed on all probe sets. The number of selected components is 993. We conduct the experiments with two sets of learned filter banks. One is learned from the original VGGFace subset, denoted as DEFNet followed by origin in parentheses. The other is learned from the 994 gallery images of color FERET, denoted as DEFNet followed by feret in parentheses. For the compared deep CNN-based methods, we deploy the pre-trained models in previous experiments to extract 512-dimensional feature representations of all the images. The dimension of feature representations is further reduced by PCA. Then, the reduced features are given to a NN classifier with cosine distance metric. Here, the inputs of all the methods are color images without gray-scale conversion. The recognition results compared with five other methods are presented in Table 5.

4.2.5. Face verification on LFW dataset

Finally, we apply the proposed DEFNet on the benchmark LFW [32] dataset for face verification. The LFW dataset contains 13,233 images of 5749 people, all of which are under unconstraint environment. We follow the “unsupervised setting”, which is most appropriate for our approach. Under the view 2, there are 10 subsets of pairs of images, and each subset contains 300 matched pairs and 300 unmatched pairs. We evaluate DEFNet with other unsupervised methods using 10-fold cross-validation. Specifically, each time, 9 partitions of them are used for training and the best

Table 6
Verification rates (%) on LFW using the unsupervised setting

Method	Accuracy
POEM [39]	82.70 ± 0.59
High-dim. LE [40]	84.58
PCANet [20]	85.20 ± 1.46
OCLBP [41]	$\mathbf{86.66 \pm 0.30}$
DEFNet	84.60 ± 0.95
DEFNet (fusion)	86.55 ± 1.72

Table 7
Results of computational time (s) on training and test

Methods		Training				Test
		L	R	C	Total	
DEFNet	3×3	306	725	217	1248	1.9
	5×5	568	1816	357	2741	2.5
	7×7	506	2418	518	3442	3.2
VGGNet [13]		–	–	–	5437	8.1
SphereNet [36]		–	–	–	2796	4.0

threshold is determined on those 5400 pairs of images. The rest partition of image pairs is only used for testing. We repeat the procedure 10 times and report the mean verification accuracy. The images used are aligned by deep funneling, provided in [42] and they are cropped and resized to 192×160 pixels.

Our approach is implemented as followings. We apply the pre-trained model on VGGFace dataset with patch size 5×5 to obtain feature representations. For HOG part, we choose the non-overlapped blocks, i.e. 1×1 cell per block. The cell size is chosen to 8×8, which makes 30 blocks per feature map in our case. A fusion of local HOG features and global HOG features is used to calculate the distance of each pair of images. Specifically, denote the outputs of image I produced by DEFNet as $\overset{l_3}{\underset{k=1}{}}\{f_{I,k}\}$, where l_3 is the number of filters in the final stage. Suppose the number of blocks of each feature map is B, the correspond HOG feature vectors of $f_{I,k}$ are $\overset{B}{\underset{b=1}{}}\{h^b_{I,k}\}$. we gather all the HOG feature vectors with the same block index b of each feature map, then we can obtain $H^b_I = [h^b_{I,1}, h^b_{I,2} \cdots h^b_{I,l_3}]$. We call H^b_I as one local HOG feature vector of image I. By grouping the local feature vectors with index b of all the training images, PCA is applied to project H^b_I into $\tilde{H}^b_I$. Concatenate all the reduced local HOG feature vectors $\overset{B}{\underset{b=1}{}}\{\tilde{H}^b_I\}$ to obtain local HOG representation H^{local}_I of image I. Denote the PCA-projected HOG feature in Eq. (34) as global HOG representation H^{global}_I. The final distance of one pair of images (u, v) is the weighted sum of global distance and local distance, formulated as $w * \text{dist}^{\text{local}}_{u,v}(\cdot) + (1-w) * \text{dist}^{\text{global}}_{u,v}(\cdot)$, where w is the fusion weight and $\text{dist}^{\text{local}}_{u,v}(\cdot)$ is the cosine distance between H^{local}_u and H^{local}_v. The concatenated H^{local}_u and H^{global}_u in our experiment are 2769 and 349 dimensions, respectively. And the optimal fusion weight w is set to 0.65 from experimental results.

The results on LFW are listed in Table 6. Note the DEFNet with fusion in parentheses represents the DEFNet applying the fusion strategy on local features and global features to calculate pair-wise distance. One can find the fusion-DEFNet outperforms the original DEFNet. This boost on performance may owe to the utilization of local HOG features of each region of multiple feature maps. Since the fusion weight of local distance is larger than that of global distance, it means local HOG representations take a more important role than global HOG representations in face verification. One can also observe fusion-DEFNet achieves an accuracy of 86.55%, which is competitive to the state-of-the-art method on LFW based on the unsupervised setting. It proves that the proposed DEFNet can learn invariant and discriminative features under unconstraint environment.

4.3. Experiments on computational cost

One highlight feature of our proposed approach is it can be efficiently implemented on GPU which utilizes the batch matrix multiplication for acceleration. The implementations of our approach are based on PyTorch with one NVIDIA 2080Ti GPU. We compare the computational cost of training phase and test phase in our approach with two other deep CNNs architectures used in previous experiments, i.e. VGGNet and SphereNet. The experiment configurations are exactly the same as in Section 4.2.1. In training, all the methods are run on GPU. In the testing mode, our DEFNet is tested on CPU while other methods are tested on GPU. We consider the total prediction time on all the test samples as testing computational time. For a fair comparison, the applied batch size of each method is selected to make full use of the GPU memory individually. Specifically, we divide the training phase of our method into three parts i.e. learning filter banks (L), obtaining feature representations (R) and building classifier (C), and we record the computational time of each part separately. The mean computational time of 10-time experiments is used as a result given in Table 7.

From Table 7, we can see when using patch size 3×3, the proposed approach is most efficient among all the compared methods. Besides, considering our approach is tested on CPU, the testing cost is quite tiny compared with other deep CNNs. In terms of training cost, the computation expenses mainly come from propagating the network to compute feature representations. Note the expense on filter learning of case 5×5 is slightly larger than that of case 7×7 in our approach. The reason is the stride in the latter is 2 which reduces the spatial dimension of all activations by half in the intermediate stage. This reduction on spatial dimension helps to offset the increase in channel dimension of outputs, thus a larger batch size of inputs can be fed to GPU compared to the 5×5 case. Overall, the total training cost and testing cost of our approach is competitive to those of deep CNNs, especially when adopting small patch size in training.

5. Analysis on the strategy of proposed deep eigen-filters approach

In this section, we provide theatrical analysis to justify the strategy used in the proposed approach. The analysis includes two aspects, i.e. filter-mean subtraction and multi-structure filters learning.

5.1. Filter-mean vs patch-mean

Filter-mean subtraction is employed during the whole procedure of learning filters and obtaining feature representations. In our work, filter-mean subtraction is opposite to the patch-mean subtraction introduced in [20]. From Eq. (10), filter-mean is the statistics moment along the second axis of the patch vector matrix, whereas the patch mean is the average of each patch vector, formulated as $\mu' = (1^T m_{i,j})/q^2$, where $m_{i,j}$ is the jth patch vector of image I_i and 1 is a vector of ones with the dimension of patch size q squared. When obtaining feature representation, filter-mean is prefixed from filter learning while the patch-mean removal is implemented online for each sample.

Consider the case when applying 1×1 convolution, filter-mean removal represents subtracting the mean of each channel of outputs estimated from the training dataset, which corresponds to the batch normalization [43] applied in CNNs. However, the patch mean is equal to the average output along the channel dimension. We adopt the filter-mean following the idea in PCA where training samples are centered along the feature dimension for computing covariance matrix before projected into a lower-dimensional

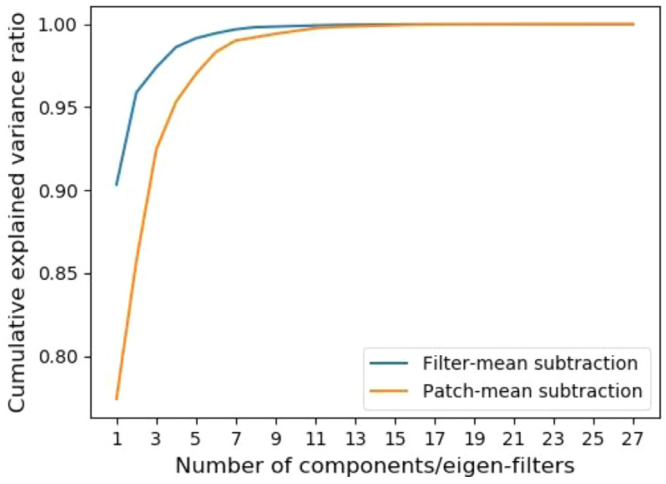


Fig. 10. Comparison on cumulative explained variance ratio between two mean removal method.

space. In the case of 3×3 configuration, in the first stage, we select top j components from the total 27 components of the covariance matrix. The cumulative explained variance can be defined as $\sigma_j = \sum_{i=1}^{j} \lambda_i / \sum_{i=1}^{27} \lambda_i$, where λ_i is the eigenvalue of the ith component calculated from Eq. (13). Fig. 10 shows the comparison of σ_j between filter-mean subtraction and patch-mean subtraction against the increasing number of components. One can see the filter-mean removed covariance matrix owns more dominant information from the first component than that of the patch-mean removed case. Hence, it takes fewer filters to reach a higher cumulative variance ratio in filter selection.

5.2. Why learning multiple stacked layers with multi-structure filters

In [20], researchers observed that the performance of stacking two PCA layers on the input is better than merging two layers into a single layer with an equivalent number of PCA filters. In our work, we also apply a three-stage deep architecture to learn the filter banks. Regarding why using deep architectures, the advantages can be summarized from two perspectives theoretically. From the view of computation, assume a single-stage eigen-filters approach producing $l_1 l_2$ outputs per image with patch size $p \times p$. Given one RGB image as input, it requires learning filters with a total number of $3p^2 l_1 l_2$ parameters. Whereas, by stacking two layers with standard filters, the number of parameters required is reduced to $3p^2 l_1 + p^2 l_2$. When applying our proposed approach, it requires learning two-stage filter banks with $3p^2 l_1 + l_1 p^2 l_2$ parameters. Compared with the single-stage case, we can prove both two approaches of stacking multiple layers have a reduction on the number of learned parameters when $l_2 > 1$, i.e. when the second stage makes sense. This reduction may contribute to a lower chance of overfitting the dataset. One can also see the advantage from the view of computational cost. For simplicity, assume the $l_1 l_2$-channel output has the same spatial size $D_H \times D_W$ as input. The number of multiplications required for single-layer approach is $3p^2 l_1 l_2 D_H D_W$. While for approaches with two stacked layers, the number of multiplications required is shown in Eq. (35):

$$3p^2 l_1 D_H D_W + p^2 l_2 D_H D_W l_1 = p^2 l_1 D_H D_W (3 + l_2) \tag{35}$$

The most right-hand side of Eq. (35) is smaller than $3p^2 l_1 l_2 D_H D_W$ if only $l_2 > 1$. In summary, stacking multiple layers helps to learn the same channel dimension of output with less number of learnable parameters and lower computational cost.

The other benefit of stacking multiple layers is the larger receptive field. For example, when applying the convolution with kernel size 3×3, padding size $p = 1$ and stride $s = 1$ on a 224×224 input. For a single layer PCA, the receptive field of the center point on the output feature map is 3. In the case of two stacked standard layers, the receptive field of that grows to 5. While in our approach, since a max-pooling layer with kernel 2×2 and stride 2 is added between two stages, the receptive field is extended to 8. It's well known that the larger receptive field effectively captures more holistic observations of the object, which results in learning more semantically-related feature representations [15,28].

In our approach, we novelly extend the stacked multi-layer of standard filters to multi-structure filters including channel-wise filters and point-wise filters. One may wonder why using multi-structure filters. We denote the outputs of the first stage as $\{Z_{i,j}\}_{i=1,\ j=1}^{N,\ l_1}$, where $Z_{i,j}$ is the jth feature map of image I_i. $Z_{i,j}$ is produced from the convolution of W_j^1 and I_i, where W_j^1 is the jth eigen-filter in first stage constructed from the eigenvector $\boldsymbol{u_j}$ given in Eq. (14). For each feature map with index j, there is a corresponding eigenvector $\boldsymbol{u_j}$. The eigenvectors are sorted in descending order regarding its eigenvalues and each eigenvector focuses on specific regions with different weights. We consider the second layer with standard filters as a suboptimal choice. The reason is some features produced by low-ranking eigenvectors in the first stage may fade out when eigen-decomposition is applied to all the feature maps of each image. The channel-wise filters are proposed in order to preserve the discriminant information between different channels of outputs while resisting the redundancy among the training samples. To achieve this, we split $\{Z_i\}_{i=1}^{N}$ into l_1 groups corresponding to the channel index. Each group contains feature maps with the same index from all the training samples. Then, we learn l_2 filters from each group separately.

Based on the concatenated output feature maps Z_i^2 (filter-mean removed) with $l_1 l_2$ channels per image, the point-wise eigen-filters are proposed to reduce the channel dimension without any spatial information loss. By l_3 filters of final stage and the condition $l_3 \ll l_2 l_1$, the channel dimension of outputs in second stage is reduced from $l_2 l_1$ to l_3 following the same characteristic as 1×1 convolution in CNNs. We denote the learned point-wise filters in final stage as $\{W_k^3\}_{k=1}^{l_3}$. The convolution of each filter W_k^3 and Z_i^2 can be viewed as a weighted sum of Z_i^2 across the channel dimension, where the weight $c_{k,j}$ is the jth value in the vectorized W_k^3 assigned to jth channel of Z_i^2 . Moreover, the number of learned parameters required by point-wise filters is only $l_1 l_2 l_3$, which is much smaller than the $p^2 l_1 l_2 l_3$ parameters required by standard filters.

6. Conclusion

The traditional deep CNNs suffer the drawbacks of a large number of learnable parameters and expensive computational cost. Based on these problems, we propose a novel three-stage approach to learn multi-structure filters from training data alternatively. Inspired from variations of standard convolution in CNNs i.e. 1×1 convolution and depth-wise separable convolution, different structures of filters are designed for filter learning in each stage, then followed by eigendecomposition to obtain the eigen-filters. We observe the linear relation between our learned filters and convolution kernels of pre-trained CNNs. Then a reconstruction error-based criterion is proposed to select and determine the most representative eigen-filers. Based on learned three-stage filters, we build a network (DEFNet) followed by HOG-based feature extraction for feature representation. The proposed DEFNet shows competitive performance with superior computational cost on face recognition. It outperforms other deep CNNs-based methods on subsets of unconstrained datasets including VGGFace and FaceScrub. The pre-trained model generalizes well on the controlled datasets including AR and color FERET and it shows great

robustness to facial expression and illumination. For unconstraint face verification, it also achieves a promising result on LFW. Consider the unsupervised setting of our approach, it provides a perspective connecting deep CNNs and traditional feature descriptors, which is highly encouraging to apply especially the cases with small size training sets.

In the future, we will work on integrating our learned multi-structure filters with conventional deep CNNs. In this case, we may additionally consider the functionalities and effect of other units of CNNs, e.g. non-linear activation functions and fully-connected layers. Besides, when viewing the proposed three-stage layers as a module, whether it benefits from stacking multiple modules for building a much deeper architecture is thought-provoking.

Declaration of Competing Interest

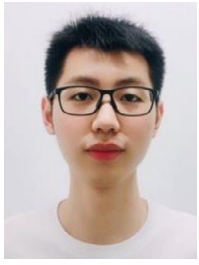
None.

Acknowledgments

This work is supported by Hong Kong Research Grants Council (Project C1007-15G) and City University of Hong Kong (Projects 9610034 and 9610460). The authors would like to thank Dr. Xuefei Zhe for his valuable suggestion.

References

[1] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, J. Cai, T. Chen, Recent advances in convolutional neural networks, Pattern Recognit. 77 (2018) 354–377.
[2] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, 2014 arXiv:1409.1556.
[3] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet Classification with Deep Convolutional Neural Networks, in: NIPS, pp. 1097-1105.
[4] H. Wang, Y. Wang, Z. Zhou, X. Ji, D. Gong, J. Zhou, Z. Li, W. Liu, in: Cosface: Large Margin Cosine Loss for Deep Face Recognition, CVPR, 2018, pp. 5265–5274.
[5] Y. Wen, K. Zhang, Z. Li, Y. Qiao, in: A Discriminative Feature Learning Approach for Deep Face Recognition, EECV, 2016, pp. 499–515.
[6] X. Zhe, S. Chen, H. Yan, Directional statistics-based deep metric learning for image classification and retrieval, Pattern Recognit. 93 (2019) 113–123.
[7] J. Yu, M. Tan, H. Zhang, D. Tao, Y. Rui, Hierarchical deep click feature prediction for fine-grained image recognition, IEEE Trans. Pattern Anal. Mach. Intell. (2019).
[8] Z. Zhang, C. Xu, J. Yang, Y. Tai, L. Chen, Deep hierarchical guidance and regularization learning for end-to-end depth estimation, Pattern Recognit. 83 (2018) 430–442.
[9] B. Li, Y. Dai, M. He, Monocular depth estimation with hierarchical fusion of dilated cnns and soft-weighted-sum inference, Pattern Recognit. 83 (2018) 328–339.
[10] T. Ahonen, A. Hadid, M. Pietikainen, Face description with local binary patterns: application to face recognition, IEEE Trans. Pattern Anal. Mach. Intell. (2006) 2037–2041.
[11] N. Danal, in: Histgram of Oriented Gradients for Human Detection, CVPR, 2005, pp. 886–893.
[12] K. He, X. Zhang, S. Ren, J. Sun, in: Deep Residual Learning for Image Recognition, CVPR, 2016, pp. 770–778.
[13] O.M. Parkhi, A. Vedaldi, A. Zisserman, in: Deep Face Recognition, BMVC, 2015, p. 6.
[14] T. Kobayashi, in: Analyzing Filters Toward Efficient ConvNet, CVPR, 2018, pp. 5619–5628.
[15] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT press, 2016.
[16] J. Bruna, S. Mallat, Invariant scattering convolution networks, IEEE Trans. Pattern Anal. Mach. Intell. 35 (2013) 1872–1886.
[17] J.-H. Jacobsen, J. van Gemert, Z. Lou, A.W. Smeulders, in: Structured Receptive Fields in CNNS, CVPR, 2016, pp. 2610–2619.
[18] C. Hong, J. Yu, J. Zhang, X. Jin, K.-H. Lee, Multi-modal face pose estimation with multi-task manifold deep learning, IEEE Trans. Ind. Inf. (2018).
[19] J. Yu, Y. Rui, D. Tao, Click prediction for web image reranking using multimodal sparse coding, IEEE Trans. Image Process. 23 (2014) 2019–2032.
[20] T.-H. Chan, K. Jia, S. Gao, J. Lu, Z. Zeng, Y. Ma, PCANet: A simple deep learning baseline for image classification? IEEE Trans. Image Process. 24 (2015) 5017–5032.
[21] K. Sun, J. Zhang, H. Yong, J. Liu, FPCANet: Fisher discrimination for principal component analysis network, Knowl.-Based Syst. 166 (2019) 108–117.
[22] D. Yu, X.-J. Wu, 2DPCANet: a deep leaning network for face recognition, Multimedia Tools Appl. 77 (2018) 12919–12934.
[23] J. Zhang, J. Yu, D. Tao, Local deep-feature alignment for unsupervised dimension reduction, IEEE Trans. Image Process. 27 (2018) 2420–2432.
[24] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, in: Going Deeper with Convolutions, CVPR, 2015, pp. 1–9.
[25] M. Lin, Q. Chen, S. Yan, Network in network, 2013, arXiv:1312.4400.
[26] A.G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, H. Adam, Mobilenets: efficient convolutional neural networks for mobile vision applications, 2017, arXiv:1704.04861.
[27] F. Chollet, in: Xception: Deep Learning with Depthwise Separable Convolutions, CVPR, 2017, pp. 1251–1258.
[28] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, Z. Wojna, in: Rethinking the Inception Architecture for Computer Vision, CVPR, 2016, pp. 2818–2826.
[29] H.-W. Ng, S. Winkler, in: A Data-Driven Approach to Cleaning Large Face Datasets, ICIP, 2014, pp. 343–347.
[30] A.M Martinez, The AR face database, 1998 CVC Technical Report 24.
[31] P.J. Phillips, H. Moon, S.A. Rizvi, P.J. Rauss, The FERET evaluation methodology for face-recognition algorithms, IEEE Trans. Pattern Anal. Mach. Intell. 22 (2000) 1090–1104.
[32] G.B. Huang, M. Mattar, T. Berg, E. Learned-Miller, in: Labeled faces in the wild, University of Massachusetts, Amherst, 2007, pp. 07–49. Technical Report.
[33] L. Sifre, S. Mallat, Rigid-motion scattering for image classification, 2014.
[34] O. Déniz, G. Bueno, J. Salido, F. De la Torre, Face recognition using histograms of oriented gradients, Pattern Recognit. Lett. 32 (2011) 1598–1603.
[35] D.E. King, Dlib-ml: a machine learning toolkit, J. Mach. Learn. Res. 10 (2009) 1755–1758.
[36] W. Liu, Y. Wen, Z. Yu, M. Li, B. Raj, L. Song, in: Sphereface: Deep Hypersphere Embedding for Face Recognition, CVPR, 2017, pp. 212–220.
[37] M. Yang, L. Zhang, J. Yang, D. Zhang, in: Robust Sparse Coding for Face Recognition, CVPR, 2011, pp. 625–632.
[38] Z. Liu, J. Yang, C. Liu, Extracting multiple features in the CID color space for face recognition, IEEE Trans. Image Process. 19 (2010) 2502–2509.
[39] N.-S. Vu, A. Caplier, Enhanced patterns of oriented edge magnitudes for face recognition and image matching, IEEE Trans. Image Process. 21 (2011) 1352–1365.
[40] D. Chen, X. Cao, F. Wen, J. Sun, in: Blessing of Dimensionality: High-Dimensional Feature and its Efficient Compression for Face Verification, CVPR, 2013, pp. 3025–3032.
[41] O. Barkan, J. Weill, L. Wolf, H. Aronowitz, in: Fast High Dimensional Vector Multiplication Face Recognition, CVPR, 2013, pp. 1960–1967.
[42] G. Huang, M. Mattar, H. Lee, E.G. Learned-Miller, in: Learning to Align from Scratch, NIPS, 2012, pp. 764–772.
[43] S. Ioffe, C. Szegedy, in: Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, ICML, 2015, pp. 448–456.
[44] Shu Chang, Xiaoqing Ding, Chi Fang, Histogram of the oriented gradient for face recognition, Tsinghua Sci. Technol. 16.2 (2011) 216–224.



Ming Zhang received his B.Eng. degree in Automation from Nanjing Normal University, Nanjing, China in 2017, and MSc. degree in Electronic Information Engineering with Distinction from City University of Hong Kong, Hong Kong, China in 2018. Currently, he is working towards the Ph.D. degree in the Department of Electrical Engineering, City University of Hong Kong. His research interests include machine learning and computer vision.

Sheheryar Khan is currently a Postdoctoral fellow at CUHK lab of AI in radiology (CLAIR) in Department of Imaging and Interventional Radiology, The Chinese University of Hong Kong. He received his Ph.D. degree in Electrical Engineering from City University of Hong Kong and MSc degree in Signal Processing from Lancaster University, UK with distinction in 2010. He also served as a lecturer in COMSATS University Islamabad, Pakistan. His research interests include computer vision, medical imaging and machine learning.

Hong Yan received his Ph.D. degree from Yale University. He was Professor of Imaging Science at the University of Sydney and is currently Chair Professor of Computer Engineering at City University of Hong Kong. His research interests include image processing, pattern recognition and bioinformatics, and he has over 600 journal and conference publications in these areas. Professor Yan is an IEEE Fellow and IAPR Fellow, and he received the 2016 Norbert Wiener Award from the IEEE SMC Society for contributions to image and biomolecular pattern recognition techniques.