A Sparse Target Matrix Generation Based Unsupervised Feature Learning Algorithm for Image Classification

Dan Zhao[1], Baolong Guo[1], Yunyi Yan[1]
[1] School of Aerospace Science and Technology, Xidian University
Xi'an 710071, Shaanxi – P.R. China
[e-mail: fengjiran@foxmail.com; {blguo, yyyan}@xidian.edu.cn]
*Corresponding author: Baolong Guo

Received Novermber 9, 2017; revised January 20, 2018; accepted February 7, 2018; published June 30, 2018

Abstract

Unsupervised learning has shown good performance on image, video and audio classification tasks, and much progress has been made so far. It studies how systems can learn to represent particular input patterns in a way that reflects the statistical structure of the overall collection of input patterns. Many promising deep learning systems are commonly trained by the greedy layerwise unsupervised learning manner. The performance of these deep learning architectures benefits from the unsupervised learning ability to disentangling the abstractions and picking out the useful features. However, the existing unsupervised learning algorithms are often difficult to train partly because of the requirement of extensive hyperparameters. The tuning of these hyperparameters is a laborious task that requires expert knowledge, rules of thumb or extensive search. In this paper, we propose a simple and effective unsupervised feature learning algorithm for image classification, which exploits an explicit optimizing way for population and lifetime sparsity. Firstly, a sparse target matrix is built by the competitive rules. Then, the sparse features are optimized by means of minimizing the Euclidean norm (L_2) error between the sparse target and the competitive layer outputs. Finally, a classifier is trained using the obtained sparse features. Experimental results show that the proposed method achieves good performance for image classification, and provides discriminative features that generalize well.

Keywords: Neural network, unsupervised learning, sparse feature, classification

This research was supported by the National Natural Science Foundation of China under Grants No. 61571346, 61671356.

http://doi.org/10.3837/tiis.2018.06.020 **ISSN : 1976-7277**

1. Introduction

In the machine learning community, feature engineering plays an important role in the applications of many machine learning algorithms. The performance of many machine learning algorithms is heavily dependent on the choice of data features on which they are applied. For this reason, many significant efforts have been devoted to designing appropriate feature representations of data in several fields. In last several decades, a great number of widely used handcrafted computer vision features, such as HOG [1], SURF [2], SIFT [3], Textons [4], RIFT [5], and GLOH [6], have been developed. Although these handcrafted features paved the way for many breakthroughs in computer vision, they still have some shortcomings [7]. First, these features need a huge amount of domain-expert knowledge and time consuming hand-tuning. Second, the features in one domain can not be generalized to other domains. Both of these issues are universally true in most domains. Therefore, it is highly desirable to use the automatic feature learning algorithms from data instead of handcrafting, to make machine learning algorithms less dependent on feature engineering.

In recent years, unsupervised feature learning method has been extensively used in a wide spectrum of computer vision applications along with the development of deep learning. It becomes a promising tool to automatically learn effective features from unlabeled data. Theoretical studies have indicated that unsupervised feature learning is helpful in greedy layerwise pre-training phase of many deep learning algorithms [10-13]. The features obtained by unsupervised learning algorithms have shown to achieve a similar or even better performance than manually designing ones [8-9]. Though researches on deep learning have shown that deep architectures could learn multi-level hierarchies of features and obtain constantly improvements in many fields, the shallow architectures are still favored and commonly used. There are two reasons for this. First, shallow architectures could be used as building blocks of deep architectures, so they can influence the whole performance of the deep architectures. Second, compared with deep architectures, shallow architectures are simple and easy to use.

Among the well-behaved unsupervised features, the sparse feature is popular not only in computational neuroscience and signal processing but also in machine learning. Commonly, in order to obtain sparse features, the core idea of many unsupervised feature learning algorithms is imposing the L_1-norm constraint to the objective function. But the objective function with L_1-norm is nonsmooth, which leads to optimization problems. Another criticism to current unsupervised feature learning algorithms is that they require many hyperparameters [14]. Tuning these hyperparameters is a laborious task that requires expert knowledge, rules of thumb or extensive search. For example, the sparse RBM (Restricted Boltzmann Machines) has up to half a dozen hyperparameters and an intractable objective function. Thus, it is difficult to tune the hyperparameters and monitor the convergence of sparse RBM. Therefore, automatic sparse feature learning algorithms with few hyperparameters are desirable. High computational complexity is also a drawback

of many unsupervised feature learning algorithms. ICA (Independent Component Analysis) requires an expensive orthogonalization to be computed at each iteration. SC (Sparse Coding) has an expensive inference, which requires a prohibitive iterative optimization. Significant amount of works have been done in order to overcome this limitation [19]. For example, PSD (Predictive Sparse Decomposition) [20] is a successful variant of sparse coding, which uses a predictor to approximate the sparse representation and solves the computationally expensive encoding step of sparse coding.

Therefore, the motivation of this paper can be summarized as three respects. Firstly, unsupervised feature learning has extensively used in a wide spectrum of computer vision applications along with the development of deep learning. Unsupervised feature learning has shown to be helpful in greedy layerwise pre-training of deep architectures. Secondly, sparsity is among the desirable properties of a good output representation. In order to obtain sparse features, the core idea of many algorithms is imposing the L_1-norm constraint to the objective function. But the objective function with L_1-norm is nonsmooth, which leads to optimization problems, and L_1 penalty does not optimize for an explicit level of sparsity in the features. Thirdly, many current unsupervised feature learning algorithms are hard to use because they require a good deal of hyperparameter tuning.

In this work, we present a new unsupervised sparse feature learning algorithm based on sparse target matrix generation inspired by the competitive learning algoithm. This algorithm could control the sparse level and has low computational complexity. The competitive learning is an artificial neural network learning process. Different neurons in the networks complete to be activated to represent the current input. The purest form of competitive learning is the so-called winner-take-all networks where only one neuron that best represents the input is allowed to be activated.

In the proposed algorithm, the competitive learning is introduced to obtain sparse features without modeling the data distribution. Firstly, a sparse target matrix is built by the competitive rules. Then, the sparse features are optimized by means of minimizing the L_2 error between the sparse target and the competitive layer outputs. Finally, a softmax classifier is trained using the obtained sparse features.

The rest of this paper is organized as follows. Relevant research background is introduced in Section 2. Section 3 presents the details of the proposed unsupervised sparse feature learning algorithm. Experiments are carried out and analyzed in Section 4 to demonstrate the effectiveness of the proposed method. Section 5 outlines the paper.

2. Related Works

There are a lot of commonly used unsupervised feature learning algorithms. RBM [10] and sparse auto-encoder [11] are both generative stochastic artificial neural networks that can learn a probability distribution over the datasets. However, they have a lot of hyperparameters to tune. RSSL (Robust Structured Subspace Learning) [35] algorithm integrates image understanding and feature learning into a joint learning framework. The learnt subspace is adopted as an intermediate space ro reduce the semantic gap between the low-level visual features and the high-level semantics. Other algorithms such as ICA

[16-17], reconstruction ICA [18], sparse filtering [17], EPLS (Enfocing Population and Lifetime Sparsity) [30] and OMP-k (Orthogonal Matching Pursuit) [22, 24] have also been used to extract sparse feature representations. Although ICA variants provide good results at object recognition tasks [15, 18], the algorithms scale poorly to large datasets and rely on whitening operation. The sparse filtering works by imposing contraints on sparse distribution of the features. However, it does not consider the property of the mapping learning matrix itself. EPLS is a new algorithm of optimizing for sparsity without modeling the data distribution, but it does not explicitly control the sparsity level of the features, so that this algorithm is not flexible enough. These algorithms could be divided into two categories: explicitly modeling or not the input data distribution. Sparse RBM, sparse auto-encoder, sparse coding, PSD, OMP-k and reconstruction ICA explicitly model the input data distribution by minimizing the reconstruction error. Although learning a good approximation of the data distribution may be desirable, algorithms such as sparse filtering and EPLS show that this seems not so important if the goal is to have a discriminative system. Sparse filtering and EPLS do not attempt to explicitly model the input distribution but focus on the properties of the output distribution instead. In order to remove the redundancies and noise in the high-dimensional features, feature selection technique [36-37] and nonnegative matrix factorzation (NMF) [38] are often adopted to identify a subset of the most useful features.

Among the unsupervised feature learning algorithms, introducing the restriction of sparsity is a commonly used method to restrict the representations. Sparsity is one of the desirable properties of a good output representation [14, 18, 26-27]. It is proposed by modeling the simple cells in the visual cortex [32]. Therefore, sparsity is usually used to learn similar representations to that of visual cortex. Sparse features consist of a large amount of outputs, which response rarely and provide high responses when they do respond. By introducing sparsity, the feature extractor is able to learn low-level structures such as edges and high-level structures such as local curvatures and shapes. Sparsity can be described in terms of population sparsity and lifetime sparsity [17, 29]. Both lifetime and population sparsity are important properties of the output distribution. On the one hand, lifetime sparsity plays an important role in preventing bad solutions such as numerous dead outputs. Lifetime sparsity can overcome such dangerous solutions to ensure similar statistics among outputs [25, 28-29]. On the other hand, population sparsity helps providing a simple imterpretation of input data such as the ones found in early visual areas.

Many algorithms optimize for either one or both sparsity forms in their objective functions. The first successful algorithm introducing sparsity is an energy model based on auto-encoder [34]. In this algorithm, sparsity is achieved by a sparsifying logistic module, which is a nonlinear front-end to the decoder that transforms the code vector into a sparse vector with positive components. In the logistic module, a user-defined parameter is used to control the sparseness of the code. Sparse auto-encoders optimize the target activation allowing to deal with lifetime sparsity, nevertheless, the target activation requires tuning and does not explicitly control the level of population sparsity. Another sparse feature learning model based on auto-encoder is introduced in the sparse encoding symmetric machine (SESM). This model constrains the sparsity of the code by a Student-t prior. For

sparse RBM, its objective function is the sum of a log-likelihood term and a regularization term, where a constant in the regularization term controls the sparseness of the hidden units. The three methods mentioned above seek sparsity using the L_1 penalty and do not optimize the explicit level of sparsity in their outputs. In addition, OMP-k define the level of population sparsity by setting k to the maximum expected number of non-zero elements per output code, but do not explicitly define the proportion of outputs expected to be active at the same time.

In the proposed algorithm, the sparse features are obtained by the means of minimizing the L_2 error between the sparse target martrix and the competitive layer outputs, and the sparse level of the learned features can be adjusted flexibly by the sparseness which is a continuous value which lies in the interval [0, 1]. The learned sparse features have the properties of lifetime and population sparsity. These properties guarantee the output distribution to be composed of only a few activations. Furthermore, the lifetime sparsity ensures the absence of dead outputs.

3. The Sparse Target Matrix Generation Based Unsupervised Feature Learning Algorithm

This section describes the procedure of learning the sparse feature representations in terms of the defined population and lifetime sparsity and the optimization strategies. In the following, Section 3.1 defines the ideally sparse target matrix properties of sparseness, population and lifetime sparsity. Section 3.2 highlights the sparse target matrix generation algorithm in terms of the defined population and lifetime sparsity. Section 3.3 provides the implementation details of the unsupervised sparse feature learning framework. The computational complexity is presented in Section 3.4.

3.1 Properties of the Sparse Target Matrix

For clarity, consider a feature distribution matrix generated by a system over a finite dataset including N training samples and an output of dimensionality N_h. Each row and column of the feature distribution matrix represents a sample and a feature, respectively. The properties of the sparse target matrix are defined as,

(1) *Sparseness*: The number of zero-valued elements divided by the total number of elements in a vector is called the sparseness. Sparsity has become a concept of interest, not only in statistics and signal processing but also in computational neuroscience and machine learning. In the procedure of sparse feature representation learning, the sparseness is an important parameter used to control the number of zero-valued elements in the sparse feature representations.

(2) *Population sparsity*: Each training sample should be represented by only a few active (non-zero) features. Concretely, only a small number of elements should be activated for each row (one sample) in the feature matrix. The proportion of activated elements is the parameter of sparseness mentioned in the property (1). For

instance, an image can be represented by the objects in it, while many possible objects can appear, only a few are typically presented at a single time. This notion is known as the population sparsity and is considered as a principle adopted by the early visual cortex.

(3) *Lifetime sparsity*: The feature vectors must be composed solely of active and inactive units (no intermediate values between two fixed scalars are allowed). Activation is exactly distributed among the N_h outputs. The lifetime sparsity definition is a more strict requirement than the high dispersal concept. The high dispersal only requires the mean squared activations of each feature should be roughly the same for all features. In addition, it attempts to diversify the learned bases, but it does not guarantee the output distribution to be composed of only a few activations. Furthermore, the definition of lifetime sparsity ensures the absence of dead outputs.

The fundamental of the proposed approach is to appropriately generate an sparse target matrix that fulfils the properties (2) and (3), and then learns the parameters of the system by minimizing the L_2 error between the system output and the sparse target matrix generated by the system during training. In this way, both population and lifetime sparsity are optimized in an explicit way.

The key component of the proposed method is how to obtain the sparse target matrix based on the three properties mentioned above. However, to ensure the convergence of the system parameters, the sparse target matrix should be defined as the best approximation of the system outputs by means of L_2 error fulfilling properties (2) and (3).

3.2 The Sparse Target Matrix Generation (STMG) Algorithm

The generation of the sparse target matrix is based on the competitive rules, which consist of the following three basic elements,

(1) A set of neurons that are all the same except for some randomly distributed synaptic weights, and which therefore respond differently to a given set of input pattens;
(2) A limit imposed on the strength of each neuron;
(3) A mechanism that permits the neurons to compete for the right to respond to a given subset of inputs, such that only a small number of output neurons are active (i.e. on) at a time. The neurons that win the competition are called the “winner-take-all” neurons.

Therefore, the individual neuron of the network learns to specialize on ensembles of similar patterns and in so doing become “feature detectors” for different classes of input patterns.

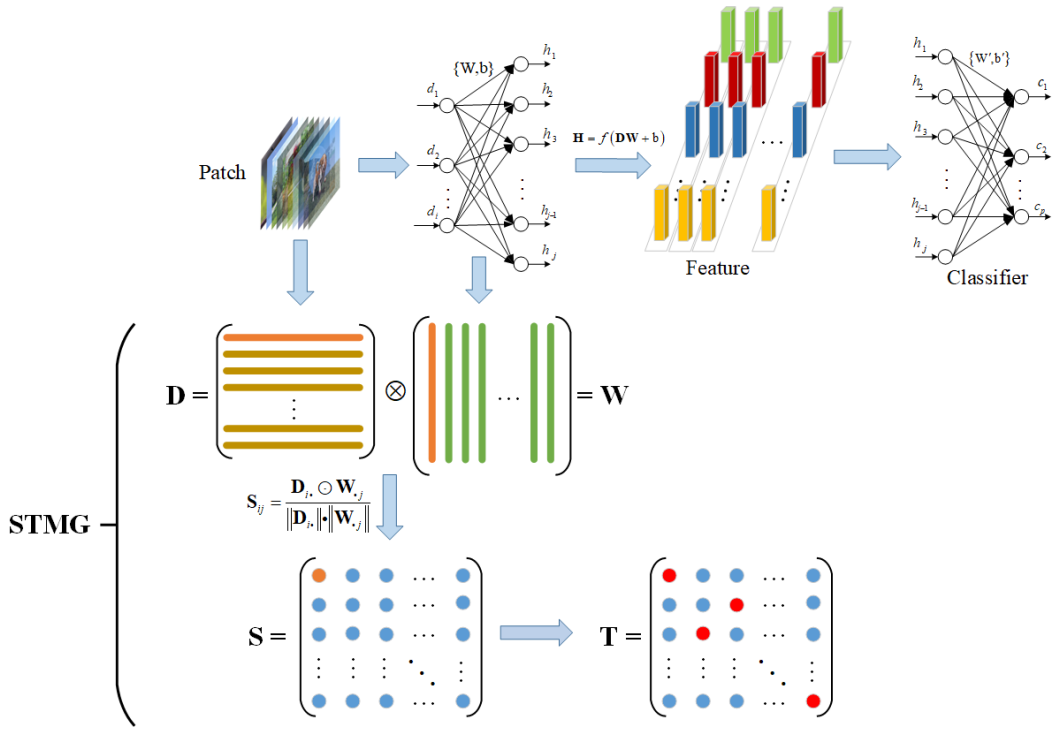


Fig. 1. The neural network architecture and STMG algorithm pipeline

Assuming that there is a neural network parameterized by $\Gamma=\{W,b,W',b'\}$, as illustrated in the upper half part of **Fig. 1**, with activation function f. This neural network contains the input layer, the competitive layer (hidden layer) and the softmax layer (classifier layer), which takes an input vector $\boldsymbol{d}$ and produces the competitive output vector $\boldsymbol{h}$. Let the notation $\boldsymbol{h}_j$ be the jth element of $\boldsymbol{h}$. In the competitive layer, every neuron competes with each other for the right to respond to a given subset of inputs. The classified probabilities $\boldsymbol{c}_p$ are computed by the softmax layer. Define an input data matrix $\mathbf{D}$ composed of N_b rows and N_d columns, where N_b is the batch size such that $N_b \leqslant N$, and N_d is the input dimensionality. The competitive layer output $\mathbf{H}$ is given as follows,

$$\mathbf{H} = f\left(\mathbf{DW} + \mathrm{b}\right) \tag{1}$$

Define $\mathbf{T}$ as the sparse target matrix, which has the same size with the competitive layer output $\mathbf{H}$. Algorithm 1 details the proposed method to generate the sparse target matrix $\mathbf{T}$. For the sake of simplicity, every step of the algorithm where the subscript j (output feature index) appears must be applied $\forall j \in \{1, 2..., N_h\}$.

As shown in the lower half part of **Fig. 1** and Algorithm 1, Line 1 calculates the similarity matrix **S** of input matrix **D** and the weights **W**. In the competitive layer, every competitive neuron is described by a column of weights **W**. The similarity matrix **S** is defined as: (2) where

$$\mathbf{S}_{ij} = \frac{\mathbf{D}_{i\cdot} \odot \mathbf{W}_{\cdot j}}{\left\|\mathbf{D}_{i\cdot}\right\| \bullet \left\|\mathbf{W}_{\cdot j}\right\|} \tag{3}$$

where $\mathbf{D}_{i\cdot}$ is the *i*th row of **D** and $\mathbf{W}_{\cdot j}$ is the *j*th column of **W**, $\odot$ represents inner product. For every input vector $\mathbf{D}_{i\cdot}$, the competitive neuro $\mathbf{S}=\mathbf{D}\otimes\mathbf{W}$ ns compete with each other to find the most similar one to that particular input vector according to the cosine similarity of equation (3).

After computing the similarity of each row in **D** and each column in **W** by the equation (2), the similarity matrix **S** is obtained. Then the similarity matrix **S** is normalized thus its elements all lie in the interval of [0, 1]. The sparse target matrix **T** is set to zero in the beginning. The row vectors ***s***, ***t*** from **S**, **T** are processed individually at each iteration. The crucial steps are performed in Lines 8 to 10. The parameter *p* is the sparseness that can be adjusted, thus *k* is the number of neurons to be activated in the *n*th row of **T**. Line 10 gets the indices of the largest *k* values in ***s*** minus the inhibitor ***a***. The inhibitor ***a*** is a vector that has the same size with ***s***. The elements $\boldsymbol{a}_j$ can be seen as an accumulator that "counts" the number of times an output *j* has been selected, increasing its inhibitor progressively by $\delta=1/\left((1-p)N\right)$ until reaching the maximal inhibition. This prevents the selection of an output that has already been activated $N(1-p)$ times. The rational is that while selecting the largest *k* responses in the similarity matrix **S**, they should be uniformly distributed among all outputs (in order to ensure the lifetime sparsity). Using this strategy, it is demonstrated that the resulting matrix **T** perfectly fulfills properties of lifetime sparsity and population sparsity. In Line 11, the algorithm activates the elements whose indices are *idx* in *n*th row of the target matrix **T**, followed by the update of the inhibitor ***a*** in Line 12. Finally, the sparse target matrix **T** is remapped to the active/inactive values of the corresponding function.

Algorithm 1 The Sparse Target Matrix Generation (STMG) Algorithm

Require: $\mathbf{D}$, $\mathbf{W}$, $\mathbf{a}$, N, p;
Ensure: $\mathbf{T}$, $\mathbf{a}$;
1: Compute the similarity matrix $\mathbf{S}$ of $\mathbf{D}$ and $\mathbf{W}$: $\mathbf{S}_{ij} = \frac{\mathbf{D}_{i\cdot} \odot \mathbf{W}_{\cdot j}}{\|\mathbf{D}_{i\cdot}\| \cdot \|\mathbf{W}_{\cdot j}\|}$
2: $\mathbf{S} = \frac{\mathbf{S} - min(\mathbf{S})}{max(\mathbf{S}) - min(\mathbf{S})}$
3: Set $\mathbf{T} = 0$
4: $\delta = \frac{1}{(1-p)\cdot N}$ // Progressive step of inhibitor $\mathbf{a}$
5: **for** $n = 1 \to N_b$ **do**
6: $\mathbf{s} = \mathbf{S}_n$ // The nth row of $\mathbf{S}$
7: $\mathbf{t} = \mathbf{T}_n$ // The nth row of $\mathbf{T}$
8: $k = (1-p) \cdot N_h$
9: $idx = argsort(\mathbf{s} - \mathbf{a})$ // Get the indices of $(\mathbf{s} - \mathbf{a})$
10: $idx = idx[0 : k-1]$ // Get the indices of largest k value in $(\mathbf{s} - \mathbf{a})$
11: $\mathbf{t}[idx] = 1$
12: $\mathbf{a}[idx] = \mathbf{a}[idx] + \delta$ // Update $\mathbf{a}$
13: **end for**
14: Remap $\mathbf{T}$ to active/inactive values
15: **return** $\mathbf{T}$;

3.3 STMG Based Unsupervised Feature Learning Framework

The STMG based unsupervised feature learning framework is described in this section. At high-level, the framework performs the following steps to learn the sparse feature representations.

(1) Collect a set of small patches from the training set randomly;
(2) Apply a pre-processing stage to the patches;
(3) Learn a feature-mapping from input patches to sparse feature representations using the unsupervised feature learning framework, as shown in **Fig. 2.** Given the input data $\mathbf{D}$, the algorithm computes the similarity matrix $\mathbf{S}$ and competitive layer output $\mathbf{H}$, invokes STMG to generate the sparse target $\mathbf{T}$ and then learns its parameters $\mathbf{\Gamma}$ by minimizing the error $\mathbf{E} = \|\mathbf{H} - \mathbf{T}\|_2^2$.

(4) After the learning process, the sparse features for a new image are extracted using the learned feature-mapping by convolution and pooling operations.

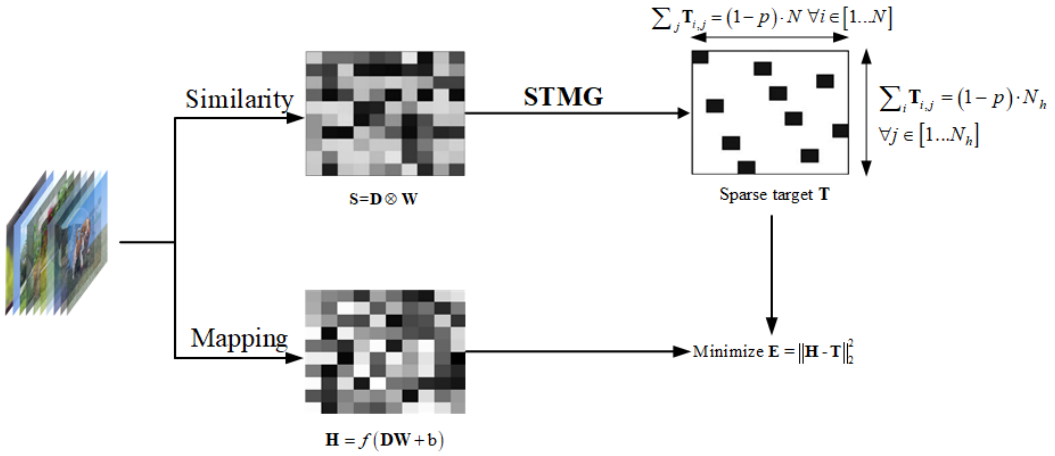


Fig. 2. Pipeline of the sparse feature representations learning algorithm

3.3.1 Patch Extraction and Pre-processing

Let $\mathbf{I} = \left\{I^{(1)}, I^{(2)}, ..., I^{(m)}\right\}, I^{(i)} \in \mathbf{R}^{n\times n\times d}$ be the training set (d is the channel). The system begins by extracting patches at random locations of all the images in the training set. Each patch has a dimension $w \times w$ and d channels, with w referred to the "receptive field size". Each $w \times w$ patch can be represented as a vector in $\mathbf{R}^{N_d}$ of pixel intensity values, with $N_d = w\times w\times d$. Then the N randomly extracted patches construct a dataset $\mathbf{D}=\left\{\boldsymbol{d}^{(1)}, \boldsymbol{d}^{(2)}, ..., \boldsymbol{d}^{(N)}\right\}, \boldsymbol{d}^{(i)} \in \mathbf{R}^{N_d}$.

Usually, several simple normalization steps are performed before attempting to generate features from the dataset. In this section, the dataset **D** is preprocessed by substracting the mean and dividing by the stardand deviation, which is equivalent to the local brightness and contrast normalization. The dataset is optionally whitened.

3.3.2 Unsupervised Feature Learning

After pre-processing, the unsupervised feature learning algorithm is used to discover sparse features from the patches. The unsupervised features learning algorithm can be seen as a "black box" that takes the dataset **D** and outputs a feature extractor $g: \mathbf{R}^{N_d} \rightarrow \mathbf{R}^{N_h}$ that maps an input vector $\boldsymbol{d}^{(i)}$ to a new vector of N_h features.

The feature extractor is learned by means of the mini-batch stochastic gradient descent (SGD) method, as shown in Algorithm 2. The mini-batch size N_b can be set to any value. In the following experiments N_b = 1000. Starting with $\boldsymbol{\Gamma}$ that is set to small random values (Line 1). At each epoch, we shuffle the samples of the training set (Line 3), reset the inhibitor $\boldsymbol{a}$ to zero (Line 4) and process all mini-batches.

For each mini-batch b, samples $\mathbf{D}^{(b)}$ are selected (Line 6), the output $\mathbf{H}^{(b)}$ is computed (Line 7) and the STMG algorithm is invoked to compute $\mathbf{T}^{(b)}$, followed by the update of the inhibitor $\boldsymbol{a}$ (Line 8). After that, the gradient of the L_2 error between $\mathbf{H}^{(b)}$ and $\mathbf{T}^{(b)}$ is

computed (Line 9). The system parameters are then updated to minimize the L_2 error $\mathbf{E}^{(b)}$ (Line 10). This whole procedure is repeated until a stop condition (when the relative decrement error between epochs is small ($<10^{-6}$)) is met. This unsupervised feature learning optimization strategy is valid for any activation function.

Algorithm 2 Standard Unsupervised Learning Method

Require: $\mathbf{D}$, η;
Ensure: Γ;
1: Γ=small random values sampled from Normal Distribution
2: **repeat**
3: Shuffle $\mathbf{D}$ randomly
4: $\mathbf{a} = 0$
5: **for** $b = 1 \rightarrow \lfloor N/N_b \rfloor$ **do**
6: Select mini-batch samples $\mathbf{D}^{(b)}$
7: $\mathbf{H}^{(b)} = f(\mathbf{D}^{(b)}, \Gamma)$
8: $(\mathbf{T}^{(b)}, \mathbf{a}) = STMG(\mathbf{D}^{(b)}, \mathbf{W}, \mathbf{a}, N, p)$
9: $\mathbf{G} = \nabla_{\Gamma} \| \mathbf{H}^{(b)} - \mathbf{T}^{(b)} \|_2^2$
10: $\Gamma = \Gamma - \eta \mathbf{G}$
11: **end for**
12: **until** stop condition verified

3.3.3 Feature Extraction and Image Classification

The above steps yield a feature extractor $g : \mathbf{R}^{N_d} \rightarrow \mathbf{R}^{N_h}$ that transforms a $w \times w$ input patch into a new sparse feature representation $\boldsymbol{h} \in \mathbf{R}^{N_h}$. This feature extractor can be applied to the labeled training images for classification. Given an image of $n \times n$ pixels (with d channels), a feature representation with dimension of $(n\text{-}w+1) \times (n\text{-}w+1) \times N_h$ is obtained by the convolution operation, where the step-size (or "stride") s is equal to 1 pixel.

Before classification, the dimensionality of the image representation should be typically reduced by pooling operation. For the stride of s = 1, the feature mapping produces a dimension of $(n\text{-}w+1) \times (n\text{-}w+1) \times N_h$ representation. The pooling works by splitting a feature mapping image into four equal-sized quadrants, and either summing up or getting the maximum of each quadrant into a vector. This yields a reduced (N_h-dimension) representation of each quadrant, for a total of $4N_h$ features that used for classification.

Given the pooled feature vectors for each training image and the corresponding label, a standard linear classifier such as softmax can be applied for classification.

3.4 Computational Complexity Analysis

In this section, the computational complexity of the STMG algorithm is theoretically analyzed. The common way to express the complexity of one algorithm is using the big O

notation.

As stated in Algorithm 1, the similarity matrix is first computed. The corresponding time complexity is $O\left(N_b N_h N_d\right)$, where N_b is the mini-batch size, N_h is the dimension of the features and N_d is the dimension of samples. Then the proposed algorithm iteratively creates the sparse target matrix T that fulfills the properties pre-defined in Section 3.1. In each iteration, the major step is a sort algorithm whose time complexity is $O\left(N_h lgN_h\right)$. Thus the time complexity of all iterations is $O\left(N_b N_h lgN_h\right)$. In the STMG algorithm, creating the sparse target T is analogous to solving an assignment problem. The Hungarian method [33] is a combinatorial optimization algorithm, which solves the assignment problems. However, its time complexity $O\left(\left(N_b N_h\right)^{3/2}\right)$ is prohibitive. Thus the total time complexity of Algorithm 1 is $O\left(N_b N_h N_d + N_b N_h lgN_h\right)$.

The memory complexity is related to the mini-batch size N_b. Consequently, the method can scale gracefully to very large datasets. Theoretically, it requires memory to store the mini-batch input data $\mathbf{D}^{(b)}$ ($N_b N_d$ elements), similarity matrix $\mathbf{S}$ ($N_b N_h$ elements), output $\mathbf{H}^{(b)}$ ($N_b N_h$ elements), sparse target $\mathbf{T}$ ($N_b N_h$ elements) and the system parameters to optimize $\mathbf{\Gamma}$ ($N_h(N_d + 1)$ elements); a total amount of $N_h(N_d + 1) + N_b(N_d + 3N_h)$ elements.

4. Experimental Results and Analysis

The performance of competitive sparse feature learning method has been analyzed on the dataset CIFAR-10. CIFAR-10 dataset provides a fully labeled training set used for both unsupervised and supervised learning, and consists of 32×32 pixel color images belonging to 10 different classes. The dataset has a large amount of labeled data (60K images) to be used for training and testing.

4.1 Classification Accuracy

The experiments follow the pipeline described in **Section 3.3**. In the unsupervised learning phase, we use a receptive field of 6×6 pixels and the sparseness of p = 0.8. The number of outputs is set to N_h = 1600 for fair comparison with the state-of-the-art algorithms. Firstly, the random patches are densely extracted from the training images as the dataset **D** for unsupervised learning phase. Then all the samples are normalized for local brightness and contrast as pre-processing. In the experiments, the number of samples in the resulting dataset **D** is 400K. There are no regularization terms in the object function of the STMG algorithm.

Fig. 3 shows a subset of 100 randomly selected filters learned by the proposed unsupervised feature learning algorithm with the ReLU activation function, 6×6 pixels receptive field and a system of N_h = 1600 outputs. As shown in **Fig. 3**, the method learns not only common filters such as oriented edges/ridges and colors but also corner detectors, tri-banded colored filters. This suggests that enforcing lifetime sparsity helps the system to learn a set of complex, rich and diversified filters.

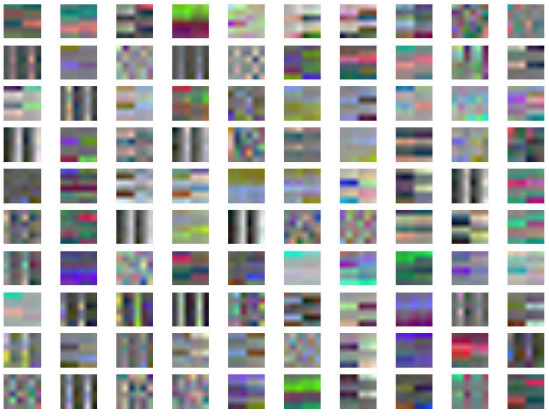


Fig. 3. Random subset of filters learned by the proposed unsupervised feature learning algorithm

In the fine-tuning phase, the well pre-trained feature extracting model in the pre-training phase is applied to retrieve sparse features of patches covering the input image (stride s = 1), and the sparse feature are pooled into four quadrants and finally used to train a softmax classifier. The parameters of the softmax are tuned by five-fold cross validation. In order to prevent overfitting, the dropout is used during training. The results of the algorithm with sign split (N_h = 1600×2, using **W** and –**W** for encoding as in [24]) are also provided. The test dataset consists of 10K color images belonging to 10 different classes and it is also pre-processed like the train dataset.

Table 1. Classification Accuracy on CIFAR-10

Algorithm	**Accuracy**
OMP-*k* (*k*=1) (whitening, 1600×2) [24]	79.4%
OMP-*k* (*k*=1) (1600×2) [30]	73.5%
OMP-*k* (*k*=10) (whitening, 1600×2) [24]	80.1%
OMP-*k* (*k*=10) (whitening, 6000×2) [24]	81.5%
Sparse RBM (1600) [8]	72.4%
Sparse coding (1600×2) [24]	78.8%
Sparse auto-encoder (1600) [8]	73.4%
K-means (whitening, 1600) [8]	77.9%
Sparse filtering (1600) [30]	71.2%
EPLS (6000×2) [30]	81.5%
Shallow CNN (40000) [39]	75.9%
PCANet [40]	78.7%
Exemplar-CNN [41]	82.0%
Zero-bias CNN [42]	86.4%
Ours (1600)	**83.7%**
Ours (1600×2)	**85.1%**

Table 1 presents the results obtained on CIFAR-10 dataset compared to the OMP-k, sparse RBM, sparse coding, sparse auto-encoder, K-means, sparse filtering, EPLS, shallow CNN, PCANet, Exemplar-CNN, and Zero-bias CNN methods. These are all shallow models of image classification except for the CNN based models. As shown in **Table 1**, the proposed algorithm provides competitive results that compare to other algorithms. For OMP-k variants, when the hyperparameter k is properly tuned, it can also achieve good results. However, OMP-k strongly relies on data whitening pre-processing. The performance of OMP-k (k = 1) decreases 5.9 percent when there is no whitening pre-processing. The whitening such as ZCA might be critical, since it can not always be computed exactly for high dimensional data. The EPLS algorithm achieves a good performance when N_h = 6000×2 among the mentioned shallow models. But its sparsity level is too high (for N_h = 1600, the sparsity level is 99.94%). From [31], too high sparsity level will hurt the final classification performance. Sparse coding provides competitive results when properly combining its training and inference parameters. But sparse coding involves the expensive training and inference as well as hyperparameter tuning to achieve the best performance. K-means also achieves good results due to whitening pre-processing and the large amount of labeled data that CIFAR-10 provides. The shallow CNN has three convolutional layers. But the convolutional filters are specified and remain fixed, e.g. taken from OverFeat, only the classifier weights are learnt during training. This operation will hurt the classification accuracy. This suggests that designing enhanced methods of regularization (e.g. methods similar to dropout in the convolutional stage, or data augmentation) are necessary to improve the performance of the shallow CNN. During training of the proposed algorithm, dropout is used to prevent overfitting. PCANet is a simple network for image classification which comprises cascaded PCA components. The classification accuracy of PCANet is not so competitive due to the discriminative ability of PCA. The exemplar-CNN also takes advantage of unsupervised feature learning. The features learnt by the exemplar-CNN yield a large improvement in classification accuracy. One potential shortcoming of the exemplar-CNN is that in its current state it does not scale to arbitrarily large datasets. The Zero-bias CNN method analyzes the benefits of unsupervised pre-training in the context of recent deep learning innovations. This method shows that unsupervised pre-training can provide a large gain in performance. Benefit from the convolutional architecture and the unsupervised pre-training, the classification accuracy obtained by the Zero-bias CNN has increased by 1.3 percent point than the proposed algorithm. But the features learnt by this method have no desired sparsity property.

4.2 Analysis of Population and Lifetime Sparsity

In this section, the achieved population and lifetime sparsity of the proposed method are analyzed and compared to the state-of-the-art methods of sparse coding, sparse auto-encoder, OMP-1, sparse filtering and EPLS. In this experiment, we set the parameters N = 400K color patches of 6×6 pixels of CIFAR-10 dataset, and only N_h = 100 outputs. The majority of the state-of-the-art methods ensure lifetime sparsity by enforcing similar mean activation among outputs. In order to compute the lifetime sparsity, the system takes

the training set and generates the sparse feature representations. After that, the mean activation of each feature is computed, then the normalization operation is applied in order to obtain a probability distribution *P*. Then the Kullback-Leibler divergence (KL) between the resulting probability distribution and an uniform distribution *Q* with equal probability $1/N_h$ is computed as follows,

$$KL(P \| Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)} \tag{3}$$

This is not a strict measure of lifetime sparsity, but a measure of the activation statistics among the outputs. The lower the value, the closer to the objective of maintaining similar mean activation among the outputs.

Likewise, the population kurtosis proposed in [29] is used as the measure of population sparsity. Population kurtosis measures the infrequency of strong neural response. If there are *M* neurons, then the population kurtosis is defined as:

$$K_p = \left\{ \frac{1}{M} \sum_{j=1}^{M} \left(\frac{r_j - \bar{r}}{\sigma_r} \right)^4 \right\} - 3 \tag{4}$$

where $\bar{r}$ and σ_r are the mean and standard deviation of the responses, respectively. The population kurtosis increases as the population sparsity increases.

Table 2. Lifetime (KL) and Population Sparsity of Algorithms

Algorithm	**Lifetime (KL)**	**Population (Kurtosis)**
Sparse Coding [30]	0.0371	45.44
Sparse Auto-encoder [30]	0.0002	18.24
OMP-1 [30]	0.0869	45.55
Sparse Filtering [30]	0.0189	3.48
EPLS [30]	0.0006	27.42
Ours	0.0012	45.75

Table 2 presents the Lifetime (KL) and population sparsity of several algorithms. Sparse coding, OMP-1 and the proposed algorithm have the largest population kurtosis, that is, sparse coding, OMP-1 and the proposed algorithm typically have a good population sparsity property. However, sparse coding and OMP-1 do not achieve good lifetime sparsity. The proposed algorithm obtains the highest population sparsity level and a relatively good lifetime sparsity level due to the introduction of competition mechanism between the output neurons. Sparse filtering claims to ensure both population and lifetime sparsity, but it seems to achieve the lowest level of population sparsity when compared to

other methods. Sparse auto-encoder explicitly enforces lifetime sparsity, thus it achieves the highest lifetime sparsity level. This method also provides reasonable population sparsity. EPLS achieves a good compromise between lifetime and population sparsity by imposing sparsity in a very strict way.

To emphasize the benefits of encouraging lifetime sparsity, the impact of allowing each output to activate for a different number of input is analyzed as the pipeline in [30]. To do so, a coefficient $\lambda \leq 1$ is applied to δ in Line 12 of Algorithm 1 to decrease the inhibition of each output and to allow each output to activate $1/\lambda$ times more than the others, while maintaining the population sparsity property. For each λ, we train a network of N_h = 100 outputs and measure the accuracy of the network as well as the achieved lifetime sparsity (KL).

Table 3. Lifetime (KL) and Classification Accuracy with different λ s

Coefficient λ	Lifetime (KL)	Accuracy
1	0.0012	65.8%
1/2	0.0067	65.7%
1/4	0.0164	65.1%
1/8	0.0238	64.0%
1/16	0.042	61.6%
1/32	0.083	61.2%
1/64	0.1091	60.7%

Table 3 shows the lifetime sparsity (KL) and classification accuracy as the coefficient λ changes. Enforcing Strong lifetime Sparsity (λ=1) achieves the lowest KL and the maximum accuracy. As shown in **Fig. 4**, the accuracy (red) and lifetime sparsity (blue) are the functions of the coefficient λ. The farther we move from the Strong lifetime Sparsity definition (λ decreases), the higher the KL and the lower the accuracy. Therefore, this experiment suggests that enforcing strong lifetime sparsity is indeed crucial to achieve good results.

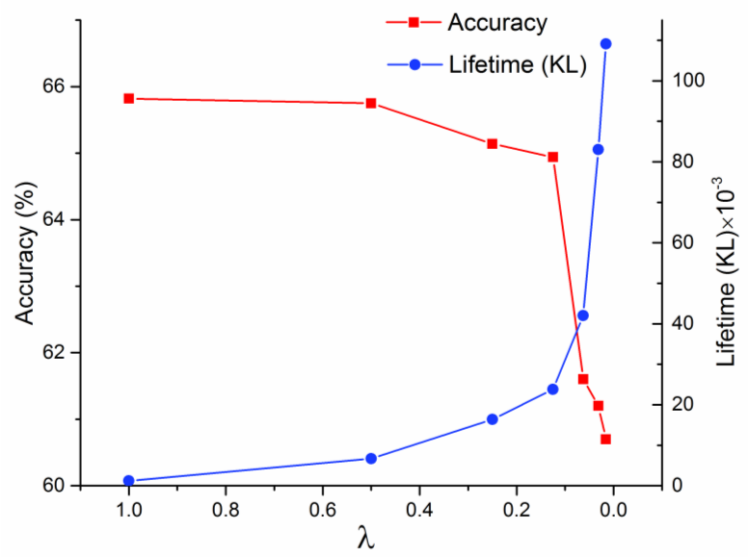


Fig. 4 Classification accuracy and lifetime sparsity in the network of N_h = 100

5. Conclusion

In this paper, we propose a simple and effective unsupervised feature learning algorithm based on sparse target matrix generation method. This algorithm enforces both population and lifetime sparsity in an explicit way in order to learn discriminative features. In the unsupervised feature learning phase, the sparseness is introduced in order to control the sparsity level. After then, the well trained feature extracting model is applied to retrieve sparse features for image classification. Experimental results on the CIFAR-10 dataset suggest that the proposed unsupervised feature learning algorithm is able to learn sparse features that generalize well on new data. Moreover, the proposed unsupervised feature learning algorithm could learn sparse features in terms of population and lifetime sparsity by imposing sparsity in an explicit and strict way. Specifically, strong lifetime sparsity is crucial to achieve good performance.

References

[1] Navneet Dalal, and Bill Triggs, "Histograms of oriented gradients for human detection," in *Proc. of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 1, pp. 886-893, 2005. Article (CrossRef Link).

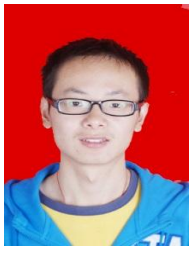
[2] Herbert Bay, Andreas Ess, Tinne Tuytelaars and Luc Van Gool, "Speeded-up robust features (SURF)," *Computer vision and image understanding*, vol. 110, no. 3, pp. 346-359, 2008. Article (CrossRef Link).

[3] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proc. of The proceedings of the seventh IEEE international conference on computer vision*, pp. 1150-1157, 1999. Article (CrossRef Link).

[4] T. Leung, J. Malik, "Representing and recognizing the visual appearance of materials using three-dimensional textons," *International Journal of Computer Vision*, vol. 1, no. 43, pp. 29-44, 2001. Article (CrossRef Link).
[5] S. Lazebnik, C. Schmid and J. Ponce, "Semi-local affine parts for object recognition," in *Proc. of Proceeding of the British Machine Vision Conference*, pp. 779-788, 2004. Article (CrossRef Link).
[6] K. Mikolajczyk, C. Schmid. "A performance evaluation of local descriptors," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 10, no. 27, pp. 1615-1630, 2005. Article (CrossRef Link).
[7] Sargan, Angelov and Habib, "A comprehensive review on handcrafted and learning-based action representation approaches for human activity recognition," *Applied Sciences*, vol. 7, no. 1, pp. 110, 2017. Article (CrossRef Link).
[8] Coates, Adam, A. Y. Ng, and H. Lee, "An Analysis of Single-Layer Networks in Unsupervised Feature Learning," *Proceedings of Machine Learning Research*, vol. 15, pp. 215-223, 2011. Article (CrossRef Link).
[9] Jianchao Yang *et al*, "Linear spatial pyramid matching using sparse coding for image classification," in *Proc. of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 1794-1801, 2009. Article (CrossRef Link).
[10] Geoffrey E. Hinton, Simon Osindero and Yee-Whye Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, pp. 1527-1554, 2006. Article (CrossRef Link).
[11] Yoshua Bengio *et al*, "Greedy layer-wise training of deep networks," *Advances in Neural Information Processing Systems*, vol. 19, pp. 153-160, 2007. Article (CrossRef Link).
[12] Hugo Larochelle *et al*, "Exploring strategies for training deep neural networks," *Journal of Machine Learning Research*, vol. 1, no. 10, pp. 1-40, Jan. 2009. Article (CrossRef Link).
[13] Dumitru Erhan *et al*, "Why does unsupervised pre-training help deep learning?" *Journal of Machine Learning Research*, vol.11, no. 3, pp. 625-660, Feb. 2010. Article (CrossRef Link).
[14] Yoshua Bengio, Aaron Courville and Pascal Vincent, "Representation Learning: A Review and New Perspectives," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1798-1828, 2013. Article (CrossRef Link).
[15] A. Hyvärinen and E. Oja, "Independent component analysis: algorithms and applications," *Neural networks*, vol. 13, no. 4, pp. 411-430, 2000. Article (CrossRef Link).
[16] Aapo Hyvärinen, Juha Karhunen and Erkki Oja, "Independent component analysis," vol. 46. *John Wiley & Sons*, 2004. Article (CrossRef Link).
[17] Jiquan Ngiam *et al*, "Sparse filtering," *Advances in Neural Information Processing Systems*, pp. 1125-1133, 2011. Article (CrossRef Link).
[18] Quoc V Le *et al*, "ICA with reconstruction cost for efficient overcomplete feature learning," *Advances in Neural Information Processing Systems*, pp. 1017-1025, 2011. Article (CrossRef Link).
[19] Honglak Lee *et al*, "Efficient sparse coding algorithms," *Advances in Neural Information Processing Systems*, vol. 19, no. 801, 2007. Article (CrossRef Link).
[20] Koray Kavukcuoglu *et al*, "Learning convolutional feature hierarchies for visual recognition," *Advances In Neural Information Processing Systems*, pp. 1090-1098, 2010. Article (CrossRef Link).
[21] Rajat Raina *et al*, "Self-taught learning: transfer learning from unlabeled data," in *Proc. of Proceedings of the 24th International Conference on Machine learning*, vol. 227, pp. 759-766, 2007. Article (CrossRef Link).

[22] Y. C. Pati, R. Rezaiifar and P. S. Krishnaprasad, "Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition," *Signals, Systems and Computers*, vol. 1, pp. 40-44 1993. Article (CrossRef Link).
[23] Blumensath, Thomas and Mike E. Davies, "On the difference between orthogonal matching pursuit and orthogonal least squares," 2007. Article (CrossRef Link).
[24] Adam Coates and Andrew Y. Ng, "The importance of encoding versus training with sparse coding and vector quantization," in *Proc. of Proceedings of the 28th International Conference on Machine Learning*, pp. 921-928, 2011. Article (CrossRef Link).
[25] Marc'Aurelio Ranzato, Christopher Poultney, Sumit Chopra and Yann LeCun, "Efficient learning of sparse representations with an energy-based model," *Advances in Neural Information Processing Systems*, pp. 1137-1144, 2006. Article (CrossRef Link).
[26] Honglak Lee, Chaitanya Ekanadham and Andrew Y. Ng, "Sparse deep belief net model for visual area V2," *Advances in Neural Information Processing Systems*, pp. 873-880, Dec. 2007. Article (CrossRef Link).
[27] Bruno A. Olshausen and David J. Field, "Sparse coding with an overcomplete basis set: A strategy employed by V1," *Vision research*, vol. 37, no. 23, pp. 3311-3325, 1997. Article (CrossRef Link).
[28] David J. Field, "What is the goal of sensory coding?" *Neural Computation*, vol. 6, no. 4, pp. 559-601, 1994. Article (CrossRef Link).
[29] Benjamin Willmore and David J. Tolhurst, "Characterizing the sparseness of neural codes," *Network: Computation in Neural Systems*, vol. 12, no. 3, pp. 255-270, 2001. Article (CrossRef Link).
[30] Adriana Romero, Petia Radeva and Carlo Gatta, "Meta-parameter free unsupervised sparse feature learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 8, pp. 1716-1722, 2015. Article (CrossRef Link).
[31] Xavier Glorot, Antoine Bordes and Yoshua Bengio, "Deep Sparse Rectifier Neural Networks," *Proceedings of Machine Learning Research*, vol. 15, no. 106, pp. 315-323, 2011. Article (CrossRef Link).
[32] Bruno A. Olshausen and David J. Field. "Emergence of simple-cell receptive field properties by learning a sparse code for natural images," *Nature*, vol.381, no.6583, pp. 607-609, 1996. Article (CrossRef Link).
[33] H. W. Kuhn, "The Hungarian method for the assignment problem," *Naval Research Logistics*, vol. 52, no.1, pp. 7–21, 2010. Article (CrossRef Link).
[34] Marc'Aurelio Ranzato, Christopher Poultney, Sumit Chopra and Yann LeCun, "Efficient learning of sparse representations with an energy-based model," in *Proc. of Proceedings of the 19th International Conference on Neural Information Processing Systems*, pp. 1137-1144, 2006. Article (CrossRef Link).
[35] Li Z, Liu J, Tang J *et al*, "Robust structured subspace learning for data representation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 10, pp. 2085-2098, 2015. Article (CrossRef Link).
[36] Li Z, Liu J, Yang Y *et al*, "Clustering-guided sparse structural learning for unsupervised feature selection," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 9, pp. 2138-2150, 2014. Article (CrossRef Link).
[37] Li Z, Tang J, "Unsupervised feature selection via nonnegative spectral analysis and redundancy control," *IEEE Transactions on Image Processing*, vol. 24, no. 12, pp. 5343-5355, 2015. Article (CrossRef Link).

[38] Li Z, Tang J, He X, "Robust Structured Nonnegative Matrix Factorization for Image Representation," *IEEE Transactions on Neural Networks and Learning Systems*, vol. PP, no. 99, pp. 1-14, 2017. Article (CrossRef Link).
[39] McDonnell, Mark D. and Tony Vladusich, "Enhanced image classification with a fast-learning shallow convolutional neural network," in *Proc. of IEEE International Joint Conference on Neural Networks (IJCNN), pp: 1-7, 2015.* Article (CrossRef Link).
[40] Chan, Tsung-Han *et al*, "PCANet: A simple deep learning baseline for image classification," *IEEE Transactions on Image Processing*, 24(12): 5017-5032, 2015. Article (CrossRef Link).
[41] Dosovitskiy, Alexey *et al*, "Discriminative unsupervised feature learning with exemplar convolutional neural networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 9, pp: 1734-1747, 2016. Article (CrossRef Link).
[42] Paine, Tom Le *et al*, "An analysis of unsupervised pre-training in light of recent advances," *arXiv preprint arXiv:1412.6597, 2014.* Article (CrossRef Link).



Dan Zhao received his B.S. degrees in control technology and instrument from Xidian University, Xi'an, and P.R. China in 2012. He is currently a Ph.D. student with the school of Aerospace Science and Technology, Xidian University. His main research interests revolve around unsupervise and supervised deep learning.

Baolong Guo received his B.S., M.S. and Ph.D. degrees from Xidian University in 1984, 1988 and 1995, respectively, all in communication and electronic system. From 1998 to 1999, he was a visiting scientist at Doshisha University, Japan. He is currently the director of the Institute of Intelligent Control & Image Engineering (ICIE) at Xidian University. His research interests include neural networks, pattern recognition, intelligent information processing, and image communication.

Yunyi Yan received his B.S. degree in automation, M.S. degree in pattern recognizaion andintelligent system, and Ph.D degree in electronic science and technology from Xidian University in 2001, 2004 and 2008 respectively in China. Since 2004, he has been with ICIE Institute of Areospace Science and Technology School of Xidian University, where he is currently an associated professor. Since 2013, he served as the director of Intelligent Detection Department in Xidian University. He worked as a visiting scholar in Montreal Neurological Institute of McGill University, Canada from 2010 to 2011. His research interests include system reliablity analysis, medical image processing, and swarm intelligence.