Stacked Denoise Autoencoder Based Feature Extraction and Classification for Hyperspectral Images

Article *in* Journal of Sensors · January 2016

DOI: 10.1155/2016/3632943

3 authors, including:

Li Ma
Natural Resources Research Institute
95 PUBLICATIONS **3,325** CITATIONS

SEE PROFILE

Xiaoquan Yang
Huazhong University of Science and Technology
80 PUBLICATIONS **581** CITATIONS

SEE PROFILE

Hindawi Publishing Corporation
Journal of Sensors
Volume 2016, Article ID 3632943, 10 pages
http://dx.doi.org/10.1155/2016/3632943

Research Article

Stacked Denoise Autoencoder Based Feature Extraction and Classification for Hyperspectral Images

Chen Xing,[1] Li Ma,[1] and Xiaoquan Yang[2]

[1]*Faculty of Mechanical and Electronic Information, China University of Geosciences, Wuhan, Hubei 430074, China*
[2]*Wuhan National Laboratory for Optoelectronics, Huazhong University of Science and Technology, Wuhan, Hubei 430074, China*

Correspondence should be addressed to Li Ma; maryparisster@gmail.com

Received 25 December 2014; Revised 11 May 2015; Accepted 21 June 2015

Academic Editor: Jonathan C.-W. Chan

Deep learning methods have been successfully applied to learn feature representations for high-dimensional data, where the learned features are able to reveal the nonlinear properties exhibited in the data. In this paper, deep learning method is exploited for feature extraction of hyperspectral data, and the extracted features can provide good discriminability for classification task. Training a deep network for feature extraction and classification includes unsupervised pretraining and supervised fine-tuning. We utilized stacked denoise autoencoder (SDAE) method to pretrain the network, which is robust to noise. In the top layer of the network, logistic regression (LR) approach is utilized to perform supervised fine-tuning and classification. Since sparsity of features might improve the separation capability, we utilized rectified linear unit (ReLU) as activation function in SDAE to extract high level and sparse features. Experimental results using Hyperion, AVIRIS, and ROSIS hyperspectral data demonstrated that the SDAE pretraining in conjunction with the LR fine-tuning and classification (SDAE_LR) can achieve higher accuracies than the popular support vector machine (SVM) classifier.

1. Introduction

Hyperspectral remote sensing images are becoming increasingly available and potentially provide greatly improved discriminant capability for land cover classification. Popular classification methods like k-nearest-neighbor [1], support vector machine [2], and semisupervised classifiers [3] have been successfully applied to hyperspectral images. Besides, some feature matching methods in the computer vision area can also be generalized for spectral classification [4, 5].

Feature extraction is very important for classification of hyperspectral data, and the learned features may increase the separation between spectrally similar classes, resulting in improved classification performance. Commonly used linear feature extraction methods such as principal component analysis (PCA) and linear discriminant analysis (LDA) are simple and easily implemented. However, these methods fail to model the nonlinear structures of data. Manifold learning methods, which are proposed for nonlinear feature extraction, are able to characterize the nonlinear relationships between data points [1, 6, 7]. However, they can only process a limited number of data points due to their high computational complexity. Deep learning methods, which can also learn the nonlinear features, are capable of processing large scale data set. Therefore, we utilized deep learning for feature extraction of hyperspectral data in this paper.

Deep learning is proposed to train a deep neural network for feature extraction and classification. The training process includes two steps: unsupervised layer-wise pretraining and supervised fine-tuning. The layer-wise pretraining [8] can alleviate the difficulty of training a deep network, since the learned network weights which encode the data structure are used as the initial weights of the whole deep network. The supervised fine-tuning that is performed by logistic regression (LR) approach aims to further adjust the network weights by minimizing the classification errors of the labeled data points. Training the network can achieve both high level features and classification simultaneously. Popular deep learning methods include autoencoders (AE) [9], denoised autoencoders (DAE) [10], convolutional neural networks

(CNN) [11], deep belief networks (DBN) [12], and convolutional restricted Boltzmann machines (CRBM) [13]. In the field of hyperspectral data analysis, Chen utilized AE for data classification [14], and Zhang utilized CNN for feature extraction [15].

In this paper, we focus on the stacked DAE (SDAE) method [16], since DAE is very robust to noise, and SDAE can obtain higher level features. Moreover, since sparsity of features might improve the separation capability, we utilized rectified linear unit (ReLU) as activation function in SDAE to extract high level and sparse features. After the layer-wise pretraining by SDAE, LR layer is used for fine-tuning the network and performing classification. The features of the deep network that are obtained by SDAE pretraining and LR fine-tuning are called tuned-SDAE features, and the classification approach that utilizes LR classifier on the tuned-SDAE features is hereafter denoted as SDAE_LR in this paper.

The organization of the paper is as follows. Section 2 describes the DAE, SDAE, and SDAE_LR approaches. Section 3 discussed the experimental results. Conclusions are summarized in Section 4.

2. Methodology

Given a neural network, AE [14] trains the network by constraining the output values to be equal to the input values, which also indicates that the output layer has equally many nodes as the input layer. The reconstruction error between the input and the output of network is used to adjust the weights of each layer. Therefore, the features learned by AE can well represent the input data. Moreover, the training of AE is unsupervised, since it does not require label information. DAE is developed from AE but is more robust, since DAE assumes that the input data contain noise and is suitable to learn features from noisy data. As a result, the generalization ability of DAE is better than that of AE. Moreover, DAE can be stacked to obtain high level features, resulting in SDAE approach. The training of SDAE network is layer-wise, since each DAE with one hidden layer is trained independently. After training the SDAE network, the decoding layers are removed and the encoding layers that produce features are retained. For classification task, a logistic regression (LR) layer is added as output layer. Moreover, LR is also used to fine-tune the network. Therefore, the features are learned by SDAE pretraining in conjunction with LR fine-tuning.

2.1. Denoise Autoencoder (DAE). DAE contains three layers: input layer, hidden layer, and output layer, where the hidden layer and output layer are also called encoding layer and decoding layer, respectively. Suppose the original data is $\mathbf{x} \in R^d$, where d is the dimension of data. DAE firstly produces a vector $\tilde{\mathbf{x}}$ by setting some of the elements to zero or adding the Gaussian noise to $\mathbf{x}$. DAE uses $\tilde{\mathbf{x}}$ as input data. The number of units in the input layer is d, which is equal to the dimension of the input data $\tilde{\mathbf{x}}$. The encoding of DAE is obtained by a nonlinear transformation function:

$$\mathbf{y} = f_e\left(\mathbf{W}\tilde{\mathbf{x}} + \mathbf{b}\right), \tag{1}$$

where $\mathbf{y} \in R^h$ denotes the output of the hidden layer and can also be called feature representation or code, h is the number of units in the hidden layer, $\mathbf{W} \in R^{h \times d}$ is the input-to-hidden weights, $\mathbf{b}$ denotes the bias, $\mathbf{W}\tilde{\mathbf{x}} + \mathbf{b}$ stands for the input of the hidden layer, and $f_e()$ is called activation function of the hidden layer. We chose ReLU function [17] as the activation function in this study, which is formulated as

$$f_e\left(\mathbf{W}\tilde{\mathbf{x}} + \mathbf{b}\right) = \max\left(0, \mathbf{W}\tilde{\mathbf{x}} + \mathbf{b}\right). \tag{2}$$

If the value of $\mathbf{W}\tilde{\mathbf{x}} + \mathbf{b}$ is smaller than zero, the output of the hidden layer will be zero. Therefore, ReLU activation function is able to produce a sparse feature representation, which may have better separation capability. Moreover, ReLU can train the neural network for large scale data faster and more effectively than the other activation functions.

The decoding or reconstruction of DAE is obtained by using a mapping function $f_d()$:

$$\mathbf{z} = f_d\left(\mathbf{W}'\mathbf{y} + \mathbf{b}'\right), \tag{3}$$

where $\mathbf{z} \in R^d$ is the output of DAE, which is also the reconstruction of original data $\mathbf{x}$. The output layer has the same number of nodes as the input layer. $\mathbf{W}' = \mathbf{W}^T$ is referred to as tied weights. If $\mathbf{x}$ is ranged from 0 to 1, we choose softplus function as the decoding function $f_d()$; otherwise we preprocess $\mathbf{x}$ by zero-phase component analysis (ZCA) whitening and use a linear function as the decoding function:

$$f_d\left(\mathbf{a}\right) = \begin{cases} \log\left(1 + e^{\mathbf{a}}\right), & \mathbf{x} \in [0, 1] \\ \mathbf{a}, & \text{otherwise,} \end{cases} \tag{4}$$

where $\mathbf{a} = \mathbf{W}'\mathbf{y} + \mathbf{b}'$. DAE aims to train the network by requiring the output data $\mathbf{z}$ to reconstruct the input data $\mathbf{x}$, which is also called reconstruction-oriented training. Therefore, the reconstruction error should be used as the objective function or cost function, which is defined as follows:

$$\text{Cost} = \begin{cases} -\dfrac{1}{m}\displaystyle\sum_{i=1}^{m}\sum_{j=1}^{d}\left[\mathbf{x}_j^{(i)} \log\left(\mathbf{z}_j^{(i)}\right) + \left(1 - \mathbf{x}_j^{(i)}\right)\log\left(1 - \mathbf{z}_j^{(i)}\right)\right] + \dfrac{\lambda}{2}\left\|\mathbf{W}\right\|^2, & \mathbf{x} \in [0, 1], \\ \dfrac{1}{m}\displaystyle\sum_{i=1}^{m}\left\|\mathbf{x}^{(i)} - \mathbf{z}^{(i)}\right\|^2 + \dfrac{\lambda}{2}\left\|\mathbf{W}\right\|^2, & \text{otherwise,} \end{cases} \tag{5}$$

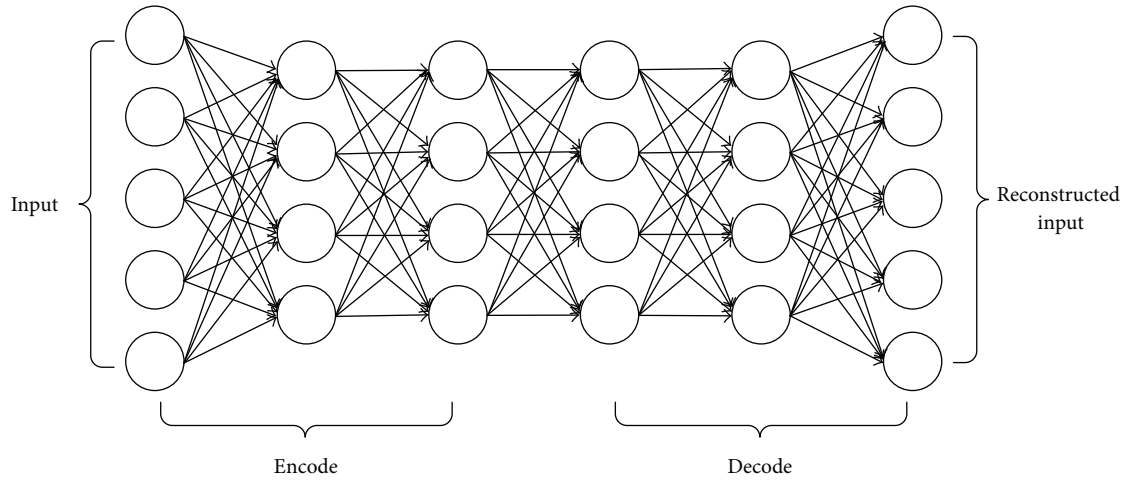


Figure 1: The SDAE network is stacked by two DAE structures.

where cross-entropy function is used when the value of input $\mathbf{x}$ is ranged from 0 to 1; the square error function is used otherwise. $\mathbf{x}_j^{(i)}$ denotes jth element of the ith sample and $\|\mathbf{W}\|^2$ is L2-regularization term, which is also called weight decay term. Parameter λ controls the importance of the regularization term. This optimization problem is solved by using minibatch stochastic gradient descent (MSGD) algorithm [18], and m in (5) denotes the size of the minibatch.

2.2. Stacked Denoise Autoencoder (SDAE). DAE can be stacked to build deep network which has more than one hidden layer [16]. Figure 1 shows a typical instance of SDAE structure, which includes two encoding layers and two decoding layers. In the encoding part, the output of the first encoding layer acted as the input data of the second encoding layer. Supposing there are L hidden layers in the encoding part, we have the activation function of the kth encoding layer:

$$\mathbf{y}^{(k+1)} = f_e\left(\mathbf{W}^{(k+1)}\mathbf{y}^{(k)} + \mathbf{b}^{(k+1)}\right), \quad k = 0, \ldots, L-1, \quad (6)$$

where the input $\mathbf{y}^{(0)}$ is the original data $\mathbf{x}$. The output $\mathbf{y}^{(L)}$ of the last encoding layer is the high level features extracted by the SDAE network. In the decoding part, the output of the first decoding layer is regarded as the input of the second decoding layer. The decoding function of the kth decode layer is

$$\mathbf{z}^{(k+1)} = f_d\left(\mathbf{W}^{(L-k)T}\mathbf{z}^{(k)} + \mathbf{b}'^{(k+1)}\right), \quad k = 0, \ldots, L-1, \quad (7)$$

where the input $\mathbf{z}^{(0)}$ of the first decoding layer is the output $\mathbf{y}^{(L)}$ of the last encoding layer. The output $\mathbf{z}^{(L)}$ of the last decoding layer is the reconstruction of the original data $\mathbf{x}$.

The training process of SDAE is provided as follows.

Step 1. Choose input data, which can be randomly selected from the hyperspectral images.

Step 2. Train the first DAE, which includes the first encoding layer and the last decoding layer. Obtain the network weights $\mathbf{W}^{(1)}$ and $\mathbf{b}^{(1)}$ and the features $\mathbf{y}^{(1)}$ which are the output of the first encoding layer.

Step 3. Use $\mathbf{y}^{(k)}$ as the input data of the $(k+1)$th encoding layer. Train the $(k+1)$th DAE and obtain $\mathbf{W}^{(k+1)}$ and $\mathbf{b}^{(k+1)}$ and the features $\mathbf{y}^{(k+1)}$, where $k = 1, \ldots, L-1$ and L is the number of hidden layers in the network.

It can be seen that each DAE is trained independently, and therefore the training of SDAE is called layer-wise training. Moreover, the trained network weights by SDAE acted as the initial weights in the following LR fine-tuning phase. Therefore, SDAE pretrains the network.

2.3. SDAE_LR. SDAE_LR includes SDAE pretraining and LR fine-tuning. SDAE trains the network weights and obtains features by the reconstruction-oriented learning, and the learned weights acted as the initial weights of the network. Further, LR is used to fine-tune the network weights and obtain the fine-tuned features. It is worth noting that SDAE is unsupervised, while LR is supervised and only the data with labeled information can be used in LR stage. The SDAE_LR network is illustrated in Figure 2, which shows a two-category classification problem (there are two output values). We can see that the decoding part of SDAE is removed and the encoding part of SDAE is retained to produce the initial features. In addition, the output layer of the whole network, which is also called LR layer, is added. The following sigmoid function is used as activation function of LR layer:

$$h(\mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{W}\mathbf{x} - \mathbf{b})}, \quad (8)$$

where $\mathbf{x}$ is the output $\mathbf{y}^{(L)}$ of the last encoding layer. It is also the deep features that are pretrained by SDAE method. The

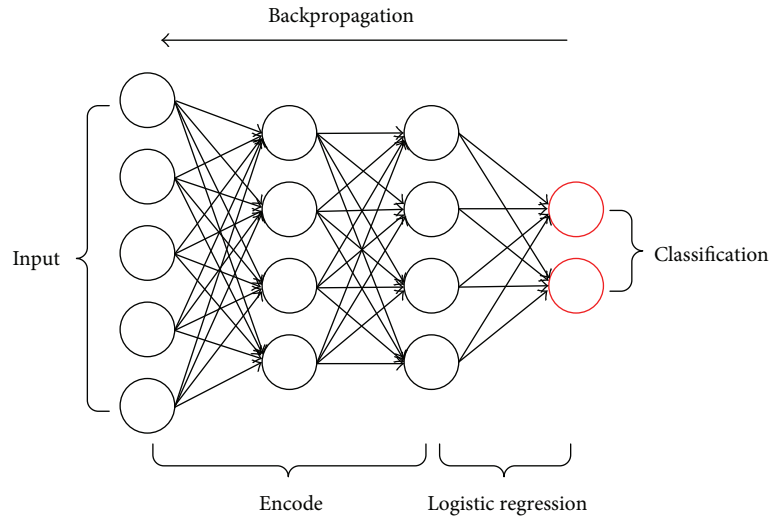


Figure 2: SDAE_LR structure includes the encoding part of SDAE for feature extraction and LR for fine-tuning and classification.

output of sigmoid function is between 0 and 1, which denotes the classification results.

Labels are associated with the training data points, and therefore we can use the errors between the predicted classification results and the true labels to fine-tune the whole network weights. The cost function is defined as the following cross-entropy function:

$$\text{Cost} = -\frac{1}{m}\left[\sum_{i=1}^{m} l^{(i)} \log\left(h\left(\mathbf{x}^{(i)}\right)\right) + \left(1 - l^{(i)}\right) \log\left(1 - h\left(\mathbf{x}^{(i)}\right)\right)\right], \tag{9}$$

where $l^{(i)}$ denotes the label of the sample $\mathbf{x}^{(i)}$. Minimizing the cost function, we can update the network weights. This optimization problem is also solved by MSGD method.

The steps of SDAE_LR network training are as follows.

Step 1. SDAE is utilized to train the initial network weights, described in Section 2.2.

Step 2. Initial weights of the LR layer are randomly set.

Step 3. Training data are used as input data, and their predicted classification results are produced with the initial weights of the whole network.

Step 4. Network weights are iteratively tuned by minimizing the cost function in (9) using MSGD optimization method.

After the network training, we can calculate the features of any input data, which are the output of the last encoding layer. We call the features learned by SDAE pretraining and LR fine-tuning tuned-SDAE feature. It is worth noting that LR classifier is a part of the network. The output of LR layer, which is also the output of the whole network, denotes the classification results. Therefore, SDAE_LR obtains feature extraction and classification simultaneously. In addition, besides LR, other supervised classifiers like support vector machine (SVM) can also be combined with the tuned-SDAE features.

3. Experimental Results and Analysis

3.1. Data Description. Three hyperspectral images were used for experiments. One was collected over Indian Pine (INP) in 1992. The spatial resolution of this image is 20 m; the available band for analysis of the image is 200 after removal of noisy and water absorption bands. One was acquired by Hyperion instrument over the Okavango Delta, Botswana (BOT), in May 2001. The 224-band Hyperion data have 10 nm spectral resolution over the range of 400 nm–2500 nm. The last high spatial resolution hyperspectral image was collected by reflective optics system imaging spectrometer (ROSIS) over the University of Pavia (PU), Italy. This data set has 103 dimensions of a spectral range from 430 nm to 860 nm, and its spatial resolution is 1.3 m. Both BOT and PU data contain 9 land cover types, and INP has 13 land cover types. Figure 3 shows the RGB images and the ground referenced information with class legends of BOT, PU, and INP images. Table 1 lists the class names and number of the three data sets.

3.2. Network Configuration. We firstly normalized the data in the range between 0 and 1 and then randomly selected 20 thousand data points from BOT, PU, and INP images, which were used for unsupervised pretraining of SDAE. In supervised LR training stage, we randomly divided the labeled data into training data, validation data, and testing data, with a ratio of 5 : 2 : 3. The training data are used in LR for fine-tuning, the validation data are for parameter tuning and termination of the iteration in MSGD method, and the testing data are for evaluating the algorithm.

Network configuration contains three parameters, which are the number of hidden layers, the number of units

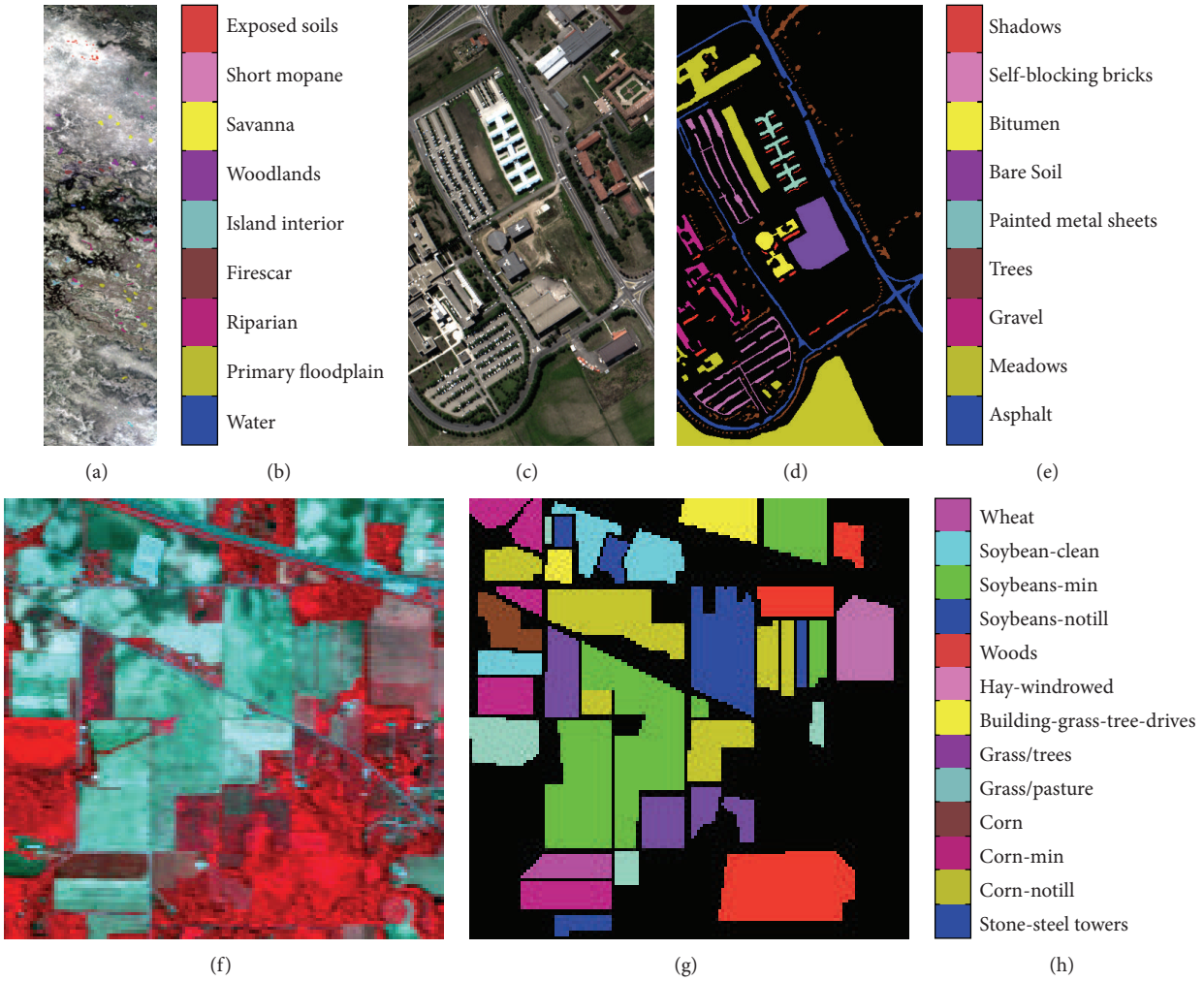
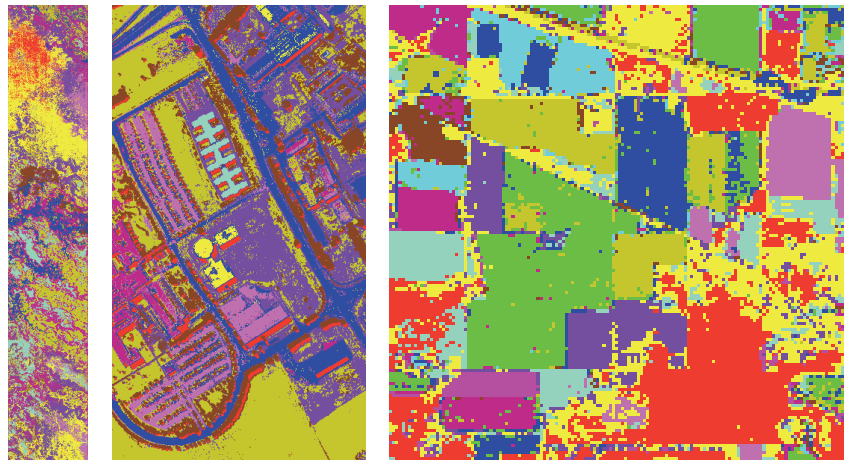


Figure 3: Three band false color composite and ground references. (a) False color composite of BOT image with ground reference. (b) Class legend of BOT image. (c) False color composite of PU image. (d) Ground reference of PU image. (e) Class legend of PU image. (f) IND PINE scene. (g) Ground reference of IND PINE image. (h) Class legend of IND PINE image.

Table 1: Class information of three datasets and the number of labeled samples in each class.

BOT		INP		PU	
ID	Class Name	ID	Class Name	ID	Class Name
1	Water (158)	1	Stone-steel Towers (95)	1	Asphalt (6631)
2	Floodplain (228)	2	Corn-notill (1434)	2	Meadows (18649)
3	Riparian (237)	3	Corn-min (834)	3	Gravel (2099)
4	Firescar (178)	4	Corn (234)	4	Trees (3064)
5	Island Interior (183)	5	Grass/Pasture (497)	5	Painted metal Sheets (1435)
6	Woodlands (199)	6	Grass/Trees (747)	6	Bare Soil (5029)
7	Savanna (162)	7	Building-Grass-Tree-Drives (380)	7	Bitumen (1330)
8	Mopane (124)	8	Hay-windrowed (489)	8	Self-Blocking Bricks (3682)
9	Exposed Soils (111)	9	Woods (1294)	9	Shadows (947)
		10	Soybeans-notill (968)		
		11	Soybeans-min (2468)		
		12	Soybean-clean (614)		
		13	Wheat (212)		



(a) (b) (c)

Figure 4: Classification results of the whole image on BOT (a), PU (b), and INP (c) data set.

in hidden layer, and the standard deviation of Gaussian noise. The number of hidden layers is selected in the range from 1 to 5, the number of units is chosen from [10, 30, 60, 100, 200, 300, 400], and the standard deviation of Gaussian noise is selected from [0.2, 0.4, 0.6, 0.8]. The optimal selection of these parameters is obtained according to the optimal classification results on the validation data. For BOT, PU, and INP data, the optimal number of layers is 4, 3, and 3, respectively; the best options for the number of units are 100, 300, and 200, respectively; the optimal selections of the standard deviation of Gaussian noise are 0.6, 0.6, and 0.2, respectively. In addition, network training includes two parameters: the epochs of pretraining and fine-tuning are set to be 200 and 1500, and the learning rates of pretraining and fine-tuning are selected as 0.01 and 0.1 empirically.

We used Theano for conducting the SDAE_LR classification. Theano is a Python library that can define, optimize, and evaluate mathematical expressions involving multidimensional arrays efficiently and can use GPU to speed up the calculation.

3.3. SDAE_LR Classification Performance. SDAE_LR method is compared with SVM classifier in this section, where SVM classifiers with linear and RBF kernels on the original data were conducted, which are denoted as LSVM and RSVM, respectively. The parameters in RSVM classifier are tuned by cross-validation method, and the penalty parameter in LSVM is set to be 2. The comparison results using overall accuracies (OA%) are shown in Table 2. It can be seen that the SDAE_LR outperformed LSVM for all the three data sets and obtained higher accuracies than RSVM on PU and INP data. It demonstrates that the features learned by the SDAE pretraining and LR fine-tuning can effectively increase the separation between classes. Figure 4 shows the classification results of the whole images using SDAE_LR for the three images. The acceptable results demonstrate good generalization ability of the SDAE_LR approach.

Table 2: Comparison of SDAE_LR and SVM classifier (OA%).

Data	LSVM	RSVM	SDAE_LR
BOT	92.88	**96.88**	95.53
PU	80.11	93.62	**95.97**
INP	76.15	90.63	**92.06**

Table 3: Comparison of computational time of SDAE_LR and SVM classifier (seconds).

Data	LSVM	RSVM	SDAE_LR
BOT	0.2632	142.1	94.68
PU	3.782	>12 h	1495
INP	2.727	5814	387.7

Using a machine with Intel Xeon CPU I7-4770, GPU NVIDIA Q4000, and 8 G RAM, the computational time of the three classifiers on BOT, PU, and INP data is shown in Table 3, where the LSVM and RSVM are implemented using CPU and SDAE_LR utilized GPU for computation. LSVM costs least time and RSVM is the most time-consuming because of the parameter tuning. We did not provide the exact time for RSVM on PU data since it is longer than 12 hours. The proposed SDAE_LR is much faster than RSVM, since it is implemented using Theano which accelerates the computation significantly. It is worth noting that the SDAE pretraining is fast and the LR fine-tuning costs time, because the former is layer-wise training and the latter propagates errors through the whole network.

3.4. Comparison of Different Feature Extraction Methods. Features of SDAE_LR network are obtained by SDAE pretraining and LR fine-tuning, which is called tuned-SDAE features. We compare the proposed method with four popular feature extraction methods, including PCA, Laplacian Eigenmaps (LE), locally linear embedding (LLE), and LDA.

Table 4: OA% using LSVM on different features.

Data	Raw	PCA	LE	LLE	LDA	Tuned-SDAE
BOT	92.88	88.62	93.70	94.03	89.28	**97.02**
PU	80.12	77.9	82.46	83.34	78.44	**96.59**
INP	76.15	66.56	75.06	71.3	74.32	**91.92**

Table 5: OA% using RSVM on different features.

Data	Raw	PCA	LE	LLE	LDA	Tuned-SDAE
BOT	**96.88**	90.83	92.27	94.14	96.57	95.69
PU	93.62	96.03	85.37	84.65	93.04	**96.52**
INP	90.63	90.52	78.44	73.86	86.1	**92.47**

The first three methods are unsupervised methods and LDA is supervised. In addition, PCA and LDA are linear methods, while LE and LLE are nonlinear methods. We set the number of features to be 50 for PCA, LE, and LLE empirically. The tuned-SDAE features are obtained by using the same network configuration described in Section 3.2.

After feature extraction by PCA, LE, LLE, LDA, and SDAE_LR, we used SVM classifiers (LSVM and RSVM) for classification. In addition, we also conducted SVMs on the raw hyperspectral data. Tables 4 and 5 show the overall accuracies of these methods using LSVM and RSVM, respectively. Several observations can be obtained: (1) for different feature extraction methods, tuned-SDAE performed the best. It significantly outperformed the others with the LSVM classifier for all the three data sets. When the RSVM classification was employed, the tuned-SDAE features also obtained the highest accuracies on most of the data sets; (2) compared to the SVM classification on the raw hyperspectral data, the four feature extraction methods (PCA, LE, LLE, and LDA) may not improve the accuracies, while the proposed tuned-SDAE features can consistently obtain better performance on most data sets; (3) in the four feature extraction methods (PCA, LE, LLE, and LDA), we cannot find one method that is consistently better than the others. The features obtained by SDAE_LR produced stable and good performances on all the data sets; (4) RSVM performed better than LSVM on the raw data and the features extracted by PCA, LE, LLE, and LDA, while RSVM and LSVM provided similar results on the tuned-SDAE features.

From the last column of Tables 2, 4, and 5, we can also observe that, with the tuned-SDAE features, different classifiers (LR, LSVM, and RSVM) resulted in similar performances. Within the three classifiers, LR is simplest since it is a part of the network, and the output of the network is the LR classification results.

Computational times of different feature extraction methods on the three data sets are listed in Table 6. Since the computational complexity of LE and LLE is $O(dN^2)$, where d is the number of dimension and N is the number of points, LE and LLE cannot process the large scale data sets. For PU data, we randomly selected 5000 data points for LE and LLE, and the features of the reminding data points are calculated by a kernel-based generalization method [1]. We can see that PCA and LDA are very fast. For BOT data, LE and LLE cost little time, while for INP and PU data, LE is very time-consuming and LLE also costs time, since the numbers of processed data points of INP and PU are much larger than BOT data. Feature extraction of SDAE_LR also requires times, especially for PU data where 20 thousand data points are used in LR fine-tuning stage.

Table 6: Comparison of computational time of different feature extraction methods (seconds).

Data	PCA	LE	LLE	LDA	Tuned-SDAE
BOT	1.775	5.206	5.596	0.2864	94.68
PU	2.45	1022	70.14	0.3953	1495
INP	0.1918	9362	564.7	0.2788	387.7

3.5. Analysis of SDAE_LR Network Configuration. Firstly, we provided sensitivity analysis of three parameters in network configuration (number of hidden layers, number of units in each hidden layer, and standard deviation of Gaussian noise). Secondly, we demonstrated the effect of ReLU activation function. Thirdly, we tested the classification performances relative to different rates of training data.

Figure 5 shows the results of parameter analysis. When one parameter was tested, the values of other parameters were set to be values described in Section 3.2. (1) For the layers of the deep network, we tested five different values (1, 2, 3, 4, and 5), and the classification results are shown in Figure 5(a). For INP and PU data, the best number of layer is 3; for BOT data, the optimal selection is 4. Results on BOT and PU data are not sensitive to these parameters when the number of layer is larger than 2, while results on INP data indicate that only values of 2, 3, and 4 produced satisfactory performance. (2) For the number of units in each hidden layer, we evaluated seven different values (10, 30, 60, 100, 200, 300, and 400). As is shown in Figure 5(b), the best numbers of unit are 100, 300, and 200 for BOT, PU, and INP data, respectively. For INP data, small values like 10 deteriorate the classification performance. However, SDAE_LR is not very sensitive to this parameter in a large range (number of units > 100). (3) For the standard deviation of Gaussian noise, we tested four different values (0.2, 0.4, 0.6, and 0.8). The classification results with respect to this parameter is shown in Figure 5(c). The optimal values are 0.6, 0.6, and 0.2 for BOT, PU, and INP data, respectively. It can be been that SDAE_LR is not very sensitive to this parameter.

Selection of activation function of the network is very important, and we chose ReLU function as activation function in this paper, since it is able to produce sparse features. To demonstrate the effectiveness of the sparsity, we compared two activation functions: ReLU function and sigmoid function, where the latter cannot obtain sparse features. The extracted features of SDAE_LR are the outputs of the last hidden layer, and therefore the dimensionality of features is equal to the number of units in the hidden layer. We define sparsity rate as the ratio of the number of zeros in the feature to the dimensionality of the feature. A high sparsity rate means there are many zeros in the feature and the feature is

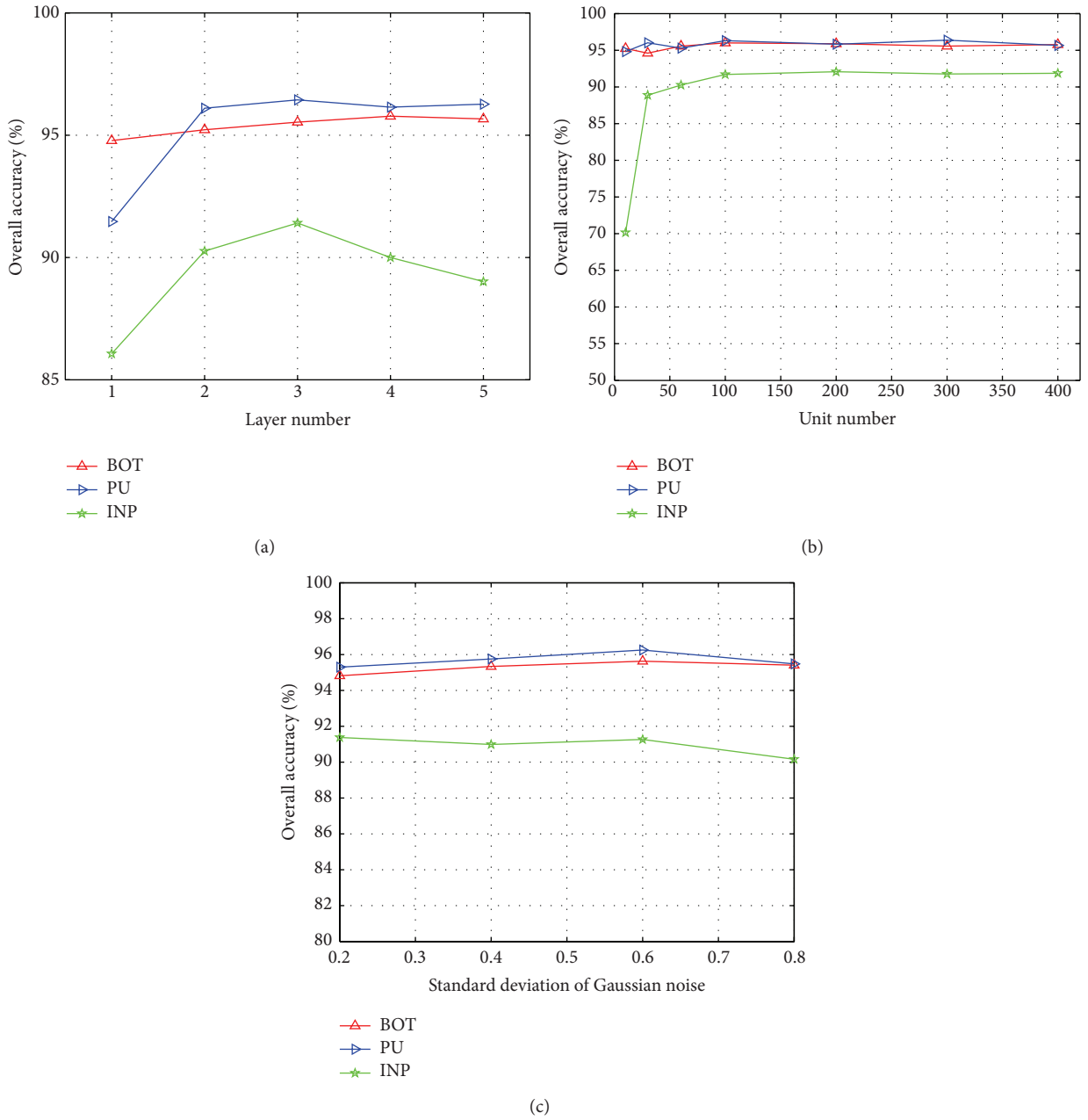


Figure 5: Parameter analysis of SDAE_LR approach. (a) For the parameter of the number of hidden layers. (b) For the parameter of the number of units in hidden layer. (c) For the parameter of standard deviation of Gaussian noise.

highly sparse. Figure 6 plots the sparsity rates versus different unit numbers of hidden layer using the ReLU activation function. With different number of units, the sparsity rate is high, and the number of nonzero values in the feature is small. Take PU data for example; when the number of unit is 400, the sparsity rate is 0.9626. It means the number of zeros in the feature is 385, and the feature only contains 15 nonzero values. Table 7 shows the OA using SDAE_LR with ReLU function and sigmoid function. It can be seen that ReLU function outperformed sigmoid function on all the three data sets, which demonstrates the efficiency of the sparse features using ReLU function.

Table 7: OA% with different activation functions.

Data	ReLU	Sigmoid
BOT	95.53	93.44
PU	95.97	76.12
INP	92.06	88.95

The number of training data also affects the network training, since LR fine-tuning is supervised and only training data can be used to further adjust the network weights. Figure 7 shows the SDAE_LR performance with respect to

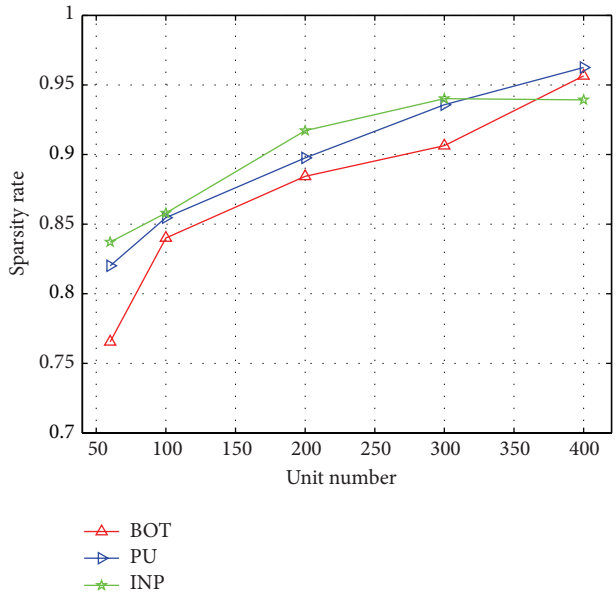


Figure 6: Sparsity rate of the network with different unit number of hidden layer.

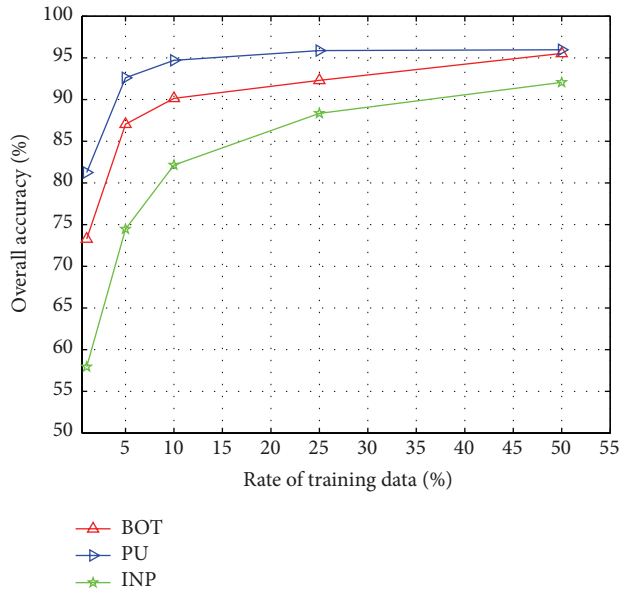


Figure 7: SDAE_LR classification performance with respect to the rates of training data.

different rates of training data (1%, 5%, 10%, 25%, and 50%). In general, high training data rates resulted in high accuracies, since LR performs supervised fine-tuning and classification.

4. Conclusion

Deep learning by SDAE_LR is proposed for hyperspectral feature extraction and classification, where SDAE pretrains the network in an unsupervised manner, and LR fine-tunes the whole network and also performs classification. The features are learned by SDAE pretraining and LR fine-tuning. In the network, ReLU activation function was exploited to achieve the sparse features, which may improve the separation capability of the features. In experiments, SDAE_LR outperformed the popular SVM classifier with linear and RBF kernels. The tuned-SDAE features also provide better classification accuracies than several popular feature extraction methods, which demonstrates the good discriminant ability of extracted features. In SDAE, the utilized ReLU function performed better than sigmoid function, indicating the effect of the sparsity of features.

In SDAE_LR method, we only utilized spectral features of data. Plenty of spatial information of hyperspectral images can also be extracted and exploited [2, 19], such as the texture feature, morphological feature, the spatial coordinate information, and the relations between spatial adjacent pixels. Our further work is to combine spatial information in the SDAE_LR framework to further improve the classification performance.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work was supported by National Natural Science Foundations of China (61102104, 81201067), the Fundamental Research Funds for National University, China University of Geosciences (Wuhan) (CUG120408, CUG120119), and Institute of Automation, Chinese Academy of Sciences.

References

[1] L. Ma, M. M. Crawford, and J. Tian, "Local manifold learning-based k-nearest-neighbor for hyperspectral image classification," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 48, no. 11, pp. 4099–4109, 2010.

[2] X. Huang and L. Zhang, "An SVM ensemble approach combining spectral, structural, and semantic features for the classification of high-resolution remotely sensed imagery," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 51, no. 1, pp. 257–272, 2013.

[3] L. Ma, M. M. Crawford, X. Yang, and Y. Guo, "Local-manifold-learning-based graph construction for semisupervised hyperspectral image classification," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 53, no. 5, pp. 2832–2844, 2015.

[4] J. Ma, W. Qiu, J. Zhao, Y. Ma, A. L. Yuille, and Z. Tu, "Robust L_2E estimation of transformation for non-rigid registration," *IEEE Transactions on Signal Processing*, vol. 63, no. 5, pp. 1115–1129, 2015.

[5] J. Y. Ma, J. Zhao, Y. Ma, and J. W. Tian, "Non-rigid visible and infrared face registration via regularized gaussian fields criterion," *Pattern Recognition*, vol. 48, no. 3, pp. 772–784, 2015.

[6] L. J. P. van der Maaten, E. O. Postma, and H. J. van den Herik, "Dimensionality reduction: a comparative review," 2008, http://www.iai.uni-bonn.de/~jz/dimensionality_reduction_a_comparative_review.pdf.

[7] L. Zhang, Q. Zhang, L. Zhang, D. Tao, X. Huang, and B. Du, "Ensemble manifold regularized sparse low-rank approximation for multiview feature embedding," *Pattern Recognition*, vol. 48, no. 10, pp. 3102–3112, 2014.

[8] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle, U. D. Montral, and M. Qubec, "Greedy layer-wise training of deep networks," in *Advances in Neural Information Processing Systems*, vol. 19, pp. 153–160, 2007.

[9] C. C. Tan and C. Eswaran, "Reconstruction of handwritten digit images using autoencoder neural networks," in *Proceedings of the IEEE Canadian Conference on Electrical and Computer Engineering*, pp. 465–469, May 2008.

[10] P. Vincent, H. Larochelle, Y. Bengio, and P. A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th International Conference on Machine Learning*, pp. 1096–1103, 2008.

[11] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," in *Neural Information Processing Systems*, vol. 25, pp. 1097–1105, 2012.

[12] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[13] M. Norouzi, M. Ranjbar, and G. Mori, "Stacks of convolutional restricted boltzmann machines for shift-invariant feature learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2735–2742, June 2009.

[14] Y. S. Chen, Z. H. Lin, X. Zhao, G. Wang, and Y. F. Gu, "Deep learning-based classification of hyperspectral data," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 7, no. 6, pp. 2094–2107, 2014.

[15] F. Zhang, B. Du, and L. Zhang, "Saliency-guided unsupervised feature learning for scene classification," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 53, no. 4, pp. 2175–2184, 2015.

[16] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: learning useful representations in a deep network with a local denoising criterion," *Journal of Machine Learning Research*, vol. 11, pp. 3371–3408, 2010.

[17] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, vol. 15, pp. 315–323, 2011.

[18] L. Bottou, "Online algorithms and stochastic approximations," in *Online Learning and Neural Networks*, D. Saad, Ed., pp. 9–42, Cambridge University Press, Cambridge, UK, 1998.

[19] L. Zhang, L. Zhang, D. Tao, and X. Huang, "On combining multiple features for hyperspectral remote sensing image classification," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 50, pp. 879–893, 2012.