LEt-SNE: A Hybrid Approach to Data Embedding and Visualization of Hyperspectral Bands in Satellite Imagery

A Dissertation
Submitted in partial fulfillment of
the requirements for the degree of
Master of Technology
by

Megh Shukla
(173310008)

Supervisors:
Prof. Krishna Mohan Buddhiraju
and
Prof. Biplab Banerjee

Centre of Studies in Resources Engineering
Indian Institute of Technology Bombay
Mumbai 400076 (India)

4 June 2019

Dedicated to my grandfather and my parents, whose constant support motivates me to keep giving my best efforts in all the endeavours I undertake…

Approval Sheet

This dissertation entitled "**LEt-SNE: A Hybrid Approach to Data Embedding and Visualization of Hyperspectral Bands in Satellite Imagery**" by Megh Shukla is approved for the degree of Master of Technology.

Examiners

Supervisor (s)

Chairman

Date: 4th June, 2019

Place: IIT Bombay, Mumbai

Declaration

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I declare that I have properly and accurately acknowledged all sources used in the production of this report. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be a cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Megh Shukla
(173310008)

Date: 4 June 2019

Abstract

Hyperspectral Images are characterized by a large number of contiguous, correlated bands having a high spectral resolution in the region of *10 nm*. It is common to see hyperspectral images having 100-200 bands, as opposed to ~7 in Multispectral images. These band values then form the spectrum of the underlying pixel. Storing and processing the vast amounts of data contained in hyperspectral images is cumbersome and expensive which leads us to an extensively researched topic, *Dimensionality Reduction.*

Dimensionality Reduction aims to reduce the number of bands present in the original data, with the aim of preserving as much information present as possible. Many Dimensionality Reduction techniques exist, such as the *Principal Component Analysis*, which focusses on preserving large pairwise distances. Another well known technique is *Laplacian Eigenmap*, which uses Graph Theory to preserve Distances among neighbouring points. Recent advancements in this domain include *t-SNE*, a probabilistic approach towards preserving distances, as well as *UMAP (Uniform Manifold Approximation and Projection)*, which is based on projecting the points on a Riemannian manifold. Techniques based on Machine Learning such as *Denoising Autoencoders* are also used for Dimensionality Reduction.

In this thesis, we examine the application of the aforementioned current state of the art techniques to the Hyperspectral Imagery domain. We also propose a new technique, ***LEt - SNE***, which combines *Laplacian Eigenmaps* with a modified version of *t-SNE*. This algorithm can be used visualize the manifold of the underlying data as well as cluster our dataset in supervised and unsupervised setting. The proposed algorithm achieves an improvement over the state of the art results for dimensionality reduction and visualization of Hyperspectral images. To support the claim, a comparative study is performed by applying the algorithms on three datasets: *Indian Pines, Salinas* and *Pavia University.* We restrict the work in this thesis to data embedding in 2-3 dimensions, where the manifold can be easily visualized by humans.

Table of Contents

List of Figures

List of Tables

List of Symbols

Roman Symbols

$\mathcal{L}$	Graph Laplacian	19
$\mathcal{A}$	Graph Adjacency Matrix	19
$\mathcal{D}$	Graph Degree Matrix	19
$\mathcal{X}$	Samples in the original space	4
$\mathcal{Y}$	Samples in the reduced space	4

Greek Symbols

η	Learning rate hyperparameter	35
α	Momentum hyperparameter	35

Superscripts

i	i^{th} sample	9

Subscripts

j	j^{th} component or dimension in the sample	9

Acronyms

ANN	Artificial Neural Network	10
CF	Compression Factor	59
CNN	Convolutional Neural Networks	37
DAE	Denoising Autoencoder	31

DoF	Degree of Freedom	25
HSI	Hyperspectral Imagery	1
K-PCA	Kernel Principal Component Analysis	4
LDA	Linear Discriminant Analysis	16
LLE	Locally Linear Embedding	4
MNF	Minimum Noise Fraction	12
MSE	Mean Square Error	35
PCA	Principal Component Analysis	4
CAE	Contractive Autoencoder	31
QDA	Quadratic Discriminant Analysis	16
RAG	Region Adjacency Graph	67
ReLU	Rectified Linear Unit activation	35
SLIC	Simple Linear Iterative Clustering Algorithm	67
SNE	Stochastic Neighburhood Embedding	4
sSNE	Spherical Stochastic Neighbour Embedding	49
t-SNE	t-Stochastic Neighbourhood Embedding	4
UMAP	Uniform Manifold Approximation and Projection	4

Other Symbols

m	Number of dimensions in $\mathcal{X}$	13
N	Number of samples in $\mathcal{X}$	13
n	Number of dimensions in $\mathcal{Y}$	13

Chapter 1

Introduction

This chapter contains a brief overview of hyperspectral images, along with an introduction to the problem statement. We address the necessity of Data Embedding and Visualization, and frame the objectives which form the basis for the rest of the thesis.

1.1 Hyperspectral Imagery

A hyperspectral image *(HSI)* is typically represented in the form of a hyperspectral cube (Figure: 1.1), with two axes being formed by the spatial extent of the image and the third axis formed by the spectral component. A hyperspectral image is superior to the multispectral image primarily because it can accurately capture the target object's spectral signature, which is its unique identity. No two targets have an identical spectral signature, unless they are of the same kind. Hyperspectral sensors are differentiated from their multispectral counterparts by a large number of narrow, contiguous spectral bands. A typical hyperspectral sensor has hundreds of 10nm bands covering a wide spectrum range. Figure 1.2 displays the spectrum of various varieties of agricultural crops.

The recent past has seen has seen large scale progress in the field of remote sensing and application of satellite imagery. As time progresses, newer and better technologies have evolved with increasing spectral as well as spatial resolution of the sensors. Some of the recently developed sensors such as the *AVIRIS – NG*, boast a spatial resolution of 4m and a spectral resolution of 5nm. Increasing resolution leads to better imaging of the target, capturing higher amount of detail. Other satellite sensors include the *AVIRIS* and *Hyperion*, with spatial resolutions of 20m and 30m respectively, and spectral resolution of 10nm.

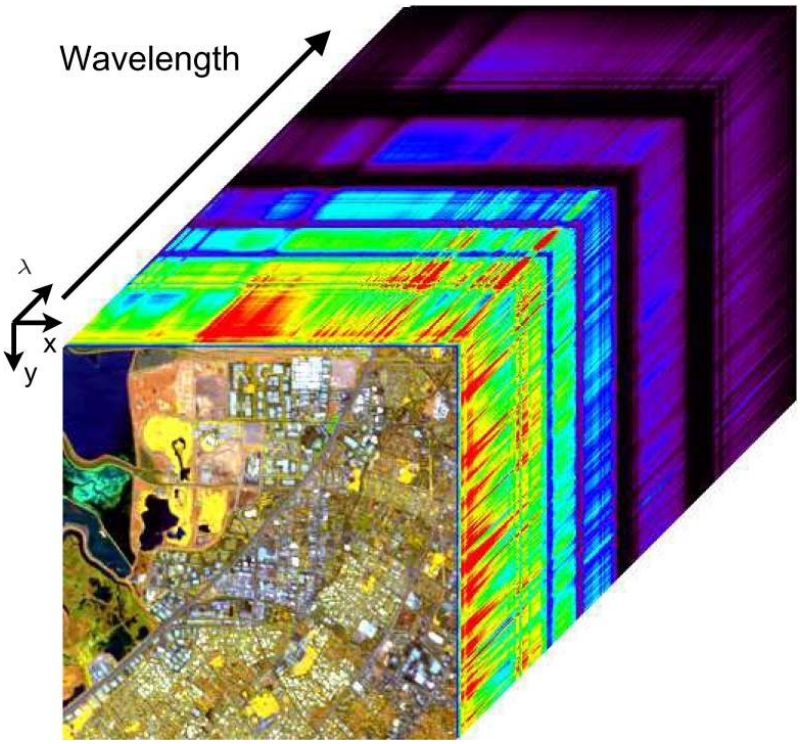


Figure 1.1: A Hyperspectral Cube [8]

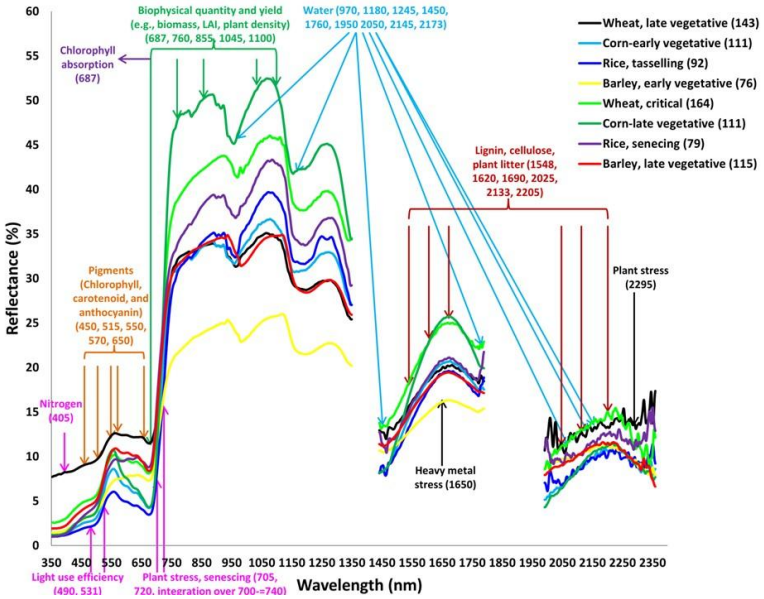


Figure 1.2: Spectrum with atmospheric absorption correction [48]

The high level spectral resultion of the hyperspectral sensor has its own pitfalls. The increased spectral resolution comes at a tradeoff with spatial resolution. Since the band widths are narrow (~5nm), the energy captured by each band is small, making it prone to atmosphere and instrument induced noise. To compensate for the high spectral resolution, the pixel size of the sensor is increased so as to capture energy from a larger area. This calls for a spatial/spectral tradeoff.
We summarize the characteristics of Hyperspectral Imagery in comparison to conventional images:

1. Spectrum is formed from **multiple narrow, contiguous** bands
2. **Scale**: Hyperspectral imagery taken from airborne/spaceborne vehicles span kilometers when compared to conventional images
3. **Correlated noise**, prone to atmospheric induced noise and attenuation

1.2 Curse of Dimensionality

> "In view of all that we have said in the foregoing sections, the many obstacles we appear to have surmounted, what casts the pall over our victory celebration? It is the curse of dimensionality, a malediction that has plagued the scientist from the earliest days." - Richard Bellman [4]

In the previous section 1.1, we discussed the nature of hyperspectral images, characterized by ~200 dimensional data. A natural question arises:

Would it not be more effective to use the complete spectrum for analysis rather than using a reduced form of the original data?

The answer to the query posed above lies in understanding the behaviour of distances in a high dimensional space in comparison to a low dimensional space. Figure 1.3 demonstrates the variation of distances between points as we increase the dimensionality. Assume our data points are sampled from $\mathbb{R}^d space$, where d denotes the dimensionality of our data. As the dimensionality d increases:

$$\lim_{d\to\infty} \mathbb{E}\left(\frac{dist_{max}(d) - dist_{min}(d)}{dist_{min}(d)}\right) \to 0 \tag{1.1}$$

For low dimensional data, the concept of *neighbourhood points* clearly exists, as the euclidean distance between samples has a large variation. This indicates that there exists a greater separation between non-adjacent points in the lower dimensions, making the data

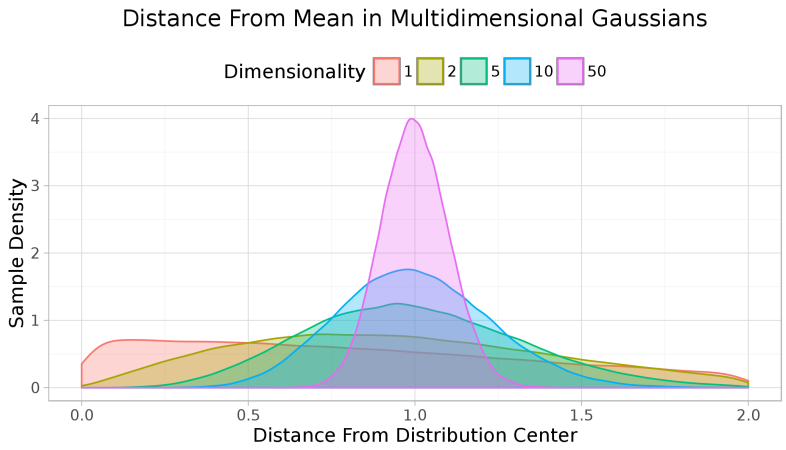


Figure 1.3: Distribution of pairwise distances between samples for data of various dimensionality [19]

suitable for many applications such as *k-means clustering*. However, as d increases, the variation of the pairwise euclidean distance between points decreases when compared to the minimum distance between a point and its neighbours. The physical interpretation of Equation 1.1 indicates that points are equally spread out in higher dimensions, thus losing the notion of neighbour and adjacency. This renders many algorithms that depend on euclidean distances ineffective, bringing the need for Dimensionality Reduction.

1.3 Recent techniques in Dimensionality Reduction

Dimensionality Reduction has been a well explored research avenue, with early papers going back all the way to 1901. No technique of dimensionality reduction is complete without including the *Principal Component Transform (PCA)* [37], which is a technique that rotates the samples along principal component which explains the maximum variance of the data. PCA has been a reliable technique which acts as a pre-processing step for many algorithms. However, being a linear technique it fails to perform when handling non-linearities in the data. To counter this drawback, a new technique was proposed; *Kernel Principal Component Analysis [44]*, which introduces a non-linear kernel inspired from the *Support Vector Machine*. [10] A major drawback with *K-PCA* is its prohibitive space $O(n^2)$, and time complexity, $O(n^3)$, which limits its practical applications.

A technique similar to *PCA* is the *Linear Discriminant Analysis* [13], a supervised approach which uses class labels to maximize the separation between different clusters. There are quite a few limiting assumptions made by the algorithm, which include class conditioned normal distribution and *homoscedasticity assumption*.

Another category of Dimensionality Reduction techniques include Graph algorithms such as *Laplacian Eigenmaps* [3] and *Locally Linear Embedding (LLE)* [42]. These algorithms create an adjacency graph between samples in $\mathcal{X}$ using *K-Nearest Neighbours* which is based on the Euclidean Distance between the samples. We will cover these algorithms in the later chapters.

Recent advancements include *Stochastic Neighbour Embedding(SNE)* [22] and *t-SNE* [32], which induce a probability distribution over all points in the dataset. We then use KL divergence to minimze the difference in the probability distribution between the original points $\mathcal{X}$ and embedded data points $\mathcal{Y}$. A new approach towards dimensionality reduction was published in 2018, *UMAP* [34] which relies on projecting points along a Reimannian manifold.

Deep Learning has also been used extensively, with the introduction of Autoencoders, first mentioned in 1989 [30] [21]. Since then, many variants of Autoencoders have been introduced. In this work, we also draw comparisons with a particular class of autoencoders, the *Denoising Autoencoders* [36], which adds random noise to the input and guides the network to learn the clean reconstruction. This enables the network to learn powerful features in the bottleneck layer which acts as lower dimensional data embedding.

1.4 LEt-SNE: A Brief overview

As we have seen in section 1.3, a plethora of algorithms already exist to tackle the issue of dimensionality reduction. One can't help but wonder the necessity for another new approach towards solving the problem. Some important questions remain:

1. How robust are these techniques when applying to Hyperspectral Imagery?
2. How do they tackle the Curse of Dimensionality [Sec: 1.2]?
3. Do they preserve the original manifold of the data?
4. Given the rich spectral resolution of Hyperspectral Images, a pair of classes could be separated by the finest of margins. Can our algorithms preserve as much as a separation as possible?
5. Can we preserve natural clustering of samples in our dataset in an unsupervised setting?

LEt-SNE, a hybrid combination of *Laplacian Embedding* and *t-SNE* attempts to provide a solution to the questions posed above. Some of the novel ideas of this technique include:

1. Algorithm that can be used for manifold visualization in two/three dimensions. It can also be used to cluster data in a supervised as well as unsupervised setting. The algorithm does not use classification gradients at any stage, which has a tendency to break the manifold as well as overfit solely to the task of classification.
2. Stochastic approach to embedding by implementing mini-batches for *LEt-SNE*. This is especially benifical for situations where training data is too large to fit in the RAM and also leads to faster convergence. Stochasticity also has an inherent Regularization property which aids in generalization. [15]
3. Introduces *Compression Factor*, a multiplicative term which squeezes space between neighbouring points, tackling the problem of Curse of Dimensionality.
4. Proposes the use of image segmentation methods, which when combined with *Compression Factor*, boosts the quality of clustering in absence of labelled data for points in low dimensional space.
5. By modifying *t-SNE* and combining it with *Laplacian Embedding*, the resultant algorithm provides embeddings which on classification using labels exceeds the state of the art performance.

1.5 Organization of the Report

The content of the thesis has been split into six sections.

- **Chapter 1:** This chapter contains an overview of hyperspectral images and introduces relevant techniques used for dimensionality reduction. We also briefly introduce the novelty in our work.
- **Chapter 2:** This chapter is dedicated to literature review, identifying existing implementations and highlighting their strengths and limitations.
- **Chapter 3:** We identify the Problem statement which we attempt to solve in the subsequent chapters. Details regarding our dataset are presented in this chapter.
- **Chapter 4:** We conduct a detailed discussion on our approach towards solving the problem, in both the supervised and unsupervised setting.
- **Chapter 5:** In this chaper we compare the result of the proposed algoritm with the state of the art. We highlight the advantages and potential pitfalls in our algorithm.
- **Chapter 6:** In the last chapter, we present our conclusion, as well as talk about various areas that could be explored from our current work.

Chapter 2

Literature Survey

Dimensionality Reduction can be broadly divided into two categories, Feature Selection and Feature Extraction. As the name suggests, features selection picks those features (bands in the case of *HSI*) that best describe our dataset. While Feature selection retains the original information without any transformation, a major drawback with this approach is that it does not help us in understanding the underlying manifold. It also lacks discriminative powers associated with Feature Extraction algorithms. Feature Extraction, on the other hand attempts to find the relation among various features or samples in the dataset. This is usually accompanied with some form of feature transformation having a higher discriminative power. On the basis of the kind of transformation, we can further subdivide feature transformation into linear and non-linear methods. Fig 2.1 provides a non-exhaustive depiction of various algorithms for Dimensionality Reduction.

In this section, we will be emphasizing on Feature transformation techniques, as our objective lies in understanding the manifold of our data. We include in our discussion a select few important dimensionality reduction algorithms, since the vast literature available which would be too extensive to cover in this dissertation.

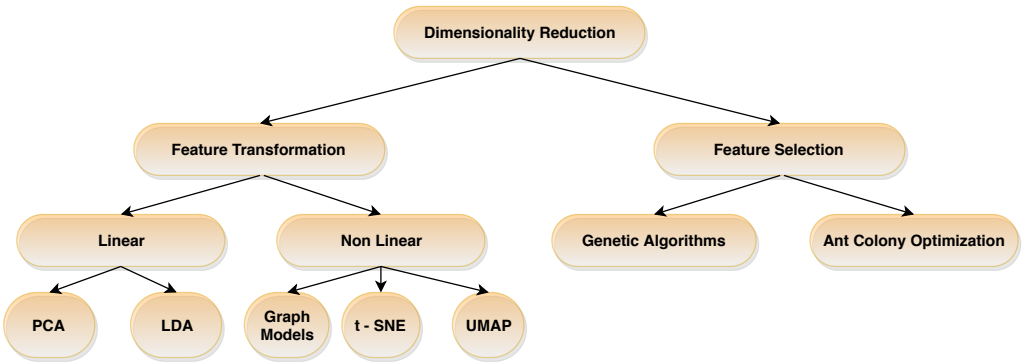


Figure 2.1: A brief overview of various dimensionality reduction algorithms

2.1 Principal Component Analysis

The Principal Component Analysis [37] is an unsupervised linear transformation which aims to preserve large distances in the original space $\mathcal{X}$. It is essentially a rotation of the axes such that the variance along the principal component is maximized. PCA is widely used as a pre-processing step in the analysis of Hyperspectral Imagery because of its interpretability and extensive study of the algorithm. In this section, we first review the theory behind PCA and then discuss the rationale behind the use of PCA.

2.1.1 Theory

The objective of the Principal Component Transform is to explain the variance within our data by rotating the axes. Like some other Dimensionality Reduction methods, we start by formalizing an error term [15] to be minimzed, which can be written as :

$$\mathcal{Y}^* = arg\min_{\mathcal{Y}} \| \mathcal{X} - g(\mathcal{Y}) \|_2^2 \tag{2.1}$$

The intuition behind this equation is to minimize the squared reconstruction error between the sample in the original space $\mathcal{X}$ and it's transformation g from $\mathcal{Y}$ in the reduced space. We let D be an orthonormal transformation matrix such that $\widehat{\mathcal{X}} = D * \mathcal{Y}$. Eq: 2.1 can also be written as:

$$arg\min_{\mathcal{Y}} (\mathcal{X} - D * \mathcal{Y})^T (\mathcal{X} - D * \mathcal{Y}) \tag{2.2}$$

On computing individual terms and dropping those terms not dependent on $\mathcal{Y}$, we can simplify the equation to:

$$\mathcal{Y}^* = arg\min_{\mathcal{Y}} -2\mathcal{X}^T D\mathcal{Y} + \mathcal{Y}^T DD^T \mathcal{Y} \tag{2.3}$$

Substituting the orthonormal constraint imposed on D in Eq: 2.3, we arrive at:

$$\mathcal{Y}^* = arg\min_{\mathcal{Y}} -2\mathcal{X}^T D\mathcal{Y} + \mathcal{Y}^T \mathcal{Y} \tag{2.4}$$

We note that the above equation is quadratic *wrt* $\mathcal{Y}$, and has a single minima which is also it's global minima. We solve for the value of $\mathcal{Y}$ by equating the gradient of 2.4 to 0.

$$\nabla_{\mathcal{Y}} (-2\mathcal{X}^T D\mathcal{Y} + \mathcal{Y}^T \mathcal{Y}) = 0 \tag{2.5}$$

We therefore obtain the encoding matrix:

$$\mathcal{Y} = D^T \mathcal{X} \tag{2.6}$$

Till now we have proved that if D is the *orthonormal* matrix that is responsible for inverting the transformation, then the transpose of this matrix is responsible for the encoding.

We can verify the above since $\mathcal{X} - D(D^T(\mathcal{X})) = 0$ minimizes our objective function. We know that D and D^T form the transform pair, how do we estimate the values of D? To answer this, we reframe the objective function as the minimzation of the *Frobenius norm* of the error matrix:

$$D^* = arg \min_D \sum_{i,j} (\mathcal{X}_j^{(i)} - r(\mathcal{X}_j^{(i)}))^2 \tag{2.7}$$

where $r(x) = DD^T(x)$, the transformation and inversion function. Let us assume that we wish to preserve only one component, then the objective is reduced to:

$$d^* = arg \min_d \sum_i \| \mathcal{X}^{(i)} - dd^T \mathcal{X}^{(i)} \|_2^2 \tag{2.8}$$

The encoder d is a single vector of size $(1, m)$ and m is the number of dimensions in $\mathcal{X}$, with the constraint subject to $\| d \|_2 = 1$. The frobenius norm of matrix A can also be written as: $\| A \|_F = \sqrt{Tr(AA^T)}$. Using this representation and rearranging the terms in Equation 2.8 we get:

$$d^* = arg \min_d Tr(\mathcal{X} - \mathcal{X}dd^T)^T (\mathcal{X} - \mathcal{X}dd^T) \tag{2.9}$$

Expanding and eliminating the terms that do not depend on d, Equation 2.9 simplifies to:

$$d^* = arg \min_d -2\, Tr(\mathcal{X}^T \mathcal{X} dd^T) + Tr(\mathcal{X}^T \mathcal{X} dd^T dd^T) \tag{2.10}$$

Using the orthonormal constraint on the values of d and inverting the negative *arg* min to *arg* max we get a result of the form:

$$d^* = arg \max_d Tr(d^T \mathcal{X}^T \mathcal{X} d) \tag{2.11}$$

Eq: 2.11 uses the property that the Trace of a matrix product is invariant to cyclic changes in the order in which the matrices undergo multiplication. The solution to Equation 2.11 can be obtained by letting d be the eigenvector corresponding to the largest eigenvalue of $\mathcal{X}^T \mathcal{X}$. This also satisfies the constraint $\| d \|_2 = 1$ since eigenvectors of a real symmetric matrix are orthogonal. This concludes a brief proof for using eigenvectors as transformation matrices.

2.1.2 Application

PCA is a fairly simple method which yields interpretable results without consuming too much compute time and memory. It can deal with high dimensional data better than many other techniques such as *t-SNE*. Hyperspectral bands have high correlation among

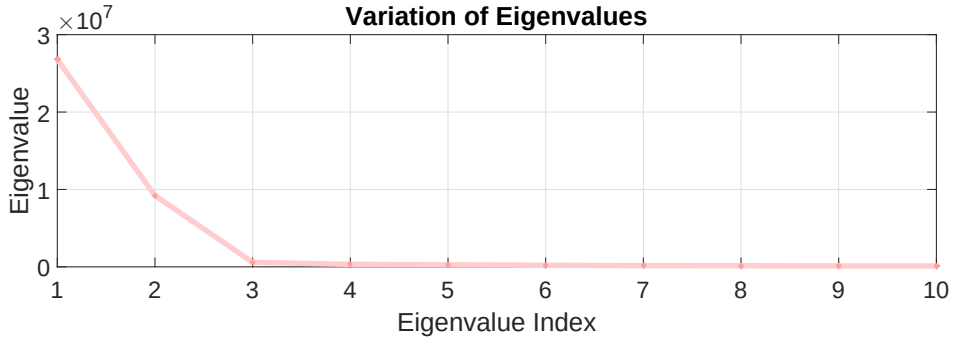


Figure 2.2: Variation of eigenvalues after computing the PCT of Indian Pines

neighbouring bands, which leave them unsuitable for direct application in Machine Learning algorithms. PCA on the other hand, considers the correlation among the bands as information content, and projects this variation along components which are uncorrelated. Advantages of uncorrelated data include faster convergence when supplied as input to machine learning algorithms [17]. A small proof as to why correlated inputs can slow down learning is as follows:

Let $\mathcal{X}$ be our input data, where $\mathcal{X}_j$ indicates the j^{th} component of $\mathcal{X}$. Let $\mathcal{X}_{j+1} = k * \mathcal{X}_j$, creating perfect positive correlation between the two variables. Our target data $\mathcal{Y}$ is continous, so we can formulate the regression problem as $\mathcal{X} * W = \mathcal{Y}$. Here, W represents the learnable parameters for our regression model. To compute W, we take the pseudo-inverse of $\mathcal{X}$: $W = (\mathcal{X}^T\mathcal{X})^{-1}\mathcal{X}^T\mathcal{Y}$ which is defined iff $\mathcal{X}^T\mathcal{X}$ is a non singular matrix. If our variables are correlated as is our case, $|\mathcal{X}^T\mathcal{X}| = 0$, which renders the closed form solution invalid. In the case where we wish to estimate the parameters in an iterative way, such as using an ANN model, convergence takes a huge time.

PCA helps side-step this problem by decorrelating the samples which is performed by projecting the points along orthogonal axes. However, care needs to be taken to not directly use these transformed samples as input to machine learning algorithms, due to the large difference in variance (eigenvalues) along those bands. A large eigenvalue, as shown in Figure 2.2 indicates high variance of samples along that principal component. This leads to the loss contour as a function of weights to be highly elongated which is shown in Figure 2.3. To understand why the loss function is elongated, let us assume $\mathcal{X}_1$ to be in the range [-1 to 1] and $\mathcal{X}_2$ to be in the range [0 to 1000]. As always, we let $\mathcal{Y} = W * \mathcal{X}$. If we compute the gradient of $\mathcal{Y}$ *wrt* W, $\frac{d\mathcal{Y}}{dW} = \mathcal{X}$. Thus, the gradient of our loss, which is directly proportional to $\frac{d\mathcal{Y}}{dW}$, is sensitive to the range of our inputs. Large values of $\mathcal{X}_2$ relative to the other inputs can cause oscillation in gradient descent as shown in Figure 2.3.

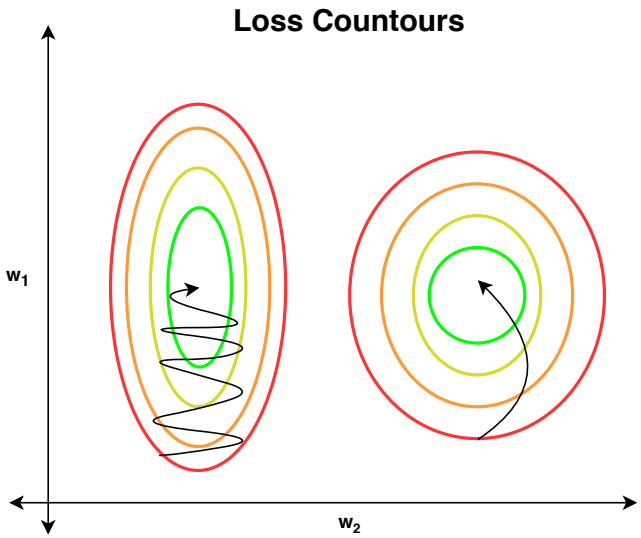


Figure 2.3: Effect of input standardization on the loss contour.
(L): Unscaled inputs (R): Scaled inputs

Due to the scale of $\mathcal{X}_2$, tiny changes in W_2 have the same effect on our loss as compared to moderate swings in W_1. The effect of uniform standardization of our input variables can be seen in the right contour. Larger learning rates can be used, with smoother convergence to the minima.

We have seen that Principal Component Analysis with scaling of inputs is suitable for dimensionality reduction tasks. However, certain limitations [45] exist with this technique.

1. PCA is optimal only for the task that involve learning simple manifolds. PCA involves a linear transformation of the original samples $\mathcal{X}$, and cannot model manifolds having non-linearity effectively. An example of this can be seen in Figure 2.4 (L), which is a toy dataset consisting of two classes arranged in a half-moon pattern. If we consider the the highlighted point of class 0, the distance to another point of the same class is much larger than the distance to a point of class 1. Yet, the euclidean distance between the points with dissimilar classes will be smaller than the ones with the same class on projecting them on the principal components. The principal component in this case will be the *X-axis*, explaining the maximum possible variance in the dataset. We can clearly see for this example that the projection of points along the principal components fails to provide insight into the manifold. Thus, our data cannot be represented as a linear combination of the principal components effectively.

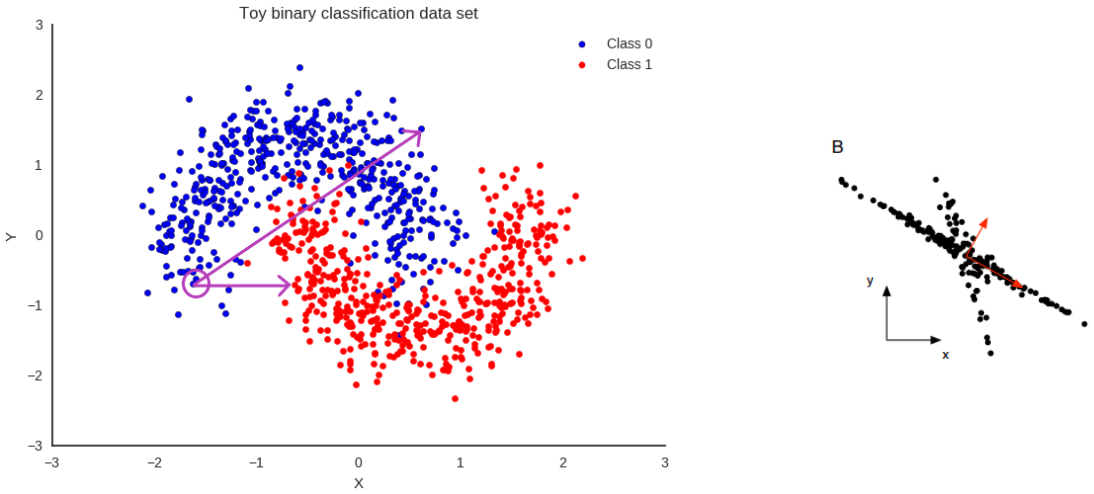


Figure 2.4: Toy Non-linear datasets highlighting the shortcomings of PCA [45]

2. PCA makes a strong assumption that the principal components are orthogonal. As can be seen in Fig 2.4 (R), this assumption may not always hold true. The principal component axis aligns along the longer arm of the cross, but due to orthnormal constraint, the second Principal component explains lower variance as compared to the scenario where the principal components had no orthonormal constraints.

3. The objective function of PCA rotates the axis in such a way that the Principal component explains maximum variance of the samples projected along it. This causes pairwise distant points in $\mathcal{X}$ to remain far apart in $\mathcal{Y}$. However, it is not necessary for nearby points in $\mathcal{X}$ to remain close together in $\mathcal{Y}$. This leads to the PCA focussing more on *retaining the global structure*. For manifold learning algorithms, the goal should be to retain *local structure.* Other algorithms thus do a more suitable role in learning the manifold in comparison to the Principal Component Analysis.

4. PCA considers correlated data as a source of information. While this is true with Hyperspectral Imagery due to the high correlation between neighbouring bands, noise corruption also occurs along contiguous bands, thus making noise correlated. Fortunately there are ways to counter this, such as applying the Minimum Noise Fraction algorithm *MNF*, which applies PCA initially on the noise signal alone. This causes noise to be uncorrelated once the Hyperspectral Image is projected along the Principal Components corresponding to noise. PCA is chained with the output of the previous step to obtain our final transformation, thus eliminating the influence of noise.

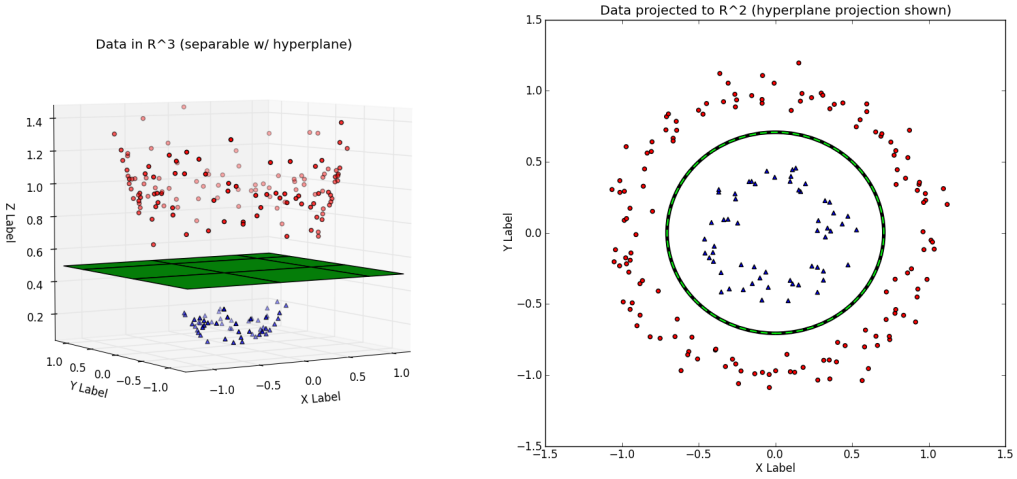


Figure 2.5: Non-linear decision boundary in $\mathbb{R}^m$ can be converted into a linear boundary in $\mathbb{R}^n$ where $m < n$. https://www.eric-kim.net/eric-kim-net/posts/1/kernel_trick.html

2.1.3 Kernel-PCA

Kernel Principal Component Analysis [44] is a technique that effectively models non-linearities in the data. It is based on the kernel trick proposed by [10] which introduces the Gaussian, Polynomial and the Cosine kernel as a transformation to higher dimensions. The advantages of such a transformation are shown in Figure 2.5. Figure 2.5 (R) shows a dataset of concentric circles, which cannot be separated into two distinct regions. Let us define a transformation $(x, y, z) = (x, y, x^2 + y^2)$, which transforms the dataset from Figure 2.5 (R) to Figure 2.5 (L). It is evident that now a linear technique can help identify two distinct regions which can now be used for dimensionality reduction. Apart from the standard kernels, we can define new ones provided they satisfy *Mercer's Condition*, which states that our Kernel inner product matrix should be **positive definite.** This requires that our kernel inner product matrix $\mathcal{K}$ be symmetric. The magic behind kernel trick lies in the fact that we do not need to define the transform ϕ that maps x from $\mathbb{R}^m$ to $\mathbb{R}^n$; the inner-product between all two points in $\mathbb{R}^n$ is sufficient. In fact, our points can be projected into ∞ dimensions yet have a inner product defined between pairwise points. This fact is exploited in the Gaussian Kernel. To see how this applies to Gaussian kernels, we begin by defining it as:

$$K(x_i, x_j) = e^{-\frac{(x_i - x_j)^2}{2\sigma^2}} \tag{2.12}$$

On expanding the square term:

$$K(x_i, x_j) = e^{-\frac{(x_i^2 + x_j^2 - 2x_i x_j)}{2\sigma^2}} \tag{2.13}$$

We then consider the squared terms separately as some constant c, so now our equation is reduced to:

$$K(x_i, x_j) = c * e^{(2x_i x_j)} \tag{2.14}$$

Using the taylor series expansion for $e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots$, we can write Equation 2.14 as:

$$K(x_i, x_j) = c * (1 + \frac{2x_i x_j}{1!} + \frac{(2x_i x_j)^2}{2!} + \frac{(2x_i x_j)^3}{3!} + \ldots) \tag{2.15}$$

Where $c = e^{-\frac{(x_i^2 + x_j^2)}{2\sigma^2}}$. We can split the terms contaning $x_i x_j$ as product of individual x_i and x_j:

$$K(x_i, x_j) = e^{-\frac{(x_i^2 + x_j^2)}{2\sigma^2}} * (1 * 1 + \sqrt{\frac{2}{1!}} x_i * \sqrt{\frac{2}{1!}} x_j + \sqrt{\frac{2^2}{2!}} x_i^2 * \sqrt{\frac{2^2}{2!}} x_j^2 + \ldots) \tag{2.16}$$

We observe that Eq: 2.16 can be rewritten as an inner product between two transformed points as $K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$ where $\phi(x) = e^{-\frac{x^2}{2\sigma^2}}(1, \sqrt{\frac{2}{1!}}x, \sqrt{\frac{2^2}{2!}}x^2, \sqrt{\frac{2^3}{3!}}x^3, \ldots)$. This proof shows that as long as $K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$ is *defined and is positive definite*, the transformation from x to $\phi(x)$ can be unknown and $\phi(x)$ can be of infinite dimensions.

We will now describe briefly the use of kernels in *K-PCA*. A detailed theoretical background behind this technique can be found in [44]. The crux of the algorithm lies in expressing the eigenvector u_a as a linear combination of the sample points x_i:

$$u_a = \sum_{i=1}^{N} \alpha_i^a x_i \tag{2.17}$$

Where u_a, $a \in \{1, m\}$, is the a^{th} eigenvector, α_i is a scalar quantity, and x_i is a $[m, 1]$ sample from $\mathcal{X}$ where $\mathcal{X} \in \mathbb{R}^{m\,x\,N}$. The value of α^a is estimated from Equation 2.18:

$$K\,\alpha^a = \widehat{\lambda_a}\,\alpha^a \tag{2.18}$$

Thus, α^a is the a^{th} eigenvector of the Kernel Matrix $\mathcal{K}$ which is normalized such that the constraint $\| u_a \| = 1$ is satisfied. This result is intuitive when compared to the ordinary Principal Component Analysis, where we compute the eigenvectors of the covariance matrix $C = \frac{X^T X}{N}$. Now C is replaced by $\mathcal{K}(x_i, x_j) = \phi(x_i)^T \phi(x_j)$ and the eigenvector obtained determines the linear combinations of the data samples in $\mathcal{X}$.

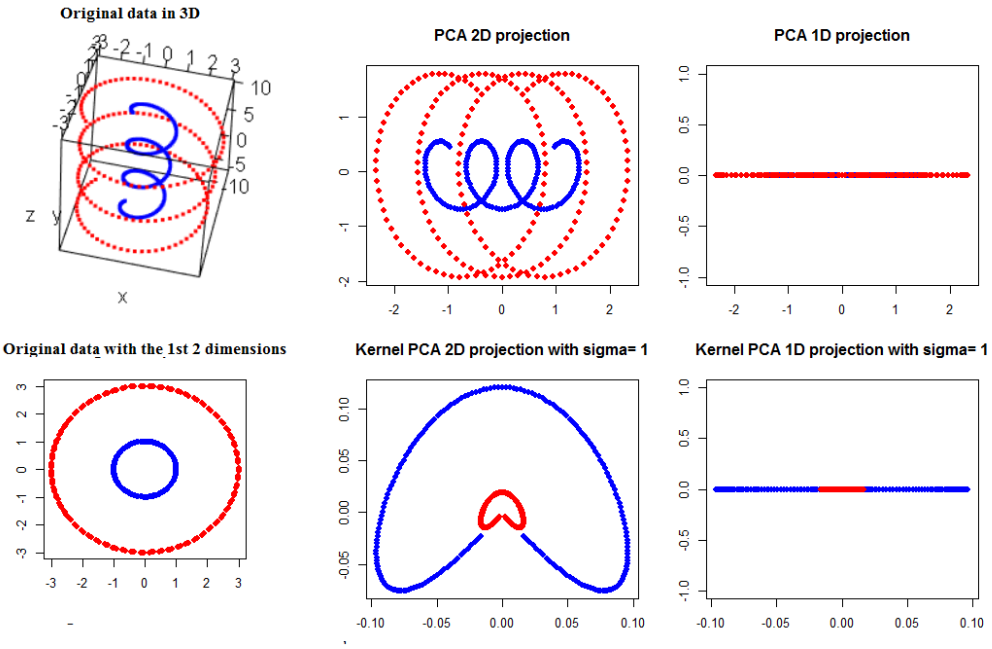


Figure 2.6: Comparison between manifold learnt by PCA and Kernal PCA
https://rpubs.com/sandipan/197468

The biggest advantage presented by the Kernel Principal Component Analysis is that it transforms our data in a non-linear manner such that we can then use linear techniques to embed the data in $\mathbb{R}^n$. We then assume that our embedded data reasonably shares characteristics possessed by the high-dimensional dataset.

Figure 2.6 highlights the key difference between Kernel Principal Component Analysis and traditional PCA. The two helices (indicated in red and blue) share a common axis. Computing a PCA on them projects points along the Principal Component alinged along this axis. This is verified by observing the 1D and 2D projections. On the other hand, k-PCA uses a Gaussian Kernel to first compute the pairwise similarity between all samples. Thus the similarity between values among blue samples will be higher than among the red samples, since the helix radius of blue samples is smaller. This inversion produced by the gaussian kernel helps in understanding why the blue bands explain more variance when projected along the principal components in k-PCA.

While Kernel PCA is a powerful technique that introduces non-linearities in the feature transformation pipeline, it has its limitations. K-PCA has a space complexity $O(n^2)$ and time complexity $O(n^3)$ [53]. This prevents the user from applying K-PCA to very large datasets. Development of powerful algorithms that demonstrate better manifold learning capabilities in comparison to Kernel PCA has limited it's scope for practical applications.

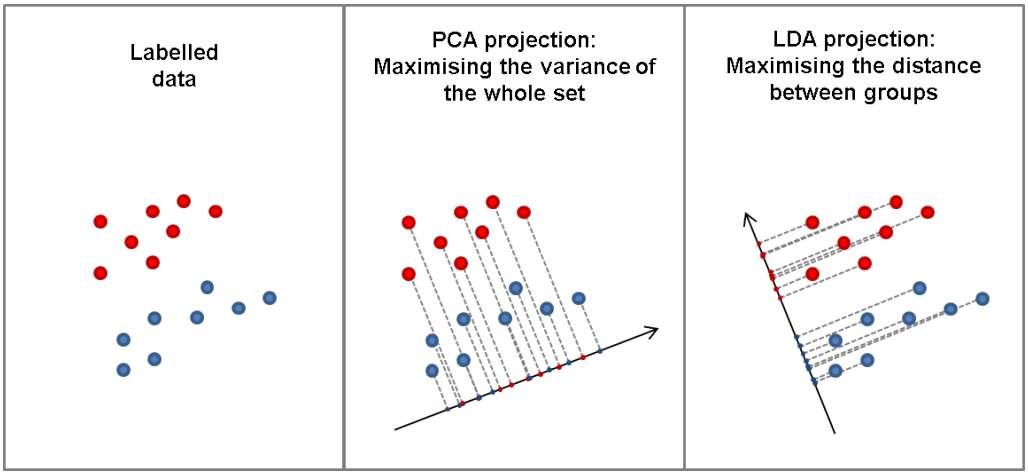


Figure 2.7: Visualizing the difference between PCA and LDA on labelled dataset.
https://www.idtools.com.au/classification-nir-spectra-linear-discriminant-analysis-python/

2.2 Linear Discriminant Analysis

The Linear Discriminant Analysis is a supervised approach for dimensionality reduction which encourages clustering of sample points. The essence of the algorithm is to project points along axes that preserve the inter-class separation of the samples in $\mathcal{X}$. To maintain the separation, LDA maximizes the inter-class mean of the sample points and minimizes the intra-class variance within the points. This is pictorially depicted in 2.7.

2.2.1 Theory

In this section, we will review the mathematical foundation[1] behind the two class LDA. We start by assuming that our samples for each class come from a gaussian distribution. Like before, we let $\mathcal{X}$ and $\mathcal{Y}$ be our original and reduced space respectively, where $\mathcal{Y} = W^T * \mathcal{X}$. Our goal is to find the hyperplane W such that the two classes are well resolved in $\mathcal{Y}$, as shown in Figure 2.8. This requires us to define an objective to maximize, such as separation of means between the two classes. Let $\mu_i = \frac{1}{N_i} \sum_{x \in \omega_i} x$ and $\tilde{\mu_i} = \frac{1}{N_i} \sum_{y \in \omega_i} y$. Here, μ_i indicates the class mean for $\mathcal{X}$ and $\tilde{\mu_i}$ indicates the class mean for $\mathcal{Y}$. If we substitute the value of $\mathcal{Y} = W^T * \mathcal{X}$, we obtain the relation between the true mean and projected mean as:

$$\tilde{\mu_i} = W^T * \mu_i \tag{2.19}$$

Our objective function is then defined to be:

$$\mathcal{J}(w) = |\tilde{\mu_1} - \tilde{\mu_2}| = |W^T(\mu_1 - \mu_2)| \tag{2.20}$$

[1]Referred from : CSCE 666 Pattern Analysis | Ricardo Gutierrez-Osuna | CSE@TAMU

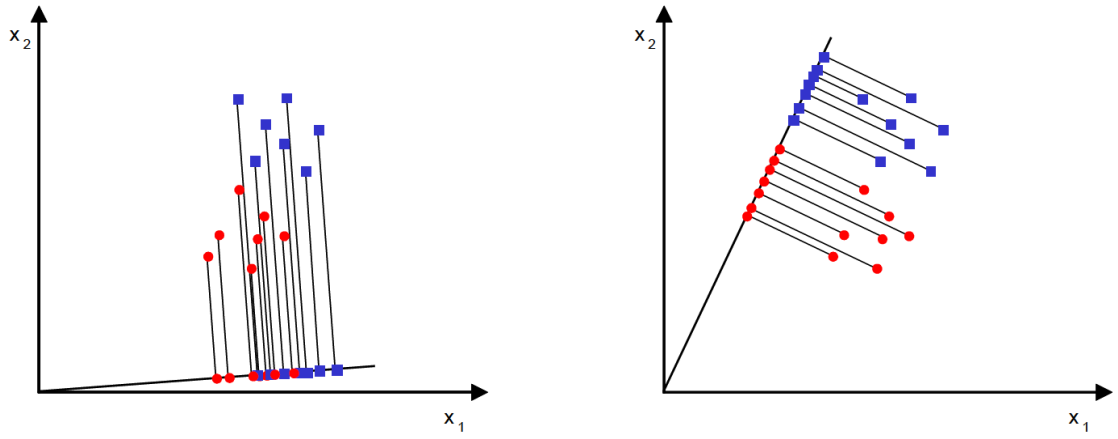


Figure 2.8: Choosing the projection W such that the two classes are resolved in $\mathbb{R}^n$.

Unfortunately, Equation 2.20 is not a good objective for minimization, since it does not take into account the intra-class variance. If we consider the objective of maximizing $|\tilde{\mu_1} - \tilde{\mu_2}|$, the resultant projection has partial overlap between classes. Incorporating the intra-class variance does a better job of separating the two classes. This can be seen in Figure 2.9. Fisher's solution proposes the use of *scatter* to fix the problem. The scatter is defined as $\tilde{s_i}^2 = \sum_{y \in \omega_i} (y - \tilde{\mu_i})(y - \tilde{\mu_i})^T$. The modified objective function to maximize now is:

$$\mathcal{J}(w) = \frac{|\tilde{\mu_1} - \tilde{\mu_2}|^2}{\tilde{s_1}^2 + \tilde{s_2}^2} \tag{2.21}$$

Intuitively, the function maximizes inter-class separation and minimzes intra-class variance. Similarly, we define scatter in $\mathcal{X}$ as $s_i = \sum_{x \in \omega_i} (x - \mu_i)(x - \mu_i)^T$. The within class scatter matrix S_w is the sum of $s_1 + s_2$. Substituting $y = w * x$ and the value of S_w in $\tilde{s_i}^2$, we get the value of the denominator of Equation 2.21:

$$\tilde{s_1}^2 + \tilde{s_2}^2 = w^T S_w w \tag{2.22}$$

The numerator is can be simplified into:

$$|\tilde{\mu_1} - \tilde{\mu_2}|^2 = w^T S_B w \tag{2.23}$$

S_B is the *between-class scatter* $(\mu_1 - \mu_2)(\mu_1 - \mu_2)^T$. Now we can express the Fisher criterion in terms of the projection vector w as:

$$\mathcal{J}(w) = \frac{w^T S_B w}{w^T S_w w} \tag{2.24}$$

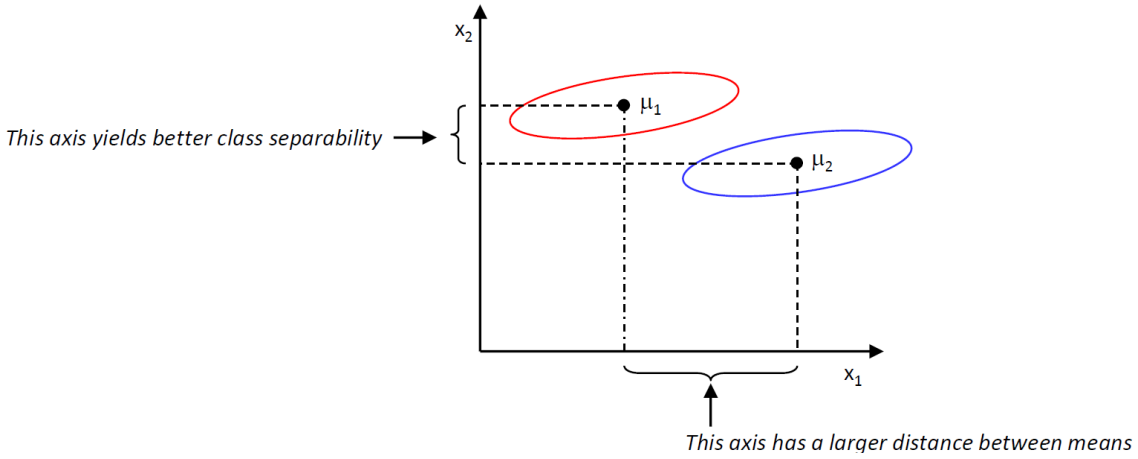


Figure 2.9: The choice of objective function results in vastly different projections.

Computing the derivative of Equation 2.24 *wrt* w and simplifying, we reduce the equation to $S_w^{-1} S_B\, w = \mathcal{J} w$. Hence, computing the value of w is now posed to us as an eigendecomposition problem. The value $w*$ that maximizes $\mathcal{J}(w)$ is given by Equation 2.25. Intuitively, $(\mu_1 - \mu_2)$ represents the vector that connects the means of the two classes. The scatter term S_w^{-1} transforms this difference of means vector to account for the intra-class variance of the dataset.

$$w^* = S_w^{-1}(\mu_1 - \mu_2) \tag{2.25}$$

2.2.2 Applications

We have seen that LDA being a supervised technique can take advantage of class labels to project points along components that maintain separability in lower dimensional space. However, does the information of class labels always provide better results in comparison to unsupervised techniques such as PCA? Unfortunately, there are instances where class labelling causes LDA to deliver worse performance compared to PCA. Figure 2.10 highlights such a scenario where PCA performs better compared to LDA. In the given dataset, the two classes are better explained by variance than their means, which LDA is unable to capture. The first principal component of PCA points along the direction of maximum variance, and hence provides better discriminatory power. Another major limitation with LDA is that it can provide atmost k - *1* components, where k is the number of classes present in the dataset. Hence, if we wish to embed 3 classes originally in $\mathbb{R}^m$ space, we can only do so in $\mathbb{R}^n$ space where $n \leq 2$. LDA assumes that samples belong to classes which are represented by unimodal Gaussian distributions; an assumption not valid for hyperspectral datasets. LDA and PCA being linear techniques do a poor job in learning non-linear manifolds.

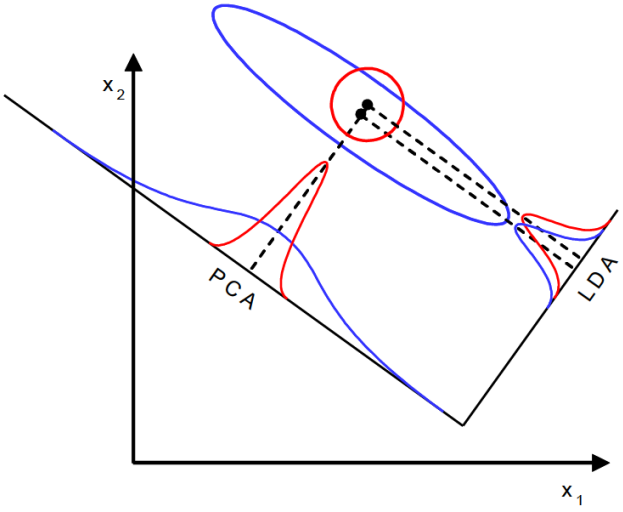


Figure 2.10: LDA fairs poorly in comparison to PCA since the main discriminatory power for the dataset liess in the within-class variance, and not their means

2.3 Laplacian Eigenmaps

The previous approaches for Dimensionaliy Reduction were more focussed on preserving large distances in the embedding, at the cost of neglecting small distances. The main aim of manifold learning is to understand the local spread of points in the neighbourhood. Hence, subsequent algorithms which we will discuss put a major emphasis on preserving the local structure over the global structure.

2.3.1 Theory

Laplacian Eigenmaps is a graph based method to preserve local structure of the manifold. It encourages the discovery of natural clusters in the dataset. An added advantage is that the algorithm is more robust to outliers. The algorithm constructs a Adjacency matric from samples in the dataset. While the original paper [3] provides two methods to construct the adjacency matrix; using $d < \epsilon$ neighbours and *k-Nearest Neighbours* approaches, we will explore the latter in this dissertation. In Graph theory, we will frequently encounter terms such as $\mathcal{L}$: The Graph Laplacian, which is defined as $\mathcal{L} = \mathcal{D} - \mathcal{A}$ where $\mathcal{D}$ is a diagonal matrix whose entries $\mathcal{D}_{ii}$ represent the number of neighbours a sample $\mathcal{X}_i$ has. The Adjacency matrix $\mathcal{A}$ indicates if two vertices $\mathcal{V}_i$ and $\mathcal{V}_j$ have an edge $\mathcal{E}$ between them, where edge $\mathcal{E} \in \{0, 1\}$ or could be defined as a heat kernel; $\mathcal{E} = e^{-\frac{\|x_i - x_j\|^2}{k}}$. The objective to minimize for the Laplacian Eigenmaps is:

$$\mathcal{J}(y) = \sum_{i,j}(y_i - y_j)^2 W_{ij} \tag{2.26}$$

The objective in Equation 2.26 promotes clustering of connected points. If $W_{ij} = 1$, the algorithm will try to minimize the euclidean distance between their low dimension embeddings in $\mathbb{R}^n$. However, if two vertices are not connected in the graph ($W_{ij} = 0$), the algorithm does not necessarily keep them distant. Equation 2.26 can also be written in terms of $\mathcal{L}$, $\mathcal{D}$ as:

$$\begin{aligned}\sum_{i,j}(y_i - y_j)^2 W_{ij} &= \sum_{ij}(y_i^2 + y_j^2 - 2y_i y_j)W_{ij} \\ &= \sum_i y_i^2 D_{ii} + \sum_j y_j^2 D_{jj} - 2\sum_{ij} y_i y_j W_{ij} \\ &= 2\mathcal{Y}^T L\mathcal{Y}\end{aligned} \tag{2.27}$$

The Laplacian Matrix $\mathcal{L}$ is symmetric and hence postitive semi-definite. The solution to equation 2.27 reduces to finding $\mathcal{Y}^*$ that minimizes it. A sharp observer will notice that one solution of Equation 2.27 is the eigenvector $[1, 1, \ldots]^T$. The eigenvalue for this eigenvector is 0. However, this solution is not very useful to us. Also, we impose an additional constraint on the minimization problem, $\mathcal{Y}^T D\mathcal{Y} = 0$, which helps fix the scale of $\mathcal{Y}$. To minimize, we compute the derivative of equation 2.27 with the lagrangian constraint:

$$\begin{aligned}\nabla_{\mathcal{Y}}((\mathcal{Y}^T\mathcal{L}\mathcal{Y}) - \lambda(\mathcal{Y}^T\mathcal{D}\mathcal{Y})) = \mathcal{L}\mathcal{Y} - \lambda\mathcal{D}\mathcal{Y} = 0 \\ \mathcal{L}\mathcal{Y} = \lambda\mathcal{D}\mathcal{Y}\end{aligned} \tag{2.28}$$

If $\mathcal{X} \in \mathbb{R}^{Nxm}$, then our Laplacian Matrix $\mathcal{L}$ is of the size *[N, N]*. Hence, there will be N eigenvector solutions to $\mathcal{Y}$. Let $\lambda_1, \lambda_2, \lambda_3, \ldots$ be the corresponding eigenvalues (where $\lambda_1 \leq \lambda_2 \leq \lambda_3 \ldots$). If we retain n components where $n << m$, then each x_i is mapped to a corresponding y_i where $y_i = \mathcal{Y}_j^{(i)}$; $j \in 1, \ldots, n$. Unlike in our previous methods, we do not project $\mathcal{X}$ along the eigenvectors $\mathcal{Y}$, since $\mathcal{Y}$ are the embeddings themselves. For the n-dimensional case, the objective function is modified to minimize: $Tr(\mathcal{Y}^T\mathcal{L}\mathcal{Y})$.

2.3.2 Applications

Laplacian Eigenmaps is the first unsupervised technique we have seen that emphasizes on retaining the local structure, albeit at the expense of the global arrangement of our data.

It excels at discovering clusters within the dataset, and embedding them into lower dimensions. However, this algorithm has certain limitations which has lead to development of newer algorithms. Given a new sample, LE needs to compute the pairwise distance between all pre-existing samples and the new sample, which could potentially lead to a different adjacency matrix, and the need to recompute all the eigenvectors for the dataset. The absence of parametric form prevents fast computation of embeddings for unseen samples. For large datasets, the embedding computation is prohibitively expensive, since the algorithm involves eigen-decomposition, which has a time complexity of $O(n^3)$. The algorithm falls short in performance to other Dimensionality Reduction algorithms if the dataset lacks natural clusters. This algorithm also assumes a uniform density over all the datapoints; an assumption unlikely to hold for real world datasets. 'Fringe' samples for a given class are less likely to be sampled from that class distribution. However, these samples could show a higher likelihood of being obtained from neighbouring classes, where they could also be involved in the adjacency matrix computation for samples of that class, hindering with the embedding process. Laplacian Eigenmaps may also behave unpredictably for manifolds having boundaries [3].

2.4 Locally Linear Embedding

In this section, we cover another graph based method, called Locally Linear Embedding [42]. The *LLE* algorithm is based on learning a weight vector W_i for each sample such that the neighbours of this sample are able to reconstruct it. Like *LE*, it uses the notion of k-Nearest Neighbours to reconstruct the sample under observation. *LLE* is a non-linear Dimensionality Reduction technique which relies on optimal sampling of the manifold such that in small regions, the manifold can be linearly approximated.

2.4.1 Theory

LLE can be divided into two phases; estimating the weights $\mathcal{W}$ for all the samples, and fixing the weights and obtaining $\mathcal{Y}$. The weight matrix helps to learn features in the embedding, which are then used mimic the manifold in lower dimensions. The algorithm relies on an error term to minimize, which is defined as:

$$\mathcal{E}(W) = \sum_i | x_i - \sum_j W_{ij} x_j |^2 \tag{2.29}$$

The neighbours can be assigned based on the *k-Nearest Neighbours* or considering all samples within ϵ distance as neighbours for a given sample. The constraints on W_{ij}

include $\sum_j W_{ij} = 1$ and $W_{ij} = 0$ if j is not a neighbour of i. The weights help to capture the samples spread in the local neighbourhood. It also is invariant to rotation, rescaling and translation. The solution to the values of W is reduced to least squares estimate of Equation 2.29, which can be written as:

$$\begin{aligned} \mathcal{E}(W) &= \sum_j W_j \, (x - \eta_j)^2 \\ &= \sum_{jk} w_j w_k C_{jk} \end{aligned} \tag{2.30}$$

C_{jk} is defined as the local covariance matrix $(x - \eta_j)(x - \eta_k)$. Solving 2.30 with the lagrangian constraint $\sum_j w_j = 1$, we obtain:

$$w_j = \frac{\sum_k C_{jk}^{-1}}{\sum_{lm} C_{lm}^{-1}} \tag{2.31}$$

The weight vector that we have obtained describes the structure of manifold in small regions. Using this weight vector, we can reconstruct the manifold in low dimensions by minimizing the reconstruction error keeping the weights fixed:

$$\sum_i | y_i - \sum_j w_{ij} y_j |^2 \tag{2.32}$$

The solution to this equation involves eigendecomposition by expanding the square term. We also fix the scaling of the embedding $\mathcal{Y}$ by keeping it's covariance matrix identity $(\frac{1}{N}\mathcal{Y}\mathcal{Y}^T = I_N)$.

2.4.2 Application

Embeddings through LLE can be easily expanded into multiple dimensions by incorporating subsequent eigenvectors without any extra calculations. Unlike in the case of Autoencoder or t-SNE (will be covered later), we do not need to compute the embeddings afresh when adding multiple dimensions. However, if we wish to obtain embeddings for new unseen samples, we need to recompute the weight vectors and hence the embeddings again. As with the other approaches, computational cost for generating low dimensional representation increases as the size of our dataset increases. A common tool used in all the algorithms we have discussed is the eigendecomposition, which does not scale well with the size of our data. LLE also tends to cluster samples around the mean, with outliers and noise located far away so as to balance the variance explained by the eigenvectors. A comparison between LLE and various algorithms including t-SNE is given in [32]. While

LLE is a powerful tool for non-linear dimensionality reduction, it suffers from problems such as ill-conditioned eigenproblems [7]. LLE is also ineffective if the manifold is highly non-linear. An assumption of LLE includes that local manifolds are linear even if global manifolds are non-linear. The linearity assumption is true only if there is higher sampling of points along the non-linear parts of manifold, which helps in approximating the sample as a weighted sum of its neighbours. Hence compared to other techniques, LLE requires denser connections between points to establish local linearity assumption.

Till now we have seen techniques which involve eigendecomposition as an optimization method. Algorithms that we will discuss subsequently will focus more on using Stochastic Gradient Descent as an optimization techniques. A few other algorithms that are important and have not been covered in the dissertation include ISOMAP [47], Sammon Mapping [43], which are based on similar principles to the algorithms already discussed. We now move our focus to two popular algorithms, t-SNE and UMAP, which are state of the art for non-linear dimensionality reduction and data visualization.

2.5 t - Distributed Stochastic Neighbor Embedding

One of the most popular techniques currently in use, t-SNE [32] is an award winning algorithm which helps in two or three dimensional visualizations of manifolds embedded in high dimensional space. The crux of the algorithm is to encode the sample distances into probabilistic relation between the samples. We then imitate the probability values in the original space in the reduced space. t-SNE is proposed as an improvement over the SNE algorithm [22], which helps eliminate the crowding problem and improves convergence.

2.5.1 Stochastic Neighbour Embedding

Earlier techniques for Dimensionality Reduction preserved large distances in the embedding or were useful for linear manifolds. With SNE, we compute a conditional probability distribution over all the samples:

$$p_{j|i} = \frac{exp(-\| x_i - x_j \|^2 / 2\sigma_i^2)}{\sum_{k \neq i} exp(-\| x_i - x_k \|^2 / 2\sigma_i^2)} \tag{2.33}$$

Let i be the sample under consideration. The for each $j \neq i$ we determine the probability of j being the neighbour of i. To do this, we assume a gaussian probability distribution centred around the sample under consideration, and consider the euclidean distance between this sample and other points in the dataset. This is followed by a softmax as shown in Equation 2.33. Since we are computing the probability of j being a neighbour of i,

we assume $p_{i\,|\,i} = 0$. SNE uses a novel method to estimate the variance of the probability distribution for each sample x_i, using something called as *Perplexity* which we will cover in detail later. Similar to the method of computing the probabilities in $\mathbb{R}^m$, we compute $q_{i\,|\,j}$ in $\mathbb{R}^n$.

$$q_{j\,|\,i} = \frac{exp(-\|\,y_i - y_j\,\|^2)}{\sum_{k \neq i}\, exp(-\|\,y_i - y_k\,\|^2)} \tag{2.34}$$

We notice the absence of σ in Equation 2.34 since when computing the low dimensional embeddings, σ just provides a scaling effect. Changes in σ will not alter the manifold, only affecting the increase/decrease in distances between the embeddings $\mathcal{Y}$. Now that we have generated our probabilities p and q in $\mathcal{X}$ and $\mathcal{Y}$ respectively, we need to reduce the distance between the two distributions. One effective tool to measure the dissimilarity between two distributions is the KL divergence, which is given as $KL(p\,\|\,q) = p\,log\,\frac{p}{q}$. Now we can define the cost function to minimize for SNE as:

$$C = \sum_i KL(\,P_i\,\|\,Q_i\,) = \sum_i \sum_j p_{j\,|\,i}\,log\,\frac{p_{j\,|\,i}}{q_{j\,|\,i}} \tag{2.35}$$

Since KL divergence is not symmetric, it is not a true distance metric. In our case, KL divergence will incur a high cost when $q_{j|i}$ is of a low value (embeddings are far apart) and $p_{j|i}$ has a high value (the points are actually close in the original space). Hence, our objective function tries to preserve the local manifold, keeping neighbourhood samples in $\mathcal{X}$ close together in $\mathcal{Y}$ too.

To compute the variance associated with the Gaussian for each sample, the authors define a new concept, *Perplexity (Perp).* This is defined as:

$$Perp\,(P_i) = 2^{H(P_i)} \tag{2.36}$$

$H(P_i)$ represents the Shannon Entropy of the distrubution P_i which is formulated as $H(P) = -\sum_i\, p_i\,log\,p_i$. We recall that for a uniform distribution the Shannon Entropy has the highest value, which indicates that there is maximum uncertainity in the output as all outcomes are equally likely. For a more deterministic distribution (sparse probability assignments), the Shannon Entropy has a smaller value, as fewer outcomes are possible. In our Gaussian distribution over each sample, the value of σ controls the spread of the distribution. As $\sigma \rightarrow \infty$, the gaussian conditional probability tends towards uniform distribution, having higher Shannon Entropy. For small values of σ, the conditional probability distribution becomes more deterministic, and as a consequence the Shannon Entropy value goes down. The choice of σ controls the number of neighbours a point

has. Hence by fixing the *Perplexity* to a fixed value, we are freezing the value of the Shannon Entropy our distribution has, and hence establish a loose control over the number of neighbours x_i could have by searching over the values of σ that would let the distribution match the perplexity. Perplexity has gives us direct control on the number of neighbours. An important question arises, why does each point have a separate σ? This arises from the uneven nature of manifolds, where we could have densely sampled regions as well as poorly sampled regions. For a high density of samples, we fix a smaller σ which in turn helps us to capture the local arrangement of points in space. For sparsely populated regions, we need to increase the value of σ to identify neighbouring points. Hence fixing *Perplexity* to be the same for all samples in $\mathcal{X}$ searches the manifold for neighbors adapting to the dense and sparse regions around the point under consideration. Since *Perplexity* monotonically increases with the value of σ, we can use Binary Search to estimate the values of σ_i where $i \in \{1, 2, 3, \ldots N\}$. The gradient obtained on differentiating 2.35 *wrt* y_i is:

$$\frac{\delta C}{\delta y_i} = 2 \sum_j (p_{j|i} - q_{j|i} + p_{i|j} - q_{i|j})(y_i - y_j) \tag{2.37}$$

The gradient in 2.37 can be interpreted as a spring between y_i and y_j. THe direction of application of force is along $(y_i - y_j)$. The stiffness (or force multiplier) is the mismatch between the conditional probabilities for samples in $\mathcal{X}$ and their embeddings in $\mathcal{Y}$. The gradient makes it easier for us to see why t-SNE encouranges local structures. If two samples are close in $\mathcal{X}$ and their embeddings are far apart in $\mathcal{Y}$, this implies $p_{i|j} + p_{j|i} \geq q_{i|j} + q_{j|i}$ and $(y_i - y_j)$ is a large quantity; thus the resultant attractive force has a strong magnitude.

2.5.2 t-SNE

t-SNE [32] was proposed as an improvement over the SNE algorithm, which overcomes the limitations of the traditional SNE by using:

1. Symmetric cost function: SNE uses conditional probabilities (Equation: 2.35) to define the cost function. The use of conditional probabilities implies $p_{i|j} \neq p_{j|i}$ in most of the cases. t-SNE on the other hand defines joint probabilities over p and q, which leads to simple gradients for optimization.
2. Cauchy distribution: t-Distribution (1 DoF) or Cauchy distribution was introduced to combat overcrowding as was present in SNE. t-Distribution with 1 degree of freedom has a heavier tail than Gaussian, thus allowing more space around a sample

to accomodate mid-range points. We will discuss more about the crowding problem in detail.

Instead of defining the cost function as Equation 2.35, we now use conditional probabilities:

$$KL(P||Q) = \sum_i \sum_j p_{ij} log \frac{p_{ij}}{q_{ij}} \tag{2.38}$$

Where, $p_{ij} = \frac{p_{i|j} + p_{j|i}}{2N}$ and $q_{ij} = \frac{exp(-||y_i - y_j||^2)}{\sum_{k \neq l} exp(-||y_k - y_l||^2)}$. Note that the computation of q_{ij} involves computing the softmax over entire q matrix and not over the rows of q. We could have defined p_{ij} similar to q_{ij}, but the outliers would then have no contribution towards the cost function as $p_{ij} \approx 0$. To counter this, the current definition of p_{ij} guarantees that $\sum_j p_{ij} \geq \frac{1}{2N}$, which boosts its contribution. The gradient of the joint distribution is similar to Equation 2.37 but much more easier to compute:

$$\frac{\delta C}{\delta y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j) \tag{2.39}$$

To understand the crowding problem, imagine a ten dimensional volume, which can accomodate at the maxmimum eleven points which are equidistant from each other whereas four mutually equidistant points form a tetrahedron in a 3-Dimensional space. Hence, representing distances faithfully in lower dimensions is impossible. A similar problem to the one mentioned above is the scaling of hypervolume as we increase dimensionality of our dataset. To quote from [32]

> "The volume of a sphere centred on a datapoint i scales as r^m, where r is the radius and m is the dimensionality of the sphere."
>
> - van Der Maaten and G. Hinton [32]

As we scale the dimensions (2D: πr^2, 3D: $k * \pi r^3 \ldots$), more hypervolume is available for our samples to occupy. As a consequence, when we reduce the dimensionality of our dataset, the same number of samples need to be accomodated in smaller areas, which introduces crowding problems for points that are mid-range from the sample under consideration. To circumvent this problem, the authors suggested the use of t-Distribution which has a heavier tail in comparison to the Gaussian function. The t-Distribution with 1 DoF is given as:

$$q_{ij} = \frac{(1 + ||y_i - y_j||^2)^{-1}}{\sum_{k \neq l} (1 + ||y_k - y_l||^2)^{-1}} \tag{2.40}$$

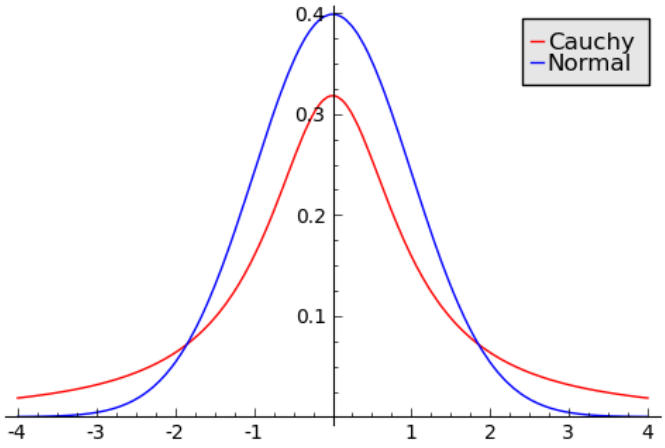


Figure 2.11: The Cauchy distribution (t - Distribution with 1 DoF) has a heavier tail than the Gaussian, which means that more probability mass is distributed away from the mean. [9]

Figure: 2.11 shows a visual difference between the two distributions. We can see that the Cauchy distribution provides more hypervolume for mid-distant samples and potentially avoids the crowding problem. Another attractive feature is that the squared decay of euclidean distances in Cauchy distribution minimizes the influence of distant clusters in the embedding. The complete algorithm us shown in Figure 2.12.

t-SNE has been a popular technique since it's publication in 2008, as it captures the relation beteen samples that form the manifold accurately in low dimensional spaces. Yet certain limitations exist particularly in it's application to Hyperspectral Imagery which we hope to overcome in this dissertation. While t-SNE has a parametric form [31], it relies on using Restricted Boltzmann Machines (derived from Markov Random Fields) to estimate the parameters, using Contrastive Divergence with Greedy layer wise training. In practice, training the network is considerably more complicated than it's traditional implementation which is why the use of parametric t-SNE is not widespread.

t-SNE uses euclidean distance to compute the probability associated with our samples. For highly complex manifolds, euclidean distance is not the optimal criteria for neighbourhood points selection as the choice of euclidea distance accompanies the assumption that our manifold could be divided into locally linear regions.

While applying t-SNE (`scikit-learn` implementation) on hyperspectral images, we found that the algorithm could not faithfully represent spatial relation between classes, which we suspect is because of the curse of dimensionality. Class A and B could be ad-

Algorithm 1: Simple version of t-Distributed Stochastic Neighbor Embedding.

Data: data set $\mathcal{X} = \{x_1, x_2, ..., x_n\}$,
cost function parameters: perplexity $Perp$,
optimization parameters: number of iterations T, learning rate η, momentum $\alpha(t)$.
Result: low-dimensional data representation $\mathcal{Y}^{(T)} = \{y_1, y_2, ..., y_n\}$.
begin
  compute pairwise affinities $p_{j|i}$ with perplexity $Perp$ (using Equation 1)
  set $p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$
  sample initial solution $\mathcal{Y}^{(0)} = \{y_1, y_2, ..., y_n\}$ from $\mathcal{N}(0, 10^{-4}I)$
  for t=*1* **to** T **do**
    compute low-dimensional affinities q_{ij} (using Equation 4)
    compute gradient $\frac{\delta C}{\delta \mathcal{Y}}$ (using Equation 5)
    set $\mathcal{Y}^{(t)} = \mathcal{Y}^{(t-1)} + \eta \frac{\delta C}{\delta \mathcal{Y}} + \alpha(t)\left(\mathcal{Y}^{(t-1)} - \mathcal{Y}^{(t-2)}\right)$
  end
end

Figure 2.12: t-SNE algorithm

jacent in one t-SNE embedding whereas they could be diagonally opposite in the second embedding. t-SNE being a non-convex optimization technique could have local minimas and thus variations in the data embedding, but spatial relation between classes should remain constant accross various runs of the algorithm. This point will be elaborated further in the methodologies chapter.

2.6 Uniform Manifold Projection and Approximation

In this section, we will briefly review the underlying principles behind UMAP, a general purpose Dimensionality Reduction technique [34]. A rigourous mathematical treatment is beyond the scope of this dissertation. Apart from the publication, an excellent *YouTube*[2] video by the author is available which beautifully explains the concept of UMAP. Images in this section have been taken from this video.

Previous techniques which we have seen either focus on maintaining the local or global structure. With UMAP, the authors attempt to preserve both. The idea behind UMAP involves explaing the underlying manifold using a simplicial complex which defines the neighbourhood of samples in our dataset. Figure 2.13 displays the simplicial complex which is used to build the topology. The Nerve theorem [12] states that building a simplical complex in a topology helps in recovering the topology information.

[2]Referred from : https://www.youtube.com/watch?v=nq6iPZVUxZU

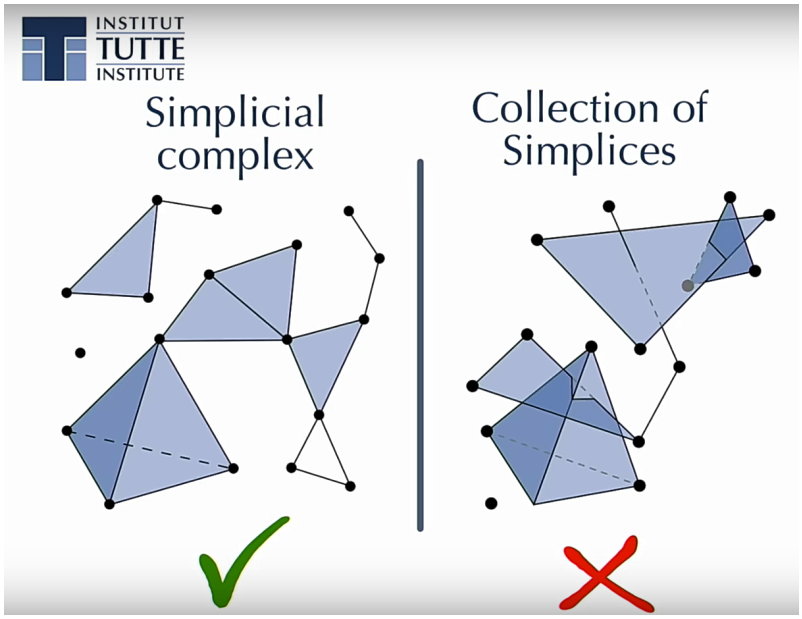


Figure 2.13: (L) shows a simplicial complex, which is similar to a clique in graphs. A non-intersecting combination of simplices make a complex, similar to Delauney triangulation for building topologies.
https://www.youtube.com/watch?v=nq6iPZVUxZU

Let us build a synthetic dataset that mimics a sinusoidal curve with uneven sampling. To construct a topology, we assume an ϵ-distance volume centered around our datapoints. A 2-simplex is constructed when volumes of two samples intersect, 3-simplex is obtained when volumes of three samples intersect and so on. This is shown in Figure 2.14. Notice that certain points in the topology are isolated and have no links to the connected regions. This is because of the assumption made that the points are uniformly distributed along the manifold. Unfortunately, this assumption is rarely true for real world datasets. To model the uneven distribution of data, the authors suggested to define a Reimannian Metric on the manifold to make this assumption true. Figure 2.15 shows the Reimannian manifold in blue. All points on the manifold are assumed to be uniformly distributed. This manifold consists of multiple patches (or ϵ-regions in this case) stitched together. On projecting individual patches into euclidean space $\mathbb{R}^n$, each patch could have it's own scale and distance measure. Therefore, certain regions could be more compact and have different notion of distance in the euclidean space depending on the sample density. With this projection, we have a notion of varying distance on the manifold. Note that the varying distance concept is common with t-SNE, where we use *Perplexity* to model the neighbourhood. Certain patches could have overlapping regions as shown in the figure. In such regions, the distance scale of both the patches hold true, and hence one could transition between them if the distance transformation τ is known.

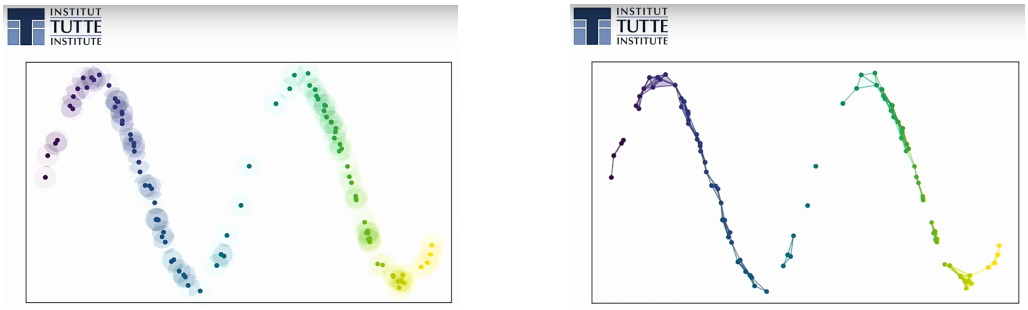


Figure 2.14: Computing the Simplical Complex for our dataset
https://www.youtube.com/watch?v=nq6iPZVUxZU

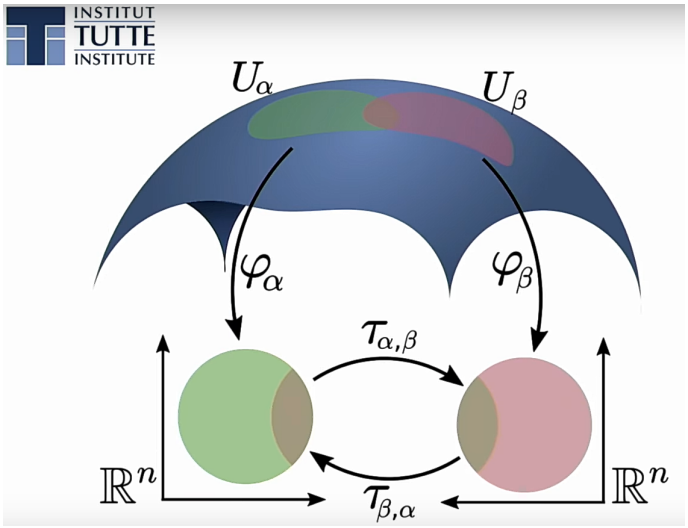


Figure 2.15: Transformation to Reimannian Manifold and Euclidean space
https://www.youtube.com/watch?v=nq6iPZVUxZU

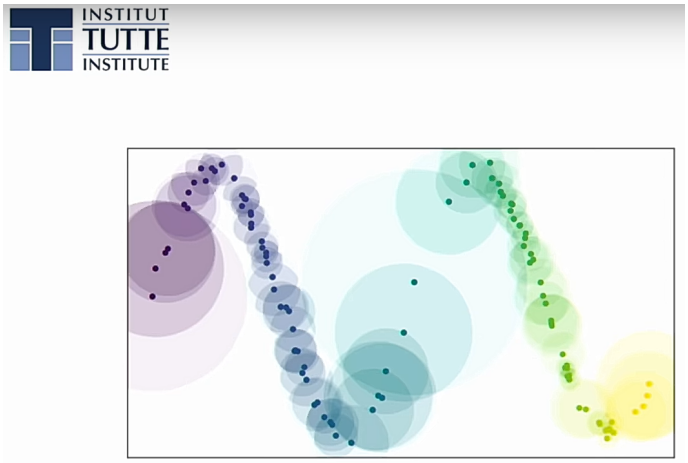


Figure 2.16: Using a Reimannian Metric, we could scale distances along the manifold depending on the sample density, similar to perplexity in t-SNE.
https://www.youtube.com/watch?v=nq6iPZVUxZU

Figure 2.16 shows the effect of using the Reimannian metric on our manifold, which results in a better construction of the simplicial complex. Instead of using crisp boundaries, samples could also have fuzzy boundaries. With the locally connected neighbourhood assumption, which states that no sample lies in isolation on the manifold, the ϵ-region extends to the nearest sample, from where the boundary starts becoming fuzzy. The edges that form the simplicial complex could be given a probabilistic fuzzy interpretation, where the weight on the edge is given by $f(\alpha, \beta) = \alpha + \beta - \alpha\beta$. Here, α and β indicate the fuzzy value of the ϵ-regions of the two samples between which there is an edge. Now that we have our simplicial complex, we could use it to reproduce our low dimensional embeddings. A similar graph could be computed in $\mathcal{Y}$ where the edges are associated with a fuzzy probabilistic number. To ensure that a similar topology is built in the lower dimensional space, we use cross entropy between the topologies in $\mathbb{R}^m$ and $\mathbb{R}^n$, where we minimize the difference between the probabilistic values of edges in $\mathcal{X}$ and $\mathcal{Y}$.

UMAP is a technique that is computationally faster than t-SNE, however in practice we found that the computation time scales up with the *number of neighbours* hyperparameter we supply to the algorithm. UMAP is a general dimensionality reduction technique unlike t-SNE which specializes in data visualization. UMAP can also be used with different distance metric apart from the standard euclidean distance. This allows UMAP to work better with categorical data other than t-SNE. On the other hand, like t-SNE, UMAP is not a parametric technique [23] which gives an input-output mapping between the original samples $\mathcal{X}$ in $\mathbb{R}^m$ and $\mathcal{Y}$ in $\mathbb{R}^n$. For hyperspectral dataset, we found that the embeddings given by UMAP were not interpretable, and we will elaborate more on this in the results section. Supervised UMAP has a very high tendency to overfit the train dataset, which could be misleading to users on its effectiveness on the test dataset for classification.

In the next section will have an elaborate discussion on Machine Learning, in particular neural networks, and discuss their applications to the field of dimensionality reduction and classification for remotely sensed imagery.

2.7 Autoencoders for Dimensionality Reduction

Autoencoders [21] are an unsupervised Deep Learning approach to Dimensionality Reduction. The main idea behind Autoencoders as a manifold learning tool is to have a bottleneck layer, which forces the network to learn the coordinate space of the underlying manifold. Let $\mathcal{X}$ be the input to our model, h (or z) be the low dimensional embedding and $\tilde{\mathcal{X}}$ be the reconstruction from h. The objective of the autoencoder is to minimize the

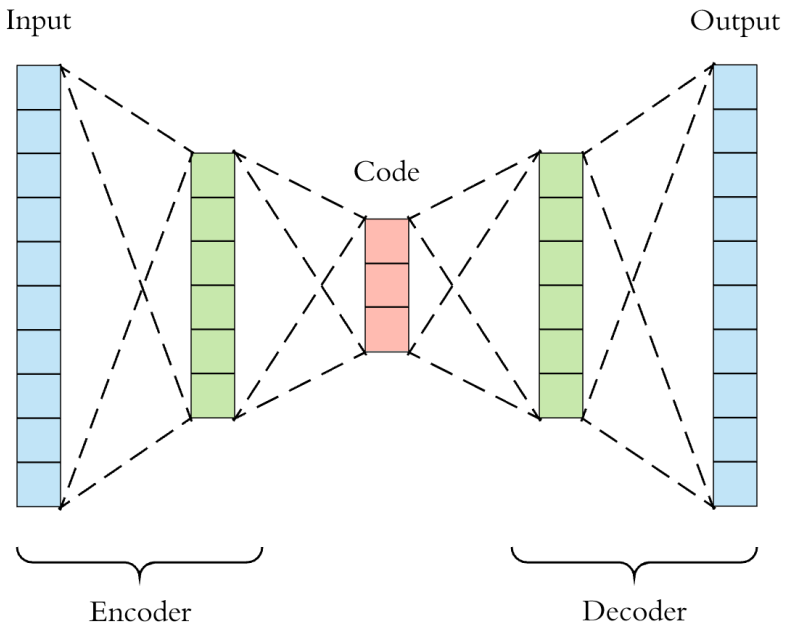


Figure 2.17: An Undercomplete Autoencoder

reconstruction loss Equation 2.41. Autoencoders are split into an encoding and decoding unit. The encoder is a mapping from $\mathcal{X}$ to h, where $h = f(\mathcal{X})$. Similarly, the decoder can be written as $\tilde{\mathcal{X}} = g(h)$. Figure 2.17 shows the model of an autoencoder. Autoencoders can be overcomplete, or undercomplete, depending on the number of units in the hidden layer(s). :

$$\mathcal{L} = (\mathcal{X} - \tilde{\mathcal{X}})^2 \tag{2.41}$$

Reconstruction loss is typically the mean square error between the predicted output $\widehat{x}$ and the original input x. Undercomplete autoencoders are typically used for data embedding and manifold learning. Other variations include Denoising Autoencoder [36], where noise is added sampled from $N \sim (0, \sigma^2)$ and Contractive Autoencoder [41] loss whose objective is to minimize the frobenius norm of the Jacobian Matrix:

$$\mathcal{L} = (\frac{d\,h}{d\,x})^2 \tag{2.42}$$

DAEs are used to make the decoder network insensitive to finite changes in the input, whereas CAEs reduce the change in code layer with infinitesimal changes in the input. Both the Autoencoders are widely used for manifold learning approaches, with h (or z) representing the low dimensional embeddings. Another family of Autoencoders include the Variational Autoencoders [26], which are used to maximize the log likelihood of $P(x)$. Using Bayes rule, we can frame the objective as:

$$logP(x) = log \int_z P(x|z)\,P(z) \tag{2.43}$$

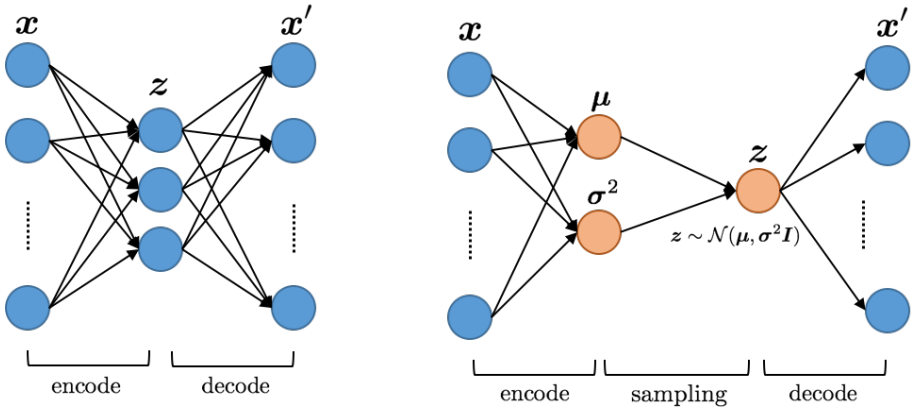


Figure 2.18: Comparison between Autoencoders and VAEs

Using Variational Inference and Jensen's Inequality:

$$logP\,(x) \geq log \int_z q\,(z|x)\, log\, \frac{P\,(x|z)\,P\,(z)}{P\,(z)} \tag{2.44}$$

Simplifying, we get:

$$log\, P(x) = \mathbb{E}_{q(z|x)}\,[log\, P(x|z)] - KL(q(z|x), P(z)) \tag{2.45}$$

The first part of equation 2.45 represents the Maximum likelihood error which measures the reconstruction error between the prediction and input. The second term acts as a prior belief over z and acts as a regularizer. Figure 2.18 compares the architecture of Autoencoder and Variational Autoencoders. The latent variable z can thus for the data embedding in low dimensional spaces.

2.8 Machine Learning for Remote Sensing

> "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E." -Tom Mitchell

Perhaps a poetic definition, but one that does the job nevertheless, Machine Learning is nothing more than finding information from data. This information could be discriminatory, such as classification, or depends on understanding the data distribution, such as many unsupervised approaches. Machine Learning has been a well studied area since the 20th century, with a variant of it called *Deep Learning* benefitting from advancements in hardware and compute capability in general. This section will cover a non-exhaustive overview of Neural Networks, including recent advancements in Deep Learning. We also cover Generative Adversarial Networks, an interesting scenario where Dimensionality Reduction could potentially be applied.

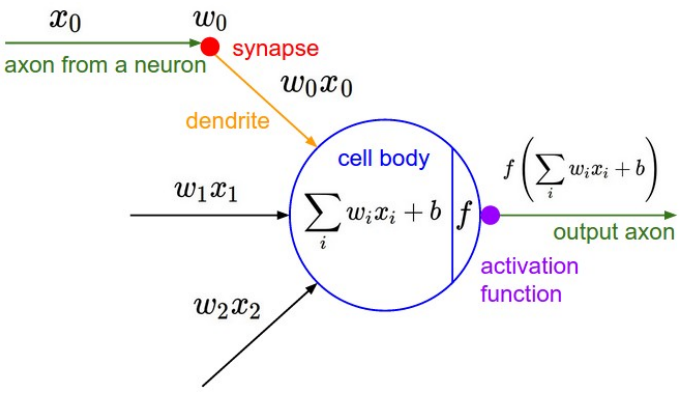


Figure 2.19: Perceptron model.
https://nasirml.wordpress.com/2017/11/19/single-layer-perceptron-in-tensorflow/

2.8.1 Perceptron: Building Blocks of Neural Networks

Perceptrons (Figure: 2.19) can be described as objects that take in weighted combinations of outputs of other perceptrons connected to it, and then applying some non-linearity to the combination. It is based on the principles of Hebbian Learning, where locally grouped perceptrons stimulate each other to fire (excitatory) or inhibitory response. Neurons in the human body have one shot firing, where they produce a 2-level discrete output depending on the stimuli it receives. Machine Learning algorithms modify the perceptron to produce a continuous output. Each perceptron is associated with weights w_{ij} that determine the strength of connections between the neurons. A higher weight indicates dominance of a particular neuron in producing a response from the target neuron. A linear combination of the inputs is then applied to a non-linear activation function, which helps in modelling non-linearities in the input dataset. It also normalizes the function to a given range (usually -1 to 1), without which the values in the network would have caused stability issues.

Multilayer perceptron is a directed graph with the perceptron forming its nodes. It is a bipartite graph, with within layer connections prohibited. Hence all connections between perceptrons are formed across the layers in the network. Some layers could have skip connections, but those are not denoted here for simplicity. The question arises; why do we need multi-layer perceptrons when input-output relations can be drawn between two perceptron layers?

Perceptrons by itself have limited classification abilities. They form a linear hyperplane in the n-dimensional space, hence each perceptron can at most have two classes. A combination of these perceptrons can give rise to complex hyperplanes in the n-dimensional space hence increasing the complexity of the MLP classifier.

1. $\frac{\partial E}{\partial w_{ij}} = \delta_j * x_i$	for all weights and biases
2. $o_j' = o_j * (1 - o_j)$	for output layer nodes using softmax
3. $\varphi_j' = (1 - \varphi_j) * (1 + \varphi_j)$	for hidden layer nodes using tanh
4. $\varphi_j' = \varphi_j * (1 - \varphi_j)$	for hidden layer nodes using logistic sigmoid
5. $e_j = (o_j - t_j)$	for hidden and output layer nodes
6. $\delta_j = e_j * o_j'$	if *j* is an output node
7. $\delta_j = (\sum \delta_j w_j) * \varphi_j'$	if *j* is a hidden node
8. $\Delta w_{ij} = \alpha * \frac{\partial E}{\partial w_{ij}}$	delta for all weights and biases
9. $w_{ij}' = w_{ij} + \Delta w_{ij}$	update for all weights and biases

Figure 2.20: The Backpropagation algorithm update rules
https://visualstudiomagazine.com/Articles/2015/04/01/Back-Propagation-Using-C.aspx?

The advantage of the hidden layer lies in the fact that the complexity of the classifier can be further increased by allowing the network to learn higher dimensional features from combinations of the input. For each hidden layer neuron, it receives weighted input from the previous layer neuron layer. Thus, the weights between the hidden layers allows combinations of inputs to be represented as higher dimensional features for classification. The concept of backpropagation algorithm for learning the hidden layer weights revolutionized the Machine Learning academia. The ground-breaking paper [11] demonstrated a way to apply chain rule to learn hidden layer weights which was a novel approach to an unsolved problem in ML research. In the subsequent section, we discuss the backpropagation algorithm, along with various activation functions used for the same.

2.8.2 Backpropagation

The backpropagation algorithm uses the chain rule of differential calculus to calculate the derivative of Error with respect to the weights of the layers. The objective of this algorithm is to minimize the error produced by the target neurons by establishing a chain of derivatives linking the error with neuron's outputs, activation functions and the corresponding weights. This minimization is iteratively achieved. The gradient of the error for given weight values are calculated, and weight updation opposes this gradient to modify itself in the direction of the minima. This change of weight depends on the learning rate η that determines the step size taken for the update. Backpropagation has the tendency to

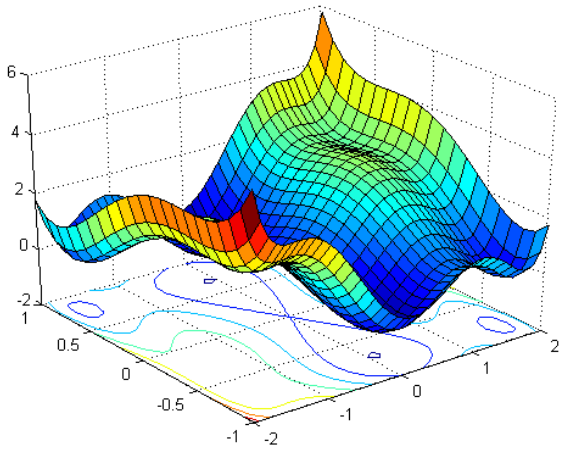


Figure 2.21: Error surface plotted against weight dimensions
https://i.stack.imgur.com/TY1L1.png

get stuck in a local minimum, and several approaches have been proposed to address this, the most prominent being the addition of momentum term α.

The essence of this algorithm lies in the calculation of sensitivity of each neuron, which recursively passes information regarding the weight updation process to the next layer of neurons thus avoiding duplicated calculations, as shown in Figure 2.20. For each set of weights, a matrix multiplication of the sensitivity values of the adjacent layer, the output of the present layer and learning rate determine the total change of weights taking place. Thus, for each iteration, the weight changes are propagated against the forward flow of values in the network. While backpropagation is a powerful algorithm that adaptively updates its weights based on the MSE or Cross-entropy loss functions, it suffers from limitations which we discuss below.

Limitations of Backpropagation

The backpropagation algorithm uses gradient descent / stochastic gradient descent to update the weights in the direction of the minima. When we plot the Loss function against the weights, we expect to find a large number of local minima spread throughout the domain. Also, there are large "flat" regions in the surface where gradients are zero, called *saddle points*. These regions cause the derivative of the error surface to be zero thereby preventing weight updates from taking place. A solution to this problem is the use of momentum α. The momentum term is a fraction between zero and one which is used to include the previous weight update in the equation. The previous weight update helps to overcome local minima by providing a small boost in the weight change which might

push it out of the locally convex curve. For saddle points, the momentum could help shift the gradient point out of the plateau region towards the local / global minima that exist in the vicinity of saddle points. A drawback with this approach is that a large momentum might push the weight updates beyond the global minima region which is undesirable. Thus, a careful tradeoff between a large and small momentum value is desirable which can be tuned using validation of the dataset. Similarly, the learning rate η determines the magnitude of weight shift taking place during an iteration; with large step values exceeding the global minima leading to the error spiraling out of control. Hence if we see the error plot increasing as a function of epochs one possible conclusion could be that the learning rate is too high.

Backpropagation is also prone to *vanishing/exploding gradients*, which states that deeper the network, more likely that the initial few layers will receive insignificant weight updates. This happens due to the derivative of activation functions associated with the weight updates. For a logistic activation function, the maximum value of its derivative is 0.25, hence corresponding to weight change for each layer, the successive product with the layer's derivative brings the net change in weights to near zero values. This is the reason traditional Neural Networks do not have "deep" architecture, with most architectures involving two to three hidden layers at the most. Another drawback with the logistic function is the near zero gradients at small distances from the origin, which inhibits the process of learning. Many of the limitations of the logistic function can be overcome by recently devised activation functions such as the Rectified Linear Units (ReLU) [35]. ReLU is a piecewise linear function which takes on Identity values for inputs greater than a threshold and zero value for inputs less than the threshold. The attractive feature of ReLUs is only 2 derivatives exist {1,0} and hence is less likely to suffer from the vanishing gradients problem. It also has zero gradient similar to sigmoid on one side of the input plane. This however is not a significant issue as there will be some inputs which will operate on the linear region of ReLU causing weight updates to take place by a large margin if weights to the ReLU are saturated at values far from the origin. A comparison between ReLU and sigmoid is shown in Table 2.1 and the functions are shown in Figure **??** ReLUs are computationally cheaper to evaluate which makes them an attractive option and preferred choice of the Deep Learning community.

2.9 Convolutional Neural Networks

In this section we present a short primer on Convolutional Neural Networks and how they are different from Fully Connected Networks, with a brief summary on different archi-

Table 2.1: Comparison between Sigmoid and ReLU

Comparison	Sigmoid	ReLU
Function	$\frac{1}{1+e^{-x}}$	$\max(0, x)$
Gradient magnitude	0 to 0.25	$\{0, 1\}$
Vanishing Gradients	Yes	No
Exploding Gradients	No	Yes
Nature of Curve	Continuous; monotonically increasing	Piecewise linear; monotonically non-decreasing
Computation	Derivative computation is time consuming	Derivatives are quick to obtain

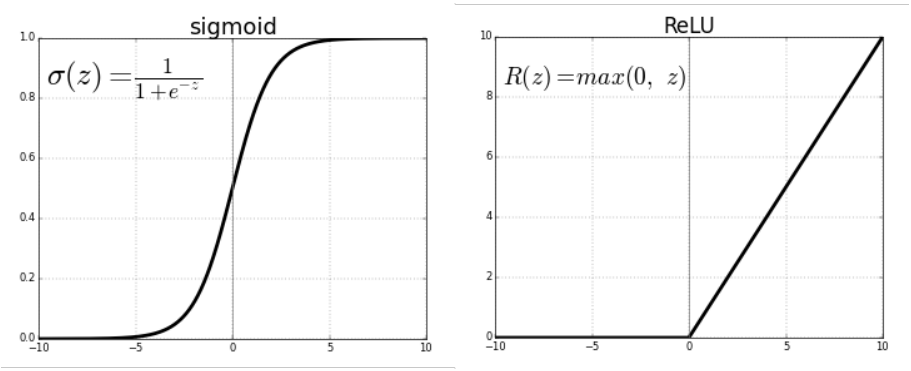


Figure 2.22: Sigmoid and ReLU activation functions
https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

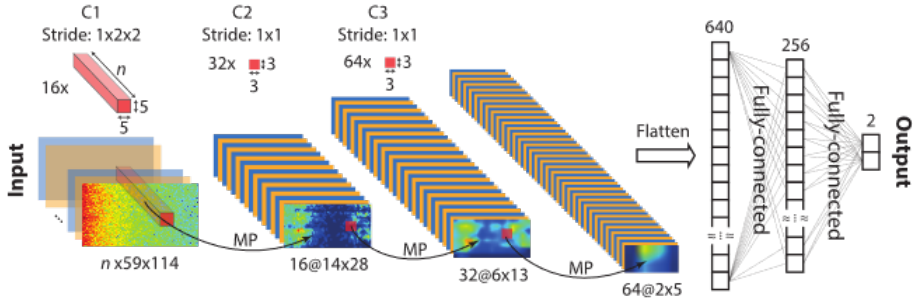


Figure 2.23: CNN architecture
https://www.groundai.com/project/a-generalised-seizure-prediction-with-convolutional-neural-networks-for-intracranial-and-scalp-electroencephalogram-data-analysis/

tectures. An overview of Batch Normalization [25] will be presented, which is used in the proposed work. We will also see applications of Autoencoders and CNNs for Hyperspectral Image classification. To complete the discussion, we will present Generative Adversarial Networks [16], a generative model which is extensively used to learn the probability distribution of the input data. We will show a potential use case of Dimensionality Reduction in such a setting.

2.9.1 Overview

CNNs were formed in what were the first steps in Deep Learning. It countered the limitations of traditional ANNs which suffered from the problems of vanishing and exploding gradients. Other issues with ANNs were translational variance, rotational variance to name a few. To achieve high classification accuracy, ANNs used a lot of parameters (in the form of weights) which were computationally expensive to estimate and were prone to overfitting.

A revolutionary change in the field of computer vision took place when Yann LeCun published the first known paper on convolutional neural networks [28] with the introduction of LeNet-5 for digit recognition on the widely used MNIST dataset. While some of the ideas proposed in that paper are no longer in use (due to faster computational power available now), the essence of the paper which used Convolutional Neural Networks remains. We will briefly go through some of the intricacies of CNNs before moving on to various architectures which are used today.

As the name suggests, CNNs employ extensive convolutional operations to extract features. It is important to note that CNN learns by itself which features to extract, which is a major improvement over other Machine Learning algorithms. CNNs are able to do so by implementing 3 dimensional kernels which convolve over 2 spatial dimensions. Es-

sentially convolution can be viewed as a traditional neural network with highly restricted receptive field and extensive weight sharing between all the neurons in the layer. Modern CNNs (Figure 2.23) are composed of three parts:

1. Convolution
2. Pooling
3. Fully Connected Layer

The concept behind convolutions is vital behind the CNNs being translational invariant. The kernels convolve around the entire image with the same weights, meaning identical features are captured by the layer irrespective of the location. Pooling layers, are often accused of robbing the network of localization ability but are used nevertheless to make CNN computations faster. There are two popular techniques for pooling: Max pooling and Average pooling. Max pooling places a *n* x *n* (usually 2x2) window over the image, and selecting the maximum value in it to propagate to the next layer. The input to the next layer is the image with its spatial resolution reduce by the pooling window size. The reduced size of the image means fewer computations for the convolutional network to perform. An intuitive interpretation of the pooling operations is that it does not matter where precisely an object has been detected. The fact that a feature exists is enough to form higher level features which contain that object. The accurate location of an edge in an eye does not matter if the edge is displaced by a minor amount, the existence of the edge suffices to form features out of it. As we go deeper into the layers, the features start taking a global outlook, thus individual sub-feature determination can be sacrificed for higher computational speed. Fully connected layers are traditional neural networks which are formed by unravelling the 3-dimensional feature space obtained as the output of the convolutional-pooling layers. The final FC layer is capped with a softmax layer to provide the class probabilities for the network. We present few popular architectures that have shaped the use of CNNs.

LeNet - 5

LeNet-5 [28] is one of the earliest implementations of Convolutional Neural Network by Yann LeCun et.al. The '5' in the name denotes the width of the kernel implemented in their convolutional neural network. LeNet-5 was tested on MNIST dataset where it achieved better than state of the art results. Although small by today's standards, Le-Net is included in this thesis due to the supreme importance it has in Deep Learning history.

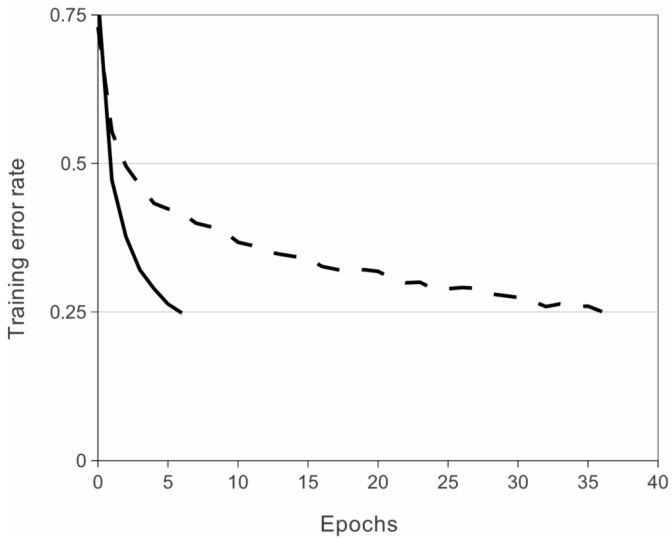


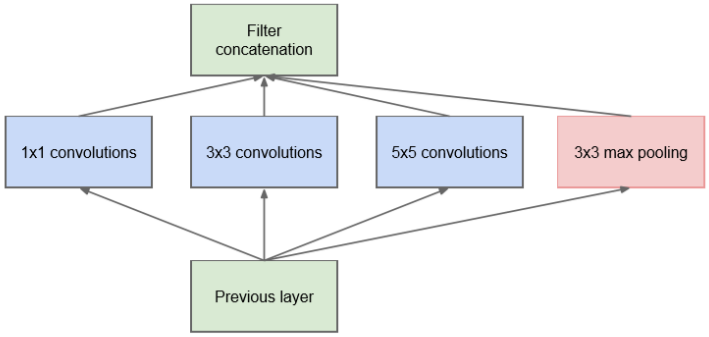
Figure 2.24: The use of ReLU shows speedier convergence when compared to *tanh*. [27]

AlexNet

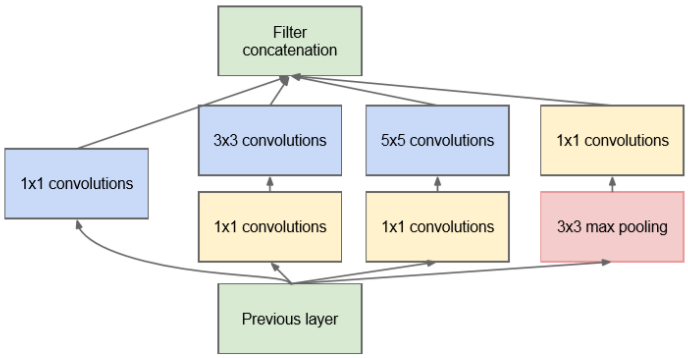
AlexNet [27] introduced several ideas to incorporate in the convolutional neural network such as the use of Local Response Normalization, Drop-out Regularization, use of multi-GPU configurations. This network was the winning entry for the ILSVRC-2012 contest which is based on classification of high-resolution images into a thousand categories including 100s of sub categories. The key ideas presented in the paper were communication between two GPUs to train the ConvNet. The image was bifurcated into two parts to train on the individual GPUs with interaction happening at select levels only. The other idea put forth was the use of Rectified Linear Units, their advantages are discussed earlier. To reiterate, ReLU eased the computation burden in CNNs due to their simple representation. The authors also stated that ReLU significantly helped in reducing overfitting, as shown in Figure 2.24.

GoogLeNet

The InceptionNet or GoogLeNet [46] was another set of architectures that brought in radical changes is the design of Convolutional Neural Networks. The InceptionNet defined new state of the art results with its winning entry in ILSVRC-14. With InceptionNet, the authors aimed to bring Deep Learning computations to the mobile platform which have limited computational resources. Thus, they aimed at a pragmatic approach to support Deep Learning on mobile hardware and platforms. Inception networks use the concatenation of various filters to provide output at different scales, as shown in Figure 2.25 This could be summarized by a quote from the original paper:



(a) Inception module, naïve version



(b) Inception module with dimensionality reduction

Figure 2.25: The inception module allows to extract features at different scales. [46]

> "Furthermore, the design follows the practical intuition that visual information should be processed at various scales and then aggregated so that the next stage can abstract features from the different scales simultaneously." [46]

VGG - 16

The VGG model was proposed at the same time as the Inception Net architecture, but used a less complex network. VGG implemented a fixed 3x3 kernel, the idea being that the receptive field of the kernel would increase with depth as multiple 3x3 kernels are stacked upon each other. The number of channels double after each convolutional step, giving the network a sense of uniformity. The VGG architecture gave 2 important results: Local Response Normalization was ineffective in reducing the error. Also, the paper proved that network accuracy increased as depth of the network was increased for simple 3x3 kernels. The model also provided state of the art results for object localization as a part of their wining entry for ILSVRC – 2014.

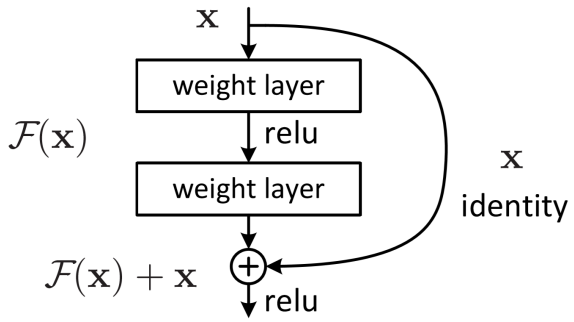


Figure 2.26: Allowing skip connections prevents the hindering of training process for deeper networks. [20]

Residual CNN

ResNet [20] was designed to combat the issue of vanishing and exploding gradients as convolutional neural networks tend to go deeper. With increase in depth, the neural network starts suffering from near zero gradients which do not cause any learning to happen. Additionally, deeper neural networks struggle to find an *Identity* mapping between the input and the output, thus causing difficulties in learning. Residual network counters the effect of deep networks by introducing *skip connections* as shown in Figure 2.26.

These skip connections have a shortcut path that connects two layers directly, which allows forward propagation of values along the network. In deep layers, weight magnitudes are small due to regularization which is done to prevent overfitting. The small weight magnitudes suppress information passing through them therefore information disappears from the network. The skip connection therefore plays the role of providing a value such that the final value post activation is equivalent to the activation of the first layer of the skip connection. Essentially, the entire skip connection provides the role of Identity function. Hence Residual networks guarantee that the training error will not detoriate as the number of layers are increased.

R-CNN

Region based CNNs [14] were designed to output accurate bounding boxes, which meant better localization for an object. The approach was divided into two parts: Region Proposal and Feature Extraction. Region proposals were required as there are large patches in the image which represent background; no objects are present in those locations. R-CNN aims to save computation power by identifying proposals depending on the presence of any kind of object at that location. Once the region is identified, it is used for feature extraction and classification.

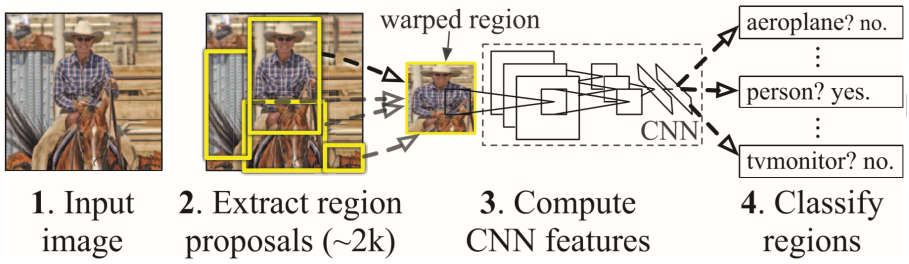


Figure 2.27: R-CNN first performs region proposal, after which the regions are resized to fit as input to the CNN. [14]

Region Proposal involves using image segmentation techniques to narrow down search regions for objects in the image. R- CNN does not specify any particular technique for Region proposal, but the faster R-CNN version advocates the use of CNN for quick region proposals. After obtaining regions proposals, the algorithm warps the region into an image which could be fed to the Neural Network. This warped image is then classified by the Convolutional Neural Network to output not only the class probabilities but also the bounding box for the image. Thus, the use of Region proposals helps to give the algorithm a rough estimate of the bounding box area and size. The summary of this is shown in Figure 2.27.

A significant drawback of the R-CNN algorithm was the computation time taken by the region proposal and classification stages. Two solutions are presented which solve this problem: Fast R-CNN and Faster R-CNN. Fast R-CNN uses the concept of convolutional sliding windows to expedite the process of region Classification. Instead of providing a single input image to the classifier, multiple such images could be grouped with a convolutional window passing through each of them. The output of the classifier will be class probabilities of classes for each image which are obtained at the same time. The approach of sliding windows is used to avoid duplicate calculations which occur if we classified the images one at a time. These duplications occur due to same set of kernels and FC weights present for the images.

The results of the R-CNN algorithm for object localization indicate a significant rise in performance over the state of the art SegDPM in the Pascal VOC competition. These results are further boosted if CNN based Region proposals are used as presented by the Faster R-CNN model.

2.9.2 Deep Learning for Hyperspectral Imagery

The state of the art classification for hyperspectral and multispectral images has involved Support Vector Machines [6] and Deep Belief Networks. With the rise of powerful computing hardware, Convolutional Neural Networks are soon being applied to the task of hyperspectral image classification. In this dissertation, we look at four such applications in detail.

Pretraining For Hyperspectral Convolutional Neural Network Classification

The authors [51] discuss various measures to apply CNN based classification using pre-training. Pre-training is a term used when the given convolutional network is trained on a large, generalized set of data, and the user's application requires dealing with a subset of those classes. Pre-training is especially useful when we have small amount of training data. CNNs usually require large amount of data to learn the weights and prevent overfitting. Using pre-training, we fix all but the final layer of weights, and replace the softmax classification of the output with the softmax output nodes that the user desires. The advantage of this technique is less amount of data as well as computation time is required compared to training a CNN from scratch. We build upon the knowledge base of the CNN by using majority of its weights and fine-tuning the final set of weights to enable classification according to user-based class outputs.

While pre-training is a widely used strategy to speed up the convergence of the network, application to hyperspectral imagery requires some modifications. The challenges of dealing with hyperspectral images are various sensors operating in different spectra. Sensors such as AVIRIS, Hyperion, HyMap and others have different number of bands spanning different set of wavelengths which causes difficulties in merging datasets. Hence a particular band number in both the sensors could indicate different spectra causing misinterpretation of the input. Another vital difference is the platform from which the images are captured. Space-borne, aircraft-borne and field-borne images have different characteristic, with space-borne images being prone to low Signal to Noise ratio. Merging of these datasets require special care.

The two approaches that were used to handle data dissimilarity between sensors were interpolation of reflectance for missing wavelengths and normalization of the spectra to values between 0 to 1. This approach hence forms a union over all the spectrum bands available in the dataset, as well as gets rid of individual sensor peculiarities with respect to reflectance sensing. The authors used spectral band for convolution, making it a 1-

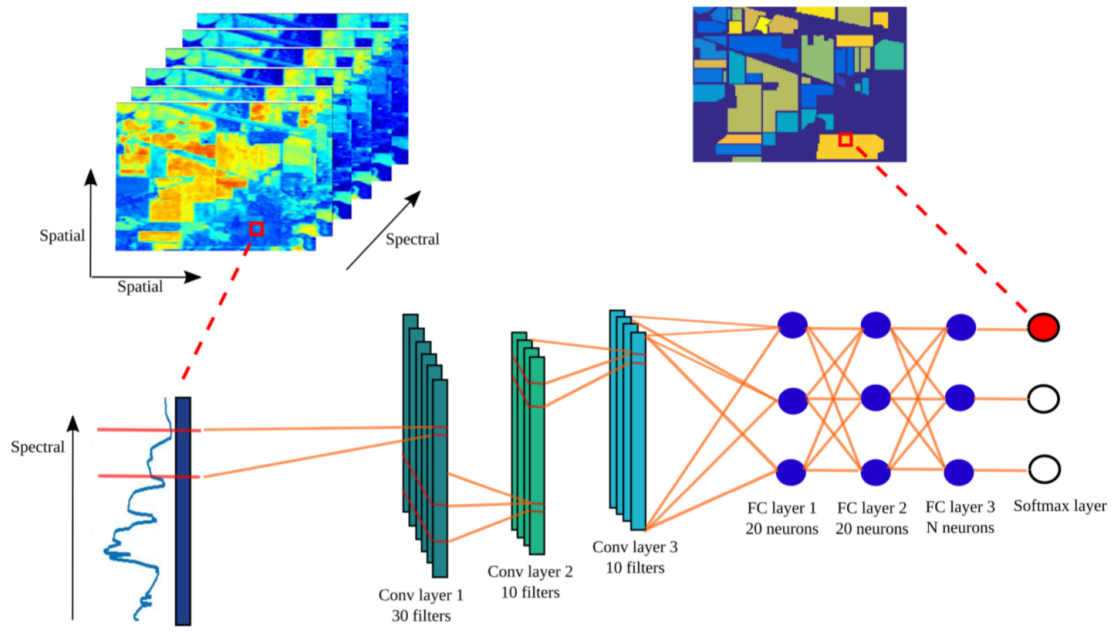


Figure 2.28: Spectral convolution is employed for the purpose of classification. Note the absence of pooling layers.[51]

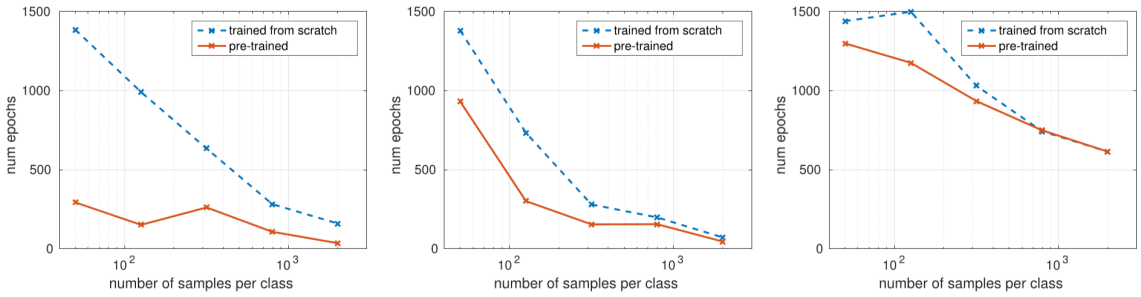


Figure 2.29: Comparing Pretraining and Training from Scratch.[51]

dimensional convolution, as shown in Figure 2.28.

An interesting observation is that no convolution is taking place in the spatial domain, hence spatial feature detection can not take place. To preserve spectral resolution, the authors have avoided the use of Pooling layers. We also note the limited depth of the network, which prevents the network from learning higher order features. Perhaps this depth limitation is to reduce computation expense which is increased with the lack of pooling layers. The comparison of Pre-training and training from scratch is shown in Figure 2.29. It is evident from the result that pre-training helps speed up convergence and provides similar accuracy compared to training from scratch if the training dataset is small in size.

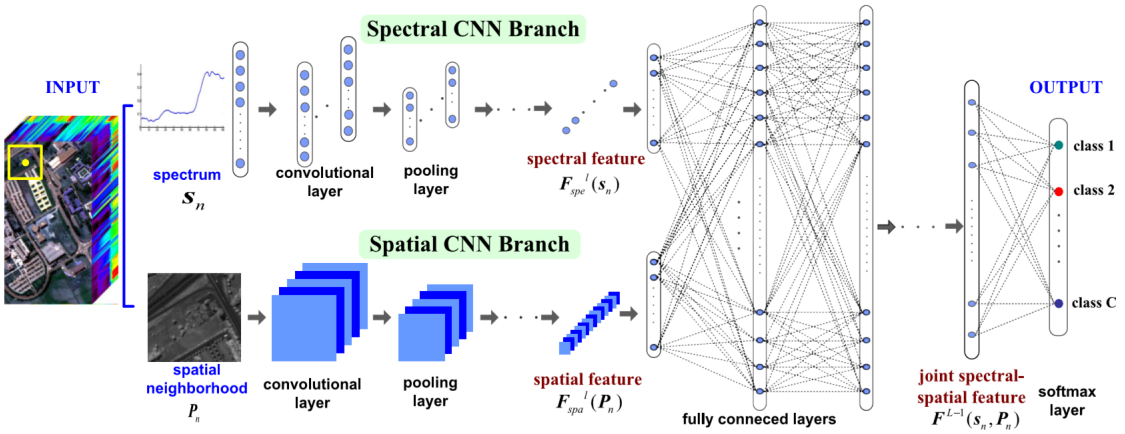


Figure 2.30: Architecture of the Spatio-Spectral classifier for classification. [52]

Hyperspectral Image Analysis Using Deep Learning – A Review

The review paper [39] compares different deep learning and machine learning approaches to classification on standard datasets such as Pavia, Salinas, Indian Pines. It compares Deep Belief Networks, auto encoders and convolutional neural networks and compares it with Machine Learning algorithms such as Support Vector Machines. The results indicate Deep Learning approaches match state of the art for Hyperspectral Image classification, with an added advantage of algorithms learning which features to extract in comparison to ML approaches that require hand-crafted features.

Learning And Transferring Deep Joint Spectral–Spatial Features For Hyperspectral Classification

Earlier implementations of CNN for hyperspectral imagery used kernel convolution only along the spectral domain. An attempt is made by the authors to obtain spatial as well as spectral features using a dual architecture CNN, with one CNN track corresponding to the Spectral features, the other track corresponding to the spatial dependency. The two tracks individually extract features and merge together to form the Fully Connected Neural Network. Thus, at this stage the classification incorporates spatial as well as spectral information, which is shown in Figure 2.30 As proposed in [51], the authors similarly use transfer learning to increase the speed of network convergence.

The authors have also evaluated the effect of kernel size on the classification accuracy. Spectral domain kernels have larger sizes than the spatial domain due to the large number of spectral bands present in HSI. The results of this study are depicted in Figure 2.31. The results indicate that small spatial kernels preserve locality and are simpler to compute over the larger spatial kernels. A greater proportion of labelled data used for training gives

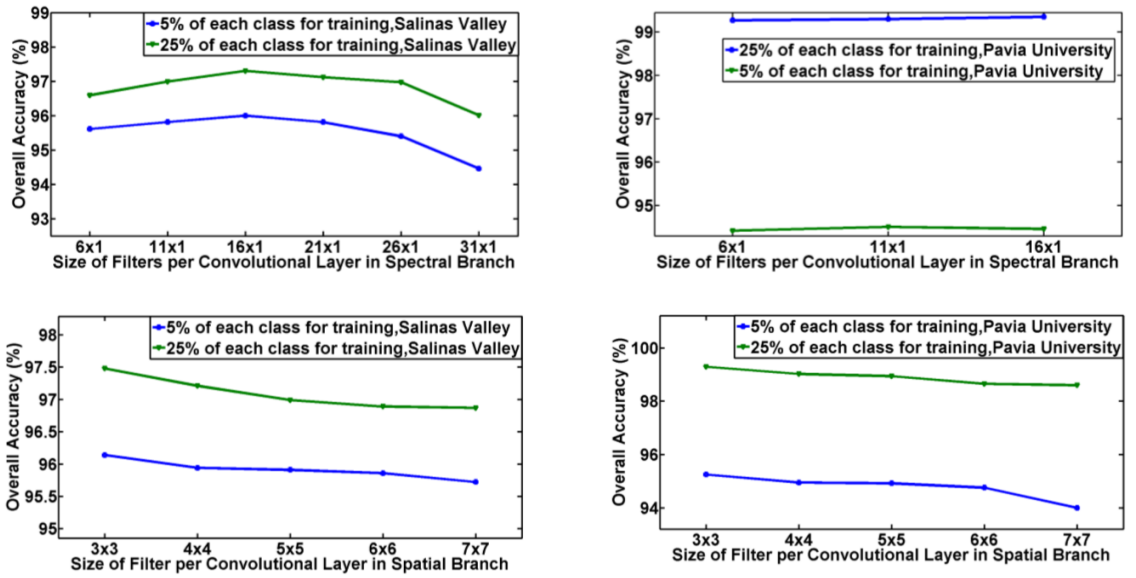


Figure 2.31: Effect of kernel size of spatial/spectral branch on accuracy . [52]

better results over marginal, pre-trained dataset. The spectral kernel response peaks at 11 to 16 kernel width and drops off in either direction.

Deep Supervised Learning For Hyperspectral Data Classification Through Convolutional Neural Networks

In this paper the authors [33] have sought to reduce the spectral dimensionality of the data using Principal Component Analysis. One could evaluate the approach taken by the author and compare it with using 1x1 kernels for dimensionality reduction. The advantage of using 1x1 kernels also includes non-linearity modelling. The results of the classification have been shown in Figure 2.32. While the approach performs well on sample dataset, it remains to be seen if similar accuracy is achieved for more complex, noisy datasets.

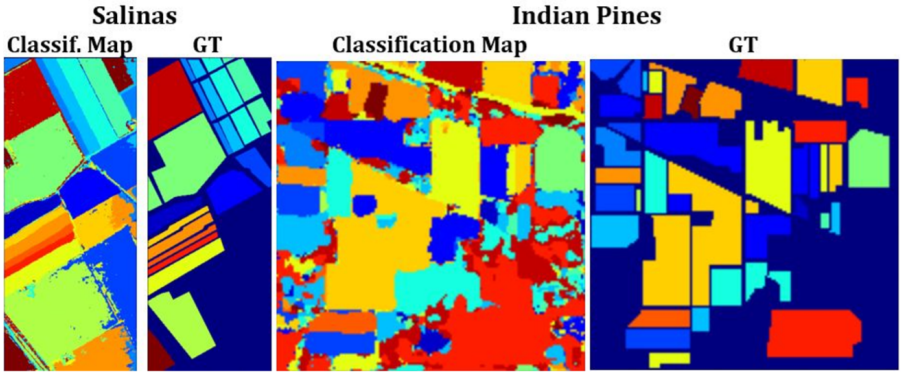


Figure 2.32: Results of HSI classification on the Indian Pines dataset. [33]

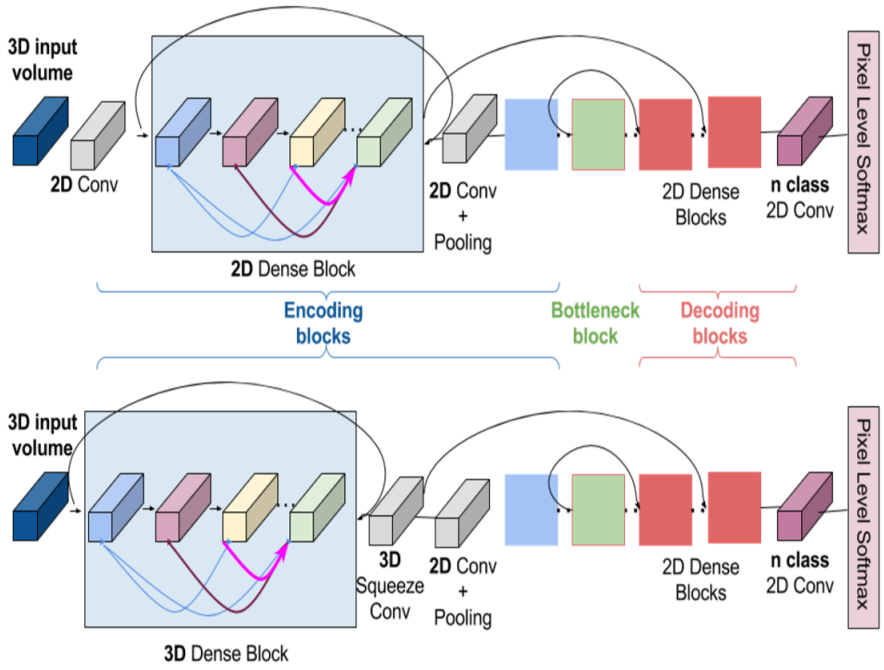


Figure 2.33: Implementation of 3D convolution using DenseNet. [18]

Deep Learning For Semantic Segmentation of Remote Sensing Images With Rich Spectral Content

In a novel approach, the authors of this paper attempted to use 3D convolution in the spectral spatial domain to jointly estimate these features. The authors have implement 3D convolution based on modifications of the recently proposed architecture of DenseNet. DenseNet uses a concept similar to Residual Networks where data from a given layer is fed to layer separated from it. The layers are then treated with 2D analysis of these dense blocks followed by a pixel level softmax activation to obtain class probabilities. Figure 2.33 shows the proposed architecture.

Spherical Stochastic Neighbour Embedding of Hyperspectral Data

In this paper [29], the authors attempt to bring the ideas of t-SNE for manifold learning in Hyperspectral Imagery. The authors use concepts from Spherical embedding to construct an n dimensional manifold in $\mathbb{R}^{n+1}$ space. However, manifold learning is performed with a focus on Hyperspectral Image Classification in this paper. Samples in $\mathbb{R}^m$ are projected onto a unit hypersphere as in Spherical Embedding, where the spatial arrangement on the surface of the hypersphere is determined by *sSNE*. The authors define a new method of estimating probabilites p_{ij} and q_{ij} inspired from a bilateral filtering approach and an m-dimensional Brownian motion. We define the $m - dimensional$ hypersphere as:

$$\mathbb{S}^n = \{ y \in \mathbb{R}^{n+1} : \| y \|_2 = 1 \} \tag{2.46}$$

The parametrization of points on the given hypersphere is given by $\{\theta^1, \theta^2, \ldots, \theta^n\}$. To map data points onto a Spherical manifold, two approaches exist in literature, l_2 normalization and Spherical Embedding. As the name suggests, l_2 normalization involves scaling down the points to a unit norm sample. Drawbacks of this technique include samples that have different norms but have the same direction are collapsed onto a single point and hence lose identity. The Spherical Embedding approach involves:

$$\mathcal{Y}^* = \arg\min_{y} | \mathcal{Y}\mathcal{Y}^T - \hat{S} | \tag{2.47}$$

The solution to Equation 2.47 involves the eigenvectors corresponding to the largest eigenvalues. The Spherical Embedding approach like many other before suffer from the crowding problem, where samples are pulled towards the centre, with few samples spread out to explain the variance corresponding to the eigenvalues.

To take into account the spatial spread of the samples, the authors turn to t-SNE, which involves imposing a probability distribution over the samples in $\mathcal{X}$ and $\mathcal{Y}$. Like t-SNE, the minimization between p and q is performed using KL divergence. The choice of unnormalized p is based on a bilateral filtering approach with two components for spatial and spectral values. We define it as:

$$\begin{aligned} K_s^{(i,j)} &= exp\{ -\|s_i - s_j\|^2 / h_s^2 \} \\ K_r^{(i,j)} &= exp\{ -(r_i - r_j)^T \Sigma_r^{-1} (r_i - r_j) \} \\ p(s_i, s_j, r_i, r_j) &= K_s^{(i,j)} * K_r^{(i,j)} \end{aligned} \tag{2.48}$$

$K_s^{(i,j)}$ involves the spatial distance between i and j, whereas $K_r^{(i,j)}$ is the mahalanobis distance between the two spectra. Now that we have defined p, the probability distribution on $\mathcal{X}$, we turn our attention to q. To define probabilities over $\mathcal{Y}$, we use an exit function based on an m-dimensional Brownian motion. We define q_{ij} as:

$$q_{ij} = \frac{\| y_j - \rho y_i \|^{-n}}{\sum_{k \neq l} \| y_k - \rho y_l \|^{-n}} \tag{2.49}$$

y_j represents the anchor point around which the distribution is symmetric for a fixed j. The minimization objective for embedding is $arg\min_y KL(p\|q)$ with the lagrangian constraint $\mathcal{Y}\mathcal{Y}^T = 1$.

2.10 Summary

This chapter introduces us to various techniques in Dimensionality Reduction, highlighting Matrix Factorization methods as well as Neighbourhood Graph algorithms. We have also seen in the previous section varied approaches of CNNs applied for the purpose of Hyperspectral Image classification. Recently, other architectures which derive from Neural Networks have been proposed, such as CapsuleNet for HSI classification [40]. Another interesting application includes Generative Adversarial Nets, which will be covered later. We have also reviewed a manifold visualization approach inspired from tSNE for Remotely Sensed Imagery. In the next chapter, we try to identify areas of improvememt in the existing literature. We formally state the problem statement, and attempt to solve it in this dissertation.

Chapter 3

Problem Statement and Data Sets

In this chapter, we identify and set the objective of the research included in this thesis. We also describe the datasets that we will use in brief.

3.1 Problem Statement

In the previous chapter, we have seen techniques such as the Principal Component Analysis (Sec: 2.1) and the Linear Discriminant Analysis (Sec: 2.2) which have highly interpretable results. A major drawback for these algorithms is that they are linear dimensionality reduction algorithms, thus cannot be used for datasets having a high degree of non-linearity. PCA and LDA also assume orthogonal components, which may not necessarily be true with the dataset. By virtue of the objective function, PCA preserves large pairwise distances, maintaining the global structure. Manifold learning and data visualization requires the preservation of local distance to see how the samples are arranged in space relative to each other. Transforming samples from $\mathbb{R}^m$ to $\mathbb{R}^n$ is easier with PCA since the transformation involves a simple projection along the principal components. PCA also allows us to easily extend the dimensionality of our dataset, by selecting the next eigenvector to project along. This leads to the Principal Component Analysis being a fast algorithm with practical usage. Linear Discriminant Analysis makes limiting assumptions: Gaussian Distributions with the discriminatory power between classes residing in their means. Also, if our dataset has *'C'* classes, LDA can give us atmost C-1 *components*. Kernel-PCA (Sec: 2.1.3) uses the kernel trick as a tool for non-linear dimensionality reduction, but suffers from high time and space complexity.

Laplacian Embedding (Sec: 2.3) and Locally Linear Embedding (Sec: 2.4) as we have seen are non-linear embedding techniques hence perform better than PCA and LDA. However, these techniques suffer from crowding problem, where more samples

are present at the centre. These techniques are also not parameterized, making it computationally expensive to obtain the embeddings for unseen samples. Laplacian embedding relies on natural clusters in the dataset to give good results, whereas LLE performs poorly if the sampling is inadequate for highly non-linear manifolds.

t-SNE (Sec: 2.5) is a powerful algorithm aimed at manifold visualization by converting the dataset into a graph where the edges are weighed by the corresponding probabilities. UMAP (Sec: 2.6) draws similarities with tSNE such that both account for the uneven density distribution of samples. Unfortunately, both the techniques are not parameterized to embed the unseen datapoints into reduced space without re-running certain parts of the algorithm. tSNE is in particular vulnerable to the curse of dimensionality. UMAP on the other hand is prone to overfitting when performing classification tasks on the embeddings. Practically, the data generated by UMAP was not as intuitive to us as compared to tSNE or Principal Component Analysis.

Autoencoders 2.7 are a Deep Learning based approach for dimensionality reduction. The result obtained by Autoencoders are the least explicable of all the algorithms we have seen so far. Although they are parameterized, their *VC dimension* is very high due to the large number of trainable parameters. On sample datasets which have been used in this dissertation, Autoencoders have a very high tendency to overfit. Training of autoencoders in low (two or three) dimensions is difficult due to the periodic spikes appearing in the reconstruction loss, which indicates some level of overfitting. Also, the training time taken by Autoencoders is very high, inspite of the fact that we used mini-batches with stochastic gradient descent.

sSNE 2.9.2 was proposed as a dimensionality reduction tool specifically aimed at pixel wise classification of Hyperspectral Imagery. It uses domain knowledge of Hyperspectral Imagery such as Illumination, BRDF effects and the high correlation between bands. While it has managed to generalize tSNE for n-dimensions, the visualization result is not informative, especially in the case where $S^n \in \mathbb{R}^1$ is a manifold in $\mathbb{R}^2$. sSNE is a computationally expensive algorithm, which includes computing the covariance matrix $O(Nm^2)$, matrix inversion $O(m^3)$, apart from the time complexity of t-SNE which is $O(N^2)$. Although the worst case complexity turns out to be $O(N^2)$, the practical running time of the algorithm is much longer than the other algorithms which we have discussed. Thus, the ability to generalize sSNE beyond sample remote sensing datasets remains to be seen.

In this dissertation, we frame the problem statement as three objective to be accomplished:

1. **Present an algorithm for data embedding and manifold visualization of hyperspectral bands in $\mathbb{R}^{2/3}$.**
2. **Propose a supervised variant of the algorithm for sample clustering while retaining the global structure of the manifold**.
3. **Develop a proof-of-concept for an unsupervised variant of the algorithm for sample clustering while retaining the global structure of the manifold**.

While designing the algorithm, we keep the following points in mind:

1. *Time complexity*: Most of the algorithms which have been studied scale quadratically with the number of samples in the dataset. Hyperspectral Images are known to be voluminous, hence the algorithm should be able to handle data of that scale. Our approach to this issue is by using *mini-batches* of our samples. Implementation of mini-batches helps in faster convergence and less use of memory resources. By using stochastic gradient descent, we inject controlled noise in the gradient, improving the generalization ability to new datasets.

2. *Paremeterization*: Using trainable parameters gives us the ability to map unseen samples from the original space to the reduced space without repeatedly undergoing the embedding process as done in most of the algorithms presented in this dissertation. Thus, we wish to accomplish: $\mathbb{R}^m \xrightarrow{f(X,w)} \mathbb{R}^n$, an efficient way of embedding samples.

3. *General Purpose Embeddings*: Approaches such as sSNE and UMAP are excellent for dimensionality reduction as a pre-processing tool for classification. At the expense of promoting clustering, the algorithms trade-off local variations within similar points. For tasks where we need to identify the true manifold [54], these techniques may not deliver the most appropriate results.

4. *Curse of Dimensionality*: An important aspect of working with hyperspectral bands is the Curse of Dimensionality. Algorithms such as t-SNE perform poorly in Hyperspectral spaces due to this phenomenon. To work with hyperspectral imagery, our algorithm should be robust against it.

To provide a fair comparison between the proposed algorithm and various other dimensionality reduction approaches, we do not include factors such as the correlation between

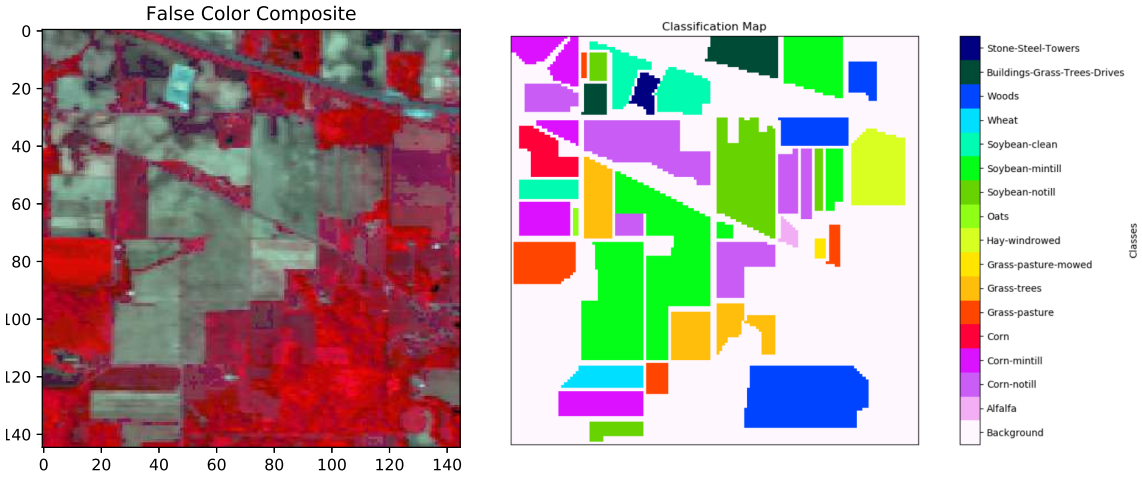


Figure 3.1: (L): Indian Pines FCC (R): Indian Pines Ground Truth

bands and illumination effects on spectra. This translates to not using Convolutions, Mahalanobis distances or Cosine similarity between the samples. We feel that our decision is justified since the existing algorithms studied in this dissertation do not factor in domain-specific knowledge of Hyperspectral Imagery, and any comparison with them will result in ambiguity on whether the proposed algorithm delivered results or the pre-processing steps (or both)!

3.2 Datasets

We use three datasets to verify our results: Indian Pines, Salinas and Pavia University. All the three are hyperspectral images well recognized in the Remote Sensing Community. We thank *Grupo de Inteligencia Computacional (GIC)* for maintaining the datasets on their webpage.

1. *Indian Pines*: The Indian Pines scene was acquired by NASA's AVIRIS Spectrometer in 1992. It is a $145 * 145$ size image consisting of 200 bands, after removing the bands affected by atmoshperic effects. The wavelengths captured were in the range $0.4\mu m$ to $2.5\mu m$ with spatial resolution of $10m$. The ground truth contains 16 classes which are listed in Table 3.1a and shown in Figure 3.1. It is important to note that the classes are not mutually exclusive. $2/3^{rd}$ of the land cover consists of Agriculture, with the remaining $1/3^{rd}$ consisting of Forests and natural vegetation in the scene.

2. *Salinas*: Salinas is another dataset acquired by AVIRIS, but having a superior spatial resolution of $3.7m$. Like Indian Pines, Salinas has 16 classes, but

has considerably lesser overlap in comparison to Indian Pines. The classes consists of bare soil, vegetable and vineyards. The Dataset and Ground Truth is shown in Figure 3.2. Table 3.1b contains the list of classes present in the dataset.

3. *Pavia University*: This scene was acquired by the *ROSIS: Reflective Optics System Imaging Spectrometer* sensor in Italy. This sensor has a spatial resolution of $1.3m$ with a spectral resolution of $4nm$. It consists of 103 hyperspectral bands spanning wavelengths from $430nm$ to $860nm$ with an image size of $610 * 610$. Nine classes are present in the scene, but many pixels in the dataset are unlabeled. The FCC and ground truth is shown in Figure 3.2. Various classes present in the scene are shown in Table 3.1c.

(a) Class Distribution of Indian Pines

Number	Class	Samples
1	Alfalfa	46
2	Corn notill	1428
3	Corn mintill	830
4	Corn	237
5	Grass pasture	483
6	Grass trees	730
7	Grass pasture mowed	28
8	Hay windrowed	478
9	Oats	20
10	Soybean notill	972
11	Soybean mintill	2455
12	Soybean clean	593
13	Wheat	205
14	Woods	1265
15	Buildings Grass Trees Drives	386
16	Stone Steel Towers	93

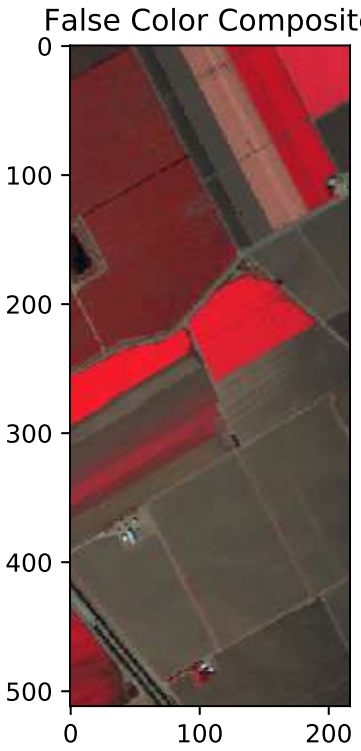
(b) Class Distribution of Salinas

Number	Class	Samples
1	Brocoli green weeds 1	2009
2	Brocoli green weeds 2	3726
3	Fallow	1976
4	Fallow rough plow	1394
5	Fallow smooth	2678
6	Stubble	3959
7	Celery	3579
8	Grapes untrained	11271
9	Soil vinyard develop	6203
10	Corn senesced green weeds	3278
11	Lettuce romaine 4wk	1068
12	Lettuce romaine 5wk	1927
13	Lettuce romaine 6wk	916
14	Lettuce romaine 7wk	1070
15	Vinyard untrained	7268
16	Vinyard vertical trellis	1807

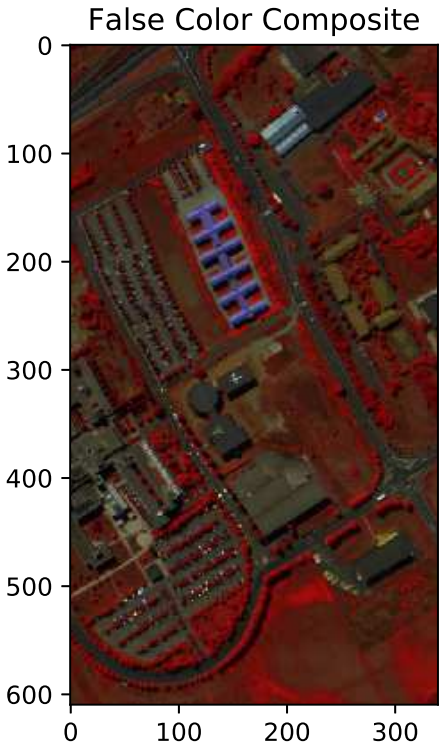
(c) Class Distribution of Pavia University

Number	Class	Samples
1	Asphalt	6631
2	Meadows	18649
3	Gravel	2099
4	Trees	3064
5	Painted metal sheets	1345
6	Bare Soil	5029
7	Bitumen	1330
8	Self-Blocking Bricks	3682
9	Shadows	947

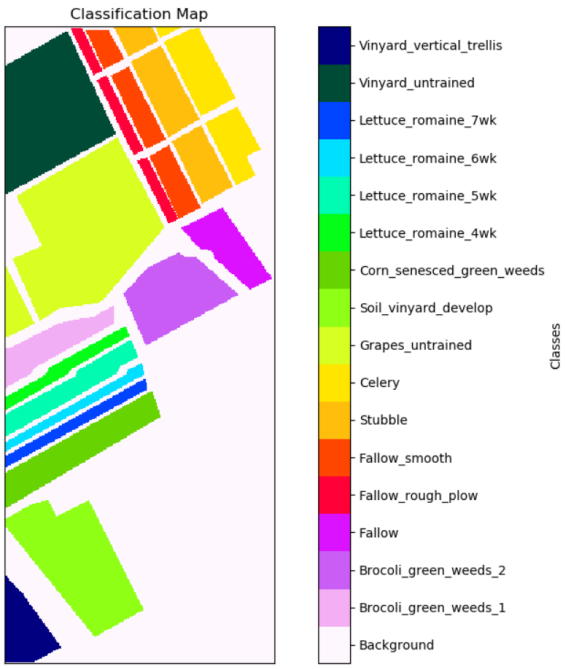
Table 3.1: Classes present in the Hyperspectral Datasets



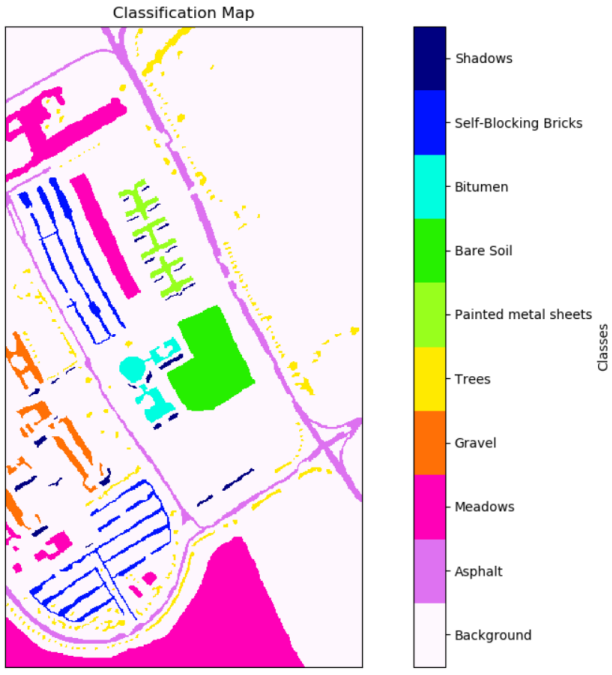
(a) Salinas FCC



(b) Pavia University FCC



(c) Salinas Ground Truth



(d) Pavia University GT

Figure 3.2: Salinas and Pavia University Datasets with Ground Truth

Chapter 4

Methodology

In this chapter we will discuss the approach used in an attempt to answer the questions posed in 3.1. The chapter is divided into two sections; the first section is used to design the algorithm for data visualization and manifold learning while the second section will cover the implementation details.

4.1 Algorithm

In the previous chapter, we have elaborated on the qualities that a Dimensionality Reduction algorithm should possess, which includes parameterization. A natural candidate for this is a fully connected neural network, which as we recall is a function mapping: $\mathcal{Y} = f(\mathcal{X}, w)$. An immediate impact of using a function mapping is that the initial embeddings, although random, still reflect some amount of spatial relation between the samples in $\mathcal{X}$. By employing neural networks, we run the risk of overfitting, which we limit by fixing the depth of the network to have only a single hidden layer. The objective function minimized by the neural network combines Laplacian Eigenmaps with a modification of t-SNE. This objective is minimized using any gradient descent optimization technique which operates on mini-batches of samples.

4.1.1 Revisiting Laplacian Eigenmaps

Laplacian Eigenmaps (Sec: 2.3) promotes clustering of samples in the reduced space. In a neural network setting, the output of our network is the embedding $\mathcal{Y}$, which is then plugged into the cost function of the Laplacian Eigenmaps; $\mathcal{J}(w) = \mathrm{Tr}(\mathcal{Y}^T \mathcal{L} \mathcal{Y})$. We note that the neural network maps $\mathcal{X} \xrightarrow{f(\mathcal{X},w)} \mathcal{Y}$. When doing matrix factorization of the LE objective function, the eigenvalues corresponding to the eigenvectors prevented the samples from collapsing to the centre of the embedding. Inspite of that, LE suffers

from crowding problem. In the network parameterized version of LE, we have no such constraint on the embeddings, and thus LE projects all points into a small, dense region. If we examine the objective function, $\mathcal{Y} = [1, 1, \dots]^T$ or $[0, 0, \dots]^T$ minimizes the objective without learning any useful information. To prevent this, we can impose the constraint: $\mathcal{Y}^T \mathcal{D} \mathcal{Y} = \mathcal{I}$. In our expirements, minimizing the objective function subject to the constraint proved difficult, with the loss failing to converge to a low value. To summarize, a neural network based Laplacian Embedding provided us with a powerful method to cluster the samples together in $\mathcal{R}^n$, but a method needed to be devised to keep dissimilar points apart.

4.1.2 Revisiting t-SNE

t-SNE (Sec: 2.5) is a well established technique for data visualization in two or three dimensions. But, a major drawback with the technique includes it's inability to deal with very high-dimensional data, such as the spectrum of our hyperspectral image. Practicioners of t-SNE initially reduce the dimensionality of the data to ~50 before applying t-SNE. As is the case with PCA, it tends to maintain the global structure at the expense of the local structure. By applying PCA as a pre-processing technique, we lose out some information on the local structure of the data. However, t-SNE provides powerful gradients that prevent the general collapse of samples into dense regions. The logical question arises, could we use this property of t-SNE such that it counters the disadvantages of neural network based Laplacian Eigenmaps?

There are few more steps left before combining Laplacian Embedding and t-SNE into a parameterized algorithm. The immediate hurdle is t-SNE's vulnerability to the curse of dimensionality 1.2. Recall that the curse of dimensionality states that variation between the nearest and farthest neighbour of a sample taken as a ratio to the nearest neighbour distance tends to zero as the dimensionality increases. t-SNE forms a probability distribution over other samples being a neighbour by considering the euclidean distance between the neighbours. If the Curse of Dimensionality holds true, $exp\,(-\frac{\|x_i - x_j\|^2}{2\sigma^2})$ will be similar for many values which are at a small to moderate distance away from the sample under consideration, thus giving a distorted interpretation of the manifold. As a trick to circumvent the Curse of Dimensionality, we define a new term, **Compression Factor**. Compression factor uses the Adjacency matrix generated in Laplacian Embedding to give an illusion of greater distance between samples in $\mathcal{X}$. To see how Compression Factor (CF) achieves it's goal, we recall that t-SNE converts euclidean distances into probabilities. Let us use the adjacency matrix that is computed in the Laplacian Embedding

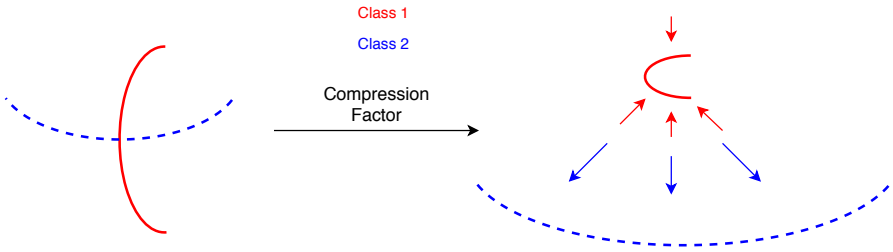


Figure 4.1: The Compression Factor can incorporate class labels if available to create an illusion of altering the manifold by compressing the space around the samples of the same class, causing points from the other class to become more distant. (Figure shows compression from the perspective of Class 1, similar compression happens with Class 2)

algorithm. Note that the adjacency matrix is computed using the k-Nearest Neighbouring points (other methods of computation will be discussed in later sections). If we artificially inflate the conditional probabilities corresponding to nearest points in p, an illusion is created that these points are closer together in $\mathcal{X}$ than they actually are. In a sense, we are compressing the distance in p by inflating the $p_{i|j}$ values for nearest neighbours. As a consequence of increasing select $p_{i|j}$ values, other probabilities $p_{k|j}$ will decrease, creating an illusion that these points are located farther away in $\mathcal{X}$ than they actually are. We compress the manifold for neighbourhood points as:

$$\tilde{p}_{i|j} = p_{i|j} * \{(CF - 1) * \mathcal{A} + 1\} \tag{4.1}$$

For values of $CF > 1$, we scale the adjacency graph (which is a 1 or 0 matrix) by $CF - 1$ and add 1 so that no element in $\mathcal{A}$ remains 0. We then take the hadamard product between our conditional probability matrix p and the resultant matrix of the previous step, and thus boosting the probability values associated with the adjacent points. After renormalizing p, we find that $\forall i \in nbr(j)$, $p_{i|j} \uparrow$, whereas $\forall k \in \{N_{-i}\}$, $p_{k|j} \downarrow$. This approach helps in limiting the effect of curse of dimensionality, by creating the illusion that the difference in sample distances are larger than they appear. Figure 4.1 shows pictorially the idea behind Compression Factor. This is an idealized case, with real world datasets having more complex interactions between various clusters present. We are more likely to see clusters formed by neighbouring points, as repulsion between moderate to distant points will be negated by different clusters.

We also modify t-SNE with the use of conditional probabilities instead of joint probabilities. t-SNE proposed the use of conditional probabilities over joint probabilities since the loss function is symmetric and easier to optimize. This, however weakens the magnitude of the gradient as seen in the paper [32]. The cause for weak gradients lies in the normal-

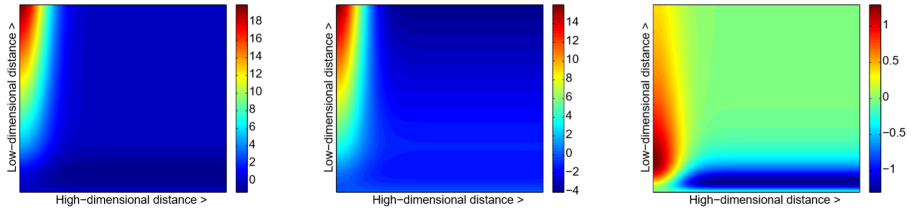


Figure 4.2: Colour map of Gradients for (L): SNE (M): UNI-SNE (R): t-SNE [32]

izing term, where p_{ij} is divided by $2N$, and q_{ij} is obtained by normalizing over all the elements in q. In comparison to $q_{i|j}$ and $p_{i|j}$, the values of q_{ij} and $p_{i|j}$ will be significantly smaller, which results in weaker gradients, which become insignificant when compared to Laplacian Embedding. Another reason for modification includes sensitivity of t-SNE to outliers. To prevent outliers from having a negligible impact on the cost function, t-SNE limits $p_{ij} > \frac{1}{2N}$, whereas for SNE, $p_{i|j} \rightarrow 0 \ \forall j$ if i is an outlier. As a part of our embeddings, we want p to have negligible effect if it is an outlier, which is more likely in the case of conditional probabilities in SNE. At this point, we would like to ponder on the unsymmetrical nature of the KL divergence. Could we use $KL(\,p\,\|\,q\,)$ and $KL(\,q\,\|\,p\,)$ to give different representations of the same data?

4.1.3 Batch Normalization

To accelerate the process of parameter learning by our algorithm, we include Batch Normalization [25] in our training process. We first take about Batch Normalization during training and the during inference.

Batch Normalization During Training

To understand the need for Batch normalization, let us revisit the network training process: Backpropagation (Sec: 2.8.2). The update of weights nearest to the target layer is performed first, followed by the penultimate layer of weights and so on. The weights are updated independent of those layers which precede it. These updates are in response to the distribution of the outputs of the previous layer. Hence, when the first layer weights are updated, the resultant output from the forward pass from the first layer will belong to a different distribution from the one the second layer of weights were tuned to. This variation in the outputs is propagated along the forward pass. Only when the change in the weights of the first layer is negligible, the second layer starts to learn and so on. This is beautifully depicted in Figure: 4.3.

Taking cue from the practice of normalizing inputs, we introduce a similar notion inside

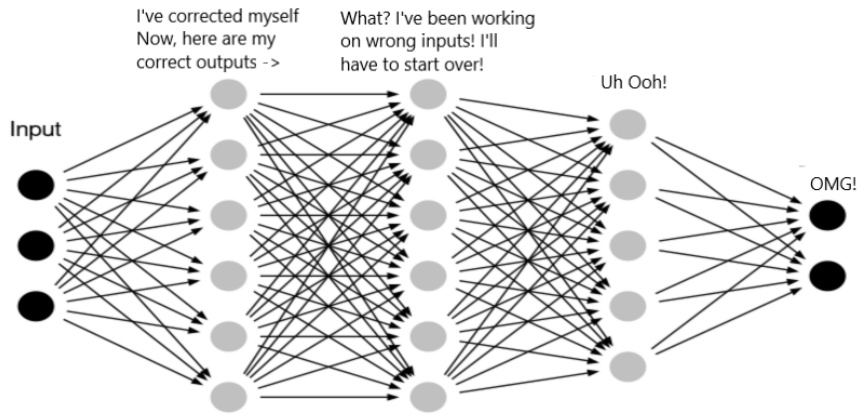


Figure 4.3: Learning process slows down due to the internal covariance shift in the distribution of the output neurons
https://medium.com/@ilango100/batch-normalization-speed-up-neural-network-training-245e39a62f85

the neural network. Thus, the first step in batch normalization is to scale the input to zero mean and unit variance along all dimensions independent of each other.

$$\hat{x}_j^i = \frac{x_j^i - \mu_j}{\sigma_j} \tag{4.2}$$

By normalizing the output of the layers, we are constraining the values that the output can take (most will lie in the range [-1 to 1]), and thus there is less shift in the distribution output. A careful observer will realize that in this range, all our activation functions are either linear (*tanh*, *sigmoid*), or they block half the inputs (ReLU). To provide a way for the network to adjust the output, we include two new learnable parameters per Batch Norm layer, γ and β. Our new output is thus: $y = \gamma\hat{x}_j + \beta$, which allows the network to modify the mean and variance which it deems optimal.

Batch Normalization During Inference

We borrow some terminology from Statistics, specifically the difference between samples and population. Samples as the name suggests are observations drawn from the population, whose statistics such as mean and variance are computable. Based on our sample statistics, we infer the population mean and variance. Our population is the unknown entity, which we are trying to approximate by using sample statistics. We also need to account for biases in the sampling process, such that our population parameters are not skewed.

This brings us to the question, is it appropriate to use the mean and variance as computed

in the testing batch, or compute the average mean and variances of all the training mini-batches for use during testing to avoid bias? We choose the latter option, as some testing batches could be skewed away from the population. Hence to estimate statistics during inference, we use Equation: 4.3.

$$\begin{aligned} \mathbb{E}_x &= \mathbb{E}_{\mathcal{B}}[\mu_b] \\ \sigma_x^2 &= \frac{m}{m-1}\,\mathbb{E}_{\mathcal{B}}[\,\sigma_{\mathcal{B}}^2\,] \end{aligned} \tag{4.3}$$

Note that the above equation computes an unbiased estimate. In practice, storing the statistics of each mini-batch may be difficult, and the use of moving averages of the statistics is prevalent.

Batchnorm has been proven to speed up training by reducing the number of iterations required to learn. It has also been shown that it acts as a weak regularizer to prevent overfitting, by injecting noise as the algorithm operates on mini-batches of data. Another aspect of Batchnorm that we greatly benefit from is the ability to change the mean and variance of our layer outputs. The computation of $q_{i|j}$ matrix with a gaussian distribution works on an arbitrary σ hyperparameter, and the output of our network can easily scale to to fit this distribution with the help of Batch Normalization. With the discussion so far, we can make an attempt to answer the first two parts of our problem statement, manifold visualization and supervised sample clustering.

4.1.4 Manifold Visualization

The algorithm to visualize the manifold is straightforward. We load the dataset and normalize each band to have zero mean and unit variance. After removing the background samples, we split the dataset into 50% training and 50% testing (Validation split could be included in the training set) and shuffle so as to increase the stochasticity of the network inputs. Recall that our network is parameterized by w, hence our objective function is:

$$w^* = arg\min_w \mathbb{E}_x \left(\mathcal{Y}^T \mathcal{L} \mathcal{Y} + \sum_{i,j} \tilde{p}_{i|j}\, log\, \frac{\tilde{p}_{i|j}}{q_{i|j}} \right) \text{ where } \mathcal{Y} = f(\mathcal{X}, w) \tag{4.4}$$

Using monte carlo approximation; where $\mathbb{E}_{\mathcal{X}}[f(x)] = \int_x f(x)\,dx = \frac{1}{M}\sum_m [f(x)]$, our objective Equation 4.4 is reduced to:

$$w^* = arg\min_w \frac{1}{M} \sum_m \left(y_m^T \mathcal{L} y_m + \sum_{(i,j) \in m} \tilde{p}_{i|j}\, log\, \frac{\tilde{p}_{i|j}}{q_{i|j}} \right) \text{ where } m : \text{mini-batch} \tag{4.5}$$

This objective could be minimized using any standard Gradient Descent Optimization algorithm such as Adam Optimizer. The network architecture is shown in Figure 4.4.

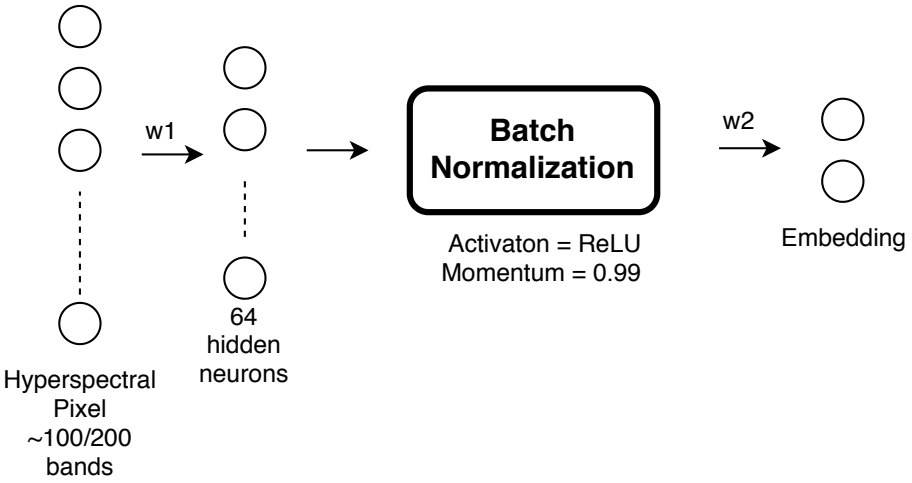


Figure 4.4: A simple one hidden layer architecture with fewer parameters is chosen so as to minimize the chances of overfitting

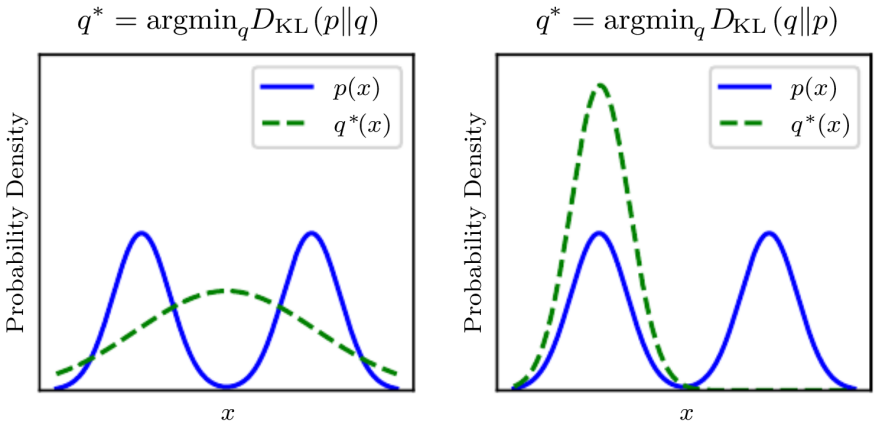


Figure 4.5: Difference in minimizing $KL(p\,\|\,q)$ and $KL(q\,\|\,p)$ where q tries to approximate p. [15]

4.1.5 Labelled Manifold Clustering

Let us revisit KL divergence, which is a similarity measure between two probability distributions. An interesting property of KL divergence is that it is not symmetric; implying $KL(p\,\|\,q) \neq KL(q\,\|\,p)$. The consequence of this is shown in Figure 4.5. The left side of the image tries to place a higher probability mass of q where p has a larger mass; the right hand side shows depicts lower probability mass of q coincides with a low probability mass of p. The latter has interesting applications for clustering. With the use of Compression Factor, we can squeeze the probability mass in p such that connected points have a high value of $p_{i|j}$ which forces the remaining points have a low value. If we try to minimize $KL(q\,\|\,p)$, large portions of p will have small values; the remaining points being connected to each other and having boosted probabilities. In this scenario, $KL(q\,\|\,p)$

will try to impose low probabilities in the region between the clusters in p, thereby giving rise to dense clusters in q.

In this supervised approach, we attempt to use the global structure of the manifold to cluster the samples in our embedding. Instead of computing the adjacency matrix with k-Nearest Neighbours, we use class information to denote connected points. We seperate samples of the same class appearing in the mini-batch and connect edges between them. This gives rise to graph clusters in the original space. The graph adjacency is then used to compute the graph laplacian for the Laplacian Embedding and also supplied to the inverted t-SNE for use along with the Compression Factor. A question may arise to the reader, are we not losing out on spatial information between the samples when using class labels in Laplacian Embedding? Luckily for us, the neural network provides spatial mapping between $\mathcal{X}$ and $\mathcal{Y}$, while the Laplacian Embedding objective encourages clustering between points of the same label. Note that we have not used Classification gradients at any point to cluster our samples, due to its tendency to destroy the manifold structure and overfit to one particular task. We allow the algorithm to find structure within the manifold which supports clustering by inverting the KL divergence. The objective function (Equation: 4.5) is now modified to:

$$w^* = arg\min_{w} \frac{1}{M} \sum_{m} \left(y_m^T \mathcal{L} y_m + \sum_{(i,j) \in m} q_{i|j} \, log \, \frac{q_{i|j}}{\tilde{p}_{i|j}} \right) \text{ where } m : \text{mini-batch} \quad (4.6)$$

4.1.6 Unlabelled Manifold Clustering

In this section, we try to explore clustering of our samples in $\mathcal{X}$ in the absence of labelled information. Could we explore some alternative ways of coming up with an Adjacency matrix such that we could utilize the objective in Equation 4.6? We turn our attention to the first law of geography as coined by Waldo Tobler [49]:

> "Everything is related to everything else, but near things are more related than distant things"

Hyperspectral Images taken from airborne platforms such as UAVs enjoy very high spatial resolution. In our datasets that we use, we have spatial resolution of $10m$. At this scale, contiguous pixels share similar properties such as being associated with the same object. We use the first law of geography to draw similarity between the samples in lieu of labelled information. To do this, we dive into the concept of Image Segmentation. Image segmentation as the name suggests involves identify regions in the image with similar properties with the help of algorithms. These algorithms can be broadly divided into two

segments, Region Growing and Region Splitting. Region growing involves the process of merging small local regions which share common properties. This continues until the remaining regions share no common properties. We could see Region merging as a tree traversal from leaf to root node. Region splitting starts with a large heterogenous region, where the splits are made if the region is not homogenous. This continues until all the remaining regions are homogenous, or the threshold for the smallest size a region could have is reached. Region splitting can be seen as a tree traversal from the root node to it's various leaf nodes. In this chapter, we discuss two algorithms:

1. Watershed Algorithm: Segmentation of Grayscale images.
2. SLIC algorithm: Segmentation of 3-Channel (typically RGB) images.

By using image segmentation algorithms, we identify disjoint regions in the image which share common properties, and thus are considered connected componenets when computing the adjacency matrix, when labelling is unavailable.

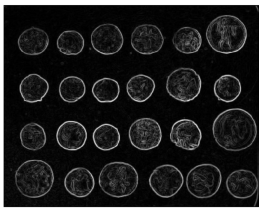
Watershed Algorithm

The Watershed algorithm [5] is an Image Processing technique based on Grayscale Morphology. It considers the image as a topology, where the various regions are considered as catchment areas that are determined by flooding the topology. Large grayscale values are considered peaks, and smaller grayscale values are considered as valleys. Markers are either picked by choosing local minimas or can be separately provided by the user. The gradient of the topology is then computed using morphological gradient: $\nabla_u = (u \oplus D) - (u \ominus D)$. The topology is then flooded via the markers, with the rate of flooding determined by the topology. When flooding from two different markers come in contact, a boundary is formed between the two different regions. Watershed algorithm delivers quick and optimal results when the grayscale image has well define boundaries. We also label disjoint regions separately, with the help of binary morphological operations depending on whether the regions have 1-connectivity or 2-connectivity. In our dissertation, we use grayscale Watershed morphology for performing clustering on the Salinas dataset.

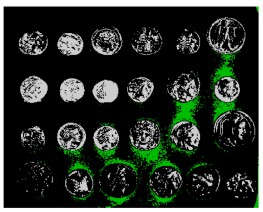
Simple Linear Iterative Clustering Algorithm

SLIC [2] is a computationally cheap image segmentation algorithm which identifies superpixel in 3-channel images. Superpixels are small homogeous regions, consisting of a collection of pixels from the images. SLIC combines colour and spatial image into a single 5-Dimensional coordinate, which is expressed as: $[R_x, G_x, B_x, d_x, d_y]$. Instead of

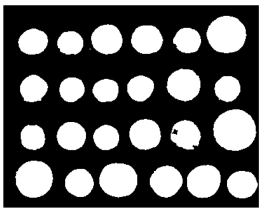
(a) Coins - Grayscale Image



(b) Elevation Map - Coins



(c) Markers corresponding to Coins



(d) Watershed segmentation of Coins

Figure 4.6: Watershed algorithm provides excellent results when the image has sharp, clean edges as shown in (a). This gives rise to a sharp topology as shown in (b). For this example, we manually selected the range of gray levels which will act as markers as shown in (c). Flooding from various markers does not cross the boundaries of our coins due to the sharp gradients present in the topology, giving rise to clean segments. https://scikit-image.org/docs/dev/auto_examples/applications/plot_coins_segmentation.html

using Euclidean distance in 5-D space, the authors propose a new metric (Equation 4.7). Here, m is a parameter which controls the tradeoff between spatial and colour distance. As the value of m increases, spatial distance dominates the distance metric, and vice versa. S is the average distance between the centre of two superpixels, and is given by: $S = \sqrt{\frac{N}{K}}$, where N is the number of pixels in the image and K is a user defined parameter on how many superpixels to define.

$$\begin{aligned} d_{rgb} &= \left((r_i - r_j)^2 + (g_i - g_j)^2 + (b_i - b_j)^2 \right)^{0.5} \\ d_{xy} &= \left((x_i - x_j)^2 + (y_i - y_j)^2 \right)^{0.5} \\ d &= d_{rgb} + \frac{m}{S} d_{xy} \end{aligned} \tag{4.7}$$

The algorithm first initializes the superpixels uniformly and shifts them and searches in a $3x3$ neighbourhood the pixel having the least gradient. These location are chosen as centres of the superpixels at $t^{(0)}$. We now apply the regular k-Means algorithm with the distance metric as defined in Equation 4.7, to give us our superpixels.

Adapting Watershed and SLIC to LEt-SNE

Watershed algorithm works with grayscale images, whereas SLIC works with 3-channel images. To make our dataset, which has $100s$ of dimensions compatible with the segmentation algorithms, we suggest two approaches:

1. Watershed: Using a grayscale version of the False Colour Composite, or the first Principal Component after applying PCT to the dataset.
2. SLIC: Using False Colour Composite directly, or concatenating the first three principal components of PCT.

For Salinas; we have used the grayscale version of the FCC *(Y = 0.2125*R + 0.7154*G + 0.0721*B)* for image segmentation with watershed algorithm. With the Indian Pines and the Pavia University dataset, we have used the 3-band concatenation of the Principal Component Transformation as the image to segment. Our justification for PCA resides in the fact that it tries to explain the variance in the dataset in an unsupervised environment, thus helping identify regions which could not be visually perceived with the False Colour Composite. A common problem with Watershed and SLIC is oversegmentation of the image. To tackle oversegmentation in SLIC, we use Region Adjacency Graph or *RAG*. RAG, as the name suggests creates a graph where regions are represented by their labels $\{\mathcal{V}\}$; the colour dissimilarity between them forming the weights on the edges $\{\mathcal{E}\}$. Similar coloured regions have low weights on the edges compared to dissimilar regions. Given a threshold, Graph Cut combines all the vertices in RAG having with weights smaller than the threshold. Thus, regions that are different are kept separate, whereas similar regions are combined into a single region. The use of RAG + Graph Cut significantly reduces the oversegmentation performed by SLIC.

4.1.7 Summary

In this chapter, we have desribed *LEt-SNE* in detail. We have combined the strengths of Laplacian Embedding and t-SNE; and wrapped it in a parametric form. By performing mini-batch optimization, we significantly speed up the processing time required by the algorithm. A new hyperparameter, *Compression Factor* has been proposed to tackle the Curse of Dimensionality. Our algorithm can operate in three settings: Manifold Visualization, Supervised Clustering as well as Unsupervised clustering. We have demonstrated how labels could be included in Laplacian Eigenmaps and t-SNE. In the absence of labels, we provide alternatives based on the spatial similarity shared by regions in the image. The unsupervised approach uses segmentation to identify different regions in the image which could be used to compute the adjacency graph instead of k-Nearest Neighbour. For this dissertation, we have used two pre-existing segmentation algorithms; Watershed and SLIC to provide dissimilar regions.

Chapter 5

Results and Discussions

The results are divided into three sections; Manifold Visualization, Manifold Based Clustering (Supervised) and Manifold Based Clustering (Unsupervised). For Manifold visualization, since there is no popular metric to analyze the quality of our representation, we use visual assesment to determine the suitability of the representation. Metrics for the remaining two sections involves Overall Accuracy (OA), Confusion Matrix and *kappa*. Confusion Matrix extends the concept of True and False Positives to n-classes. Kappa is an accuracy metric that accounts for class imbalances and random guesses by the classifier. A higher value indicates better classification. A loose interpretation of the various kappa values [3] are:

- 0: agreement equivalent to chance
- 0.1 – 0.20: slight agreement
- 0.21 – 0.40: fair agreement
- 0.41 – 0.60: moderate agreement
- 0.61 – 0.80: substantial agreement
- 0.81 – 0.99: near perfect agreement
- 1: perfect agreement

To ensure repeatability and consistency, each experiment has been performed five times, with the average of them being reported in the results.

5.1 Manifold Visualization

The hyperparameters used for Manifold Visualization are shown in Table: 5.1. We first visualize the manifold in two dimensions for the Indian Pines dataset. This is displayed in Figure 5.1.

[3] https://www.statisticshowto.datasciencecentral.com/cohens-kappa-statistic/

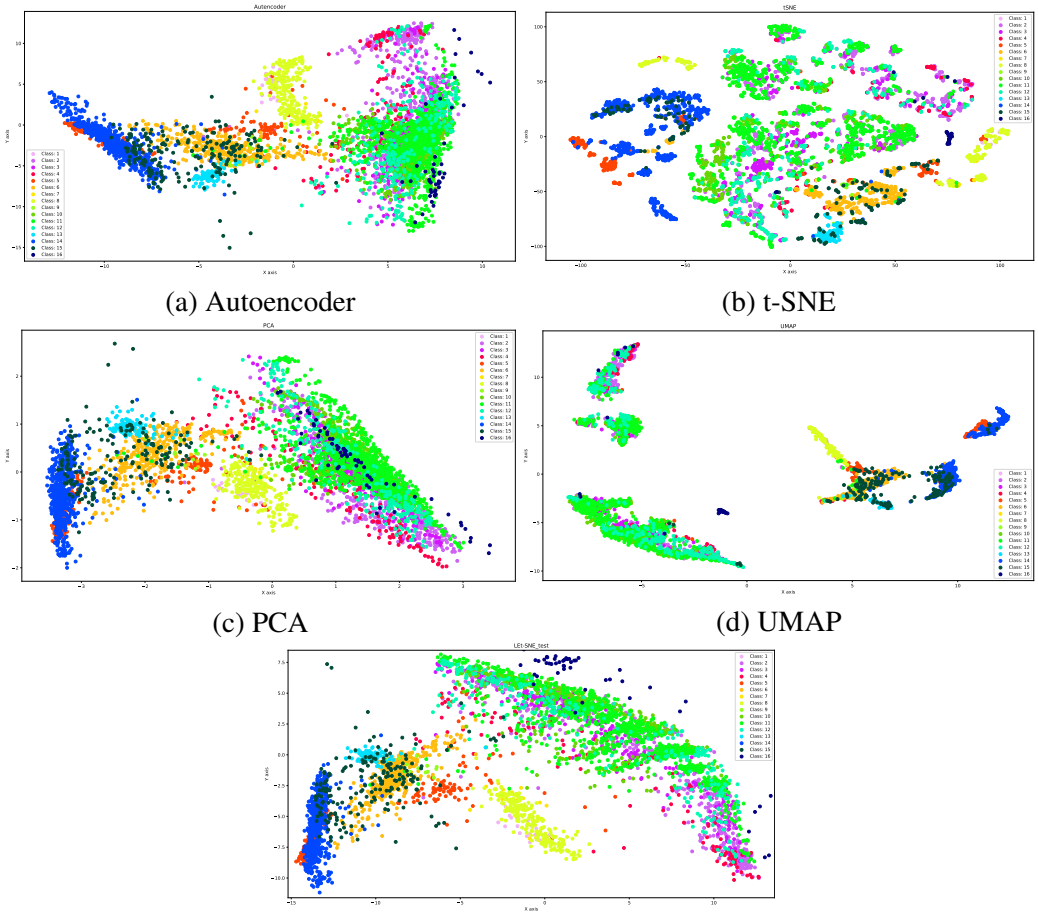
Hyperparameters	Values
Train:Test Split	0.5
Epochs	100
Perplexity	25
Compression Factor	5
Batch Size	1024
$Loss_{LE}$: $Loss_{t\hat{S}NE}$	1:100
Number of Neighbours	$\frac{batch\_size}{5 * num\_classes}$

(a) Manifold Visualization

Hyperparameters	Values
Train:Test Split	0.5
Epochs	300
Perplexity	50
Compression Factor	200
Batch Size	1024
$Loss_{LE}$: $Loss_{t\hat{S}NE}$	1:100
Number of Neighbours	-

(b) Manifold Based Clustering

Table 5.1: Hyperparameter selection for LEt-SNE



(a) Autoencoder

(b) t-SNE

(c) PCA

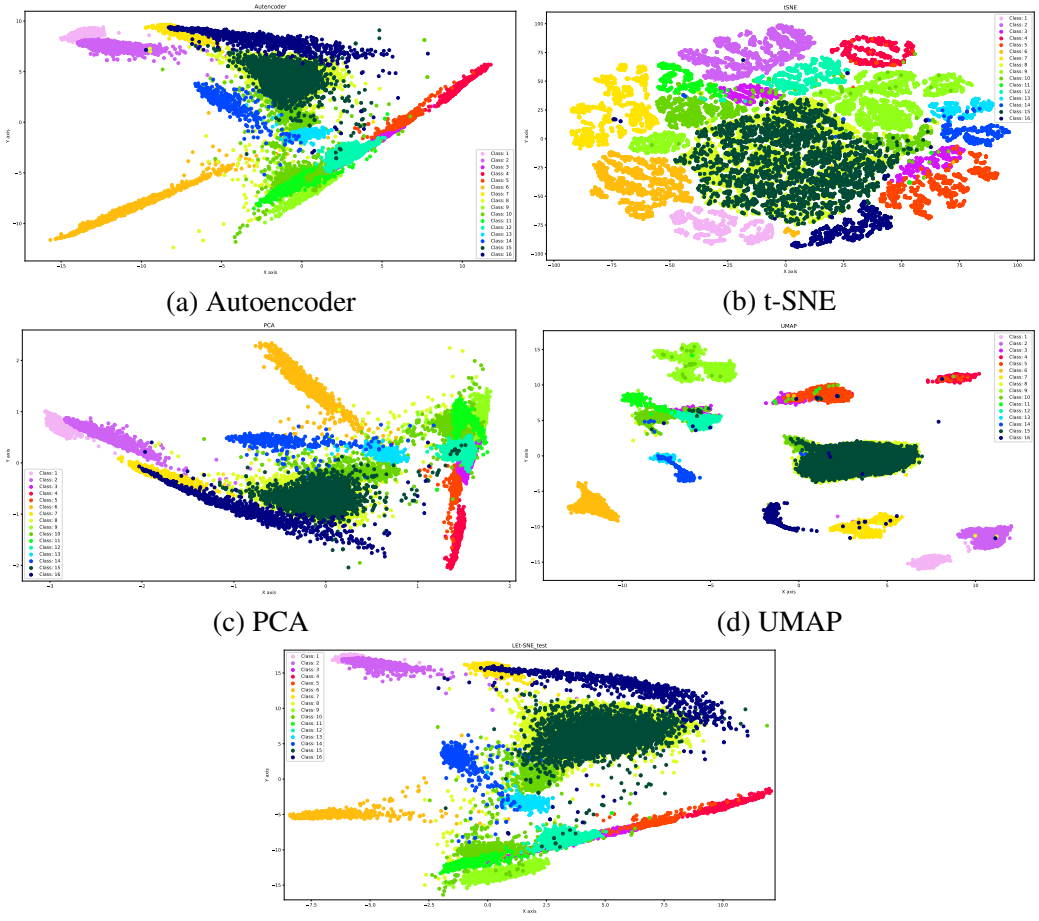
(d) UMAP

(e) LEt-SNE

Figure 5.1: Manifold Visualization For Indian Pines

5.1.1 Indian Pines

Indian Pines has 16 classes[4] with not all classes being mutually exclusive. This is seen in Figure 5.1. Our first observation shows that PCA, Autoencoder, UMAP and LEt-SNE show similar manifold characteristics, such as elongated strips of *wood* on one side and *soybean variants* on the other side; indicating significant merger of classes. Inspite of not using a linear decoder, the autoencoder learns a similar low dimensional representation as PCA. UMAP manages to cluster the classes together, but lacks the fine structure as shown in LEt-SNE. If we look at t-SNE, the spatial relation between the classes is not clear. LEt-SNE provides information on the spatial spread of the classes along with the fine local structure.



(a) Autoencoder (b) t-SNE

(c) PCA (d) UMAP

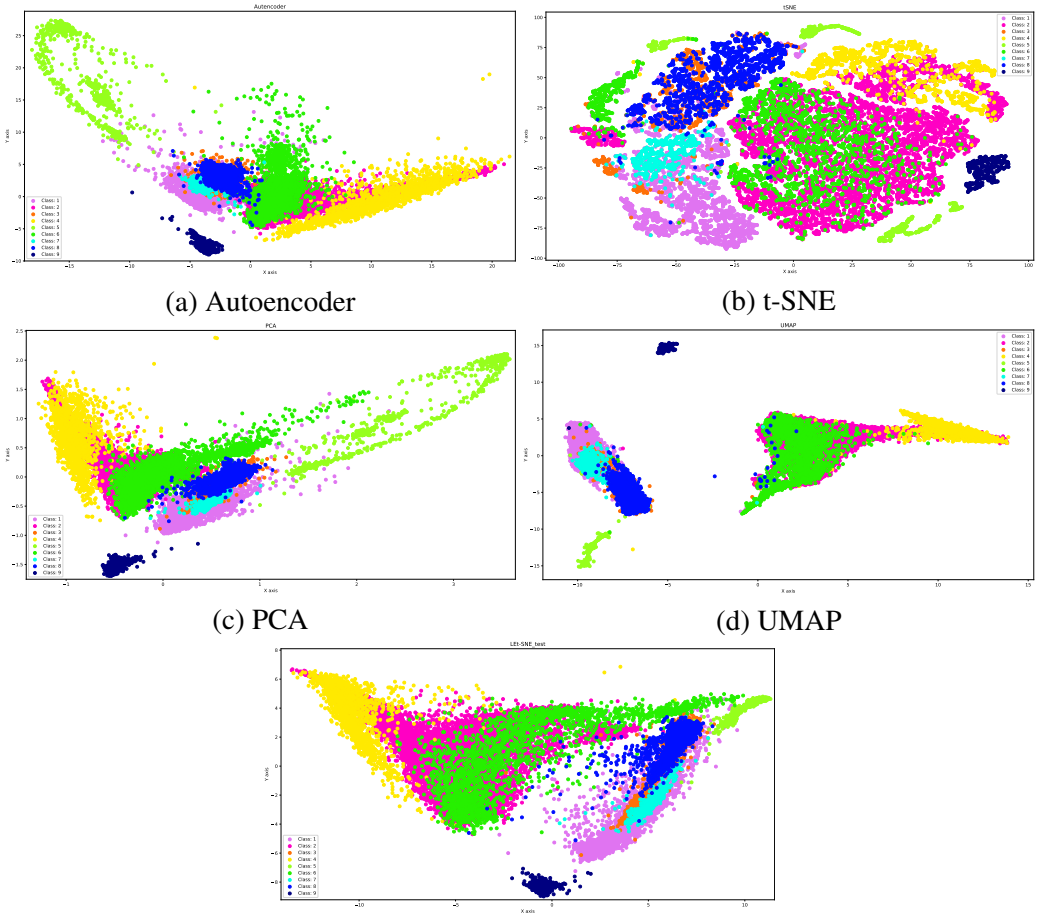
(e) LEt-SNE

Figure 5.2: Manifold Visualization For Salinas

[4] 1: 'Alfalfa', 2: 'Corn-notill', 3: 'Corn-mintill', 4: 'Corn', 5: 'Grass-pasture', 6: 'Grass-trees', 7: 'Grass-pasture-mowed', 8: 'Hay-windrowed', 9: 'Oats', 10: 'Soybean-notill', 11: 'Soybean-mintill', 12: 'Soybean-clean', 13: 'Wheat', 14: 'Woods', 15: 'Buildings-Grass-Trees-Drives', 16: 'Stone-Steel-Towers'

5.1.2 Salinas

We now turn our focus to Salinas Dataset[5], which has a significant overlap between class *Grapes untrained* and *Vineyard untrained*. The manifold visualization is shown in Figure: 5.2. We notice that here too, PCA, Autoencoders and LEt-SNE have a largely similar structure. This could be because of the fact that hyperspectral pixels have very high correlation, which causes PCA to excel. Unlike with Indian Pines, we do not see any local structure being exhibited by our dataset. UMAP manages to again cluster our data, but completely overlaps Class 8 and 15.



(a) Autoencoder (b) t-SNE

(c) PCA (d) UMAP

(e) LEt-SNE

Figure 5.3: Manifold Visualization For Pavia University

[5] 1: 'Brocoli green weeds 1', 2: 'Brocoli green weeds 2', 3: 'Fallow', 4: 'Fallow rough plow', 5: 'Fallow smooth', 6: 'Stubble', 7: 'Celery', 8: 'Grapes untrained', 9: 'Soil vinyard develop', 10: 'Corn senesced green weeds', 11: 'Lettuce romaine 4wk', 12: 'Lettuce romaine 5wk', 13: 'Lettuce romaine 6wk', 14: 'Lettuce romaine 7wk', 15: 'Vinyard untrained', 'Vinyard vertical trellis'

5.1.3 Pavia University

We now turn our attention to Pavia University[6], which again exhibits a merger between various classes. Although class imbalances were present in the previous two datasets, Pavia University exhibits an extreme case where the class *Meadows* dominates with ~18000 samples. Figure 5.3 compares the data visualization between other algorithms and LEt-SNE. Yet again, PCA, Autoencoder, UMAP and LEt-SNE share similarities. One observation is that LEt-SNE and UMAP are more *compact* than Autoencoder and PCA. While PCA tilts heavily towards retaining the global structure, the Compression Factor in LEt-SNE could have brought the samples closer as explained in 4.1.2. Another observation is that UMAP and PCA both show class *Bare soil* as a convex structure trailing the triangular border of class *meadows*, a detail not accurately captured by PCA and Autoencoder. All the visual representations of Pavia University show a considerable overlap between various clusters, which explains why there is ambiguity between samples of certain classes.

5.1.4 General Observations

As part of multiple experiments, we found that t-SNE did not provide credible spatial relations between classes. Our motive behind data visualization is that representations in lower dimensions should faithfully represent the relation between pixels in higher dimensions. t-SNE did not show repeatability between various simulations of dimensionality reduction. t-SNE is a non-convex optimization and hence could encounter local minimas which are different in each simulation. At the maximum, these local minimas should differ by rotation / reflection / translation / inversion or similar transforms. However with t-SNE, the spatial relation between classes completely changes. In one simulation, class A and B could be adjacent, whereas in other simulations they could be diagonally opposite, which is not desirable in a dimensionality reduction algorithm. Certain attempts to solve this problem include reducing the original data with PCA before applying t-SNE, but this could lead to losing out on local manifold structures which PCA does not preserve.

Unlike t-SNE, *perplexity* does not play the role of determining the number of neighbours a sample has in LEt-SNE. This role is fulfilled by the *Number of Neighbours* hyperparameter which we have included as a part of LEt-SNE. Combined with Compression Factor, we could manipulate how our embeddings look like. In LEt-SNE, perplexity plays the role of determining the scale of our embeddings. For the task of manifold visualization,

[6] 1: 'Asphalt', 2: 'Meadows', '3: Gravel', 4: 'Trees', 5: 'Painted metal sheets', 6: 'Bare Soil', 7: 'Bitumen', 8: 'Self-Blocking Bricks', 9: 'Shadows'

it is more suitable to keep a low value for the number of neighbours hyperparameter, as well as a low value of compression factor (~5).

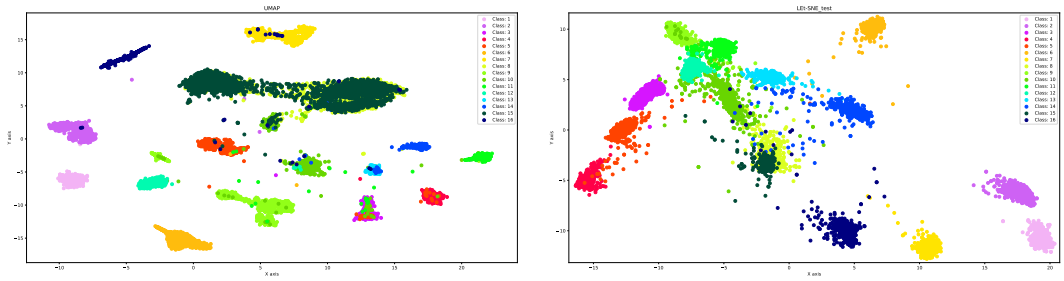
5.2 Manifold Based Labelled Clustering

We compare LEt-SNE with a supervised variant of UMAP which is considered a powerful dimensionality reduction tool. We also compare it with other unsupervised approaches to determine till what extent has labelling information helped in clustering our datasets. The hyperparameters that we have used for Clustering are shown in Table: 5.2.

We compare the Overall Accuracy (SVM + Neural Network) and Kappa (SVM) values for the different algorithms. We also include confusion matrices to help determine the suitability of each algorithm.

Table 5.2: Accuracy comparison: Salinas

Metric	**LEt-SNE** (sup)	UMAP (sup)	Autoencoder	PCA	UMAP (unsup)
SVM (OA)	**0.9286**	0.899	0.8358	0.8296	0.8524
NeuralNet (OA)	**0.9303**	0.9014	0.841	0.8401	0.86272
Kappa (κ)	**0.9234**	0.8876	0.8178	0.811	0.8361

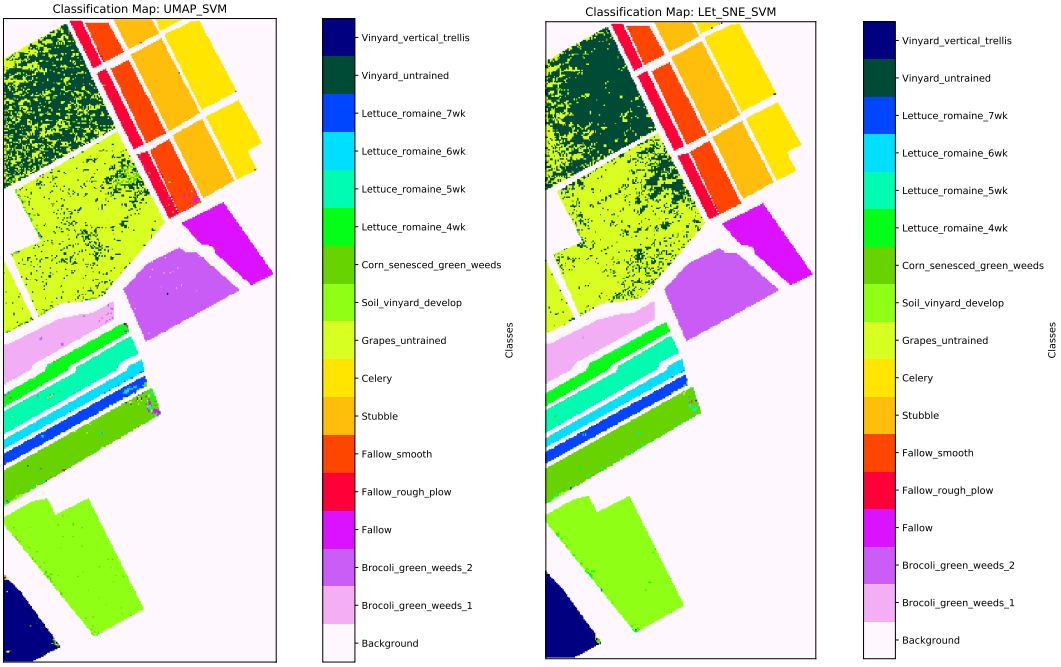


(a) Salinas Embedding: UMAP (b) Salinas Embedding: LEt-SNE

Figure 5.4: Comparison between the Embeddings of UMAP and LEt-SNE supervised varaints

5.2.1 Salinas

Table 5.2 compares our algorithm with the supervised variant of UMAP along with other unsupervised approaches. We notice that our supervised approach outperforms the rest with a significant margin. To ascertain the reason behind this, we compare the classification maps, embeddings and confusion matrix of the supervised variants of LEt-SNE



(a) Salinas Classification (SVM): UMAP (b) Salinas Classification (SVM): LEt-SNE

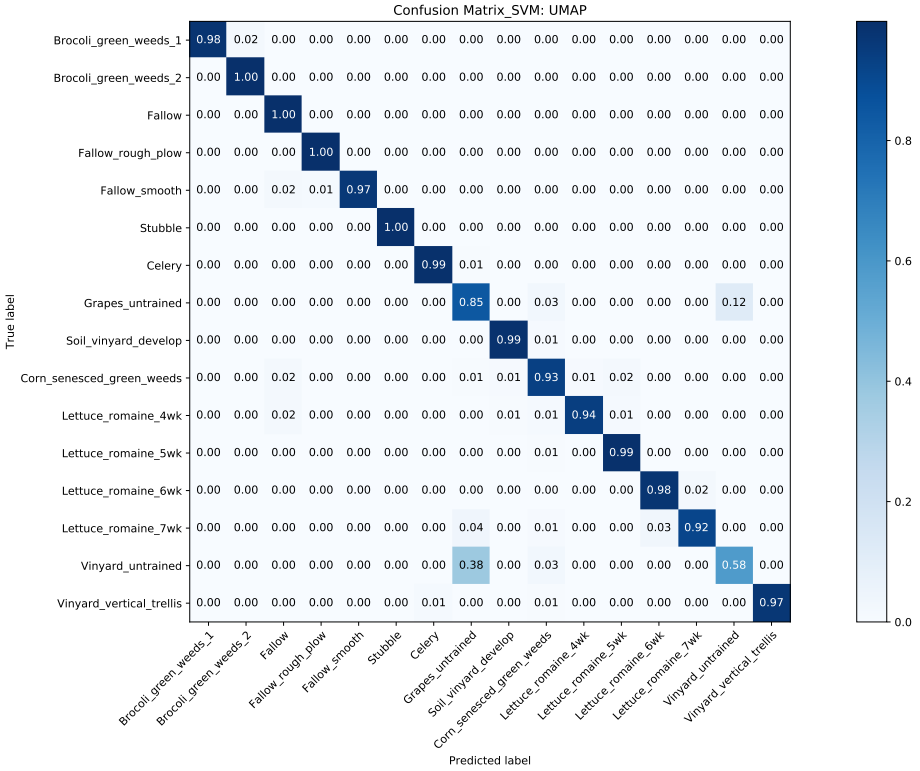
Figure 5.5: Comparison between the SVM classification maps of UMAP and LEt-SNE (supervised)

and UMAP. Figure 5.6 shows the confusion matrix for the two algorithms. From the confusion map, it is evident that UMAP could not succesfully discern *Grapes Untrained* from *Vinyard Untrained* and confuses between the two. In comparison, LEt-SNE does a much better job separating the two classes, with a higher True Positive rate. The embeddings for UMAP and LEt-SNE are shown in Figure 5.5. The embeddings confirm the reasoning, since UMAP embeddings show a complete overlap between classes 8 and 15, as opposed to the partial overlap between the classes in LEt-SNE. LEt-SNE also shows a better clustering quality in comparison to UMAP. To complete the discussion, we compare the SVM classification maps between UMAP and LEt-SNE in Figure: 5.4.

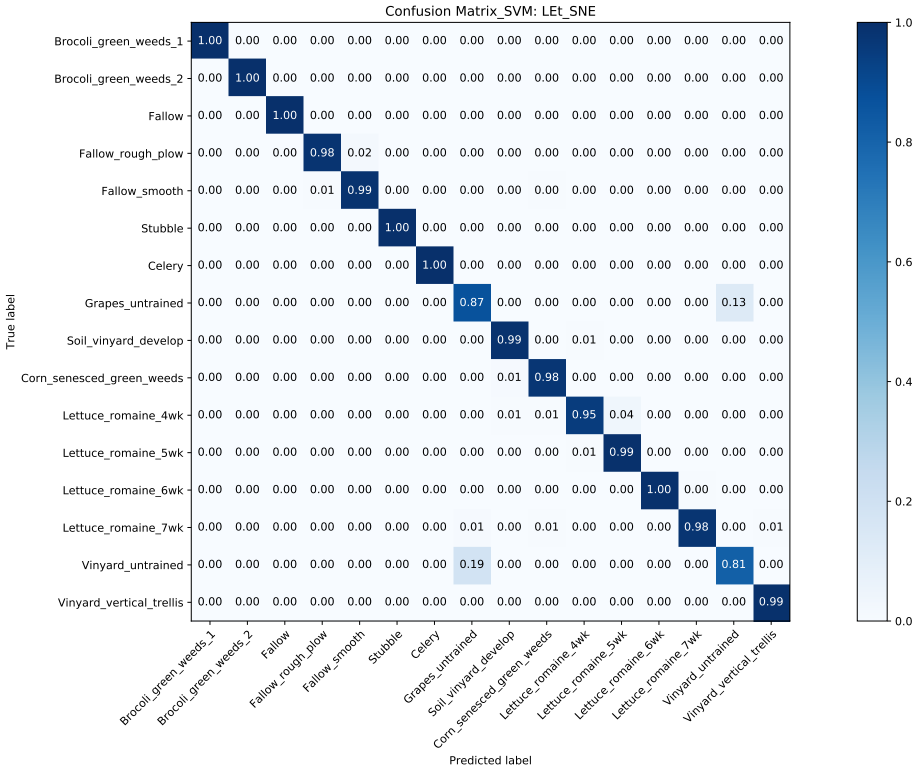
From the classification maps, we see that UMAP overcommits to *Grapes Untrained*, and thus encounters a significant number of misclassification driving it's kappa score down. LEt-SNE, on the other hand is better at resolving the two classes.

5.2.2 Pavia University

We compare our results for the Supervised Manifold based Clustering in Table: 5.3. LEt-SNE outperforms the supervised variant of UMAP by a significant margin. This is a consequence of the algorithm being able to separate out various classes into distinct clus-



(a) Salinas Confusion Map: UMAP

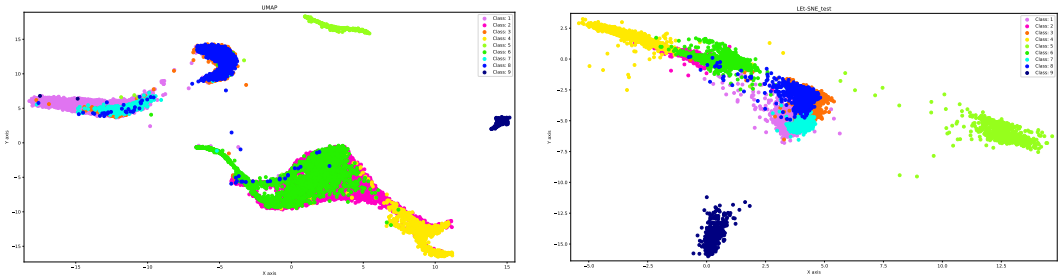


(b) Salinas Confusion Map: LEt-SNE

Figure 5.6: Comparison between the Confusion Matrix of UMAP and LEt-SNE supervised variants

Table 5.3: Accuracy comparison: Pavia University

Metric	**LEt-SNE** (sup)	UMAP (sup)	Autoencoder	PCA	UMAP (unsup)
SVM (OA)	**0.89**	0.7539	0.6976	0.6754	0.7113
NeuralNet (OA)	**0.9065**	0.808	0.7967	0.801	0.7863
Kappa (κ)	**0.8561**	0.6863	0.6242	0.6027	0.6385



(a) Pavia University Embedding: UMAP (supervised)

(b) Pavia University Embedding: LEt-SNE (supervised)

Figure 5.7: Comparison between the Embeddings of UMAP and LEt-SNE (supervised)

ters. We investigate the result in detail by examining the Confusion Matrix in Figure 5.10. Classes *Meadows* and *Self-Blocking Bricks* have a higher misclassification rate in UMAP than in LEt-SNE. Since *Meadows* class forms the bulk of the samples in Pavia University, this drives down the classification accuracy of UMAP. We can probe further into the reason behind this misclassification by looking at the nature of embeddings computed by UMAP and LEt-SNE, which are displayed in Figure 5.7. Even with Classification Labelling, UMAP compresses multiple classes into common clusters, thus detoriating the classification performance. In comparison, LEt-SNE's objective forces dissimilar classes to remain at larger distances in the reduced space. A close-up view of the cluster containing classes 3, 1, 7, 8 shows that although the classes are grouped together, each class has it's own cluster; albeit with partial overlap between neighbours. Figure 5.8 confirms the ability of LEt-SNE to resolve finely separated classes.

We confirm our findings by inspecting the embeddings generated by UMAP and LEt-SNE (Figure 5.7). Even in a supervised use, UMAP is unable to separate out various classes. In constrast, LEt-SNE makes a visible effort to group datapoints into meaningful clusters which lead to a much better visual interpretation of the classification process.

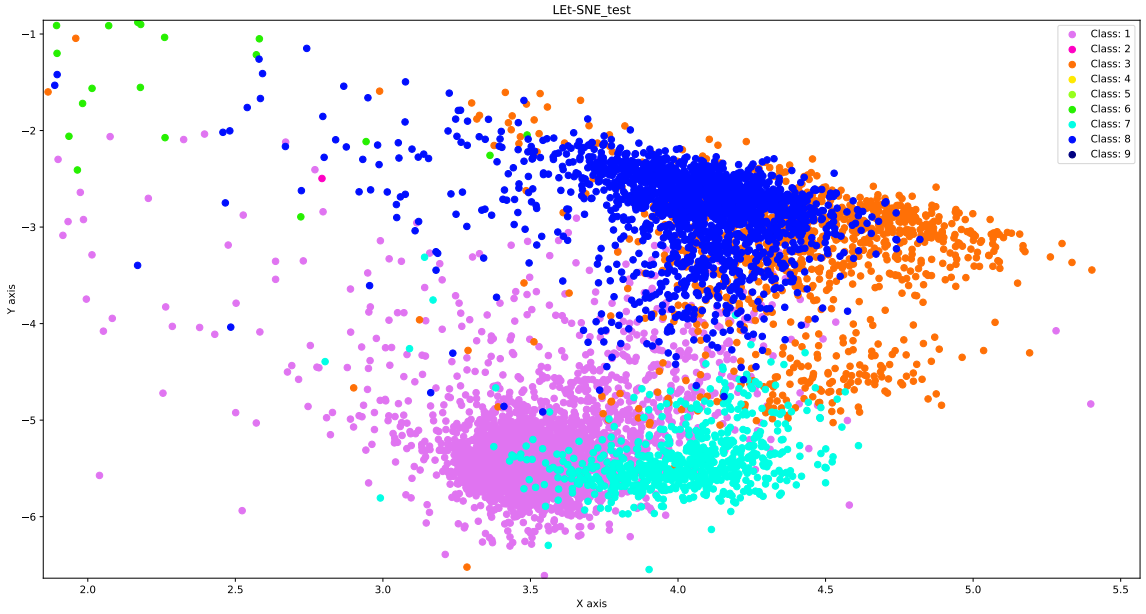


Figure 5.8: Pavia University: LEt-SNE shows an increased ability to resolve similar classes in comparison to other approaches

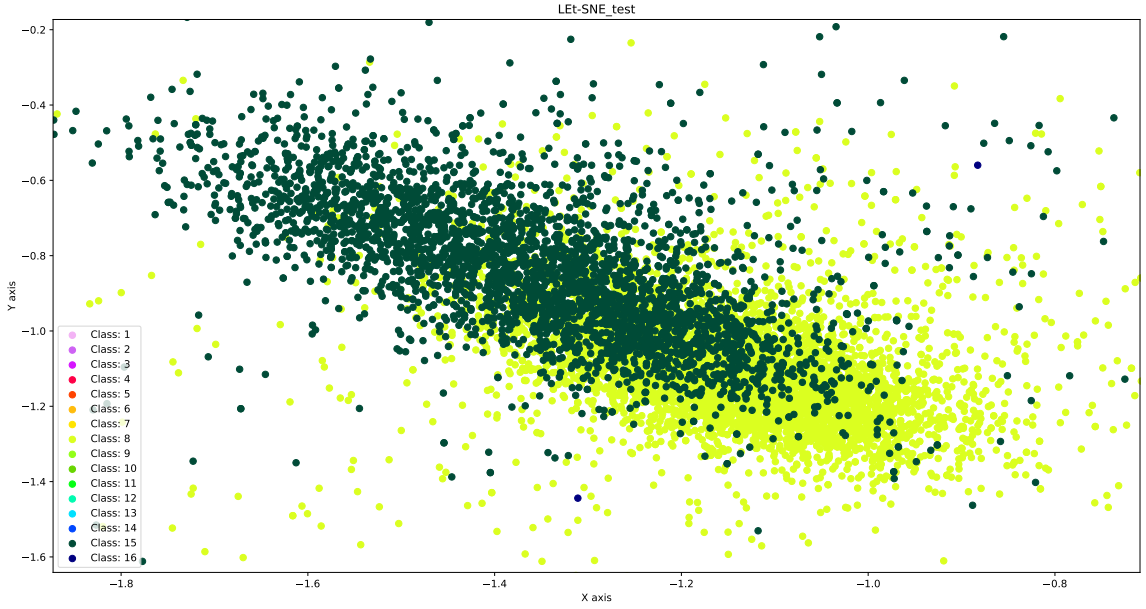


Figure 5.9: Salinas: LEt-SNE able to partially resolve similar classes *Grapes Untrained* and *Vineyard Untrained*

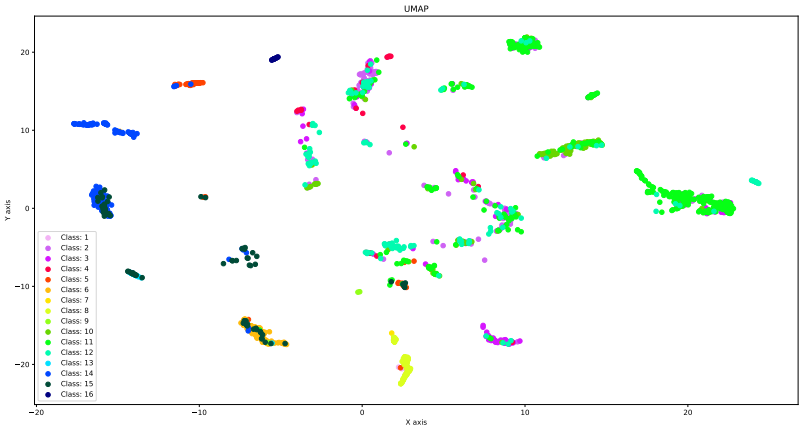
(a) Pavia University Confusion Map: UMAP

(b) Pavia University Confusion Map: LEt-SNE

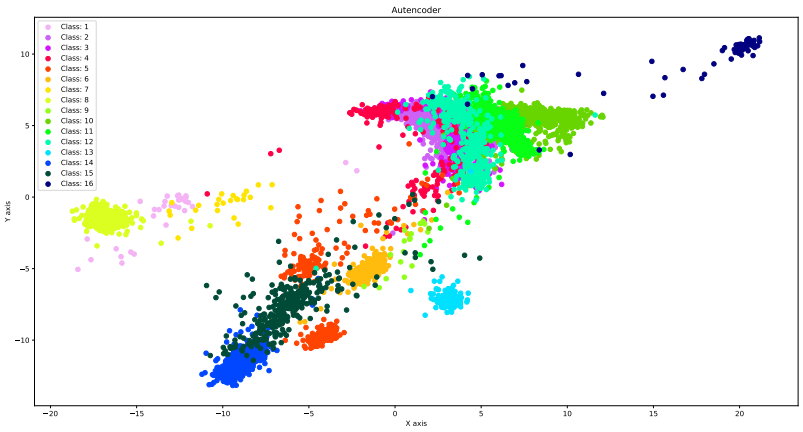
Figure 5.10: Comparison between the Confusion Matrices of UMAP and LEt-SNE (supervised)

Table 5.4: Accuracy comparison: Indian Pines

Metric	**LEt-SNE** (sup)	UMAP (sup)	Autoencoder	PCA	UMAP (unsup)
SVM (OA)	**0.7373**	0.7305	0.5459	0.4927	0.5685
NeuralNet (OA)	**0.7682**	0.7405	0.6188	0.5912	0.6074
Kappa (κ)	**0.7053**	0.6909	0.4916	0.4397	0.5174



(a) Indian Pines Embedding: UMAP (supervised)

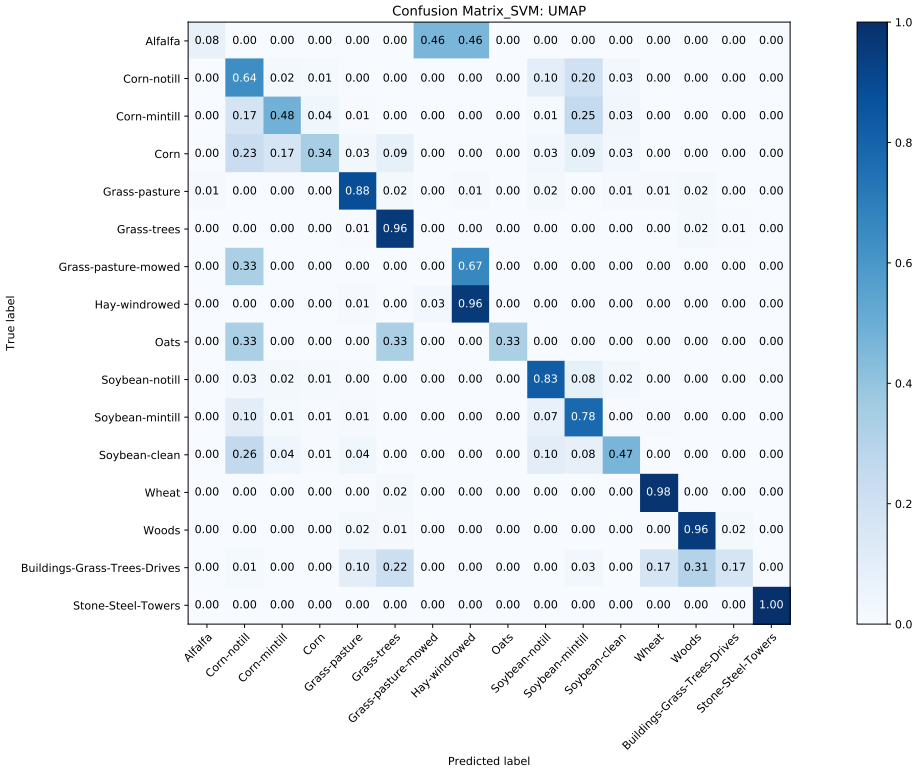


(b) Indian Pines Embedding: LEt-SNE (supervised)

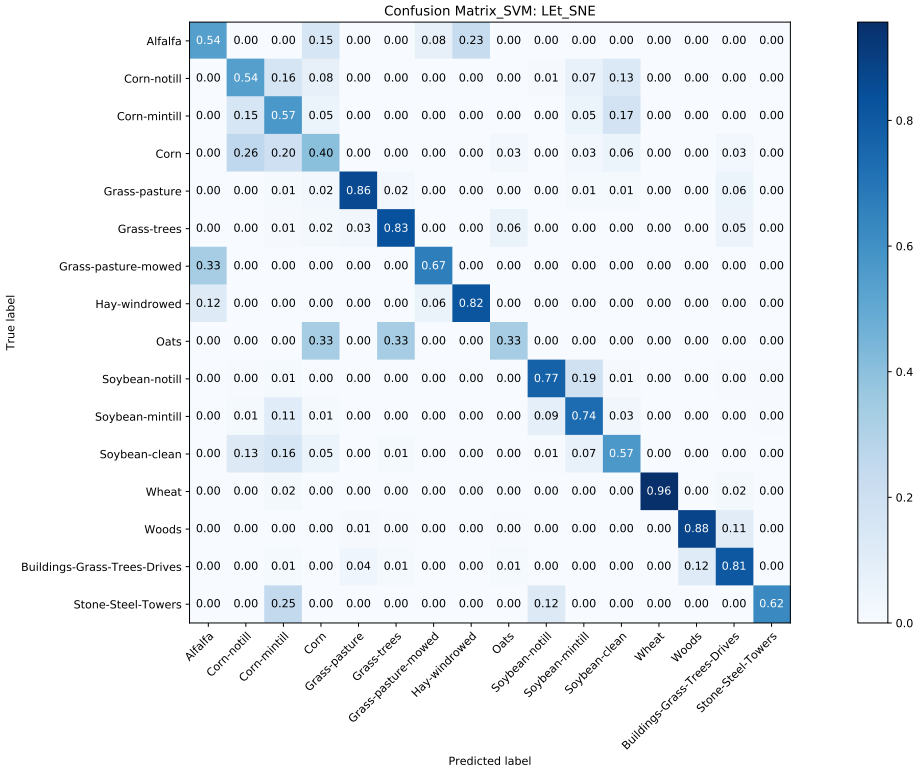
Figure 5.11: Comparison between the Embeddings of UMAP and LEt-SNE (supervised)

5.2.3 Indian Pines

The Indian Pines dataset provides a stiffer challenge than the other two, due to the fact that the classes are not mutually exclusive. There is considerable overlap between classes, with each class having a large amount of variation. As before, we compare the result of LEt-SNE with other algorithms, shown in Table 5.4. We note that in this case, the κ values are similar for LEt-SNE and UMAP. However, an interesting insight is available when we examine the Confusion matrix shown in Figure 5.12.



(a) Indian Pines Confusion Map: UMAP (supervised)

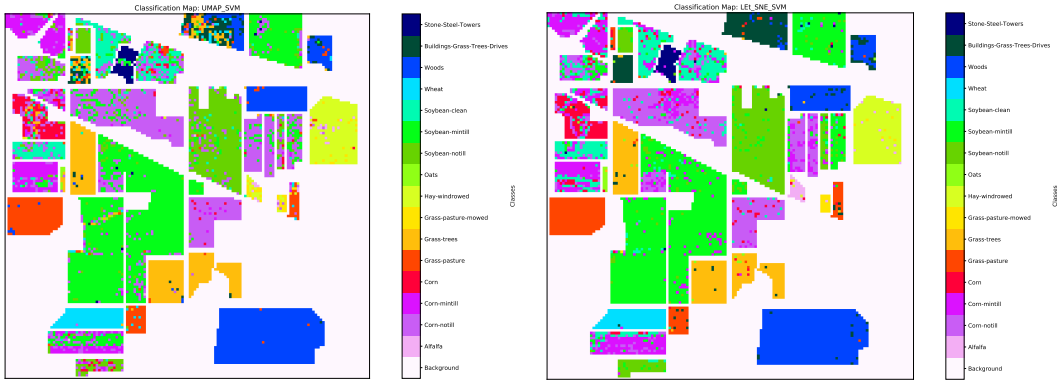


(b) Indian Pines Confusion Map: LEt-SNE (supervised)

Figure 5.12: Comparison between the Confusion Matrices of UMAP and LEt-SNE (supervised)

Observe the class *Grass-Pasture-Mowed* for UMAP; which shows 0% accuracy in detection. We also look at *Building-Grass-Trees-Drives*, again having a significantly lower accuracy in comparison to it's LEt-SNE counterpart. UMAP achievs it's accuracy rate by overcommiting to the class dominating in the cluster embedding, sidelining the rest. In comparison, LEt-SNE has a higher success rate in determining rare classes, and thus better suited for data embedding in a imbalanced class dataset.

The classification maps shown in Figure 5.13 completes our discussion on Manifold based Supervised Clustering. We see that UMAP is not able to accurately pick out classes such as *buildings-grass-trees-drives* which is better identified with LEt-SNE. However, UMAP has a higher success rate with classes having a larger representation in the dataset.



(a) Indian Pines Classification (SVM): UMAP

(b) Indian Pines Classification (SVM): LEt-SNE

Figure 5.13: Comparison between the Classification Maps of UMAP and LEt-SNE (supervised)

5.2.4 General Observations

We have seen that given labelled information, LEt-SNE can provide superior clusterings which we can use as a pre-processing step for classification problems. An advantage of LEt-SNE is the resistance to overfitting, since we do not supply classification gradients in the process of clustering our samples. The LEt-SNE objective consists of functions that are in opposition to each other, and aided with Batch Normalization provide embeddings with added robustness against overfitting. The figures in the preceding subsection show that LEt-SNE is much likelier to correctly identify rare classes than UMAP.

5.3 Manifold Based Unlabelled Clustering

As we recall, we used segmentation techniques for the task of unsupervised clustering in the absence of labels. For Indian Pines and Pavia University, we use a combination of *SLIC* and *RAG* to provide disjoing segmentation labels. With Salinas, we use the *Watershed* algorithm combined with morphological processing to provide disjoint segment labels.

Table 5.5: Accuracy comparison: Salinas (unsupervised)

Metric	**LEt-SNE** (seg)	Autoencoder	PCA	UMAP (unsup)
SVM (OA)	**0.9125**	0.8358	0.8296	0.8524
NeuralNet (OA)	**0.9128**	0.841	0.8401	0.8627
Kappa (κ)	**0.9026**	0.8178	0.8111	0.8361

5.3.1 Salinas

We compare the results of the unsupervised variant of LEt-SNE with Autoencoders, PCA and the unsupervised variant of UMAP in Table: 5.5. We have demonstrated that even in the absence of Labeled samples, our algorithm is able to incorporate spatial similarity and regions that form in Hyperpectral Imagery. The process of obtaining Segmentation labels for Salinas is shown in Figure 5.14. After converting the image to grayscale, we convolve it with Sobel operator to identify the topology. Markers are chosen manually by obtaining visual cues from the grayscale image. After obtaining our segmentation map from the watershed algorithm, we are left with the task of labelling disjoint regions, which is evaluated by checking for 1/2-connectivity between regions having the same label. The absence of any connectivity implies separate regions. Disjoint region labelling is then used to compute the adjacency matrix.

We display the confusion matrix and embeddings generated by LEt-SNE in Figure 5.15. An interesting observation is that the other unsupervised approaches make inaccurate predictions $1/3^{rd}$ of the time with classes *Vineyard untrained* and *Grapes Untrained*. Since the two regions are disjoint in the segmentation map, we are able to classify them accurately. To summarize the discussion on Salinas, we compare the classification maps obtained by PCA, UMAP (unsupervised), Autoencoders and LEt-SNE (unsupervised) in Figure 5.16. It is evident that LEt-SNE has had more success in separating the merged classes in comparison to the other algorithms.

Table 5.6: Accuracy comparison: Pavia University (unsupervised)

Metric	**LEt-SNE** (seg)	Autoencoder	PCA	UMAP (unsup)
SVM (OA)	**0.7957**	0.6976	0.6754	0.7113
NeuralNet (OA)	**0.8433**	0.7967	0.801	0.7863
Kappa SVM (κ)	**0.7398**	0.6242	0.6027	0.6385

5.3.2 Pavia University

We repeat the procedure that we followed for Salinas; accuracy metrics are presented first, followed by a description of the segmentation process. We analyze the confusion matrix to understand the quality of classification followed by comparing the classification maps of the algorithms. Our results are tabulated in Table 5.6, which show a gap of ~10% between the kappa score of LEt-SNE and the next highest score. The steps taken to segment the dataset are shown in Figure 5.17. We first use PCA to reduce the dimensionality of the dataset to three. This serves as an input to SLIC, which computes a 5D coordinate of each sample and uses k-Means to determine the superpixels. SLIC as discussed in the methodologies chapter, has a high tendency to oversegment the image. By using Region Adjancency Graph and Graph Cut, we reduce the number of segments in the image to approximately 350-400. The advantage that *SLIC + RAG* provides over Watershed algorithm is the minimal human intervention is required for the purpose of segmentation. It also benefits from using 3-channels as opposed to one with Watershed. Figure 5.18 shows the embeddings learnt by LEt-SNE. We compare the classification maps between LEt-SNE, Autoencoder, PCA and UMAP in Figure 5.19. We notice that LEt-SNE has the least overlap between the classes *Bare Soil* and *Meadows*, which can be explained by the embeddings learnt by LEt-SNE since it yields greater class separation than the other approaches.

5.3.3 Indian Pines

Indian Pines presents a unique challenge since many of the classes are derived from the same parent class, such as *corn, grass and soybean*. Within the same class too there can be a large amount of variation. Our attempt at resolving the classes in an unsupervised environment are shown in 5.7. We notice that our attempt has delivered mixed results. Although the resultant Kappa values are not high, we still manage to outperform the other dimensionality reduction approaches by a significant margin for embeddings in $\mathbb{R}^2$. We do expect better metric values by increasing the dimensionality of the embeddings. Similar to

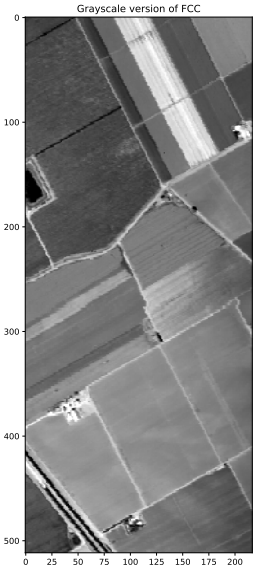
Table 5.7: Accuracy comparison: Indian Pines (unsupervised)

Metric	**LEt-SNE** (seg)	Autoencoder	PCA	UMAP (unsup)
SVM (OA)	**0.6488**	0.5459	0.4927	0.5685
NeuralNet (OA)	**0.6995**	0.6188	0.5912	0.6074
Kappa SVM (κ)	**0.604**	0.4916	0.4397	0.5173

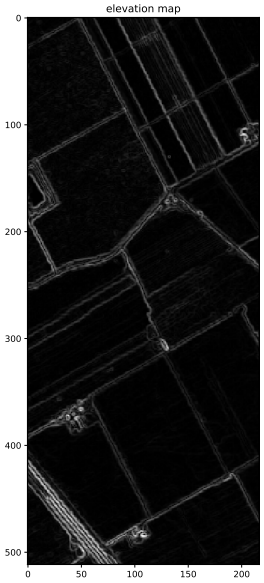
Pavia University, we compute region based segmentations using *SLIC*. The intermediate and final results on segmenting the dataset is shown in Figure 5.20. Using the disjoint region labels to construct the adjacency matrix, the resulting embedding and confusion map is shown in Figure 5.21. We see that there is a small amount of ambiguity between sub-classes of *corn* and *soybean* and across classes containing *notill* and *mintill*. This observation is reflected in the embeddings too, with merging of clusters corresponding to the ambiguous classes. We compare the classification maps in Figure 5.22. We observe that out of all the approaches, the classification map of LEt-SNE has accurately identified all the regions in the dataset; albeit with misclassifications. Other approaches have misclassified entire regions, such as *Buildings, Grass, Trees, Drives*.

5.4 Summary

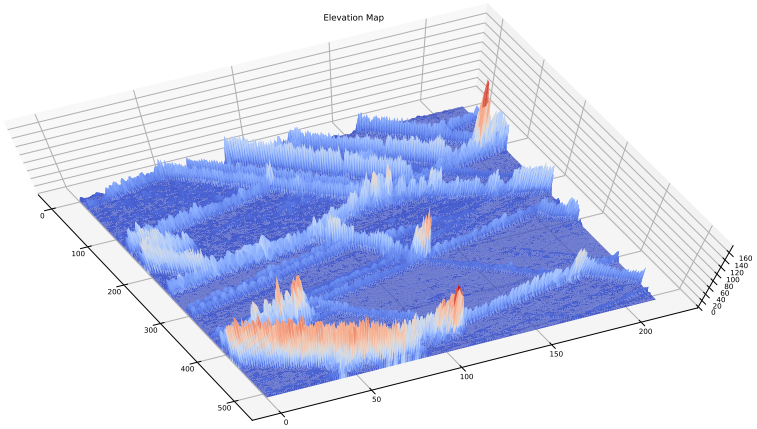
In this chapter, we have compared LEt-SNE with popular pre-existing algorithms such as UMAP, Autoencoders, PCA and t-SNE in both supervised and unsupervised settings. We have seen that LEt-SNE consistently outperforms the other algorithms. Our results use both Overall Accuracy (OA) and Kappa, a metric that handles class imbalances for assessing the quality of classification. While visualizing the manifold, we are able to capture intricate structures in the data not visible in PCA and autoencoders. The data samples are not overclustered like with UMAP. Our attempts at manifold based supervised clustering has shown superior results in comparison to UMAP, which is the current state of the art in Dimensionality Reduction along with t-SNE. We also offer a proof of concept solution to an unsupervised manifold based clustering algorithm which uses region segmentation to improve clustering quality in absence of labels. In this situation too LEt-SNE provides reasonable improvements over existing implementations. We would in particular like to highlight the results with Indian Pines, a difficult dataset to classify owing to the overlap between multiple classes. Unlike UMAP, autoncoders and PCA, LEt-SNE is able to correctly identify distinct regions in the image. The source code along with the experimentation data is available at: `https://github.com/meghshukla/LEt-SNE/`



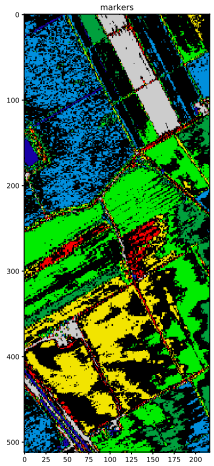
(a) Salinas FCC converted to Grayscale



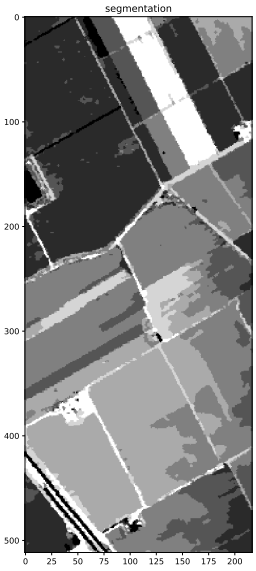
(b) Edge Detection: Sobel Operator



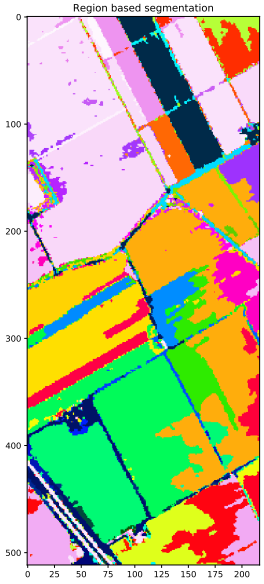
(c) Topological View of the Elevation Map



(d) Markers for the Watershed Algorithm

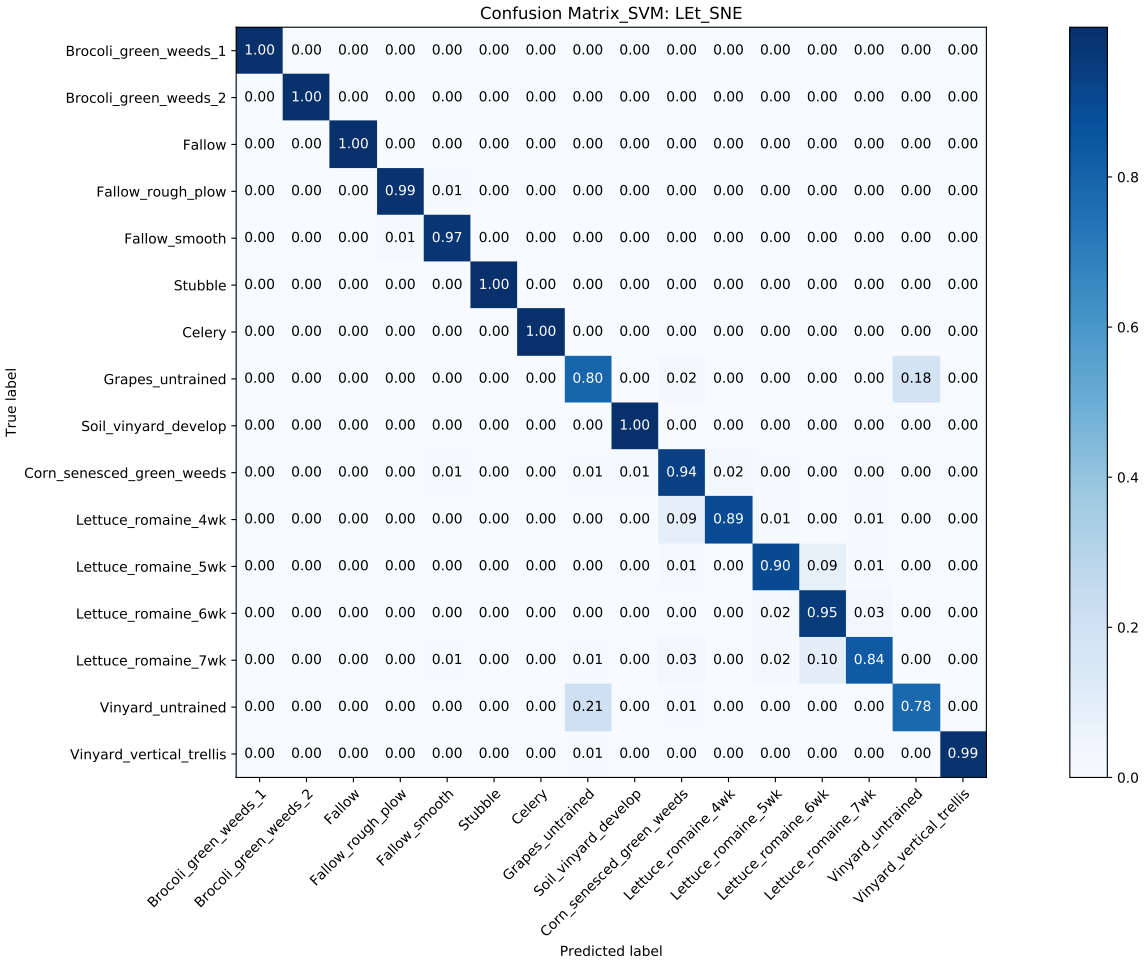


(e) Outcome of the Watershed Algorithm

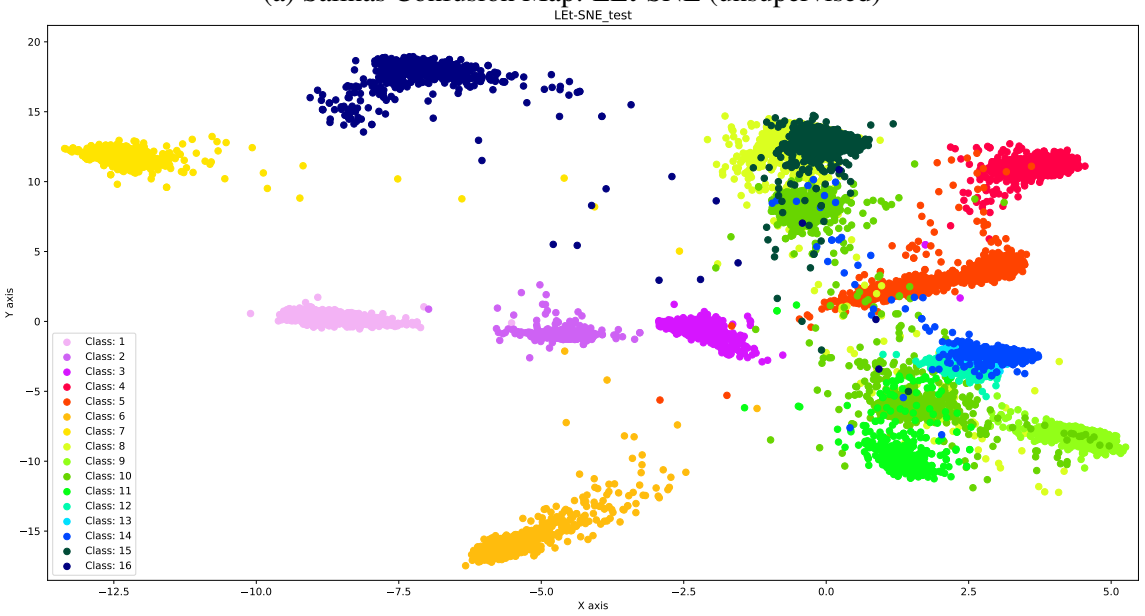


(f) Disjoint Regions identification

Figure 5.14: Obtaining segmentation labels for Salinas using the Watershed Algorithm

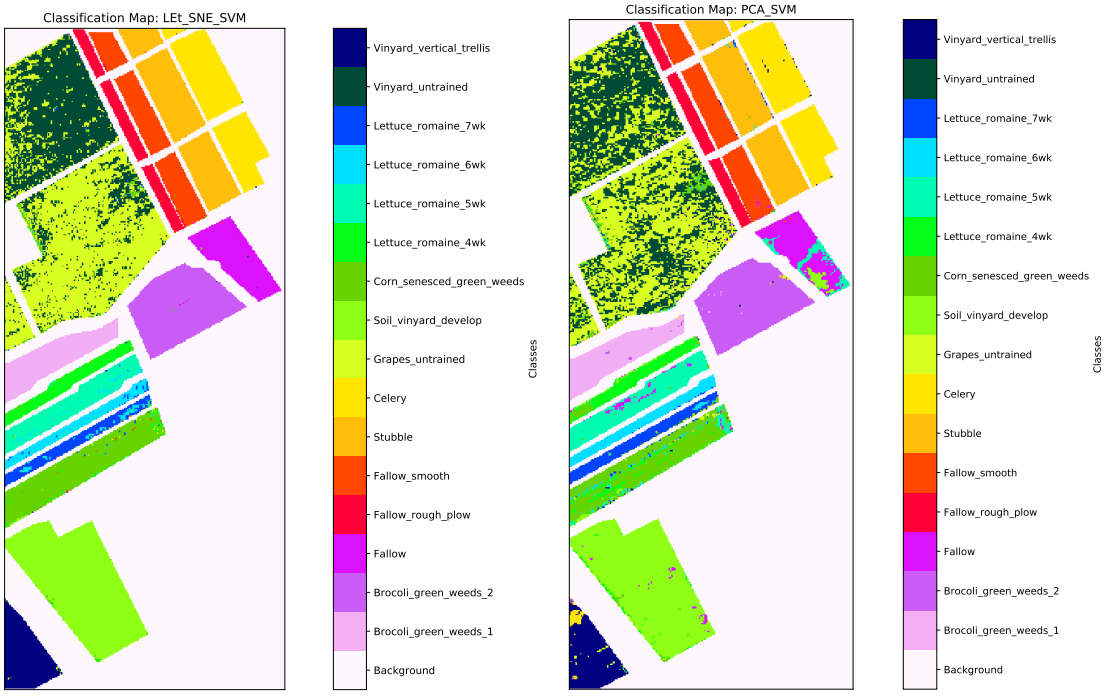


(a) Salinas Confusion Map: LEt-SNE (unsupervised)

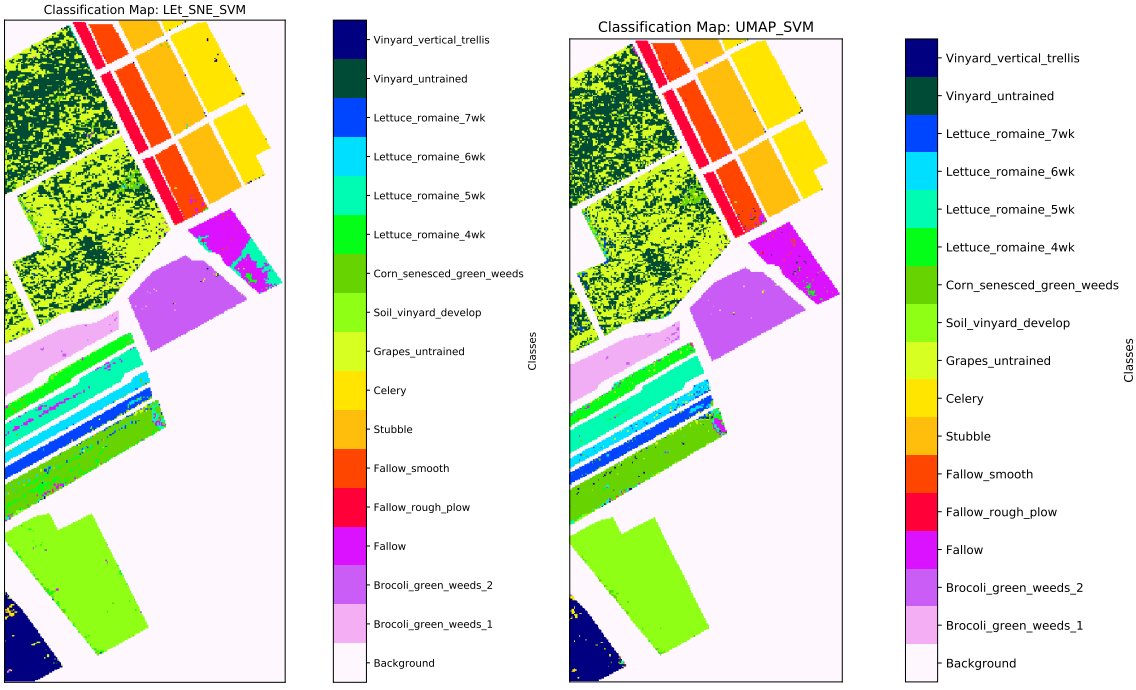


(b) Salinas Embedding: LEt-SNE (unsupervised)

Figure 5.15: Salinas: Confusion Matrix and Embeddings for LEt-SNE (unsupervised)

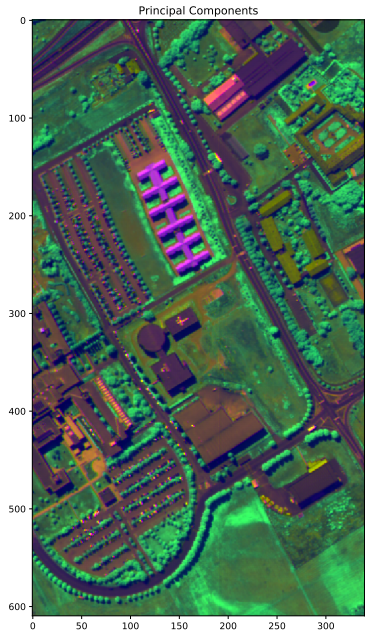


(a) Salinas Classification Map: LEt-SNE (b) Salinas Classification Map: PCA

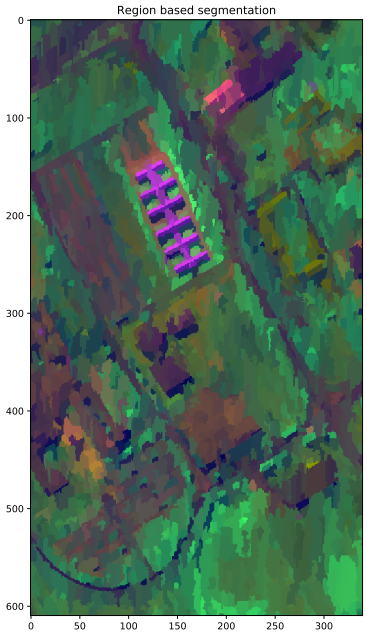


(c) Salinas Classification Map: Autoencoder (d) Salinas Classification Map: UMAP

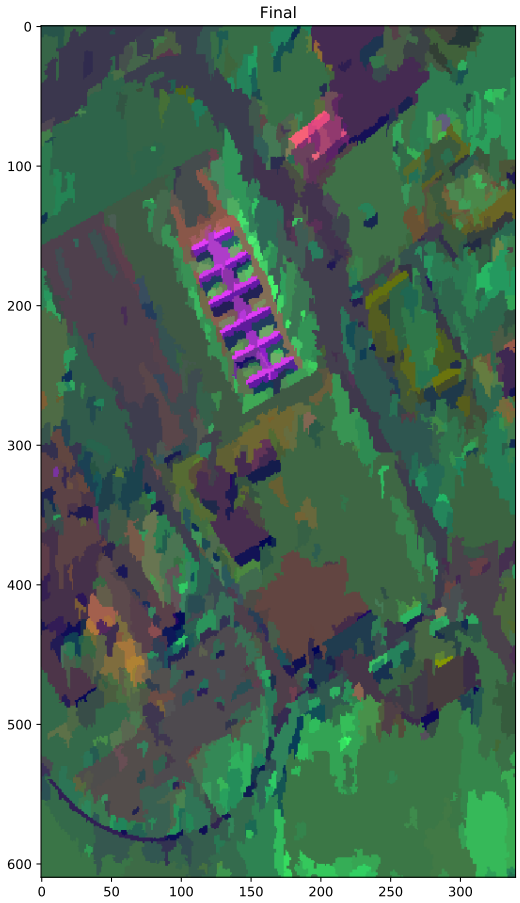
Figure 5.16: Salinas: Classification Maps for unsupervised approaches



(a) PCA transform of the Pavia University



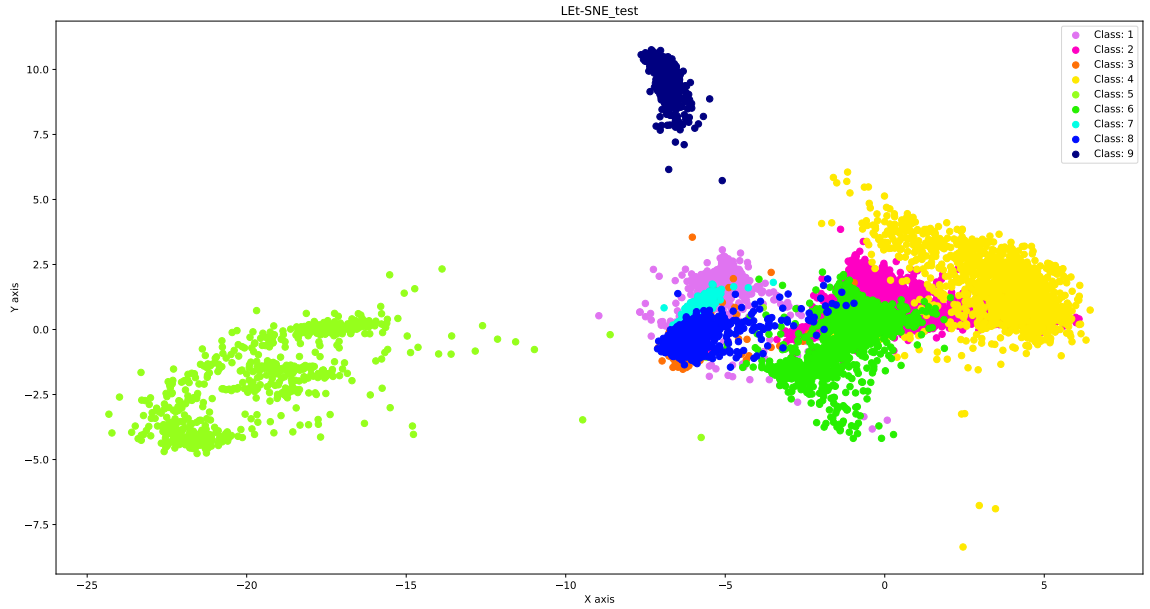
(b) Computing SLIC segmentation



(c) Reducing the oversegmentation: SLIC + RAG

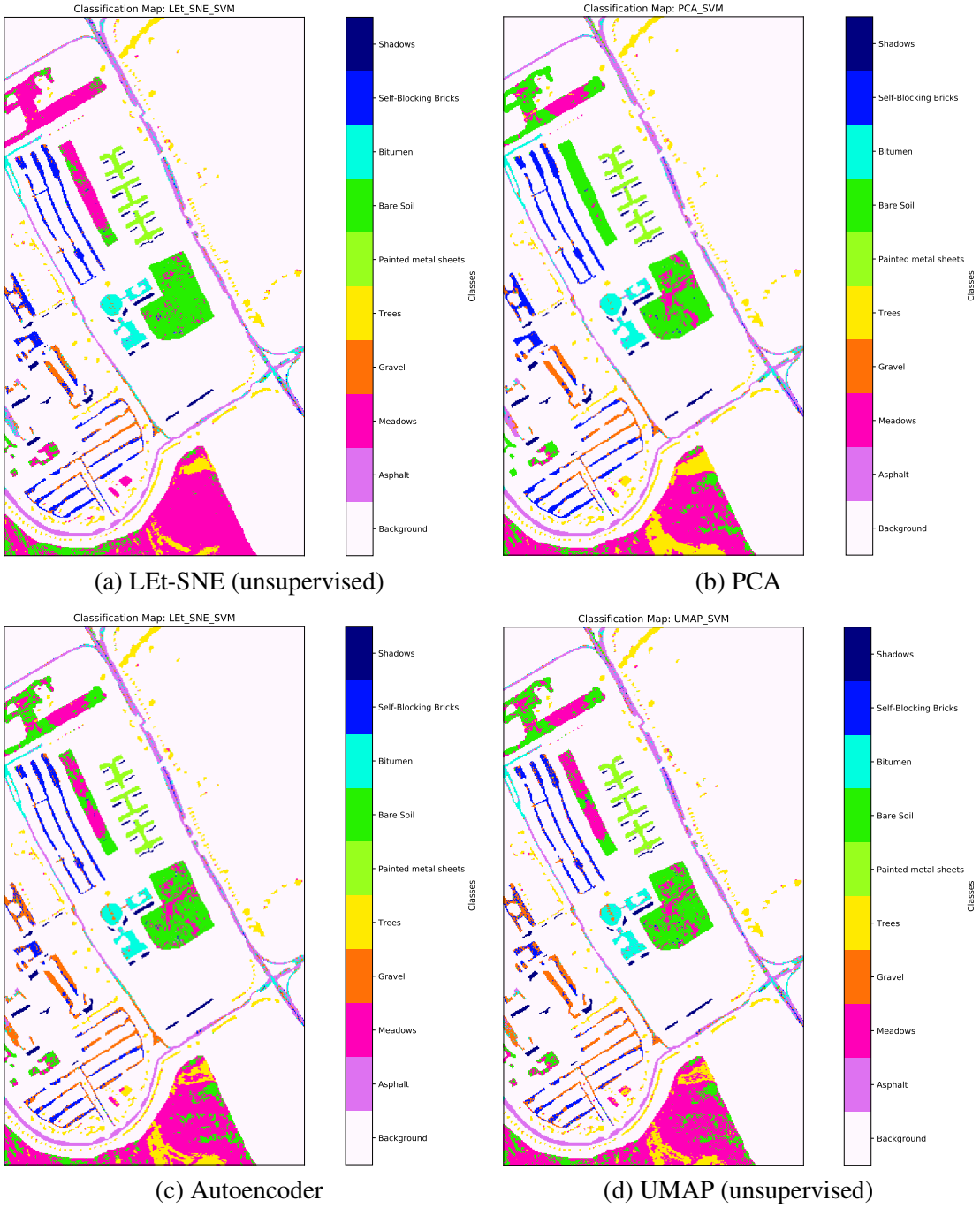
Figure 5.17: Pavia University: Unsupervised Region Segmentation using SLIC + RAG

(a) Pavia University Confusion Map: LEt-SNE (unsupervised)



(b) Pavia University Embedding: LEt-SNE (unsupervised)

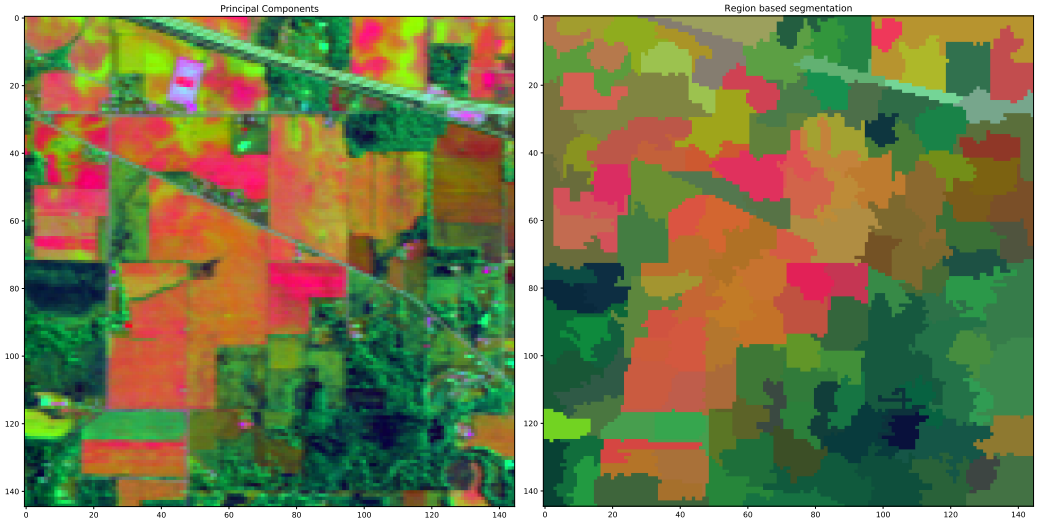
Figure 5.18: Pavia University: Confusion Matrix and Embeddings for LEt-SNE (unsupervised)



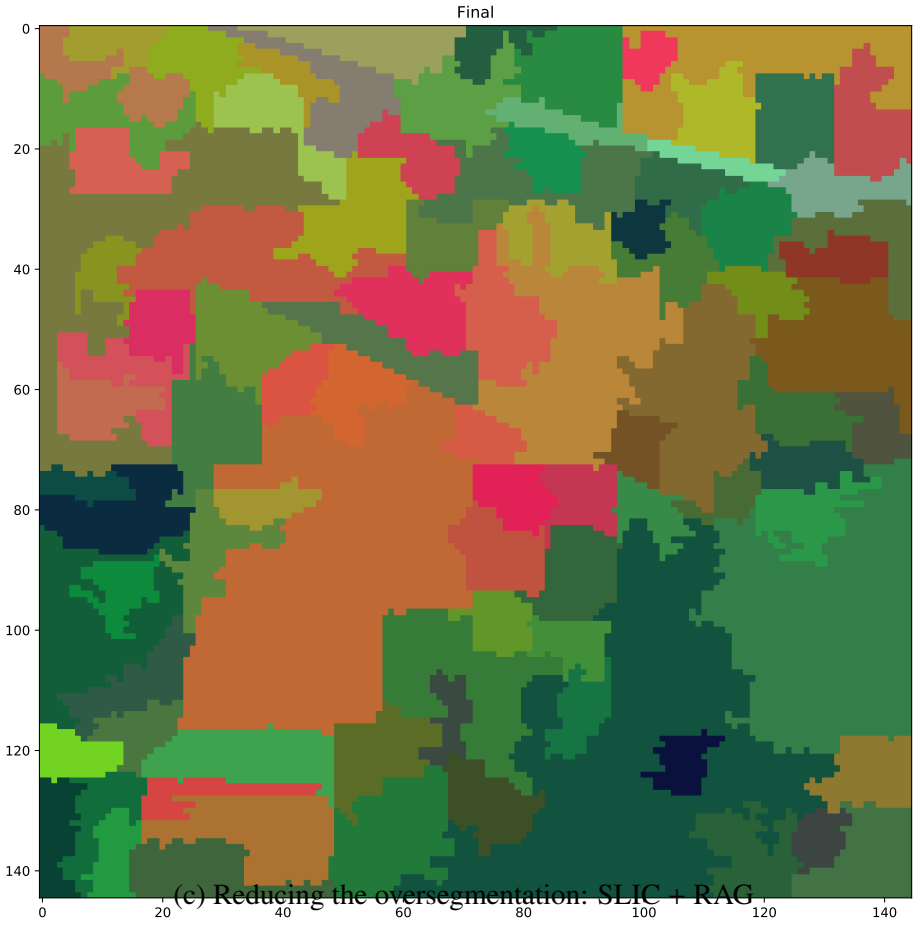
(a) LEt-SNE (unsupervised) (b) PCA

(c) Autoencoder (d) UMAP (unsupervised)

Figure 5.19: Pavia University: Classification Maps for unsupervised approaches

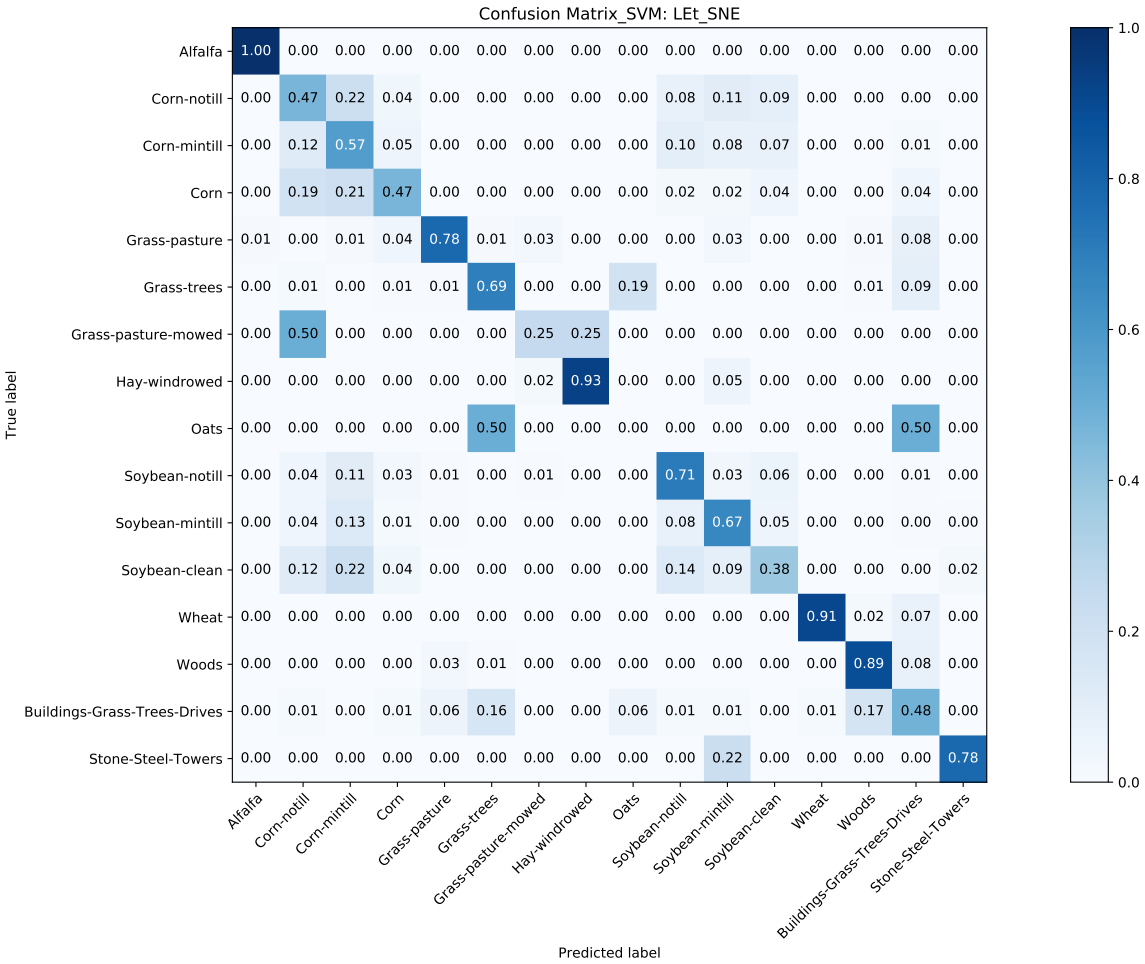


(a) PCA transformation (b) Computing SLIC segmentation

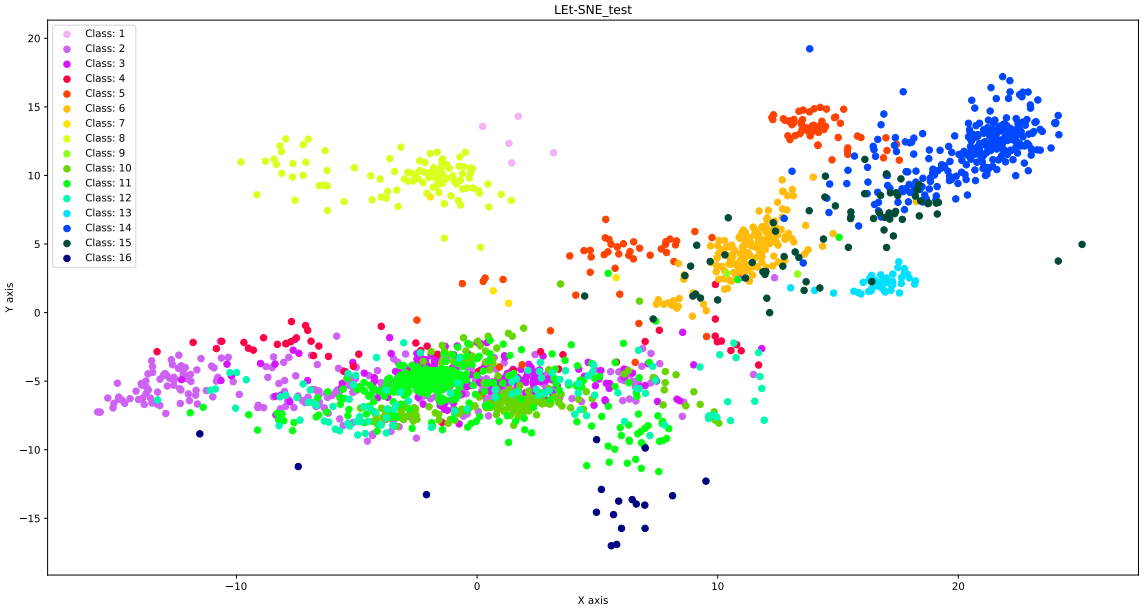


(c) Reducing the oversegmentation: SLIC + RAG

Figure 5.20: Indian Pines: Unsupervised Region Segmentation using SLIC + RAG

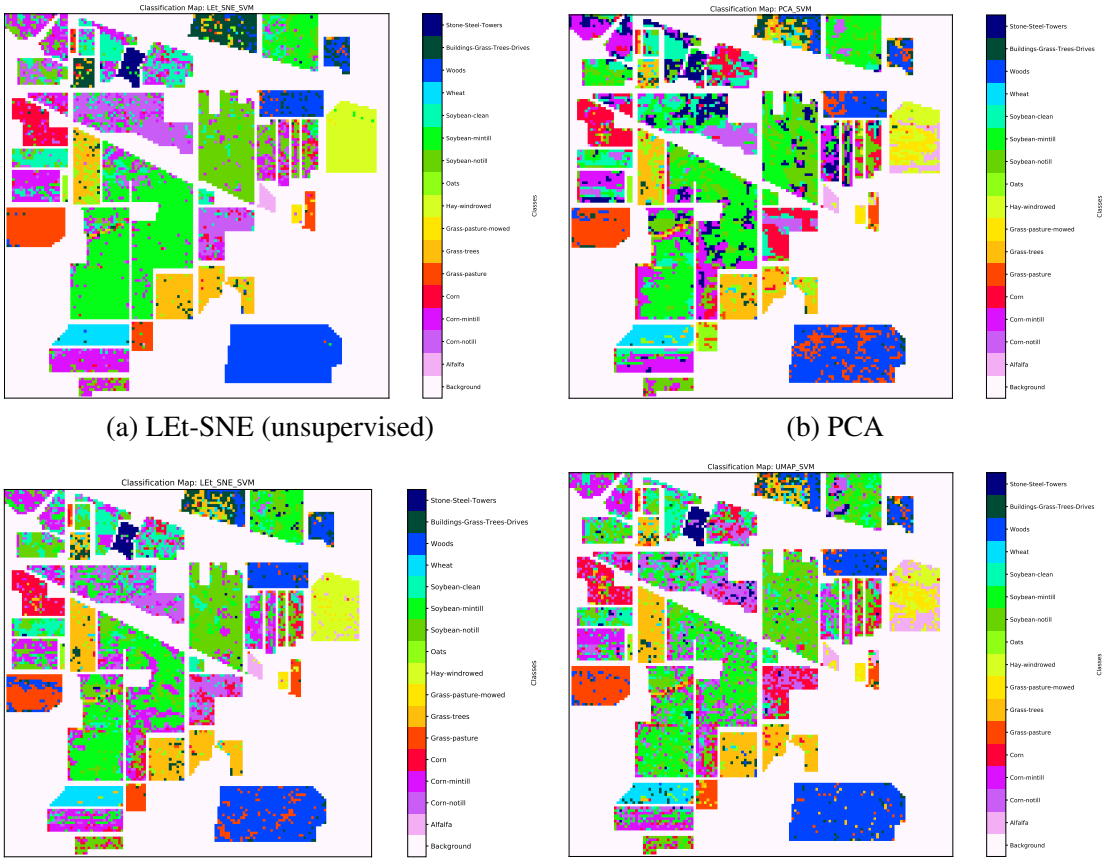


(a) Indian Pines Confusion Map: LEt-SNE (unsupervised)



(b) Indian Pines Embedding: LEt-SNE (unsupervised)

Figure 5.21: Indian Pines: Confusion Matrix and Embeddings for LEt-SNE (unsupervised)



(a) LEt-SNE (unsupervised) (b) PCA

(c) Autoencoder (d) UMAP (unsupervised)

Figure 5.22: Indian Pines: Classification Maps for unsupervised approaches

Chapter 6

Conclusion and Future Scope

Conclusion

In this dissertation, we have briefly described the problem of dimensionality reduction with the primary aim of data visualization and manifold based clustering. We have studied various pre-existing implementations such as the Principal Component Analysis, Linear Discriminant Analysis, Laplacian Embedding, Locally Linear Embedding, t-SNE and UMAP. There were some limitations that were observed, such as the absence of parameterization, overcrowding, or the computational time required on real world Hyperspectral datasets. Some techniques had an inherent linearity in them, which provided approximate results on real world datasets. In Section 3.1, we listed down qualities that our data visualization algorithm should possess. We also framed the objectives to be achieved in this thesis: data visualization and manifold based clustering approaches. We have used the strength of t-SNE and LE to merge them into a single algorithm which is parameterized by a neural network. Comparisons were carried out on mutiple datasets with various other algorithms, which suggested that LEt-SNE provided better visualization and clustering for Hyperpectral datasets. LEt-SNE is quick in generating embeddings due to the implementation of mini-batches. Since it is parameterized, the algorithm can provide embeddings on the fly, as is the case with PCA. The algorithm is resistant to overfitting due to competing loss objectives, which prevent a collapse into local minima, and also from the fact that classification gradients were not used at any stage in the process of embedding generation. We have provided a trick to combat the Curse of Dimensionality phenomenon by implementing *Compression Factor*. Certain limitations exist with our approach, such as the need to have a fresh run of the algorithm if we wish to increase the dimensionality of our dataset. We hope to address this issue in our future work. LEt-SNE could be applied for general dimensionality reduction tasks such as classification. One interesting application is learning the probability distribution of the input with the help of

GANs as explored in [54]. The authors have used PCA as a preprocessing tool for Hyperspectral Imagery before applying them to GANs. PCA as we have seen fails to learn the non-linearities associated with the data. The results in this dissertation have shown that LEt-SNE learns better representations of the underlying manifold than PCA, leaving it more suitable for such applications.

Future Scope

An area of exploration could be modifying LEt-SNE into a general purpose dimensionality reduction algorithm. In this dissertation, we have restricted the dimensionality of the embeddings to two which facilitates visual depiction of the embeddings for the human observer. As suggested in t-SNE, our algorithm could be extended to higher dimensions by using cauchy distributions with more degrees of freedom. This would require hyperparameter tuning in the current setup of LEt-SNE.

We have proposed supervised and unsupervised variants of LEt-SNE in this dissertation. A recent development in the Machine Learning community is semi-supervised learning, where a limited quantity of our samples have labels. Hyperspectral Imagery contains a large amount of unlabelled data; but wit the progress of time, a small section of them could be labelled. By using semi-supervised learning, our algorithm can take into account the limited labelling available for embedding generation. UMAP has an implementation of semi-supervised labelling.

We could also explore various segmentation techniques which could be used for the task of unsupervised manifold based clustering in LEt-SNE. Currently, we use a grayscale version of the FCC or the first three principal components of our dataset as the input in our segmentation tasks. This may not be the best approach always, especially since FCC is not guaranteed to provide visually separable regions. Other approaches we could experiment with include Feature Selection techniques such as Random Forest and Genetic Algorithms to select the best bands representative of our dataset. Deep Learning based segmentation techniques could also be used to generate image segmentations.

Alternative distance metrics could be incorporated into LEt-SNE instead of euclidean distances. Hyperspectral Imagery could have variable illumination and BRDF effects on the scene, which may not leave euclidean distance to be the best metric for use. Alternatives include using Cosine distances between samples, which negates the effects of illumination on our dataset. We could also use Mahalanobis distance in the computation of t-SNE due to the correlated nature of our hyperspectral bands.

Appendix A

Appendix

In the appendix, we discuss the results of generalizing LEt-SNE to dimensions greater than three, and compare it with other supervised dimensionality reduction algorithms. This discussion is followed by a brief mention of our implementation, including the packages used for the experimentation contained in this dissertation.

A.1 General Purpose Dimensionality Reduction

In this section we experiment with increasing the dimensionality of the embeddings beyond three. We select the Indian Pines dataset due to the inability to represent the embeddings in two dimensions, as portrayed by the accuracy metrics. We assume access to labels, and keep the dimensionality of our embeddings as ten. We also incorporate the results of Linear Discriminant Analysis and compare it with LEt-SNE in Table A.1. We observe that UMAP is not able to scale with the number of dimensions; requiring extensive hyperparameter tuning to achieve better results. Similar performance is obtained between LEt-SNE and UMAP, with highest accuracy obtained by LEt-SNE edging out it's LDA counterpart by ~2%. LEt-SNE shows promise as a general purpose dimensionality reduction with significantly lesser computational costs than t-SNE.

Table A.1: Accuracy comparison: Indian Pines (Dimensionality = 10)

Metric	**LEt-SNE** (sup)	UMAP (sup)	LDA
SVM (OA)	**0.8311**	0.7242	0.8263
NeuralNet (OA)	**0.8538**	0.7485	0.8341
Kappa SVM (κ)	**0.8081**	0.6864	0.8021
Kappa NN (κ)	**0.8333**	0.7118	0.8105

A.2 Programming

The Program has been written in `Python-3` with the following libraries:

1. `TensorFlow v1.13`
2. `Scikit-learn`
3. `Scikit-image`
4. `Matplotlib`
5. `SciPy`
6. `NumPy`

`TensorFlow` [1] is a powerful matrix manipulation library which harnesses the power of Graphics Processing Unit for mathematical computations. It facilitates GPU acceleration depending on the available hardware; in the absence of GPUs it defaults to CPU execution. Keeping in mind the transition phase of `TensorFlow` from 1.x to 2.x, we have avoided the use of sessions, instead settling for custom estimators which are supported in both the versions of `TensorFlow`. The core LEt-SNE algortihm is written in `TensorFlow`. For other algorithms such as LDA, SVM, PCA we use their corresponding implementations available in `Scikit-learn` [38]. Visualization was facilitated by `Matplotlib` [24], with image segmentation algorithms (Watershed, SLIC) used from `Scikit-image` [50].

References

[1] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X., 2015, "TensorFlow: Large-scale machine learning on heterogeneous systems," software available from tensorflow.org.

[2] Achanta, R., Shaji, A., Smith, K., Lucchi, A., Fua, P., and Süsstrunk, S., 2010, "Slic superpixels," , 15.

[3] Belkin, M., and Niyogi, P., 2003, "Laplacian eigenmaps for dimensionality reduction and data representation," *Neural Computation* **15**, 1373–1396.

[4] Bellman, R., 1957, *Dynamic Programming*, 1st ed. (Princeton University Press, Princeton, NJ, USA).

[5] Beucher, S., 1994, "Watershed, hierarchical segmentation and waterfall algorithm," in *Mathematical Morphology and Its Applications to Image Processing*, edited by Serra, J. and Soille, P. (Springer Netherlands, Dordrecht). ISBN 978-94-011-1040-2, pp. 69–76.

[6] Buddhiraju, K. M., and Rizvi, I. A., 2010 July, "Comparison of cbf, ann and svm classifiers for object based classification of high resolution satellite images," in *2010 IEEE International Geoscience and Remote Sensing Symposium*, pp. 40–43.

[7] CHEN, J., and MA, Z., 2011, "Locally linear embedding: A review," *International Journal of Pattern Recognition and Artificial Intelligence* **25**, 985–1008.

[8] Christophe, E., Mailhes, C., and Duhamel, P., 2008 Dec, "Hyperspectral image compression: Adapting spiht and ezw to anisotropic 3-d wavelet coding," *IEEE Transactions on Image Processing* **17**, 2334–2346.

[9] (https://stats.stackexchange.com/users/29694/louis cialdella), L. C., 2017, "Difference between long tail and short tail distribution?." Cross Validated, URL:https://stats.stackexchange.com/q/285020 (version: 2018-11-01).

[10] Cortes, C., and Vapnik, V., 1995 Sep., "Support-vector networks," *Mach. Learn.* **20**, 273–297.

[11] E. Rumelhart, D., E. Hinton, G., and J. Williams, R., 1986 10, "Learning representations by back propagating errors," *Nature* **323**, 533–536.

[12] Edelsbrunner, H., and Harer, J., 2010 01, *Computational Topology: An Introduction*, ISBN 978-0-8218-4925-5

[13] FISHER, R. A., 1936, "The use of multiple measurements in taxonomic problems," *Annals of Eugenics* **7**, 179–188.

[14] Girshick, R., Donahue, J., Darrell, T., and Malik, J., 2014 June, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 580–587.

[15] Goodfellow, I., Bengio, Y., and Courville, A., 2016, *Deep Learning* (MIT Press).

[16] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y., 2014, "Generative adversarial nets," in *Advances in Neural Information Processing Systems 27*, edited by Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q. (Curran Associates, Inc.). pp. 2672–2680.

[17] Halkjær, S., and Winther, O., 1996, "The effect of correlated input data on the dynamics of learning," in *NIPS*

[18] Hamida, A. B., Benoit, A., Lambert, P., Klein, L., Amar, C. B., Audebert, N., and Lefèvre, S., 2017, "Deep learning for semantic segmentation of remote sensing images with rich spectral content," *2017 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, 2569–2572.

[19] Hamner, B., 2016, Kaggle Blog

[20] He, K., Zhang, X., Ren, S., and Sun, J., 2016 June, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778.

[21] Hinton, G. E., and Salakhutdinov, R. R., 2006, "Reducing the dimensionality of data with neural networks," *Science* **313**, 504–507.

[22] Hinton, G. E., and Roweis, S. T., 2003, "Stochastic neighbor embedding," in *Advances in Neural Information Processing Systems 15*, edited by Becker, S., Thrun, S., and Obermayer, K. (MIT Press). pp. 857–864.

[23] amoeba (https://stats.stackexchange.com/users/28666/amoeba), 2019, "How can t-sne or umap embed new (test) data, given that they are nonparametric?." Cross Validated, URL:https://stats.stackexchange.com/q/398806 (version: 2019-03-21).

[24] Hunter, J. D., 2007, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering* **9**, 90–95.

[25] Ioffe, S., and Szegedy, C., 2015, "Batch normalization: Accelerating deep network training by reducing internal covariate shift,"

[26] Kingma, D. P., and Welling, M., 2014, "Auto-encoding variational bayes," *CoRR* **abs/1312.6114**

[27] Krizhevsky, A., Sutskever, I., and Hinton, G. E., 2012, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, edited by Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q. (Curran Associates, Inc.). pp. 1097–1105.

[28] Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P., 1998 Nov, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE* **86**, 2278–2324.

[29] Lunga, D., and Ersoy, O., 2013 Feb, "Spherical stochastic neighbor embedding of hyperspectral data," *IEEE Transactions on Geoscience and Remote Sensing* **51**, 857–871.

[30] Luttrell, S. P., 1989 Oct, "Hierarchical self-organising networks," in *1989 First IEE International Conference on Artificial Neural Networks, (Conf. Publ. No. 313)*, pp. 2–6.

[31] van der Maaten, L., 2009, "Learning a parametric embedding by preserving local structure," in *AISTATS*

[32] van der Maaten, L., and Hinton, G., 2008, "Visualizing Data using t-SNE," *Journal of Machine Learning Research* **9**, 2579–2605.

[33] Makantasis, K., Karantzalos, K., Doulamis, A., and Doulamis, N., 2015 July, "Deep supervised learning for hyperspectral data classification through convolutional neural networks," in *2015 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, pp. 4959–4962.

[34] McInnes, L., and Healy, J., 2018, "Umap: Uniform manifold approximation and projection for dimension reduction," *arXiv*

[35] Nair, V., and Hinton, G. E., 2010, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10 (Omnipress, USA). pp. 807–814.

[36] Pascal, V. e. a., 2010, "Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion," *Journal of Machine Learning Research* **11**, 3371–3408.

[37] Pearson, K., 1901, "On lines and planes of closest fit to system of points in space," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* **6**, 559–572.

[38] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E., 2011, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research* **12**, 2825–2830.

[39] Petersson, H., Gustafsson, D., and Bergstrom, D., 2016 Dec, "Hyperspectral image analysis using deep learning — a review," in *2016 Sixth International Conference on Image Processing Theory, Tools and Applications (IPTA)*, pp. 1–6.

[40] PV, A., Buddhiraju, K. M., and Porwal, A., 2019, "Capsulenet-based spatial–spectral classifier for hyperspectral images," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 1–17.

[41] Rifai, S., Vincent, P., Muller, X., Glorot, X., and Bengio, Y., 2011, "Contractive auto-encoders: Explicit invariance during feature extraction," in *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11 (Omnipress, USA). pp. 833–840.

[42] Roweis, S. T., and Saul, L. K., 2000, "Nonlinear dimensionality reduction by locally linear embedding," *Science* **290**, 2323–2326.

[43] Sammon, J. W., 1969 May, "A nonlinear mapping for data structure analysis," *IEEE Transactions on Computers* **C-18**, 401–409.

[44] Schölkopf, B., Smola, A., and Müller, K.-R., 1997, "Kernel principal component analysis," in *Artificial Neural Networks — ICANN'97*, edited by Gerstner, W., Germond, A., Hasler, M., and Nicoud, J.-D. (Springer Berlin Heidelberg, Berlin, Heidelberg). pp. 583–588.

[45] Shlens, J., 2014, "A tutorial on principal component analysis," *CoRR* **abs/1404.1100**

[46] Szegedy, C., Wei Liu, Yangqing Jia, Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A., 2015 June, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–9.

[47] Tenenbaum, J. B., Silva, V. d., and Langford, J. C., 2000, "A global geometric framework for nonlinear dimensionality reduction," *Science* **290**, 2319–2323.

[48] Thenkabail, P. S., Mariotto, I., Gumma, M. K., Middleton, E. M., Landis, D. R., and Huemmrich, K. F., 2013 April, "Selection of hyperspectral narrowbands (hnbs) and composition of hyperspectral twoband vegetation indices (hvis) for biophysical characterization and discrimination of crop types using field reflectance and hyperion/eo-1 data," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* **6**, 427–439.

[49] Tobler, W. R., 1970, "A computer movie simulating urban growth in the detroit region," *Economic Geography* **46**, 234–240.

[50] van der Walt, S., Schönberger, J. L., Nunez-Iglesias, J., Boulogne, F., Warner, J. D., Yager, N., Gouillart, E., Yu, T., and the scikit-image contributors, 2014 6, "scikit-image: image processing in Python," *PeerJ* **2**, e453.

[51] Windrim, L., Melkumyan, A., Murphy, R. J., Chlingaryan, A., and Ramakrishnan, R., 2018 May, "Pretraining for hyperspectral convolutional neural network classification," *IEEE Transactions on Geoscience and Remote Sensing* **56**, 2798–2810.

[52] Yang, J., Zhao, Y., and Chan, J. C., 2017 Aug, "Learning and transferring deep joint spectral–spatial features for hyperspectral classification," *IEEE Transactions on Geoscience and Remote Sensing* **55**, 4729–4742.

[53] Zhang, L., Yang, T., Yi, J., Jin, R., and Zhou, Z.-H., 2016, "Stochastic optimization for kernel pca,"

[54] Zhu, L., Chen, Y., Ghamisi, P., and Benediktsson, J. A., 2018 Sep., "Generative adversarial networks for hyperspectral image classification," *IEEE Transactions on Geoscience and Remote Sensing* **56**, 5046–5063.

Acknowledgements

This dissertation is the culmination of my Master's journey in IIT Bombay, a journey that has pushed my intellectual abilities to the extreme. Completing this dissertation had its own share of peaks and troughs, much like the elevation maps in the Watershed algorithm! It is with immense gratitude that I thank those who helped me in this journey, particularly my guide, Prof. Krishna Mohan Buddhiraju. Prof BK Mohan has been my mentor throughout my MTech program, and has allowed me freedom to explore many ideas that I wished to pursue. His suggestions and insights have been valuable in preparing me as a researcher. I would also like to thank my co-guide, Prof Biplab Banerjee, whom I frequently used to approach with queries regarding various algorithms. Another mentor who has shaped my Master's journey is Prof. Alok Porwal, who instilled within me the curiosity to ask questions. Never was there a moment where his students were not encouraged to be inquisitive, inside the classroom or outside!

I express my gratitute towards the engineers at `HARMAN X`, the research team at `HARMAN` India, for providing me with technical support to gain practical skills in TensorFlow during my internship. The duration of time that I spent at `HARMAN` helped me in realizing the importance of doing research that has practical implications. A special mention goes to Aashish Kumar, Srinivas Sai Kruthiveni, Pratyush Sahay and Sambuddha Saha, who mentored me during the internship period.

Finally, I take this opportunity to thank Centre of Studies in Resources Engineering as a whole. It is very rare that in a technical institution the size of IIT that the professors take care for the betterment of each student in the department. CSRE has provided me a platform where I could have as peers some of the brightest minds in India! My journey with CSRE has been an enrichening experience as a whole.

Megh Shukla
IIT Bombay
4 June 2019