Efficient Nonlinear Dimensionality Reduction for Pixel-wise Classification of Hyperspectral Imagery

Xuewen Zhang
xz9052@rit.edu

Follow this and additional works at: https://scholarworks.rit.edu/theses

Recommended Citation

This Dissertation is brought to you for free and open access by RIT Scholar Works. It has been accepted for inclusion in Theses by an authorized administrator of RIT Scholar Works. For more information, please contact ritscholarworks@rit.edu.
Efficient Nonlinear Dimensionality Reduction
for Pixel-wise Classification of Hyperspectral Imagery

by

Xuewen Zhang

B.S. Harbin Institute of Technology, 2010
M.S. Harbin Institute of Technology, 2012

A dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy
in the Chester F. Carlson Center for Imaging Science

College of Science
Rochester Institute of Technology

February 15, 2019

Signature of the Author

Accepted by

Coordinator, Ph.D. Degree Program

Date
The Ph.D. Degree Dissertation of Xuewen Zhang
has been examined and approved by the
dissertation committee as satisfactory for the
dissertation required for the
Ph.D. degree in Imaging Science

Dr. Nathan Cahill, Dissertation Advisor

Dr. David Messinger

Dr. John Kerekes

Dr. Kara Maki
Title of Dissertation:
Efficient Nonlinear Dimensionality Reduction
for Pixel-wise Classification of Hyperspectral Imagery

I, Xuewen Zhang, hereby grant permission to Wallace Memorial Library of R.I.T. to reproduce my thesis in whole or in part. Any reproduction will not be for commercial use or profit.

Signature

Date
Efficient Nonlinear Dimensionality Reduction
for Pixel-wise Classification of Hyperspectral Imagery

by

Xuewen Zhang

Submitted to the
Chester F. Carlson Center for Imaging Science
in partial fulfillment of the requirements
for the Doctor of Philosophy Degree
at the Rochester Institute of Technology

Abstract

Classification, target detection, and compression are all important tasks in analyzing hyperspectral imagery (HSI). Because of the high dimensionality of HSI, it is often useful to identify low-dimensional representations of HSI data that can be used to make analysis tasks tractable. Traditional linear dimensionality reduction (DR) methods are not adequate due to the nonlinear distribution of HSI data. Many nonlinear DR methods, which are successful in the general data processing domain, such as Local Linear Embedding (LLE) [1], Isometric Feature Mapping (ISOMAP) [2] and Kernel Principal Components Analysis (KPCA) [3], run very slowly and require large amounts of memory when applied to HSI. For example, applying KPCA to the 512 × 217 pixel, 204-band Salinas image using a modern desktop computer (AMD FX-6300 Six-Core Processor, 32 GB memory) requires more than 5 days of computing time and 28GB memory!

In this thesis, we propose two different algorithms for significantly improving the computational efficiency of nonlinear DR without adversely affecting the performance of classification task: Simple Linear Iterative Clustering (SLIC) superpixels and semi-supervised deep autoencoder networks (SSDAN). SLIC is a very popular algorithm developed for computing superpixels in RGB images that can easily be extended to HSI. Each superpixel includes hundreds or
thousands of pixels based on spatial and spectral similarities and is represented by the mean spectrum and spatial position of all of its component pixels. Since the number of superpixels is much smaller than the number of pixels in the image, they can be used as input for nonlinear DR, which significantly reduces the required computation time and memory versus providing all of the original pixels as input. After nonlinear DR is performed using superpixels as input, an interpolation step can be used to obtain the embedding of each original image pixel in the low dimensional space. To illustrate the power of using superpixels in an HSI classification pipeline, we conduct experiments on three widely used and publicly available hyperspectral images: Indian Pines, Salinas and Pavia. The experimental results for all three images demonstrate that for moderately sized superpixels, the overall accuracy of classification using superpixel-based nonlinear DR matches and sometimes exceeds the overall accuracy of classification using pixel-based nonlinear DR, with a computational speed that is two-three orders of magnitude faster.

Even though superpixel-based nonlinear DR shows promise for HSI classification, it does have disadvantages. First, it is costly to perform out-of-sample extensions. Second, it does not generalize to handle other types of data that might not have spatial information. Third, the original input pixels cannot approximately be recovered, as is possible in many DR algorithms. In order to overcome these difficulties, a new autoencoder network - SSDAN is proposed. It is a fully-connected semi-supervised autoencoder network that performs nonlinear DR in a manner that enables class information to be integrated. Features learned from SSDAN will be similar to those computed via traditional nonlinear DR, and features from the same class will be close to each other. Once the network is trained well with training data, test data can be easily mapped to the low dimensional embedding. Any kind of data can be used to train a SSDAN, and the decoder portion of the SSDAN can easily recover the initial input with reasonable loss. Experimental results on pixel-based classification in the Indian Pines, Salinas and Pavia images show that SSDANs can approximate the overall accuracy of nonlinear DR while significantly improving computational efficiency. We also show that transfer learning can be use to finetune
features of a trained SSDAN for a new HSI dataset. Finally, experimental results on HSI compression show a trade-off between Overall Accuracy (OA) of extracted features and Peak Signal to Noise Ratio (PSNR) of the reconstructed image.

**Keywords:** hyperspectral image, nonlinear dimensionality reduction, autoencoder, super-pixel
Acknowledgements

First of all, I want to thank RIT and the Chester F. Carlson Center for Imaging Science for offering me admission to the Imaging Science Ph.D Program, and for providing teaching assistantships and research assistantships. Without them, I would not have been able to carry out my Ph.D in the US.

Next, I want to offer my sincerest gratitude to my advisor, Nathan D. Cahill. During my five years as a Ph.D. student, you always supported me and my dreams. You gave me a lot of advice for my life, my courses, and my research, and you also show me the culture of America. Your passion in math, coding and research impresses me a lot. Whenever I have troubles in my research, you are so patient and professional and give me strength to help me overcome those troubles. I can’t forget that you have helped me revise many abstracts, proposals, and papers. You are my good friend, advisor, and mentor in my whole life.

I would like to thank all of my colleagues in the research team: Tyler Hayes, Renee Meinhold, Selene Chew and Eman Johnson. Thanks to all of you for offering me advice and smiles during our conversations. And I’m also very overwhelmed with longing for those days of attending the SPIE conference together with you.

I would also like to thank all of my colleagues in the office: Zhenlin Xu, Chi Zhang, Geifei Yang and Kamal Jnawali. Thanks to all of you for accompanying me for so many days in the office. When I have a question, you help me answer it. When I have good news, I want to share it with you. You are very good people.

My gratitude also goes to all of my friends in Rochester: Zhaoyu Cui, Fan Wang, Runchen Zhao, Wei Yao, Can Jin, Chao Zhang, Fei Zhang and Yawen Lu. Thanks for your advice on my Ph.D and on the following career. I hope we can be still good friends in my following life.

Finally, I want to thank my family: my wife Runzi Wang, my daughter Emily Zhang, my mother Haiyan Chen and my father Hengsheng Zhang, for your support and understanding. I’m happiest when I see you live happily. Thanks for all the moments we shared together.
Contents

1 Introduction 18

2 Background 20

  2.1 Hyperspectral imaging (HSI) 20
  2.2 HSI Analysis Tasks 24
    2.2.1 Pixel-based Classification 24
    2.2.2 Target Detection 25
    2.2.3 Compression 27
  2.3 Dimensionality Reduction for HSI 28
    2.3.1 Linear Dimensionality Reduction 29
    2.3.2 Nonlinear Dimensionality Reduction 30
  2.4 Example Nonlinear DR Methods 36
    2.4.1 Spatial Spectral Schroedinger Eigenmaps (SSSE) 36
    2.4.2 Kernel Principle Component Analysis (KPCA) 39
  2.5 Summary 42

3 Superpixel-based Dimensionality Reduction of HSI 43

  3.1 Review of Superpixel Construction Techniques 44
  3.2 Simple Linear Iterative Clustering (SLIC) 45
  3.3 Applications of Superpixels in HSI 47
3.4 Proposed HSI Classification Framework ........................................... 52
3.5 Experiments .................................................................................. 53
  3.5.1 Data sets ................................................................................. 53
  3.5.2 Experimental setup ................................................................. 54
  3.5.3 Results .................................................................................. 56
3.6 Conclusion ..................................................................................... 74

4 Semi-Supervised Deep Autoencoder Networks .................................. 75
  4.1 Autoencoders .............................................................................. 76
    4.1.1 Background .......................................................................... 77
    4.1.2 Applications ....................................................................... 83
  4.2 Proposed Autoencoder Network: SSDAN ..................................... 87
  4.3 Pixel-wise HSI Classification Experiments ................................. 90
    4.3.1 Experimental Setup .............................................................. 90
    4.3.2 Comparison with Other Methods ......................................... 91
    4.3.3 Data augmentation ............................................................. 93
    4.3.4 Effectiveness of the loss function ........................................ 95
    4.3.5 Different activations ............................................................ 98
    4.3.6 Different numbers of layers ................................................ 98
    4.3.7 Different numbers of neurons .............................................. 99
    4.3.8 Different dimensions .......................................................... 100
    4.3.9 Different number of training samples .................................. 101
    4.3.10 Transfer learning ............................................................... 102
  4.4 HSI Compression Experiments .................................................... 106
  4.5 Summary ................................................................................... 112

5 Concluding remarks ........................................................................ 114
5.1 Contributions ................................................................. 114
5.2 Future work ................................................................. 116
  5.2.1 HSI classification .................................................. 116
  5.2.2 HSI target/anomaly detection .................................... 116
  5.2.3 Regular image compression ....................................... 117
  5.2.4 Modify the loss function for GAN ............................... 117
  5.2.5 Convolutional network to simulate nonlinear DR method . 118

Appendices 119

List of Figures

2.1 An illustration of an HSI scene, taken from [4]. In the figure, the sun, air, tank
and satellite correspond to the four basic elements in the hyperspectral remote
sensing system: source, atmospheric path, target and sensor. ....................... 22
2.2 Schematic diagram of HSI spectrometer, taken from [5]. Many detector arrays
are used to measure reflectance with different locations and bands. .............. 22
2.3 Schematic illustration of a hyperspectral image as a "cube" of data. Tens or
hundreds of narrow bands (different layers in the image) .......................... 23
2.4 PCA and KPCA results for linear data. In the left column, the original data and
PCs in PCA are shown. In the middle column, the results of PCA in 2-D and
1-D are shown. In the right column, the results of KPCA in 2-D and 1-D are
shown. ................................................................. 32
LIST OF FIGURES

2.5 PCA and KPCA results for nonlinear data1. In the left column, the original data and PCs in PCA are shown. In the middle column, the results of PCA in 2-D and 1-D are shown. In the right column, the results of KPCA in 2-D and 1-D are shown. Figure from [6]. .................................................. 33

2.6 PCA and KPCA results for nonlinear data2. In the left column, the original data and PCs in PCA are shown. In the middle column, the results of PCA in 2-D and 1-D are shown. In the right column, the results of KPCA in 2-D and 1-D are shown. Figure from [6]. .................................................. 34

2.7 DR results for S-shaped Data. In the leftmost column, the 3-D original S-shaped data is shown. Eight DR results are shown in the right including PCA and seven nonlinear DR results. Figure from [7]. .................................................. 35

3.1 Left: Indian Pines image (spectral bands 29, 15, 12) and manually labeled reference data (16 classes). Right: SLIC superpixels for various choices of $s$ (superpixel size) and $r$ (superpixel regularity). .................................................. 48

3.2 Left: Salinas image (spectral bands 29, 15, 12) and manually labeled reference data (16 classes). Right: SLIC superpixels for various choices of $s$ (superpixel size) and $r$ (superpixel regularity). .................................................. 49

3.3 Left: Pavia image (spectral bands 48, 17, 6) and manually labeled reference data (9 classes). Right: SLIC superpixels for various choices of $s$ (superpixel size) and $r$ (superpixel regularity). .................................................. 50

3.4 Pipeline of proposed method for pixel-wise HSI classification. .................................................. 53

3.5 Reference data and class maps from classification of Indian Pines, Salinas, and Pavia imagery using SSSE and KPCA embeddings computed from original image pixels. .................................................. 57
LIST OF FIGURES

3.6 Overall classification accuracy (top) and computation time (bottom) as super-

pixel size ($s$) varies for SLIC superpixel-based nonlinear DR via SSSE (left) and

KPCA (right) on Indian Pines. ....................................................... 59

3.7 Overall classification accuracy (top) and computation time (bottom) as super-

pixel size ($s$) varies for SLIC superpixel-based nonlinear DR via SSSE (left) and

KPCA (right) on Salinas. ............................................................ 61

3.8 Overall classification accuracy (top) and computation time (bottom) as super-

pixel size ($s$) varies for SLIC superpixel-based nonlinear DR via SSSE (left) and

KPCA (right) on Pavia. ............................................................. 62

3.9 Overall classification accuracy (top) and computation time (bottom) as super-

pixel regularity ($r$) varies for SLIC superpixel-based nonlinear DR via SSSE (left)

and KPCA (right) on Indian Pines. ............................................. 63

3.10 Overall classification accuracy (top) and computation time (bottom) as super-

pixel regularity ($r$) varies for SLIC superpixel-based nonlinear DR via SSSE (left)

and KPCA (right) on Salinas. .................................................... 64

3.11 Overall classification accuracy (top) and computation time (bottom) as super-

pixel regularity ($r$) varies for SLIC superpixel-based nonlinear DR via SSSE (left)

and KPCA (right) on Pavia. ...................................................... 65

3.12 Left: Indian Pines reference data; right: Classification maps based on SSSE

embeddings computed with SLIC superpixels for various choices of $s$ (size) and

$r$ (regularity). ................................................................. 66

3.13 Left: Indian Pines reference data; right: Classification maps based on KPCA

embeddings computed with SLIC superpixels for various choices of $s$ (size) and

$r$ (regularity). ................................................................. 67
LIST OF FIGURES

3.14 Left: Salinas reference data; right: Classification maps based on SSSE embeddings computed with SLIC superpixels for various choices of $s$ (size) and $r$ (regularity). ................................................................. 68

3.15 Left: Salinas reference data; right: Classification maps based on KPCA embeddings computed with SLIC superpixels for various choices of $s$ (size) and $r$ (regularity). ................................................................. 69

3.16 Left: Pavia reference data; right: Classification maps based on SSSE embeddings computed with SLIC superpixels for various choices of $s$ (size) and $r$ (regularity). ................................................................. 70

3.17 Left: Pavia reference data; right: Classification maps based on KPCA embeddings computed with SLIC superpixels for various choices of $s$ (size) and $r$ (regularity). ................................................................. 71

4.1 Structure of an autoencoder network with three hidden layers. Figure from [8]. . 78

4.2 SSDAN Structure ................................................................. 88

4.3 Flowchart showing the use of a SSDAN as a preprocessing step for classification. 89

4.4 Left: reference data of Indian Pines; middle: Classification maps for SSDAN-based SSSE method of Indian Pines; right: Classification maps for SSDAN-based KPCA method of Indian Pines. ................................................................. 93

4.5 Left: reference data of Salinas; middle: Classification maps for SSDAN-based SSSE method of Salinas; right: Classification maps for SSDAN-based KPCA method of Salinas. ................................................................. 94

4.6 Left: reference data of Pavia; middle: Classification maps for SSDAN-based SSSE method of Pavia; right: Classification maps for SSDAN-based KPCA method of Pavia. ................................................................. 94
LIST OF FIGURES

4.7 OA of pixel-based HSI classification when different numbers of sub-loss terms are used. One sub-loss: autoencoder loss; two sub-losses: autoencoder + mean loss; three sub-losses: autoencoder + mean + manifold loss; four sub-losses: autoencoder + mean + manifold + class loss. 97

4.8 OA of pixel-based HSI classification when different coefficient values are chosen for each sub-loss term. 98

4.9 OA of pixel-based HSI classification using SSDANs with different activation functions. 99

4.10 OA of pixel-based HSI classification using SSDANs with different numbers of layers. 100

4.11 OA of pixel-based HSI classification using SSDAN with different numbers of neurons. 101

4.12 OA of pixel-based HSI classification using SSDAN with different feature dimensions. 101

4.13 OA of pixel-based HSI classification using SSDANs as a function of different training set sizes. 102

4.14 The Berlin pseudo-color image (spectral bands 2, 4, 6 in Landsat 8) and manually labeled reference data (11 classes). 105

4.15 The Paris pseudo-color image (spectral bands 1, 3, 5 in Landsat 8) and manually labeled reference data (12 classes). 105

4.16 OA of pixel-based HSI classification using SSDANs with transfer learning. 106

4.17 The original and recovered data. 107

4.18 PSNR and CR versus OA of pixel-based HSI classification for Indian Pines, Salinas and Pavia. 109

4.19 Histogram of mean square error (MSE) for all of the pixels in Indian Pines, Salinas and Pavia with PCA and SSDAN methods. 110
4.20 L2 norm of the differences between actual spectra and reconstructed spectra at each pixel for Indian Pines, Salinas and Pavia. The left column corresponds to PCA reconstruction error and right column corresponds to SSDAN reconstruction error.

List of Tables

3.1 Names and the number of samples for each reference data class from each image. 54
3.2 Definitions of per-class classification performance measures, in terms of true positives (TP), false positives (FP), false negatives (FN), and true negatives (TN). 55
3.3 Baseline classification results for Indian Pines, Salinas, and Pavia, without using superpixels. Performance measures include precision (Pr), sensitivity (Se), overall accuracy (OA), average accuracy (AA), kappa coefficient (κ), and computing time. 58
3.4 Comparison with state-of-the-art classification methods on Indian Pines. 73
3.5 Comparison with state-of-the-art classification methods on Salinas. 73
3.6 Comparison with state-of-the-art classification methods on Pavia. 74
4.1 Results of pixel-wise classification using original HSI data, and using low-dimensional embeddings constructed from PCA, traditional SSSE, traditional KPCA, SSDAN-KPCA and SSDAN-SSSE. 93
4.2 Results of different data augmentation methods for SSDAN-SSSE. 96
4.3 Results of different data augmentation methods for SSDAN-KPCA. 96
4.4 Number of neurons in each layer of encoder network. 99
4.5 Number of neurons in each layer of encoder network. 100
4.6 Names and the number of samples for each reference class in Berlin and Paris image. ................................................. 113
4.7 Fraction of variance for PCA ................................................. 113

1 Confusion matrix of superpixel-based SSSE method with $s = 15$ and $r = 1$ for Indian Pines. Rows correspond to the reference data and columns correspond to the predicted result. ................................................. 120
2 Confusion matrix of superpixel-based KPCA method with $s = 15$ and $r = 1$ for Indian Pines. Rows correspond to the reference data and columns correspond to the predicted result. ................................................. 120
3 Confusion matrix of superpixel-based SSSE method with $s = 15$ and $r = 1$ for Salinas. Each row corresponds to the reference data and each column corresponds to the predicted result. ................................................. 121
4 Confusion matrix of superpixel-based KPCA method with $s = 15$ and $r = 1$ for Salinas. Each row corresponds to the reference data and each column corresponds to the predicted result. ................................................. 121
5 Confusion matrix of superpixel-based SSSE method with $s = 15$ and $r = 1$ for Pavia. Each row corresponds to the reference data and each column corresponds to the predicted result. ................................................. 121
6 Confusion matrix of superpixel-based KPCA method with $s = 15$ and $r = 1$ for Pavia. Each row corresponds to the reference data and each column corresponds to the predicted result. ................................................. 122
7 Confusion matrix of SSDAN-based SSSE method for Indian Pines. Each row corresponds to the reference data and each column corresponds to the predicted result. ................................................. 122
8 Confusion matrix of SSDAN-based KPCA method for Indian Pines. Each row corresponds to the reference data and each column corresponds to the predicted result. ................................................................. 122
9 Confusion matrix of SSDAN-based SSSE method for Salinas. Each row corresponds to the reference data and each column corresponds to the predicted result. 123
10 Confusion matrix of SSDAN-based KPCA method for Salinas. Each row corresponds to the reference data and each column corresponds to the predicted result. ................................................................. 123
11 Confusion matrix of SSDAN-based SSSE method for Pavia. Each row corresponds to the reference data and each column corresponds to the predicted result. 123
12 Confusion matrix of SSDAN-based KPCA method for Pavia. Each row corresponds to the reference data and each column corresponds to the predicted result. 124
Chapter 1

Introduction

Nonlinear dimensionality reduction (DR) methods such as Local Linear Embedding (LLE) [1], Isometric Feature Mapping (ISOMAP) [2], Kernel Principal Components Analysis (KPCA) [3], Laplacian Eigenmaps (LE) [9], Schroedinger Eigenmaps (SE) [10] and Spatial Spectral Schroedinger Eigenmaps (SSSE) [11] have been widely used to generate representations of hyperspectral imagery (HSI) that are used as input for clustering, segmentation, classification, target detection, and anomaly detection algorithms. Compared with linear DR methods such as Principal Components Analysis (PCA) [12], nonlinear DR methods are much more effective at yielding low-dimensional representations that reflect the structure of the manifolds in high-dimensional space on which the original data reside. However, they often suffer computation/memory issues because of the need to use the full set of training samples to construct adjacency matrices or kernels, and they require solving large generalized eigenvalue problems.

In this thesis, we explore two different ideas for significantly improving computational and memory efficiency of nonlinear DR methods: using superpixel representations of HSI as input to the DR methods, and approximating the solutions to the DR methods with deep autoencoder networks (DANs). For the first idea, we pre-cluster the hyperspectral image into Simple Linear Iterative Clustering (SLIC) superpixels. Each superpixel may represent tens or hundreds of
original image pixels. Performing nonlinear DR with the superpixels as input significantly reduces the computational effort required; however, care must be taken to choose appropriate values for hyperparameters and to interpolate the resulting embeddings back to pixel resolution.

For the second idea, we develop a semi-supervised deep autoencoder network (SSDAN) that is capable of generating mappings that approximate the embeddings computed by the nonlinear DR methods. The SSDAN can be trained with only a small subset of the original data. The SSDAN enables an expert user to provide constraints that can bias data points from the same class towards being mapped closely together. Once the SSDAN is trained on a small subset of the data, it can be used to map the rest of the data to the lower dimensional space, without requiring complicated out-of-sample extension procedures.

In order to validate these two ideas, we carry out a set of experiments on three publicly-available hyperspectral images: Indian Pines, Salinas, and Pavia, with two popular nonlinear DR methods: KPCA and SSSE. Features are extracted from the two algorithms to simulate traditional nonlinear DR methods, and they will be provided as inputs to pixel-based HSI classification task.

The remainder of this thesis is organized as follows. Chapter 2 provides background of HSI processing and analysis, and it describes dimensionality reduction techniques and their use in HSI applications. Chapter 3 describes how SLIC superpixels can be used as input to nonlinear DR methods, and it shows how the resulting embeddings can be used to significantly improve the computational efficiency of pixel-wise HSI classification. Chapter 4 details the SSDAN shows how it can be applied to the problems of pixel-wise classification and compression of HSI. Finally, Chapter 5 summarizes the results of this thesis and provides ideas for future work.
Chapter 2

Background

In this Chapter, we first introduce the concept and characteristics of hyperspectral imaging (HSI). We then describe various HSI tasks in which machine learning techniques play a role, including classification, target detection, and compression. Next, we introduce linear and non-linear dimensionality reduction (DR) techniques, with emphasis on how they are used in HSI applications. Finally, we summarize limitations of current DR techniques for HSI.

2.1 Hyperspectral imaging (HSI)

Hyperspectral imaging (HSI) originated in the 1980’s when researchers at the Jet Propulsion Laboratory tried to develop new measurement and observation instruments such as Airborne Visible/Infrared Imaging Spectrometer (AVIRIS) [13]. Different from other traditional RGB images or multispectral images, HSI can acquire tens or hundreds of narrow, contiguous spectral bands in the same scene simultaneously [14], ranging from visible light to infrared. Airborne and satellite HSI systems have since undergone rapid development and the famous systems include the Hyperspectral Digital Imagery Collection Experiment (HYDICE) [15], Airborne Real-time Cueing Hyperspectral Enhanced Reconnaissance (ARCHER) [16], Advanced Responsive Tactically Effective Military Imaging Spectrometer (ARTEMIS) [17] systems and the hyperspectral
sensors currently operating in space are Hyperion (USA) [18].

A hyperspectral remote sensing system usually includes at least four elements: a source (e.g. the sun), a target or region of interest (ROI), an atmospheric path, and a sensor [4]. We show that in Figure 2.1. Light is emitted from the source, travels through the atmosphere to the target, reflects back through the atmosphere, and finally is captured by the sensor which is on a satellite or aircraft. Figure 2.2 shows how a push broom HSI scanner [19] works. The light is collected by telescope and comes through the small slit. Through the diffraction grid, light with different wavelengths is projected into different domains of a CCD sensor. As shown in Figure 2.3, pixels in different classes have different spectral curves. A hyperspectral image can be interpreted as a 3D cube in which two of the dimensions represent spatial coordinates and the other dimension represents spectral coordinates (wavelengths). Each slice of an HSI at a specific spectral wavelength can be thought of as a monochrome image.

The primary advantage of HSI systems is that they can capture much more spectral information than other imaging systems, making them especially useful for applications such as precision agriculture, mineralogy, and surveillance [20]. A current “hot topic” involves using HSI to explore the effect of oil leakages in the sea based on the spectral signatures of oil and gas [21]. Another example is in military surveillance, where HSI is used to detect objects that are not visible to the human eye [20]. In spite of the power of HSI systems, their main disadvantage is the amount of memory and computational complexity required to store, transmit, and analyze the images. In addition, the high number of spectral bands can make it difficult to estimate the parameters necessary for various analysis algorithms. In these cases, dimensionality reduction (DR) can be used as a preprocessing step to simplify the resulting algorithms. In the following parts of this chapter, we will introduce different applications of HSI in which DR has the potential to be useful, including pixel-wise classification, target detection and compression, and we will describe how specific DR methods have been used in HSI analysis.
CHAPTER 2. BACKGROUND

Figure 2.1: An illustration of an HSI scene, taken from [4]. In the figure, the sun, air, tank and satellite correspond to the four basic elements in the hyperspectral remote sensing system: source, atmospheric path, target and sensor.

Figure 2.2: Schematic diagram of HSI spectrometer, taken from [5]. Many detector arrays are used to measure reflectance with different locations and bands.
Figure 2.3: Schematic illustration of a hyperspectral image as a "cube" of data. Tens or hundreds of narrow bands (different layers in the image) are acquired simultaneously to provide detailed reflectance spectra for each pixel in the image, taken from [22].
2.2 HSI Analysis Tasks

2.2.1 Pixel-based Classification

Pixel-wise classification of HSI is very useful in many applications. Based on rich spectral information of HSI, the classification aims to distinguish different objects in the scene [23]. Existing algorithms for HSI classification are mostly based on conventional machine learning techniques and deep learning [24]. In the following sections, we will describe many of these methods.

Classical machine learning techniques for pixel-wise HSI classification have been extensively studied since the 1990’s. Tso et al. use the genetic algorithm and Markov random fields to including neighboring information of each pixel for spatio-contextual image classification [25]. Melgani et al. use support vector machine (SVM) as classifier to achieve the state-of-the-art classification accuracy of HSI [26]. Multinomial logistic regression has been used to simulate different class distributions in a Bayesian framework, but it will sometimes suffer from generalization issue [27]. Sparse multinomial logistic regression is more generalizable, although it is computationally intensive [28]. Kernel methods [29, 30, 31, 32] and decision trees / random forests [33, 34] have shown successful on pixel-wise HSI classification, but they can be computationally expensive to train and can rely on hand-crafted features.

Deep learning models have achieved a breakthrough in both image-wise and pixel-wise RGB image classification, and they have become popular for HSI classification. Makantasis et al. [35] proposed a convolutional neural network (CNN) architecture for HSI classification. Pixels are combined with their neighbors and are fed into the network, and the network outputs the class label for each central pixel. The advantage of this method is that spatial information is combined together with the spectral information. Romero et al. [36] proposed using a layer-by-layer greedy unsupervised learning method to formulate a CNN model for remotely sensing imagery. Cao et al. [37] combine Markov random fields and CNNs to perform pixe-wise HSI classification.
Zhang et al. [38] propose that the outputs from the top layers of a network can be directly used for a subsequent classifier for pixel-based HSI classification. Mou et al. [39] model HSI pixels as sequential data and use a recurrent neural network (RNN) model for HSI classification. RNNs exploit features in the both the current sequence and the history to improve accuracy. Chen et al. [40] combine principal components analysis (PCA) and hierarchical learning-based feature extraction to yield features that are input into logistic regression for classification. Zhao et al. [41] extract features from different levels of a multiscale CNNs to perform classification.

In summary, a variety of different algorithms have been proposed for pixel-wise HSI classification domain, and many of these algorithms report good accuracy. However, many are limited in their ability to handle large amounts of data with a reasonable amount of computation and memory. Deep learning approaches appear particularly promising, however, deep CNN/RNN architectures require a large amount of labeled training data, but the amount of available HSI reference data is limited.

### 2.2.2 Target Detection

Target detection (TD) in HSI is widely used in both military and civilian applications [4]. The general goal of TD is to find small and rare objects in a HSI that exhibit a particular “target” spectrum. Target detection is challenging due to the small size of objects, and due to nonlinear relationships between the actual target spectrum (which may be measured under laboratory conditions) and the observed spectrum of the object (which may be distorted due to noise, atmosphere, etc.). Moreover, computing time could be very significant because the number of HSI bands is large.

Reed and Yu develop the RX algorithm for anomaly detection in HSI [42], and this algorithm has been extended to handle target detection and change detection. It is considered to be a benchmark for comparison for other proposed unsupervised TD algorithms. In the RX algorithm, the Mahalanobis distance is computed between the mean spectral vector for the
area of interest and the spectral vector of the background. A larger distance means the area of interest significantly differs from the background, so it is more likely to be an anomaly or the target. Stocker et al. [43] modify RX to utilize spectral and spatial information provided by a passive infrared sensor to improve detectability of the target. Other popular supervised TD methods are constrained energy minimization (CEM) [44] and matched filter (MF) [45].

In CEM, a weight vector is formed by the autocorrelation matrix of the entire image and the target is detected by minimizing the average output power; if the inner product between the weight vector and a pixel is close to 1, it is more likely that the pixel belongs to the target. In MF, the covariance matrix of the entire image is efficiently exploited to extract background information uniformly distributed in the image. The main disadvantage of RX, CEM and MF is that they usually require that the target spectrum be very precise, and they are not robust to situations where this is not the case.

Recently, TD based on graph-based methods such as Laplacian Eigenmaps (LE) [46] have become popular. In LE, each pixel in a HSI is interpreted as a vertex in an undirected weighted graph, and the weighted adjacency matrix is used to represent the relationships between different pixels. The generalized eigenvectors of the graph Laplacian matrix corresponding to a few small generalized eigenvalues can be used as a basis for a low-dimensional space in which local distances between data points are preserved. Schroedinger Eigenmaps [47, 48] extends LE by adding a potential matrix to the graph Laplacian that enables incorporating expert information about some subset of pixels. In Dorado Munoz et al. [49], a barrier potential matrix encoding target information is added to the graph Laplacian matrix so that points close to the target will be pulled to the origin in the low-dimensional embedding. In Cahill et al. [50], Spatial-Spectral Schroedinger Eigenmaps (SSSE) is proposed, in which both the spatial and spectral information in a HSI are combined together to form a cluster potential matrix that is added to the graph Laplacian matrix. The resulting low-dimensional embedding pulls together points that are close in terms of both spatial and spectral distances. Maji et al. [51] propose biased normalized cuts.
(BNC) as a way to modify a pre-computed embedding to bias it towards particular pixels. Zhang et al. [52] show how BNC and SSSE could be combined to create a HSI target detection algorithm that yields state-of-the-art results.

Deep learning has not yet appeared to be popular for HSI target detection, which is likely due to the difficulty of finding enough prior target information for training. Classical methods have been demonstrated that they are very powerful because there is usually enough spectral information in the target objects.

2.2.3 Compression

Compression of HSI is an attractive topic since HSI datasets can be massive and often need to be transmitted as efficiently as possible. Compression methods can be separated into two classes: lossless and lossy [53]. In the lossless case, the original HSI data can be exactly recovered from the compressed data, while in the lossy case, only an approximation of the original data can be recovered.

Traditional HSI compression algorithms have used Principal Components Analysis (PCA) or Independent Component Analysis (ICA) [54] to reduce the number of spectral bands to a small number of uncorrelated or independent bands. However, they typically ignore spatial correlations/dependencies in the data. In order to move beyond that limitation, some methods consider spatial and spectral correlations together. Those algorithms include set partitioned embedded block (SPECK) and set partitioning in hierarchical trees (SPIHT) [55]. Experimental results show that the SPECK is more efficient than SPIHT. Another technique that incorporates spatial correlations is introduced in [56]. Here, PCA is combined with JPEG2000, and a few of the principal components are retained and encoded. Gisela et al. [57] also develop a compression algorithm based on JPEG2000. They choose the frequency of the lossy layer to achieve near-lossless compression.

There are some other kinds of compression methods for HSI. Huang et al. [58] propose to use
a lookup table to estimate current pixel values from co-located pixels for lossless compression. Their method has low time complexity and could be used in an on-board system. Nian et al. [59] propose to use distributed source coding and rate allocation and quantization for a near-lossless compression. Their algorithm could also be implemented on-board, but more explorations are needed to reduce the bit rate. In [60], a lossy compression algorithm based on multistage lattice vector quantization is proposed. The algorithm focuses on using multistage lattice vector quantization to explore the correlations between different bands. Jia et al. [61] propose the FIVQ algorithm in which four images are checked simultaneously to determine whether they have similar mean values. Lucana et al. [62] propose to use H.264/AVC in HSI. H.264/AVC is a video coding standard, and HSI band corresponds to each frame in the video. The algorithm has been tested in a large AVIRIS images and preforms very well.

In summary, there are many different HSI compression methods. However, it seems to be difficult to compare among them unless data sets and performance measures are the same.

2.3 Dimensionality Reduction for HSI

HSI data typically has tens or hundreds of spectral bands. These bands are almost certainly correlated, and the pixels are also likely spatially correlated if the ground sample distance is small compared to objects/regions of interest. Hence, there is definitely redundant information in an HSI [63, 64, 65]. Furthermore, large HSI datasets are cumbersome, hindering data transmission and mining [66, 67]. In [67], David shows that for HSI data, the high-dimensional spectral space is mostly empty, and the data exists primarily in a subspace. Dimensionality reduction (DR) aims at generating a low-dimensional representation or embedding that reveals meaningful structures in high-dimensional multivariate data [68]. The use of low-dimensional embeddings can also decrease space and time complexity of various analysis tasks and relieve ill-conditioned situations by removing redundant features [69, 70]. DR can also enable useful features to be efficiently extracted from HSI [71]. Because of this, it may be possible to improve the accuracy
CHAPTER 2. BACKGROUND

of pixel-wise classification, and it may also be possible to easily visualize the distribution of different classes [68, 72]. In summary, DR can be advantageous as a pre-processing tool for various HSI tasks. In the following parts, we will discuss linear versus nonlinear DR, and then we will focus on two important nonlinear DR methods: Kernel Principal Components Analysis (KPCA) and Spatial Spectral Schroedinger Eigenmaps (SSSE). Finally, we will summarize the advantages and disadvantages of nonlinear methods, leading to the proposed improvements in later chapters of this thesis.

2.3.1 Linear Dimensionality Reduction

Linear DR assumes that the data resides in or is close to a linear subspace of the high dimensional space. Due to its simplicity and efficiency, the most popular linear DR method is principal component analysis (PCA) [73, 74, 75, 76], which produces a set of uncorrelated axis ordered in terms of decreasing variance by orthogonal projection. The eigenvalues of the covariance matrix of the data are used to determine the significance of the principal components, and DR is accomplished by keeping only a small number of components corresponding to the largest eigenvalues [54]. A recent improvement to PCA is noise-adjusted PCA [77], which orders the data in terms of signal-to-noise ratio, so that noise is de-emphasized in the DR result [78]. Another modification of PCA is segmented PCA [79], which is applied to group original bands in HSI into subsets of highly correlated adjacent bands.

Fisher’s linear discriminant analysis (LDA) [80, 81, 82, 83, 84] is another popular linear DR method. Different from PCA, LDA is a supervised method that requires class labels for the data. LDA minimizes a loss function so that the distances between samples in the same classes are small while the distances between samples from different classes are large. This has the impact of ensuring that samples from the same class will be closer, and samples from different classes with be farther, in the low-dimensional representation.

For both PCA and LDA, a key limitation is that the class-conditional distributions are
CHAPTER 2. BACKGROUND

assumed to be multi-variate Gaussian [82]. However, real-world HSI data is often not Gaussian, and PCA and LDA will not yield useful embeddings under those conditions. An alternative linear DR method that does not assume Gaussianity is Independent Component Analysis (ICA) [85]. It transforms data to another space in which the components are statistically independent. In contrast to PCA, ICA not only decorrelates 2nd-order statistics but also removes all higher-order dependencies.

A disadvantage of both LDA and ICA is that they do not consider spatial correlations or dependencies between different pixels, similar to the disadvantage of PCA [76]. One remaining linear DR method, Locality Preserving Projections (LPP) [86], has the potential to incorporate both spectral and spatial dependencies, but it has not been applied to HSI data.

2.3.2 Nonlinear Dimensionality Reduction

Compared with linear DR methods, nonlinear DR methods do not assume that data in the high-dimensional space resides in a (linear) subspace. Instead, many nonlinear DR methods assume that the data lie on or near a manifold that might have nontrivial curvature. If the data lies on a non-linear manifold, linear methods will provide poor low-dimensional representations, and they will usually overestimate the intrinsic dimensionality of the manifold [71]. Nonlinear DR methods focus on computing low-dimensional representations that preserve the structure of the manifold, and they are capable of integrating both the spatial and spectral information inherent in HSI. They have been shown to provide low-dimensional representations that can be effectively used as input for clustering, segmentation, and classification of HSI.

A variety of nonlinear approaches to dimensionality reduction have been investigated with respect to applications in HSI, including Local Linear Embedding (LLE) [1], Isometric Feature Mapping (ISOMAP) [2], Kernel Principal Components Analysis (KPCA) [3], Laplacian Eigenmaps (LE) [9], Diffusion Maps [87], Stochastic Proximity Embedding (SPE) [88], Local Tangent Space Analysis (LTSA) [89], linear local tangent space alignment (LLTSA) [90],
CHAPTER 2. BACKGROUND

31

t-Distributed Stochastic Neighbor Embedding (t-SNE) [91], curvilinear components analysis (CCA) [92], Maximum Variance Unfolding (MVU) [93], Schroedinger Eigenmaps (SE) [94] and Spatial Spectral Schroedinger Eigenmaps (SSSE) [11]. However, nearly all of these nonlinear DR methods have been applied only to small images or tiles. Two of the greatest barriers to effective use of nonlinear DR methods in HSI processing are their computational complexity and memory requirements. Fong [95] shows that LLE, LTSA and LLTSA are incapable of handling HSI having more than $70 \times 70$ pixels; for SPE, KPCA and CFA, this number of pixels reduces to $50 \times 50$. Although [95] is now over ten years old, its conclusions have not changed dramatically in the last decade. In attempting to run various nonlinear DR algorithms on the $512 \times 217$ pixel Salinas image [96] on a modern desktop computer (AMD FX-6300 Six-Core Processor, 24 GB memory), we run out of memory when attempting to perform LLE, ISOMAP, LTSA and t-SNE. LE does successfully run under the constraint that only a small number (20) of neighbors can be used to construct the graph; however, the accuracy of a subsequent random forest classifier is worse than that achieved with PCA dimension reduction. SSSE and KPCA DR algorithms also successfully run on the same desktop computer, and subsequent random forest classification is superior than when PCA is used for DR; however, computation time is huge: 1,716 seconds for SSSE and 432,173 seconds (more than 5 days) for KPCA.

In Figures 2.4–2.7, linear and nonlinear DR methods are illustrated and compared using toy datasets. Figure 2.4 shows the results of PCA and KPCA on a 2-D linearly separable dataset. When the dimension is decreased to one, the DR results for both PCA and KPCA are good in that only a few points in the two classes are overlapping. In Figures 2.5–2.6, PCA and KPCA are compared using two different nonlinear datasets. For both of these datasets, the points in KPCA result can be separated linearly, but PCA not. When the dimension is decreased to one, the KPCA embeddings totally separate the two classes in both datasets, but the PCA embeddings do not. In Figure 2.7, 3-D S-shaped data is transformed to 2-D with one linear (PCA) and seven nonlinear DR methods. All of the DR methods preserve properties of the original data.
CHAPTER 2. BACKGROUND

Figure 2.4: PCA and KPCA results for linear data. In the left column, the original data and PCs in PCA are shown. In the middle column, the results of PCA in 2-D and 1-D are shown. In the right column, the results of KPCA in 2-D and 1-D are shown.
Figure 2.5: PCA and KPCA results for nonlinear data. In the left column, the original data and PCs in PCA are shown. In the middle column, the results of PCA in 2-D and 1-D are shown. In the right column, the results of KPCA in 2-D and 1-D are shown. Figure from [6].
Figure 2.6: PCA and KPCA results for nonlinear data. In the left column, the original data and PCs in PCA are shown. In the middle column, the results of PCA in 2-D and 1-D are shown. In the right column, the results of KPCA in 2-D and 1-D are shown. Figure from [6].
However, if we just focus along the horizontal axes, we can see that the Isomap result is much better than that of PCA. All of points with different colors in the Isomap embedding are barely overlapping in the first dimension but it’s obvious that there is the mixture between red and yellow points in the PCA embedding.

Because HSI data can be very nonlinear, nonlinear DR can achieve much more useful embeddings for HSI tasks than linear DR; however, nonlinear DR methods are not very efficient. In the remaining chapters of this thesis, we will propose two ideas to make them more computational tractable and memory efficient. Before that, however, we will focus on two nonlinear DR methods (KPCA and SSSE) that will be detailed below. In later chapters, we will compare our proposed ideas with these two nonlinear DR methods by performing extensive experiments.
2.4 Example Nonlinear DR Methods

In the previous section, we showed examples of how nonlinear DR can achieve better performance than linear DR. In this section, we will detail two nonlinear DR methods, SSSE and KPCA, which will be used in subsequent experiments.

2.4.1 Spatial Spectral Schroedinger Eigenmaps (SSSE)

SSSE [97] is a nonlinear DR algorithm that generalizes both Laplacian Eigenmaps (LE) and Schroedinger Eigenmaps (SE) by integrating spatial and spectral information together when constructing the adjacency matrix of the graph representing the HSI. We will denote $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ to be a set of $N$ points on a manifold $\mathcal{M} \in \mathbb{R}^n$, where $n$ is assumed to be large. Each dimensionality reduction algorithm will identify a set of corresponding points $\mathcal{Y} = \{\mathbf{y}_1, \ldots, \mathbf{y}_N\}$ in $\mathbb{R}^m$, where $m \ll n$, so that the relationships of points in $\mathcal{Y}$, such as whether they are neighbors, are similar to the relationships of the corresponding points in $\mathcal{X}$.

Laplacian Eigenmaps

Laplacian Eigenmaps (LE) [98] is a popular graph-based dimensionality reduction algorithm that was introduced by Belkin and Niyogi in 2003 and involves the following three steps:

1. Construct an undirected graph $\mathcal{G} = (\mathcal{X}, \mathcal{E})$ whose vertices are the points in $\mathcal{X}$ and whose edges $\mathcal{E}$ are defined based on proximity between vertices.

2. Define weights for the edges in $\mathcal{E}$ to form the matrix $\mathbf{W}$. The entry $W_{i,j}$ is the weight for the edge connecting $\mathbf{x}_i$ and $\mathbf{x}_j$.

3. Compute the smallest $m + 1$ eigenvalues and eigenvectors of the generalized eigenvector problem $\mathbf{Lf} = \lambda \mathbf{Df}$, where $\mathbf{D}$ is the diagonal weighted degree matrix defined by $D_{i,i} = \sum_j W_{i,j}$, and $\mathbf{L} = \mathbf{D} - \mathbf{W}$ is the Laplacian matrix. If the resulting eigenvectors $f_0, \ldots,$
$f_m$, are ordered so that $0 = \lambda_0 \leq \lambda_1 \leq \cdots \leq \lambda_m$, then the points $y_1^T, y_2^T, \ldots, y_3^T$ are defined to be the rows of $F = [f_1 \ f_2 \ \cdots \ f_m]$.

One of the great strengths of LE is its flexibility in allowing different ways to define edges and edge weights. Common ways to define edges use $\epsilon$-neighborhoods or (mutual) $k$-nearest neighbors search in some metric space. To define edge weights, the heat kernel is a common choice; i.e., the weight $W_{i,j}$ is defined to be $\exp \left( -\|x_i - x_j\|^2 / \sigma \right)$ if an edge exists between $x_i$ and $x_j$ or zero otherwise.

**Schroedinger Eigenmaps**

Schroedinger Eigenmaps (SE) [10, 94], a straightforward, yet powerful, generalization of LE, incorporates a potential matrix $V$ that encodes extra information about the data that may be available. The potential matrix includes barrier potentials for points in $\mathcal{X}$ that pull the corresponding points in $\mathcal{Y}$ towards the origin, and/or cluster potentials for points in $\mathcal{X}$ that pull the corresponding points in $\mathcal{Y}$ towards each other.

Barrier potentials are created by defining $V$ to be a nonnegative diagonal matrix, with $V_{i,i}$ defined to be positive for each of the selected $x_i$'s. Cluster potentials are created by defining $V$ to be a weighted sum of nondiagonal matrices $V^{(i,j)}$ that encode individual cluster potentials between $x_i$ and $x_j$:

$$V_{k,\ell}^{(i,j)} = \begin{cases} 
1, & (k, \ell) \in \{(i,i), (j,j)\} \\
-1, & (k, \ell) \in \{(i,j), (j,i)\} \\
0, & \text{otherwise}
\end{cases}$$  \hspace{1cm} (2.1)

With a potential matrix defined, SE proceeds in the same manner as LE, but with the generalized eigenvector problem in step 3 replaced by the problem $(L + \alpha V)f = \lambda Df$, where $\alpha$ is a parameter chosen to relatively weight the contributions of the Laplacian matrix and potential matrix.
Spatial Spectral Schroedinger Eigenmaps

Cahill et al. [97] proposed the Spatial Spectral Schroedinger Eigenmaps (SSSE) algorithm that defines graphs with spectral information and uses cluster potentials to encode spatial proximity. \( \mathbf{x}^f \) and \( \mathbf{x}^p \) can be used to represent spectral coordinates and spatial vectors of \( \mathbf{x} \).

Edges are defined based on proximity between the spectral components of the vertices, and edge weights are defined according to:

\[
W_{i,j} = \begin{cases} 
\exp \left( -\frac{\| \mathbf{x}^f_i - \mathbf{x}^f_j \|^2}{\sigma_f^2} \right), & (\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{E} \\
0, & \text{otherwise} 
\end{cases} 
\]  
(2.2)

A cluster potential matrix \( \mathbf{V} \) is defined to encode proximity between the spatial components of the vertices:

\[
\mathbf{V} = \sum_{i=1}^{k} \sum_{\mathbf{x}_j \in \mathcal{N}^p(\mathbf{x}_i)} V^{(i,j)} \cdot \gamma_{i,j} \cdot \exp \left( -\frac{\| \mathbf{x}^p_i - \mathbf{x}^p_j \|^2}{\sigma_p^2} \right),
\]  
(2.3)

where \( \mathcal{N}^p(\mathbf{x}_i) \) is the set of points in \( \mathcal{X} \) whose spatial components are in an \( \epsilon \)-neighborhood of the spatial components of \( \mathbf{x}_i \); i.e.,

\[
\mathcal{N}^p(\mathbf{x}_i) = \{ \mathbf{x} \in \mathcal{X} - \mathbf{x}_i \ \text{s.t.} \ \| \mathbf{x}^p_i - \mathbf{x}^p \| \leq \epsilon \} ,
\]  
(2.4)

\( V^{(i,j)} \) is defined as in (2.1), and \( \gamma_{i,j} \) can be chosen in a manner that provides greater influence for spatial neighbors having nearby spectral components. Cahill et al. [97] proposed two possibilities for \( \gamma_{i,j} \); in this work, we use \( \gamma_{i,j} = \exp \left( -\frac{\| \mathbf{x}^f_i - \mathbf{x}^f_j \|^2}{\sigma_f^2} \right). \)
2.4.2 Kernel Principle Component Analysis (KPCA)

KPCA [99] is a nonlinear extension of PCA [100]. By introducing kernels, the principle components are effectively computed in a higher dimensional space generated from a nonlinear mapping of the data. This section will introduce PCA and KPCA.

Principle Component Analysis (PCA)

PCA is a linear DR technique that assumes that the input data arises from a multivariate normal distribution, and it identifies an orthogonal set of basis vectors under which the data is whitened [101]. Assuming that the data is zero-centered/ i.e.,

$$\frac{1}{N} \sum_{i=1}^{N} x_i = 0,$$  \hspace{1cm} (2.5)

the sample covariance matrix is given by:

$$C = \frac{1}{N} \sum_{i=1}^{N} x_i x_i^T.$$  \hspace{1cm} (2.6)

The orthogonal basis vectors are the eigenvectors of the covariance matrix; i.e.,

$$Cv = \lambda v.$$  \hspace{1cm} (2.7)

The eigenvectors corresponding to the largest $m$ eigenvalues are the principal components and form the basis for the low-dimensional embedding, and the components of the representation $y_i$ of data point $x_i$ are given by the scalar projections of $x_i$ onto the principal components.

Kernel Principle Component Analysis (KPCA)

To generalize PCA in a kernel Hilbert space where the data can be linearly separated, consider a nonlinear function $\phi$ that maps each data point $x_i$ into a (higher-dimensional) feature vector
\( \phi(x_i) \). Now, consider that we do not explicitly know \( \phi(x) \), but we do know the \( N \times N \) kernel matrix \( K \), defined component-wise by:

\[
K_{i,j} = k(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle = \phi(x_i)^T \phi(x_j),
\]

\( i = 1, \ldots, N ; \quad j = 1, \ldots, N \).

The kernel \( k \) computes inner products in the feature space without actually explicitly mapping into the feature space. Common choices of kernel include the polynomial kernel

\[
k(x_i, x_j) = (x_i^T x_j + 1)^2,
\]

and the Gaussian kernel

\[
k(x_i, x_j) = \exp \left( -\frac{\|x_i - x_j\|^2}{2\sigma^2} \right).
\]

Suppose we wish to perform PCA in the feature space. If we assume for the moment that the feature vectors \( \{ \phi(x_1), \ldots, \phi(x_N) \} \) are centered, then the principal components are eigenvectors satisfying:

\[
\frac{1}{N} \sum_{i=1}^{N} \phi(x_i)^T \phi(x_i) v = \lambda v.
\]

(2.11)

If we write the eigenvector \( v \) as a linear combination of the feature vectors; i.e., \( v = \sum_{j=1}^{N} a_j \phi(x_j) \), and we define \( a = [a_1, \ldots, a_N]^T \), then (2.11) can be written as:

\[
\frac{1}{N} \sum_{i=1}^{N} \phi(x_i) \sum_{j=1}^{N} a_j K_{i,j} = \lambda \sum_{i=1}^{N} a_i \phi(x_i).
\]

(2.12)
This simplifies to:

\[
\frac{1}{N} \sum_{i=1}^{N} \phi(x_i) \left[ \sum_{j=1}^{N} a_j K_{i,j} - N \lambda a_i \right] = 0 .
\] (2.13)

Since the features are centered, (2.13) implies that \( \sum_{j=1}^{N} a_j K_{i,j} - N \lambda a_i \) equals a constant (say \( c \)) for all \( i = 1, \ldots, N \). Certainly, this is true for \( c = 0 \), in which case, we can write:

\[
K a = N \lambda a .
\] (2.14)

Now, the scalar projection of the feature vector \( \phi(x_i) \) onto the eigenvector \( v \) can be expressed as:

\[
comp_v \phi(x_i) = \frac{\phi(x_i) \, v}{v^T v} = \frac{\sum_{j=1}^{N} a_j \phi(x_i) \, v}{\sum_{i=1}^{N} \sum_{j=1}^{N} a_i a_j \phi(x_i) \, \phi(x_j)} = \frac{\sum_{j=1}^{N} a_j K_{i,j}}{\sum_{i=1}^{N} \sum_{j=1}^{N} a_i a_j K_{i,j}} = \frac{N \lambda a_i}{a^T Ka} = \frac{a_i}{a^T a} .
\] (2.15)

Hence, up to scale, the vector \( a \) contains the projections of the features onto one of the principal components. This result is very significant: PCA can be performed in the feature space using knowledge of the eigenvalues/eigenvectors of the kernel matrix \( K \) without ever requiring the features to be explicitly computed.

Note that (2.11)–(2.15) required assuming that the features are centered. This is not normally the case, however. In theory, one must subtract the mean features from \( \phi(x_i) \) and \( \phi(x_j) \) in order to compute the feature sample covariance matrix. This turns out to be equivalent to centering the kernel matrix: define \( K' \) by

\[
K' = K - 1_N K - K 1_N + 1_N K 1_N ,
\] (2.16)
where $1_N$ represents a $N$-by-$N$ matrix in which every element's value is $1/N$. Eigendecomposition via (2.14) can be done with $K'$ in place of $K$.

Based on the description of SSSE and KPCA, the reason why both of them are very slow is that an eigendecomposition needs to be solved for a $N$-by-$N$ matrix where $N$ is the number of pixels in the image. If we want to speed that up, one idea is to decrease $N$ and the other one is that we can try to skip the eigendecomposition and feed the data one by one. Superpixel-based algorithm corresponds the first idea. Some pixels are grouped together and the mean value and mean position are used to represent all of the pixels in each group. We also propose an autoencoder network which corresponds to the second idea to simulate nonlinear mapping function. When the model is trained, a batch of samples is fed into the network instead of all of them at the same time. The details of the superpixel and autoencoder ideas will be introduced in the following two chapters.

2.5 Summary

In this chapter, we introduced HSI and described some of the challenges associated with various analysis tasks. We then introduced the idea of using DR as a general preprocessing tool for HSI tasks. Linear DR, which is simple and fast but is not suitable for HSI data that can be highly nonlinear. Nonlinear DR methods are more suitable for HSI data than linear DR but they usually suffer from computation and memory issues. We detailed two specific nonlinear DR methods, SSSE and KPCA, and we saw why nonlinear DR methods are slow: they often require intensive linear algebra operations, like eigendecomposition, on large matrices. In the following chapters of this thesis, we propose two methods that can be used to overcome these computational issues of nonlinear DR techniques without sacrificing performance of particular HSI analysis tasks.
Chapter 3

Superpixel-based Dimensionality Reduction of HSI

Superpixels are spatially connected sets of pixels with similar intensities/spectra that are constructed and used to improve the computational efficiency and robustness of various image analysis systems. The mean intensity/spectra values and centroid positions of each superpixel are used in place of pixel values/locations for subsequent processing. There are a variety of different algorithms that have been developed for computing superpixels from RGB imagery, and some of these algorithms have been used in multispectral and hyperspectral image processing. In this chapter, we propose using superpixels to significantly reduce the amount of data required for nonlinear DR algorithms. Since superpixels contain groups of spatially contiguous pixels, low-dimensional embedding coordinates of the superpixels can be quickly computed and then easily interpolated back to the original pixel grid, eliminating the need for complicated out-of-sample extension procedures. Through a variety of experiments, we show that this strategy is highly effective for providing low-dimensional embeddings that can be used as inputs for pixel-based HSI classification in a fraction of the time that would be required to perform nonlinear DR directly on the full set of pixels.
3.1 Review of Superpixel Construction Techniques

Many superpixel construction techniques rely on graph-based image models, where each pixel in the image is considered as a vertex in a graph, and edges in the graph are defined between the vertices representing neighboring pixels. The Normalized Cuts superpixel construction algorithm [102] approximately minimizes a cost function defined in terms of contour and texture cues, generating a partitioning of the image (graph) into superpixels (subgraphs). Moore et al.’s algorithm [103] generates a lattice of superpixels that conform to a regular grid. Both of these algorithms have the potential to produce very regular superpixels; however, both algorithms have been proposed for grayscale/RGB imagery and require significant computational effort, making generalization to higher numbers of bands difficult. Felzenszwalb and Huttenlocher [104] propose a greedy algorithm that iteratively groups vertices in a graph to form superpixels. Their algorithm is computationally efficient, having complexity that is linear in the number of graph edges. However, it does not allow the user to control the size, shape, or compactness of the superpixels.

Veksler et al. [105] propose an energy minimization algorithm based on graph cuts, in which superpixels are constructed by assigning each pixel to an image patch and stitching together patches having the lowest energy. Two versions of this method include “variable patch” superpixels, which is more computationally efficient, and “constant intensity” superpixels, which has better boundary recall. The advantages of graph cuts energy minimization, as opposed to other graph-based methods for superpixel construction, are computational efficiency, principled optimization, and generalizability to three dimensions. However, superpixels generated from this method may widely vary in size. Liu et al.’s Entropy Rate algorithm [106] finds a partitioning of graph vertices that optimizes an energy function comprising two terms: the entropy rate of a random walk on the graph and a balancing term. The entropy rate term controls compactness and homogeneity of superpixels, whereas the balancing term biases superpixels towards having similar size. The energy function can be optimized efficiently using a greedy algorithm.
CHAPTER 3. SUPERPIXEL-BASED DIMENSIONALITY REDUCTION OF HSI

The TurboPixel algorithm [107] computes a set of regularly-distributed superpixels by dilating a set of seeds using geometric flow. The superpixels typically have relatively uniform size and boundary adherence. Compared to the graph-based superpixel techniques that rely on greedy algorithms, however, the TurboPixel algorithm is substantially slower and has poor boundary of segmentation, often yielding superpixels that are overly regular and compact.

Other techniques for determining superpixels are based on mode-seeking algorithms. In the Mean Shift algorithm [108], pixels associated with the same mode of an estimated probability density function form a superpixel. The algorithm uses an iterative scheme that is computationally intensive compared to some of the graph-based methods. In addition, although the user can tune a parameter describing the analysis resolution, there is no way to directly control the size, number, or compactness of superpixels. Quick Shift [109] is a different type of mode-seeking algorithm that uses medioid shift to more efficiently associate pixels with the nearest mode of the probability density function. While the quick shift algorithm allows the user to balance under- and over-fragmentation by tuning a parameter in the clustering procedure, it is still not possible to explicitly control the size or number of superpixels.

The method that has emerged as state-of-the-art for superpixel construction is the Simple Linear Iterative Clustering (SLIC) algorithm [110]. Compared with other superpixel methods, SLIC has three advantages: 1) it can be performed rapidly and with limited memory; 2) it enables the user to easily control the extent to which the superpixels adhere to edges/boundaries; and, 3) it is easy to generalize to higher dimensions. The SLIC algorithm is detailed in the following subsection.

3.2 Simple Linear Iterative Clustering (SLIC)

The SLIC algorithm [110] is a relatively recent contribution to superpixel methods; it can be thought of as a version of $k$-means clustering applied in a feature space that includes both the spectral (color) and spatial features of each pixel. It has parameters that enable control of
Suppose a three-channel image contains a set of \( N \) pixels. Define \( \mathcal{X}_p = \{x^p_1, \ldots, x^p_N\} \) to be the set of \( 2 \times 1 \) vectors containing the spatial coordinates of each pixel, and define \( \mathcal{X}_f = \{x^f_1, \ldots, x^f_N\} \) to be the set of \( 3 \times 1 \) vectors containing the intensities for each channel at each pixel location. Each \( x^p_i \) and \( x^f_i \) can be concatenated to form the set \( \mathcal{X} = \{x_1, \ldots, x_N\} \), where the column vector \( x_i \) is \( 5 \times 1 \). Using this notation, SLIC superpixels can be computed in the following manner:

1. Construct a weighted feature vector \( \Psi(\lambda)(x_i) \) for each of the \( N \) pixels in the image:

\[
\Psi(\lambda)(x_i) = \begin{bmatrix} \lambda x^p_i \\ x^f_i \end{bmatrix},
\]

(3.1)

where \( \lambda \) is a parameter that trades off the impact of spatial and spectral information. The parameter \( \lambda \) can be expressed as the ratio \( r/s \), where a superpixel is nominally assumed to contain \( s \times s \) pixels, and where \( r \) is directly related to superpixel regularity.

2. Construct an initial set of cluster centers \( C_k = \Psi(\lambda)(x_k) \) on a regular grid of step size \( s \). Move each cluster center to the lowest gradient position in the \( 3 \times 3 \) neighborhood. The reason is that some corner cases such as edge or noisy can be avoided. The image gradients can be computed as:

\[
G(i, j) = \|I(i+1, j) - I(i-1, j)\|^2 + \|I(i, j+1) - I(i, j-1)\|^2,
\]

(3.2)

where \( \| \cdot \| \) is the \( L_2 \) norm and \( I(i, j) \) represents \( x^f_i \) in the position of \( (i, j) \). Spectral information is used to find the lowest gradient for each cluster center.
3. Assign each pixel $x_i$ to the closest cluster center according to the euclidean distance.

To accelerate the algorithm, this step can be simplified by only searching cluster centers within a $2s \times 2s$ neighborhood.

4. Update each cluster center based on the centroid of pixels with which it has been identified.

5. Repeat until the distance between successive cluster center updates is below a predetermined threshold.

6. Relabel disjoint segments to be connected to the largest neighboring cluster.

To illustrate SLIC superpixels visually, consider three publicly available hyperspectral images: Indian Pines, Salinas and University of Pavia (Pavia). Indian Pines and Salinas were captured by AVIRIS spectrometers with 224 spectral bands and Pavia was captured by ROSIS sensor with 103 spectral bands. The images have been partially labeled; there are 16 classes in the reference data for Indian Pines and Salinas and 9 classes in the reference data for Pavia. Figures 3.1–3.3 show the pseudo color images, reference data and superpixel results for the three images for different choices of size ($s$) and regularity ($r$). SLIC superpixels were computed using all of the available spectral bands, even though the figures show only three bands. As $s$ increases, the size of the superpixels increases, and as $r$ increases, the superpixels become more regular.

### 3.3 Applications of Superpixels in HSI

Since their introduction for use in RGB image processing, superpixels have been extended for use in a variety of multispectral and hyperspectral image processing applications. Compared with RGB images, multispectral and hyperspectral images have more bands, which increases the required computational effort for applications such as classification and target detection. Hence, extending superpixel algorithms to HSI could provide significant computational sav-
Figure 3.1: Left: Indian Pines image (spectral bands 29, 15, 12) and manually labeled reference data (16 classes). Right: SLIC superpixels for various choices of $s$ (superpixel size) and $r$ (superpixel regularity).
Figure 3.2: Left: Salinas image (spectral bands 29, 15, 12) and manually labeled reference data (16 classes). Right: SLIC superpixels for various choices of $s$ (superpixel size) and $r$ (superpixel regularity).
Figure 3.3: Left: Pavia image (spectral bands 48, 17, 6) and manually labeled reference data (9 classes). Right: SLIC superpixels for various choices of $s$ (superpixel size) and $r$ (superpixel regularity).
ings for subsequent algorithms. However, not all superpixel algorithms are easily generalizable from RGB imagery to HSI, and very few of the superpixel algorithms have publicly available implementations.

A few superpixel methods have been directly modified to generalize to HSI. For example, Jordan et al. [111] modify the meanshift algorithm [108] to handle spectral gradient. Given the superpixels, Spectral Angle Mapper (SAM) and Spectral Information Divergence (SID) [112] [113] are used to discriminate different materials. Saranathan et al. [114] generalizes Felzenswalb’s algorithm [104], yielding an agglomerative method in which a threshold is set to manage the maximum variability inside segments for segment uniformity.

The entropy rate superpixel algorithm [106] has a publicly available implementation, and a number of papers have used it for HSI analysis. This has not been done, however, by directly generalizing the algorithm to handle a large number of spectral bands. Rather, PCA is used to decrease the dimension of HSI data to 1 or 3, and then the resulting low-dimensional embedding is directly input to greyscale or RGB-based entropy rate superpixel construction. Some examples of this strategy include [115, 116, 117, 118, 119, 120]. Even though this strategy for constructing superpixels from HSI appears to be popular, it ignores any information that is not contained within the first three principal components, and this information could be vital in identifying good data representations.

SLIC generalizes easily to HSI, since the vectors in $X_f$ can be of any dimension. A number of papers have used SLIC for HSI analysis applications. Vargas et al. [121] use SLIC to make an active learning process feasible for very high resolution imagery. Roscher et al. [122] use SLIC to build a superpixel-based classifier for landcover mapping: superpixel incorporate neighborhood information and are input into a hierarchical conditional random field classifier. Liu et al. [123] provides superpixels as input to a stacked denoising autoencoder (SDA) deep network for classification. Sun et al. [124] use SLIC to group pixels into perceptually meaningful atomic regions and then perform active learning to find the most informative samples to increase the
number of the training data. SVM is used as the final classifier to evaluate the result. Cui et al. [125] adapt SLIC to generate homogeneous small regions efficiently so that the background in each superpixel can satisfy the condition of low-rank. Caliskan et al. [126] illustrate that spectral angle mapping, spectral information divergence and spatial coherence distance are more suitable to HSI processing when SLIC superpixels are provided as input. Liang et al. [127] develop a SLIC-based matched-filter detector to find subpixel targets. Psalta et al. [128] use Fractional Distance (FD) in conjunction with SLIC superpixels to perform HSI unmixing. Cordeiro [129] performs denoising by using SLIC to segment images into small regions with approximately constant intensity. Wang [130] uses SLIC as a preprocessing step for a blind source separation algorithm for HSI unmixing.

Even though SLIC superpixels have been used in a variety of HSI applications, most papers do little to assess the tradeoff between the improved computational efficiency and the potential loss in accuracy when using superpixels. Since SLIC superpixel construction has hyperparameters $s$ and $r$ that control the size and regularity of superpixels, it is possible that poor choices of the hyperparameters could cause a significant loss in accuracy of whatever algorithm they are provided to as input. Furthermore, in many HSI applications, it would be useful to propagate/interpolate results from superpixel-based analysis schemes back to the original pixel grid; however, this is not typically done in the literature. In this research, we will address these issues in the context of nonlinear DR for pixel-level HSI classification by determining optimal ranges of hyperparameters for superpixel construction, and by interpolating the results of nonlinear DR back to a pixel grid so that approximate embedding coordinates for each pixel can be provided for subsequent analysis.

### 3.4 Proposed HSI Classification Framework

Figure 3.4 shows the proposed system for pixel-wise HSI classification. Given a hyperspectral image, SLIC superpixels are computed as described in Section 3.2. Next, a nonlinear DR
method such as SSSE or KPCA is used to compute embeddings from the average spectra in each superpixel, as described in Section 2.4. Then, the low-dimensional embeddings at each superpixel are spatially interpolated to approximate the values of the low-dimensional embedding at each pixel. The interpolation is performed by triangulating the superpixel centroids and linearly interpolating each spectral band in each triangle. Finally, a standard machine learning classifier is applied to the pixel-wise low-dimensional representations to predict class labels.

![Diagram](image.png)

Figure 3.4: Pipeline of proposed method for pixel-wise HSI classification.

## 3.5 Experiments

In the section, we test the proposed pixel-wise HSI classification system on three publicly available HSI data sets with expert-labeled reference data classes: Indian Pines [131], Salinas [132] and Pavia (University of Pavia) [133]. We perform various types of experiments in order to assess the impact of using SLIC superpixels on the computational burden of the nonlinear DR step and the accuracy of the final pixel-wise classification step.

### 3.5.1 Data sets

Indian Pines image was collected by an AVIRIS sensor and shown in Figure 3.1 with its reference data map. It contains 224 bands in the wavelength range from 400 to 2500 nm. Twenty-four of the bands were removed because of water absorption. The image contains 145 × 145 pixels, and there are 16 classes in the reference data. Roughly two-thirds of the scene contains agriculture classes such as corn and wheat, and the other one-third contains forest or other vegetation such
as grass-trees and grass-pasture. The Salinas image was also gathered from an AVIRIS sensor and shown in Figure 3.2 with its reference data map. It contains the same 224 total bands as Indiana Pines, and has had 20 bands removed due to water absorption. The Salinas image is $512 \times 217$ pixels, which is significantly larger than Indian Pines. It also contains 16 classes in the reference data, including vineyard fields, bare soils, and vegetables. The Pavia image is acquired by a ROSIS sensor during a flight campaign over Pavia, Italy and shown in Figure 3.3 with its reference data map. It contains 103 bands, has $610 \times 610$ pixels, and has nine reference data classes. In order to avoid memory issues when running nonlinear DR on the original pixel grid, we use only a $441 \times 200$ portion extracted from the bottom-left of the Pavia image, selected so that there is significant heterogeneity in class labels. Table 3.1 provides information about the reference data classes for each image.

### Table 3.1: Names and the number of samples for each reference data class from each image.

<table>
<thead>
<tr>
<th>Class</th>
<th>Indian Pines Classes</th>
<th>Number of Samples</th>
<th>Salinas Classes</th>
<th>Number of Samples</th>
<th>Pavia Classes</th>
<th>Number of Samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Alfalfa</td>
<td>46</td>
<td>broccoli_green_weeds_1</td>
<td>2001</td>
<td>Aspalt</td>
<td>2644</td>
</tr>
<tr>
<td>2</td>
<td>Corn-notill</td>
<td>838</td>
<td>Fallow</td>
<td>1976</td>
<td>Gravel</td>
<td>2699</td>
</tr>
<tr>
<td>3</td>
<td>Corn-mintill</td>
<td>287</td>
<td>Fallow_rough_grass</td>
<td>1194</td>
<td>Trees</td>
<td>1891</td>
</tr>
<tr>
<td>4</td>
<td>Grass-pasture</td>
<td>483</td>
<td>Fallow_smooth</td>
<td>2678</td>
<td>Painted metal sheats</td>
<td>1423</td>
</tr>
<tr>
<td>5</td>
<td>Grass-trees</td>
<td>739</td>
<td>Strubble</td>
<td>3959</td>
<td>Bare Soil</td>
<td>8042</td>
</tr>
<tr>
<td>6</td>
<td>Grass-pasture-mowed</td>
<td>28</td>
<td>Celery</td>
<td>3979</td>
<td>Bitumen</td>
<td>1488</td>
</tr>
<tr>
<td>7</td>
<td>Hay-windrowed</td>
<td>478</td>
<td>Grapes_untrained</td>
<td>11273</td>
<td>Self-Blocking Bricks</td>
<td>3848</td>
</tr>
<tr>
<td>8</td>
<td>Oats</td>
<td>20</td>
<td>Soil_vinyard_develop</td>
<td>6203</td>
<td>Shadows</td>
<td>855</td>
</tr>
<tr>
<td>9</td>
<td>Soybean-notill</td>
<td>972</td>
<td>Corn_grass_2</td>
<td>3278</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Soybean-mintill</td>
<td>2455</td>
<td>Lettuce_romanze_4wk</td>
<td>1068</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Soybean-clean</td>
<td>483</td>
<td>Lettuce_romanze_5wk</td>
<td>1927</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Wheat</td>
<td>205</td>
<td>Lettuce_romanze_6wk</td>
<td>916</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Woods</td>
<td>1205</td>
<td>Lettuce_romanze_7wk</td>
<td>1070</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Buildings-Grass-Trees-Drives</td>
<td>396</td>
<td>Vinyard_untrained</td>
<td>7268</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Stone-Steel-Towers</td>
<td>93</td>
<td>Vinyard_vertical_trellis</td>
<td>1407</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 3.5.2 Experimental setup

For each of the three hyperspectral images (HSIs), 10% of the data are randomly chosen as training data for the classifier, and the remaining 90% are testing data. For the parameters $s$ and $r$ that control the size and regularity of SLIC superpixels, we sample $s$ linearly from 4 to 18 for all of HSIs, and we sample $r$ logarithmically from $10^{-1}$ to $10^1$ for Indiana Pines and Salinas and from $10^0$ to $10^{1.5}$ for Pavia. We choose a different range of $r$ for Pavia because the SLIC
Table 3.2: Definitions of per-class classification performance measures, in terms of true positives (TP), false positives (FP), false negatives (FN), and true negatives (TN).

<table>
<thead>
<tr>
<th>Measure</th>
<th>Definition</th>
<th>Measure</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>$\frac{TP + TN}{TP + FP + FN + TN}$</td>
<td>Sensitivity</td>
<td>$\frac{TP}{TP + FN}$</td>
</tr>
<tr>
<td>Precision</td>
<td>$\frac{TP}{TP + FP}$</td>
<td>Specificity</td>
<td>$\frac{TN}{FP + TN}$</td>
</tr>
</tbody>
</table>

algorithm appears to be unstable for small $r$ for that image.

For dimensionality reduction and classification, the same parameters are used for all three HSIs. For SSSE, 20 nearest neighbors are used to construct each graph. The $\alpha$ parameter is set to 10 and the number of dimensions after SSSE is 50. For KPCA, we use the polynomial kernel and also keep 50 dimensions after DR. For classification, we use a random forest (RF) with 100 trees. We implement KPCA and RF with the sklearn package \[134\] and use default value for all other parameters.

We validate classification performance according to a variety of measures, including overall accuracy (OA), average accuracy (AA), Kappa coefficient ($\kappa$) and computing time for the whole dataset and precision (Pr) and sensitivity(Se) for each class. OA is defined as the ratio of the number of correctly predicted test pixels to the total number of test pixels. AA is the average of the per-class accuracy. $\kappa$ measures how the classification result is better than the result in which class labels are randomly assigned to the pixels. For the random result, $\kappa$ equals 0. When all of numbers are in the diagonal of the confusion matrix, $\kappa$ equals 1. Pr and Se are computed from each class confusion matrix as shown in Table 3.2. For the computing time, it is the total time for normalizing the data, calculating SLIC superpixels, performing nonlinear DR, interpolating, training the classifier, and classifying the testing data. We also run experiments using original pixels (without the superpixel calculation step) for comparison.
3.5.3 Results

As a baseline, Table 3.3 shows various classification performance measures when nonlinear DR is performed using the original pixels as input (i.e., without intermediate superpixel construction). Figure 3.5 shows the corresponding class maps for each image. There are a few interesting points to note. First, for the KPCA method, the classification precision of the ninth class in Indiana Pines is just 11.11%, indicating that many samples from other classes are misclassified into this class. From Table 3.1, we can see that class nine has only 20 samples, which means that only two training samples were used for that class. Second, for the Salinas image, both SSSE and KPCA generate very high accuracies; for Indiana Pines and Pavia, SSSE yields very high accuracy, but KPCA yields only 87.61% and 95.36% OA. Third, KPCA is much slower than SSSE, and especially for the Salinas image, it took 432173 seconds (more than 5 days).

Classification results for systems that include a SLIC superpixel construction step are shown in the following figures. Figures 3.6–3.8 show overall accuracy (OA) and computation time for classification of each image assuming that SLIC superpixels are constructed based on different choices of size parameter $s$. The first row of each figure shows OA for SSSE- and KPCA-based nonlinear DR. The $x$-axes show different values of $s$ that range from 4 to 18, and the plots illustrate five different curves corresponding to a range of values of regularity parameter $r$.

For Indian Pines, When $s$ is small and SSSE is performed, the OA is relatively low. When $s \in [12, 17]$, OA is high: approximately 97%, which is very close to the OA achieved using the original pixels as the inputs to nonlinear DR. When KPCA is performed, OA is also close to 97%, which is much higher than the corresponding pixel-based KPCA result. The second row of Figure 3.6 shows the computing time required for SSSE and KPCA as a function of superpixel size. We can see that all of the curves exhibit similar trends. When $s$ is small, the time is relatively large. As $s$ increases, time decreases quickly and stabilizes for $s > 10$. In the range where $s \in [12, 17]$ where OA is high for both SSSE and KPCA, both methods run in around 10s. This is a $10 \times$ computational improvement over SSSE based on original image pixels, and
Figure 3.5: Reference data and class maps from classification of Indian Pines, Salinas, and Pavia imagery using SSSE and KPCA embeddings computed from original image pixels.
Table 3.3: Baseline classification results for Indian Pines, Salinas, and Pavia, without using superpixels. Performance measures include precision (Pr), sensitivity (Se), overall accuracy (OA), average accuracy (AA), kappa coefficient (κ), and computing time.

<table>
<thead>
<tr>
<th>class</th>
<th>Indian SSSE</th>
<th>Indian KPCA</th>
<th>Salinas SSSE</th>
<th>Salinas KPCA</th>
<th>Pavia SSSE</th>
<th>Pavia KPCA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Pr</td>
<td>Se</td>
<td>Pr</td>
<td>Se</td>
<td>Pr</td>
<td>Se</td>
</tr>
<tr>
<td>1</td>
<td>97.56</td>
<td>100</td>
<td>84.37</td>
<td>100</td>
<td>97.79</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>92.30</td>
<td>98.18</td>
<td>84.90</td>
<td>75.35</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>96.52</td>
<td>97.43</td>
<td>79.79</td>
<td>90.03</td>
<td>100</td>
<td>99.94</td>
</tr>
<tr>
<td>4</td>
<td>99.53</td>
<td>99.53</td>
<td>40.38</td>
<td>79.63</td>
<td>99.92</td>
<td>99.37</td>
</tr>
<tr>
<td>5</td>
<td>98.85</td>
<td>99.54</td>
<td>91.72</td>
<td>95.45</td>
<td>99.63</td>
<td>99.96</td>
</tr>
<tr>
<td>6</td>
<td>99.24</td>
<td>100</td>
<td>99.09</td>
<td>88.57</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>7</td>
<td>92.00</td>
<td>100</td>
<td>72.00</td>
<td>94.74</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>8</td>
<td>100</td>
<td>100</td>
<td>99.53</td>
<td>97.94</td>
<td>94.13</td>
<td>93.91</td>
</tr>
<tr>
<td>9</td>
<td>83.33</td>
<td>100</td>
<td>11.11</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>10</td>
<td>95.54</td>
<td>97.66</td>
<td>82.63</td>
<td>82.53</td>
<td>99.76</td>
<td>99.80</td>
</tr>
<tr>
<td>11</td>
<td>99.37</td>
<td>93.97</td>
<td>91.72</td>
<td>86.62</td>
<td>99.58</td>
<td>100</td>
</tr>
<tr>
<td>12</td>
<td>93.82</td>
<td>97.66</td>
<td>68.35</td>
<td>87.32</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>13</td>
<td>99.46</td>
<td>98.92</td>
<td>96.76</td>
<td>98.35</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>14</td>
<td>99.82</td>
<td>99.56</td>
<td>98.51</td>
<td>96.81</td>
<td>99.90</td>
<td>100</td>
</tr>
<tr>
<td>15</td>
<td>99.71</td>
<td>98.86</td>
<td>83.29</td>
<td>92.04</td>
<td>90.55</td>
<td>90.90</td>
</tr>
<tr>
<td>16</td>
<td>100</td>
<td>100</td>
<td>95.45</td>
<td>84.52</td>
<td>98.61</td>
<td>100</td>
</tr>
<tr>
<td>OA</td>
<td>97.49</td>
<td>87.61</td>
<td>97.46</td>
<td>98.21</td>
<td>99.27</td>
<td>95.36</td>
</tr>
<tr>
<td>AA</td>
<td>99.69</td>
<td>98.45</td>
<td>99.68</td>
<td>99.78</td>
<td>99.84</td>
<td>98.97</td>
</tr>
<tr>
<td>κ</td>
<td>0.97</td>
<td>0.86</td>
<td>0.97</td>
<td>0.98</td>
<td>0.99</td>
<td>0.95</td>
</tr>
<tr>
<td>Time(s)</td>
<td>106</td>
<td>23316</td>
<td>1716</td>
<td>432173</td>
<td>602</td>
<td>1297</td>
</tr>
</tbody>
</table>

a 2000× improvement over KPCA on original image pixels.

Another important observation is that when \( s \in [12, 18] \), it is apparent from Figure 3.1 that each superpixel contains pixels from different classes; however, as illustrated in Figure 3.6, this does not reduce overall classification accuracy. A likely reason for this is that the interpolation of the embedding computed over the superpixels to generate embedding values at each original pixel location enables preservation of features that enable classification into distinct classes. As we will see in Figures 3.7–3.8, similar behavior is seen for larger values of \( s \) for the Salinas and Pavia images, as well.

Figures 3.7 and 3.8 show overall accuracy and computation time for Salinas and Pavia using SLIC superpixels for different choices of \( s \). For the SSSE dimensionality reduction, the results
are similar to those for the Indian Pines image. The overall accuracy is relatively high for \( s \in [12, 17] \). For Salinas, use of superpixels in SSSE results in higher OA than using original image pixels (99.41% \((s = 16)\) vs 97.46%), but it yields lower OA for Pavia (97.19% \((s = 16)\) vs 99.27%). For KPCA-based DR, for both Salinas and Pavia, the use of superpixels yields OA that is stable with respect to \( s \) and is higher than the OA achieved by using original image pixels (99.83% \((s = 16)\) vs 98.21% for Salinas and 98.31% \((s = 14)\) vs 95.36% for Pavia). For the computing time, superpixel-based SSSE is 30\( \times \) faster than original pixel-based SSSE for
Salinas and \(20\times\) faster for Pavia; for KPCA, these figures jump to \(70000\times\) faster for SSSE and \(40\times\) faster for Pavia. The confusion matrices for all of the three images with superpixel-based SSSE and superpixel-based KPCA methods are shown from Table 1 to 6.

So for all three images, overall classification accuracy of superpixel-based DR methods for \(s \in [12, 17]\) is similar or better than the accuracy of the corresponding DR method when all original pixels are used as input, and computational speed can increase from one to almost five orders of magnitude. Even when \(s\) is large enough so that pixels from multiple classes are included in a single superpixel, the interpolation of the superpixel embeddings back to the original pixel grid yields features that preserve high classification accuracy.

In order to explore the effect of regularity \((r)\), Figures 3.9–3.11 show overall classification accuracy and computation time for the three images for SLIC superpixel constructions of varying regularity \(r\) and this time different curves correspond to different sizes \((s)\). In each figure, the first row shows the OA curves for various sizes \(s\), and the second row shows the computation times. The left columns show results for SSSE-based DR, and the right columns for KPCA. The x-axes show \(r\) ranging from 0.1 to 10 for Indiana Pines and Salinas, and from 1 to \(10^{1.5}\) for Pavia. Five different curves corresponding to a range of sizes \(s\) are shown in each figure. We can see that for fixed sizes, both OA and computation time are almost the same over the entire range of regularity values. We did experiment with \(r\) in the range of \(10^{-3}\), however SLIC superpixel construction was unstable for this range because the pixels in one superpixel could not be four-connected.

Figures 3.12–3.17 illustrate classification maps that result for the various images when performing nonlinear DR using SLIC superpixel representations as input. For the classification maps we notice two interesting things. The first is that when \(s\) is small (as when no superpixels are used; see Figures 3.5), the classification results are spatially noisy. As \(s\) increases, spatial noise is regularized quickly, even though some errors in the edges of regions are still apparent. The second item to note is that for Indiana Pines and Salinas, the reference maps for each
Figure 3.7: Overall classification accuracy (top) and computation time (bottom) as superpixel size \(s\) varies for SLIC superpixel-based nonlinear DR via SSSE (left) and KPCA (right) on Salinas.
Figure 3.8: Overall classification accuracy (top) and computation time (bottom) as superpixel size ($s$) varies for SLIC superpixel-based nonlinear DR via SSSE (left) and KPCA (right) on Pavia.
Figure 3.9: Overall classification accuracy (top) and computation time (bottom) as superpixel regularity ($r$) varies for SLIC superpixel-based nonlinear DR via SSSE (left) and KPCA (right) on Indian Pines.
Figure 3.10: Overall classification accuracy (top) and computation time (bottom) as superpixel regularity ($r$) varies for SLIC superpixel-based nonlinear DR via SSSE (left) and KPCA (right) on Salinas.
Figure 3.11: Overall classification accuracy (top) and computation time (bottom) as superpixel regularity (r) varies for SLIC superpixel-based nonlinear DR via SSSE (left) and KPCA (right) on Pavia.
class have regular shape and are well-separated, making it difficult to assess whether linear interpolation has any advantages. In these images, most of the pixels comprising a superpixel come from the same class. However, the Pavia image is more complex, and many superpixels contain pixels from a number of different classes. In this image, the classification results are fairly accurate, demonstrating the effectiveness of interpolation.

Figure 3.12: Left: Indian Pines reference data; right: Classification maps based on SSSE embeddings computed with SLIC superpixels for various choices of $s$ (size) and $r$ (regularity).

In the previous experiments on Indian Pines, Salinas and Pavia, when $s$ and $r$ are in the reasonable range (for example $s$ equals 15 and $r$ equals 1), the classification result is relatively good. The mixture of pixels from different classes in a superpixel doesn’t appear to cause classification result to significantly deteriorate. For the new image dataset, if there is reference
Figure 3.13: Left: Indian Pines reference data; right: Classification maps based on KPCA embeddings computed with SLIC superpixels for various choices of $s$ (size) and $r$ (regularity).
Figure 3.14: Left: Salinas reference data; right: Classification maps based on SSSE embeddings computed with SLIC superpixels for various choices of $s$ (size) and $r$ (regularity).
Figure 3.15: Left: Salinas reference data; right: Classification maps based on KPCA embeddings computed with SLIC superpixels for various choices of $s$ (size) and $r$ (regularity).
Figure 3.16: Left: Pavia reference data; right: Classification maps based on SSSE embeddings computed with SLIC superpixels for various choices of $s$ (size) and $r$ (regularity).
Figure 3.17: Left: Pavia reference data; right: Classification maps based on KPCA embeddings computed with SLIC superpixels for various choices of $s$ (size) and $r$ (regularity).
data for the training set, the cross validation can be used to find the best $s$ and $r$ based on a grid of values. If there is no reference data, we can firstly guess the size of small objects in the image and initialize the $s$ around that. When we got the classification map, we can add that on the top of the superpixel segmentation result to visually check which result looks like most reasonable. Based on the best result with suitable $s$, we can finetune the regularity $r$ in the similar way to find the final best combination of $s$ and $r$ for the new image.

Except for exploiting the effect of different size $s$ and regularity $r$, we also compare the accuracy and computing time for our proposed method with those of recent state-of-the-art nonlinear methods that are kernel-based, graph-based, and/or CNN (deep learning)-based. A direct comparison is not necessarily straightforward, because many of these methods use different numbers of training samples, and they use implementations on different CPU/GPUs. Given these differences, we list overall classification accuracy, computing time, and CPU/GPU reported on Indian Pines, Salinas, and Pavia from various papers in Tables 3.4–3.6. In each table, the last four rows show the results from the pixel-based and superpixel-based methods described in this paper.

In Table 3.4, we see that the overall classification accuracy of our superpixel-based method is slightly lower than Kemker et al. [135], Li et al. [136], and Li et al. [137] but is higher than the other methods. The methods described in Kemker et al. [135] and Li et al. [136] are CNN-based methods that require a long training time. The method described in Li et al. [137] is a kernel-based method that does not report the computing time. Our superpixel-based methods require only 8 seconds to complete all steps and yield a competitively high overall accuracy on Indian Pines. We see similar situations in Tables 3.5 – 3.6 for the Salinas and Pavia images, demonstrating the effectiveness of our proposed method with respect to other state-of-the-art methods.
Table 3.4: Comparison with state-of-the-art classification methods on Indian Pines

<table>
<thead>
<tr>
<th>Citation</th>
<th>Method</th>
<th>Training Size</th>
<th>CPU/GPU</th>
<th>OA</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Camps-Valls et al.</td>
<td>Kernel-Based method</td>
<td>51%</td>
<td>GPU</td>
<td>94.44</td>
<td></td>
</tr>
<tr>
<td>Chen et al.</td>
<td>LLE</td>
<td>10%</td>
<td>GPU</td>
<td>83.13</td>
<td>5.2min</td>
</tr>
<tr>
<td>Chen et al.</td>
<td>CNN</td>
<td>90%</td>
<td>GPU</td>
<td>87.81</td>
<td>27min</td>
</tr>
<tr>
<td>Faurel et al. [140]</td>
<td>Spectral Spatial</td>
<td>7.46%</td>
<td>AMD FX-6300</td>
<td>90.83</td>
<td></td>
</tr>
<tr>
<td>He et al. [141]</td>
<td>Laplacian-Like Regularization</td>
<td>1%</td>
<td>GeForce GTX 465</td>
<td>90.16</td>
<td>4300s</td>
</tr>
<tr>
<td>Hu et al. [142]</td>
<td>CNN</td>
<td>4.2%</td>
<td>GeForce GTX 465</td>
<td>90.16</td>
<td>4300s</td>
</tr>
<tr>
<td>Jia et al. [143]</td>
<td>Gabor collaborative representation</td>
<td>1%</td>
<td>Intel 4 Xeon</td>
<td>81.2</td>
<td>200s</td>
</tr>
<tr>
<td>Kemker et al. [135]</td>
<td>CNN</td>
<td>10%</td>
<td>GPU QUADRO K2200</td>
<td>94.34</td>
<td>6h</td>
</tr>
<tr>
<td>Li et al. [136]</td>
<td>CNN</td>
<td>19.5%</td>
<td>GeForce GTX 465</td>
<td>92.60</td>
<td>70min</td>
</tr>
<tr>
<td>Li et al. [137]</td>
<td>Generalized Composite Kernel</td>
<td>27.8%</td>
<td>Intel Core i5-3470</td>
<td>93.87</td>
<td>9.19s</td>
</tr>
<tr>
<td>Li et al. [145]</td>
<td>Multiple Feature Learning</td>
<td>4.9%</td>
<td>GeForce GTX 465</td>
<td>94.59</td>
<td>23.64s</td>
</tr>
<tr>
<td>Li et al. [146]</td>
<td>Nearest-subspace</td>
<td>17.3%</td>
<td>GeForce GTX 680</td>
<td>99.75</td>
<td>220s</td>
</tr>
<tr>
<td>Li et al. [137]</td>
<td>Kernel w/ Tikhonov Regularization</td>
<td>10%</td>
<td>GeForce GTX 680</td>
<td>98.22</td>
<td></td>
</tr>
<tr>
<td>Wang et al. [147]</td>
<td>Manifold Ranking</td>
<td>78</td>
<td>AMD FX-6300</td>
<td>97.46</td>
<td>106s</td>
</tr>
<tr>
<td>Our paper</td>
<td>SSSE</td>
<td>10%</td>
<td>AMD FX-6300</td>
<td>97.46</td>
<td>106s</td>
</tr>
<tr>
<td>Our paper</td>
<td>KPCA</td>
<td>10%</td>
<td>AMD FX-6300</td>
<td>97.46</td>
<td>106s</td>
</tr>
<tr>
<td>Our paper</td>
<td>Superpixel-based SSSE</td>
<td>10%</td>
<td>AMD FX-6300</td>
<td>97.46</td>
<td>106s</td>
</tr>
<tr>
<td>Our paper</td>
<td>Superpixel-based KPCA</td>
<td>10%</td>
<td>AMD FX-6300</td>
<td>97.46</td>
<td>106s</td>
</tr>
</tbody>
</table>

Table 3.5: Comparison with state-of-the-art classification methods on Salinas

<table>
<thead>
<tr>
<th>Citation</th>
<th>Method</th>
<th>Training Size</th>
<th>CPU/GPU</th>
<th>OA</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>He et al. [141]</td>
<td>Laplacian-Like Regularization</td>
<td>1%</td>
<td>GeForce GTX 465</td>
<td>96.64</td>
<td></td>
</tr>
<tr>
<td>Hu et al. [142]</td>
<td>CNN</td>
<td>5.9%</td>
<td>GeForce GTX 465</td>
<td>92.60</td>
<td>70min</td>
</tr>
<tr>
<td>Kemker et al. [135]</td>
<td>CNN</td>
<td>5%</td>
<td>GeForce GTX 465</td>
<td>99.75</td>
<td></td>
</tr>
<tr>
<td>Li et al. [144]</td>
<td>CNN</td>
<td>5.9%</td>
<td>GPU QUADRO K2200</td>
<td>94.80</td>
<td>12h</td>
</tr>
<tr>
<td>Li et al. [136]</td>
<td>Deep Belief Networks</td>
<td>8.9%</td>
<td>Intel Core i5-3470</td>
<td>96.22</td>
<td></td>
</tr>
<tr>
<td>Plaza et al. [148]</td>
<td>NN</td>
<td>4%</td>
<td>Intel 4 Xeon</td>
<td>95.08</td>
<td></td>
</tr>
<tr>
<td>Quesada-Barriuso et al. [149]</td>
<td>Spectral-Spatial</td>
<td>10%</td>
<td>GeForce GTX 680</td>
<td>94.37</td>
<td>58.8s</td>
</tr>
<tr>
<td>Wang et al. [147]</td>
<td>Manifold Ranking</td>
<td>89</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wang et al. [150]</td>
<td>NN</td>
<td>30%</td>
<td></td>
<td>99.64</td>
<td></td>
</tr>
<tr>
<td>Our paper</td>
<td>SSSE</td>
<td>10%</td>
<td>AMD FX-6300</td>
<td>97.46</td>
<td>29min</td>
</tr>
<tr>
<td>Our paper</td>
<td>KPCA</td>
<td>10%</td>
<td>AMD FX-6300</td>
<td>98.21</td>
<td>120h</td>
</tr>
<tr>
<td>Our paper</td>
<td>Superpixel-based SSSE</td>
<td>10%</td>
<td>AMD FX-6300</td>
<td>99.47</td>
<td>68.13s</td>
</tr>
<tr>
<td>Our paper</td>
<td>Superpixel-based KPCA</td>
<td>10%</td>
<td>AMD FX-6300</td>
<td>99.86</td>
<td>83.06s</td>
</tr>
</tbody>
</table>
Table 3.6: Comparison with state-of-the-art classification methods on Pavia

<table>
<thead>
<tr>
<th>Citation</th>
<th>Method</th>
<th>Training Size</th>
<th>CPU/GPU</th>
<th>OA</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chen et al. [139]</td>
<td>LLE</td>
<td>10%</td>
<td>GPU</td>
<td>85.46</td>
<td>6.23min</td>
</tr>
<tr>
<td>Chen et al. [139]</td>
<td>CNN</td>
<td>90%</td>
<td>GPU</td>
<td>92.28</td>
<td>45min</td>
</tr>
<tr>
<td>He et al. [141]</td>
<td>Laplacian-Like Regularization</td>
<td>9.2%</td>
<td></td>
<td>78.62</td>
<td></td>
</tr>
<tr>
<td>Hu et al. [142]</td>
<td>CNN</td>
<td>4.2%</td>
<td>GeForce GTX 465</td>
<td>92.56</td>
<td>9min</td>
</tr>
<tr>
<td>Kemker et al. [135]</td>
<td>CNN</td>
<td>10%</td>
<td></td>
<td>99.88</td>
<td></td>
</tr>
<tr>
<td>Li et al. [144]</td>
<td>CNN</td>
<td>4.2%</td>
<td>GPU QUADRO K2000</td>
<td>94.80</td>
<td>4h</td>
</tr>
<tr>
<td>Li et al. [136]</td>
<td>Deep Belief Networks</td>
<td>6.3%</td>
<td>Intel Core 15-4760</td>
<td>98.96</td>
<td></td>
</tr>
<tr>
<td>Li et al. [145]</td>
<td>Generalized Composite Kernel</td>
<td>8.4%</td>
<td></td>
<td>98.05</td>
<td>94.8s</td>
</tr>
<tr>
<td>Li et al. [145]</td>
<td>Multiple Feature Learning</td>
<td>8.4%</td>
<td></td>
<td>97.80</td>
<td>2082s</td>
</tr>
<tr>
<td>Li et al. [146]</td>
<td>nearest-subspace</td>
<td>16.6%</td>
<td>3.2-GHz machine</td>
<td>92</td>
<td></td>
</tr>
<tr>
<td>Li et al. [137]</td>
<td>Kernel with Tikhonov Regularization</td>
<td>1.3%</td>
<td></td>
<td>94.81</td>
<td></td>
</tr>
<tr>
<td>Quesada-Barriuso et al. [149]</td>
<td>Spectral-Spatial</td>
<td>7.5%</td>
<td>GeForce GTX680</td>
<td>94.63</td>
<td>17.3s</td>
</tr>
<tr>
<td>Wang et al. [147]</td>
<td>Manifold Ranking</td>
<td></td>
<td></td>
<td>87</td>
<td></td>
</tr>
<tr>
<td>Wang et al. [150]</td>
<td>NN</td>
<td>30%</td>
<td></td>
<td>99.51</td>
<td></td>
</tr>
<tr>
<td>Our paper</td>
<td>SSSE</td>
<td>10%</td>
<td>AMD FX-6300</td>
<td>99.27</td>
<td>602s</td>
</tr>
<tr>
<td>Our paper</td>
<td>KPCA</td>
<td>10%</td>
<td>AMD FX-6300</td>
<td>95.30</td>
<td>1297s</td>
</tr>
<tr>
<td>Our paper</td>
<td>Superpixel-based SSSE</td>
<td>10%</td>
<td>AMD FX-6300</td>
<td>97.49</td>
<td>32.22s</td>
</tr>
<tr>
<td>Our paper</td>
<td>Superpixel-based KPCA</td>
<td>10%</td>
<td>AMD FX-6300</td>
<td>98.22</td>
<td>35.58s</td>
</tr>
</tbody>
</table>

3.6 Conclusion

In this chapter, we proposed using superpixel representations for performing nonlinear dimensionality reduction of hyperspectral imagery. By using SLIC superpixels as input to nonlinear DR algorithms, the number of inputs for decreases drastically, which significantly increases the computational efficiency of the nonlinear DR algorithm. Spatial interpolation of the resulting low-dimensional embeddings of the superpixels makes it possible to easily extend the embeddings to each pixel. With experiments on three different hyperspectral images, we showed that using superpixel-based nonlinear DR as a preprocessing step for pixel-wise HSI classification yields results that have comparable accuracy to state-of-the-art methods, but with much faster computing time.
Chapter 4

Semi-Supervised Deep Autoencoder Networks

In the previous chapter, a superpixel-based algorithm was proposed to improve the efficiency of nonlinear DR, and it was shown experimentally to yield good results when used in a pixel-wise HSI classification task. However, there are still some limitations of the superpixel approach.

A first limitation is that the superpixel approach does not easily generalize to new data from previously unseen images. In PCA, for example, it is simple to project new unseen data into computed low-dimensional embeddings. However, it is more difficult to do this with nonlinear DR methods. Bengio et al. [151] has proposed a way to perform out-of-sample extensions for graph-based DR methods such as LLE, LE and ISOMAP, although this technique is not fast and it is difficult to extend to other nonlinear DR methods. In superpixel-based nonlinear DR, it is highly likely that embeddings of the new data would be very different from the embeddings that have been computed using a previous image.

A second limitation of superpixel-based nonlinear DR is that it is only applicable in an image domain. This is because spatial information must be available in order to compute the superpixels. It would not be meaningful to extend this idea to other different types of data such
A third limitation is that superpixel-based nonlinear DR does not make it possible to recover or reconstruct a good approximation of the original data. In DR methods, sometimes we not only care about the low dimensional embedding itself, but we also want to be able to recover the initial input, for example, in tasks such as compression. In PCA, theory exists that enables one to choose an appropriate number of components to be able to reconstruct data to a particular accuracy. For superpixel-based nonlinear DR, however, reconstruction accuracy is limited by the accuracy of interpolating from superpixels back to the original pixel grid.

In order to address these difficulties, we propose a semi-supervised deep autoencoder network (SSDAN) for generating low-dimensional representations that approximate those of other nonlinear DR methods. Training data can be used to train the SSDAN, and unseen test data can be mapped directly and efficiently by the network into the low dimensional space. A SSDAN with tens of layers can be trained in less than half an hour in a Nvidia GeForce GTX 1070 GPU, and it can evaluate more than thousands of samples/second in the test procedure. Additionally, SSDANs can be trained with different kinds of data, not only data that is represented on a pixel grid. Finally, the decoder portion of the SSDAN can be used to recover/reconstruct the initial input, and so it is potentially useful for compression tasks.

The remainder of this chapter is organized as follows: Section 4.1 provides background and applications of autoencoders, Section 4.2 details our proposed semi-supervised deep autoencoder network (SSDAN), Section 4.3 performs various experiments using SSDANs for pixel-wise HSI classification, Section 4.4 performs experiments using SSDANs for compression, and Section 4.5 provides concluding remarks.

4.1 Autoencoders

The autoencoder was first proposed in the 1980s by Rumelhart et al. [152] in conjunction with a back-propagation algorithm for learning. One of the earliest applications of autoencoders was
for denoising data [153]. Another early paper [154] showed how to interpret the role of the
different parameters in an autoencoder. Since these initial papers, autoencoders have become
popular for identifying/extracting feature representations from a dataset that can be used to
recover or reconstruct the initial data. In this section, we will introduce the basic autoencoder
and some variations, and then we will show applications in data processing, especially for HSI.

4.1.1 Background

The autoencoder (or autoencoder network) is an unsupervised learning method that can be
used for dimensionality reduction. It usually has an input layer, one or more hidden layers
with fewer neurons than the input layer, and an output layer with the same number of neurons
as the input layer. The goal of learning the parameters of an autoencoder is to identify a set
of parameters that cause the output layers to faithfully reconstruct the input data. Adjacent
layers in an autoencoder are fully connected, and there are no connections between non-adjacent
layers. Figure 4.1 illustrates an autoencoder network with 3 hidden layers. As shown in Figure
4.1, autoencoder networks usually have a symmetrical shape, with the encoder network being
the portion of the autoencoder that encodes input data into lower-dimensional features, and
the decoder network being the portion that recovers or reconstructs the input information from
the features produced by the encoder network.

We denote $X = \{x^{(1)}, \ldots, x^{(m)}\}$ to be the initial high-dimensional input data, and $Z = \{z^{(1)}, \ldots, z^{(m)}\}$ to be a low-dimensional extracted feature which is also the output of the encoder
network and can be used to reconstruct $X$ faithfully. In the simplest autoencoder network
(containing only a single hidden layer), the encoder network takes the input $X$ and maps it to $Z$ via the transformation:

$$z^{(i)} = \sigma_1(W_1x^{(i)} + b_1) , \quad i = 1, \ldots, m ,$$

(4.1)
where $W_1$ is a weight matrix and $b_1$ is a bias vector, and where $\sigma_1$ is a nonlinear element-wise activation function. Usual choices of activation functions include the sigmoid function, tanh function, rectified linear unit (ReLU) [155] and exponential linear unit (ELU) [156], given by:

\[
\text{Sigmoid: } f(x) = \frac{1}{1 + e^{-x}} \tag{4.2}
\]

\[
\text{Tanh: } f(x) = \tan^{-1}(x) \tag{4.3}
\]

\[
\text{ReLU: } f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases} \tag{4.4}
\]

\[
\text{ELU: } f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases} \tag{4.5}
\]

After the encoder network generates $\mathcal{Z}$, the decoder network maps $\mathcal{Z}$ back to the recon-
In the decoder network, $\sigma_2, W_2$ and $b_2$ may be different from $\sigma_1, W_1$ and $b_1$. The loss in the autoencoder quantifies how far apart the reconstructed data is from the original data. It is usually defined in terms of mean squared error (MSE):

$$L(X, \tilde{X}) = \frac{1}{m} \sum_{i=1}^{m} \| x(i) - \tilde{x}(i) \|^2 = \frac{1}{m} \sum_{i=1}^{m} \| x(i) - \sigma_2 \left( W_2 \sigma_1 \left( W_1 x(i) + b_1 \right) + b_2 \right) \|^2 .$$

(4.7)

Learning the weights and biases of an autoencoder network involves minimizing the loss; a variety of minimization approaches can be used, but stochastic gradient descent (SGD) is recommended for many types of neural networks.

If activation functions are defined to be identity functions (i.e., $\sigma_1(x) = \sigma_2(x) = x$, the autoencoder network is called a linear autoencoder network. The encoder network in a linear autoencoder network with only one hidden layer yields features identical to those computed by PCA [152]. For autoencoder networks with nonlinear activation functions, training was very difficult to perform robustly prior to 2006. This problem was solved by Hinton et al. [157], who suggested initially training a deep autoencoder network layer-by-layer, and then fine-tuning the weights across the whole network.

Some popular variations of autoencoder networks are used for various applications. In the denoising autoencoder [158], random noise is added to the initial input data to form a “corrupted” data set, and the corrupted data is used to train an autoencoder network. Adding noise to the input data actually makes the autoencoder network more robust in its ability to capture interesting structure in the input distribution.

Another popular variation is the sparse autoencoder [159], in which the hidden units are forced to be sparse during training. This can be accomplished by activating a neuron if its
input value is close to one and deactivating it if it is close to 0. Sparse autoencoders can be trained by constraining neurons to be inactive most of the time. By imposing sparsity, the autoencoder more easily learns useful structures, which is very helpful in pretraining tasks. If $a_j$ represents the activation function of the $j$th neuron, the average activation result of the $j$th neuron for all of the samples can be calculated:

$$ p_j = \frac{1}{m} \sum_{i=1}^{m} (a_j x^{(i)}) . $$

(4.8)

The constraint in the sparse autoencoder is that $p_j = p$, where $p$ is a sparsity parameter which is usually very close to 0 (such as 0.05). For sparse autoencoder networks with just one hidden layer, $p_j$ can be calculated by the mean value of $Z$. Kullback-Leibler (KL) divergence is typically added to the loss function as a penalty:

$$ L(X, \tilde{X}) = \frac{1}{m} \sum_{i=1}^{m} \| x^{(i)} - \tilde{x}^{(i)} \|^2 + \beta \sum_{i=1}^{s} \text{KL}(p \parallel p_j) , $$

(4.9)

where

$$ \text{KL}(p \parallel p_j) = p \log \frac{p}{p_j} + (1 - p) \log \frac{1 - p}{1 - p_j} , $$

(4.10)

$s$ is the number of the neurons in the hidden layer, and $\beta$ is a parameter to control the weight of sparsity term. The KL divergence reaches the minimum value 0 at $p = p_j$ and then diverges. Hence, minimizing KL divergence can cause $p_j$ to be close to $p$.

The contractive autoencoder (CAE) [160] adds a new penalty item to the loss function that calculates the sum of squares of all partial derivatives of the features, resulting in a localized space contraction so that the network will be more robust. The loss function for CAE is given
CHAPTER 4. SEMI-SUPERVISED DEEP AUTOENCODER NETWORKS

by:

\[
\mathcal{L}(\mathcal{X}, \tilde{\mathcal{X}}) = \frac{1}{m} \sum_{i=1}^{m} \| x^{(i)} - \tilde{x}^{(i)} \|^2 + \lambda \| J_f(x) \|^2_F, \tag{4.11}
\]

where

\[
\| J_f(x) \|^2_F = \sum_{ij} \left( \frac{\partial z^{(j)}}{\partial x^{(i)}} \right)^2, \tag{4.12}
\]

and where \( \lambda \) is a parameter to control the weight of penalty term. The experiments show the penalty term (4.12) will lead to a localized space contraction and better capture the local directions of variation which can produce robust features. The penalty term can be added to the loss functions of both original and denoising autoencoder networks, providing a link between deterministic and non-deterministic autoencoders. A network constructed in this manner has the same or better results on a range of datasets as opposed to a standard denoising autoencoder network.

The smooth autoencoder [161] learns robust features by reconstructing samples as weighted combinations of their neighbors. It uses a loss function given by:

\[
\mathcal{L}(\mathcal{X}, \tilde{\mathcal{X}}) = \frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{k} \omega(x^{(i)}, x^{(j)}) \| x^{(j)} - \tilde{x}^{(i)} \|^2 + \beta \sum_{i=1}^{s} \text{KL}(p \| p_j), \tag{4.13}
\]

where \( \omega \) is a weight function (a kernel) used to calculate the distances between different samples, and where \( k \) represents the number of neighbors of the input data. KL divergence is also used for sparsity. By reconstructing samples from their neighbors, the learned features are more consistent across local neighborhoods, enabling the smooth autoencoder to be more robust to variations in the input data. By adding supervised information into a smooth autoencoder, features of the same class can be biased towards being closer to each other, which is very useful when used as a preprocessing step for classification or target detection.
CHAPTER 4. SEMI-SUPERVISED DEEP AUTOENCODER NETWORKS

The variational autoencoder (VAE) [162] uses a variational approach for to learn a latent representation describing a directed probabilistic model. Stochastic Gradient Variational Bayes (SGVB) estimator is used to perform approximate inference of the posterior distribution of the latent representation given the data. The data is assumed to generated by \( p(x \mid y) \) and \( y \) has some prior distribution. \( q_{\phi}(z \mid y) \) represents an estimation for the encoder procedure to the posterior distribution \( q_{\theta}(z \mid y) \) and \( \phi \) and \( \theta \) are the variational parameters and generative parameters. The loss function of VAE is given by:

\[
L(\phi, \theta, x) = -D_{KL}(q_{\phi}(y \mid x) \parallel p_{\theta}(y)) + \mathbb{E}_{q_{\phi}(y \mid x)}(\log p_{\theta}(x \mid y)) ,
\]

where \( D_{KL} \) represents the KL divergence. The first term is the KL divergence of the approximate posterior from the true posterior, and the second term is the variational lower bound on the marginal likelihood. The multivariate symmetric gaussian \( P_{\theta}(z) = \mathcal{N}(0, I) \) is the distribution of the latent variable. Recently, some asymmetric distributions have also been proposed [163].

Finally, the convolutional autoencoder [164] uses the same idea as in convolutional layers of a neural network that have successfully been adopted in image processing and computer vision [165]. Instead of having fully connected layers, the connections are reduced and the same weights are shared across neurons in the same layer. This significantly reduces the number of parameters that must be learned by the autoencoder network, and it generates representations that are translation invariant. Equation (4.15) shows the latent representation for the \( k \)-th feature map for input \( x^{(i)} \) (it is a mono-channel input) where \( * \) denotes the 2D convolution. The reconstruction is obtained via (4.16), where \( H \) identifies the whole group of the latent feature maps. The loss function is the same as that of the original autoencoder (MSE). For general convolutional neural networks (CNNs), a max pooling layer is often used to downsample the latent representation by passing only the maximum value over each pooling unit. Max pooling can also used in convolutional autoencoders to introduce sparsity over the hidden layers, which helps
extract more abstract features and speeds up the computation. As in deconvolutional layers, an upsampling step also needs to be included in the decoder network in order to recover/reconstruct each initial data value. The encoder and decoder networks can be described via:

\[
    z_k^{(i)} = \sigma_1(W_{1k} \ast x^{(i)} + b_1), \quad i = 1, \ldots, m, \quad k \in H, \quad (4.15)
\]

\[
    \tilde{x}^{(i)} = \sigma_2(\sum_{k \in H} W_{2k} \ast z_k^{(i)} + b_2), \quad i = 1, \ldots, m. \quad (4.16)
\]

### 4.1.2 Applications

Recently, autoencoder networks (especially deep autoencoder networks), have shown to be very successful in a number of applications, due to their ability to efficiently extract good features from different kinds of data. In the following, we will introduce some general applications of the autoencoder. Then, we will discuss some semi-supervised and supervised extensions of autoencoders in which some supervised information is provided to the autoencoder network, making autoencoders useful for classification or target detection tasks. Finally, we will discuss applications of autoencoders HSI.

**General applications**

In [166], a stacked autoencoder network with three hidden layers is used to control the parameter space of an audio effect and adapt to a new audio spectrum. Compared with nine other DR methods, the stacked autoencoder exhibits the lowest reconstruction variance and provides the most accurate results. Yuan et al. [167] applies stacked sparse autoencoders to the problem of recognize polyps in wireless capsule endoscopy images. The autoencoder network has two hidden layers and provides a more robust way to extract features from the endoscopy images, enabling physicians to have a system that automatically recognizes polyps. Makhzani et al. [168]
combines autoencoders with generative adversarial networks (GANs) to perform variational inference while enabling the decoder network to generate new data. This framework can also be extended for semi-supervised classification, data visualization and disentangling style. Socher et al. [169] applies autoencoder networks to computational linguistics problems and achieves state-of-the-art results for sentence-level prediction. Gao et al. [170] uses autoencoder networks to extract features from images for human face recognition. They add a new item into the loss function to calculate the similarity between images of same person from different poses, so that the structure is more robust to pose variability. In [171], Erhan et al. propose several techniques to interpret the features computed by hidden neurons, similarly to visualizing filters in convolutional neural networks. They found that most random initializations of the parameters will surprisingly generate roughly the same patterns, and additionally that filters in the first layer resemble Gabor-like features.

**Semi-supervised and Supervised Autoencoders**

By incorporating class labels for some or all of the input data into the autoencoder network, the resulting features can be biased towards having some specific distribution that is helpful for the further processing. Wang et al. [172] propose a semi-supervised autoencoder that iteratively explores the relationship between samples and use this relationship to build the manifold structure. Wei et al. [173] use a semi-supervised autoencoder to extract latent representations of documents in which the local perspective correlates with nearby documents. Euclidean distance is calculated to define neighbors, and the loss function includes the difference between input data and output reconstructions as well as the difference between neighbors of input and output. In [174], Lu et al. build a regularized stacked autoencoder with three hidden layers, and they add a manifold regularization term into the loss function in order to recover both the initial raw training features and their corresponding transformations. In [175], Kamyshanska et al. propose a new way to evaluate the similarity between the input and output, which they refer to as
the autoencoder confidence score. The autoencoder is regarded as a dynamical system, and an energy function is used to derive this score via integration. Multiple, unnormalized scores can be combined to form a generative classifier. Ranzato et al. [176] propose a stacked semi-supervised autoencoder which is trained layer-by-layer. In this network the decoder network is combined with a classifier so that the features from different classes are more separated. After training steps for the network, the decoder network and classifier are simply discarded, leaving just the encoder network for extracting features. In the paper [177], the center loss which calculates the difference between each sample and its class center in feature domain is added into total loss function. The center loss pushes features from the same class closer together, improving the performance of the subsequent classification procedure. It also explores how the number of layers affect the result, although it only uses a maximum number of layers of six. In [178], Zhuang et al. propose an innovative autoencoder with two encoding layers: the embedding layer in which the distances between source and target are minimized, and the label layer in which a softmax regression model is included for the multiclass classification. Rifai et al. propose a supervised autoencoder [179] in which supervised information is added to contractive autoencoder so that the extracted features for each class are more robust and closer to each other. In summary, semi-supervised or supervised information that is provided to the autoencoder network enables the features to be learned so that features from the same classes will be closer to each other.

**HSI Applications**

Recently, autoencoder networks have become popular for various problems in HSI processing/analysis. Chen et al. [180] propose a deep learning spatial-spectral framework for hyperspectral image classification, in which two autoencoder networks are trained independently and then the features are merged together. One feature represents spectral information, and the other represents spatial information. Zhao et al. [181] upsample images to different spatial resolutions and then combine each pixel with its neighbors to provide fused spatial and spectral
information as input to a stacked sparse autoencoder. The features from the autoencoder are then used to train a random forest classifier. Tao et al. [182] uses a similar framework to [181] to extract the spatial information; however, they use windows with different sizes instead of upsampling the original image. Mugees et al. [183] also uses a stacked autoencoder network to extract spectral features from an image, but for extracting spatial features, they use a boundary adjustment technique. Han et al. [184] develop a spatial-spectral convolutional autoencoder network for the classification, using filters of different sizes to extract different spatial features.

Liu et al. [123] use autoencoder networks for dimensionality reduction and classification, with superpixels as input; majority voting is adopted for each part in the superpixel map based on the classification result. In [185], Sun et al. combine active learning and autoencoder networks: active learning is used to search for the most informative training samples, and then those samples (as opposed to all of the data) are used to train an autoencoder to extract features for classification. Bati et al. [186] propose using autoencoder networks for hyperspectral anomaly detection: the output of the encoder network is a set of features that are representative of the differences between normal data and abnormal data. Ozkan et al. [187] use sparse autoencoder networks to extract endmembers for hyperspectral unmixing. They use batch normalization to make the learning process more robust, and they adopt a loss function that is based on SAM instead of MSE.

Zhu et al. [188] develop a system that combines maximum noise fraction (MNF) for reducing the inherent spectral dimensionality and an autoencoder network for extracting high-level features. Zhong et al. [189] develop a diversified deep belief network through regularizing, pretraining and fine-tuning procedures. In [190], an autoencoder network helps find the most informative HSI bands for DR. Ma et al. [191] use autoencoder networks with small training sets: two regularizer terms are added into the loss function to minimize the discrepancy within each class and maximize the difference among different classes. Guo et al. [192] uses an autoencoder cascade (two autoencoder networks) for hyperspectral unmixing. One autoencoder is
used for denoising and the other is used to estimate the endmembers. To sum up, different kinds of autoencoders have been widely used in a variety of HSI applications, including classification, unmixing, target detection, anomaly detection, and denoising.

Based on all of the above papers, we can see that autoencoder networks have emerged as powerful tools for extracting features from different kinds of data, and it is straightforward to provide them with supervised information. However, some questions still arise about how autoencoders are currently used in HSI analysis. First, although most of the papers claim that their autoencoder networks are “deep,” the vast majority have less than ten layers, even less than five layers. Second, most of autoencoder networks applied to supervised HSI applications have relatively simple loss functions, resulting in potentially limited improvements over other techniques. Finally, most applications simply use the encoder network to extract features and ignore the decoder network entirely, making it difficult to assess whether or not the features are actually best for reconstructing the original data. In order to deal with these difficulties, and to connect autoencoders to other nonlinear DR methods, we develop our semi-supervised deep autoencoder network (SSDAN) in the next section.

4.2 Proposed Autoencoder Network: SSDAN

In order to address the prohibitive computational and memory burden of nonlinear DR, we propose a semi-supervised deep autoencoder network (SSDAN) that approximates the results of nonlinear DR algorithms such as SSSE in a manner that quickly approximates the resulting embeddings and enables simple extensions to unseen data. The structure of the SSDAN is shown in Figure 4.2. If we define $\hat{Z} = \{\hat{z}^{(1)}, \ldots, \hat{z}^{(m)}\}$ to be the low-dimensional embeddings generated by the desired nonlinear DR algorithm (e.g., SSSE), and $Y = \{y^{(1)}, \ldots, y^{(m)}\}$ to be
the class labels for each data point, then the loss function for the SSDAN is given by:

$$\mathcal{L}(X, \hat{X}; Z, \hat{Z}, \hat{Y}) = \eta \mathcal{L}_{MSE}(X, \hat{X}) + \alpha \mathcal{L}_{\text{Manifold}}(Z, \hat{Z}) + \beta \mathcal{L}_{\text{Mean}}(Z, \hat{Z} ; Y) + \gamma \mathcal{L}_{\text{Class}}(Z ; Y),$$  

(4.17)

where $\mathcal{L}_{MSE}$ is the MSE loss defined by (4.7), and where

$$\mathcal{L}_{\text{Manifold}}(Z, \hat{Z}) = \frac{1}{m} \sum_{i=1}^{m} \left\| z^{(i)} - \hat{z}^{(i)} \right\|^2, \quad (4.18)$$

$$\mathcal{L}_{\text{Mean}}(Z, \hat{Z}; Y) = \frac{1}{m} \sum_{i=1}^{m} \left\| z^{(i)} - \frac{\sum_{y^{(j)} = y^{(i)}} \hat{z}^{(j)}}{\sum_{y^{(j)} = y^{(i)}} 1} \right\|^2, \quad (4.19)$$

$$\mathcal{L}_{\text{Class}}(Z; Y) = -\frac{1}{m} \sum_{i=1}^{m} \log \left( \frac{\exp \left( z^{(i)T} \omega_{y^{(i)}} \right)}{\sum_{k} \exp \left( z^{(i)T} \omega_{y^{(k)}} \right)} \right), \quad (4.20)$$
and $\Omega = [\omega_1, \omega_2, \ldots]$ is a weight matrix having rows equal to the number of dimensions in the feature space and columns equal to the number of class labels.

The SSDAN loss function has four terms, and $\eta, \alpha, \beta$ and $\gamma$ control the relative weight of these terms. The first term, $L_{\text{MSE}}$, is called the autoencoder loss, as it is used to help reconstruct the initial input based on the learned features. The second term, $L_{\text{Manifold}}$, is called the manifold loss, as it biases the output of the encoder network to be similar to the results of a specific nonlinear DR method (SSSE) applied to the training data. (Note that the training data can be selected to be a small enough subset of the original data so that computing $\hat{Z}$ becomes practical and efficient.) The third term, $L_{\text{Mean}}$, is called the mean loss, which pushes learned features for data from the same class to be closer to each other. The last term, $L_{\text{Class}}$, is called the class loss, which is a softmax loss function that helps separate features from different classes. With the integrated loss function, the specific nonlinear DR can be simulated. Moreover, the class information is combined that is useful for the further classification. Finally, the structure of autoencoder can help recover the initial data for compression or some other applications.

Figure 4.3: Flowchart showing the use of a SSDAN as a preprocessing step for classification.
using a traditional nonlinear DR method (such as SSSE). Next, we input the training data as well as their low-dimensional representations into the SSDAN to train it until the loss is small and stable. Then, we input both the training and testing data directly into the trained SSDAN in order to approximate low-dimensional representations. Finally, the approximate low-dimensional representations of the training and testing features computed from the SSDAN are passed to the classifier for training and testing.

4.3 Pixel-wise HSI Classification Experiments

4.3.1 Experimental Setup

To test how well the SSDAN can generate features that are useful for pixel-wise HSI classification, we apply it to classify the Indiana Pines, Salinas and Pavia images. For each image, half of the samples from each class are randomly chosen as training data and others are testing data. As a baseline comparison, we also use the original hyperspectral data and the data after projection onto the first 20 principal components as the input to the classifier. In all of the experiments, random forests (with 100 trees) are used for classification.

Two different nonlinear DRs will be simulated with SSDANs: SSSE and KPCA; they will be referred to as SSDAN-SSSE and SSDAN-KPCA. In both versions of SSDAN, we use nine hidden layers and reduce the dimensionality of the data to 20 dimensions. The number of neurons in each encoder hidden layer is 150, 100, 70, 40 and 20 and the decoder network has the symmetric structure, yielding a total of nine hidden layers. The number of the first input layer and the last output layer is same to the dimension of the input data. The training data is also used to calculate SSSE or KPCA for use in the manifold loss term. Weights and biases are initialized using a truncated normal distribution with mean 0 and standard deviation 1. $\eta, \alpha, \beta$ and $\gamma$ in the loss function have the values 1, 5, 5 and 5. We use a sigmoid activation function and transform all of the SSSE/KPCA embeddings to -1 to 1 so that the sigmoid function does not
squash them. Adam is used as the optimizer, the initial learning rate is $2 \times 10^{-4}$, and the learning rate is set to decay every 10000 steps with a base of 0.93. Because the initial data set is imbalanced, training samples in small classes are copied so that all of the classes have a similar number of training samples. We use a batch size of 200 and train for 300,000 steps.

We use the same measurements as in the superpixel-based algorithm to validate classification performance for SSDAN: overall accuracy (OA), kappa coefficient ($\kappa$) and computing time (Time) for the whole dataset, and precision (Pr) and sensitivity (Se) for each class. Computer time includes all of the time from the data normalization to the final classification.

### 4.3.2 Comparison with Other Methods

Table 4.1 shows the results of classification using original HSI data without modification, and using the low-dimensional embeddings constructed from PCA, traditional SSSE, traditional KPCA, SSDAN-KPCA and SSDAN-SSSE. The confusion matrices and classification map for all of the three images with SSDAN-SSSE and SSDAN-KPCA methods are shown from Table 7 to 12. and Figure 4.4 to 4.6. For the three different hyperspectral images, the results are similar for different methods.

Focusing first on the Salinas image, we see that use of the original data for classification gives an OA of 93.17%. When PCA is used as a preprocessing step, OA increases to 94.35%, but this is worse than all of the nonlinear DR methods. The poorer performance of PCA relative to the nonlinear DR methods is likely due to the lack of fit of a linear model to the HSI data. Using traditional SSSE, the OA is 98.23%, which is a large improvement over using PCA or the original data. SSDAN-SSSE yields OA of 95.21%, which is worse than traditional SSSE, but better than PCA and better than using the original data. Similar results occur with the KPCA and SSDAN-KPCA methods. Both of them have better OA than PCA and classification on the original data, and KPCA is slightly better than SSDAN KPCA. It is not surprising that SSDAN-based methods yield worse results than traditional nonlinear DR: SSDAN-based methods only
CHAPTER 4. SEMI-SUPERVISED DEEP AUTOENCODER NETWORKS

use a subset of the original data to approximate the nonlinear embedding, but traditional SSSE or KPCA include the testing data in performing dimensionality reduction. This means that the testing data is not actually “unseen” when using traditional nonlinear DR. However, the improved results of SSDAN-based methods over PCA and the use of the original data suggest the promise of SSDANs for nonlinear DR in pixel-wise HSI classification.

In investigating computing and memory requirements, we used a computer with an AMD FX-6300 Six-Core Processor and 32 GB memory. Both the baseline (original data) and PCA are very fast and can be performed with limited memory. For the Salinas image, SSSE-based classification takes 1,716 seconds and KPCA-based classification takes 432,173 seconds. For Indian Pines, these numbers are 103 seconds for SSSE and 23,316 seconds for KPCA. This illustrates how computing time increases quickly relative to the size of image. Both SSSE and KPCA require constructing an $N \times N$ matrix ($N$ is the number of pixels) that is used to solve a generalized eigenvector problem; $N = 21,025$ for Indian Pines and $N = 111,104$ for Salinas.

For the SSDAN-based methods, the time shown in table 4.1 just reflects the testing time. The training time using a NVIDIA GeForce GTX 1070 GPU is approximately 30 minutes for Indian Pines and Salinas and 20 minutes for Pavia due to the number of spectral wavelengths 200, 204, and 103 for Indian Pines, Salinas, and Pavia, respectively. In order to provide a fair comparison, we also trained the model on the CPU, and this required approximately 13 hours for Indian Pines and Salinas, and approximately 10 hours for Pavia. Once trained, low-dimensional embeddings can be computed from training and testing data at rates exceeding 1000 samples per second for CPU and 10000 samples for GPU, so the application of a trained SSDAN has complexity that is linear with the number of sample points and is much faster than with traditional nonlinear DR. In table 4.1, we can see that for the Salinas image, the testing time is 27 seconds for SSDAN-SSSE and 29 seconds for SSDAN-KPCA, and these are orders of magnitude faster than traditional SSSE (1,716 seconds) and KPCA (432,173 seconds). If a GPU can be used for SSDAN, the testing steps can be performed in less than 1 second. Similar
results occur for Indian Pines and Pavia.

Table 4.1: Results of pixel-wise classification using original HSI data, and using low-dimensional embeddings constructed from PCA, traditional SSSE, traditional KPCA, SSDAN-KPCA and SSDAN-SSSE.

<table>
<thead>
<tr>
<th></th>
<th>Indian Pines</th>
<th>Salinas</th>
<th>Pavia</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>OA</td>
<td>κ</td>
<td>Time (s)</td>
</tr>
<tr>
<td>Original</td>
<td>84.38</td>
<td>0.82</td>
<td>3.79</td>
</tr>
<tr>
<td>PCA</td>
<td>80.82</td>
<td>0.81</td>
<td>8.66</td>
</tr>
<tr>
<td>SSSE</td>
<td>99.17</td>
<td>0.99</td>
<td>103.2</td>
</tr>
<tr>
<td>SSDAN-SSSE</td>
<td>89.86</td>
<td>0.88</td>
<td>15.3</td>
</tr>
<tr>
<td>KPCA</td>
<td>94.85</td>
<td>0.94</td>
<td>23316</td>
</tr>
<tr>
<td>SSDAN-KPCA</td>
<td>91.04</td>
<td>0.90</td>
<td>16.7</td>
</tr>
</tbody>
</table>

Figure 4.4: Left: reference data of Indian Pines; middle: Classification maps for SSDAN-based SSSE method of Indian Pines; right: Classification maps for SSDAN-based KPCA method of Indian Pines.

4.3.3 Data augmentation

Next, we explore how data preprocessing affects the result of SSDAN-based pixel-wise HSI classification. Based on the information in table 3.1, we can see that each image has an imbalanced class distribution. For example, in Indian Pines, the largest class (Soybean-mintill) has 2455 samples and the smallest class (Oats) has 20 samples. We will investigate three different strategies for avoiding bias in the classifier due to the imbalanced classes. The first strategy is simply
Figure 4.5: Left: reference data of Salinas; middle: Classification maps for SSDAN-based SSSE method of Salinas; right: Classification maps for SSDAN-based KPCA method of Salinas.

Figure 4.6: Left: reference data of Pavia; middle: Classification maps for SSDAN-based SSSE method of Pavia; right: Classification maps for SSDAN-based KPCA method of Pavia.
to perform no augmentation, and to use the initial dataset to train the classifier. The second strategy is to incorporate weights into the loss function so that samples from small classes have greater weight than samples from large classes. The third strategy is to upsample the small classes so that each class has the similar number of samples.

Tables 4.2 and 4.3 show classification results for SSDAN-SSSE and SSDAN-KPCA. “Original,” “weighted,” and “augmented” in the tables correspond to the three different augmentation strategies. All of the classification results seem to be reasonable except for the ninth class in the SSDAN-SSSE method for Indian Pines. This is the smallest class, which contains just 10 samples for both training and testing. When we use all 10 samples in this class to train SSDAN, the precision is only 30%. However, weighted and augmented methods don’t improve the performance for this class and precision and sensitivity becomes even worse. For the OA with different strategies, the results are very similar for different datasets. For example, for SSDAN-SSSE method in Indian Pines, the OA for original, weighted and augmented methods are 89.54%, 89.27% and 89.38%. The original method has even a slightly better performance. In the remaining experiments, we will use the original dataset to train different SSDAN without augmentation.

4.3.4 Effectiveness of the loss function

In this section, we will demonstrate the effectiveness of our loss function. In (4.17), the loss function is composed of four sub-loss terms: the autoencoder loss, manifold loss, mean loss and class loss. We train with different values of the coefficients in the loss function. Values of the loss function are shown in Figure 4.7. Consider SSDAN-SSSE results on Indian Pines in the left column as an example: when there is only one sub-loss in the loss function (i.e., $\eta = 1$ and $\alpha = \beta = \gamma = 0$), the OA is close to 77%. This corresponds to using a simple unsupervised autoencoder for feature extraction, and it is slightly worse than using the original data or features from PCA. By setting $\beta = 5$, so that there are now two sub-loss terms, the mean loss
Table 4.2: Results of different data augmentation methods for SSDAN-SSSE.

<table>
<thead>
<tr>
<th>Class</th>
<th>Indian</th>
<th>Salinas</th>
<th>Pavia</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Original</td>
<td>Weighted</td>
<td>Augmented</td>
</tr>
<tr>
<td>Pr</td>
<td>Pt</td>
<td>Se</td>
<td>Pt</td>
</tr>
<tr>
<td>1</td>
<td>90.94</td>
<td>93.27</td>
<td>81.82</td>
</tr>
<tr>
<td>2</td>
<td>83.15</td>
<td>93.28</td>
<td>96.76</td>
</tr>
<tr>
<td>3</td>
<td>80.61</td>
<td>95.59</td>
<td>95.38</td>
</tr>
<tr>
<td>4</td>
<td>76.61</td>
<td>91.97</td>
<td>93.74</td>
</tr>
<tr>
<td>5</td>
<td>94.24</td>
<td>93.80</td>
<td>84.63</td>
</tr>
<tr>
<td>6</td>
<td>96.66</td>
<td>97.69</td>
<td>98.30</td>
</tr>
<tr>
<td>7</td>
<td>78.57</td>
<td>100</td>
<td>97.94</td>
</tr>
<tr>
<td>8</td>
<td>99.60</td>
<td>89.84</td>
<td>100</td>
</tr>
<tr>
<td>9</td>
<td>80.00</td>
<td>98.81</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>85.48</td>
<td>85.14</td>
<td>82.82</td>
</tr>
<tr>
<td>11</td>
<td>89.75</td>
<td>89.84</td>
<td>89.75</td>
</tr>
<tr>
<td>12</td>
<td>92.69</td>
<td>96.14</td>
<td>91.81</td>
</tr>
<tr>
<td>13</td>
<td>100</td>
<td>95.24</td>
<td>100</td>
</tr>
<tr>
<td>14</td>
<td>97.31</td>
<td>96.09</td>
<td>96.52</td>
</tr>
<tr>
<td>15</td>
<td>81.05</td>
<td>84.62</td>
<td>82.63</td>
</tr>
<tr>
<td>Augmented</td>
<td>94.45</td>
<td>94.89</td>
<td>94.89</td>
</tr>
</tbody>
</table>

Table 4.3: Results of different data augmentation methods for SSDAN-KPCA.

<table>
<thead>
<tr>
<th>Class</th>
<th>Indian</th>
<th>Salinas</th>
<th>Pavia</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Original</td>
<td>Weighted</td>
<td>Augmented</td>
</tr>
<tr>
<td>Pr</td>
<td>Pt</td>
<td>Se</td>
<td>Pt</td>
</tr>
<tr>
<td>1</td>
<td>77.27</td>
<td>95.90</td>
<td>95.45</td>
</tr>
<tr>
<td>2</td>
<td>83.69</td>
<td>91.82</td>
<td>95.50</td>
</tr>
<tr>
<td>3</td>
<td>87.74</td>
<td>84.13</td>
<td>88.38</td>
</tr>
<tr>
<td>4</td>
<td>78.84</td>
<td>74.18</td>
<td>89.67</td>
</tr>
<tr>
<td>5</td>
<td>94.64</td>
<td>97.44</td>
<td>95.80</td>
</tr>
<tr>
<td>6</td>
<td>96.70</td>
<td>93.45</td>
<td>97.20</td>
</tr>
<tr>
<td>7</td>
<td>92.86</td>
<td>100</td>
<td>78.51</td>
</tr>
<tr>
<td>8</td>
<td>88.75</td>
<td>97.94</td>
<td>91.17</td>
</tr>
<tr>
<td>9</td>
<td>90.00</td>
<td>100</td>
<td>90.00</td>
</tr>
<tr>
<td>10</td>
<td>92.89</td>
<td>98.98</td>
<td>96.21</td>
</tr>
<tr>
<td>Augmented</td>
<td>91.28</td>
<td>98.84</td>
<td>91.37</td>
</tr>
</tbody>
</table>

OA: 89.54 89.27 89.83 95.67 95.93 94.21 99.46 99.92 99.94
AA: 98.69 98.66 98.66 98.66 98.66 98.66 98.66 98.66 98.66
AP: 95.96 82.10 83.44 98.08 97.82 97.93 95.63 95.32 95.41
Amp: 0.88 0.88 0.88 0.96 0.96 0.96 0.95 0.94 0.94
term pushes learned features closer to their center, and the OA increases to 87%, significantly surpassing the OA using PCA features. Additionally setting $\alpha = 5$ adds the manifold loss term so that the learned features are closer what would be expected from computing SSSE. This further increases OA to 89%. Finally, setting $\gamma = 5$ so that all four terms are included in the loss function increases the OA to 90%. Similar conclusions can be drawn for Salinas and Pavia images, and also for SSDAN-KPCA based results for all of the three images, which are shown in Figure 4.7.

Figure 4.8 shows the results of sensitivity analysis for different coefficients ($\eta$, $\alpha$, $\beta$ and $\gamma$) with all of the three images. For each curve, three of the sub-loss coefficients are fixed, and the other is varied from zero to nine. The blue triangle shows our choice of “best” values ($\eta = 1$, $\alpha = 5$, $\beta = 5$ and $\gamma = 5$). For each coefficients, we can see that when its value is zero, OA is usually poor. When its value varies from one to nine, the results are similar, which shows that all of the four parameters are not very sensitive to their values. Our choice performs very well compared with all of other choices.
4.3.5 Different activations

In this section, we explore different activation functions: sigmoid, tanh, relu and elu, which are defined in (4.2)–(4.5). Figure 4.9 shows results for different choices of activation functions in the SSDAN. For initialization of neurons, different standard deviations (std) are used to avoid the “NaN” problem during training. For relu and elu, we use smaller std (0.2) compared with std (1) with sigmoid and tanh function to deal with the exploding gradients problem. These results show that the use of the sigmoid activation function can lead to the highest classification accuracy. The tanh function has similar performances to the sigmoid in the Salinas and Pavia images, but it is much worse for Indian Pines. Both ReLU and ELU exhibit poor performance for both images.

4.3.6 Different numbers of layers

In this section, we explore the effect of the number of layers in the SSDAN. Table 4.4 shows the number of hidden layers in different networks and the corresponding number of neurons in each layer. When the number of hidden layer is large, such as 15, we lightly decrease the number of neurons in the first two layers to avoid too much computation. Figure 4.10 shows different
Figure 4.9: OA of pixel-based HSI classification using SSDANs with different activation functions.

OAs for Indian Pines, Salinas and Pavia. For Indian Pines and Pavia, the highest classification accuracy is achieved when the SSDAN uses 9 hidden layers; for Salinas, the number is 15 hidden layers. Smaller or larger numbers of layers in the SSDAN appear to cause the classification performance to deteriorate.

Table 4.4: Number of neurons in each layer of encoder network

<table>
<thead>
<tr>
<th>Number of hidden layer</th>
<th>Hidden neurons in encoder layers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>100-20</td>
</tr>
<tr>
<td>5</td>
<td>140-70-20</td>
</tr>
<tr>
<td>7</td>
<td>160-120-60-20</td>
</tr>
<tr>
<td>9</td>
<td>150-100-70-40-20</td>
</tr>
<tr>
<td>15</td>
<td>120-100-100-100-80-60-40-20</td>
</tr>
<tr>
<td>19</td>
<td>120-100-100-100-100-80-60-40-20</td>
</tr>
<tr>
<td>29</td>
<td>120-100-100-100-100-100-100-100-80-60-60-40-20</td>
</tr>
</tbody>
</table>

4.3.7 Different numbers of neurons

Here we will explore how the number of neurons effects the accuracy of classification. We will assume here that the number of layers in the SSDAN is fixed at 9 hidden layers. However, the
Figure 4.10: OA of pixel-based HSI classification using SSDANs with different numbers of layers.

The number of neurons in each layer will change. Table 4.5 shows the number of neurons in each layer of different networks. Figure 4.11 shows the OAs for all three images. When the number of neurons is small, the OAs are poor in all cases. Except for SSDAN-SSSE applied to Pavia, all of other curves exhibit increasing trends. In general, it appears that increasing the number of neurons yields better classification results.

Table 4.5: Number of neurons in each layer of encoder network

<table>
<thead>
<tr>
<th>No.</th>
<th>Hidden neurons in encoder layers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>80-65-40-25-20</td>
</tr>
<tr>
<td>2</td>
<td>100-80-50-30-20</td>
</tr>
<tr>
<td>3</td>
<td>150-100-70-40-20</td>
</tr>
<tr>
<td>4</td>
<td>180-130-100-70-20</td>
</tr>
<tr>
<td>5</td>
<td>200-160-140-100-20</td>
</tr>
<tr>
<td>6</td>
<td>250-200-150-120-20</td>
</tr>
<tr>
<td>7</td>
<td>400-300-200-100-20</td>
</tr>
</tbody>
</table>

4.3.8 Different dimensions

In this section, we explore different feature dimensions. We vary the dimension from 5 to 100. Figure 4.12 shows the resulting OAs of classification. We can see that very low dimensions (such
Figure 4.11: OA of pixel-based HSI classification using SSDAN with different numbers of neurons.

as 5) or very high dimensions (such as 100) do not achieve the best results. Rather, dimensions from 20 to 50 appear to achieve better results.

Figure 4.12: OA of pixel-based HSI classification using SSDAN with different feature dimensions.

4.3.9 Different number of training samples

In order to explore how the amount of training data affects the ability of the SSDAN to yield embeddings that are good for classification, we use different amounts of training data (from 20% to 80%) to train the SSDAN. The results are shown in Figure 4.13. We can see that
when just 20% of the data is used as training data, the OA is around 87% for SSDAN-SSSE method in Indiana Pines. Then the OA increases as the number of training data increases; it is higher than 90% when 50% of the data are used to train SSDAN and then the OA starts to decrease. We can see similar situations for both SSDAN-SSSE and SSDAN-KPCA in Indiana Pines, Salinas and Pavia images. When the amount of training data is relatively small, SSDAN is overfitting and yielding poor results for the test data. When the number of training data is much larger than what is required for training SSDAN, the OA can’t continue to increase because it is underfitting.

Figure 4.13: OA of pixel-based HSI classification using SSDANs as a function of different training set sizes.

4.3.10 Transfer learning

In this section, we explore transfer learning for SSDANs. Transfer learning is the learning of a new task given knowledge that has been learned from another task [193]. Transfer learning is widely used when there is not enough data to train a model from scratch. The model can be trained, for example, on a large publicly available dataset, and then it can be finetuned with the specific training data from the new task. One of the most famous supervised image dataset is Imagenet [194] which includes millions of RGB images and the corresponding reference data. Some experimental results based on Imagenet in [195] show that low-level features, such as edges
and corners of an object, can be learned from the beginning layers of CNN, and the high-level features, such as the feature which includes shape and texture information of the whole object, can be learned from the top layers which is close to the final classifier. The low-level features are usually very similar for different kinds of images in that case pretrained model based on Imagenet is very popular and widely used in some other kinds of images. Here in order to demonstrate the effectiveness of transfer learning in my results, we will illustrate results using three different scenarios: 1) learn features on image B, and then use them to classify image B (“direct train”); 2) learn features on image A, and then use them to classify image B (“transfer without finetune”); and 3) learn features on image A, then finetune to classify image B (“transfer with finetune”). We remove the class loss and mean loss in SSDAN so that it can be used on two different dataset simultaneously. Figure 4.16 shows results for each of these scenarios with different images.

In the left sub-figure, image A corresponds to Indian Pines and image B corresponds to Salinas. Both of them come from the same sensor and they have very similar spectral range. There are 200 bands for Indian Pines and 204 bands for Salinas. The first 200 bands in Salinas are selected to use in the experiment so that they can have the same dimension for the input of the model. We can see that the result for the “transfer without finetune” scenario is much worse than the other two scenarios. This is due to the fact that some of the classes in the test data do not appear in the training dataset. The result is even worse than the result of PCA shown in Table 4.1. However, for the “transfer with finetune” scenario, the results are much better and are very similar to the results of the “direct train” scenario.

In the middle sub-figure, image A corresponds to Indian Pines and image B corresponds to Pavia. Indian Pines is collected by a different sensor than Pavia, and they have a different number of bands and spectral range. Indian Pines has 204 bands in the wavelength ranging from 400–2500 nm and Pavia has 103 bands in the wavelength ranging from 430–860 nm. Because image A and B have different dimensions, some preprocessing steps are needed. We split 204
bands of Indian Pines into 4 parts and each of them have 51 continuous bands. For each part, we randomly choose 26 contiguous bands (25 bands for the last part) and then concatenate them together based on the order of the band in the original Indian Pines. The new concatenated image can be used to pretrain the SSDAN and the spectra are very different with those in Pavia. The experimental result shows that similarly to the left figure, the “transfer without finetune” scenario does not achieve good results, but the “transfer with finetune” scenario yields much better results that are very similar to the results of the “direct train” scenario. In this experiment, the full spectra of the pixels for the two images are different, however, the low-level features (such as the increasing trend in the contiguous bands) are also similar so that transferring the prior knowledge with finetuning can help improve the results.

In order to try transfer learning in more images, we also used them on the Berlin and Paris images from the GRSS 2017 Data Fusion Contest. For each city, there are two different kinds of Landset data and each of them have 9 bands. There are also 10 sentinel2 images for every city. So the total number of bands captured for each city is 28. We concatenate them together to create single images for Berlin and Paris. The Berlin image is $666 \times 643$ and contains 11 land-use classes, and we crop the upper part ($310 \times 643$) for this experiment. The Paris image is $988 \times 1160$, and we use the whole image, which contains 12 land-use classes. The Figures 4.14–4.15 show pseudo-color images and reference data for both. Table 4.6 shows the number of samples for each land-use class. In the right sub-figure in Figure 4.16, image A corresponds to Berlin and image B corresponds to Paris. The “transfer without finetune” scenario still does not achieve good results; however, slightly differently from the left and middle sub-figures, but the “transfer with finetune” scenario yields much better results that are better than the results of the “direct train” scenario.

For the training speed, the “transfer with finetune” method just needs 100,000 iterations for the finetuning step instead of the initial 400,000 iterations from scratch in all of the three images which can save 75 percents of time compared with “direct train” method.
Figure 4.14: The Berlin pseudo-color image (spectral bands 2, 4, 6 in Landsat 8) and manually labeled reference data (11 classes).

Figure 4.15: The Paris pseudo-color image (spectral bands 1, 3, 5 in Landsat 8) and manually labeled reference data (12 classes).
In summary, transfer learning with finetuning in SSDAN has the potential to yield similar or better results than training with the original data. However, transfer learning without finetuning yields poor results.

Figure 4.16: OA of pixel-based HSI classification using SSDANs with transfer learning.

4.4 HSI Compression Experiments

In this section, we turn to the task of HSI compression. Figure 4.17 shows the results of preliminary experiments about how well a SSDAN can reconstruct spectral data. The first row shows the spectrum of one particular Salinas pixel and its recovery via SSDAN. We can see that there are 2 obvious errors in the recovered spectrum. Because the y-axis has different scales for the original and recovered figures, we crop the middle portion of the bands and show the details at the same scale in the second row. It appears that the recovered spectrum is a good reconstruction of the original spectrum except at a sparse set of points. More experiments shows that smaller standard deviation (std) of the initialization for the all of the weights can improve the recovered result. At beginning, the std is 1 for all of the weights. And then we change to 0.2 and the new recovered results shown in the last row of the figure. We can see that the recovered samples are very similar with the original data.

Now we compare the results of compression with SSDAN and compression with PCA. Learn-
Figure 4.17: The original and recovered data
ing an autoencoder with a single hidden layer and linear activation function is theoretically equivalent to PCA. Although the weight matrix in the autoencoder will not be exactly the same as the projection matrix in PCA, the subspace spanned by both matrices will be the same [196]. This illuminates a trade-off: as a SSDAN becomes larger, the features extracted from the SSDAN should differ more across different classes, which is very useful for classification, but it makes reconstruction of the original input more difficult.

To evaluate compression results, we use the Compression Ratio (CR) and Peak Signal to Noise Ratio (PSNR), which are defined in [197] and are given by:

\[
\text{CR} = \frac{\text{Band of Original Image}}{\text{Band of Output in the Encoder Network}},
\]

\[
\text{PSNR} = 10 \log_{10} \left( \frac{(255)^2}{\text{MSE}} \right),
\]

where

\[
\text{MSE} = \frac{1}{N \times M \times K} \sum_{i=1}^{N} \sum_{j=1}^{M} \sum_{k=1}^{K} (\text{Result}_{i,j,k} - \text{Reference}_{i,j,k})^2.
\]

Table 4.7 shows the fraction of variance for PCA method. We can see that even though we just keep five dimensions after PCA, 95% of the variance in the data is preserved. Figure 4.18 shows the PSNR and OA for Indian Pines, Salinas and Pavia versus CR. PCA has better PSNR than SSDAN in different CRs for all of three images, and PSNR decreases when CR increases. However, the OA for PCA is worse than the OA for SSDAN. An interesting observation is that for PCA, the highest OA is achieved when the CR is in a middle range, while OA for SSDAN has a decreasing trend with increasing CR. Figure 4.19 shows the histogram of mean square error (MSE) for all of the pixels in Indian Pines, Salinas and Pavia with PCA and SSDAN methods. For each of image, we firstly map the max value to 255 and then calculate the MSE for each of pixel and finally calculate the histogram. Comparing the histograms of PCA with
SSDAN, almost all of the points locate in the first bin for PCA methods and there are lots of points distribute in other bins for SSDAN methods. The PCA reconstruction is better than the result of SSDAN which is similar to the result from the first row of Figure 4.18. The figure 4.20 shows the map of L2 norm of the differences between actual spectra and reconstructed spectra at each pixel for Indian Pines, Salinas and Pavia. We can see that in the left column which shows the PCA results, the scale of colorbar is smaller than that in the right column which corresponding to the SSDAN results. Moreover, the temperature of color maps in the right column is obviously higher than that in the left column which also shows the reconstructed errors for PCA results are much smaller than those for SSDAN results.

![Figure 4.18: PSNR and CR versus OA of pixel-based HSI classification for Indian Pines, Salinas and Pavia.](image)

Figure 4.18: PSNR and CR versus OA of pixel-based HSI classification for Indian Pines, Salinas and Pavia.
Figure 4.19: Histogram of mean square error (MSE) for all of the pixels in Indian Pines, Salinas and Pavia with PCA and SSDAN methods.
Figure 4.20: L2 norm of the differences between actual spectra and reconstructed spectra at each pixel for Indian Pines, Salinas and Pavia. The left column corresponds to PCA reconstruction error and right column corresponds to SSDAN reconstruction error.
4.5 Summary

In this chapter, we introduced the SSDAN and its corresponding loss function that allows supervised information to drive the features from points in the same class closer to each other. We illustrated that SSDANs also approximate the results of performing nonlinear DR using much more computationally intensive methods like SSSE and KPCA. Experiments showed that SSDAN can significantly improve pixel-based HSI classification performance with respect to baseline methods, and that SSDAN is much faster than traditional nonlinear DR methods.
Table 4.6: Names and the number of samples for each reference class in Berlin and Paris image.

<table>
<thead>
<tr>
<th>Class</th>
<th>Berlin Classes</th>
<th>Number of Samples</th>
<th>Paris Classes</th>
<th>Number of Samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Built (Open high-rise)</td>
<td>124</td>
<td>Built (Compact high-rise)</td>
<td>56</td>
</tr>
<tr>
<td>2</td>
<td>Built (Open mid-rise)</td>
<td>329</td>
<td>Built (Compact mid-rise)</td>
<td>2705</td>
</tr>
<tr>
<td>3</td>
<td>Built (Open low-rise)</td>
<td>847</td>
<td>Built (Open high-rise)</td>
<td>366</td>
</tr>
<tr>
<td>4</td>
<td>Built (Large low-rise)</td>
<td>326</td>
<td>Built (Open mid-rise)</td>
<td>446</td>
</tr>
<tr>
<td>5</td>
<td>Built (Sparsely)</td>
<td>540</td>
<td>Built (Open low-rise)</td>
<td>2419</td>
</tr>
<tr>
<td>6</td>
<td>Land cover (Dense trees)</td>
<td>2992</td>
<td>Built (Large low-rise)</td>
<td>748</td>
</tr>
<tr>
<td>7</td>
<td>Land cover (Scattered trees)</td>
<td>600</td>
<td>Built (Sparsely)</td>
<td>60</td>
</tr>
<tr>
<td>8</td>
<td>Land cover (Bush, scrub)</td>
<td>544</td>
<td>Land cover (Dense trees)</td>
<td>4497</td>
</tr>
<tr>
<td>9</td>
<td>Land cover (Low plants)</td>
<td>3094</td>
<td>Land cover (Scattered trees)</td>
<td>394</td>
</tr>
<tr>
<td>10</td>
<td>Land cover (Bare soil or sand)</td>
<td>84</td>
<td>Land cover (Low plants)</td>
<td>7688</td>
</tr>
<tr>
<td>11</td>
<td>Land cover (Water)</td>
<td>208</td>
<td>Land cover (Bare rock or paved)</td>
<td>214</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td>Land cover (Water)</td>
<td>234</td>
</tr>
</tbody>
</table>

Table 4.7: Fraction of variance for PCA

<table>
<thead>
<tr>
<th>Ratio(%)</th>
<th>Dimension</th>
<th>Indian Pines</th>
<th>Salinas</th>
<th>Pavia</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>95.02</td>
<td>99.85</td>
<td>99.44</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>96.91</td>
<td>99.96</td>
<td>99.83</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>98.65</td>
<td>99.98</td>
<td>99.94</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>99.25</td>
<td>99.99</td>
<td>99.97</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>99.73</td>
<td>99.99</td>
<td>99.99</td>
<td></td>
</tr>
<tr>
<td>70</td>
<td>99.91</td>
<td>100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>99.99</td>
<td>100</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>
Chapter 5

Concluding remarks

5.1 Contributions

This thesis proposed two different ideas for improving the performance of nonlinear DR methods for various hyperspectral imaging analysis tasks: using SLIC superpixels, and using semi-supervised deep autoencoder networks (SSDAN). SLIC can be regarded as a preprocessing step to combine pixels with similar spatial and spectral information together so that the number of input data provided to the DR methods is significantly decreased. SSDAN is a way to exploit recent advances in deep learning by designing an autoencoder that can simulate embeddings computed by nonlinear DR techniques using only a small amount of training data without requiring complicated out-of-sample extension techniques to project new unseen data into the lower-dimensional embeddings.

Chapter 2 detailed the progress of HSI and its common analysis tasks that include classification, target detection, and compression. It then discussed the background and importance of dimensionality reduction in HSI. Nonlinear DR methods typically are much better than linear DR methods at capturing the manifold structure of HSI, but they require much more computational effort. Two examples of nonlinear DR, SSSE and KPCA, illustrate why nonlinear DR
is slow: both require computing an eigendecomposition of a $N \times N$ matrix, where $N$ is the number of pixels in the image. In order to overcome the computational burden, two different frameworks are proposed in Chapters 3 and 4.

Chapter 3 introduced SLIC superpixels for DR, in which the input of nonlinear DR becomes the mean value and position of all of pixels in each superpixel. Because the number of superpixels in the image is much smaller than the number of pixels, $N$ is significantly smaller, making nonlinear DR tractable. There are two important variables in SLIC: one is $s$ which controls size and the other is $r$ which decides regularity. Experiments are performed on three publicly available data sets: Indian Pines, Salinas and Pavia, and they show that computing time can be significantly reduced while retaining or even improving the accuracy of subsequent pixel-based classification. Sensitivity experiments show that superpixel regularity has much less effect than superpixel size.

Chapter 4 proposed a semi-supervised deep autoencoder network (SSDAN) for extracting features from HSI. By introducing nonlinear DR semi-supervised and class information, the features learned from SSDAN can be made similar to the features extracted from traditional nonlinear DR methods. Furthermore, the resulting embeddings will have smaller within-class distance and larger between-class distance, which is useful for subsequent classification tasks. Experiments show that SSDAN is better than using the original hyperspectral data or PCA-derived features as inputs for classification, and the results can nearly approximate results when use traditional nonlinear DR to construct inputs. As for computing time, training an SSDAN with 11 layers usually needs half an hour in a Nvidia Geforce GTX 1070 GPU (nearly 10 hours in AMD FX-6300 Six-Core CPU) and the testing time is smaller than 1 second for a 200 by 200 image, compared with traditional KPCA that requires more than 5 days on a CPU. This chapter also explored the effects of different variables in SSDAN such as dimensions, the number of layers, the number of neurons, and different activation functions. In addition, it explored the use of SSDANS in transfer learning, as well as their use in HSI compression tasks.
5.2 Future work

Based on the results we have found, there are some aspects that deserve more research in the future.

5.2.1 HSI classification

First, we can try more hyperspectral images and more nonlinear DR methods in the experiments of superpixel-based and SSDAN methods for the classification. Here, we experimented with three hyperspectral images (Indian Pines, Salinas and Pavia) and two nonlinear methods (KPCA and SSSE). With the development of better and cheaper sensors, it is easier to acquire hyperspectral image data and its corresponding reference data. There are also more effective nonlinear DR methods continually being proposed, and it would be useful to carry out experiments see if either of our proposed methods are “optimal” for some specific choice of nonlinear DR method in classification.

5.2.2 HSI target/anomaly detection

In the thesis, we focus mostly on the application of HSI in classification, and actually superpixel-based method and SSDAN-based method could also be extended to some other domains such as target detection and anomaly detection in the future. No matter which pipeline is used, the features can be extracted more quickly for the following processing. For example, for the anomaly detection, RX method [42] will compare the difference between the central pixel with its neighbors. We can firstly use SSDAN to decrease the dimension and then calculate the difference in the feature domain. Moreover, for some traditional target detection methods such as constrained energy minimization (CEM) [44] and matched filter (MF) [45], autocorrelation matrix or covariance matrix need eigendecomposition and SSDAN can help speed that up. Finally, graph-based methods such as Laplacian Eigenmaps (LE) [46], Schroedinger Eigenmaps [47, 48] and Spatial-Spectral Schroedinger Eigenmaps (SSSE) [50] are also popular to extract
features for the following target detection and we can try the similar pipeline to deal with those.

5.2.3 Regular image compression

In the thesis, I have explored how the decoder part in SSDAN reconstructs the initial input pixel and also compare the SSDAN result with PCA result. And it can be extended to some regular image datasets such as mnist. For example, in the mnist dataset [198], the shape of each image is 28 by 28 and it shows the digits from 0 to 9. Each of the digit have 6000 training samples which are enough to train a SSDAN with 11 hidden layers. So we can reshape the image to a vector with length of 784 and input that to SSDAN to extract the features. Based on the features, we can also reconstruct the initial input image and compare SSDAN result with the state-of-the-art results.

5.2.4 Modify the loss function for GAN

Generative adversarial network (GAN) [199] is very popular and hot as a generative model to produce different kinds of images. In order to speed up and improve the training of the generative model, there is the other discriminative model in GAN which can help separate the true and false image. Based on this idea, we can change the SSDAN structure and add the discriminative part in the middle hidden layer. The input for the discriminative part can be learned features from SSDAN and features learned from the tradional nonlinear DR method. In the training, the former features can have the label 0 and then convert to 1 alternately. The latter features always have the label 1. By adding the discriminative part, it should be easier for SSDAN to simulate the nonlinear DR method. The HSIs such as Indian Pines, Salinas and Pavia can be used again to test this idea.
5.2.5 Convolutional network to simulate nonlinear DR method

Except for fully-connected SSDAN, we can also try other types of neural networks to simulate nonlinear DR method, such as the U-net that is widely used in semantic segmentation [200]. In SSDAN, the input is a single pixel without its spatial information. The input of the U-net is the entire image, and convolutional and deconvolutional layers in the U-net can exploit the spatial relationships between pixels. The four different sublosses in SSDAN can also be integrated into U-net and just pixels having the label are used to update the network.
Appendices
Here we show some confusion matrices for the superpixel-based and SSDAN-based classification results in Indian Pines, Salinas and Pavia.

Table 1: Confusion matrix of superpixel-based SSSE method with \( s = 15 \) and \( r = 1 \) for Indian Pines. Rows correspond to the reference data and columns correspond to the predicted result.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>11</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1255</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>26</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>28</td>
<td>694</td>
<td>0</td>
<td>0</td>
<td>108</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>15</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>416</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>621</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>12</td>
<td>0</td>
<td>13</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>430</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>13</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>854</td>
<td>11</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>14</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>213</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>22</td>
<td>0</td>
<td>496</td>
<td>0</td>
<td>0</td>
<td>13</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>176</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>11</td>
<td>0</td>
<td>0</td>
<td>1127</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>17</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>17</td>
</tr>
</tbody>
</table>

Table 2: Confusion matrix of superpixel-based KPCA method with \( s = 15 \) and \( r = 1 \) for Indian Pines. Rows correspond to the reference data and columns correspond to the predicted result.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>16</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1255</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>27</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>28</td>
<td>694</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>18</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>196</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>16</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>419</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>644</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>25</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>13</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>7</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>855</td>
<td>9</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>0</td>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>18</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>11</td>
<td>2165</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>22</td>
<td>0</td>
<td>496</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>171</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>1138</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>342</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>78</td>
</tr>
</tbody>
</table>
Table 3: Confusion matrix of superpixel-based SSSE method with $s = 15$ and $r = 1$ for Salinas. Each row corresponds to the reference data and each column corresponds to the predicted result.

<table>
<thead>
<tr>
<th>1805</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1775</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1251</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>2999</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3563</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3201</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10068</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>136</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5583</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>47</td>
<td>14</td>
<td>2885</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>947</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>20</td>
<td>1707</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>23</td>
<td>798</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>130</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6409</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>22</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 4: Confusion matrix of superpixel-based KPCA method with $s = 15$ and $r = 1$ for Salinas. Each row corresponds to the reference data and each column corresponds to the predicted result.

<table>
<thead>
<tr>
<th>1807</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1772</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1234</td>
<td>21</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>2401</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3563</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>3219</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10143</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5583</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5295</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>31</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>930</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>13</td>
<td>1721</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>13</td>
<td>803</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>961</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6533</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1626</td>
</tr>
</tbody>
</table>

Table 5: Confusion matrix of superpixel-based SSSE method with $s = 15$ and $r = 1$ for Pavia. Each row corresponds to the reference data and each column corresponds to the predicted result.

<table>
<thead>
<tr>
<th>2310</th>
<th>6</th>
<th>7</th>
<th>30</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>23</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4793</td>
<td>0</td>
<td>8</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1865</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>15</td>
<td>9</td>
</tr>
<tr>
<td>33</td>
<td>43</td>
<td>2</td>
<td>1032</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>62</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1171</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>0</td>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2724</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1181</td>
<td>16</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5248</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>19</td>
<td>96</td>
<td>10</td>
<td>0</td>
<td>24</td>
<td>0</td>
</tr>
<tr>
<td>32</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Table 6: Confusion matrix of superpixel-based KPCA method with $s = 15$ and $r = 1$ for Pavia. Each row corresponds to the reference data and each column corresponds to the predicted result.

<table>
<thead>
<tr>
<th>2287</th>
<th>0</th>
<th>0</th>
<th>21</th>
<th>0</th>
<th>0</th>
<th>69</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4792</td>
<td>0</td>
<td>14</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1873</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>10</td>
<td>0</td>
<td>1053</td>
<td>0</td>
<td>3</td>
<td>68</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1192</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2729</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1195</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>14</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>19</td>
<td>18</td>
<td>132</td>
<td>2</td>
<td>4</td>
<td>8</td>
</tr>
</tbody>
</table>

Table 7: Confusion matrix of SSDAN-based SSSE method for Indian Pines. Each row corresponds to the reference data and each column corresponds to the predicted result.

<table>
<thead>
<tr>
<th>17</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>4</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>608</td>
<td>17</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>24</td>
<td>63</td>
</tr>
<tr>
<td>0</td>
<td>7</td>
<td>359</td>
<td>18</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>6</td>
<td>101</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>229</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>367</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>19</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>424</td>
</tr>
<tr>
<td>0</td>
<td>32</td>
<td>21</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>37</td>
<td>1084</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>275</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>617</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>7</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 8: Confusion matrix of SSDAN-based KPCA method for Indian Pines. Each row corresponds to the reference data and each column corresponds to the predicted result.

<table>
<thead>
<tr>
<th>19</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>2</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>624</td>
<td>15</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>17</td>
<td>57</td>
</tr>
<tr>
<td>0</td>
<td>6</td>
<td>366</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>7</td>
<td>101</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>226</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>370</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>250</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>9</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>441</td>
</tr>
<tr>
<td>0</td>
<td>31</td>
<td>16</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>27</td>
<td>1107</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>615</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>
Table 9: Confusion matrix of SSDAN-based SSSE method for Salinas. Each row corresponds to the reference data and each column corresponds to the predicted result.

|   | 2  | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1866 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 985 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 698 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 2 | 4 | 1328 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1977 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1785 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 5643 | 0 | 6 | 0 | 0 | 0 | 0 | 0 | 572 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3101 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 11 | 1609 | 0 | 2 | 0 | 1 | 0 | 4 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 459 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 7 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2939 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 881 |

Table 10: Confusion matrix of SSDAN-based KPCA method for Salinas. Each row corresponds to the reference data and each column corresponds to the predicted result.

|   | 2  | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1868 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 980 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 689 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 6 | 1439 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1975 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1874 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 544 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 11: Confusion matrix of SSDAN-based SSSE method for Pavia. Each row corresponds to the reference data and each column corresponds to the predicted result.

<table>
<thead>
<tr>
<th></th>
<th>1233</th>
<th>0</th>
<th>7</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>23</th>
<th>46</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2607</td>
<td>1</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>35</td>
<td>0</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
<td>913</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>127</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>14</td>
<td>0</td>
<td>614</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>655</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>35</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1457</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>624</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>1</td>
<td>88</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1684</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>420</td>
<td></td>
</tr>
</tbody>
</table>
Table 12: Confusion matrix of SSDAN-based KPCA method for Pavia. Each row corresponds to the reference data and each column corresponds to the predicted result.

<table>
<thead>
<tr>
<th></th>
<th>1230</th>
<th>0</th>
<th>9</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>31</th>
<th>39</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2607</td>
<td>0</td>
<td>9</td>
<td>0</td>
<td>33</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>3</td>
<td>962</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>137</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>615</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>654</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1464</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>629</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>1</td>
<td>94</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1673</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>419</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Bibliography


[17] L. Fang, S. Li, W. Duan, J. Ren, and J. A. Benediktsson, “Classification of hyperspectral images by exploiting spectral–spatial information of superpixel via multiple kernels,”


[125] X. Cui, Y. Tian, L. Weng, and Y. Yang, “Anomaly detection in hyperspectral imagery
1, 2014.

[126] A. Çahşkan, A. Koz, and A. A. Alatan, “Hyperspectral superpixel extraction using bound-
ary updates based on optimal spectral similarity metric,” in Geoscience and Remote Sens-

[127] Y. Liang, P. P. Markopoulos, and E. S. Saber, “Subpixel target detection in hyperspec-
tral images from superpixel background statistics,” in Geoscience and Remote Sensing

[128] A. Psalta, V. Karathanassi, and P. Kolokoussis, “Modified versions of slic algorithm for
generating superpixels in hyperspectral images,” in Hyperspectral Image and Signal Pro-
cessing: Evolution in Remote Sensing (WHISPERS), 2016 8th Workshop on, pp. 1–5,
IEEE, 2016.

EE367, 2017.

matrix factorization for hyperspectral unmixing,” IEEE Transactions on Geoscience and

image data set: June 12, 1992 indian pine test site 3,” Purdue University Research Repos-

band selection through optimum-path forest and evolutionary-based algorithms,” in Proc.


