A Branching Convolutional Encoder-based Spatio-spectral Dimensionality Reduction Model for Hyperspectral Image Classification and Change Detection

Menilk Sahlu Bayeh

A Thesis Submitted to The Department of Computer Science and Engineering
School of Electrical Engineering and Computing

Presented in Partial Fulfillment of the Requirement for the Master of Science Degree in Computer Science and Engineering

Office of Graduate Studies
Adama Science and Technology University

January, 2021
Adama, Ethiopia

A Branching Convolutional Encoder-based Spatio-spectral Dimensionality Reduction Model for Hyperspectral Image Classification and Change Detection

Menilk Sahlu Bayeh

Advisor: Yunkoo Chung (Professor)

A Thesis Submitted to The Department of Computer Science and Engineering
School of Electrical Engineering and Computing

Presented in Partial Fulfillment of the Requirement for the Master of Science Degree in Computer Science and Engineering

Office of Graduate Studies
Adama Science and Technology University

January, 2021
Adama, Ethiopia

APPROVAL OF BOARD OF EXAMINERS

We, the undersigned, members of the Board of Examiners of the final open defense by Menilk Sahlu Bayeh have read and evaluated his thesis entitled “A Branching Convolutional Encoder-based Spatio-spectral Dimensionality Reduction Model for Hyperspectral Image Classification and Change Detection” and examined the candidate. This is, therefore, to certify that the thesis has been accepted in partial fulfillment of the requirement of the Degree of Masters Computer Science and Engineering.

Name	Signature	Date
Name of Student		
Advisor		
External Examiner		
Internal Examiner		
Chair Person		
Head of Department		
School Dean		
Postgraduate Dean		

DECLARATION

I hereby declare that this MSc thesis is my original work and has not been presented for a degree in any other university, and all sources of materials used for this thesis have been duly acknowledged.

Name	Signature
Menilk Sahlu Bayeh	_____

This MSc thesis has been submitted for examination with my approval as a thesis by

Name: Yunkoo Chung (Professor)

Signature: _____

Date of submission: January 22, 2021

ADVISOR APPROVAL SHEET

To: Department of Computer Science and Engineering (CSE)

This is to certify that the thesis entitled “A Branching Convolutional Encoder-based Spatio-spectral Dimensionality Reduction Model for Hyperspectral Image Classification and Change Detection”, submitted in partial fulfillment of the requirements of the Master of Science Degree in Computer Science and Engineering, and has been carried out by Menilk Sahlu Bayeh Id No. PGR/18208/11 under my supervision. Therefore, I approve that the student has fulfilled the requirements and hereby can submit the thesis to the department.

Yunkoo Chung (Prof.)	_____	January, 2021
Name of Advisor	Signature	Date

ACKNOWLEDGEMENT

First, I would like to thank Almighty God, for helping me reach this milestone after countless ups and downs – literally countless. My deepest gratitude goes to my advisor Yunkoo Chung (Ph.D.) for his persistent guidance, valuable support, and supervision from proposal to the completion of this thesis. I am also thankful for Mesfin Abebe (Ph.D.) and Worku Jifara (Ph.D.) for their voluntary evaluation of my work and support during the research.

I am grateful for Anteneh Tilaye (MSc.), and Kirubel Abebe (MSc.), and Fetlework Kedir (MSc.) for their support and guidance, without which things would have been much more complicated. This thesis came in to being with the continued support of my friends Minaymer Gelawe (MSc.) and Yosef Dentamo(MSc.). Mule, Master Ashe, Diriba and CVR SIG member have been supportive throughout my research.

Last but not least, my love goes to my family Sahlu, Zeni, Seble, Serke, and, Redi; the other equally important family I have at Almi's, my relatives and close friends, who have been continuously nagging me to bring the research to an end. I cannot begin to comprehend the love and support you have for me. I hope the future will be bright and beautiful with you.

ABSTRACT

Hyperspectral images are characterized by having high spectral resolution allowing the representation of a reflectance of a given scene in hundreds of bands. This property however transformative it might be in different applications; the curse of dimensionality is also inevitably present when trying to extract information and analyze hyperspectral data. Several dimensionality reduction techniques have been used to reduce the dimension of hyperspectral images such as principal component analysis, and numerous autoencoder variants. More recently, deep learning architecture-based reduction schemes have been devised for tacking the curse of dimensionality in hyperspectral images. Nonetheless, these architectures are not cognizant of both the spatial and spectral information available in hyperspectral images when reduction the original dimension. In this thesis, a branching convolutional encoder-based spatio-spectral hyperspectral image dimensionality reduction technique called "BCE (BCE)" is proposed. The branching architecture consists of a pointwise separable convolution to extract spectral features, and a two-dimensional convolution network to filter spectral feature. Later, these two features are fused and fed into a decoder network which attempts to reconstruct the original image as accurately as possible. This network is trained in a similar fashion to autoencoders, using a loss function to track the similarity between the original and the reconstructed image. Classification and change detection are important applications of hyperspectral images. The branching convolutional encoder is used together with two classification and change detection models to demonstrate its feature representation performance – the raw image has redundant features and poor interclass separability. The performance of the proposed dimensionality reduction model is compared with a spatial convolutional encoder and a densely-connected encoder. The L1 loss and L2 losses were down to about two digits after decimal point which is lower than the other two methods. Classification accuracy reaches over 90% on all the datasets which outperforms the other comparative methods. Moreover, the branching encoder's representation power is observed with the change detection model; as the rate of accuracy reaches over 99% which for the Hermiston City data. This research demonstrably presents the success of a branching convolutional dimensionality encoder for classification and change detection applications.

***Keywords**: Hyperspectral image, dimensionality reduction, latent representation*

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

3D ..... *Three-Dimensional*
CCA ..... *Canonical correlation Analysis*
CCD ..... *Charge Coupled Device*
CSA ..... *Cosine Angle Similarity*
EMS ..... *Electromagnetic Spectrum*
ESA ..... *European Space Agency's*
FOV ..... *Field of View*
GAN ..... G*enerative Adversarial Network*
HSI ..... *Hyperspectral Images*
KL ..... *Kullback–Leibler*
LDA ..... *Linear Discriminant Analysis*
LSTM ..... *Long Short-Term Memory*
MSE ..... *Mean Squared Error*
MSI ..... *Multi-spectral Image*
NASA ..... *National Aeronautical Space Administration*
NDA ..... *Normal Discriminant Analysis*
PCA ..... *Principal Component Analysis*
RGB ..... Red-Green-Blue
ROSIS ..... *Reflective Optics System Imaging Spectrometer*
RS ..... *Remote Sensing*
SCM ..... *Spectral Correlation Measure*
SNR ..... *Signal-to-Noise Ratio*
SSDAN ..... *Semi-Supervised Deep Autoencoder Network*
SSSE ..... *Spatial-spectral Schrödinger Eigen maps*
t-SNE ..... *t-distributed Stochastic Neighbor Embedding*
USI ..... *Ultraspectral Image*

CHAPTER 1

INTRODUCTION

1.1. Background of the Research

Remote sensing images have three resolution characteristics that determine the quality and application of the images taken [1]. These are namely temporal resolution: refers to how often images are captured by the sensor or the time difference between two similar images captured at different times; spatial resolution: correspondence between a single pixel and actual size, such as 1 pixel covering 5m x 5m area on the ground; and spectral resolution: the number of spectral bands used sensed by the image [2]. Any application of remote sensing (RS) images should carefully choose the proper combination of the above resolution aspects [2][3][4].

State-of-the-art research and commercial applications utilize multiple spectral resolution images such as multi-spectral images (MSI), hyperspectral images (HSI), and ultra-spectral Images (USI). In all three cases, the image captured consists of pixel representation in tens or hundreds of spectral lines[2]. This enriches the image with a lot of information available allowing information processing over the wide range of Electromagnetic Spectrum (EMS) available [5]. Nevertheless, the large number of spectral representations is analogous to a lot of features being used as attributes, making the image higher dimensional than the conventional Red-Green-Blue (RGB) images. This makes processing and analysis difficult for traditional hand-crafted techniques that use complex mathematical functions to determine features for further analysis [6].

With the recent success of deep learning techniques that build hierarchical features over several iterations to represent images, numerous research works have employed deep learning architectures in remote sensing [2]. Especially convolutional neural networks have been used extensively to extract and build hierarchical features for remote sensing applications such as segmentation, anomaly detection, classification, etc. [7]. Deep learning research concerning remote sensing has been made widely accessible due to the numerous public datasets available online. Multi-spectral and hyperspectral images are collected using sensors installed on aerial vehicles and satellites like the National Aeronautical Space

Administration's (NASA) Landsat, Earth Observing-1 (EO-1) Satellite, the European Space Agency's (ESA) Sentinel, etc. [7][2][8].

Multiclass classification and change detection are important applications of these high spectral resolution images used to track changes in a specified geographic location [8][9]. Multitemporal hyperspectral images are used to track changes in precision agriculture, mineral mining and exploration, security surveillance for military applications, monitoring environment for hazard and natural disasters, etc. Considering the hundreds of bands and redundant information available in HSIs, it is quite evident that these images are useful for spectral analysis to track changes in a desired area in more detail [10]. However, this advantage also introduces computational and algorithmic complexity. Performing computations on high dimensional data is resource intensive and takes a long time. This complexity introduced by high dimensional data is called "the curse of dimensionality" [11].

This problem has been researched for over 20 years now with different techniques ranging from complex feature extractors and change detection algorithms, going through machine learning algorithms in the 2000s and most recently with deep learning architectures [12][4][3]. Feature extraction and selection algorithms were employed to reduce the dimension of the original hyperspectral data making the output easy to manipulate for further applications. But, the most feature extraction algorithms do not consider both the spatial and spectral information available in HSIs when forming a feature representation [2], [8].

In this research efficient ways of autoencoder based dimensionality reduction techniques were explored that involve a deep spatio-spectral feature extractor that creates an efficient latent feature representation for classification and change detection applications. The efficiency of the feature extractor is tested via different classification algorithms. After this has been demonstrated the feature extractor is used with the change detection algorithm. These two are trained in a coupled manner and the effectiveness of the entire framework is tested by different performance metrics. Finally, the framework's performance is compared with existing algorithms to prove its effectiveness of existing techniques.

1.2. Motivation of the Study

Ever since hyperspectral images entered the arena of remote sensing study, multi-temporal change detection whether it be binary, multiclass, or time-series change detection, has been a topic of extensive research. Throughout the years several change detection techniques that were built on top feature extractors, use complex mathematical analysis, and machine learning algorithms were being used with some success [2][13]. Change detection based on the above methods suffers from design complexity, a stagnating accuracy in the rate of detection, and does not efficiently scale up to different conditions and scenarios.

In 2005 Geoffrey Hinton and R. Salakhudinov demonstrated that neural networks, specifically autoencoders fine-tuned with gradient descent, can be used to create codes (latent representations) of high-dimensional data [14]. Several works have been done to reduce the dimension of hyperspectral images using several autoencoder variants (multilayer perceptron-based stacked autoencoders, convolutional autoencoders, etc.)[15][16][17]. Nonetheless, existing most of these works do not consider effectively representing both the spectral information and the spatial information when forming the latent rendering. This constrains the encoded information from being effectively used by classification and change detection algorithms [18][19].

Moreover, several of the existing change detection techniques reviewed also do not integrate a spatio-spectral change tracking mechanism to consider not only the spatial changes but the corresponding spectral changes. Of the existing deep learning architectures recurrent neural networks are a good candidate for this task. Recently Long Short-Term Memory (LSTM) based neural networks have shown a promising result for hyperspectral change detection [20]. In this work a three-dimensional (3D) Convolutional LSTM (ConvLSTM) architecture is customized to blend the spatial and spectral correspondence within the multitemporal hyperspectral cubes to form an accurate multiclass change detection map [21]. Since ConvLSTM has a long training time and a fairly large parameter count, a spatio-spectral convolutional feature extractor is used as a dimensionality reduction preprocessing module and trained with the ConvLSTM in an end-to-end manner.

1.3. Statement of the Problem

Hyperspectral images have hundreds of bands that create high dimensional data complexity for classification and change detection applications. Therefore, an intermediary dimensionality reduction step is required before the image is directly fed to the either the classification or change detection network. Most of the existing dimensionality reduction do not include both the spectral and spatial information when creating a feature representation of the original images. Because of this either spatial or spectral information is lost when the after the input has passed through the dimensionality reduction step.

In order to overcome this challenge, a branching convolutional encoder spatio-spectral dimensionality reduction called Branching Convolutional Encoder (BCE) is proposed in this paper. The two branches of the encoder encode spatial and spectral information together to create an effective latent representation of the original hyperspectral image inputs. This equivalent to a feature extractor step except once the encoder has been trained in an encoder decoder fashion it will be excised and merged with either the classification or change detection models.

1.4. Research Questions

In line with the problem stated above this research aims to answer the following questions to design and implement a solution.

> Research question 1: What are the state-of-the-art hyperspectral dimensionality reduction mechanisms in use today?
>
> Research question 2: How to design an effective spatio-spectral dimensionality reduction mechanism that has an efficient latent feature representation?
>
> Research question 3: How to integrate the designed dimensionality reduction model with common applications such as hyperspectral image classification and change detection, and test the model's performance?

1.5. Objectives

1.5.1. General Objective

Overall, this research is centered around the design and implementation of a spatio-spectral dimensionality reduction architecture that creates an efficient spatio-spectral latent representation for classification and change detection of hyperspectral images.

1.5.2. Specific Objectives

Having mentioned the design and implementation of a portable spatio-temporal dimensionality reduction architecture as the main objective of this research, the following specific objectives were laid out and executed.

- Acquisition and preparation of datasets for hyperspectral image applications
- Designing a deep feature extractor dimensionality reduction scheme to transform the individual high dimensional images to a manageable feature representation
- Demonstrating the effectiveness of the intermediary feature extractor under different scenarios to show its efficacy in creating a well representative latent representation
- Implementing a multiclass classification model that is integrable with the dimensionality reduction architecture for testing the dimensionality reduction's scheme performance
- Implementing a multiclass change detection model to detect changes between bi-temporal images

1.6. Significance of the Research

In the era of deep learning where large scale data is readily available and is being rapidly produced, models should not be limited a small-scale application. That is models should be applied to different scenarios and serve as an intermediary-steps for other applications. This research provides a model that can be scaled for and trained on large datasets to serve as a baseline for other works such as classification and change detection. The dimensionality reduction architecture presented in this thesis has been applied to different HSIs with success

in reconstructing the original image. Further with the portability of the architecture it can be used for different HSI applications such as binary and multiclass change detection, multiclass classification, etc. Future works might benefit from adapting this model and scaling it up to fit a large dataset so that it can serve as a benchmark for numerous applications.

1.7. Scope and Limitations of the Research

1.7.1. Scope of the Research

The research presented in this thesis covers dimensionality reduction for hyperspectral images available online. Since the datasets that were selected for this research were those that have ground truth labels, this research covers only supervised training mechanism for hyperspectral classification and change detection. Also, the feature extractor step is aimed to work with classification and change detection models including machine learning and deep learning techniques. This dimensionality reduction is modular and can be integrated with classification and change detection models and be trained in and end to end manner to further specialize the module for the specific task.

1.7.2. Limitations of the Research

One of the main challenges while doing this research was the shortage of hyperspectral images that have a corresponding ground truth label. This has limited the experiment to the small number of datasets available and made it challenging to design and train a model for large scale application. Even though the architecture can be customized to train on and fit large-scale data, it was not possible to show this in large scale due the shortage of annotated data and the large-scale computational resource required. Also, associated with shortage of annotated dataset the dimensionality reduction's performance was tested only on bitemporal image change detection. This has hindered the model from being tested on time-series hyperspectral data.

1.8. Application of the Research

Hyperspectral imagery is used in numerous applications that are not limited to remote sensing, food security, biomedical application, material study, etc. The proposed

dimensionality reduction can greatly reduce the computation time and complexity involved and eases ready application of the HSIs for hyperspectral classification and change detection.

1.9. Organization of the Thesis

This thesis has been organized in to seven chapter each of which highlights an important aspect of the research providing a rationale and

Chapter One: this chapter gives an introduction to the research by providing the necessary insight to understand the area and a reasoned justification of the research problem.

Chapter Two: in this chapter the necessary background literatures that provide a theoretical framework have been presented to highlight this research. Moreover, recent papers that focus on

Chapter Three: the third chapter gives a detailed explanation of the research methodology starting from the collection of the dataset to testing of the model's performance with relevant performance metrics.

Chapter Four: gives a detailed description of the method proposed and how this model solves the problem stated in the first chapter.

Chapter Five: is an a somewhat extensive description of the implementation details of the proposed solution. Several code snippets are provided to give a glimpse in to what the implantation source code looks like.

Chapter Six: presents the results obtained from the experiments mentioned in the fifth chapter. It contains, comparative figures, graphs, and tables. It also has a comparative description of the model's performance with other architectures under different scenarios.

Chapter Seven: summarizes the entire research by commenting upon the results obtained from the experiment and gives a short description on the implications of these results.

CHAPTER 2

THEORETICAL BACKGROUND AND RELATED WORKS

2.1. Theoretical Background

2.1.1. Hyperspectral Remote Sensing (Imaging Spectrometry)

Hyperspectral imagery involves acquiring images through the use of specially designed sensor that passively capture hundreds of contiguous narrow bands (each band can be as small as 10nm) in the visible and infrared sections of the electromagnetic spectrum. Each pixel in the hyperspectral cube contains a spectral information, and the hundreds of corresponding pixels on each raster provide rich information for analysis. Hyperspectral imaging sensors can be deployed on drones, planes, satellites, and medical scanners, etc. [3] [4][12][13][22]. The images taken can be used for remote sensing applications such as land cover classification and change detection, biomedical applications, forensic investigation, and study, etc. Specifically, for remote sensing, several hyperspectral image datasets are publicly available for classification, change detection, and time-series monitoring.

Imaging spectroscopy typically uses a 2D matrix array (e.g., a charge couple device [CCD]) to produce a three-dimensional data cube (namely the spatial dimensions (height x width) and a third spectral dimension) [4][12]. These progressive data cubes are constructed in two ways either by consecutively registering a single complete spatial image of different wavelengths one after the another, or sequentially recording one narrow image (one-pixel wide, multiple pixels long) swath after another with the corresponding spectral signature for each pixel in the path.

Although imaging spectrometry has numerous advantages over its predecessors, it's not without its downsides. For instance, during acquisition of high-quality spectral data from aerial and space vehicles, it is difficult to keep the conditions constant, optimal, and well controlled [12]. Moreover, considerable interference is encountered, such as the lower signal-to-noise ratio (SNR) observed due to the short dwell time of data acquisition over a given pixel; dissipation of reflected signal via scatterings and absorption due to gasses and aerosols; and the random luminance of external sources and the objects themselves. Spectral

data acquired in this manner introduces undesirable artifact/occlusion elements requiring several corrections (geometric and radiometric) before being directly usable for the desired application.

2.1.2. Hyperspectral Imaging Techniques

Several aerial and space vehicles have imaging spectrometer installed for remote sensing applications available for free and some at a fee [8][23]. The data collected from these platforms differ in spatial resolution, spectral resolution, temporal resolution, spectroscopy technique, etc. These sensors acquire hyperspectral images in two prominent manners namely whiskbroom imaging and pushbroom imaging.

2.1.2.1. Whiskbroom Hyperspectral Imaging (Point Scanning)

Whiskbroom scanners are designed to capture the desired field of view (FOV) using a scanning mirror that either hoovers from edge to edge or revolves with the entire sensor system. The output of these scanners is a single spatial pixel along with the several hundreds of spectral constituents (bands) captured [8]. Successive lines are formed using an across-track scanning mechanism which involves the broom heads scanning from left to right perpendicular to the direction of the aerial vehicle. Whiskbroom scanning is beneficial in that it provides a greater spectral consistency as the pixels are collected via an array of detector line allowing for a larger pixel size to be detected. Some popular imaging spectrometers designed using the point scanning principle are the air-borne AVIRIS, DAIS, and HyMap instruments, as well as the spaceborne MODIS instrument. The data formation process under whiskbroom scanning is given on Figure 2.1 [23][4].

2.1.2.2. Pushbroom Hyperspectral Imaging

A pushbroom sensor is an imaging spectrometer which acquires a series of one-dimensional samples constructed along track with the forward motion of the platform perpendicular to the platform line of flight. The image formation under the pushbroom scanners is entirely due to the forward movement of the sensor. The spectral component is acquired by dispersing the incoming radiation onto an area array [3][4][12][13][22]. A pushbroom scanner simultaneously records the across-track dimension x (perpendicular direction to the line of flight), along-track dimension y (line of flight direction) and a spectral dimension λ,

consisting of the several hundreds of bands that make up the image [13][23]. Pushbroom scanners have the advantage of a longer dwell time allowing for each individual detector element integration to form the image in comparison with scanners that work with whiskbroom scanning mechanism. Popular pushbroom-based imaging spectrometers include the airborne CASI and ROSIS instruments and the spaceborne MERIS and Hyperion. The data formation process under whiskbroom scanning is given on Figure 2.1.

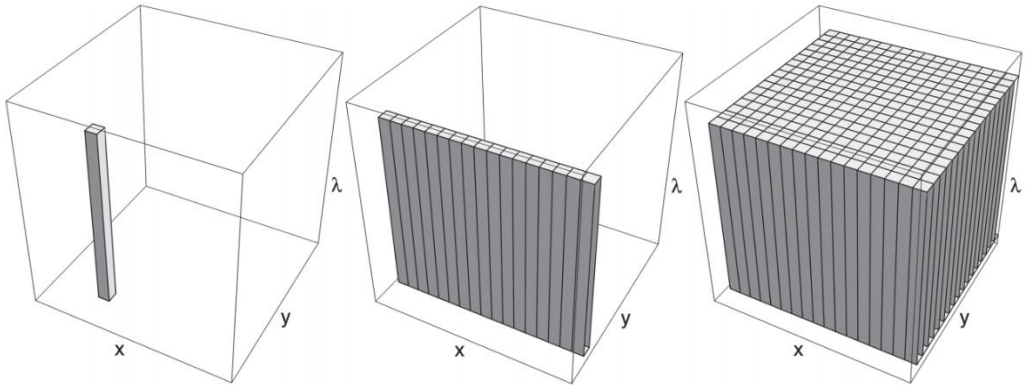


Figure 2. 1. Progressive data formation under whiskbroom scanning mechanism

(Source:[4])

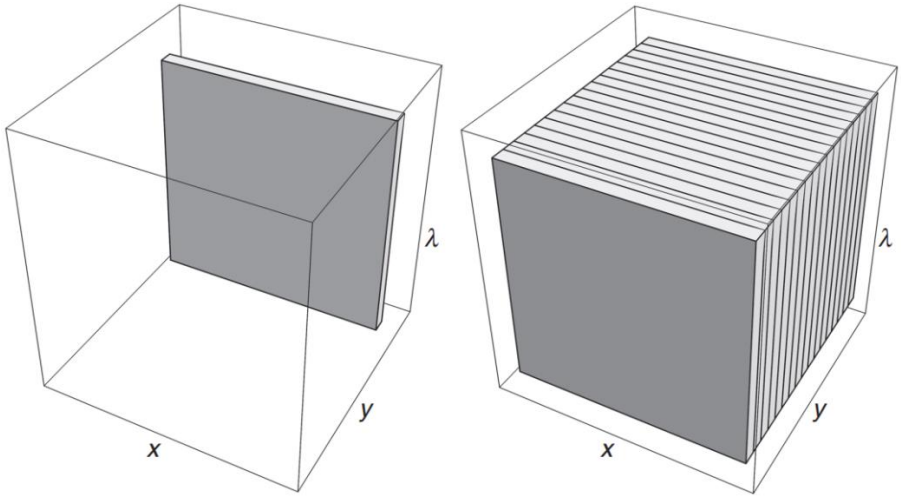


Figure 2. 2. Progressive data formation under pushbroom scanning mechanism

Source: ([4])

2.2.2. Hyperspectral Imaging Sensors

Of the several hyperspectral imaging sensors the two most popular and widely recognized spectrometers used on space shuttles and aerial vehicle platforms have been mentioned here.

2.2.2.1. Airborne Visible/Infrared Imaging Spectrometer (AVIRIS)

The AVIRIS sensor is the first whiskbroom scanning mechanism-based sensor that was designed by the National Aeronautics and Space Administration's (NASA) Jet Propulsion Laboratory to be carried by an aerial platform [23]. It has 614-pixel swaths scanned over 224 lines consisting of 224 bands each band having a spectral range of 380 – 2500 nm captured using scanning optics and a group of four spectrometers at 10nm spectral resolution. Any given scene image captured by AVIRIS consists of 614 pixel-swaths × 512 lines of scanning × 224 bands (10nm resolution). The image cube formed at the end of the scanning procedure has two axes representing the spatial dimensions (area of the scene), and the third representing a spectral dimension.[1]

2.1.2.2. Hyperion (Hyperspectral Imager, EO-1 Satellite)

Hyperion, also known as the first imaging spectrometer installed on a space craft, is one of three sensors deployed on the Earth Observing-1 (EO-1) satellite [3][4][12][13]. EO-1 was launched about 20 years ago on the 21st of November 2000 but has been decommissioned since the 20th of March 2017. EO-1 used to fly on the sun-synchronous orbit collecting identical images with the Landsat-7 satellite upon the completion of a complete orbit. Hyperion images capture 7.5 km by 100 km land area per image across 220 spectral bands (from 0.4 to 2.5 μm) with a 30-meter spatial resolution. The Hyperion acquires each frame of the data using pushbroom scanning mechanism whereby the spectrum is captured by a 30 m line alignment in the along-track direction by 7.5 km in the crosstrack direction. [2]

[1] AVIRIS Images are available online at https://aviris.jpl.nasa.gov/data/get_aviris_data.html

[2] USGS hosts Hyperion images at https://earthexplorer.usgs.gov/

2.1.4. Deep Learning: A Brief Introduction

Deep learning is a learning technique where several layers of neurons (different neuron types based on application) are stacked up to form multiple layered network capable of representation learning [24]. This representation learning process can be done in supervised, semi-supervised or unsupervised manner based on the availability of annotated/labelled data. Using multiple layers of neurons began with the demonstration that a single hidden layer perceptron network made up of non-polynomial activation functions can be used as a generic classifier even though the training process might be long and requires complicated tuning. Currently, deep learning research and application is concerned with an unbounded number of layers of bounded size, where trainings can be replicated without significant deviations amongst each other with a reasonable number of parameters to be controlled. Based on the what the models learn deep learning models can be classified in to generative modelling and discriminative modelling [25].

Discriminative Modelling: Discriminative learning includes any classification learning process that uses the *estimate of the probability P(y / x)* having no need for an explicit estimate of any of *P(x), P(y, x),* or *P(x / y),* where *y* is a categorical class and *x* is a description of an object to be classified [26][27][28]. In contrast with generative learning which classifies by using an *estimate of the joint probability P(y, x)* or of the prior probability *P(y)* and the conditional probability *P(x / y), discriminative modeling does need any of those.*

Generative Modelling: Generative learning alludes then again to any grouping learning measure that characterizes by utilizing a gauge of the joint likelihood *P(y, x)* or to any arrangement learning measure that orders by utilizing assessments of the earlier likelihood *P(y)* and the contingent likelihood *P(x / y),* where *y* is a class and *x* is a depiction of an item to be arranged [27][28]. Given such models or gauges, it is conceivable to create manufactured items from the joint appropriation. Generative learning differences to discriminative learning in which a model or gauge of *P(y / x)* is shaped without reference to an express gauge of any of *P(x), P(y, x),* or *P(x / y).* It is likewise normal to arrange as discriminative methodologies a functional mapping from input x to yield y by means of the express assessment of *P(x), P (y, x),* or *P(x | y).* In generative models, this is switched as the mapping happens without express assessment of the earlier probabilities. The standard

illustration of generative learning is gullible Bayes and of its partner discriminative learning, strategic relapse.

2.1.4.1. Common Deep Learning Architectures

The following architectures are widely used deep learning models that have been extensively used for several applications and have been implemented in this research. Here their basic description along with their block diagram representation have been given.

Autoencoders: Autoencoders fall under unsupervised learning method wherein neural organizations are utilized for the undertaking of portrayal learning[29]. In particular, the neural network architecture is planned with the end goal that a bottleneck in the organization which powers a packed information portrayal of the first info. On the off chance that the information highlights were every free of each other, this pressure and resulting recreation would be an extremely troublesome assignment. In any case, if some kind of design exists in the information this construction can be learned and subsequently utilized while compelling the contribution through the organization's bottleneck [30].

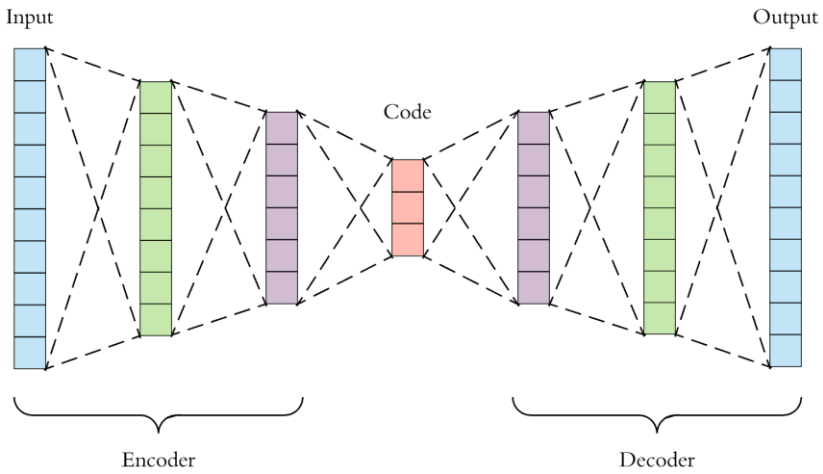


Figure 2. 3. Basic multilayer autoencoder architecture

Source: (https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798)

It is possible to take an unlabeled dataset and frame it as a supervised learning problem tasked with outputting, a reconstruction $\tilde{x}$ of the original input x. This network can be trained

by minimizing the reconstruction error, $L(x,\ \tilde{x})$, which measures the differences between our original input and the consequent reconstruction. The bottleneck is a key attribute of our network design; without the presence of an information bottleneck, our network could easily learn to simply memorize the input values bypassing these values along through the network which is plotted below on Figure 2.3.

Convolutional Neural Networks (CNNs): A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning calculation that can take in an info picture, relegate significance (learnable loads and inclinations) to different viewpoints/objects in the picture and have the option to separate one from the other [31]. The pre-handling needed in a ConvNet is a lot lower when contrasted with other arrangement calculations. While in crude strategies channels are hand-designed, with enough preparation, ConvNets can gain proficiency with these channels/attributes [32][33] .

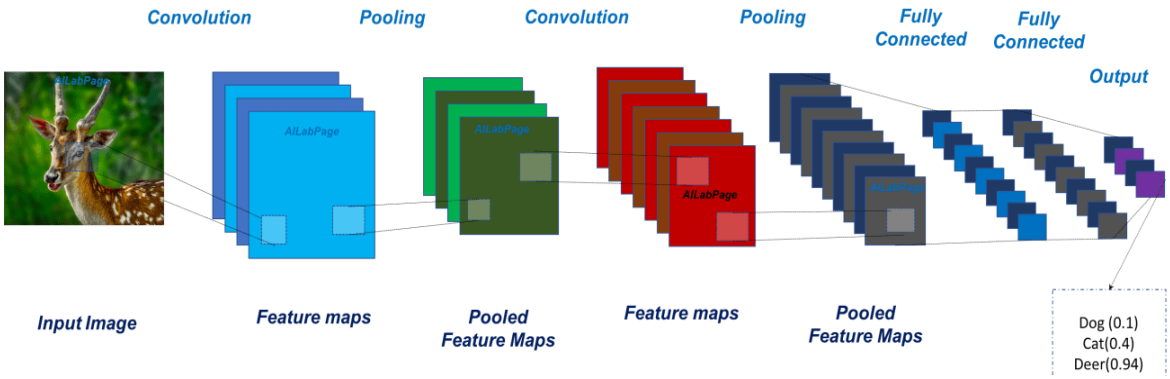


Figure 2. 4. Common pattern for convolutional neural networks classification architectures

Source:(https://i0.wp.com/vinodsblog.com/wp-content/uploads/2018/10/CNN 2.png?w=2600&ssl=1)

A ConvNet can effectively catch the Spatial and Temporal conditions in a picture through the utilization of applicable channels. The engineering plays out a superior fitting to the picture dataset because of the decrease in the number of boundaries included and the reusability of loads. All in all, the organization can be prepared to comprehend the complexity of the picture better[31].

Long Short-term Memory: Long Short-term Memory (LSTM) networks are variants of Recurrent Neural Network Architectures that are being able to do specifically recollecting designs for a long length of time [34]. It is an ideal decision to demonstrate successive information and subsequently used to learn complex elements of human action. The drawn-out memory is known as the cell state. Because of the recursive idea of the cells, past data is put away inside them. They fail to remember door set beneath the cell state is utilized to adjust the cell states. They fail to remember entryway yields esteems saying which data to fail to remember by increasing 0 to a situation in the lattice. On the off chance that the yield of the fail to remember the door is 1, the data is kept in the cell. The input gates figure out which data ought to enter the cell states. At long last, the output gate tells which data ought to be given to the following next states [35].

Two of the significant variants for the LSTM model are Deep LSTM (DLSTM) and CLSTM. DLSTM varies from the overall LSTM in the quantity of layers the model contains. A solitary layer LSTM won't have the option to get all around characterized transient data. Nonetheless, when more layers are stacked in the LSTM model, it will have the option to secure better worldly highlights, and subsequently will be more reasonable in catching movement in the time measurement. In ConvLSTM, the information is first gone through convolutional layers, which guarantee in catching the spatial highlights [36].

The yield from the CNN is given to the LSTM, which will get the transient highlights, and subsequently, the model will catch a movement regarding both realities. These two variations can likewise be consolidated to give a convolutional Deep LSTM, where the yields from a CNN are given to a multilayer stacked LSTM, which is ensured to give a superior outcome at the expense of expanded computational intricacy. Figure 2.5 below show the block diagram representation of LSTM layer. Where Xs are inputs at different points in time (*t-1, t, t+1*), plus sign represents addition (concatenation), σ is sigmoid activation, tanh is an activation block. H represents hidden states which are outputs of each cell from its current and previous state [37].

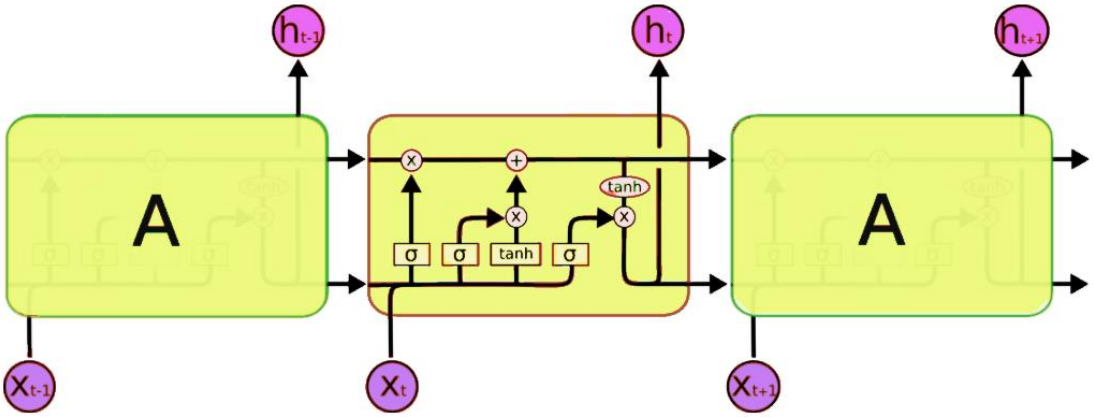


Figure 2. 5. LSTM block diagram representation and internal components

Source: (https://cdn.analyticsvidhya.com/wp-content/uploads/2017/12/10131302/13-768x295.png)

2.1.5. Dimensionality Reduction

Dimensionality decrease is the change of information from a high-dimensional space into a low-dimensional space with the goal that the low-dimensional portrayal holds some important properties of the first information, preferably near its natural measurement. Working in high-dimensional spaces can be unfortunate for some reasons; crude information is regularly scanty as a result of the scourge of dimensionality, and examining the information is typically computationally obstinate. Dimensionality decrease is regular in fields that manage enormous quantities of perceptions as well as huge quantities of factors, for example, signal processing, speech recognition, neuro-informatics, and bioinformatics [38][39][40].

Popular dimensionality reduction techniques are commonly divided into linear and non-linear approaches. Approaches can also be divided into feature selection and feature extraction based on how features are treated. Dimensionality reduction can be utilized for noise reduction, information perception, bunch investigation, or as a transitional step to facilitate different examinations [40].

2.1.5.1. Feature Selection Techniques

Feature selection strategies, where just the most significant or unmistakable dimensions/measurements are held and the leftover are disposed of. Highlight determination is the investigation of calculations for decreasing the dimensionality of information to improve AI execution [41]. For a dataset with N highlights and M measurements (or highlights, ascribes), include choice means to diminish M to M' and $M' \leq M$. It is a significant and generally utilized way to deal with dimensionality reduction. The goal of feature selection is to eliminate unessential as well as repetitive highlights and hold just pertinent highlights. Unimportant highlights can be taken out without influencing learning execution. Excess highlights are a sort of unimportant highlights. The qualification is that a repetitive component infers the copresence of another element; exclusively, each element is significant, yet the evacuation of possibly one won't influence learning execution.

2.1.5.2. Feature Projection (Feature Extraction) Techniques:

Feature projection (additionally called Feature extraction) changes the information from the high-dimensional space to a space of smaller measurements. The information change might be straight, as in Principal Component Analysis (PCA), however numerous nonlinear dimensionalities decrease methods likewise exist. For multidimensional information, tensor portrayal can be utilized in dimensionality decrease through multilinear subspace learning [41][42].

Principal Component Analysis (PCA): The Principal Component Analysis of centers in a real p-space are plans of bearing vectors, where the vector is the heading of a line that best fits the data while being balanced to the essential vectors [43]. Here, a best-fitting line is portrayed as one that restricts the typical squared partition from the concentrations to the line. These headings include an orthonormal premise in which assorted individual parts of the data are straightforwardly uncorrelated. Principal component analysis (PCA) is the path toward calculating the significant sections and using them to play out a distinction in reason on the data, every so often using simply the underlying very few head fragments and neglecting the rest. PCA is utilized in exploratory information examination and for making prescient models. It is generally utilized for dimensionality reduction by projecting every

information point onto just the initial not many head segments to acquire lower-dimensional information while protecting however much of the information's variety as could reasonably be expected[44][45]. The primary head segment can proportionately be characterized as a course that augments the change of the projected information.

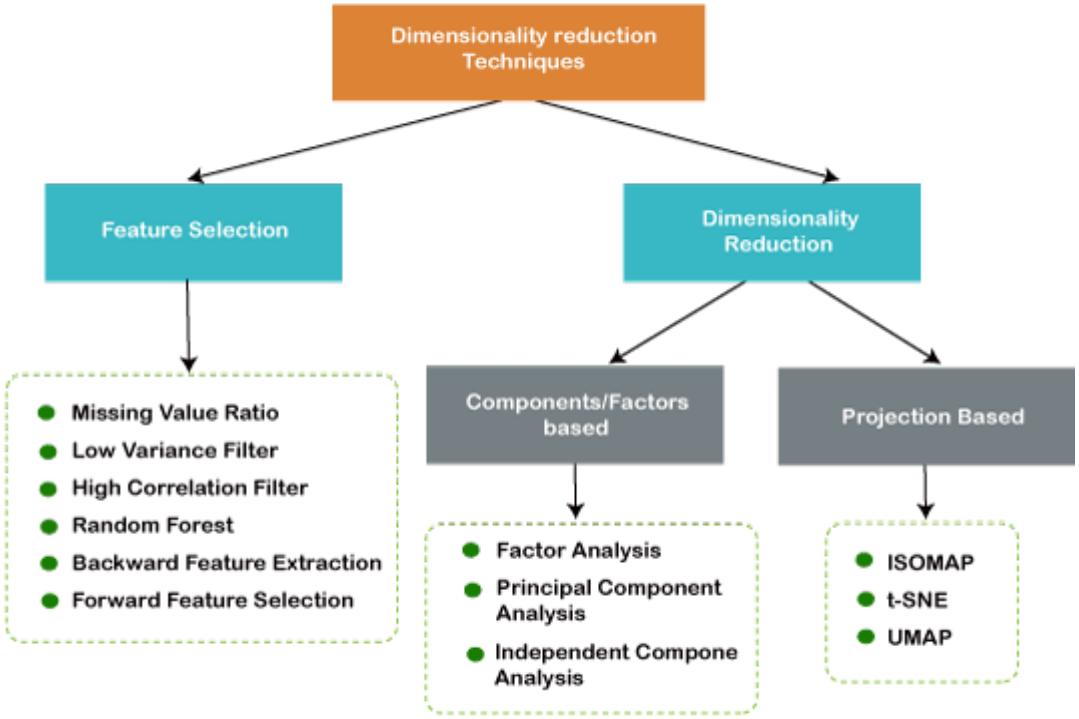


Figure 2. 6. Dimensionality reduction types

Source: (https://static.javatpoint.com/tutorial/machine-learning/images/dimensionality-reduction-technique.png)

Linear Discriminant Analysis (LDA): Linear discriminant analysis (LDA) is an abstraction of Fisher's direct discriminant, a technique utilized in insights, design acknowledgment, and AI to locate a direct blend of highlights that portrays or isolates at least two classes of articles or occasions. The subsequent mix might be utilized as a straight classifier, or, all the more ordinarily, for dimensionality decrease before later order [46][47]. LDA is additionally firmly identified with Principal Component Analysis (PCA) and consider examination that the two of them search for straight mixes of factors that best clarify the data. LDA expressly endeavors to show the contrast between the classes of information. PCA, interestingly, doesn't consider any distinction in class, and factor

investigation fabricates the component blends dependent on contrasts instead of likenesses. The discriminant analysis is likewise unique in relation to figure investigation that it's anything but an association strategy: a differentiation between free factors and ward factors (additionally called criterion variables) should be made.

2.2. Related Works

2.2.1. Hyperspectral Image Dimensionality Reduction Techniques

Lloyd Windrim et. al propose a stacked autoencoder based unsupervised feature learning for Dimensionality reduction [15]. The feature learning algorithm projects feature representation in to a latent space so as to reduce the dimension of the original data to be used for classification application. Moreover, they integrated an information theoretic measure based spectral information divergence metric to make the reconstructed output spectrally similar with the original input. Also, they used Cosine Angle Similarity (CSA) measure to train the autoencoder's latent representation efficient. However, their method does not consider the spatial aspect of the representation as they used a 1D stacked autoencoder which learns to reconstruct the original spectra.

Another work by Zhang et. al uses a semi-supervised deep autoencoder network (SSDAN) to reduce the dimension of hyperspectral data [16]. The semi-supervised approach here allows the model to learn from a small subset of the entire data and use the low-dimensional feature embeddings for pixel-wise classification algorithms. Here they used a mean squared error (MSE) loss function to compute the loss between the original and the reconstructed output. They incorporate a spatial-spectral Schrödinger Eigen maps (SSSE) algorithm that builds graphs with spectral information and uses clusters of these to encode spatial proximity between these graphs.

Ayma et. al propose an orthogonal autoencoder dimensionality reduction approach for classification of hyperspectral images [48].They introduce an orthogonal reconstruction error which is defined as the sum of mean squared error between the original and reconstructed output, and the mean squared error between the latent variable product and an identity matrix I. The orthogonality between components in the latent space is ensured by the loss function and the training optimizer so that the orthogonality of latent components

improves classification performance. They tested their models on the Pavia University, Kennedy Space Center, and Botwana hyperspectral images. However, one obvious draw back in this method is that it does not consider spatial and spatial components during the feature representation.

Filtering approach by using 1D pooling has been proposed by Paul and Chaki to reduce/select important spectral features while reducing the dimension of the original image [49]. The one benefit of this method is that it clearly reduces the computation time taken while reducing the dimension and is less complex in its computation. However, this technique is also in line with the 1D techniques that focus on spectral information primarily, and require reshaping the original input to 2D pixel vector array which jumbles spatial information along the process.

The alternative candidates to 1D autoencoders are convolutional autoencoder that capture spatial information from the hundreds of channels present in the image. Mei et. al propose a 3D convolutional autoencoder spatial-spectral feature learning for hyperspectral image classification application [19]. Elementwise 3D convolutions, 3D pooling and batch normalization have been included in this work. The encoder is trained together with a decoder counter part that attempts to reconstruct the original input from the spatial-spectral feature representation. They tested the model's performance on an SVM classifier to test the feature representation performance allowing the components to be easily classified.

Chunju Zhang et. propose a three-dimensional densely connected convolutional network (DenseNet) for hyperspectral spectral-spatial feature learning to improve classification performance [50]. DenseNet consists spatial dimension feature extractor along with the addition of a spectral dimension to the original DenseNet architecture. All operations are in 3D including the batch normalization layers which are included after convolution steps. The bottleneck layer formed by the reducing number of input size creates a spatial-spectral feature representation after several iterations.

Xin Zhang et. al propose a spatial-spectral 3D CNN architecture called (SSDANet) as a feature extractor step for hyperspectral image classification task [50]. The feature extractor is used as a dimensionality reduction step by using spectral-spatial dense connectivity to preserve the relationship between the spectral values in the pixel vector and the

neighborhood information which is responsible for preserving spatial information. Moreover, they use spatial dense connectivity block and spectral dense connectivity block which is ensured by the dense architecture that connects the intermediary convolution step outputs to successive stage convolution outputs. The output of the two dense connectivity blocks is reduced by batch normalization and 3D average pooling to reduce its size before being fed into the classifier.

There is a clear progression from 1D multilayer perceptron autoencoder techniques to higher dimensional convolutional autoencoder to train encoder to create latent representation learning either from hundreds of bands spatially, or from both the spatial and spectral representation. However, separate treatment of the spatial and spectral components is yet to be seen as spectral relationships, spatial relationships and spatio-spectral relationships need to successively considered when designing feature extractor-based dimensionality reduction techniques.

2.2.2. Hyperspectral Image Classification Techniques

It is the trend in hyperspectral image research to demonstrate the power of dimensionality reduction scheme together with hyperspectral classification techniques. Most of the papers cited for the dimensionality reduction steps are implemented for hyperspectral classification. Wei Hu et. al propose deep convolutional neural network for classification of hyperspectral images in the spectral domain. First, the original size image is reduced via several layers of convolution and pooling. Then, this reduced representation is fed to a full connected layer that has a SoftMax activation. The model's performance is tested with bench mark datasets and the results were encouraging to say the least.

Shafaey et. al adopt the popular AlexNet image classification architecture for the classification of hyperspectral images. This is done to show the transferability of architectures originally proposed for optical images for hyperspectral image classification task. There was significant improvement in comparison with traditional machine learning techniques despite the fact that these models were originally intended for RGB images.

2.2.3. Hyperspectral Image Change Detection Techniques

Change location (CD) is one significant use of remote sensing currently. CD has a few applications including yet not restricted to urban and ecological checking [51][52]. Compact disc strategies contrast pictures gained at various occasions with recognize changes in land cover estimated by the distant detecting sensors (e.g., reflected radiance for detached optical sensors). Change detection ha has been perhaps the main distant detecting research exercises (on systems and applications) in late many years. Change detection strategies have been created and effectively utilized in various far off detecting applications, for example, agribusiness and ranger service observing, catastrophic event planning, and metropolitan scene and spread examination.

Qi Wang et. al propose an end-to-end 2D CNN framework for hyperspectral image change detection called GETNET [52]. GETNET includes an intermediary sub-pixel unmixing method because most hyperspectral sensor have a low spatial resolution which mixes two or more spectra in the pixel to one spectral value. The unmixing steps extract the end members which are mixed within the and estimate the abundance of these members. Later a mixed affinity matrix is generated that contains which spectral value is favored by the sensors during capturing. This affinity matrix is fed into a convolutional neural network followed by a fully connected network that gives a binary classification result.

Ahram Song et. al propose a recurrent fully convolutional network for the hyperspectral change detection [21]. A fully convolutional network has been merged with Convolutional LSTMs to extract spatial and temporal information between multitemporal images. The authors used PCA as intermediary dimensionality reduction step and 3D patched samples were fed to the recurrent fully convolutional network. Spectral correlation measure (SCM) was used to train the network. This work has been used in this research to demonstrate the power of the hyperspectral dimensionality reduction technique.

2.3. Summary

In this chapter several literatures existing under hyperspectral dimensionality reduction, classification and change detection have been surveyed. For the dimensionality reduction spatio-spectral convolutional networks have been highlighted as the approaches that have

better feature extraction power consisting of spatial and spectral information at the same time. In addition, for the classification and change detection tasks spatio-spectral designs are expected to have the best performance out of the existing methods. Table 2.1 summarizes the related works with different comparisons.

Table 2. 1. Tabulated summary of the reviewed related works

Authors and paper	Methodology	Objective	Limitation
Lloyd Windrim et. al [15]	1D spectral autoencoder	Dimensionality reduction	Does not consider spatial relationships
Ayma et. al [48]	Orthogonality constraint for dimensionality reduction	Dimensionality reduction via orthogonality constratin	Is not spatially robust, considers only feature orthogonality
Mei et. al. [19]	3D convolutional autoencoder	Feature extraction for hyperspectral	Computationally complexity
Xin Zhang et. al [50]	3D convolutional Dense architecture SSDANet	Spatial-spectral feature extraction for classification task	As with 3D architectures require large computational resource
Qi Wang et. al [53]	GETNET: 2D change detection via affinity matrix computation	Computes affinity matrix for spectral unmixing and does feature representation	Limited to spatial computation: does not integrate spectral information
Ahram Song et. al [21]	3DRFCNN recurrent fully convolutional architecture for change detection	Use convolutional LSTMs for computing changes between bitemporal images	State-of-the art method during this research

CHAPTER 3

RESEARCH METHODOLOGY

3.1. Overall Methodology

In this chapter the entire research process has been explained, rationalized, and elaborated with diagrams and tables briefly. Starting from the selection of publicly available datasets to the necessary preprocessing steps taken to ease the training, the preparation of data for training the models have been included in the data collection and preparation section. A glimpse of the justification for selecting the dimensionality reduction (feature-selection) model and change detection scheme is briefly provided. Finally, the software and hardware used, the training process, and the evaluation metrics commonly used for hyperspectral images have been mentioned to give how the performance of the models will be evaluated.

3.2. Hyperspectral Satellite Image Datasets

Datasets that have been included in this research are freely available online made accessible by the generous efforts of remote sensing researchers and practitioners. One final note worth mentioning is that there is a shortage of publicly available preprocessed and labelled hyperspectral multitemporal datasets. This was a challenge to this research as the method proposed could not be trained and tested on a variety of datasets. In addition, the absence of annotated and compiled time-series dataset has limited this research to multi-class change detection. Four datasets were acquired to train and demonstrate the effectiveness of the autoencoder architecture proposed. Out of the four datasets three were selected for classification task due to the density of the classes present in the image. For the multiclass bitemporal image change detection the Hermiston city bitemporal change detection images were chosen since they have five class of changes, unlike other images which have two or three class of changes.

The datasets listed below have undergone initial processing by the publishers who made the images public. Some of these preprocessing steps are *raw sensor value to reflectance conversion, dead band removal, cloud, geometric,* and *radiometric correction*. This makes the images directly usable with multiple algorithms as it eliminates the tedious task of

preprocessing which severely impedes the research progress. Most of the preprocessing is done through the use of commercially available software such as ERDAS, ArcGIS, etc.

3.2.1. Pavia Center Scene Image

Pavia Center Scene dataset was organized by the Telecommunications and Remote Sensing Laboratory at the University of Pavia (Università Degli Studi di Pavia). Reflective Optics System Imaging Spectrometer (ROSIS) sensor mounted on a plane was used to capture aerial images in Pavia, Italy. The image captured has 102 spectral bands, and each raster is composed of 1096 * 715 pixels. Pavia Center Scene dataset has nine classes with one extra class consisting of unclassified pixels where the instances of each class are given on Table 3.1. Also, the ground truth image and the false-color images of the hyperspectral cube are given on Figure 3.1 and Figure 3.2 respectively. This image is used for demonstrating the representation power of the hyperspectral images and later for the classification task [54].

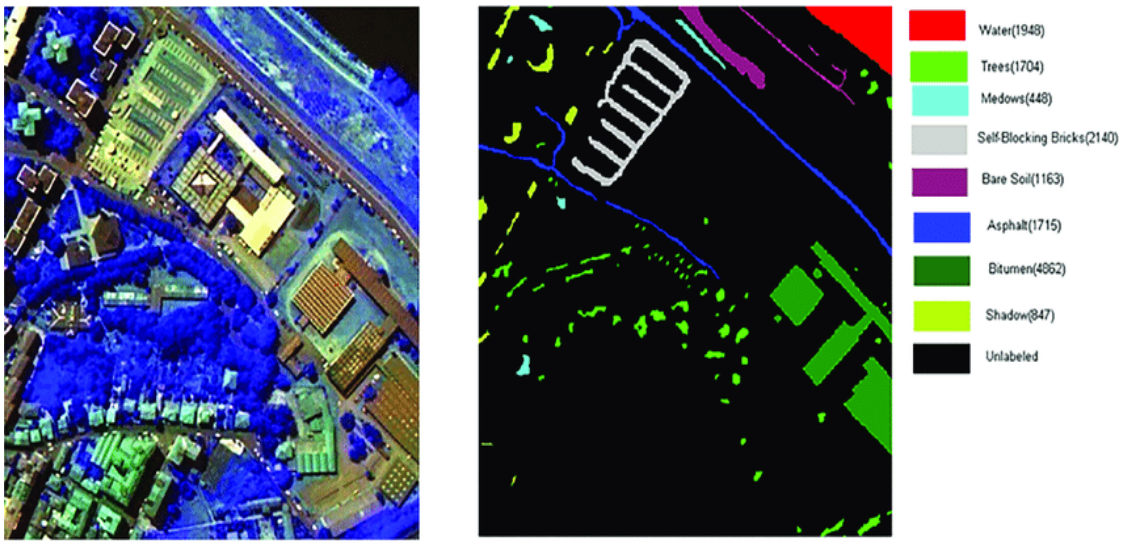


Figure 3. 1. Ground truth and false color images for a patch from Pavia Center Image (a: false color image, b: ground truth labels)

Source: (https://www.researchgate.net/figure/Part-of-pavia-centre-hyperspectral-image-a-The-HSI-in-false-color-RGB-3-65-101-b_fig1_327085101)

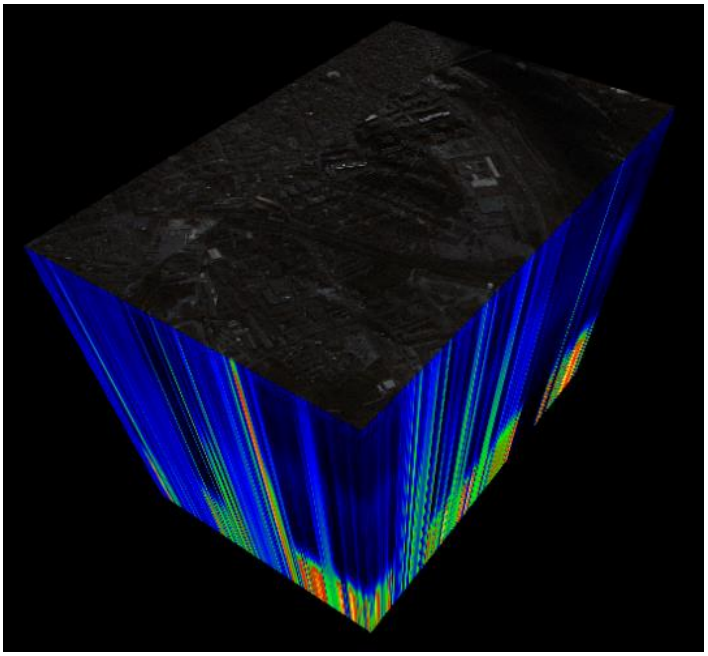


Figure 3. 2. Hyperspectral cube for Pavia Center Image

3.2.2. Pavia University Scene Image

The Pavia University dataset is closely related to the Pavia Center dataset as they were both collected during the same flight [54]. Compared with the Pavia center it has a smaller spatial dimension (610x340) but has one extra channel than the Pavia Center dataset. Both of them have the same number of classes and class descriptions given in Table 3.1.

The hyperspectral cube image and the ground truth image are given in Figure 3.4 and Figure 3.5 respectively. This data will be used to test the representation power of the dimensionality reduction model via reconstruction and will later be used to test classification performance of the models proposed. Figures 3.3 and 3.4 below show the ground truth labels, and the hyperspectral cube.[3]

[3] The PaviaC, PaviaU, Salinas, and KSC hyperspectral images can be downloaded from the following link

http://www.ehu.eus/ccwintco/index.php/Hyperspectral_Remote_Sensing_Scenes

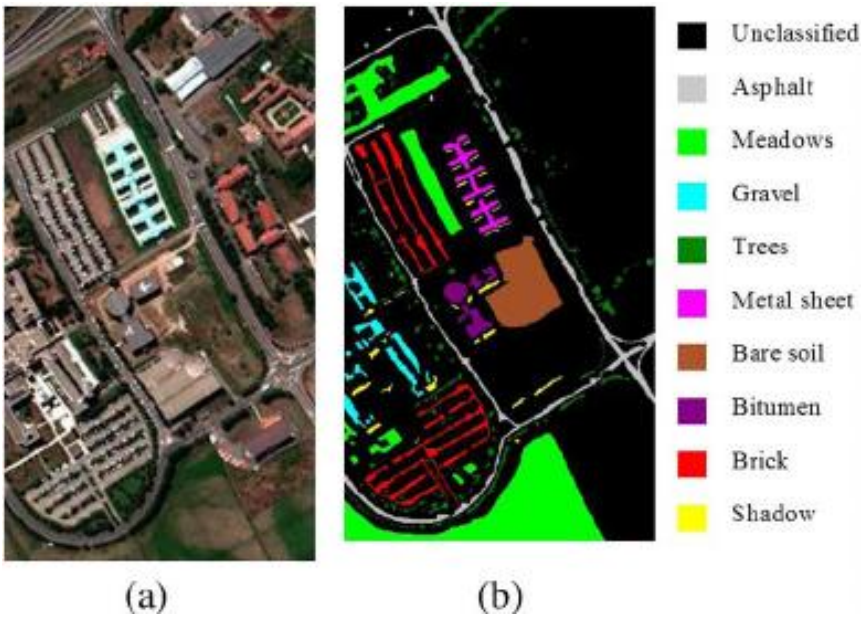


Figure 3. 3. Pavia University Scene false color and ground truth image

Source: (https://www.researchgate.net/figure/University-of-Pavia-dataset-a-Composite-image-of-hyperspectral-data-b-Ground-truth_fig3_272786930)

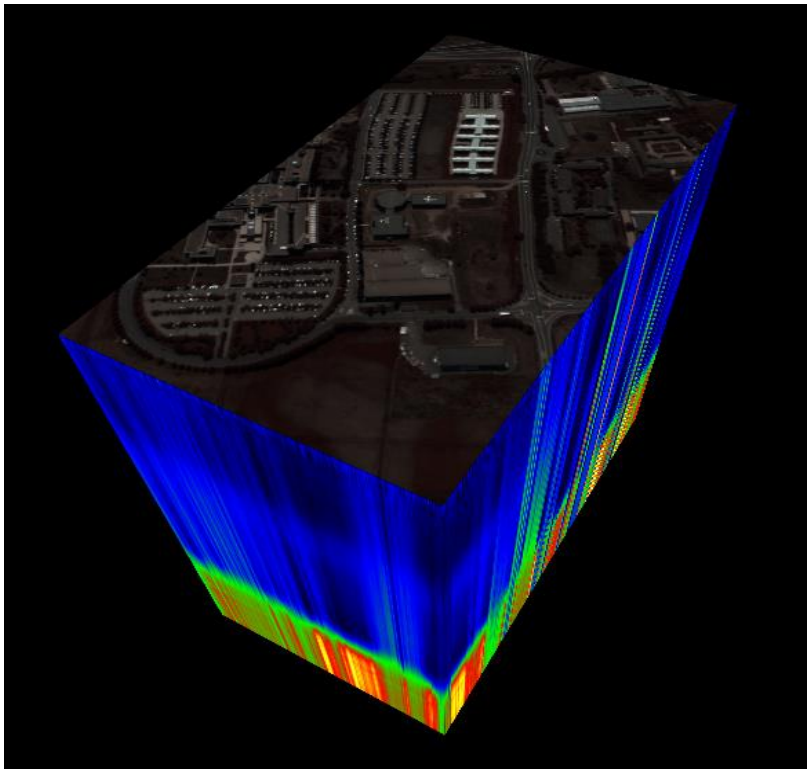


Figure 3. 4. Pavia University hyperspectral image cube

Table 3. 1. Pavia center and center ground truth classes count and

Class number	Class name	Number of samples
1	Water	824
2	Trees	820
3	Asphalt	816
4	Self-Blocking Bricks	808
5	Bitumen	808
6	Tiles	1260
7	Shadows	476
8	Meadows	824
9	Bare Soil	820

Source:(http://www.ehu.eus/ccwintco/index.php/Hyperspectral_Remote_Sensing_Scenes#Pavia_Centre_and_University)

3.2.3. Salinas Valley Scene Hyperspectral Image

A 224-band hyperspectral sensor called AVIRIS was used to collect this image taken in Salinas Valley, California [54]. The scene consists of 16 class plus one extra unclassified class labelled as zero. The surface reflectance values are available over a grid of 517x217 pixels. The class labels, the hyperspectral cube, and the ground truth images are given in Figure 3.5, Figure 3.6 and Table 3.4. This dataset is used for testing the representation power of the dimensionality reduction models and for testing classification performance.

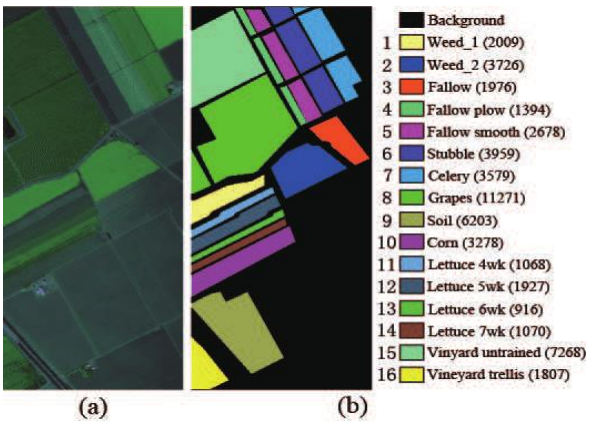


Figure 3. 5. Salinas Scene hyperspectral image false color image and ground truth labels

Source: (https://www.researchgate.net/figure/Salinas-hyperspectral-image-a-HSI-in-false-color-b-Ground-truth_fig4_318841472)

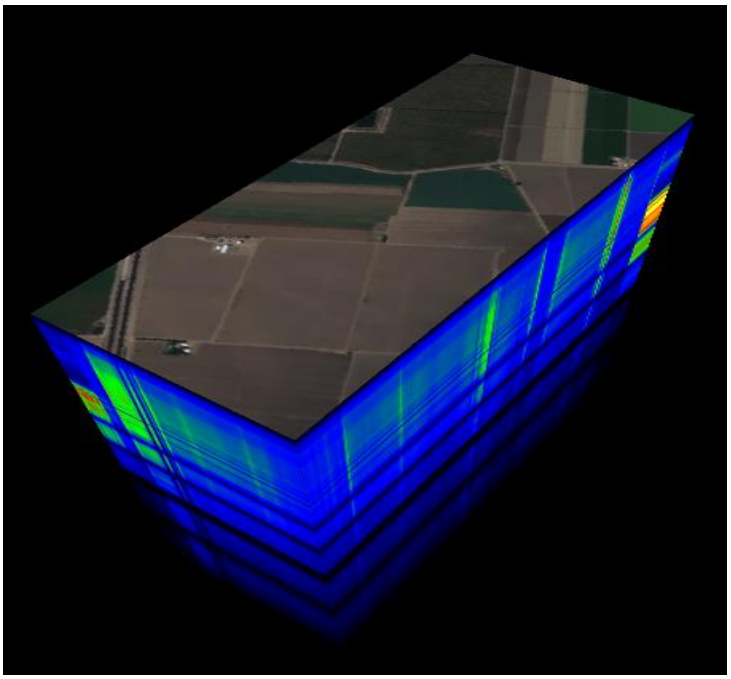


Figure 3. 6. Salinas scene hyperspectral cube

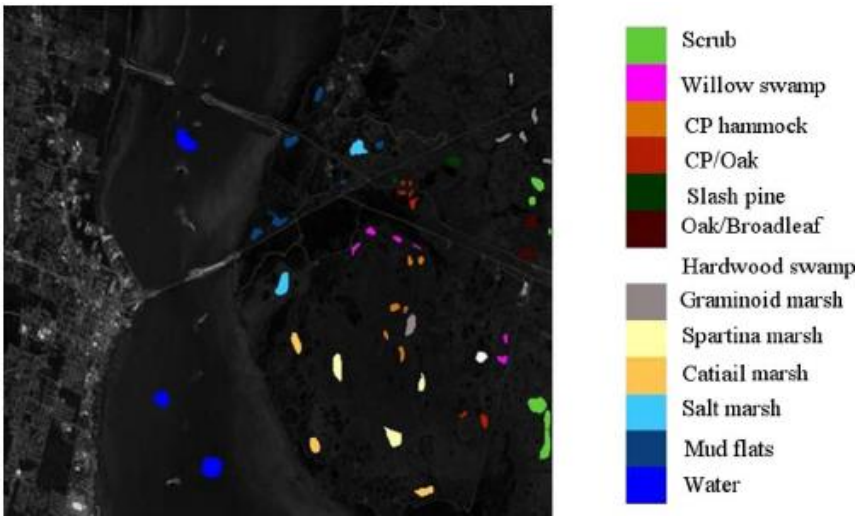


Figure 3. 7. Kennedy space center scene hyperspectral image ground truth image

Source:(https://www.researchgate.net/figure/NASA-data-KSC-Band-20-and-corresponding-ground-truth-areas-representing-13-land-cover_fig6_264564342)

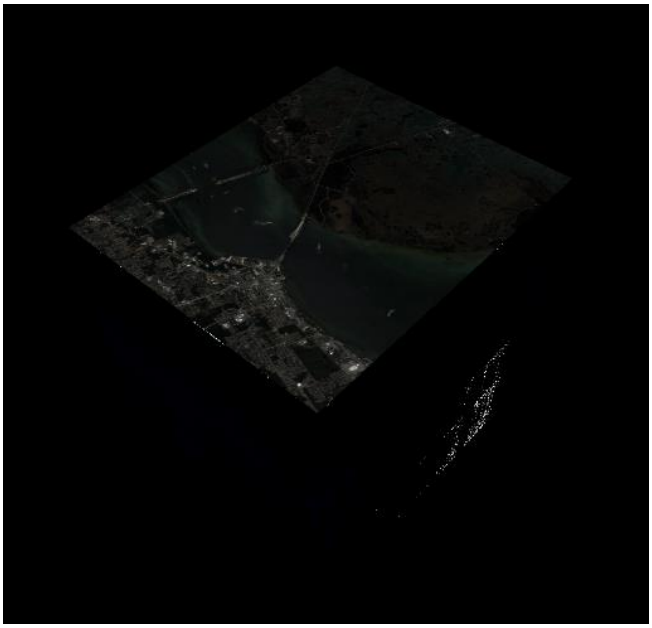


Figure 3. 8. Kennedy Space Center hyperspectral cube

Table 3. 2. Ground truth classes and frequency for Pavia University Scene

Class number	Class name	Number of samples
1	Asphalt	6631
2	Meadows	18649
3	Gravel	2099
4	Trees	3064
5	Painted metal sheets	1354
6	Bare soil	5029
7	Bitumen	1330
8	Self-blocking bricks	3682
9	Shadows	947

Source:(http://www.ehu.eus/ccwintco/index.php/Hyperspectral_Remote_Sensing_Scenes#Pavia_Centre_and_University)

3.2.4. Kennedy Space Center (KSC) Scene Hyperspectral Image

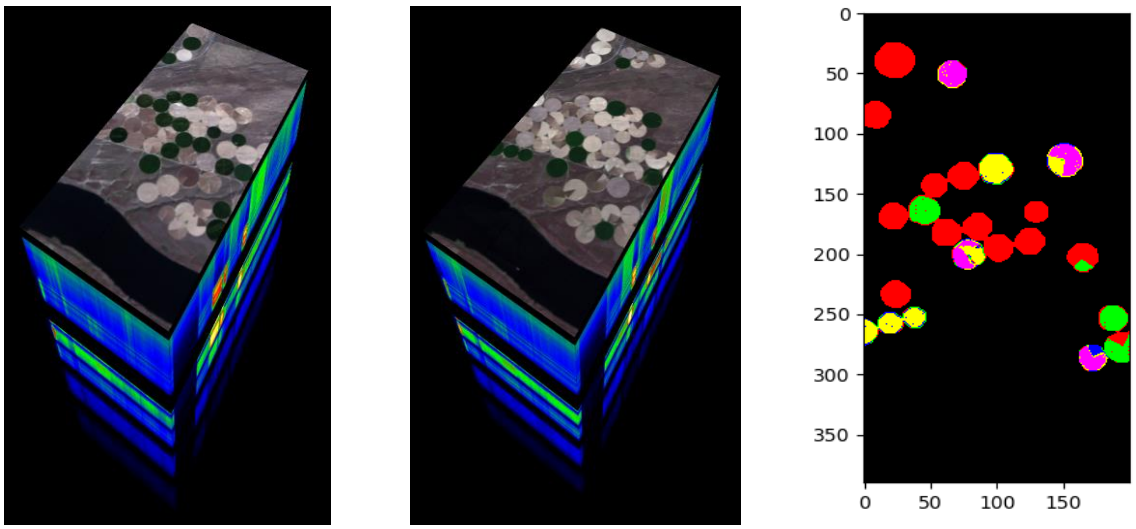
This image was captured over the Kennedy Space Center in the year 1996 using AVIRIS installed aerial vehicle [54]. Originally images captured by AVIRIS have 224 bands,

however, the KSC scene image band has been reduced to 176 bands after the removal of water absorption bands. Ground truth image, hyperspectral cube, and the available class labels are given in Figure 3.6, Figure 3.7 and Table 3.4 respectively. The hyperspectral cube's image is dark and hard to see due to the nature of the conditions when the image was taken.

3.2.5. Hermiston City Scene Hyperspectral Image

The Hermiston city scene hyperspectral image was captured by the Hyperion Sensor from the EO-1 satellite [55]. It has 242 spectral channels each of which have a 390x200 dimension monochromatic images. The bitemporal image represents crop transition of different classes in Hermiston city where center pivot irrigation creates the circular patterns in the image due to water supply. Some crops have withered and appear to be yellow in color while others are newly grown and appear green.

The brown areas are harvested areas and the green areas are green crops. The bitemporal images of Hermiston City on the years 2004 and 2007 are given on Figure 3.9 a and Figure 3.9 b. Their respective ground truth change image is given in Figure 3.9 c. The table for the ground truth classes and their frequency is on Table 3.4.



a) b) c)

Figure 3. 9. Hyperspectral cubes and ground truth images for Hermiston City image

Table 3. 3. Ground truth classes and their frequency for the Salinas scene

Class number	Class name	Samples per each class
1	Broccoli green weeds 1	2009
2	Broccoli 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 vineyard develop	6203
10	Corn senesced green weeds	3278
11	Lettuce romaine 4wk	1068
12	Lettuce romaine 5wk	1924
13	Lettuce romaine 6wk	916
14	Lettuce romaine 7wk	1070
15	Vineyard untrained	7268
16	Vineyard vertical trellis	1807

Source:(http://www.ehu.eus/ccwintco/index.php/Hyperspectral_Remote_Sensing_Scenes#Pavia_Centre_and_University)

Table 3. 4. Ground truth classes for Hermiston city images

Class of change	Number of samples
Type 1	5558
Type 2	1331
Type 3	79
Type 4	1557
Type 5	1461

Source:(https://gitlab.citius.usc.es/hiperespectral/ChangeDetectionDataset/-/tree/master/Hermiston)

Table 3. 5. Ground truth classes and their frequency for KSC scene

Class number	Class name	Samples per each class
1	Scrub	761
2	Willow swamp	243
3	Cabbage palm hammock	256
4	Cabbage palm / oak hammock	252
5	Slash pine	161
6	Oak / broadleaf hammock	229
7	Hardwood swamp	105
8	Graminoid marsh	431
9	Spartina marsh	520
10	Cattail marsh	404
11	Salt marsh	419
12	Mud flats	503
13	Water	927

Source:(http://www.ehu.eus/ccwintco/index.php/Hyperspectral_Remote_Sensing_Scenes#Pavia_Centre_and_University)

3.3. Research Tools

3.3.1. Hardware Tools

The Table 3.6 shows hardware were used to design, implement and test the proposed model for its intended task.

3.3.2. Software Tools

Software tools, programming tools, and library used in the research are listed below.

Tensorflow: is an open-source low-level API created for performing tensor computation. It also can be deployed on desktop computers, clusters, mobile, and edge devices. Google Brain team (a group of researchers and engineers) is the one that developed this library. It

supports machine learning, deep learning, and flexible numerical computation. TensorFlow 2.3.1 was used for this research.

Table 3. 6. Hardware tools used to do the research

Name	Type	Description
NVIDIA GPUs	Acceleration hardware	Used to train the model for performance enhancement
Hard disk	Long term storage	Storing datasets used train the model and save model progress
RAM Upgrade	Memory enhancement	Enhance hardware performance because most computers have small RAM size

Keras: is a high-level API that mainly runs on top of TensorFlow backend but also supports PyTorch and Theano backends. Considering the backend support it runs flawlessly on CPU and GPU. Keras is the 2nd most popular and used deep learning framework currently. For this research Keras version 2.3.0 was used.

Anaconda: It is an application used to install the Python programming language and all of its modules depending on the python edition – anaconda serves primarily as an environment manger to handle the necessary dependencies and avoid package conflict. It provides a navigator application to view different settings like Jupiter notebook, spider, vs-code, and modules installed in the environment. For the research Anaconda version 3-2020.11 was used.

Jupyter notebook: An interactive web-based application that helps configure, load python API, and write python code.

Python: is the primary programming language to write the programs and the packages used in this research.

3.4. Description of the Proposed Method

3.4.1. BCE Hyperspectral Dimensionality Reduction

The numerous literatures surveyed in the previous chapter show that most techniques do not effectively combine spatial-spectral representation for dimensionality reduction. This requires a combined dimensionality reduction architecture that integrates both the spatial-spectral features. Branching Convolutional Encoder aims to alleviate this problem by designing to separate convolutional encoder to represent the spatial and spectral information from the original hyperspectral cube. The spectral encoder consists of a point wise convolution that performs 1D convolutions with the pixel vectors which correspond to the values present at a single pixel slice having hundreds of spectral bands.

The 1D convolution operation convolves a kernel *k* having dimensions (1xB), where B is the number of channels/bands. This kernel is convolved with each and every pixel vector (slice) of the hyperspectral cube give a 2D output of shape (*M* x *N*) where *M* and *N* the number of row and column pixel count. Conv2D layer of the Keras library initializes numerous kernels of different initializations where after each 1D kernel has been convolved with the input the output shape becomes (*M, N, K*). K adjusts the output dimension of the convolution without affecting M and N.

Moving on to the spatial encoder section, 2D convolution is performed to capture spatial information and reduce the size of the output whose depth is determined by the number of kernels present. One convolution operation with a single kernel amongst the total count K' gives output shape specified by the number of strides, zero padding and kernel size. Differing kernel size are used to effectively capture the spatial features present on different neighborhood size. Stacking up these convolutions also stacks up the feature representations where at a certain point the output of the last layers contains some information about each and every pixel vector before it. This enhances spatial feature representation on a integrated 2D level.

The third step is fusing the spatial and spectral representation via a concatenation step. The concatenated output contains both the spectral and spatial features without any significant omission. This concatenated spatio-spectral representation goes to the decoder which

consists of deconvolutions (transposed convolution) and up sampling steps that attempt to reconstruct the original dimensions input the encoder network. The spatial dimensions are controlled by the kernel size, stride, and zero paddings, while the depth is maintained by the number of pixels in each ConvTranspose2D layer. Using 1D and 2D convolution will reduce the number of operations done in 3D densely connected models. Training of the encoder-decoder architecture is controlled by a loss function that computes error between the original and the reconstructed output.

$$\begin{bmatrix} X_{11} & X_{12} & X_{13} & \dots & X_{1N} \\ X_{21} & X_{22} & X_{23} & \dots & X_{2N} \\ X_{31} & X_{32} & X_{33} & \dots & X_{3N} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ X_{M1} & X_{M2} & X_{M3} & \cdots & X_{MN} \end{bmatrix}_{M \times N}$$

Unfolding of the matrix above reveals the vectorized representation and the pixel elements on the B bands available in the image. Equation (3.1) gives the flattened representation of the input where each and every element in the above matrix is expanded to reveal the corresponding pixel elements.

$$\begin{Bmatrix} X_{11} = \begin{bmatrix} X_{11}^{1} & X_{11}^{2} & X_{11}^{3} & \cdots & X_{11}^{B} \end{bmatrix}_{1 \times B} \\ X_{12} = \begin{bmatrix} X_{12}^{1} & X_{12}^{2} & X_{12}^{3} & \cdots & X_{12}^{B} \end{bmatrix}_{1 \times B} \\ \vdots \\ X_{MN} = \begin{bmatrix} X_{MN}^{1} & X_{MN}^{2} & X_{MN}^{3} & \cdots & X_{MN}^{B} \end{bmatrix}_{1 \times B} \end{Bmatrix} \tag{3.1}$$

Pointwise convolution between the elements and kernel gives the out where each point in the matrix is the convolution between the kernel and the pixel vectors at the corresponding points. The kernel width has to be set equal to the number of the bands for the first step so that the output of the convolution becomes a single matrix as shown in Equation (3.2).

$$K_1 = \begin{bmatrix} K_1^{1} & K_1^{2} & K_1^{3} & \cdots & K_1^{B} \end{bmatrix}_{(1 \times B)} \tag{3.2}$$

Equation 3.4 demonstrates the convolution between the kernel vectors and the pixel vectors.

$$K_1 * X = \begin{bmatrix} k_1 * X_{11}^T \\ k_1 * X_{12}^T \\ \vdots \\ k_1 * X_{MN}^T \end{bmatrix} \tag{3.3}$$

Where, the convolution between the several kernels and the input becomes:

$$K_1 * X_{11}^T = \left[k_1^1 \times X_{11}^T + k_1^2 \times X_{11}^2 + \cdots + k_1^B \times X_{11}^B \right] \times \frac{1}{B} \tag{3.4}$$

$$C_1 = \begin{bmatrix} k_1 * X_{11} & k_1 * X_{12} & k_1 * X_{13} & \ldots & k_1 * X_{1N} \\ k_1 * X_{21} & k_1 * X_{22} & k_1 * X_{23} & \ldots & k_1 * X_{2N} \\ k_1 * X_{31} & k_1 * X_{32} & k_1 * X_{33} & \ldots & k_1 * X_{3N} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ k_1 * X_{M1} & k_1 * X_{M2} & k_1 * X_{M3} & \cdots & k_1 * X_{MN} \end{bmatrix} \tag{3.5}$$

The final depth of the convolution step is determined by the number of kernels specified which successively drops in equal number of steps. Therefore, the output has the same shape as the input show in Equation (3.5) but different depth. Later the convolution outputs will be multiplied by weights and added with the biases specified.

3.4.2. Densely Connected Classification Network

The conventional setup of classifiers used in image classification networks has been used to design the classifier scheme. Once the dimensionality reduction model has been trained on the available data it will be integrated with the densely connected layers that consist of SoftMax layer that outputs class probabilities on the given it input classifying it to the number of classes available for that dataset. Cross categorical entropy has been used to train this model and update the weights of the network. This model will be used to test performance of BCE and the comparative dimensionality reduction techniques (Densely connected network and a 2D convolutional autoencoder). The diagrammatic representation of the architecture of this network is given in the next chapter.

3.4.3. Convolutional LSTM Change Detection Network

A convolutional LSTM change detection architecture has been adapted from the work by Ahram Song et. al [21]. This architecture uses a time distributed layer that operates on two slices of the bitemporal images from the same location. A Conv2D layer is fed to the time distributed layer which extracts features from the respective two images then the time distributed layer warps the layer. The applied result will be fed in to a ConvLSTM layer that extracts multitemporal information from the inputs. This will later be concatenated with a densely connected layer equal to the number of classes which learns to classify the output of the ConvLSTM layer to the number of changed classes present in the data. The architecture of this model is given in the next chapter.

3.5. Performance Evaluation Metrics

The following performance metrics were used to test the performance of the proposed dimensionality reduction technique with respect to reconstruction, classification and change detection tasks.

3.5.1. L1-loss Similarity Measure

The L1 loss (Mean Absolute Error) is computed to measure the power of reconstruction of the dimensionality reduction modules as shown in Equation (3.6). For the tree dimensionality reduction models, the L1 loss is computed for the datasets available to show which technique has the most representation power. Where $y^{(i)}$ is original input at the dimensionality reduction model and $\widetilde{y^{(i)}}$ is the reconstructed output, n is the number of non-overlapping patches.

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |y^{(i)} - \widetilde{y^{(i)}}| \tag{3.6}$$

3.5.2. L2-loss Similarity Measure

The L2 loss (Mean Squared Error) is computed to measure the power of reconstruction of the dimensionality reduction modules by measure the distance – or how far are the original and the reconstructed images. For the tree dimensionality reduction models, the L2 loss is

computed for the datasets available to show which technique has the most representation power. MSE is given in Equation (3.7)

$$MSE = \frac{1}{n}\sum_{n=1}^{n}(Y_i - Y_i)^2 \tag{3.7}$$

3.5.3. Confusion Matrix

Confusion matrix is widely used to measure the performance of classification models and change detection models with supervised training approach. For the four datasets (PaviaC, PaviaU, KSC, Salinas) used for classification the performance of the various dimensionality reduction techniques will be tested for their ability to correctly classify the various pixel in the datasets. The confusion matrix will also be plotted for the change detection, because there are change detection ground truth data available for the datasets (Hermiston Area, River, Bay Area) used.

3.5.4. Overall Accuracy

Overall accuracy of the two dimensionality reduction techniques including BCE, was tested in classification and change detection tasks. The results are listed and provided in the upcoming chapter in a tabular manner for ease of comparison. Overall accuracy is computed after plotting the confusion matrix as shown in Equation (3.8). X_{ii} represents every element in the diagonal of the confusion matrix. X_{ij} represents every element found within the confusion matrix. C defines the boundary of the confusion matrix which is equal to the class number.

$$P_{OA} = \frac{\sum_{i=1}^{c} x_{ii}}{\sum_{i=1}^{c}\sum_{j=1}^{c} x_{ij}} \tag{3.8}$$

3.5.5. Precision

Precision of the classification and change detection techniques for the three change detection models. The ratio shows the portion of samples correctly classified as instance of a class to

the total number of elements classified as the correct samples. Its formula is given on Equation (3.9) below.

$$Precision = \frac{TP}{TP + FP} \tag{3.9}$$

Where TP is true positive, FP is false positive, and FN is false negative.

3.5.6. Recall

Recall of the classification and change detection models is also used to measure the models' performance. The formula to calculate this is given on Equation (3.10).

$$Recall = \frac{TP}{TP + FN} \tag{3.10}$$

3.5.7. F1-score

F1 score is also used to measure the test performance of the classification and change detection models trained with the three dimensionality reduction models. The formula for the F1-score is given below on Equation (3.11).

$$F1 = \frac{2 \times precision \times recall}{precision + recall} \tag{3.11}$$

CHAPTER 4

PROPOSED DIMENSIONALITY REDUCTION TECHNIQUE

4.1. Introduction

In this chapter the proposed dimensionality reduction architecture along with the subsequent classification and change detection architectures are described in detail. First, the necessary data preprocessing steps taken to ease the training process are described. Then, the intermediate feature extractor-based dimensionality reduction step's architecture is elaborated. Also, the architectures of the classification model and the change detection model used to test the representation power of the dimensionality reduction module are included.

4.2. Preprocessing and Augmentation

4.2.1. Hyperspectral Image Pre-processing

Hyperspectral pixels contain reflectance values which are not bounded within the regular RGB image values (0 – 255). Therefore, the available data was normalized using min-max normalization to make the model learning bounded and avoid exploding gradients during training – min-max normalization is the common technique used before training a model on hyperspectral images. These images contain reflectance values and the values can range up to 9000. Neural networks trained on such large numbers have a gradient explosion problem. So, the minimum value will be set as zero, and the maximum value set to 1. The formula for the min-max normalization is given below.

$$normalized = \frac{value - min}{max - min} \tag{4.1}$$

Where, min and max are the minimum and maximum value in the hyperspectral cube, and value is the reflectance value at each and every pixel in the entire hyperspectral image. Each and every pixel will be normalized with this approximation to normalized the data. Min-max processing has been selected for normalizing the datasets because it ensures the data

points are strictly between 0 and 1, despite the fact that it does not handle outliers. This prevents the problem of exploding gradients that are quite common when training neural network architectures.

4.2.2. Data Patching to Prepare Datasets

One of the challenges in designing generic frameworks that perform dimensionality reduction, classification and change detection for hyperspectral images is the shortage of annotated data that have a corresponding ground truth image label. Since there are a small number of hyperspectral images that have a ground truth label, augmenting the existing data is important to train the models. So, the publicly available datasets that were used for this research were first patched to prepare a suitable data for the models. The sliding rate is adjusted according to the dimensions of the hyperspectral images to make the total amount of data in the dataset consistent and suited to the specific test used.

Table 4. 1. Data patching

Algorithm for patching hyperspectral data	
1.	Prepare a 64x64 window patcher
2.	For *patching window* within the boundary of the row:
3.	For *patching window* within the boundary of the column:
4.	Take 64x64 patches
5.	Save the patch in storage
6.	Slide the window right to take another patch
7.	Slide the window downwards 5 steps

4.3. Proposed Dimensionality Reduction Architecture

The dimensionality reduction model's scheme is inspired by the architecture of conventional stacked autoencoders [14], [56]. However, it has branching scheme that consists of two separate encoders merging into one to form a good representation of the original hyperspectral cube. Branching encoders in the feature extractor are trained with a corresponding decoder module that attempts to reconstruct the original hyperspectral cube. The entire process of learning model given in Figure 4.1 will eventually give an output that closely resembles the original input. In this process the branching encoders will also have

learnt to give a good latent representation of the original dataset so that the representation can be used for other applications.

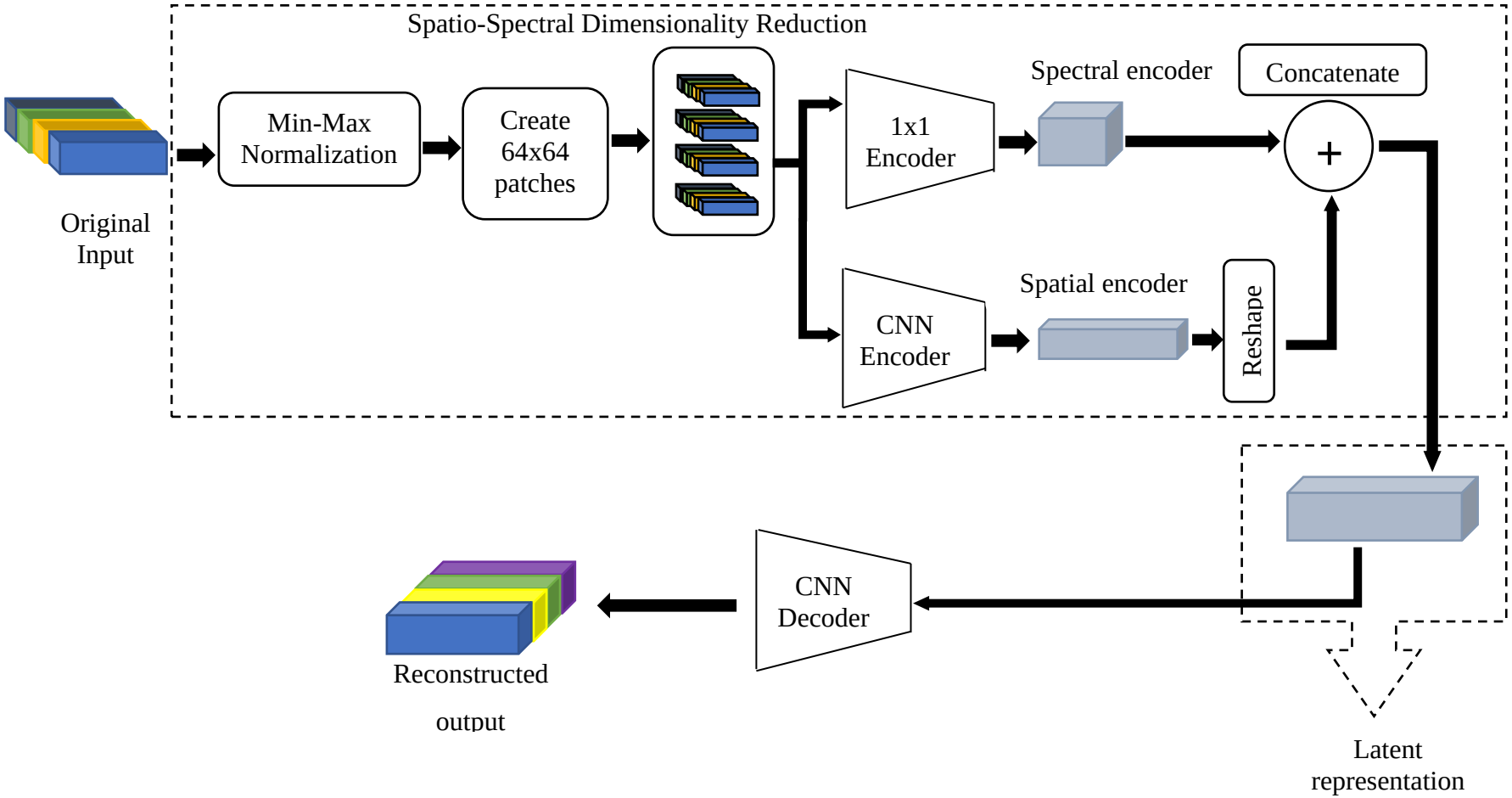


Figure 4. 1. Overall representation of the dimensionality reduction module training process

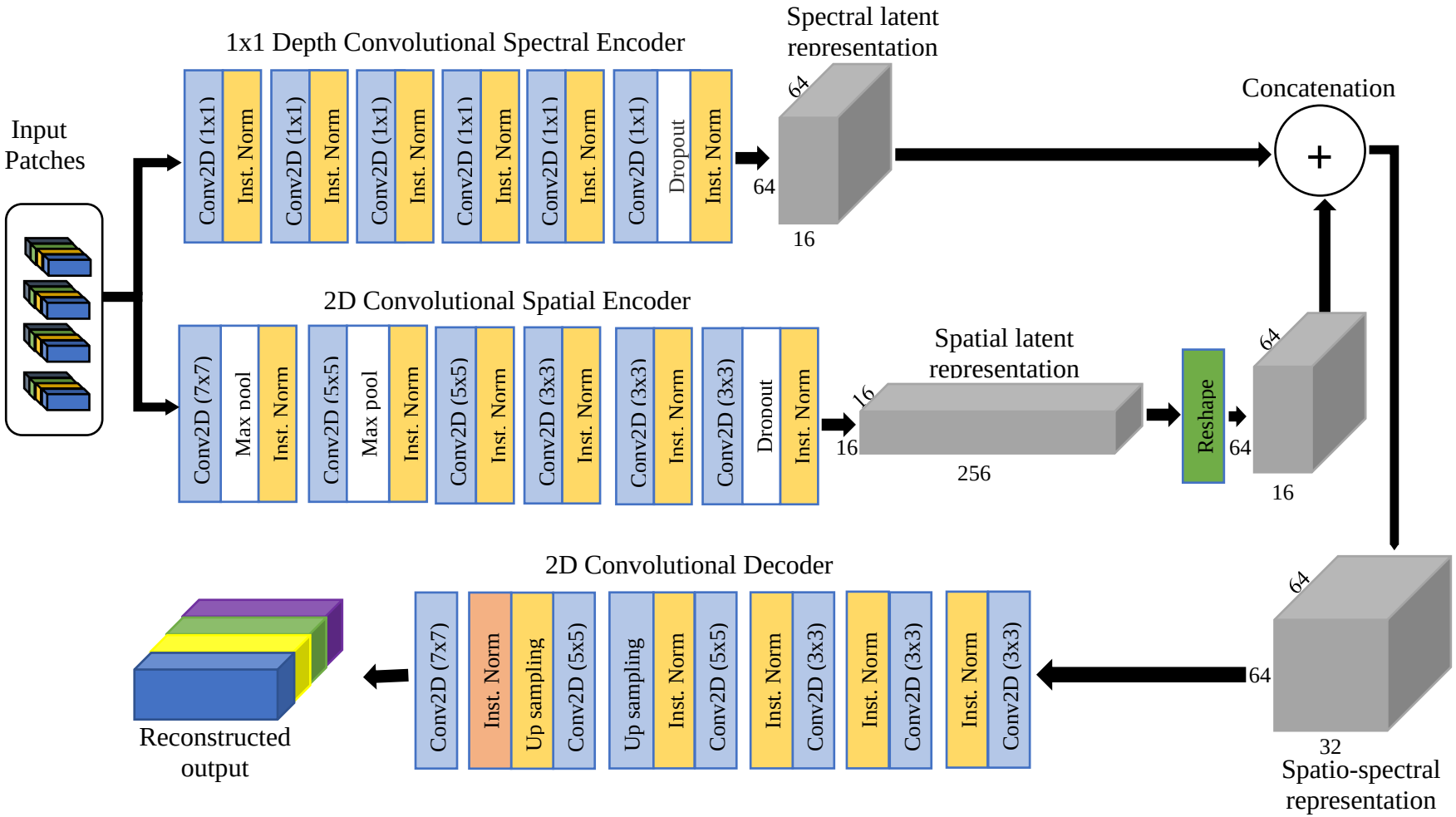


Figure 4. 2. Implementation-level block structure of the dimensionality reduction module

4.3.1. Spectral Encoder

The architecture of this encoder takes inspiration from pointwise separable convolutions, where kernel size is (1 x 1 x number_of_channels). This is clearly useful in reducing the number of channels available in the original hyperspectral images. Output depth of each Conv2D layer with these pointwise convolutions is controlled by the number of filters specified. The pointwise convolution representation creates a smaller channel image while persevering the original dimensions. It also considers reflectance relationship among the channels to create an effective representation in the output [57].

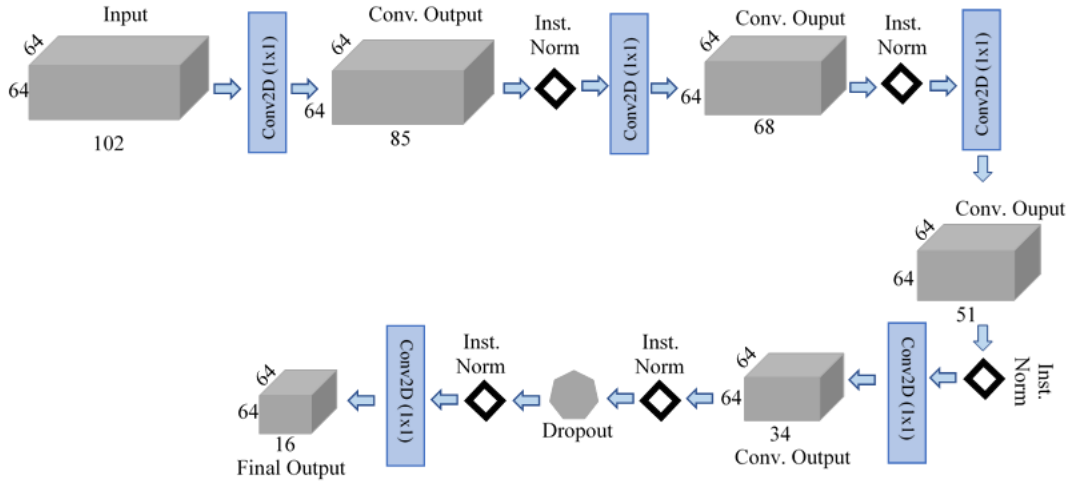


Figure 4. 3. Hyperspectral cube data flow for the spectral encoder

Table 4. 2. Pointwise convolution operation specification for the spectral encoder

Convolution Operation	Filter size	Number of Filters	Padding Size	Stride Size	Input Size	Output Size
Conv2D	(1x1x102)	85	Zero Pad	(1, 1)	64x64x102	64x64x85
Conv2D	(1x1x85)	68	Zero Pad	(1, 1)	64x64x85	64x64x68
Conv2D	(1x1x68)	51	Zero Pad	(1, 1)	64x64x68	64x64x51
Conv2D	(1x1x51)	34	Zero Pad	(1, 1)	64x64x51	64x64x34
Conv2D	(1x1x34)	16	Zero Pad	(1, 1)	64x64x34	64x64x16

Table 4.2 summarizes the details of the pointwise convolution based spectral encoder. The above table specification is for the Pavia Center dataset. Each convolution filter has the same

size as the input it is being fed. To adjust the depth of the output of the convolution filter, the number of filters is increased or decreased on each convolution layer.

4.3.2. Spatial Encoder

A conventional 2D convolution is used for the spatial encoder to capture spatial relationship among the pixels in each and every band. To enhance the feature extraction process, a blend of kernels with varying size are used for this operation. The spatial encoder representation is combined/concatenated with the spectral encoders' output giving a combined spatial-spectral latent code.

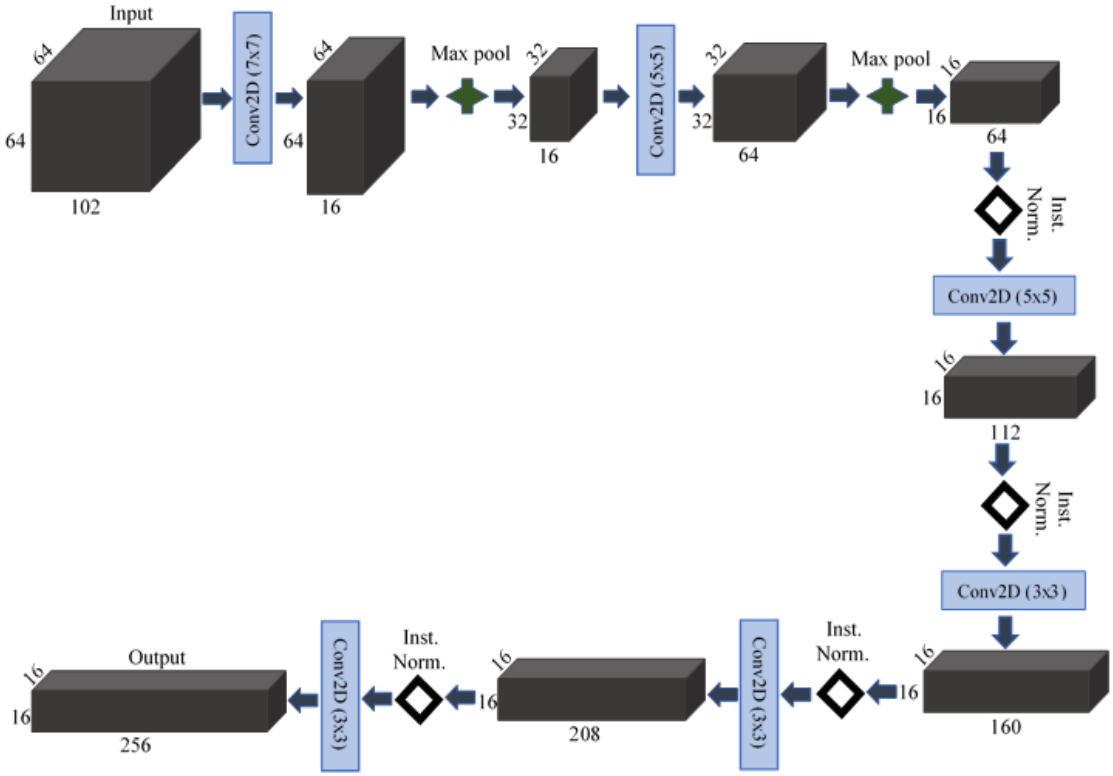


Figure 4. 4. Hyperspectral cube data flow for the spatial encoder

4.3.3. Spatial-spectral Decoder

After the outputs from the branching encoders have been concatenated together, they undergo a reshape and an intermediary convolution step before entering the decoder module. The decoder module consists of stacked convolution layers of varying kernel size followed Once the latent representation enters the decoder section its spatial dimensions are expanded

by up sampling layers to increase the size of the latent representation so that is matches the original input size for comparison. Intermediate instance normalization layers are incorporated to normalize the outputs of the activation layers of the convolution steps to normalize the output of each convolution layer. The final layer consists of a *Tanh* activation function because it has shown great success in generative adversarial networks which are widely used in content generation.

Table 4. 3. Spatial convolution specification table

Operation	Filter size	Number of Filters	Padding Size	Stride Size	Input Size	Output Size
Conv2D	(7x7x102)	16	Same	(1, 1)	64x64x102	64x64x16
MaxPool2D	(2x2)	-	Same	(1, 1)	64x64x16	32x32x16
Conv2D	(5x5x16)	64	Same	(1, 1)	32x32x16	32x32x64
MaxPool2D	(2x2)	-	Same	(1, 1)	32x32x64	16x16x64
Conv2D	(5x5x64)	112	Same	(1, 1)	16x16x64	16x16x112
Conv2D	(3x3x112)	160	Same	(1, 1)	16x16x112	16x16x160
Conv2D	(3x3x160)	208	Same	(1, 1)	16x16x160	16x16x208
Conv2D	(3x3x208)	256	Same	(1, 1)	16x16x208	16x16x256

4.4. Classification Model for Testing BCE's Representation

Several papers in the remote sensing literature pool use a classification model to test the performance of a given dimensionality reduction scheme, whether it be feature selection-based or feature extraction-based. So, a conventional classification scheme made up of densely connected layers followed by a final SoftMax layer is integrated with the dimensionality reduction module to demonstrate how the deep architecture of the DR module enhances feature separability. The number of layers in the last/output layer are adjusted according the classes present in the dataset used to train the classification model. The output of this scheme is a classification map that gives a multiclass classification of the input patches. The model is trained in a supervised manner by using ground truth classification map patches.

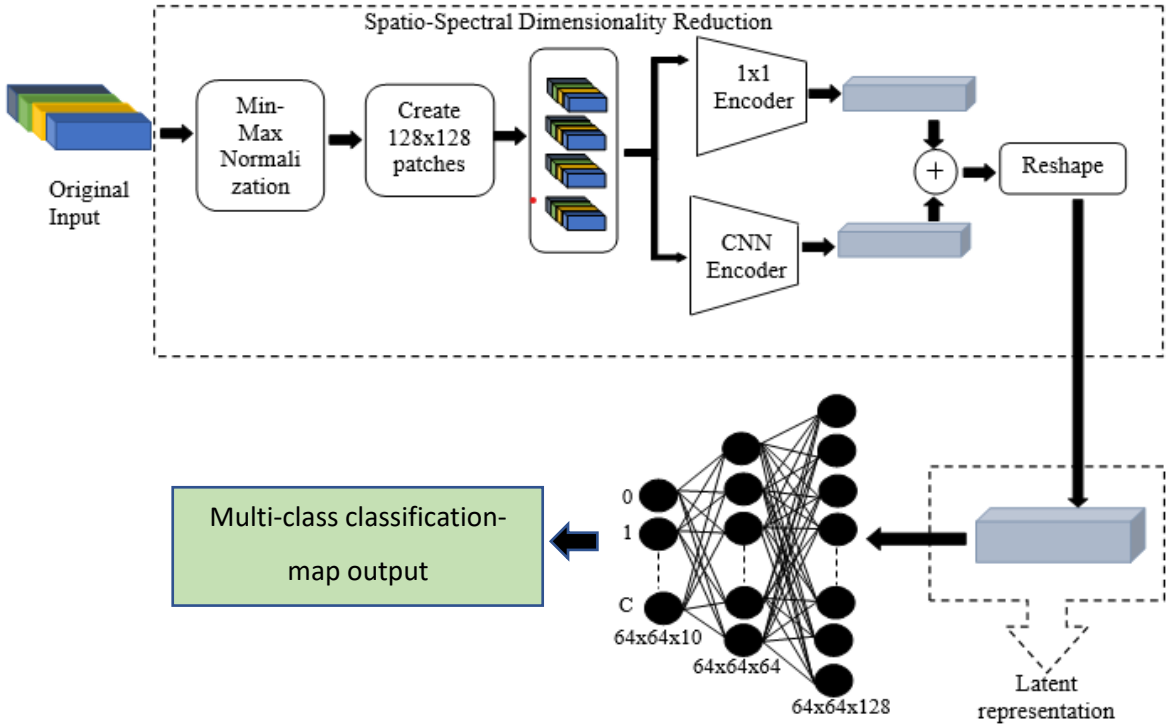


Figure 4. 5. Integrated diagram of the dimensionality reduction model and the classification network

4.5. Adapted Convolutional-LSTM Change Detection Architecture

The convolutional LSTM architecture for determining changes between two multitemporal hyperspectral images. This architecture is trained using paired patches from two multitemporal images and the resulting output is fed in to a densely connected multiclass classification network to plot the multiclass change detection map that not only gives the changed areas but also the classes to which each changed pixel belongs to. A conventional ConvLSTM block is used here extract temporal information between the bitemporal images. The cascaded ConvLSTM layers extract information from a time distributed that simultaneously applies a 2D convolutional operation and extracts features which might give information about changes in the bitemporal images. Several components make up the ConvLSTM block such as sigmoid activation represented by σ, tanh activation block, h-blocks that represent hidden layer input from previous slices, forget gates represented by f, pointwise addition and multiplication denoted by circled plus and asterisk sign, etc.

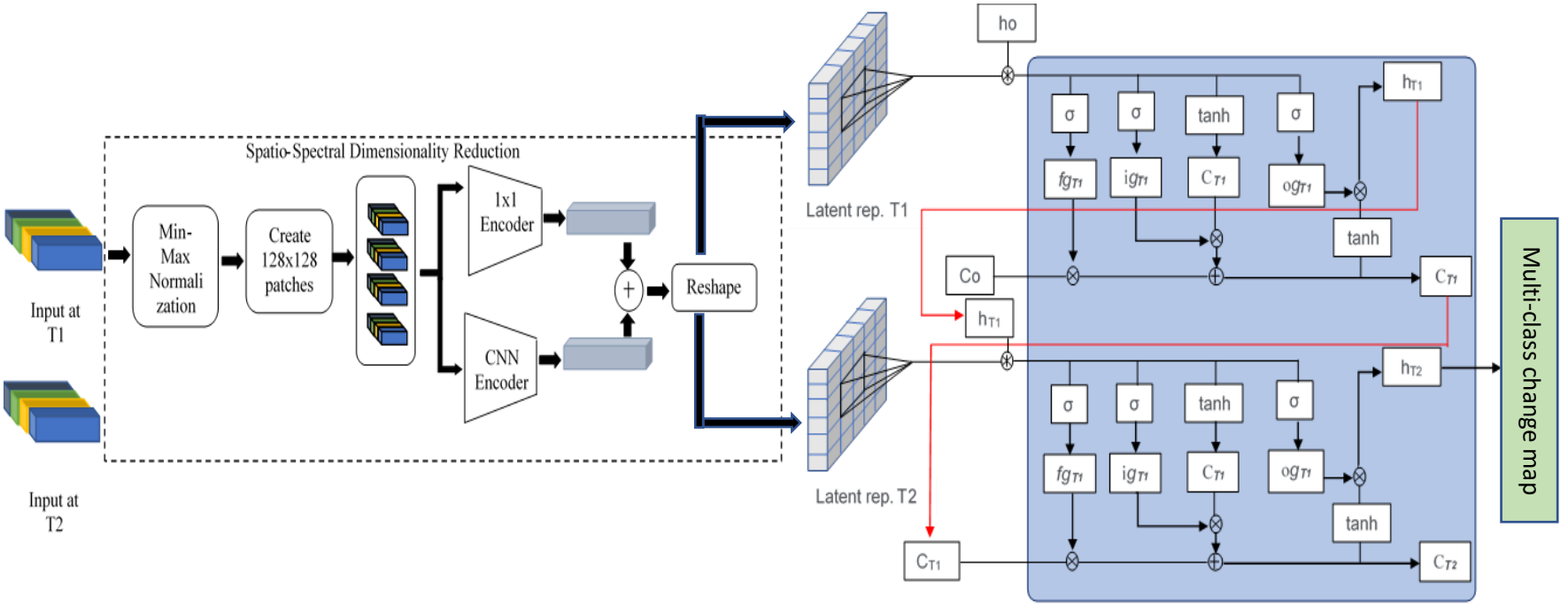


Figure 4. 6. ConvLSTM-based change detection architecture with BCE

The architecture given above is integrated with the dimensionality reduction module given above and trained in an end-to-end manner to generate the multiclass change detection map for each patch. Two images from the same spatial slice will be fed in to the dimensionality reduction model and fed to the change detection model, so that change detection model learns from the feature representations of the change detection model. Finally, the densely connected layer at the end of the ConvLSTM model outputs a feature classification map that contains pixels where changes took place and what kind of images took place in those pixels.

4.6. Training Procedures for the Proposed Models

4.6.1. Training Procedure for the Dimensionality Reduction Model

The procedure on Table 4.2 was used to train the dimensionality reduction model in the same manner with which most autoencoder networks are trained. The entire configuration and training process is aimed at reducing the difference between the original and the reconstructed image.

4.6.2. Training Procedure for the Classification Model

This step is done after the dimensionality reduction model has been trained on the datasets available. The classification model is then trained in an end-to-end manner with the dimensionality reduction model to improve the accuracy of the classification model. Here both the loss computed between the class map represented categorically, and the accuracy of classification are used together to monitor the model's progress. The densely connected layers eventually learn to create a good classification map in a supervised manner.

4.6.3. Training Procedure for the Change Detection Model

Once the dimensionality reduction model has been trained on the datasets it is integrated with the change detection model for a second round of training together with the CD model. The change detection model is then trained while receiving patches whose dimensions have reduced by the dimensionality reduction model. Afterwards, the loss is computed between the multiclass change detection map produced by the change detection model and the ground truth map. The gradient computed in this process is used to update the network's weights.

Table 4. 4. Training procedure for training the dimensionality reduction modules

Algorithm for training the dimensionality reduction model
***Input**: batches of original size hyperspectral images* ***Output**: reduced reconstructed output* 1. *Load batches of data from the dataset into memory* 2. *Give the two convolutional branches the same batch of input* 3. *The replicated batch passes through the branching encoders* *3.1. Spectral encoder forms a spectral feature representation (1D Convolution)* *3.2. Spatial encoder forms a spatial feature representation via 2D convolutions* 4. *The outputs from the branching encoders are merged together* 5. *The merged representation is fed in to a decoder network* 6. *The decoder attempts to reconstruct the original image via successive convolutions and up sampling* 7. *Mean absolute error is computed with the original and the reconstructed images* 8. *The optimizer (Adam) computes gradients and updates the network's weights until the desired loss level is reached*

Table 4. 5. Training procedure for training the classification model

Algorithm for training the classification model in an end-to-end manner
***Input**: batches of hyperspectral images and the corresponding label* ***Output**: reduced reconstructed output* 1. *Load batches (original input and ground truth image) of data from the dataset into memory* 2. *Load dimensionality reduction model's pretrained weights* 3. *Concatenate the dimensionality reduction model with the densely connected layers* 3. *Feed batches of data patches along with their corresponding labels to the network* 4. *Compute the loss and accuracy between the classification map of the model and the ground truth labels* 5. *Compute gradient and update the end-to-end model using Adam' optimizer* 6. *Repeat steps 3 to 5 until the desired level of accuracy is reached*

Table 4. 6. Training procedure for the change detection model

Algorithm for training the change detection model
***Input**: batches of paired bitemporal images with ground truth label* ***Output**: reduced reconstructed output* 1. *Form a training input dataset using batches of paired image patches taken from two bitemporal images* 2. *The label/output is also patched from the ground truth image in the same manner* 3. *Concatenate the dimensionality reduction model with the CD model* 3. *Feed batches of data patches along with their corresponding labels to the network* 4. *Compute the loss and accuracy between the change detection map of the model and the ground truth labels* 5. *Compute gradient and update the end-to-end model using Adam' optimizer* 6. *Repeat steps 3 to 5 until the desired level of accuracy is reached*

CHAPTER 5

IMPLEMENTATION DETAILS

5.1. Overview of the Implementation

Highlighted explanation and code snippets are given in this chapter to elaborate how the proposed models are implemented and trained to give the desired output. The detailed implementations are given in the appendix section to supplement the content provided in this chapter.

5.2. Working Environments

5.2.1. Hardware Specifications

There were two desktop computers that were used to train the dimensionality reduction, the classification model, and the change detection model side by side.

Desktop 1: used for implementing and training the dimensionality reduction model. The photo of this desktop is given in Figure 5.1.

- Model: DELL Optiplex 9020
- Operating system: Windows 10
- Processor: Intel ® Core™ i5-4580 CPU @ 3.29GHz x 4
- Graphics Card Upgrade: GeForce RTX 2070 Super 6 GB RAM
- Memory Size: 12.00 GB
- System Type: 64-bit Operating System, x64-based Processor

Desktop 2: used for implementing and training classification and change detection models. The photo of this desktop is given in Figure 5.2.

- Model: DELL Optiplex 3040
- Operating system: Windows 10
- Processor: Intel ® Core™ i5-4580 CPU @ 3.29GHz x 4

- Graphics Card Upgrade: GeForce RTX 2070 Super 6 GB RAM
- Memory Size: 8.00 GB
- System Type: 64-bit Operating System, x64-based Processor

5.2.2. Software Specifications

5.2.2.1. Integrated Development Environment and Virtual Environment

Table 5.1 gives a summarized list of development environments used when implementing the architecture.

Table 5. 1. Table for the list of development environments used for this research

Application Name	Type	Description
Anaconda	Virtual environment manager	Used to handle the package dependencies
Jupyter Notebook	Block-wise programming IDE	Eases the level of development and modularity
Google Colaboratory	Online Jupyter Notebook-based IDE	Provides several utilities for running codes including a library manager, GPU, TPU

5.2.2.2. Programming Language and Libraries Used

Python was used as a primary choice for the writing because of the thousands of packages available for scientific computing and the large community support for the associated libraries. Moreover, Keras serves as a deep learning architecture designing framework and eases the rate of development by minimizing the number of things to worry about and focus on the writing the code. TensorFlow seamlessly integrates with Keras and takes care of the numerous tensor operations taking place in the backend.

5.3. Training Procedure

5.3.1. Spatial-spectral Encoder-Decoder Module Implementation

The encoder decoder architecture consists of two encoder modules merging together to form a latent representation, where the first encoder is a spectral encoder, and the second encoder is a spatial encoder. The outputs of these two branches are merged/concatenated to form a spatial-spectral latent representation which is fed into a decoder module. Taking the latent representation, the decoder module performs successive transposed convolution and up-sampling operations to reconstruct the original output. Both of the encoder modules are fed with the input patches at the same time so that they are learning on from the same input sample batches to avoid mixing and delayed learning curve. The packages mentioned in Table 5.2 were used to write the implementation code.

Table 5. 2. Tabulated list of the packages and libraries used

Name	Type	Description
Python 3.x	Programming Language	Python version 3.6 was used as it was the one supported by the specific version of Anaconda installed
Keras 2.2.4	High level deep learning library	Eases the creating models with several layers and processing operations
Tensorflow 2.x	Low-level API to do tensor operations	Handles the low-level operations defined by Keras
Tensorboard 2.3.x	Visualization tool	Visualizations from training and Keras operations
Spectral Python 0.21	HSI visualization tool	Used to manipulate and visualize hyperspectral images

Each of the Conv2D layers in Figure 5.3 contains a LeakyReLU activation that has a slight negative value preventing zero values from disappearing after activation. Instance normalizations after each convolution are applied to normalize each and every instance of the dataset that passes through the output. This prevents the model outputs from being very

different from each other which later causes exploding gradients. Filter number parameter of the Conv2D specifies the output dimension of each convolution which in turn controls the depth (spectral) dimension of hyperspectral cubes. Moreover, this operation is made possible by the 1x1 kernel size which has a depth size equal to the channel number of each and every successive input to the convolution operations.

The code snippet for the spatial encoder in Figure 5.4 consists of several parameter that were tuned after several iterations to determine the optimal configuration. Starting with the kernel size, varying kernel size was used in the different level of convolution operations to capture different features of varying size. In addition, the various max pooling operations inserted after the convolutional layers reduce the size of the original image by selecting important spatial features through the filters applied. This stacking is widely followed by most convolutional architectures and has shown success in the making effective reconstruction [58].

For the final part, which is the convolutional decoder, it is trained together with the encoder module in an attempt to reconstruct the original input fed in to the autoencoder network. It is composed of transposed convolution operations followed by up sampling to regain the original shape of the input.

```
input_shape = (64, 64, 102)
theInput = Input(input_shape)

convnet = Conv2D(102, kernel_size=(1,1), padding="same", activation=LeakyReLU())(theInput)
convnet = InstanceNormalization(axis=3, center=True, scale=True, beta_initializer='random_uniform', gamma_initializer='random_uniform')(convnet)

convnet = Conv2D(85, kernel_size=(1,1), padding="same", activation=LeakyReLU())(convnet)
convnet = InstanceNormalization(axis=3, center=True, scale=True, beta_initializer='random_uniform', gamma_initializer='random_uniform')(convnet)

convnet = Conv2D(68, kernel_size=(1,1), padding="same", activation=LeakyReLU())(convnet)
convnet = InstanceNormalization(axis=3, center=True, scale=True, beta_initializer='random_uniform', gamma_initializer='random_uniform')(convnet)

convnet = Conv2D(51, kernel_size=(1,1), padding="same", activation=LeakyReLU())(convnet)
convnet = InstanceNormalization(axis=3, center=True, scale=True, beta_initializer='random_uniform', gamma_initializer='random_uniform')(convnet)

convnet = Conv2D(34, kernel_size=(1,1), padding="same", activation=LeakyReLU())(convnet)
convnet = InstanceNormalization(axis=3, center=True, scale=True, beta_initializer='random_uniform', gamma_initializer='random_uniform')(convnet)

convnet = Conv2D(16, kernel_size=(1,1), padding="same", activation=LeakyReLU())(convnet)
convnet = Dropout(0.3)(convnet)
convnet = InstanceNormalization(axis=3, center=True, scale=True, beta_initializer='random_uniform', gamma_initializer='random_uniform')(convnet)
```

Figure 5. 1. Code snippet for the 1D encoder implementation

```
conv2d_1 = Conv2D(filters = 16, kernel_size = (7, 7), padding='same', activation = LeakyReLU())(theInput)
max_1 = MaxPool2D(pool_size = (2, 2))(conv2d_1)
instance_1 = InstanceNormalization(axis=3, center=True, scale=True, beta_initializer='random_uniform', gamma_initializer='random_uniform')(max_1)
conv2d_2 = Conv2D(filters = 32, kernel_size = (5, 5), padding='same', activation = LeakyReLU())(instance_1)
max_2 = MaxPool2D(pool_size = (2, 2))(conv2d_2)
instance_2 = InstanceNormalization(axis=3, center=True, scale=True, beta_initializer='random_uniform', gamma_initializer='random_uniform')(max_2)
conv2d_3 = Conv2D(filters = 64, kernel_size = (5, 5), padding='same', activation = LeakyReLU())(instance_2)
instance_3 = InstanceNormalization(axis=3, center=True, scale=True, beta_initializer='random_uniform', gamma_initializer='random_uniform')(conv2d_3)
conv2d_4 = Conv2D(filters = 128, kernel_size = (3, 3), padding='same', activation = LeakyReLU())(instance_3)
instance_4 = InstanceNormalization(axis=3, center=True, scale=True, beta_initializer='random_uniform', gamma_initializer='random_uniform')(conv2d_4)
conv2d_5 = Conv2D(filters = 256, kernel_size = (3, 3), padding='same', activation = LeakyReLU())(instance_4)
instance_5 = InstanceNormalization(axis=3, center=True, scale=True, beta_initializer='random_uniform', gamma_initializer='random_uniform')(conv2d_5)
conv2d_6 = Conv2D(filters = 256, kernel_size = (3, 3), padding='same', activation = LeakyReLU())(instance_5)
drop = Dropout(0.5)(conv2d_6)
instance_6 = InstanceNormalization(axis=3, center=True, scale=True, beta_initializer='random_uniform', gamma_initializer='random_uniform')(drop)
```

Figure 5. 2. Code snippet for the 2D Encoder implementation

Instance normalization is included after each convolution layer to normalize each and every instance compensating for the lack of data. With the exception of the last layer, *LeakyReLU* is used as an activation function on each and every layer of the *Conv2DTranpose* operation. For the final layer *Tanh* activation used to form the final image as this activation type has shown great success amongst generative adversarial network variants in creating realistic outputs. The code snippet for the decoder network is given in Figure 5.5.

```
conv2d_1 = Conv2DTranspose(filters = 512, kernel_size = (3, 3), padding='same', activation = LeakyReLU())(concat_re)
instance_1 = InstanceNormalization(axis=3, center=True, scale=True, beta_initializer='random_uniform', gamma_initializer='random_uniform')(conv2d_1);
conv2d_2 = Conv2DTranspose(filters = 430, kernel_size = (3, 3), padding='same', activation = LeakyReLU())(instance_1)
instance_2 = InstanceNormalization(axis=3, center=True, scale=True, beta_initializer='random_uniform', gamma_initializer='random_uniform')(conv2d_2);
conv2d_3 = Conv2DTranspose(filters = 348, kernel_size = (3, 3), padding='same', activation = LeakyReLU())(instance_2)
instance_3 = InstanceNormalization(axis=3, center=True, scale=True, beta_initializer='random_uniform', gamma_initializer='random_uniform')(conv2d_3);
conv2d_4 = Conv2DTranspose(filters = 266, kernel_size = (5, 5), padding='same', activation = LeakyReLU())(instance_3)
up_1 = UpSampling2D(size = (2, 2))(conv2d_4);
instance_4 = InstanceNormalization(axis=3, center=True, scale=True, beta_initializer='random_uniform', gamma_initializer='random_uniform')(up_1);
conv2d_5 = Conv2DTranspose(filters = 184, kernel_size = (5, 5), padding='same', activation = LeakyReLU())(instance_4)
up_2 = UpSampling2D(size=(2, 2))(conv2d_5)
instance_5 = InstanceNormalization(axis=3, center=True, scale=True, beta_initializer='random_uniform', gamma_initializer='random_uniform')(up_2);
conv2d_6 = Conv2DTranspose(filters = 102, kernel_size = (7, 7), padding='same', activation = "tanh")(instance_5)
drop = Dropout(0.5)(conv2d_6)
instance_6 = InstanceNormalization(axis=3, center=True, scale=True, beta_initializer='random_uniform', gamma_initializer='random_uniform')(drop);
```

Figure 5. 3. Code snippet for the decoder architecture that reconstructs the original output

5.3.1.1. Batch Generator for Training the Autoencoder

Training deep neural networks requires lots of data which needs large memory space unaffordable by the current desktop standards. It is also an inefficient way of storing and manipulating data. For this reason, data should be loaded from secondary storage in batches and on a need basis. This eliminates the huge bulk of data that needs to be stored in the RAM that will inevitably crush the program. A data generator is designed for the purpose of loading batches of data from the dataset to train the model and validate its performance. Figure 5.6 shows that data loaded is tracked via a counter variable and the *yield* keyword at the end ensures that the sequence of data is returned with every next call. Once the batch of data is returned by the generator the batch is passed through the model and finally discarded until loaded by the generator again.

5.3.1.2. Model Training via the Keras' Fit Function

The final step is training the built model by fitting the batch generator by defining how many steps should data be fed per epoch, how many epochs should the model be trained, and defining any user defined callbacks which are to be called when the model is being trained. Fit function in the Figure 5.7 trains the model for 100 epochs visiting the entire training dataset which has been split in to training and testing dataset using a 75% to 25% ratio. Figure 5.8 depicts the code snippet to split the data into training and testing images which are used to validate on each and every step taken. The process mentioned above to train the dimensionality reduction module will be repeated for the different hyperspectral image available. Once the dimensionality reduction module has been trained on every data it is ready for integration with the classification and change detection models.

```
def myBatchGenerator(batch_size, images, labels):
    batch_counter = 0
    save_batch = 0
    while True:
        batch_image = []
        batch_label = []
        while batch_counter < batch_size + save_batch:
            #rand_ind = np.random.randint(len(images));#Random index number to access list

            image_name = images[batch_counter]
            theImage = np.load(image_name)
            label_name = labels[batch_counter]
            theLabel = np.load(label_name)
            #The input image and the batch label should be the same in this case

            batch_image.append(theImage)
            batch_label.append(theLabel)

            #print(theImage.shape)
            batch_counter = batch_counter + 1
            #print("batch_counter : "+str(batch_counter))
        #print("batch_size + save_batch : "+str(batch_size + save_batch))
        if(batch_counter == len(images)-1 or (batch_size + save_batch) > len(images)-1):
            batch_counter = 0
            save_batch = 0

        save_batch = batch_counter
        BI = np.array(batch_image)#.astype(np.float32);#Batch image
        BL = np.array(batch_label)#.astype(np.float32);#Batch labe
        yield BI, BL
```

Figure 5. 4. Batch generator for the dimensionality reduction model

```
np.random.seed(25)

train_images, test_images = train_test_split(all_images, test_size=0.25, shuffle=True)
```

Figure 5. 5. Training testing splitting function

```
history_v1 = ae_1d_2d.fit(train_batch,
                          steps_per_epoch=27,
                          epochs=100,
                          verbose=1,
                          validation_data=test_batch,
                          validation_steps=1,
                          callbacks = get_callbacks("updated")
                          )
```

Figure 5. 6. Dimensionality reduction model training function

5.3.2. Hyperspectral Image Classification Model Implementation

In order to demonstrate the representation power of the dimensionality reduction module, a customized classification densely connected architecture is implemented. For training the classification model first the pretrained dimensionality reduction was loaded. A new model

with layers starting from the input to DR model extending to the concatenation layer of the branching encoder will be extracted by slicing operation. This will later be concatenated with the densely connected layers that perform the classification operation. Loading of the pretrained model is given in the code snippet under Figure 5.9.

```
loaded_classi = tf.keras.models.load_model(r"E:\Classification-of-Hyperspectral-Image\classifier_pavC_V5_15_21_dnn.h5",
                                           custom_objects={"LeakyReLU":LeakyReLU,
                                                           "InstanceNormalization":InstanceNormalization})
```

Figure 5. 7. Code snippet to load the pretrained dimensionality encoder

With the aim of creating a classification model using the dimensionality reduction module, a densely connected layers followed by a SoftMax layer to convert input into one hot code representation which can easily be translated into categorical label. As with the convolutional encoder, LeakyReLU is used as an activation function. This process is implemented by the code given in 5.10 Later the loaded model and the densely connected are merged together to form the final model by the code given in Figure 5.11.

```
convnet = Dense(128, activation=LeakyReLU())(theInput)
convnet = Dropout(0.25)(convnet)
convnet = InstanceNormalization(axis=3, center=True, scale=True, beta_initializer='random_uniform', gamma_initializer='random_uniform')(convnet)

convnet = Dense(64, activation=LeakyReLU())(convnet)
convnet = InstanceNormalization(axis=3, center=True, scale=True, beta_initializer='random_uniform', gamma_initializer='random_uniform')(convnet)

convnet = Dense(10, activation='softmax')(convnet)
classifier = Model(inputs = theInput, outputs = convnet)
classifier.summary()
```

Figure 5. 8. Snippet to define the densely connected layer for classification

```
end2end = Sequential([dim_encoder, classifier])
end2end.summary()
```

Figure 5. 9. Snippet to join the loaded dimensionality encoder and the classifier module

```
def myBatchGenerator(batch_size, images, labels):
    batch_counter = 0
    save_batch = 0
    while True:
        batch_image = []
        batch_label = []
        while batch_counter < batch_size + save_batch:
            #rand_ind = np.random.randint(len(images));#Random index number to access list

            image_name = images[batch_counter]
            theImage = np.load(image_name)
            label_name = labels[batch_counter]
            theLabel = np.load(label_name)
            #The input image and the batch label should be the same in this case

            batch_image.append(theImage)
            batch_label.append(theLabel)

            #print(theImage.shape)
            batch_counter = batch_counter + 1
            #print("batch_counter : "+str(batch_counter))
        #print("batch_size + save_batch : "+str(batch_size + save_batch))
        if(batch_counter == len(images)-1 or (batch_size + save_batch) > len(images)-1):
            batch_counter = 0
            save_batch = 0

        save_batch = batch_counter
        BI = np.array(batch_image)#.astype(np.float32);#Batch image
        BL = np.array(batch_label)#.astype(np.float32);#Batch labe
        yield BI, BL
```

Figure 5. 10. Batch generator for training the classification model

One different implementation is the batch generator which loads the ground truth labels with the corresponding image patches. Patches from the ground truth label are yielded with the patch in Figure 5.12 so that it can be compared with classification output to compute the loss function used to update the network weights. Both the dimensionality reduction model and the densely connected network are trained in an end-to-end manner to improve classification performance implemented by the code give in Figure 5.13.

```
history_v1 = end2end.fit(train_batch,
                         steps_per_epoch=27,
                         epochs=100,
                         verbose=1,
                         validation_data=test_batch,
                         validation_steps=1,
                         callbacks = get_callbacks("updated")
                         )
```

Figure 5. 11. Model training function that trains the classification model on training batch

5.3.2. Hyperspectral Change Detection Model Implementation

As with the classification model, a change detection architecture from Ahram Song et. al. was adapted to test the performance of the dimensionality reduction module [21]. The first step given in Figure 5.14 show the cod snippet to load any of the above three dimensionality reduction techniques to reduce the dimension of input images. Since the loaded model is an autoencoder the next step is cut the encoder section of the model and for use with the change detection as show in Figure 5.15. The newly created model is integrated with the batch generator so that every coupled temporal slice pass through this model.

Later, the ConvLSTM-based change detection model is implemented via Keras TimeDistributed layers and a ConvLSTM2D layer that received input from the time distributed layer. The time distributed layer takes any operation (densely connected or convolutional) and applies this operation to batches of similar sized images. In this case a convolution operation is time distributed on the coupled patches to extract similar information form the bitemporal slices. Later two successive ConvLSTM2D layers accept these inputs one after the other, one of which has its return sequences enabled as it is not the last layer and the other has it disabled as the densely connected layer does not interpret it. The densely connected layer has 6 neurons that create/predict a map of changes between two temporal slices.

```
bce_dim = load_model("hermi_dim_reduc_v2.h5", custom_objects={"LeakyReLU":LeakyReLU()})
bce_dim.summary()
```

Figure 5. 12. Load pretrained dimensionality reduction model for training with the change detection architecture

```
output_layer = bce_dim.layers[-12].output
print(output_layer)
```

```
Tensor("concatenate/concat:0", shape=(None, 64, 64, 32), dtype=float32)
```

```
bce_encoder = Model(inputs=bce_dim.input, outputs=output_layer)
bce_encoder.trainable = False;
bce_encoder.summary()
```

Figure 5. 13. Slice the encoder section of the pretrained BCE and create a new encoder model

```
            tempImage = np.zeros((2, 64, 64, theImage1.shape[2]))
            tempImage[0, :, :, :] = theImage1;
            tempImage[1, :, :, :] = theImage2;
            tempImage = bce_encoder.predict(tempImage)
```

Figure 5. 14. Integrate the BCE with the new batch generator to reduce and every input's dimension

```
convLSTM = TimeDistributed(Conv2D(64, (5, 5), padding='same', activation=LeakyReLU()))(theInput)
convLSTM = TimeDistributed(Conv2D(32, (3, 3), padding='same', activation=LeakyReLU()))(convLSTM)
convLSTM = ConvLSTM2D(32, kernel_size = (3, 3),padding = 'same', return_sequences = True)(convLSTM)
convLSTM = ConvLSTM2D(16, kernel_size = (3, 3), padding = 'same', return_sequences = False)(convLSTM)
convLSTM = Dense(6, activation = 'softmax')(convLSTM)

lstm_model = Model(inputs = theInput, outputs = convLSTM);
lstm_model.summary();
```

Figure 5. 15. Build the ConvLSTM's layer to create the change detection architecture

```
history_v1 = lstm_model.fit(train_batch,
                            steps_per_epoch=92,
                            epochs=10,
                            verbose=1,
                            validation_data=test_batch,
                            validation_steps=1,
                            callbacks = get_callbacks("updated")
                            )
```

Figure 5. 16. Train the model using a bitemporal cube slices and a corresponding ground truth image by defining the number of epochs and steps

CHAPTER 6

RESULTS AND DISCUSSION

6.1. Chapter Overview

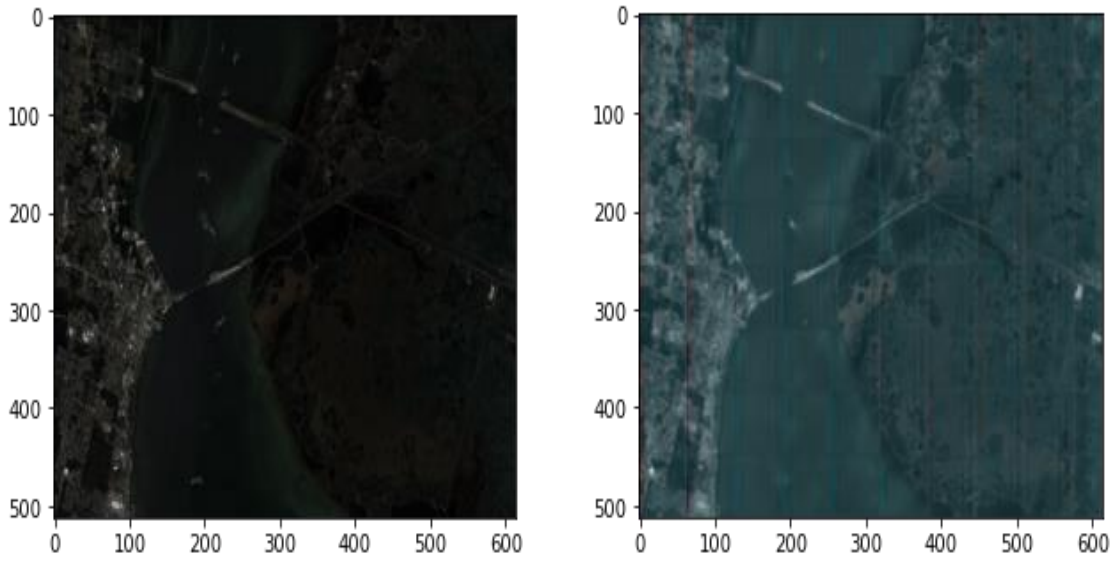
Having discussed the details of the implementations and experiments to be performed, this chapter presents the results that were obtained from the experiments. Coupled with the classification model and the change detection model, the proposed dimensionality reduction's model performance was tested using the evaluation metrics mentioned in the third chapter. Moreover, the proposed dimensionality reduction model's performance is compared with existing dimensionality reduction architectures (a densely connected autoencoder architecture and a convolutional autoencoder architecture). The comparative results are given below in combination with the classification and change detection models.

6.2. Dimensionality Reduction Model Prediction Results

Four different datasets have been used to train the proposed dimensionality reduction model with some minor modification for each dataset. These datasets are namely the Pavia University hyperspectral dataset, the Pavia Center hyperspectral dataset, the Kennedy Space Center hyperspectral dataset, and the Salinas hyperspectral dataset. After training the model on the patched dataset for each of the images above for a hundred epochs, the performance of the model was tested for its ability to reconstruct the original image. Also, to measure the spatial similarity between the original and the reconstructed input L1 and L2 similarity measures are used. Finally, the proposed dimensionality reduction's scheme performance is compared with the existing methods using the similarity measures stated in the previous chapter.

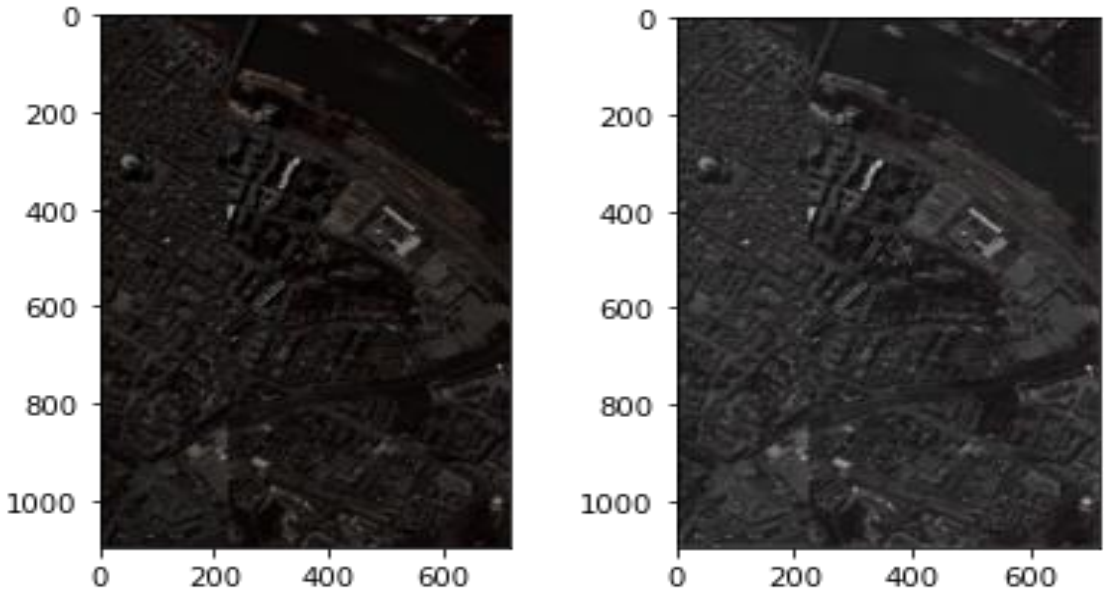
6.2.1. Convolutional Autoencoder Dimensionality Reduction Results

After training the CAE on the four datasets for about two hours each datasets the following results were obtained. The false color images below the original and reconstructed hyperspectral data are shown for visual comparison. Later tabular comparison with the L1 and L2 losses are given for the four datasets.



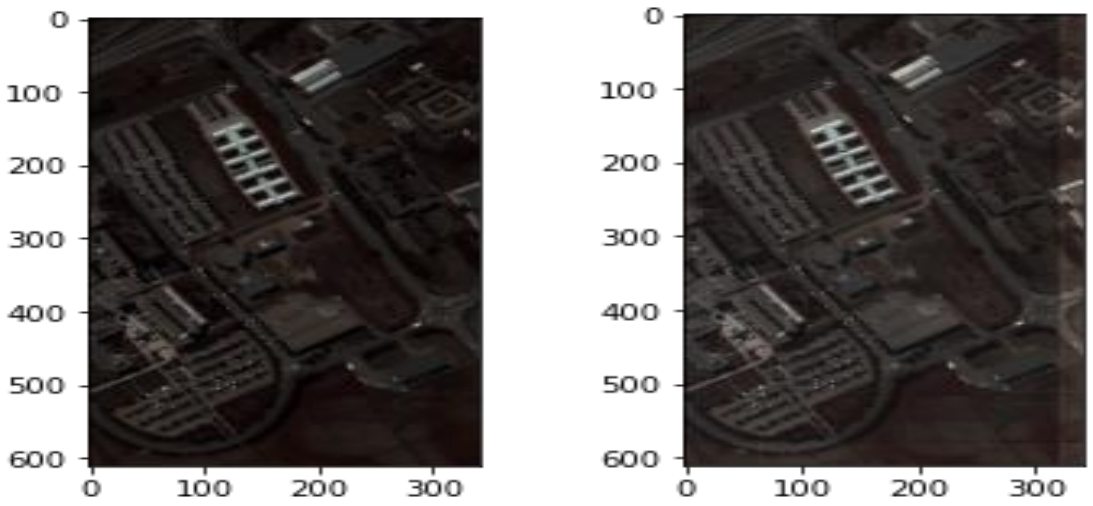
a) b

Figure 6. 1. Original and reconstructed images for the KSC dataset (a: original and b: reconstructed)



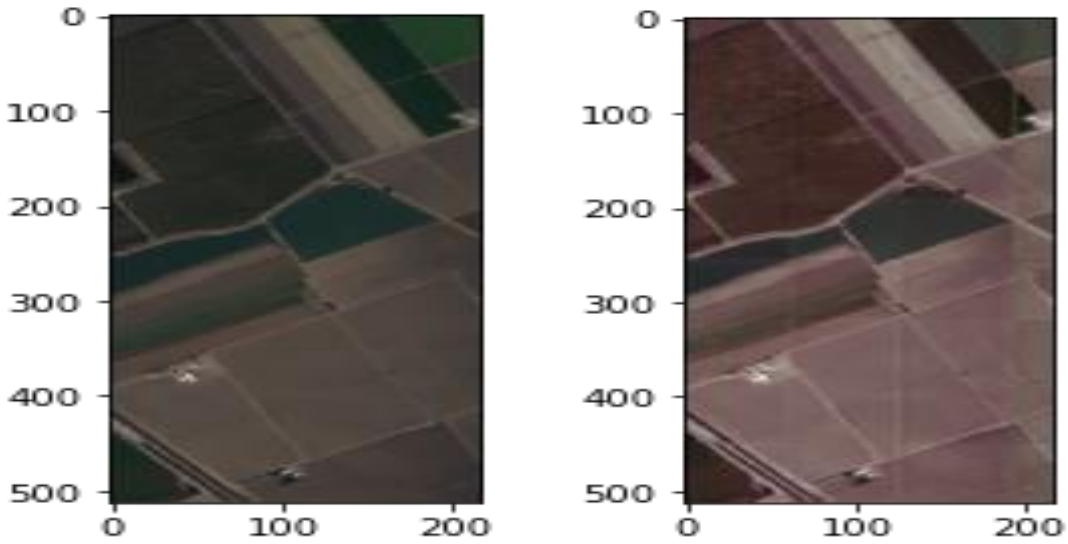
a) b)

Figure 6. 2. Original and reconstructed images for the PaviaC dataset (a: original and b: reconstructed)



a) b)

Figure 6. 3. Original and reconstructed images for the Pavia University dataset (a: original and b: reconstructed)



a) b)

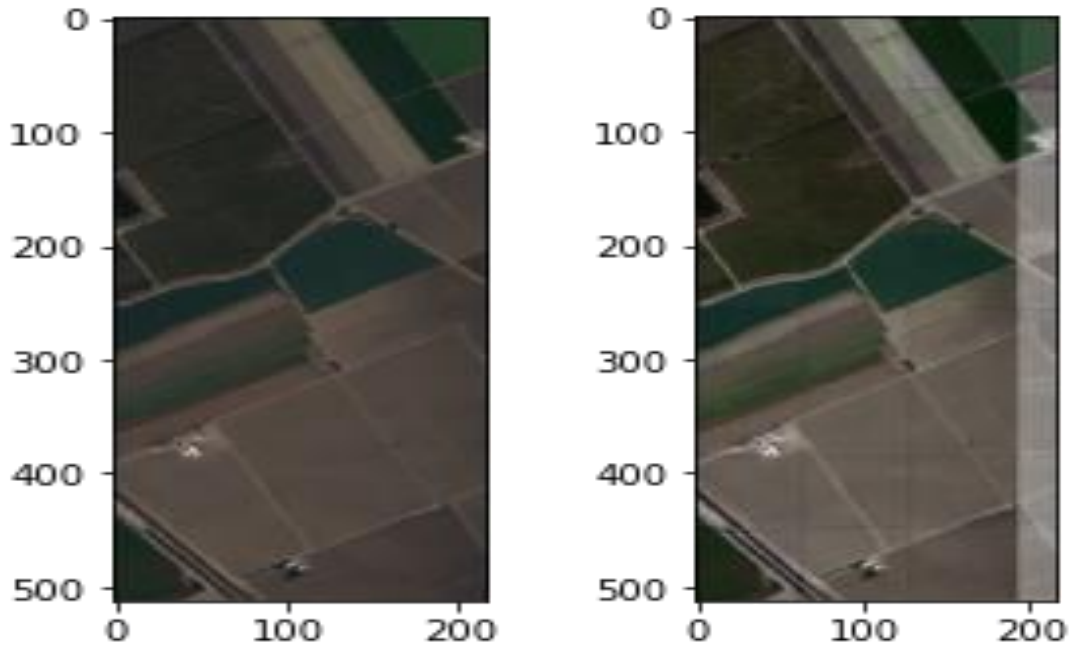
Figure 6. 4. Original and reconstructed images for the Salinas scene data using a 2D convolutional autoencoder (a: original and b: reconstructed)

It can be seen from the figures Figure 6.1 to Figure 6.4 the convolutional autoencoder has managed to capture important visual information from the latent representation and somehow reconstructed the original input.

However, there are some artifacts in the reconstructed images that were not present in the original image. In all the four reconstructed images there are blurring lines running vertically that are not in the corresponding original images. Also, it was worth mentioning sharp edges and corners are a little blurry in the reconstructed image which implies the CAE dimensionality reduction model has a drawback in reconstructing sharp edges and sharp corners.

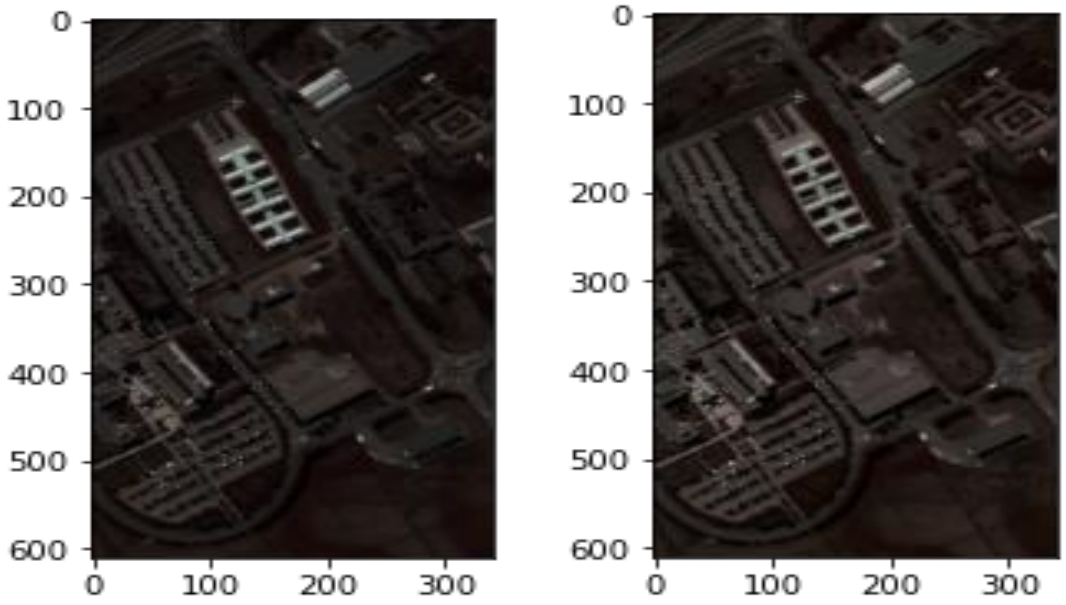
6.2.2. Densely-Connected Autoencoder Dimensionality Reduction Results

Due to the immense interconnectivity of the dense layer the dense autoencoder was trained for about two hours on the four datasets and the following results were obtained (which took about the same time that it took for the CAE and BCE). The false color images below the original and reconstructed hyperspectral data are shown for visual comparison. Later tabular comparison with the L1 and L2 losses are given for the four datasets.



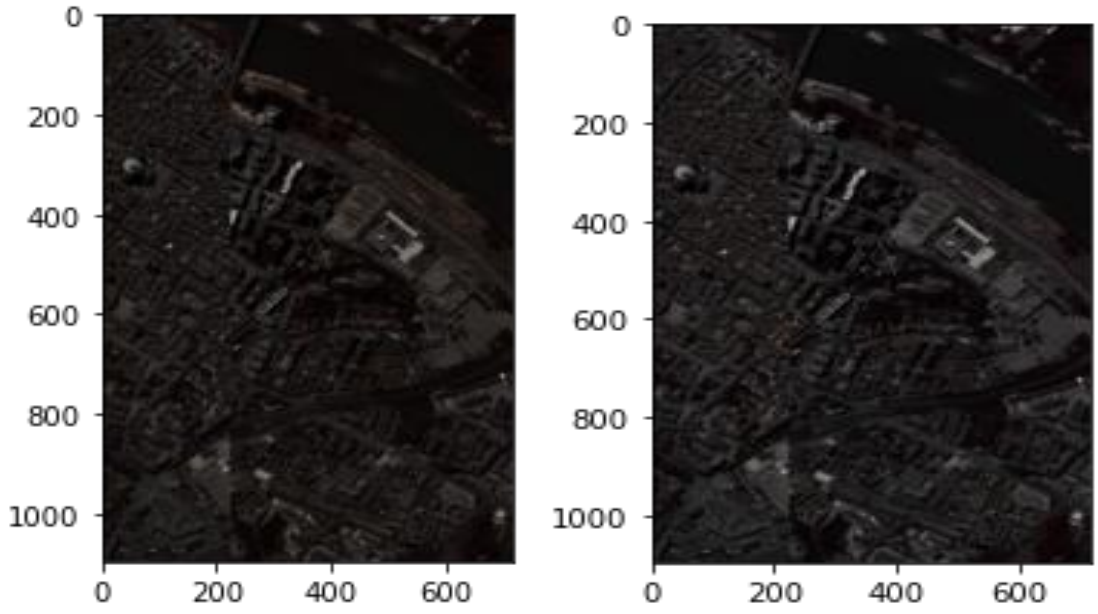
a) b)

Figure 6. 5. Original and reconstruction Salinas Scene output comparison for the dense autoencoder (a: original and b: reconstructed)



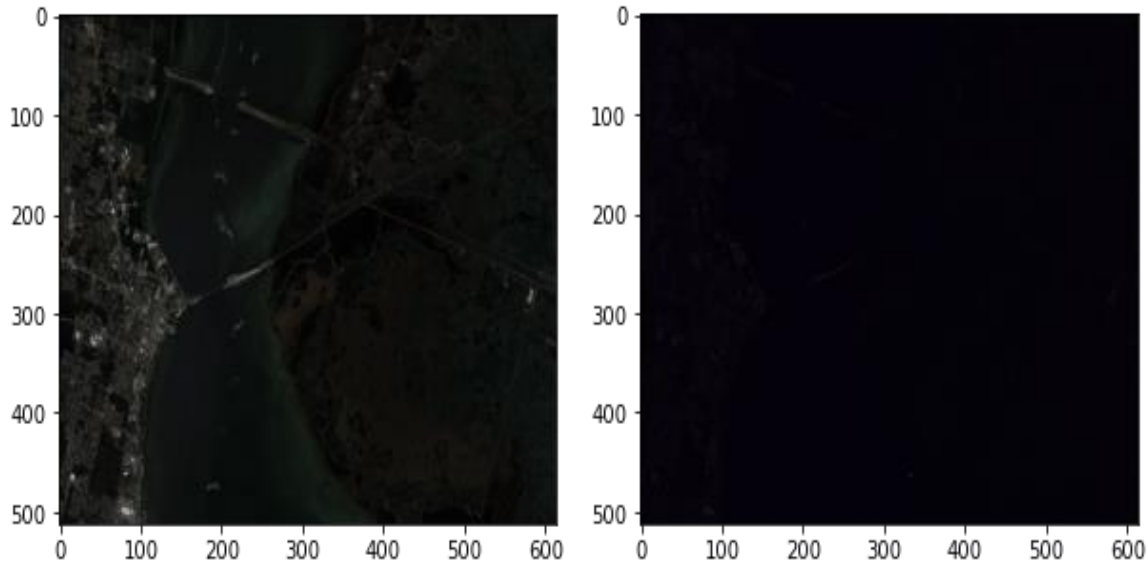
a) b)

Figure 6. 6. Original and reconstructed PaviaU image using densely connected dimensionality reduction model (a: original and b: reconstructed)



a) b)

Figure 6. 7. Original and reconstructed image results for PaviaC data using the densely connected dimensionality reduction model (a: original and b: reconstructed)



a) b)

Figure 6. 8. Original and Reconstructed Kennedy Space Center (KSC) images using dense autoencoder (a: original and b: reconstructed)

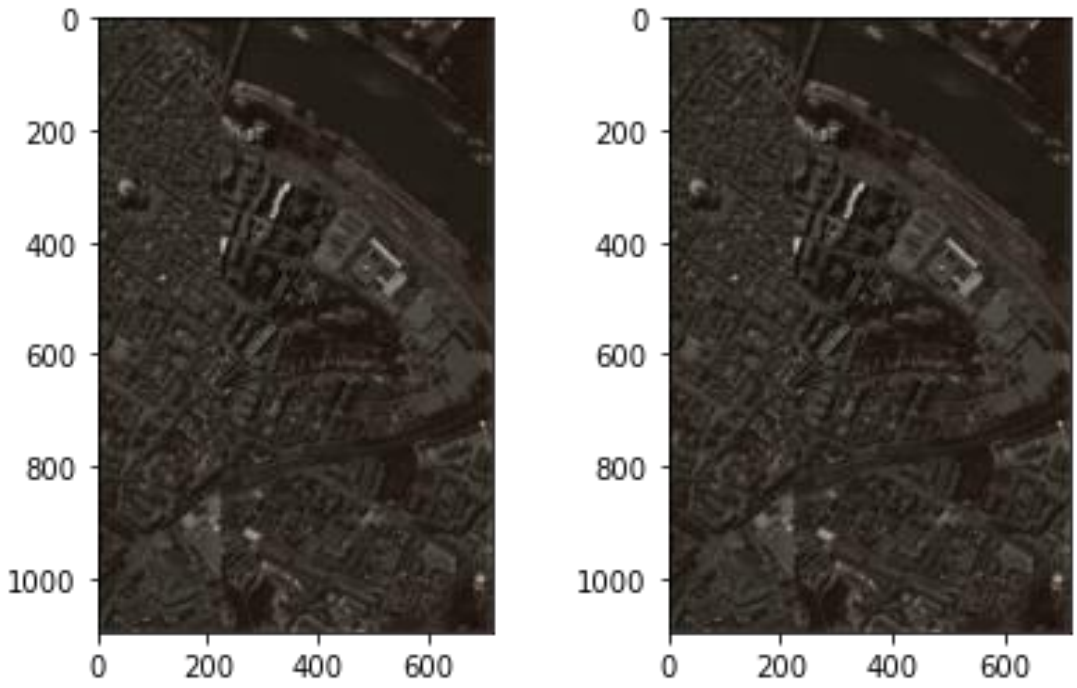
The densely connected layer performed well considering the shortage of data and the training time it took. On the first three images (Salinas Scene, PaviaU, and PaviaC) the densely connected layer has accurate reconstruction results and sometimes brining some contrast stretching manipulation into the original image which seems to have improved its visual appearance.

However, the reconstruction of the Kennedy Space Center data suffered severely. This indicates that the densely connected autoencoder performs poorly during training on an image having a large contrast range. The image is not entirely dark but has some shapes but the contrast of the image is poor and makes it difficult to see color and shapes. One final remark to make is the densely connected network has a one-to-one mapping considering the number of connections it has. Therefore, although it has performed well on the datasets this result cannot be directly attributed to its feature representation power.

6.2.3. BCE Autoencoder Dimensionality Reduction Results

The branching convolutional autoencoder architecture has the largest number of trainable parameters due to the structure of the architecture and was trained for about 70 to 100 epochs

depending on the dataset. Comparatively it has the best visual appearance and has a comparable L1 and L2 reconstruction losses on some datasets. Figures 6.9 to 6.12 illustrate the reconstruction results obtained on the four datasets.



a) b)

Figure 6. 9. Original and reconstructed image for the Pavia Center scene using BCE (a: original and b: reconstructed)

It can be seen from Table 6.1 that BCE overall has the best performance with respect to L1 and L2 similarity metrics. With the exception of the PaviaU where the DNN model has outperformed it the BCE architecture's reconstruction performance is superior to the others. With respect to the SA and CSA measures, the result seems to be mixed with the BCE performing well on most but not on all. This is due to the challenge faced when trying to strike the balance between the spectral and spatial representation. Because at times representing one might accurately might adversely affect the other and vice versa. But it seems BCE has managed to balance this trade off with the exception of two instances.

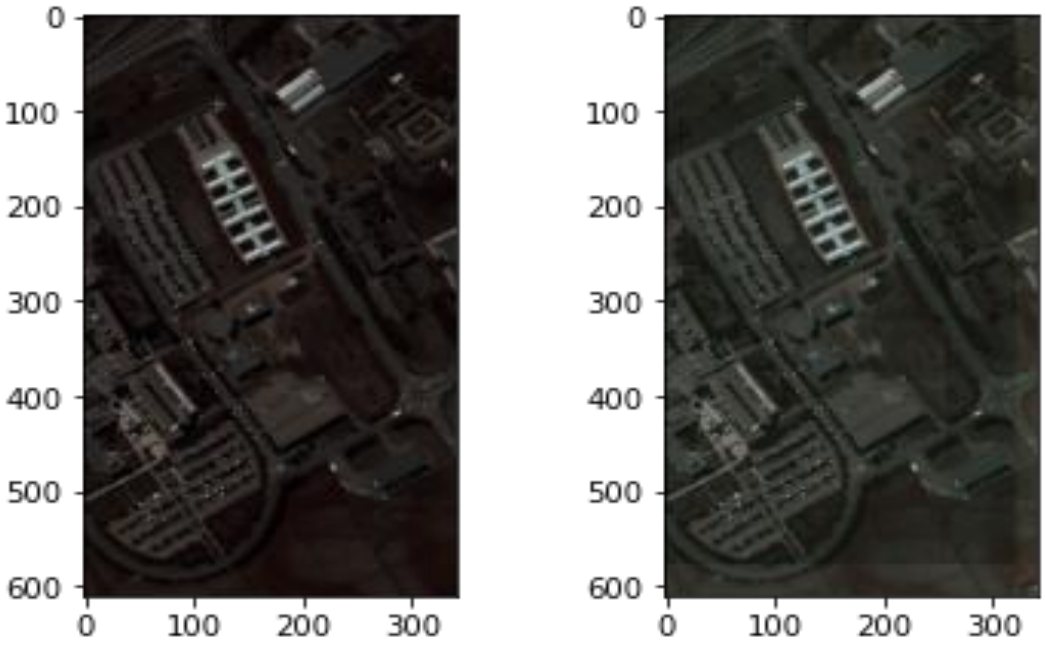


Figure 6. 10. Original and reconstructed image for the Pavia Image HSI using BCE (a: original and b: reconstructed)

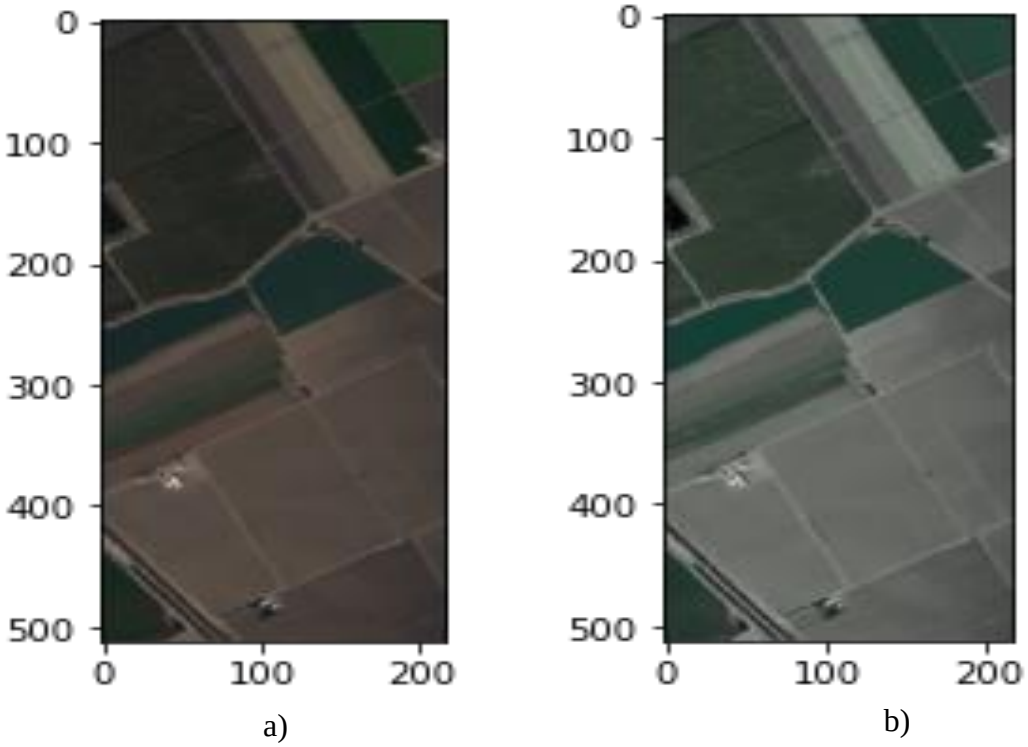


Figure 6. 11. Original and reconstructed images for the Salinas scene HSIs using BCE (a: original and b: reconstructed)

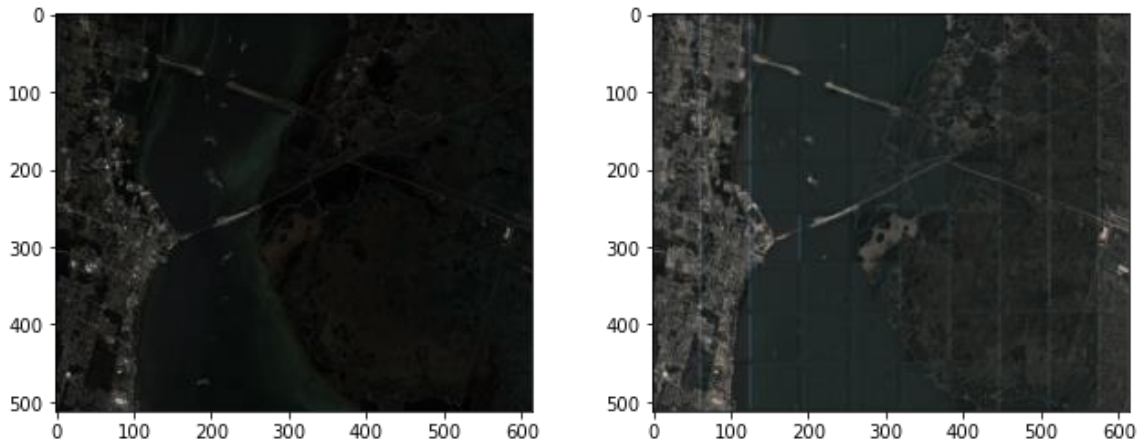


Figure 6. 12. Original and reconstructed images for the Kennedy Space Center scene HSI using BCE (a: original and b: reconstructed)

Table 6. 1. Dimensionality reduction result summary comparison summary

Dataset	DR Model	Evaluation Metrics			
		L1-metric	L2-metric	SA	CSA
KSC	CAE	**0.00223**	0.04434	1.56569	0.9949
	DNN	0.00224	**0.04333**	1.56571	0.9949
	BCAE	0.00274	**0.04433**	**1.56562**	**0.9948**
PaviaC	CAE	0.01549	0.02418	1.56113	0.9904
	DNN	0.01191	0.01817	1.56111	0.9904
	BCAE	**0.01083**	**0.01667**	**1.56107**	**0.9903**
PaviaU	CAE	0.01687	0.03106	1.56112	0.9903
	DNN	**0.01083**	**0.01667**	1.56113	**0.9902**
	BCAE	0.02116	0.03250	**1.56107**	0.9903
Salinas	CAE	0.01010	0.02351	1.56591	**0.9951**
	DNN	0.01245	0.03090	**1.56590**	**0.9951**
	BCAE	**0.00549**	**0.01008**	1.56591	**0.9951**

Table 6.1 summarizes the results of the different autoencoders to demonstrate their reconstruction ability via L1 and L2 losses for spatial similarity measure, and Spectral Angle (SA), and CSA (Cosine of Spectral angle) for spectral similarity measure. Overall, the BCAE has the best results giving the smallest SA and CSA on all four datasets with one exception. Moreover, spatially it has comparable performance on the spatial similarity measure outperforming the others on two datasets with one anomalous result shown on the PaviaU dataset.

6.3. Multiclass Classification Model Prediction Results

The proposed multiclass classification model is tested with the datasets used for training and testing the dimensionality reduction model. Moreover, popular dimensionality reduction techniques are also used to train the classification model. Later, the results from the DR model of this thesis are compared with the existing techniques using confusion matrix, overall accuracy and average accuracy.

6.3.1. Classification on the CAE Dimensionality Reduction Output

Judging from the comparative figures (i.e., predicted output and ground truth image), CAE has the least classification accuracy of all and has a lot of mis classification where it should have been dark in Figure 6.13 on the three datasets. Although this happens on all the datasets, comparatively it is worse on the classifier that has CAE dimensionality reduction step. The confusion matrix showing the number of classified pixels is given on the appendices section.

6.3.2. Classification on the DNN-based Dimensionality Reduction Step

Moving on to the classifier trained on the DNN dimensionality reduction technique the classification results are given in Figures 6.13 for the three datasets. On the classification task the classifier trained on the DNN the it performed better on the Pavia Center datasets. On the other datasets, its performance was the least of all. This could be attributed to the one-to-many mapping nature of the DNN that does not involve a significant feature mapping except the linear activations and product and bias with the weight connections. The confusion matrix showing the number of classified pixels is given on the appendices section.

6.3.3. Classification Results on the BCE Reduction Output

For the classifier trained on the BCE the results exceeded that of its competitors by a significant margin. Where the overall accuracy of classifier on all three datasets is above 93 percent. This is due to the stacked convolutional operators and the branching scheme that has high representation power when constructing the latent feature. The classifier's power is greatly observed when it is coupled with the BCE as an intermediary dimensionality reduction step. It is also worth noting that the F1-score is larger on the BCE which indicates that the recall and precision power of the classifier is higher than the other setups.

To see the detailed comparison of each dimensionality reduction techniques performance on the classification model the following results were displayed on Table 6.2. The three models are compared against each other via accuracy, recall, precision and F1-score. BCE performed exceptionally well on all metrics on the three datasets except on the recall score of the Salinas scene dataset. The performance achieved by the BCE is a testament to its feature representation power and separability of the feature representation on classification models.

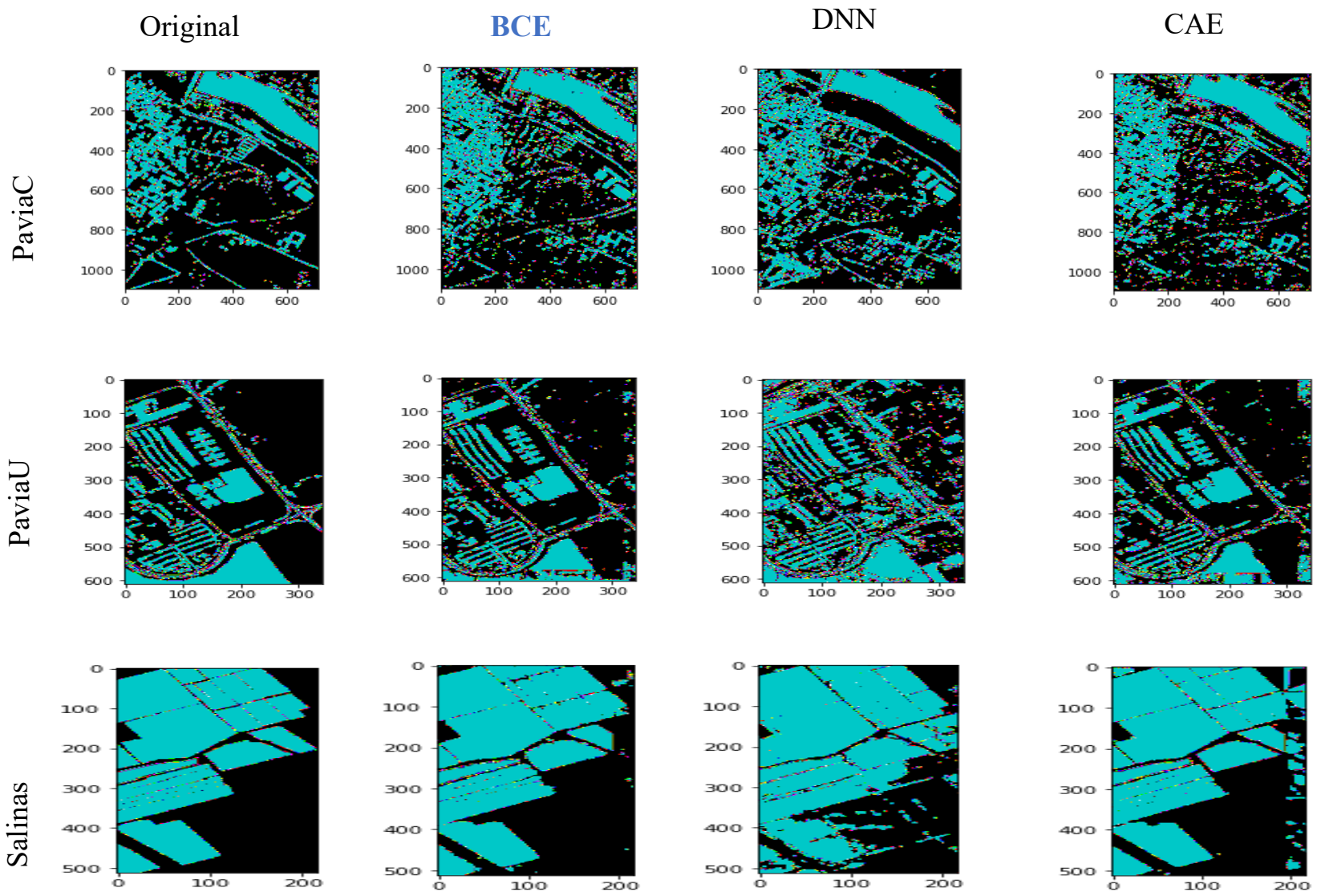


Figure 6. 13. Classification model prediction on DR models: comparison and contrast

Table 6. 2. Classification Model's performance on the different CD models vs different datasets

Data sets	Performance evaluation metric	Dimensionality reduction models		
		CAE	DNN	BCE
PaviaC (10 classes)	Accuracy	0.36	0.88	**0.93**
	Recall	0.66	0.44	**0.69**
	Precision	0.67	0.45	**0.75**
	F1-score	0.63	**0.70**	**0.70**
PaviaU (10 classes)	Accuracy	0.92	0.83	**0.94**
	Recall	0.82	0.59	**0.87**
	Precision	0.80	0.69	**0.88**
	F1-score	0.81	0.57	**0.88**
Salinas (17 classes)	Accuracy	0.94	0.65	**0.96**
	Recall	**0.96**	0.76	0.94
	Precision	0.91	0.77	**0.94**
	F1-score	0.93	0.70	**0.94**

6.4. Change Detection Model Prediction Results

The three dimensionality reduction architectures including BCE were tested for their feature representation performance against the task of hyperspectral image change detection. Hermiston and Barbara area bitemporal hyperspectral images were used to test the performance of these dimensionality reduction architectures using a 3D recurrent fully convolutional change detection network the results obtained are given below.

6.4.1. Multi-class Change Detection Results with the Convolutional Autoencoder Dimensionality Reduction Model

The convolutional autoencoder was connected with the 3D recurrent fully convolutional network to reduce the dimension of the original dataset patches. Then, the change detection algorithm was trained on the input fed from the dimensionality reduction model. After the training the model for about 10 epochs, the following results depicted on Figure 6.11 below were obtained. Visually the ground truth change-map and the predicted output by the change detection map seem to closely resemble each other with few discrepancies between the two.

There are some omissions on the change detection output owing to the fact that some classes are not sufficiently present making the dataset composed of imbalanced classes. The omissions by the CAE indicate that the dimensionality reduction framework does not sufficiently represent small shapes and areas.

This is clearly demonstrated by the confusion matrix given on Figure 6.14 which shows that pixels from the third class have been entirely misclassified by the change detection network. What explains this result is the feature representation power of the CAE dimensionality model does not work on classes that have small number of pixels present in the original hyperspectral image. Moreover, to get a clear visual picture of the change detection model' prediction result vs the actual ground truth image is shown in Figure 6.28 by taking patches and displaying them in a grid. The contrasting images have some minor differences where the change prediction has some extra artifacts and overlapping image predictions for the patches.

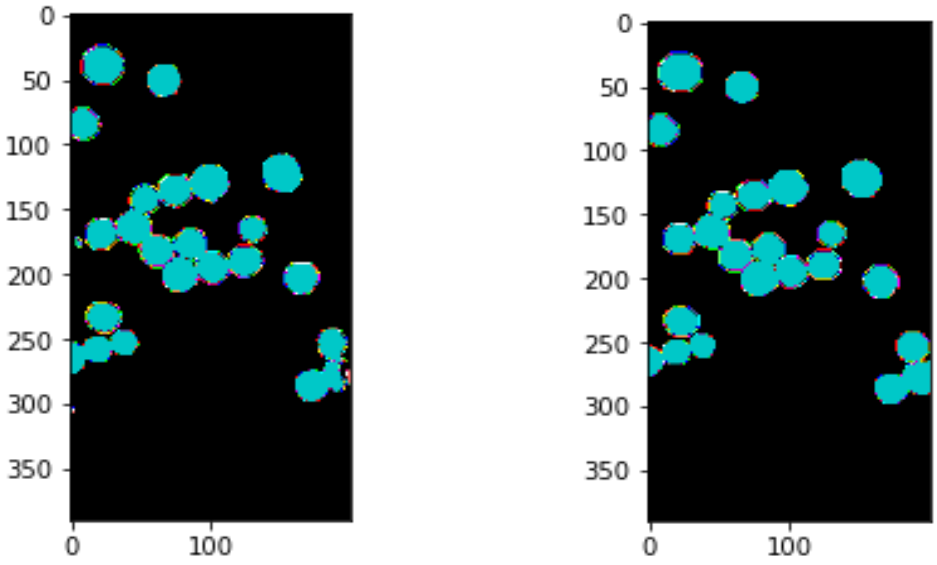


Figure 6. 14. Change detection map result using CAE dimensionality reduction (a: CD GT model output and b: ground truth image)

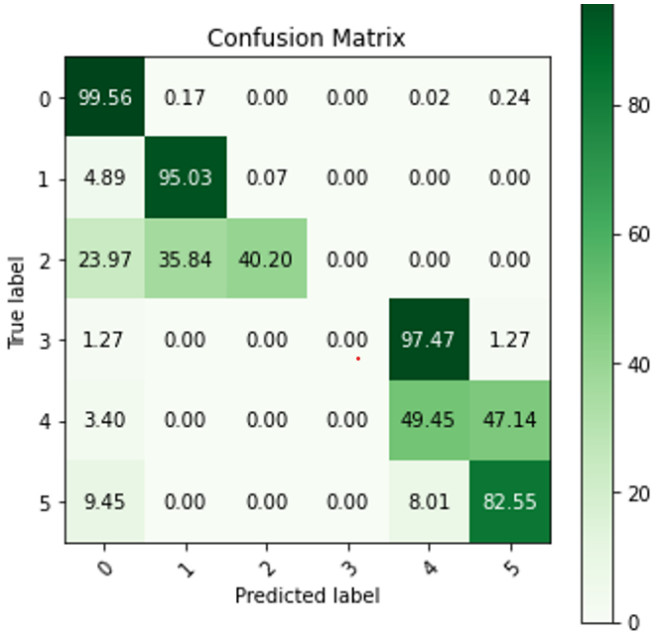


Figure 6. 15. Confusion matrix for the convolutional autoencoder model's performance on change detection

6.4.2. Multiclass Change Detection Results with the Densely Connected Network Dimensionality Reduction Model

In a manner similar to the previous model's training, the change detection model was trained with the densely connected dimensionality reduction model for about 10 epochs on the Hermiston area multitemporal hyperspectral images. The multiclass change detection map and the confusion matrix for the performance of the supervised change detection model are presented on Figure 6.24 and Figure 6.25 respectively. Figure 6.24 depicts the original change detection map and the predicted output of the model. Here, the results are quite similar to each other and even better on some instance as compared with previous model.

Even though visual comparison between the two images does not reveal any significant differences between the ground truth change map and the predicted output, the confusion matrix has a different take on its performance. Similar to the previous model, the third class has been entirely misclassified which indicates the DNN's shortcoming in representing small shapes and patterns in to the feature space. Moreover, to get a clear visual picture of the change detection model' prediction result vs the actual ground truth image is shown in Figure 6.29 by taking patches and displaying them in a grid. The contrasting images are

having some minor differences from one another especially on sharp edges around the circles.

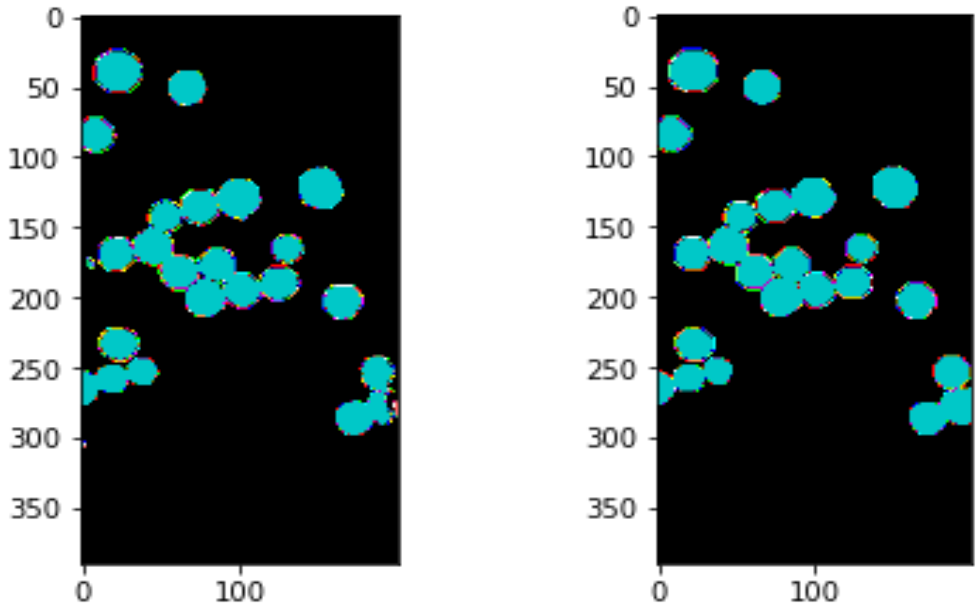


Figure 6. 16. Change detection multiclass classification using the densely connected architecture as a dimensionality reduction model (a: reconstructed and b: original)

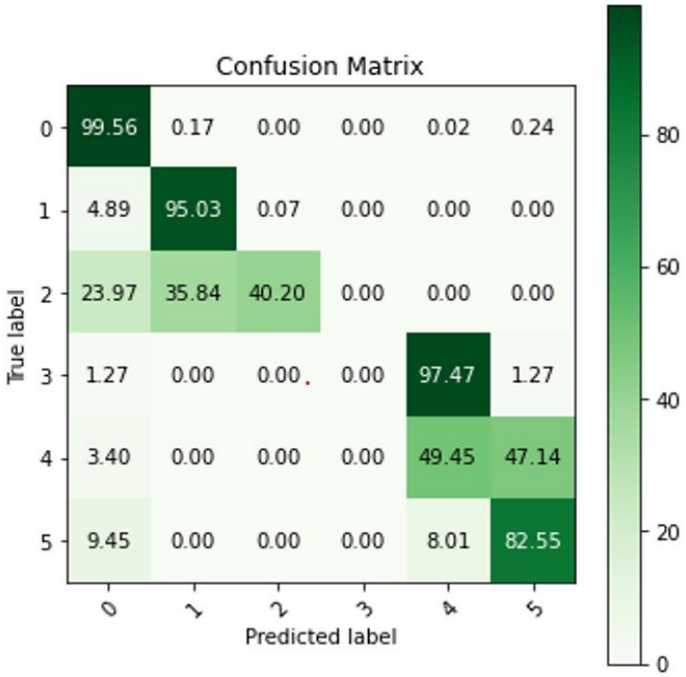


Figure 6. 17. Confusion matrix output of the change detection performance using densely connected model as a dimensionality reduction model

6.4.3. Multiclass Change Detection Results with the BCE Dimensionality Reduction Model

BCE was coupled together with the change detection architecture to reduce the dimension of paired input images. After these paired images undergo dimensionality reduction, they are fed to change detection to determine the changes that have occurred between the two patches. The prediction output and the confusion matrix plot for the BCE performance when coupled with the dimensionality reduction model are given on Figure 6.26 and Figure 6.27. It is hard to determine the performance from the maps of the ground truth image and the prediction result as they closely resemble each other. The training and testing performance plots are given in the appendices section.

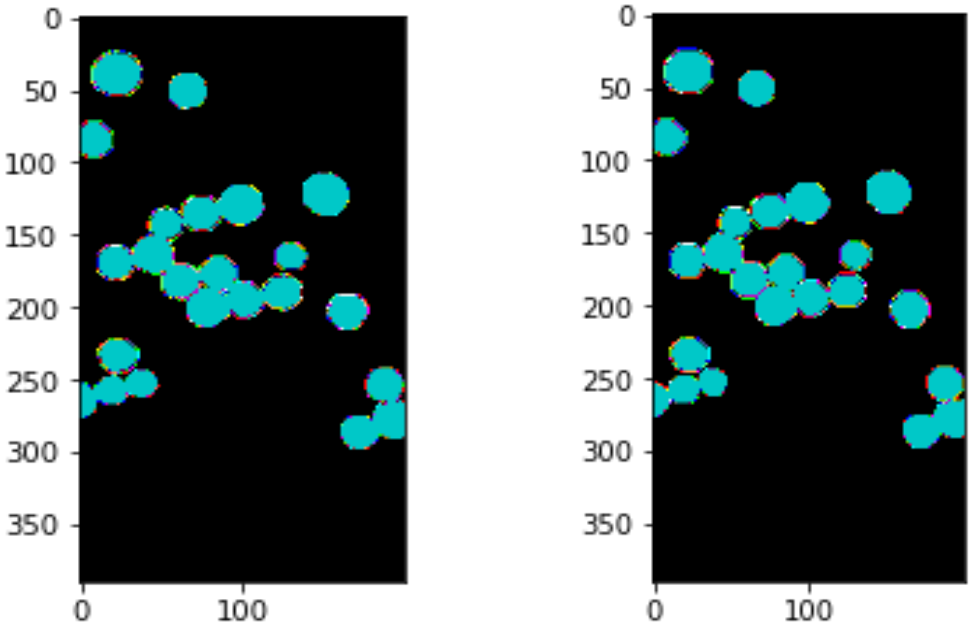


Figure 6. 18. Multiclass change detection map for the BCE dimensionality reduction model (a: reconstructed and b: original)

However, the confusion matrix shows that about one third of the elements of the third class are have been properly classified. This is a clear improvement from the previous models as both of them have misclassified all the elements of the third class. Moreover, to get a clear visual picture of the change detection model' prediction result vs the actual ground truth image is shown in Figure 6.30 by taking patches and displaying them in a grid. The contrasting images are almost indistinguishable from one another with some minor

differences on sharp edges around the circles. Here there are no visible artifacts and overlapping classifications on the circles. Clearly, the feature representation by the Branching CE has outperformed the others.

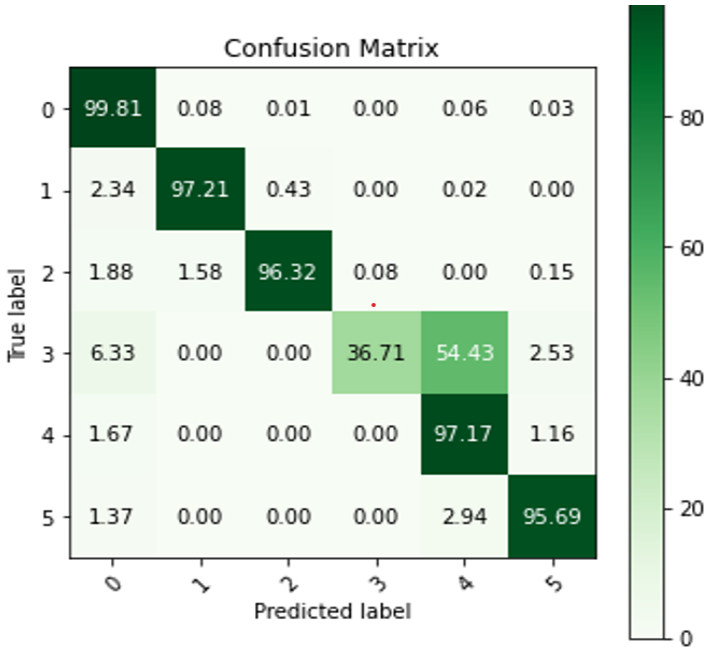


Figure 6. 19. Confusion Matrix output for the change detection prediction using BCE dimensionality reduction framework

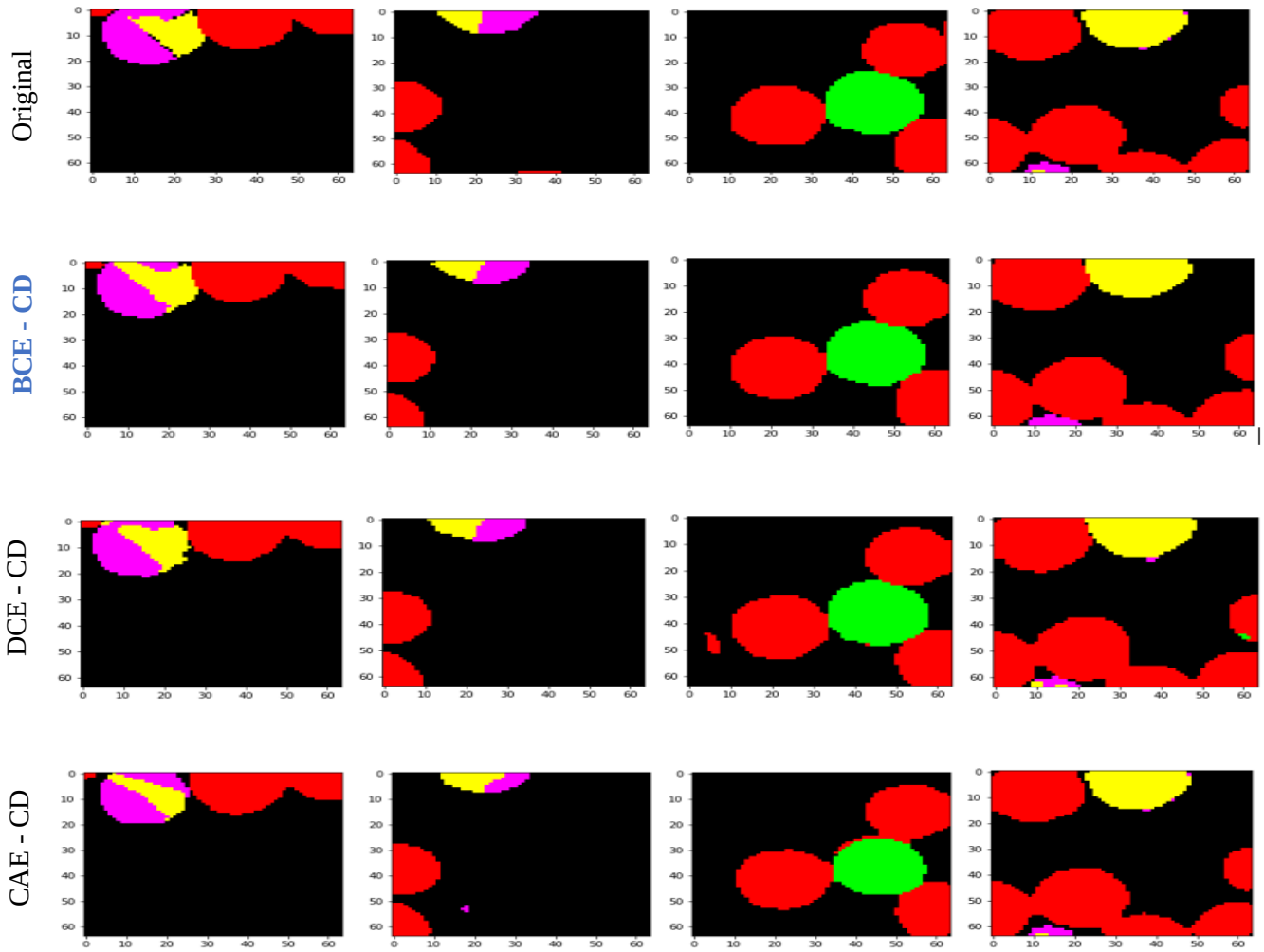


Figure 6. 20. Change detection results using the DR modules

Table 6.3 shows the performance comparison of three models on four metrics. BCE has the highest performance out of the three exception of one precision value.

Table 6. 3. Change detection performance of the three models

Dataset	Models		
	CAE	DNN	BrachingCE
Hermiston City (Accuracy)	0.96804	0.98626	**0.99372**
Hermiston City (Recall)	0.61131	0.76273	**0.87152**
Hermiston City (Precision)	0.70575	0.78573	**0.96203**
Hermiston City (F1-Score)	0.79516	**0.93854**	0.88978

6.4. Sample Training Log for the BCE Training

The following two sample training curves were given for the dimensionality reduction scheme training on the Pavia University dataset (Figure 6.31) and on the Kennedy Space Center dataset (Figure 6.32). The graphs show how the loss (vertical) drops as the epoch (horizontal) proceeds.

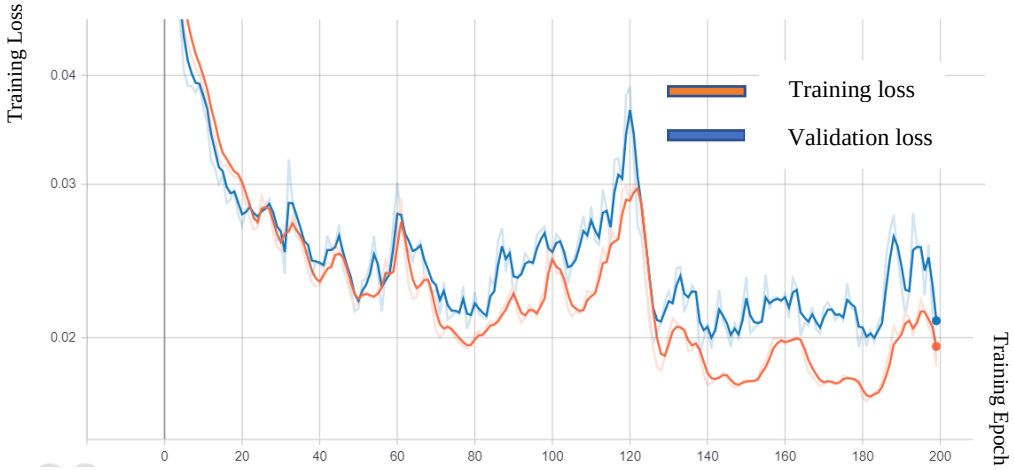


Figure 6. 21. BCE dimensionality reduction training curve on the PaviaU dataset

On both instance the model learns to reduce the L1-loss between the original and reconstructed image without overfitting or underfitting the model. The rest of the training curves are given in appendices section of the thesis.

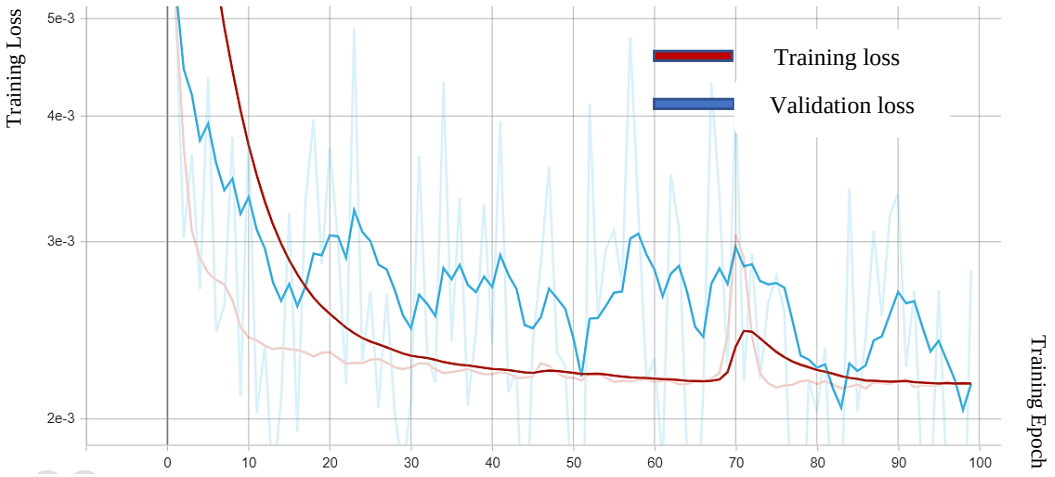


Figure 6. 22. Branching CE dimensionality reduction training curve on the KSC dataset

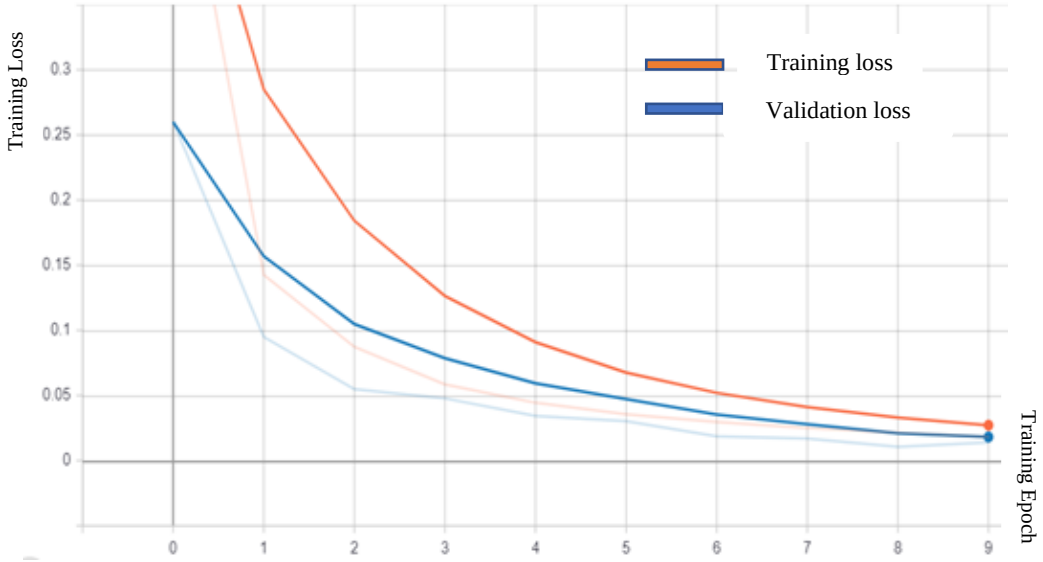


Figure 6. 23. Change Detection model training with BCE

6.5. Summary

In this chapter the experimental results for the three models namely dimensionality reduction, classification, and change detection were presented. The results of the dimensionality reduction were displayed in a manner to show the contrast between the original and the reconstructed image. There were four datasets that were selected for this task, and this task the BCE outperformed the other two models on most of the performance metrics. With the exception of some minor contrast stretching and blurring of sharp and corner and edges the reconstruction was almost identical.

In addition, the classification model and the change detection model that were trained together with the three dimensionality reduction techniques had a different degree of performance on the different datasets. For the classification task, the classifier's predictions have artifacts and some omissions on all three dimensionality reduction techniques. However, the BCE has a demonstrable performance both visually and with the accuracy, precision, etc. metrics. This also true for the change detection model which was trained on the Hermiston City dataset. The model's prediction shown with grid and the original size change map has consistent result for the BCE dimensionality reduction step. Overall, the BCE's feature extraction ability to reduce the dimension of the original image is demonstrably higher than the other two models.

CHAPTER 7

CONCLUSION AND FUTURE WORKS

7.1. Conclusion

Hyperspectral image classification and change detection are important applications of hyperspectral images for remote sensing and other fields. One of the main challenges when performing either classification or change detection on hyperspectral images is the high dimensional nature of the images due to the large number of spectral bands present. Scores of researches have been done to explore numerous dimensionality reduction schemes for hyperspectral images. This thesis presented a spatio-spectral dimensionality reduction technique using a branching autoencoder architecture. The encoder section consists of two convolutional encoder one operating on spectral sequences and the other on spatial steps.

Overall, the research was designed to explore other autoencoder techniques for the reduction of hyperspectral images. Namely a convolutional autoencoder and a densely connected autoencoder network have been selected as a comparative technique to compare the proposed techniques performance. Four popular hyperspectral images were selected to demonstrate the dimensionality reduction technique's feature representation power. Three separate experiments were done where the first one was to compare the reconstruction power of the three autoencoder models on the four datasets. After the three models were trained on the available on the images, they were later integrated with a classification model and a change detection model to further test their representation power.

The first experiment showed that branching autoencoder network has a better reconstruction capacity as demonstrated by the visual appearance of the reconstructed images and reconstruction loss being the smallest out of the three. For the classification task the dimensionality reduction models were integrated with a classifier to be trained in and to end manner. Out of the three models the classification result using the BrancingCE section had better visual consistency and performance with different metrics (overall accuracy, precision, recall, and F1-score). Finally, the change detection model was trained using the Hermiston bitemporal images in supervised manner. The change map prediction showed that the model

integrated with the BCE has fewer artifacts and overlaps on the different classes that the other two models.

BCE has demonstrably shown a better feature representation power than the other two comparative models. Moreover, this research has answered the second research question of how to design an effective dimensionality reduction technique that integrated spatial and spectral features of the data. As L1 and L2 losses for the spatial similarity between the original and the reconstructed were smaller than the other two techniques. Also, the spectral angle and the cosine of the spectral measures were somewhat better than the two. Inline with the reconstruction power the model's performance while being integrated with the classification and the change detection models, addresses the third research question of integrating the designed dimensionality reduction model with classification and change detection models.

Finally, it is reasonable to conclude that spatial-spectral design using convolutional architectures would yield a better performance due to its better spatial-spectral feature representation power. Moreover, the representation in the feature space is better discriminable as it is evident from the classification and the change detection tests. This work can be adapted to other datasets and would yield a better result.

7.2. Future Works

There was an obvious shortage of labelled data in this experiment and could not be done in a large scale. Customizing and training the model on large dataset would further demonstrate the effectiveness of the proposed model. Hence, this would make the model usable for several scenarios. Another interesting direction to further this research would be on unsupervised classification and change detection tasks. As clearly there are numerous unlabeled data available online. Taking the research in this direction would make it more robust and integrable to be used in wide scale.

Special Acknowledgement: This research was funded by Adama Science and Technology University under the grant number **ASTU/SM-R/092/19**.

Adama, Ethiopia.

REFERENCES

[1] J. E. Ball, D. T. Anderson, and C. S. Chan, “A Comprehensive Survey of Deep Learning in Remote Sensing: Theories, Tools and Challenges for the Community,” Sep. 2017, doi: 10.1117/1.JRS.11.042609.

[2] D. J. Hemanth, “Remote Sensing and Digital Image Processing Artii cial Intelligence Techniques for Satellite Image Analysis.” [Online]. Available: http://www.springer.com/series/6477.

[3] S. Lavender and A. Lavender, *Practical handbook of remote sensing*. .

[4] Timothy A Warner, *The SAGE Handbook of Remote Sensing*, 1st ed. 2009.

[5] L. Zhang and B. Du, “Recent advances in hyperspectral image processing,” *Geo-Spatial Information Science*, vol. 15, no. 3, pp. 143–156, 2012, doi: 10.1080/10095020.2012.719684.

[6] C. O. S. Sorzano, J. Vargas, and A. Pascual-Montano, “A survey of dimensionality reduction techniques.”

[7] • Griffin, “Examples of EO-1 Hyperion Data Analysis.”

[8] A. Shukla and R. Kot, “An Overview of Hyperspectral Remote Sensing and its applications in various Disciplines,” *IRA-International Journal of Applied Sciences (ISSN 2455-4499)*, vol. 5, no. 2, p. 85, Dec. 2016, doi: 10.21013/jas.v5.n2.p4.

[9] L. Bruzzone, S. Liu, F. Bovolo, and P. Du, “Change Detection in Multitemporal Hyperspectral Images,” 2016.

[10] J. M. Amigo, H. Babamoradi, and S. Elcoroaristizabal, “Hyperspectral image analysis. A tutorial,” *Analytica Chimica Acta*, vol. 896. Elsevier, pp. 34–51, Oct. 08, 2015, doi: 10.1016/j.aca.2015.09.030.

[11] M. John Canty, “Image Analysis, Classification, and Change Detection in Remote Sensing With Algorithms for Python Fourth edition.”

[12] Pu, R. (2017). Hyperspectral Remote Sensing: Fundamentals and Practices (1st ed.). Routledge. https://doi.org/10.1201/9781315120607

[13] Manolakis, D., Lockwood, R., & Cooley, T. (2016). Hyperspectral Imaging Remote Sensing: Physics, Sensors, and Algorithms. Cambridge: Cambridge University Press. doi:10.1017/CBO9781316017876

[14] G. E. Hinton, "Reducing the Dimensionality of Data with Neural Networks," *Science*, vol. 313, no. 5786, Jul. 2006, doi: 10.1126/science.1127647.

[15] L. Windrim, R. Ramakrishnan, A. Melkumyan, R. J. Murphy, and A. Chlingaryan, "Unsupervised Feature-Learning for Hyperspectral Data with Autoencoders," *Remote Sensing*, vol. 11, no. 7, Apr. 2019, doi: 10.3390/rs11070864.

[16] X. Zhang and N. D. Cahill, "Sumi-supervised deep autoencoder network for graph-based dimensionality reduction in hyperspectral imagery," in *Algorithms and Technologies for Multispectral, Hyperspectral, and Ultraspectral Imagery XXIV*, May 2018, doi: 10.1117/12.2303912.

[17] M. Ramamurthy, Y. H. Robinson, S. Vimal, and A. Suresh, "Auto encoder based dimensionality reduction and classification using convolutional neural networks for hyperspectral images," *Microprocessors and Microsystems*, vol. 79, Nov. 2020, doi: 10.1016/j.micpro.2020.103280.

[18] J. Zhang, L. Chen, L. Zhuo, X. Liang, and J. Li, "An efficient hyperspectral image retrieval method: Deep spectral-spatial feature extraction with DCGAN and dimensionality reduction using t-SNE-based NM hashing," *Remote Sensing*, vol. 10, no. 2, Feb. 2018, doi: 10.3390/rs10020271.

[19] S. Mei, J. Ji, Y. Geng, Z. Zhang, X. Li, and Q. Du, "Unsupervised spatial-spectral feature learning by 3D convolutional autoencoder for hyperspectral classification," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 57, no. 9, pp. 6808–6820, Sep. 2019, doi: 10.1109/TGRS.2019.2908756.

[20] Y. You, J. Cao, and W. Zhou, “A survey of change detection methods based on remote sensing images for multi-source and multi-objective scenarios,” *Remote Sensing*, vol. 12, no. 15, Aug. 2020, doi: 10.3390/RS12152460.

[21] A. Song, J. Choi, Y. Han, and Y. Kim, “Change detection in hyperspectral images using recurrent 3D fully convolutional networks,” *Remote Sensing*, vol. 10, no. 11, Nov. 2018, doi: 10.3390/rs10111827.

[22] S. Prasad and J. Chanussot, “Advances in Computer Vision and Pattern Recognition Hyperspectral Image Analysis Advances in Machine Learning and Signal Processing.” [Online]. Available: http://www.springer.com/series/4205.

[23] S. Lavender and A. Lavender, *Practical Handbook of Remote Sensing*. 2016.

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

[25] F. Chollet, *Deep Learning with Python*, 1st ed. Manning Publisher, 2017.

[26] G. I. Sammut Claude and Webb, Ed., “Discriminative Learning,” in *Encyclopedia of Machine Learning*, Boston, MA: Springer US, 2010, p. 288.

[27] M. Meila, “Machine learning: Discriminative and generative,” *The Mathematical Intelligencer*, vol. 28, no. 1, Dec. 2006, doi: 10.1007/BF02987011.

[28] Z. Tu, “Learning Generative Models via Discriminative Approaches,” Jun. 2007, doi: 10.1109/CVPR.2007.383035.

[29] D. P. Kingma and M. Welling, “An Introduction to Variational Autoencoders,” Jun. 2019, doi: 10.1561/2200000056.

[30] D. Bank, N. Koenigstein, and R. Giryes, “Autoencoders.” 2020.

[31] M. Sahu and R. Dash, “A Survey on Deep Learning: Convolution Neural Network (CNN),” 2021.

[32] Z. Li, W. Yang, S. Peng, and F. Liu, “A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects.” 2020.

[33] M. P. Véstias, “A Survey of Convolutional Neural Networks on Edge with Reconfigurable Computing,” *Algorithms*, vol. 12, no. 8, Jul. 2019, doi: 10.3390/a12080154.

[34] G. van Houdt, C. Mosquera, and G. Nápoles, “A review on the long short-term memory model,” *Artificial Intelligence Review*, vol. 53, no. 8, pp. 5929–5955, 2020, doi: 10.1007/s10462-020-09838-1.

[35] H. Salehinejad, J. Baarbe, S. Sankar, J. Barfett, E. Colak, and S. Valaee, “Recent Advances in Recurrent Neural Networks,” *CoRR*, vol. abs/1801.01078, 2018, [Online]. Available: http://arxiv.org/abs/1801.01078.

[36] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W. Woo, “Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting,” *CoRR*, vol. abs/1506.04214, 2015, [Online]. Available: http://arxiv.org/abs/1506.04214.

[37] Y. Yu, X. Si, C. Hu, and J. Zhang, “A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures,” *Neural Computation*, vol. 31, no. 7, Jul. 2019, doi: 10.1162/neco_a_01199.

[38] K. el Bouchefry and R. S. de Souza, “Learning in Big Data: Introduction to Machine Learning,” in *Knowledge Discovery in Big Data from Astronomy and Earth Observation*, Elsevier, 2020.

[39] L. Filchev, L. Pashova, V. Kolev, and S. Frye, “Surveys, Catalogues, Databases/Archives, and State-of-the-Art Methods for Geoscience Data Processing,” in *Knowledge Discovery in Big Data from Astronomy and Earth Observation*, Elsevier, 2020.

[40] M. Sugiyama, “Linear Dimensionality Reduction,” in *Introduction to Statistical Machine Learning*, Elsevier, 2016.

[41] A. Meyer-Baese and V. Schmid, “Feature Selection and Extraction,” in *Pattern Recognition and Signal Analysis in Medical Imaging*, Elsevier, 2014.

[42] S. Theodoridis and K. Koutroumbas, “Feature Generation I: Data Transformation and Dimensionality Reduction,” in *Pattern Recognition*, Elsevier, 2009.

[43] I. T. Jolliffe and J. Cadima, “Principal component analysis: a review and recent developments,” *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 374, no. 2065, Apr. 2016, doi: 10.1098/rsta.2015.0202.

[44] R. Bro and A. K. Smilde, “Principal component analysis,” *Anal. Methods*, vol. 6, no. 9, pp. 2812–2831, 2014, doi: 10.1039/C3AY41907J.

[45] M. Ringnér, “What is principal component analysis?,” *Nature Biotechnology*, vol. 26, no. 3, pp. 303–304, 2008, doi: 10.1038/nbt0308-303.

[46] N. Mohanty, A. L.-St. John, R. Manmatha, and T. M. Rath, “Chapter 10 - Shape-Based Image Classification and Retrieval,” in *Handbook of Statistics*, vol. 31, C. R. Rao and V. Govindaraju, Eds. Elsevier, 2013, pp. 249–267.

[47] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, “Chapter 8 - Data transformations,” in *Data Mining (Fourth Edition)*, Fourth Edition., I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, Eds. Morgan Kaufmann, 2017, pp. 285–334.

[48] V. H. Ayma, V. A. Ayma, and J. Gutierrez, “DIMENSIONALITY REDUCTION VIA AN ORTHOGONAL AUTOENCODER APPROACH FOR HYPERSPECTRAL IMAGE CLASSIFICATION,” *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XLIII-B3-2020, Aug. 2020, doi: 10.5194/isprs-archives-XLIII-B3-2020-357-2020.

[49] A. Paul and N. Chaki, “Dimensionality Reduction of Hyperspectral Images Using Pooling,” *Pattern Recognition and Image Analysis*, vol. 29, no. 1, pp. 72–78, 2019, doi: 10.1134/S1054661819010085.

[50] C. Zhang, G. Li, S. Du, W. Tan, and F. Gao, “Three-dimensional densely connected convolutional network for hyperspectral remote sensing image classification,”

Journal of Applied Remote Sensing, vol. 13, no. 01, p. 1, 2019, doi: 10.1117/1.jrs.13.016519.

[51] A. Asokan and J. Anitha, "Change detection techniques for remote sensing applications: a survey," *Earth Science Informatics*, vol. 12, no. 2, pp. 143–160, 2019, doi: 10.1007/s12145-019-00380-5.

[52] D. R. Panuju, D. J. Paull, and A. L. Griffin, "Change Detection Techniques Based on Multispectral Images for Investigating Land Cover Dynamics," *Remote Sensing*, vol. 12, no. 11, 2020, doi: 10.3390/rs12111781.

[53] Q. Wang, Z. Yuan, Q. Du, and X. Li, "GETNET: A General End-to-end Two-dimensional CNN Framework for Hyperspectral Image Change Detection," *CoRR*, vol. abs/1905.01662, 2019, [Online]. Available: http://arxiv.org/abs/1905.01662.

[54] M. Graña, M. Veganzons, and B. Ayerdi, "Hyperspectral Remote Sensing Scenes," *http://www.ehu.eus/ccwintco/index.php/Hyperspectral_Remote_Sensing_Scenes*. .

[55] J. Fandiño, "Change Detection Dataset," *https://gitlab.citius.usc.es/hiperespectral/ChangeDetectionDataset*, 2018.

[56] G. E. Hinton, A. Krizhevsky, and S. D. Wang, "Transforming Auto-encoders."

[57] A. G. Howard *et al.*, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," *CoRR*, vol. abs/1704.04861, 2017, [Online]. Available: http://arxiv.org/abs/1704.04861.

[58] W. S. Ahmed and A. a. A. Karim, "The Impact of Filter Size and Number of Filters on Classification Accuracy in CNN," in *2020 International Conference on Computer Science and Software Engineering (CSASE)*, 2020, pp. 88–93, doi: 10.1109/CSASE48920.2020.9142089.

APPENDICES

Appendix A:

Training Log Curve for Dimensionality Reduction:

The training log for the three models on the classification task is given below. The mean absolute error plot using CAE dimensionality reduction for the training and testing tasks is given below.

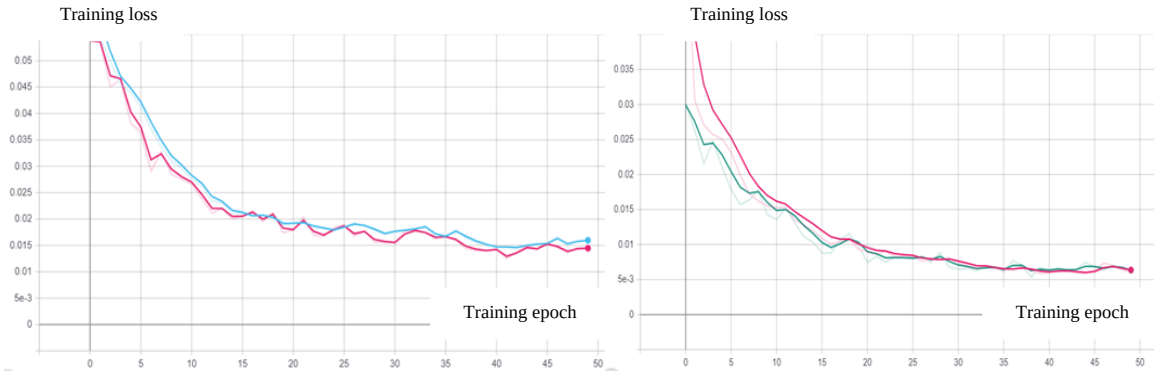


Figure A.1 CAE dimensionality reduction training and testing loss for PaviaC and Salinas (Orange: training loss, validation loss)

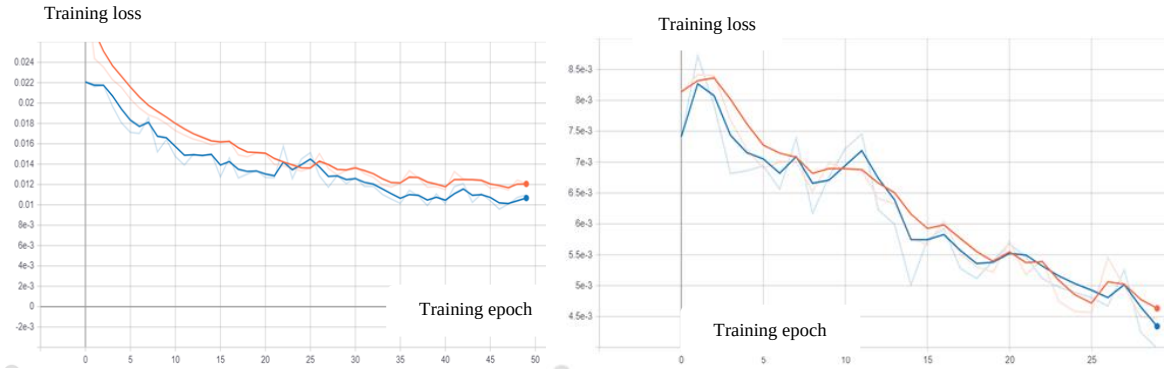


Figure A.2 CAE dimensionality reduction training and testing loss for PaviaC and Salinas (Orange: training loss, validation loss)

Training Log Curve for Dimensionality Reduction:

The training log for the three models on the change detection task using cross-categorical cross entropy and accuracy metrics.

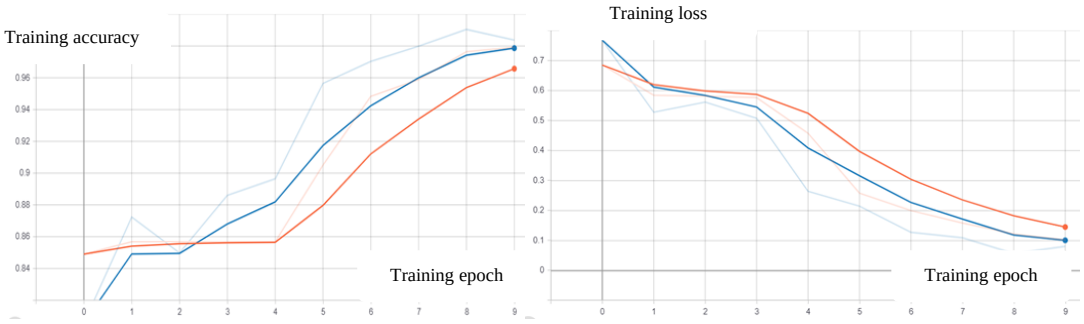


Figure A.3 Change detection training on CAE dimensionality reduction step (Orange: training loss, validation loss)

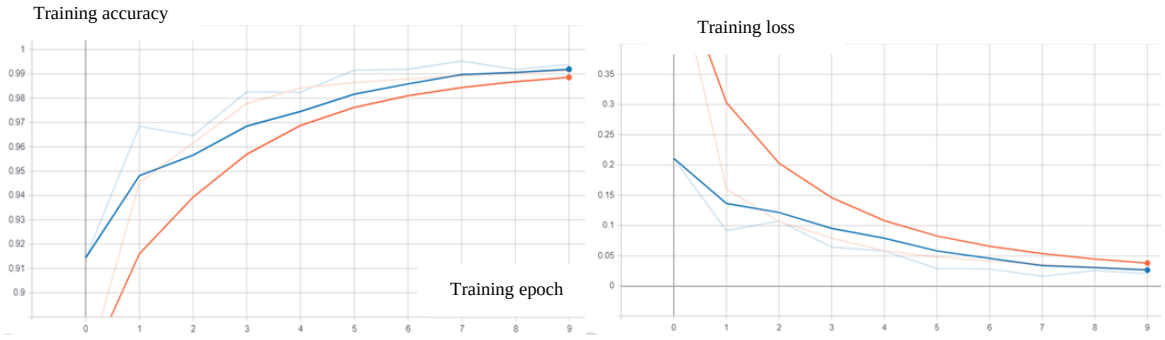


Figure A.4 Change detection training on DNN dimensionality reduction step (Orange: training loss, validation loss)

Appendix B:

Confusion matrix plot for the classification model's performance

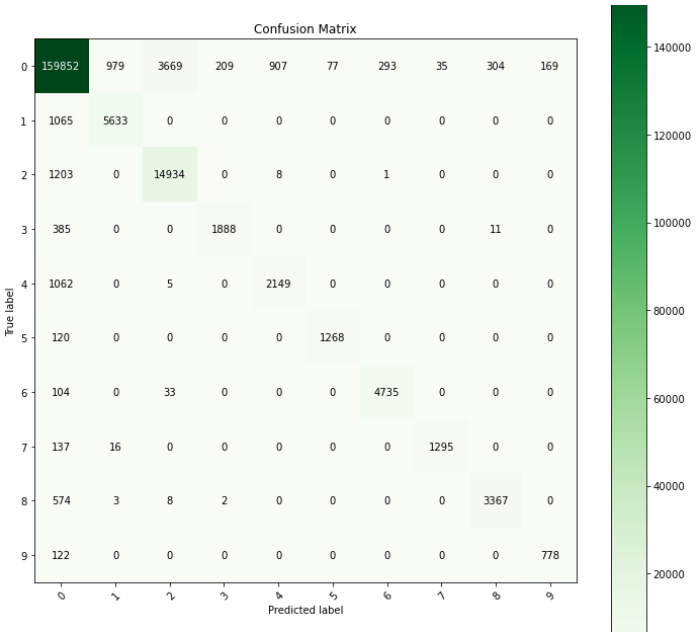


Figure B.1. Confusion matrix for PaviaU training on CAE

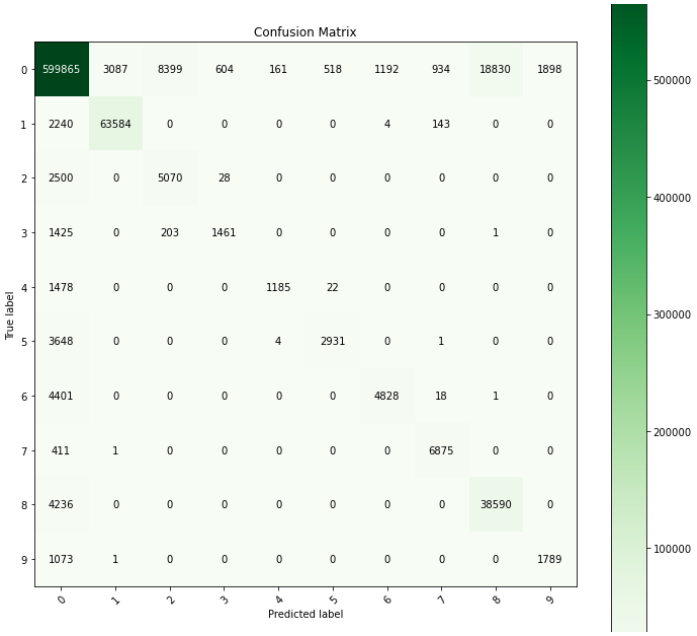


Figure B.2. Confusion matrix for PaviaC training on CAE

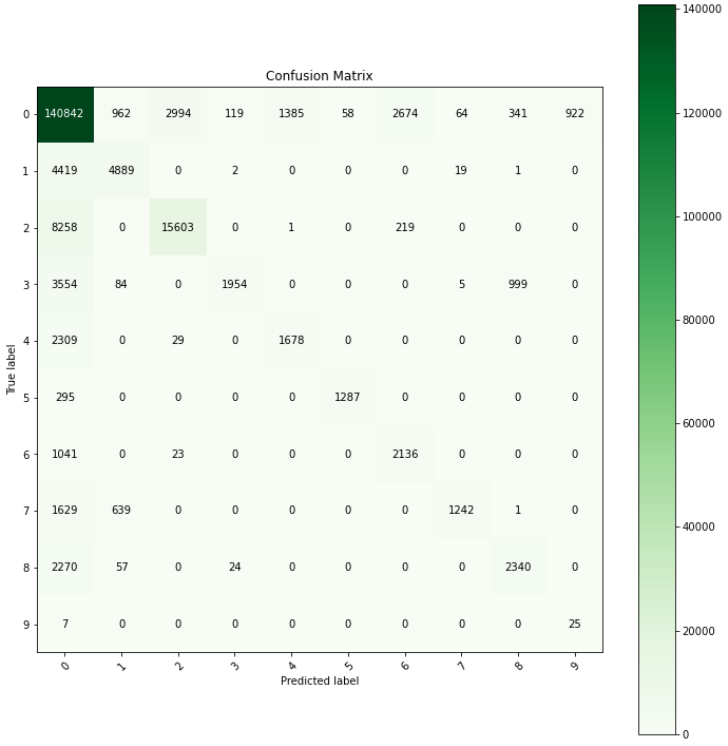


Figure B.3. Confusion matrix for PaviaU training on DNN

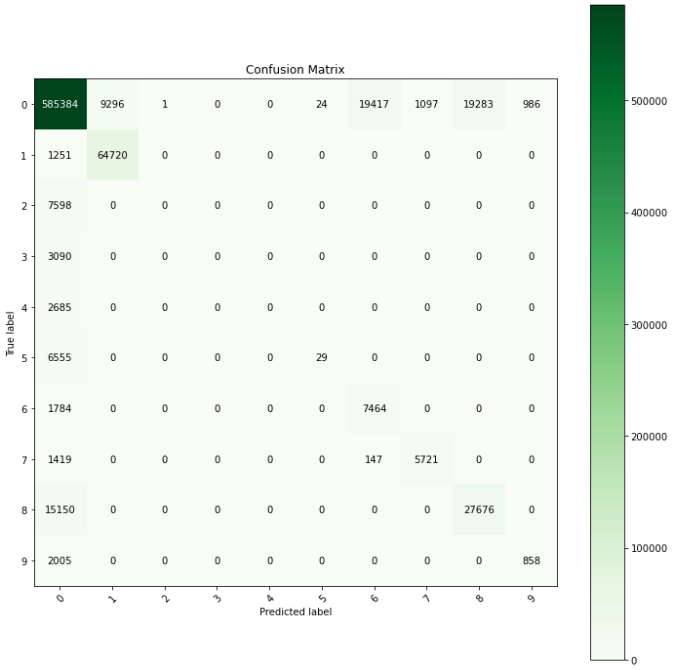


Figure B.3. Confusion matrix for PaviaC training on DNN

Sample training code for the change detection model

Batch generator for the change detection (Coupled input pipeline)

```
def myBatchGenerator(batch_size, images, labels):
    batch_counter = 0
    save_batch = 0
    while True:
        batch_image = []
        batch_label = []
        while batch_counter < batch_size + save_batch:
            #rand_ind = np.random.randint(len(images));#Random index number to access lis

            filename1 = images[batch_counter][0]
            filename2 = images[batch_counter][1]
            theImage1 = np.load(filename1)
            theImage2 = np.load(filename2)
            tempImage = np.zeros((2, 64, 64, theImage1.shape[2]))
            tempImage[0, :, :, :] = theImage1;
            tempImage[1, :, :, :] = theImage2;
            tempImage = dnn_encoder.predict(tempImage)

            labelname = labels[batch_counter]
            theLabel = np.load(labelname)
            #The input image and the batch label should be the same in this case

            batch_image.append(tempImage)
            batch_label.append(theLabel)

            #print(theImage.shape)
            batch_counter = batch_counter + 1
            #print("batch_counter : "+str(batch_counter))
        #print("batch_size + save_batch : "+str(batch_size + save_batch))
        if(batch_counter == len(images)-1 or (batch_size + save_batch) > len(images)-1):
            batch_counter = 0
            save_batch = 0

        save_batch = batch_counter
        BI = np.array(batch_image)#.as...
        BL = np.array(batch_label)#.astype(np.float32);#Batch labe
```

Figure B.4. Batch generator for the change detection model

```
def plot_confusion_matrix(cm, classes, title, normalize=False, cmap=plt.cm.Greens):

    fig, ax = plt.subplots(figsize=(5, 5))
    im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
    ax.figure.colorbar(im, ax=ax)
    # We want to show all ticks...
    ax.set(xticks=np.arange(cm.shape[1]),
           yticks=np.arange(cm.shape[0]),
           # ... and label them with the respective list entries
           xticklabels=classes, yticklabels=classes,
           title=title,
           ylabel='True label',
           xlabel='Predicted label')

    # Rotate the tick labels and set their alignment.
    plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
             rotation_mode="anchor")

    # Loop over data dimensions and create text annotations.
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, format(cm[i, j], fmt),
                    ha="center", va="center",
                    color="white" if cm[i, j] > thresh else "black")
    fig.tight_layout()
    return ax
```

Figure B.5. Confusion matrix plot for change detection model

```
def get_callbacks(name):
  return [
    tfdocs.modeling.EpochDots(),
    tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5),
    tf.keras.callbacks.TensorBoard(logdir/name),
    tf.keras.callbacks.ModelCheckpoint(
        "hermi_ch_detec__dnn_v1.h5", monitor='val_loss', verbose=1, save_best_only=True,
         mode='min', save_freq='epoch')
  ]
```

```
lstm_model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = "accuracy")
```

```
history_v1 = lstm_model.fit(train_batch,
                            steps_per_epoch=92,
                            epochs=10,
                            verbose=1,
                            validation_data=test_batch,
                            validation_steps=1,
                            callbacks = get_callbacks("updated")
                            )
```

```
Epoch 1/10
 1/92 [..............................] - ETA: 0s - loss: 1.7915 - accuracy: 0.0672WARNING:tensorflow:From c:\users\cvr 2019 202
0\anaconda3\envs\tf2\lib\site-packages\tensorflow\python\ops\summary_ops_v2.py:1277: stop (from tensorflow.python.eager.profile
r) is deprecated and will be removed after 2020-07-01.
Instructions for updating:
use `tf.profiler.experimental.stop` instead.
92/92 [==============================] - ETA: 0s - loss: 0.5220 - accuracy: 0.8905
Epoch: 0, accuracy:0.8905,  loss:0.5220,  val_accuracy:0.8996,  val_loss:0.2602,
.
Epoch 00001: val_loss improved from inf to 0.26021, saving model to hermi_ch_detec__dnn_v1.h5
92/92 [==============================] - 168s 2s/step - loss: 0.5220 - accuracy: 0.8905 - val_loss: 0.2602 - val_accuracy: 0.89
```

Figure B.6 Sample code for training and displaying progress on change detection