VISVESVARAYA TECHNOLOGICAL UNIVERSITY

Jnana Sangama, Belgaum-590018

A PROJECT REPORT **(15CSP85)** ON

"Classification of Landcover Using Data Analytics for Hyperspectral Imaging"

Submitted in Partial fulfillment of the Requirements for the Degree of

Bachelor of Engineering in Computer Science & Engineering

By

ANIMESH (1CR16CS018)

PRIYANSHU RAJ (1CR15CS120)

BARSABARAN SAHA (1CR15CS040)

ANIRUDHYA DEB (1CR15CS024)

Under the Guidance of,

PREETHI SHEEBA H.

ASSISTANT PROFESSOR,

Dept. of CSE

CELEBRATING 25 YEARS

CMRIT

CMR INSTITUTE OF TECHNOLOGY, BENGALURU.

ACCREDITED WITH A+ GRADE BY NAAC

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CMR INSTITUTE OF TECHNOLOGY

#132, AECS LAYOUT, IT PARK ROAD, KUNDALAHALLI, BANGALORE-560037

CMR INSTITUTE OF TECHNOLOGY

#132, AECS LAYOUT, IT PARK ROAD, KUNDALAHALLI,BANGALORE-560037

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

Certified that the project work entitled **"Classification of Landcover Using Data Analytics for Hyperspectral Imaging"** carried out by **Mr ANIMESH**, USN **1CR16CS018**, **Ms**. **PRIYANSHU RAJ**, USN **1CR15CS120**,**Mr BARSABARAN SAHA**, USN **1CR15CS040, Mr ANIRUDHYA DEB**, USN **1CR15CS024,** bonafide students of CMR Institute of Technology, in partial fulfillment for the award of **Bachelor of Engineering** in Computer Science and Engineering of the Visvesvaraya Technological University, Belgaum during the year 2019-2020. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the Report deposited in the departmental library.

The project report has been approved as it satisfies the academic requirements in respect of Project work prescribed for the said Degree.

(Preethi Sheeba H.)	**Dr. Prem Kumar Ramesh**	**Dr. Sanjay Jain**
(Assistant Professor)	**Professor & Head**	**Principal**
Dept. of CSE, CMRIT	**Dept. of CSE, CMRIT**	**CMRIT**

DECLARATION

We, the students of Computer Science and Engineering, CMR Institute of Technology, Bangalore declare that the work entitled "**ClassificationofLandcoverUsingDataAnalyticsforHyperspectralImaging”** has been successfully completed under the guidance of Assistant Prof.**Preethi Sheeba H**., Computer Science and Engineering Department, CMR Institute of technology, Bangalore. This dissertation work is submitted in partial fulfillment of the requirements for the award of Degree of Bachelor of Engineering in Computer Science and Engineering during the academic year 2019 - 2020. Further the matter embodied in the project report has not been submitted previously by anybody for the award of any degree or diploma to any university.

Place:

Date:

Team members:

ANIMESH (1CR16CS018) ____________________

PRIYANSHU RAJ (1CR15CS120) ____________________

BARSABARAN SAHA (1CR15CS040) ____________________

ANIRUDHYA DEB (1CR15CS024) ____________________

ABSTRACT

The idea of the project is to segregate and classify a given land cover into its respective classes.
We have taken a hyperspectral image consisting of 145*145 pixels and 224 spectral bands which is required for maximum information to be extracted.
Previous work in this field have been done using various algorithms like SVM, end member extraction etc. which is slower and is a tedious process.
Environmental Monitoring- Hyperspectral imaging is used to track forest health, water quality, and surface contamination. Hyperspectral Image classification is the process of labelling the different landscape features. In our approach, we are using Deep Learning and Neural Networks to train a model and classify an input hyperspectral image. Such classification can help to understand the landscape features of a particular area and this data can be used to predict land usage and suggest optimal use of land. Here, we are using the Indian Pines data set for training and classification. The Deep learning framework used is Tensor Flow and the resultant accuracy in prediction is about 93%.The idea of the project is to segregate and classify a given land cover into its respective classes. We have taken a hyperspectral image consisting of 145*145 pixels and 224 spectral bands which is required for maximum information to be extracted.

ACKNOWLEDGEMENT

The satisfaction and euphoria that accompany a successful completion of any task would be incomplete without the mention of the people who made it possible. So with gratitude we acknowledge all those whose guidance and encouragement served as beacon of light and crowned our effort with success. We would like to thank Dr. Sanjay Jain, Principal, CMRIT who provided us such an opportunity.

Conclusively, we could like to thank all the faculty members who have always been very cooperative and generous and The Head Of Department Dr. Prem Kumar Ramesh, Department of Computer Science & Engineering, CMRIT, Bangalore for giving us the opportunity to delve into the field of Machine Learning.

We consider it a privilege to express our sincere gratitude to our internal guide Mrs. Shashikala K.S, Asst. Professor, Dept. of Computer Science & Engineering, CMRIT, Bangalore for her valuable guidance, suggestions and inputs throughout the tenure of this project.(guide name), designation, Department of Computer Science and Engineering, for the valuable guidance throughout the tenure of this review.

I also extend my thanks to all the faculty of Computer Science and Engineering who directly or indirectly encouraged me.

Finally, I would like to thank my parents and friends for all their moral support they have given me during the completion of this work.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

CNN - Convolutional Neural Network page

AVIRIS - Airborne visible/infrared imaging spectrometer

VHR - Very High Resolution

DTM - Digital Terrain Model

IKONOS - Abbreviation not available

LIDAR - Light Detection and ranging

GIS - Geographic Information Systems

ALTM - Airborne Laser Terrain Mapper

BSP - Binary Space Partitioning

WSL - Weakly Supervised learning

PCA - Principal Component Analysis

HIS - Hyper Spectral Image

CPU - Central Processing Unit

RAM - Random Access Memory

GPU - Graphics Processing Unit

GTX - No Abbreviation available

CUDA - Compute Unified Device Architecture

TF - Tensor Flow

IDE - Integrated Development Environment

QGIS - Quantum Geographic Information Systems

AI - Artificial Intelligence

STC - Standard Test Condition

CHAPTER 1

INTRODUCTION

A hyperspectral image differs from a normal image as it contains n no. of layers meaning more no. of pixels and eventually providing deeper information. Classification is an abstract representation of the situation in the field using well-defined diagnostic criteria: the classifiers.

The idea of the project is building these classifiers using machine and deep leaning models which will ultimately solve our purpose. Recent advances in remote sensing technology have made hyperspectral data with hundreds of narrow contiguous bands more widely available. The hyperspectral data can therefore reveal subtle differences in the spectral signatures of land cover classes that appear similar when viewed by multispectral sensors. If successfully exploited, the hyperspectral data can yield higher classification accuracies and more detailed class taxonomies. However, the task of classifying hyperspectral data also has unique challenges.

The hyperspectral un-mixing problem is concerned with the decomposition of the hyperspectral image into a product form, where the spectrum in each pixel is represented as a collection of material spectra that are referred to as end members, and the mixing proportions of these materials in each pixel that are known as the abundances.

Deep learning is a subfield of machine learning which uses artificial neural networks that is inspired by the structure and function of the human brain. Despite being a very new approach, it has become very popular recently. Deep learning has achieved much higher success in many applications where machine learning has been successful at certain rates. In particular, it is preferred in the classification of big data sets because it can provide fast and efficient results. In this study, we used Tensor flow, one of the most popular deep learning libraries to classify dataset, which is frequently used in data analysis studies. Using Tensor flow, which is an open source artificial intelligence library developed by Google, we have studied and compared the effects of multiple activation functions on classification results. In this Study, Convolutional Neural Network (CNN) is used as deep learning artificial neural network. The applications of hyperspectral image classification are given below:

1) It will help us to identify different areas.

2) Fixing of tax policies by the government by knowing the rate of growth.

1.1 Relevance of the project

The conventional method of machine learning, such as k-nearest-neighbours (KNN), support vector machines (SVMs), random forests (RFs) and so on.
However, these methods often require strong background knowledge of HSI, and the process of extracting features is more troublesome and easy to lose important features.
The greatest advantage of it is that features can be extracted from the hidden layer in the network without too much pre-processing of the data.
Applications of Hyperspectral imaging are like in Pharmaceutical industries Hyperspectral infrared imagers can identify counterfeits, find defects, and eliminate prescription errors.
Hyperspectral imaging enables identification of weeds, monitoring of plant health, and evaluation of ripeness. Early detection of crop stress is a common application.

1.2 Problem Statement

Problem statement therefore is Classification of Land cover using Data Analytics for Hyperspectral Imaging with better accuracy. The idea of the project is to replace existing methodology like SVM which is a traditional machine learning algorithm with low accuracy as it's comparatively slower as compared to Neural Networks (specifically CNN'S) which is a newer approach. Also, being a deep learning framework provides more information and training the model becomes easier. Our idea in this project is to implement 2 layered Convolutional Neural Network.

1.3 Objective

The objectives of the work are as follows:

- Collect Hyperspectral Image data and analyse the data for further processing.
- Perform Pre-processing and data cleaning that will remove the unwanted spectral bands whose processing is not required.
- Design and develop a method for segmentation of land cover from hyperspectral image data.
- Test the effectiveness of the proposed method on various hyperspectral images to classify different land covers.
- Ensure that the new methods meet the particularities of the given data
- Finally we compare our output accuracy percentage with other works to obtain a higher accuracy .

1.4 Scope of the project

The most important part of this project is its usage of classifying land cover. Depending upon the requirements we can further narrow down or reduce the dimensionality for better efficiency. For instance, eliminating water bodies spectral region was our way to reduce dimensionality as we were concerned with the distinguishing of the different agricultural areas. Henceforth, we can see that a single hyperspectral image with its given ground truth can be put to use in different ways, gathering more information contributing to higher accuracy..

- The future scope of the project might be putting the classification into real time usage
- Yield estimation in wheat - Hyperspectral remote sensing was used to help predict yield in wheat as a function of fertilizer concentration.
- Food Analysis- Resonon's hyperspectral imaging systems are used in food research and industry to identify defects, characterize product quality, and locate contaminants.
- Cooked Food- Subtle color changes associated with food quality can readily be identified using hyperspectral imaging.
- Environmental Monitoring- Hyperspectral imaging is used to track forest health, water quality, and surface contamination.
- Further improvement in this project could lead to more accurate results.

1.5 Methodology

We use agile methodology to implement our system. It is a type of project management process, mainly used for software development, where demands and solutions evolve through the collaborative effort of self-organizing and cross functional terms. Thus, we perform the process in steps as described below.

First collect the Indian Pines data in the required format. We need to determine the relevant attributes needed for the prediction of Land cover prediction, this is done by cleaning the dataset by removing the noisy data. After analyzing the problem statement we design the model and identify the best algorithm with the data available. Using the algorithm determined we train the datasets to predict the landcover type. After the implementation of the algorithms we test the results by accuracy calculations.

Story ID	Requirement description	User stories/Task	Description
1	Collection of weather and crop data.	Data Collection	In .csv format.
2	Determining the relevant attributes needed for the prediction of crop production and selection and merging them into a structured form.	Cleaning the dataset.	Removal of noisy data.
3	Prediction of Algorithm	Designing the model.	Identifying the best algorithm with the data available.
4	Training the datasets.	Implementation.	Training the data to predict the crop.
5	Testing the results.	Accuracy calculation.	Calculating the accuracy of algorithm.

Fig 1: Agile Methodology

CHAPTER 2

LITERATURE SURVEY

2.1 <u>Object Detection with Deep Learning: A Review (Paper 1) July 2019</u>

DEEP NEURAL NETWORK: (Overview)

DNN is a type of artificial intelligence that imitates some functions of the person mind. DNN has a

normal tendency for storing experiential knowledge. An DNN consists of a sequence of layers, each

layer consists of a set of neurones. All neurones of every layer are linked by weighted connections to

all neurones on the preceding and succeeding layers

CHARACTERISTICS:

It uses Nonparametric approach. Performance and accuracy depends upon the network structure

and number of inputs.

ADVANTAGES:

It is a non-parametric classifier.

- It is an universal functional approximator with arbitrary accuracy.
- capable to present functions such as OR, AND, NOT
- It is a data driven selfadaptive technique
- efficiently handles noisy inputs

- Computation rate is high

Disadvantages:

- It is semantically poor.
- The training of DNN is time taking.
- Problem of over fitting.
- Difficult in choosing the type network architecture.

2.2 Semi-Supervised Deep Learning Classification for Hyperspectral Image Based on Dual-Strategy Sample Selection(2018)(paper2)

Overview:

Semi-supervised learning is a class of machine learning tasks and techniques that also make

use of unlabelled data for training – typically a small amount of labelled data with a large amount

of unlabelled data. Semi-supervised learning falls between unsupervised learning and supervised

learning

Advantage:

- Capable of reducing the dependence of deep learning method on large-scale manually labelled HSI data. The key to the framework are two parts:

(1) The spectral- and spatial-Network for extracting the spectral features and spatial features and

(2) the dual-strategy sample selection co-training algorithm for effective semi-supervised learning.

Disadvantage:

- It is robust because there are mislabelled samples.

2.3 Classification of Hyperspectral Images by SVM Using a Composite Kernel by Employing Spectral, Spatial and Hierarchical Structure Information (PAPER 3)(MARCH 2018)

SVM (SUPPORT VECTOR MECHANISM): WHAT IS IT AND WHAT ARE THE CHARACTERISTICS?

A support vector machine builds a hyper plane or set of hyper planes in a high- or

Infinite dimensional space, used for classification. Good separation is achieved by the hyper plane

that has the largest distance to the nearest training data point of any class (functional margin),

generally larger the margin lower the generalization error of the classifier.

CHARACTERISTICS:

SVM uses Nonparametric with binary classifier approach and can handle more input data very

efficiently. Performance and accuracy depends upon the hyperplane selection and kernel parameter

Advantages:

- It gains flexibility in the choice of the form of the threshold.
- Contains a nonlinear transformation.
- It provides a good generalization capability.
- The problem of over fitting is eliminated.
- Reduction in computational complexity.
- Simple to manage decision rule complexity and Error frequency

Disadvantages:

- Result transparency is low.
- Training is time consuming.
- Structure of algorithm is difficult to understand
- Determination of optimal parameters is not easy when there is nonlinearly separable training data.

2.4 HYPERSPECTRAL IMAGE ANALYSIS USING END MEMBER EXTRACTION ALGORITHM (March 12, 2015) (PAPER 4)

• Mixed pixels are frequent in remotely sensed hyperspectral images due to insufficient spatial

resolution of the imaging spectrometer, or due to intimate mixing effects.

• The rich spectral resolution available can be used to Unix hyperspectral pixels. Mixed pixels can

also be obtained with high spatial resolution data due to intimate mixtures, this means that increasing the spatial resolution does not solve the problem.

• The mixture problem can be approaches in macroscopic fashion, this means that a few

macroscopic components and their associated abundances should be derived.

• However, intimate mixtures happen at microscopic scales, thus complicating the analysis with

nonlinear mixing effects.

Disadvantages:

• Hyperspectral sensor collects hundreds of bands at different wavelengths. The resulting data

volume often comprises several Gigabytes per flight.

• However the bandwidth of the downlink connection between the sensor and the Earth station is

reduced, which limits the amount of data that can be sent to Earth.

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6480716/ (BASE PAPER)

2.5 Hierarchical Multi-Scale Convolutional Neural Networks for Hyperspectral Image Classification. (Paper 1)- 2019 April (Final paper)(paper 5)

KEYWORDS: hyperspectral image (HSI) classification, convolutional neural networks (CNNs),

Bidirectional LSTM, multi-scale features

Deep neural network is a artificial neural network composed of many layers. Convolutional Neural Networks (CNN) are very similar to ordinary Neural Networks. But instead of connecting all neuron from one layer to a single neuron in the next layer, only a patch of neuron from one layer is connected to a single neuron in the next layer. Its neurons is inspired by the organization of the animal visual cortex.

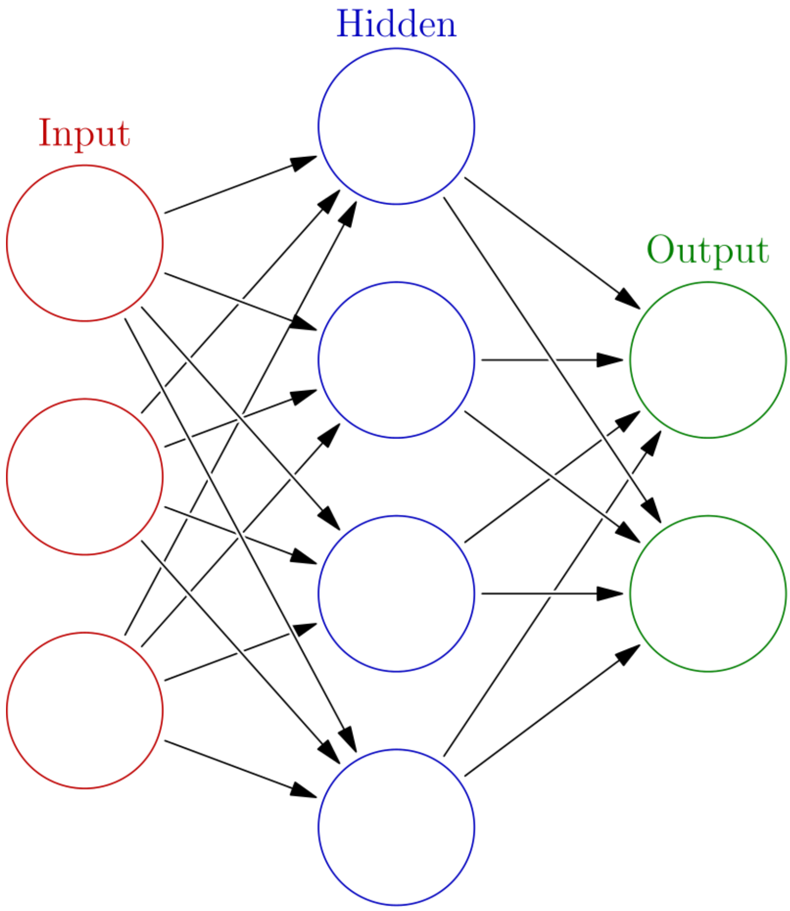


Fig 2: ORDINARY NEURAL NETWORK

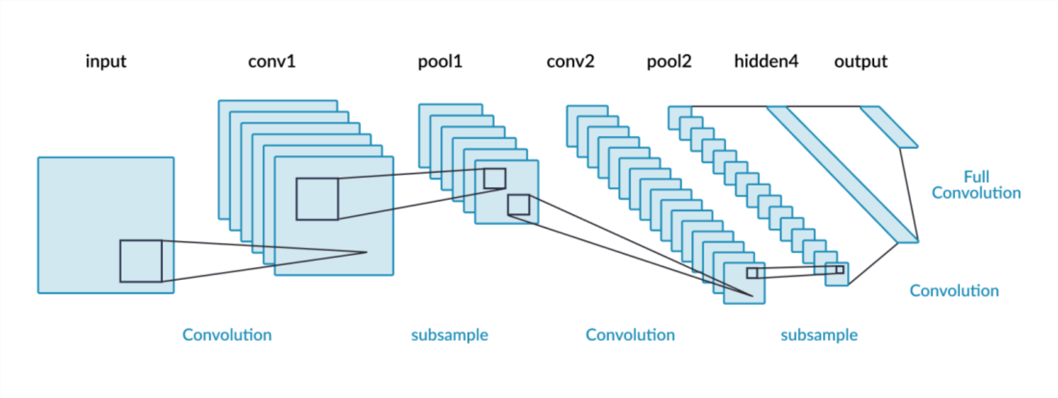


Fig 3: CONVOLUTIONAL NEURAL NETWORK

We can construct a deep neural network with stacked Convolution layers which is called as Deep Convolutional Neural Network. Its been proof that deeper neural network give much more accurate results compared to shallower networks. A convolutional neural network (CNN) is a specific type of artificial neural network that uses perceptrons, a machine learning unit algorithm, for supervised learning, to analyze data. CNNs apply to image processing, natural language processing and other kinds of cognitive tasks. CNNs are regularized versions of multilayer perceptrons. The receptive fields of different neurons partially overlap such that they cover the entire visual field. Convolutional Neural Network is a vast topic that contains many algorithms by which CNN can be applied. It may be just using simple CNN .There are more complex 2D CNN and 3D CNN which gets more complicated as data increases Advantages: Gives amazing results and accuracy. Disadvantages: -High computational cost. - If you don't have a good GPU they are quite slow to train (for complex tasks). -They use to need a lot of training data.

CHAPTER3

REQUIREMENTS SPECIFICATION

The requirements can be broken down into 3 major categories namely functional, hardware and software requirements.

Functional Requirements:

- Functional Requirement defines a function of a software system and how the system must behave when presented with specific inputs or conditions. These may include calculations, data manipulation and processing and other specific functionality. In this system following are the functional requirements:-
- To classify the given landcover of a given HSI image
- To output all the different classes of agricultural crops available.
- To give information about various crops based on wave length based on spectral signature of respective spectral bands.

Hardware Requirements:

The hardware requirement is minimal and the software can run with minimal requirements. The basic requirements are as enlisted below:

1. Processor: Intel Core2Duo processor or a processor with higher specifications
2. Processor speed: 1.5GHz or above.
3. RAM : 1GB or above
4. Storage space : 1GB or above
5. Monitor resolution: A colour monitor with a minimum resolution of 640*480

Software Requirements:

1. An MS-DOS based operating system like Windows 98/2000/XP/Vista/7/8/10/, Linux, MacOS.Anaconda Navigator
2. Python3.6
3. Keras
4. NumPy
5. Pandas
6. MatplotLib
7. SkLearn
8. Spectral
9. Indian Pines dataset, Groundtruth of dataset

CHAPTER 4

SYSTEM ANALYSIS AND DESIGN

4.1 System Architecture

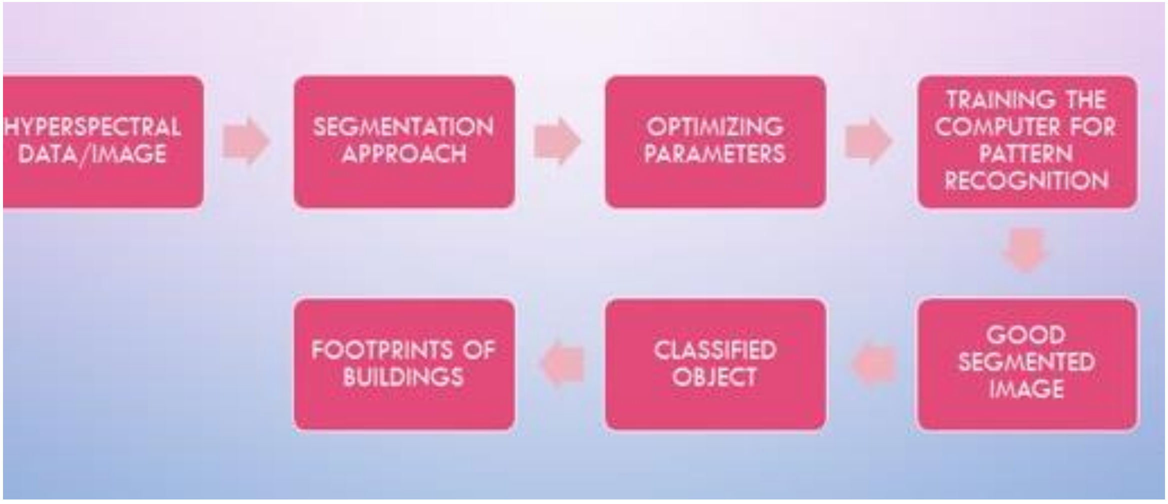


Figure 4: System Architecture and Design

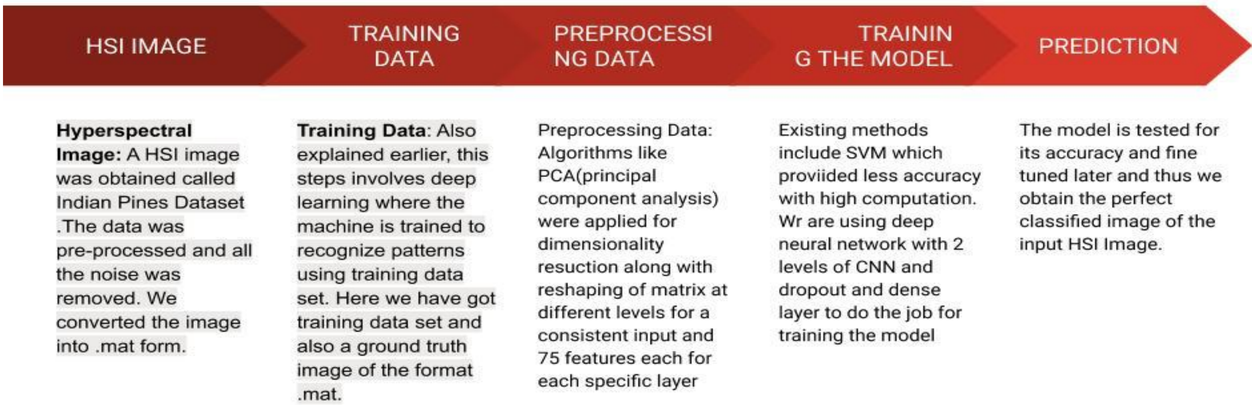


Fig 5: Further Breakdown

4.2 Process Overview

CNNs represent feed-forward neural networks which consist of various combinations of the convolutional layers, max pooling layers, and fully connected layers and exploit spatially local correlation by enforcing a local connectivity pattern between neurons of adjacent layers.

Convolutional layers alternate with max pooling layers mimicking the nature of complex and simple cells in mammalian visual cortex.

A CNN consists of one or more pairs of convolution and max pooling layers and finally ends with a fully connected neural networks. A typical convolutional network architecture is shown on the next slide

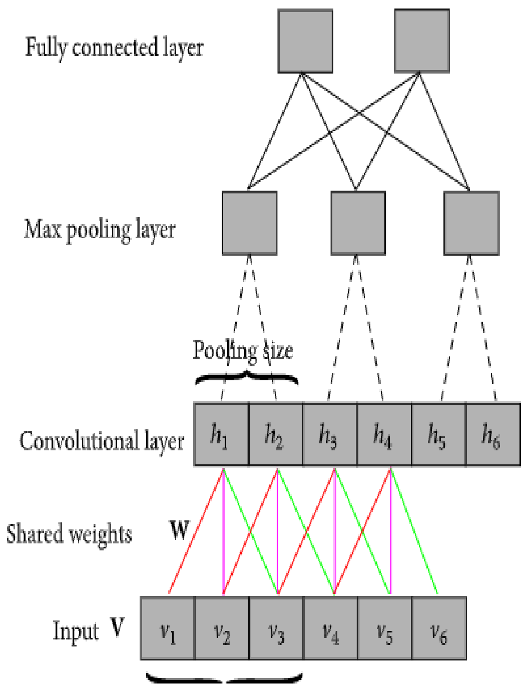


Fig 6: Layers of CNN

<u>Principle Component Analysis:</u>

- Reducing the dimensions of raw input data
- Through a statistical analysis of spectral responses of pixels that belong to the same class, we can observe that the variance of responses is very small. This suggests that pixels that belong to the same class have almost the same values at every channel. At the same time, pixels that belong to different classes present different spectral properties. Based on these characteristics a dimensionality reduction technique can be employed to reduce the dimensionality of the input data in order to speed up the training and prediction processes.

<u>CNN used in our project:</u>

- the classification of each pixel to a predefined number of classes based on their spectral and spatial properties.
- The spectral characteristics are associated with the reflectance properties at every pixel for every spectral band, while spatial information is derived by taking into consideration its neighbours.
- high-level features that encode pixels' spectral and spatial information, are hierarchically constructed using a CNN

- CNNs consist a type of deep models, which apply trainable filters and pooling operations on the raw input, resulting in a hierarchy of increasingly complex features.
- we have to decompose the captured hyperspectral image into patches, each one of which contains spectral and spatial information for a specific pixel.
- The first layer of the proposed CNN is a convolutional layer with C1 = 3 × cr trainable filters of dimension 3 × 3.
- This layer delivers C1 matrices of dimensions 3 × 3 (during convolution we don't take into consideration the border of the patch).
- the first convolutional layer is followed by a second convolutional layer with C2 = 3×C1 trainable filters. Again, the filters are 3×3 matrices.
- The second convolutional layer delivers a vector with C2 elements, which is fed as input to the MLP classifier. The number of MLP hidden units is smaller than the dimensionality of its input.

MODEL ARCHITECTURE:

Fig 7: Breakdown of all The Layers of Neural Network

Other Layers:

Dropout Layer : It is a Simple Way to Prevent Neural Networks from Overfitting.

Since the outputs of a layer under dropout are randomly subsampled, it has the effect of reducing the capacity or thinning the network during training.

Flatten Layer : In between the convolutional layer and the fully connected layer, there is a 'Flatten' layer.

Flattening transforms a two-dimensional matrix of features into a vector that can be fed into a fully connected neural network classifier.

Fully connected Layer : The fully connected (FC) layer in the CNN represents the feature vector for the input.

This feature vector/tensor/layer holds information that is vital to the input.

The convolution layers before the FC layer(s) hold information regarding local features in the input image such as edges, blobs, shapes, etc.

Variables Used:

Variables / Parameters initialised :

windowSize = 5. We are taking 5*5 matrix at a given point of time and then collaborating the output to the desired shape.

Principal component analysis (PCA) is a technique to bring out strong patterns in a dataset by supressing variations. It is used to clean data sets to make it easy to explore and analyse.

The algorithm of Principal Component Analysis is based on a few mathematical ideas namely: Variance and Convariance.

numPCA components = 30 features we want to keep.PCA is used here for dimensionality reduction.

testRatio = 0.25. The dataset has been split into 75:25 ratio(Training: Testing)

Dataset : Indian Pines Dataset with Indian pines ground truth image.

CHAPTER 5

IMPLEMENTATION

5.1 Creating the Datasets in a jupyter notebook

In [1]:

```
importnumpyasnp
fromsklearn.decompositionimport PCA
importscipy.ioassio
fromsklearn.model_selectionimport train_test_split
fromsklearnimport preprocessing
importos
importrandom
fromrandomimport shuffle
fromskimage.transformimport rotate
importscipy.ndimage
```

In [13]:

```
def loadIndianPinesData():
    data_path = os.path.join(os.getcwd(),'Data')
    data = sio.loadmat(os.path.join(data_path,
'Indian_pines_corrected.mat'))['indian_pines_corrected']
    labels = sio.loadmat(os.path.join(data_path, 'Indian_pines_gt.mat'))['indian_pines_gt']

return data, labels

def splitTrainTestSet(X, y, testRatio=0.10):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=testRatio,
random_state=345,
                                                        stratify=y)
return X_train, X_test, y_train, y_test

def oversampleWeakClasses(X, y):
    uniqueLabels, labelCounts = np.unique(y, return_counts=True)
    maxCount = np.max(labelCounts)
    labelInverseRatios = maxCount / labelCounts
# repeat for every label and concat
```

```
    newX = X[y == uniqueLabels[0], :, :, :].repeat(round(labelInverseRatios[0]), axis=0)
    newY = y[y == uniqueLabels[0]].repeat(round(labelInverseRatios[0]), axis=0)
for label, labelInverseRatio in zip(uniqueLabels[1:], labelInverseRatios[1:]):
        cX = X[y== label,:,:,:].repeat(round(labelInverseRatio), axis=0)
        cY = y[y == label].repeat(round(labelInverseRatio), axis=0)
        newX = np.concatenate((newX, cX))
        newY = np.concatenate((newY, cY))
    np.random.seed(seed=42)
    rand_perm = np.random.permutation(newY.shape[0])
    newX = newX[rand_perm, :, :, :]
    newY = newY[rand_perm]
return newX, newY

def standartizeData(X):
    newX = np.reshape(X, (-1, X.shape[2]))
    scaler = preprocessing.StandardScaler().fit(newX)
    newX = scaler.transform(newX)
    newX = np.reshape(newX, (X.shape[0],X.shape[1],X.shape[2]))
return newX, scaler

def applyPCA(X, numComponents=75):
    newX = np.reshape(X, (-1, X.shape[2]))
    pca = PCA(n_components=numComponents, whiten=True)
    newX = pca.fit_transform(newX)
    newX = np.reshape(newX, (X.shape[0],X.shape[1], numComponents))
return newX, pca

def padWithZeros(X, margin=2):
    newX = np.zeros((X.shape[0] + 2 * margin, X.shape[1] + 2* margin, X.shape[2]))
    x_offset = margin
    y_offset = margin
    newX[x_offset:X.shape[0] + x_offset, y_offset:X.shape[1] + y_offset, :] = X
return newX

def createPatches(X, y, windowSize=5, removeZeroLabels = True):
    margin = int((windowSize - 1) / 2)
    zeroPaddedX = padWithZeros(X, margin=margin)
# split patches
    patchesData = np.zeros((X.shape[0] * X.shape[1], windowSize, windowSize,
X.shape[2]))
    patchesLabels = np.zeros((X.shape[0] * X.shape[1]))
    patchIndex = 0
```

```
for r in range(margin, zeroPaddedX.shape[0] - margin):
for c in range(margin, zeroPaddedX.shape[1] - margin):
        patch = zeroPaddedX[r - margin:r + margin + 1, c - margin:c + margin + 1]
        patchesData[patchIndex, :, :, :] = patch
        patchesLabels[patchIndex] = y[r-margin, c-margin]
        patchIndex = patchIndex + 1
if removeZeroLabels:
      patchesData = patchesData[patchesLabels>0,:,:,:]
      patchesLabels = patchesLabels[patchesLabels>0]
      patchesLabels -= 1
return patchesData, patchesLabels


def AugmentData(X_train):
for i in range(int(X_train.shape[0]/2)):
      patch = X_train[i,:,:,:]
      num = random.randint(0,2)
if (num == 0):

        flipped_patch = np.flipud(patch)
if (num == 1):

        flipped_patch = np.fliplr(patch)
if (num == 2):

        no = random.randrange(-180,180,30)
        flipped_patch = scipy.ndimage.interpolation.rotate(patch, no,axes=(1, 0),
                                    reshape=False,       output=None,       order=3,
mode='constant', cval=0.0, prefilter=False)


   patch2 = flipped_patch
   X_train[i,:,:,:] = patch2

return X_train


def savePreprocessedData(X_trainPatches, X_testPatches, y_trainPatches, y_testPatches,
windowSize, wasPCAapplied = False, numPCAComponents = 0, testRatio = 0.25):
if wasPCAapplied:
with open("X_trainPatches_" + str(windowSize) + "PCA" + str(numPCAComponents) +
"testRatio" + str(testRatio) + ".npy", 'bw') as outfile:
```

```
        np.save(outfile, X_trainPatches)
with open("X_testPatches_" + str(windowSize) + "PCA" + str(numPCAComponents) + "testRatio" + str(testRatio) + ".npy", 'bw') as outfile:
        np.save(outfile, X_testPatches)
with open("y_trainPatches_" + str(windowSize) + "PCA" + str(numPCAComponents) + "testRatio" + str(testRatio) + ".npy", 'bw') as outfile:
        np.save(outfile, y_trainPatches)
with open("y_testPatches_" + str(windowSize) + "PCA" + str(numPCAComponents) + "testRatio" + str(testRatio) + ".npy", 'bw') as outfile:
        np.save(outfile, y_testPatches)
else:
with open("../preprocessedData/XtrainWindowSize" + str(windowSize) + ".npy", 'bw') as outfile:
        np.save(outfile, X_trainPatches)
with open("../preprocessedData/XtestWindowSize" + str(windowSize) + ".npy", 'bw') as outfile:
        np.save(outfile, X_testPatches)
with open("../preprocessedData/ytrainWindowSize" + str(windowSize) + ".npy", 'bw') as outfile:
        np.save(outfile, y_trainPatches)
with open("../preprocessedData/ytestWindowSize" + str(windowSize) + ".npy", 'bw') as outfile:
        np.save(outfile, y_testPatches)
```

In [14]:

```
# Load the Global values (windowSize, numPCAcomponents, testRatio) from the text file global_variables.txt
myFile = open('global_variables.txt', 'r')
file = myFile.readlines()[:]


for line in file:

if line[0:3] == "win":

      ds = line.find('=')
      windowSize = int(line[ds+1:-1],10)

elif line[0:3] == "num":

      ds = line.find('=')
      numPCAcomponents = int(line[ds+2:-1],10)
```

```
else:

        ds = line.find('=')
        testRatio = float(line[ds+1:])
```

In [15]:

```
# Global Variables
#numPCAComponents = 30
#windowSize = 5
#testRatio = 0.25
```

In [15]:

```
X, y = loadIndianPinesData()
```

In [16]:

```
X,pca = applyPCA(X,numPCAcomponents)
```

In [17]:

```
XPatches, yPatches = createPatches(X, y, windowSize=windowSize)
```

In [18]:

```
X_train, X_test, y_train, y_test = splitTrainTestSet(XPatches, yPatches, testRatio)
```

In [19]:

```
X_train, y_train = oversampleWeakClasses(X_train, y_train)
```

In [20]:

```
X_train = AugmentData(X_train)
```

In [21]:

```
savePreprocessedData(X_train, X_test, y_train, y_test, windowSize = windowSize,
                wasPCAapplied=True,                 numPCAComponents                 =
numPCAcomponents,testRatio = testRatio)
```

5.2 Train The dataset.ipyb:

train.ipynb: Define and Train the model

In [1]:

```
# Import the necessary libraries

importnumpyasnp

importscipy

importos

fromkeras.modelsimportSequential
```

```
fromkeras.layersimportDense,Dropout,Flatten
fromkeras.layersimportConv2D,MaxPooling2D
fromkeras.optimizersimportSGD
fromkeras.callbacksimportReduceLROnPlateau,ModelCheckpoint
fromkerasimportbackendasK
K.set_image_dim_ordering('th')
fromkeras.utilsimportnp_utils
#from sklearn.cross_validation import StratifiedKFold
Using TensorFlow backend.
In [2]:
# Global Variables
# The number of principal components to be retained in the PCA algorithm,
# the number of retained features  n
numPCAcomponents=30
# Patches windows size
windowSize=5
# The proportion of Test sets
testRatio=0.50
In [3]:
# load Preprocessed data from file
X_train=np.load("./predata/XtrainWindowSize"
+str(windowSize)+"PCA"+str(numPCAcomponents)+
"testRatio"+str(testRatio)+".npy")
y_train=np.load("./predata/ytrainWindowSize"
+str(windowSize)+"PCA"+str(numPCAcomponents)+
"testRatio"+str(testRatio)+".npy")
X_test=np.load("./predata/XtestWindowSize"
+str(windowSize)+"PCA"+str(numPCAcomponents)+
```

```
"testRatio"+str(testRatio)+".npy")
y_test=np.load("./predata/ytestWindowSize"
+str(windowSize)+"PCA"+str(numPCAcomponents)+
"testRatio"+str(testRatio)+".npy")
In [4]:
# Reshape data into (numberofsumples, channels, height, width)
X_train=np.reshape(X_train,(X_train.shape[0],X_train.shape[3],
X_train.shape[1],X_train.shape[2]))
X_test=np.reshape(X_test,(X_test.shape[0],X_test.shape[3],
X_test.shape[1],X_test.shape[2]))

# convert class labels to on-hot encoding
y_train=np_utils.to_categorical(y_train)
y_test=np_utils.to_categorical(y_test)

# Define the input shape
input_shape=X_train[0].shape
print(input_shape)

# number of filters
C1=3*numPCAcomponents
(30, 5, 5)
In [5]:
# Define the model structure
model=Sequential()

model.add(Conv2D(C1,(3,3),activation='relu',input_shape=input_shape))
model.add(Conv2D(3*C1,(3,3),activation='relu'))
```

```
model.add(Dropout(0.25))


model.add(Flatten())
model.add(Dense(6*numPCAcomponents,activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(16,activation='softmax'))
```

In [6]:

```
# Define optimization and train method
reduce_lr=ReduceLROnPlateau(monitor='val_acc',factor=0.9,patience=25,
min_lr=0.000001,verbose=1)
checkpointer=ModelCheckpoint(filepath="checkpoint.hdf5",verbose=1,
save_best_only=False)
sgd=SGD(lr=0.001,decay=1e-6,momentum=0.9,nesterov=True)
model.compile(loss='categorical_crossentropy',optimizer=sgd,
metrics=['accuracy'])
```

In [7]:

```
# Start to train model
history=model.fit(X_train,y_train,
batch_size=32,
epochs=100,
verbose=1,
validation_data=(X_test,y_test),
callbacks=[reduce_lr,checkpointer],
shuffle=True)
```

WARNING:tensorflow:Variable *= will be deprecated. Use variable.assign_mul if you want assignment to the variable value or 'x = x * y' if you want a new python Tensor object.

Train on 20110 samples, validate on 5183 samples

Epoch 1/100

20110/20110 [==============================] - 5s 233us/step - loss: 1.2813 - acc: 0.6164 - val_loss: 0.6084 - val_acc: 0.8057

Epoch 00001: saving model to checkpoint.hdf5

Epoch 2/100

20110/20110 [==============================] - 4s 177us/step - loss: 0.3752 - acc: 0.8783 - val_loss: 0.3269 - val_acc: 0.8956

Epoch 00002: saving model to checkpoint.hdf5

Epoch 3/100

20110/20110 [==============================] - 4s 175us/step - loss: 0.2231 - acc: 0.9304 - val_loss: 0.2492 - val_acc: 0.9168

Epoch 00003: saving model to checkpoint.hdf5

Epoch 4/100

20110/20110 [==============================] - 4s 175us/step - loss: 0.1534 - acc: 0.9529 - val_loss: 0.1856 - val_acc: 0.9429

Epoch 00004: saving model to checkpoint.hdf5

Epoch 5/100

20110/20110 [==============================] - 4s 174us/step - loss: 0.1112 - acc: 0.9662 - val_loss: 0.1563 - val_acc: 0.9518

Epoch 00005: saving model to checkpoint.hdf5

Epoch 6/100

20110/20110 [==============================] - 4s 176us/step - loss: 0.0856 - acc: 0.9748 - val_loss: 0.1393 - val_acc: 0.9510

Epoch 00006: saving model to checkpoint.hdf5

Epoch 7/100

20110/20110 [==============================] - 4s 177us/step - loss: 0.0659 - acc: 0.9807 - val_loss: 0.1164 - val_acc: 0.9628

Epoch 00007: saving model to checkpoint.hdf5

Epoch 8/100

20110/20110 [==============================] - 4s 174us/step - loss: 0.0513 - acc: 0.9866 - val_loss: 0.1127 - val_acc: 0.9618

Epoch 00008: saving model to checkpoint.hdf5

Epoch 9/100

20110/20110 [==============================] - 3s 171us/step - loss: 0.0418 - acc: 0.9889 - val_loss: 0.1063 - val_acc: 0.9637

Epoch 00009: saving model to checkpoint.hdf5

Epoch 10/100

20110/20110 [==============================] - 3s 169us/step - loss: 0.0372 - acc: 0.9902 - val_loss: 0.0968 - val_acc: 0.9689

Epoch 00010: saving model to checkpoint.hdf5

Epoch 11/100

20110/20110 [==============================] - 3s 172us/step - loss: 0.0324 - acc: 0.9912 - val_loss: 0.0882 - val_acc: 0.9714

Epoch 00011: saving model to checkpoint.hdf5

Epoch 12/100

20110/20110 [==============================] - 3s 170us/step - loss: 0.0278 - acc: 0.9933 - val_loss: 0.0910 - val_acc: 0.9703

Epoch 00012: saving model to checkpoint.hdf5

Epoch 13/100

20110/20110 [==============================] - 4s 179us/step - loss: 0.0232 - acc: 0.9946 - val_loss: 0.0852 - val_acc: 0.9730

Epoch 00013: saving model to checkpoint.hdf5

Epoch 14/100

20110/20110 [==============================] - 4s 177us/step - loss: 0.0216 - acc: 0.9950 - val_loss: 0.0834 - val_acc: 0.9728

Epoch 00014: saving model to checkpoint.hdf5

Epoch 15/100

20110/20110 [==============================] - 3s 173us/step - loss: 0.0183 - acc: 0.9956 - val_loss: 0.0842 - val_acc: 0.9726

Epoch 00015: saving model to checkpoint.hdf5

Epoch 16/100

20110/20110 [==============================] - 3s 173us/step - loss: 0.0164 - acc: 0.9966 - val_loss: 0.0833 - val_acc: 0.9751

Epoch 00016: saving model to checkpoint.hdf5

Epoch 17/100

20110/20110 [==============================] - 3s 171us/step - loss: 0.0140 - acc: 0.9973 - val_loss: 0.0827 - val_acc: 0.9732

Epoch 00017: saving model to checkpoint.hdf5

Epoch 18/100

20110/20110 [==============================] - 3s 170us/step - loss: 0.0125 - acc: 0.9975 - val_loss: 0.0805 - val_acc: 0.9753

Epoch 00018: saving model to checkpoint.hdf5

Epoch 19/100

20110/20110 [==============================] - 3s 173us/step - loss: 0.0122 - acc: 0.9975 - val_loss: 0.0789 - val_acc: 0.9745

Epoch 00019: saving model to checkpoint.hdf5

Epoch 20/100

20110/20110 [==============================] - 3s 173us/step - loss: 0.0121 - acc: 0.9976 - val_loss: 0.0768 - val_acc: 0.9753

Epoch 00020: saving model to checkpoint.hdf5

Epoch 21/100

20110/20110 [==============================] - 4s 182us/step - loss: 0.0111 - acc: 0.9976 - val_loss: 0.0776 - val_acc: 0.9757

Epoch 00021: saving model to checkpoint.hdf5

Epoch 22/100

20110/20110 [==============================] - 4s 176us/step - loss: 0.0104 - acc: 0.9975 - val_loss: 0.0781 - val_acc: 0.9749

Epoch 00022: saving model to checkpoint.hdf5

Epoch 23/100

20110/20110 [==============================] - 3s 174us/step - loss: 0.0086 - acc: 0.9985 - val_loss: 0.0771 - val_acc: 0.9759

Epoch 00023: saving model to checkpoint.hdf5

Epoch 24/100

20110/20110 [==============================] - 3s 171us/step - loss: 0.0082 - acc: 0.9986 - val_loss: 0.0761 - val_acc: 0.9755

Epoch 00024: saving model to checkpoint.hdf5

Epoch 25/100

20110/20110 [==============================] - 3s 170us/step - loss: 0.0080 - acc: 0.9986 - val_loss: 0.0785 - val_acc: 0.9761

Epoch 00025: saving model to checkpoint.hdf5

Epoch 26/100

20110/20110 [==============================] - 3s 170us/step - loss: 0.0082 - acc: 0.9983 - val_loss: 0.0771 - val_acc: 0.9776

Epoch 00026: saving model to checkpoint.hdf5

Epoch 27/100

20110/20110 [==============================] - 3s 170us/step - loss: 0.0071 - acc: 0.9988 - val_loss: 0.0770 - val_acc: 0.9778

Epoch 00027: saving model to checkpoint.hdf5

Epoch 28/100

20110/20110 [==============================] - 3s 172us/step - loss: 0.0069 - acc: 0.9987 - val_loss: 0.0773 - val_acc: 0.9782

Epoch 00028: saving model to checkpoint.hdf5

Epoch 29/100

20110/20110 [==============================] - 3s 171us/step - loss: 0.0059 - acc: 0.9993 - val_loss: 0.0783 - val_acc: 0.9770

Epoch 00029: saving model to checkpoint.hdf5

Epoch 30/100

20110/20110 [==============================] - 3s 169us/step - loss: 0.0057 - acc: 0.9991 - val_loss: 0.0783 - val_acc: 0.9757

Epoch 00030: saving model to checkpoint.hdf5

Epoch 31/100

20110/20110 [==============================] - 3s 169us/step - loss: 0.0052 - acc: 0.9993 - val_loss: 0.0789 - val_acc: 0.9765

Epoch 00031: saving model to checkpoint.hdf5

Epoch 32/100

20110/20110 [==============================] - 3s 171us/step - loss: 0.0051 - acc: 0.9994 - val_loss: 0.0775 - val_acc: 0.9774

Epoch 00032: saving model to checkpoint.hdf5

Epoch 33/100

20110/20110 [==============================] - 3s 168us/step - loss: 0.0054 - acc: 0.9989 - val_loss: 0.0767 - val_acc: 0.9778

Epoch 00033: saving model to checkpoint.hdf5

Epoch 34/100

20110/20110 [==============================] - 3s 169us/step - loss: 0.0049 - acc: 0.9993 - val_loss: 0.0773 - val_acc: 0.9776

Epoch 00034: saving model to checkpoint.hdf5

Epoch 35/100

20110/20110 [==============================] - 3s 170us/step - loss: 0.0052 - acc: 0.9991 - val_loss: 0.0771 - val_acc: 0.9786

Epoch 00035: saving model to checkpoint.hdf5

Epoch 36/100

20110/20110 [==============================] - 3s 168us/step - loss: 0.0045 - acc: 0.9992 - val_loss: 0.0751 - val_acc: 0.9782

Epoch 00036: saving model to checkpoint.hdf5

Epoch 37/100

20110/20110 [==============================] - 3s 167us/step - loss: 0.0046 - acc: 0.9993 - val_loss: 0.0752 - val_acc: 0.9776

Epoch 00037: saving model to checkpoint.hdf5

Epoch 38/100

20110/20110 [==============================] - 3s 169us/step - loss: 0.0038 - acc: 0.9993 - val_loss: 0.0763 - val_acc: 0.9782

Epoch 00038: saving model to checkpoint.hdf5

Epoch 39/100

20110/20110 [==============================] - 3s 168us/step - loss: 0.0041 - acc: 0.9993 - val_loss: 0.0766 - val_acc: 0.9780

Epoch 00039: saving model to checkpoint.hdf5

Epoch 40/100

20110/20110 [==============================] - 3s 168us/step - loss: 0.0044 - acc: 0.9989 - val_loss: 0.0763 - val_acc: 0.9794

Epoch 00040: saving model to checkpoint.hdf5

Epoch 41/100

20110/20110 [==============================] - 3s 164us/step - loss: 0.0040 - acc: 0.9995 - val_loss: 0.0755 - val_acc: 0.9774

Epoch 00041: saving model to checkpoint.hdf5

Epoch 42/100

20110/20110 [==============================] - 3s 164us/step - loss: 0.0038 - acc: 0.9992 - val_loss: 0.0741 - val_acc: 0.9782

Epoch 00042: saving model to checkpoint.hdf5

Epoch 43/100

20110/20110 [==============================] - 3s 172us/step - loss: 0.0032 - acc: 0.9997 - val_loss: 0.0762 - val_acc: 0.9772

Epoch 00043: saving model to checkpoint.hdf5

Epoch 44/100

20110/20110 [==============================] - 4s 189us/step - loss: 0.0034 - acc: 0.9995 - val_loss: 0.0752 - val_acc: 0.9784

Epoch 00044: saving model to checkpoint.hdf5

Epoch 45/100

20110/20110 [==============================] - 4s 190us/step - loss: 0.0032 - acc: 0.9995 - val_loss: 0.0770 - val_acc: 0.9788

Epoch 00045: saving model to checkpoint.hdf5

Epoch 46/100

20110/20110 [==============================] - 4s 191us/step - loss: 0.0029 - acc: 0.9998 - val_loss: 0.0768 - val_acc: 0.9776

Epoch 00046: saving model to checkpoint.hdf5

Epoch 47/100

20110/20110 [==============================] - 4s 189us/step - loss: 0.0034 - acc: 0.9995 - val_loss: 0.0767 - val_acc: 0.9774

Epoch 00047: saving model to checkpoint.hdf5

Epoch 48/100

20110/20110 [==============================] - 4s 189us/step - loss: 0.0038 - acc: 0.9994 - val_loss: 0.0786 - val_acc: 0.9778

Epoch 00048: saving model to checkpoint.hdf5

Epoch 49/100

20110/20110 [==============================] - 4s 188us/step - loss: 0.0034 - acc: 0.9995 - val_loss: 0.0788 - val_acc: 0.9780

Epoch 00049: saving model to checkpoint.hdf5

Epoch 50/100

20110/20110 [==============================] - 4s 188us/step - loss: 0.0028 - acc: 0.9995 - val_loss: 0.0785 - val_acc: 0.9784

Epoch 00050: saving model to checkpoint.hdf5

Epoch 51/100

20110/20110 [==============================] - 4s 180us/step - loss: 0.0028 - acc: 0.9995 - val_loss: 0.0763 - val_acc: 0.9788

Epoch 00051: saving model to checkpoint.hdf5

Epoch 52/100

20110/20110 [==============================] - 3s 171us/step - loss: 0.0028 - acc: 0.9995 - val_loss: 0.0765 - val_acc: 0.9782

Epoch 00052: saving model to checkpoint.hdf5

Epoch 53/100

20110/20110 [==============================] - 3s 170us/step - loss: 0.0022 - acc: 0.9998 - val_loss: 0.0781 - val_acc: 0.9788

Epoch 00053: saving model to checkpoint.hdf5

Epoch 54/100

20110/20110 [==============================] - 3s 170us/step - loss: 0.0023 - acc: 0.9998 - val_loss: 0.0771 - val_acc: 0.9790

Epoch 00054: saving model to checkpoint.hdf5

Epoch 55/100

20110/20110 [==============================] - 3s 170us/step - loss: 0.0024 - acc: 0.9996 - val_loss: 0.0776 - val_acc: 0.9784

Epoch 00055: saving model to checkpoint.hdf5

Epoch 56/100

20110/20110 [==============================] - 3s 169us/step - loss: 0.0024 - acc: 0.9997 - val_loss: 0.0774 - val_acc: 0.9790

Epoch 00056: saving model to checkpoint.hdf5

Epoch 57/100

20110/20110 [==============================] - 3s 171us/step - loss: 0.0022 - acc: 0.9999 - val_loss: 0.0786 - val_acc: 0.9784

Epoch 00057: saving model to checkpoint.hdf5

Epoch 58/100

20110/20110 [==============================] - 3s 171us/step - loss: 0.0023 - acc: 0.9998 - val_loss: 0.0784 - val_acc: 0.9786

Epoch 00058: saving model to checkpoint.hdf5

Epoch 59/100

20110/20110 [==============================] - 3s 169us/step - loss: 0.0025 - acc: 0.9993 - val_loss: 0.0788 - val_acc: 0.9784

Epoch 00059: saving model to checkpoint.hdf5

Epoch 60/100

20110/20110 [==============================] - 3s 168us/step - loss: 0.0025 - acc: 0.9995 - val_loss: 0.0788 - val_acc: 0.9788

Epoch 00060: saving model to checkpoint.hdf5

Epoch 61/100

20110/20110 [==============================] - 3s 170us/step - loss: 0.0020 - acc: 0.9997 - val_loss: 0.0788 - val_acc: 0.9792

Epoch 00061: saving model to checkpoint.hdf5

Epoch 62/100

20110/20110 [==============================] - 3s 170us/step - loss: 0.0022 - acc: 0.9998 - val_loss: 0.0781 - val_acc: 0.9786

Epoch 00062: saving model to checkpoint.hdf5

Epoch 63/100

20110/20110 [==============================] - 3s 169us/step - loss: 0.0021 - acc: 0.9997 - val_loss: 0.0767 - val_acc: 0.9788

Epoch 00063: saving model to checkpoint.hdf5

Epoch 64/100

20110/20110 [==============================] - 3s 169us/step - loss: 0.0021 - acc: 0.9998 - val_loss: 0.0770 - val_acc: 0.9797

Epoch 00064: saving model to checkpoint.hdf5

Epoch 65/100

20110/20110 [==============================] - 3s 170us/step - loss: 0.0019 - acc: 0.9997 - val_loss: 0.0769 - val_acc: 0.9792

Epoch 00065: saving model to checkpoint.hdf5

Epoch 66/100

20110/20110 [==============================] - 3s 169us/step - loss: 0.0017 - acc: 0.9998 - val_loss: 0.0768 - val_acc: 0.9788

Epoch 00066: saving model to checkpoint.hdf5

Epoch 67/100

20110/20110 [==============================] - 3s 170us/step - loss: 0.0019 - acc: 0.9997 - val_loss: 0.0789 - val_acc: 0.9792

Epoch 00067: saving model to checkpoint.hdf5

Epoch 68/100

20110/20110 [==============================] - 3s 168us/step - loss: 0.0022 - acc: 0.9997 - val_loss: 0.0795 - val_acc: 0.9794

Epoch 00068: saving model to checkpoint.hdf5

Epoch 69/100

20110/20110 [==============================] - 3s 168us/step - loss: 0.0021 - acc: 0.9997 - val_loss: 0.0802 - val_acc: 0.9786

Epoch 00069: saving model to checkpoint.hdf5

Epoch 70/100

20110/20110 [==============================] - 3s 169us/step - loss: 0.0020 - acc: 0.9996 - val_loss: 0.0765 - val_acc: 0.9797

Epoch 00070: saving model to checkpoint.hdf5

Epoch 71/100

20110/20110 [==============================] - 3s 167us/step - loss: 0.0021 - acc: 0.9997 - val_loss: 0.0763 - val_acc: 0.9803

Epoch 00071: saving model to checkpoint.hdf5

Epoch 72/100

20110/20110 [==============================] - 3s 168us/step - loss: 0.0015 - acc: 0.9999 - val_loss: 0.0755 - val_acc: 0.9795

Epoch 00072: saving model to checkpoint.hdf5

Epoch 73/100

20110/20110 [==============================] - 3s 168us/step - loss: 0.0017 - acc: 0.9997 - val_loss: 0.0755 - val_acc: 0.9790

Epoch 00073: saving model to checkpoint.hdf5

Epoch 74/100

20110/20110 [==============================] - 3s 168us/step - loss: 0.0017 - acc: 0.9997 - val_loss: 0.0766 - val_acc: 0.9795

Epoch 00074: saving model to checkpoint.hdf5

Epoch 75/100

20110/20110 [==============================] - 3s 166us/step - loss: 0.0019 - acc: 0.9998 - val_loss: 0.0762 - val_acc: 0.9794

Epoch 00075: saving model to checkpoint.hdf5

Epoch 76/100

20110/20110 [==============================] - 3s 160us/step - loss: 0.0015 - acc: 0.9999 - val_loss: 0.0779 - val_acc: 0.9792

Epoch 00076: saving model to checkpoint.hdf5

Epoch 77/100

20110/20110 [==============================] - 3s 164us/step - loss: 0.0014 - acc: 0.9999 - val_loss: 0.0804 - val_acc: 0.9786

Epoch 00077: saving model to checkpoint.hdf5

Epoch 78/100

20110/20110 [==============================] - 4s 188us/step - loss: 0.0014 - acc: 0.9999 - val_loss: 0.0794 - val_acc: 0.9794

Epoch 00078: saving model to checkpoint.hdf5

Epoch 79/100

20110/20110 [==============================] - 3s 172us/step - loss: 0.0016 - acc: 0.9998 - val_loss: 0.0817 - val_acc: 0.9784

Epoch 00079: saving model to checkpoint.hdf5

Epoch 80/100

20110/20110 [==============================] - 4s 207us/step - loss: 0.0016 - acc: 0.9998 - val_loss: 0.0794 - val_acc: 0.9782

Epoch 00080: saving model to checkpoint.hdf5

Epoch 81/100

20110/20110 [==============================] - 3s 169us/step - loss: 0.0013 - acc: 1.0000 - val_loss: 0.0791 - val_acc: 0.9797

Epoch 00081: saving model to checkpoint.hdf5

Epoch 82/100

20110/20110 [==============================] - 3s 168us/step - loss: 0.0020 - acc: 0.9995 - val_loss: 0.0795 - val_acc: 0.9788

Epoch 00082: saving model to checkpoint.hdf5

Epoch 83/100

20110/20110 [==============================] - 4s 175us/step - loss: 0.0014 - acc: 0.9998 - val_loss: 0.0781 - val_acc: 0.9795

Epoch 00083: saving model to checkpoint.hdf5

Epoch 84/100

20110/20110 [==============================] - 3s 166us/step - loss: 0.0017 - acc: 0.9996 - val_loss: 0.0773 - val_acc: 0.9797

Epoch 00084: saving model to checkpoint.hdf5

Epoch 85/100

20110/20110 [==============================] - 3s 173us/step - loss: 0.0012 - acc: 0.9999 - val_loss: 0.0777 - val_acc: 0.9795

Epoch 00085: saving model to checkpoint.hdf5

Epoch 86/100

20110/20110 [==============================] - 3s 170us/step - loss: 0.0015 - acc: 0.9998 - val_loss: 0.0783 - val_acc: 0.9790

Epoch 00086: saving model to checkpoint.hdf5

Epoch 87/100

20110/20110 [==============================] - 3s 165us/step - loss: 0.0016 - acc: 0.9996 - val_loss: 0.0803 - val_acc: 0.9788

Epoch 00087: saving model to checkpoint.hdf5

Epoch 88/100

20110/20110 [==============================] - 4s 174us/step - loss: 0.0015 - acc: 0.9998 - val_loss: 0.0802 - val_acc: 0.9790

Epoch 00088: saving model to checkpoint.hdf5

Epoch 89/100

20110/20110 [==============================] - 4s 177us/step - loss: 0.0011 - acc: 0.9999 - val_loss: 0.0796 - val_acc: 0.9794

Epoch 00089: saving model to checkpoint.hdf5

Epoch 90/100

20110/20110 [==============================] - 4s 175us/step - loss: 0.0014 - acc: 0.9998 - val_loss: 0.0782 - val_acc: 0.9792

Epoch 00090: saving model to checkpoint.hdf5

Epoch 91/100

20110/20110 [==============================] - 3s 174us/step - loss: 0.0014 - acc: 0.9998 - val_loss: 0.0797 - val_acc: 0.9786

Epoch 00091: saving model to checkpoint.hdf5

Epoch 92/100

20110/20110 [==============================] - 3s 172us/step - loss: 0.0014 - acc: 0.9999 - val_loss: 0.0797 - val_acc: 0.9790

Epoch 00092: saving model to checkpoint.hdf5

Epoch 93/100

20110/20110 [==============================] - 3s 172us/step - loss: 0.0013 - acc: 0.9998 - val_loss: 0.0813 - val_acc: 0.9792

Epoch 00093: saving model to checkpoint.hdf5

Epoch 94/100

20110/20110 [==============================] - 3s 172us/step - loss: 0.0012 - acc: 0.9999 - val_loss: 0.0790 - val_acc: 0.9799

Epoch 00094: saving model to checkpoint.hdf5

Epoch 95/100

20110/20110 [==============================] - 3s 173us/step - loss: 0.0011 - acc: 0.9998 - val_loss: 0.0781 - val_acc: 0.9801

Epoch 00095: saving model to checkpoint.hdf5

Epoch 96/100

20110/20110 [==============================] - 3s 170us/step - loss: 0.0012 - acc: 0.9998 - val_loss: 0.0787 - val_acc: 0.9801

Epoch 00096: saving model to checkpoint.hdf5

Epoch 97/100

20110/20110 [==============================] - 3s 170us/step - loss: 0.0014 - acc: 0.9998 - val_loss: 0.0787 - val_acc: 0.9792

Epoch 00097: ReduceLROnPlateau reducing learning rate to 0.0009000000427477062.

Epoch 00097: saving model to checkpoint.hdf5

Epoch 98/100

20110/20110 [==============================] - 3s 172us/step - loss: 0.0012 - acc: 0.9998 - val_loss: 0.0801 - val_acc: 0.9795

Epoch 00098: saving model to checkpoint.hdf5

Epoch 99/100

20110/20110 [==============================] - 3s 172us/step - loss: 0.0010 - acc: 1.0000 - val_loss: 0.0798 - val_acc: 0.9794

Epoch 00099: saving model to checkpoint.hdf5

Epoch 100/100

20110/20110 [==============================] - 3s 171us/step - loss: 0.0010 - acc: 0.9999 - val_loss: 0.0811 - val_acc: 0.9799

Epoch 00100: saving model to checkpoint.hdf5

In [8]:

 # save the model with h5py

importh5py

fromkeras.modelsimportload_model

model.save('./model/HSI_model_epochs100.h5')

In [9]:

 # using plot_model module to save the model figure

```
fromkeras.utilsimportplot_model
plot_model(model,to_file='./model/model.png',show_shapes=True)
print(history.history.keys())

# show the model figure
importmatplotlib.pyplotasplt
%matplotlibinline
model_img=plt.imread('./model/model.png')
plt.imshow(model_img,shape=(10,10))
plt.show()
dict_keys(['val_loss', 'val_acc', 'loss', 'acc', 'lr'])

In [10]:
# summarize history for accuracy
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.grid(True)
plt.legend(['train','test'],loc='upper left')
plt.savefig("./result/model_accuracy_100.svg")
plt.show()

# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
```

```
plt.ylabel('loss')
plt.xlabel('epoch')
plt.grid(True)
plt.legend(['train','test'],loc='upper left')
plt.savefig("./result/model_loss_100.svg")
plt.show()
```

5.3 Validation and Classification:

```
"""Python script to classify the image."""

    # Import the necessary libraries
    from sklearn.decomposition import PCA
    import os
    import scipy.io as sio
    import numpy as np
    from keras.models import load_model
    from keras.utils import np_utils
    from sklearn.metrics import classification_report, confusion_matrix
    import spectral
    import cv2

    # Global Variables
    windowSize = 5
    numPCAcomponents = 30
    testRatio = 0.25

    PATH = os.getcwd()
    print(PATH)
```

```
def loadIndianPinesData():
    """Method to load IndianPines."""
    data_path = os.path.join(os.getcwd(), 'data')
    data = sio.loadmat(os.path.join(data_path,
                   'Indian_pines_corrected.mat'))['indian_pines_corrected']
    labels = sio.loadmat(os.path.join(data_path,
                    'Indian_pines_gt.mat'))['indian_pines_gt']

    return data, labels


def reports(X_test, y_test):
    Y_pred = model.predict(X_test)
    y_pred = np.argmax(Y_pred, axis=1)
    target_names = ['Alfalfa', 'Corn-notill', 'Corn-mintill', 'Corn',
                'Grass-pasture', 'Grass-trees', 'Grass-pasture-mowed',
                'Hay-windrowed', 'Oats', 'Soybean-notill',
                'Soybean-mintill', 'Soybean-clean', 'Wheat',
                'Woods', 'Buildings-Grass-Trees-Drives',
                'Stone-Steel-Towers']

    classification = classification_report(np.argmax(y_test, axis=1),
                              y_pred, target_names=target_names)
    confusion = confusion_matrix(np.argmax(y_test, axis=1), y_pred)
    score = model.evaluate(X_test, y_test, batch_size=32)
    Test_Loss = score[0]*100
    Test_accuracy = score[1]*100

    return classification, confusion, Test_Loss, Test_accuracy


def applyPCA(X, numComponents=75):
    newX = np.reshape(X, (-1, X.shape[2]))
    pca = PCA(n_components=numComponents, whiten=True)
    newX = pca.fit_transform(newX)
    newX = np.reshape(newX, (X.shape[0], X.shape[1], numComponents))
    return newX, pca
```

```
def Patch(data, height_index, width_index):
    # transpose_array = data.transpose((2,0,1))
    # print transpose_array.shape
    height_slice = slice(height_index, height_index+PATCH_SIZE)
    width_slice = slice(width_index, width_index+PATCH_SIZE)
    patch = data[height_slice, width_slice, :]

    return patch


X_test = np.load(PATH + "/trainingData/" + "XtrainWindowSize" +
                       str(windowSize) +
                       "PCA" + str(numPCAcomponents) +
                       "testRatio" + str(testRatio) +
                       ".npy")

y_test = np.load(PATH + "/trainingData/" + "ytrainWindowSize" +
                       str(windowSize) +
                       "PCA" + str(numPCAcomponents) +
                       "testRatio" + str(testRatio) +
                       ".npy")

X_test = np.reshape(X_test, (X_test.shape[0],
                             X_test.shape[3],
                             X_test.shape[1],
                             X_test.shape[2]))

y_test = np_utils.to_categorical(y_test)

# load the model architecture and weights
model = load_model('hyperspectralModel.h5')

classification, confusion, Test_loss, Test_accuracy = reports(X_test, y_test)
classification = str(classification)
confusion = str(confusion)
filename = "reportWindowSize"
filename += str(windowSize)
filename += "PCA"
filename += str(numPCAcomponents)
filename += "testRatio"
filename += str(testRatio)
```

```
filename += ".txt"


with open(filename, 'w') as x_file:
    x_file.write('{} Test loss (%)'.format(Test_loss))
    x_file.write('\n')
    x_file.write('{} Test accuracy (%)'.format(Test_accuracy))
    x_file.write('\n')
    x_file.write('\n')
    x_file.write('{}'.format(classification))
    x_file.write('\n')
    x_file.write('{}'.format(confusion))

# load the original image
X, y = loadIndianPinesData()

X, pca = applyPCA(X, numComponents=numPCAcomponents)

height = y.shape[0]
width = y.shape[1]
PATCH_SIZE = 5
numComponents = 30

# calculate the predicted image
outputs = np.zeros((height, width))
for i in range(height-PATCH_SIZE+1):
    for j in range(width-PATCH_SIZE+1):
        target = int(y[i+PATCH_SIZE//2, j+PATCH_SIZE//2])
        if target == 0:
            continue
        else:
            image_patch = Patch(X, i, j)
            # print (image_patch.shape)
            X_test_image = image_patch.reshape(1, image_patch.shape[2],
                                    image_patch.shape[0],
                                    image_patch.shape[1]).astype('float32')
            prediction = (model.predict_classes(X_test_image))
            outputs[i+PATCH_SIZE//2][j+PATCH_SIZE//2] = prediction+1

ground_truth = spectral.imshow(classes=y, figsize=(5, 5))
spectral.save_rgb("ground_truth.png", y, colors=spectral.spy_colors)
```

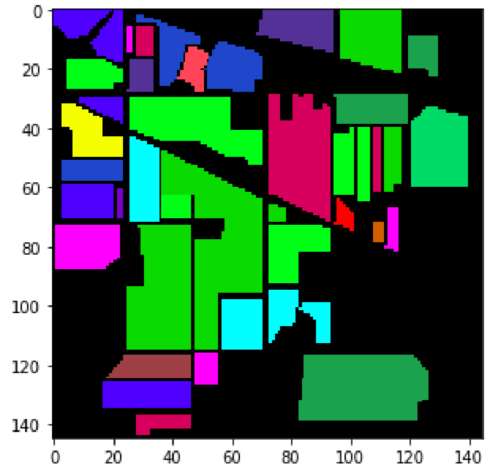
```
predict_image = spectral.imshow(classes=outputs.astype(int),
                                figsize=(5, 5))
spectral.save_rgb("predict_image.png", outputs.astype(int),
                  colors=spectral.spy_colors)
ground = cv2.imread("ground_truth.png")
cv2.resize(ground, (100, 100))
cv2.imshow("Ground Truth Image", ground)
predict = cv2.imread("predict_image.png")
cv2.resize(ground, (100, 100))
cv2.imshow("Classified Image", predict)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

CHAPTER 6

RESULTS AND DISCUSSION

```
In [12]: # Plot the Ground Truth Image
         ground_truth = spectral.imshow(classes = y,figsize =(5,5))
```




```
In [13]: # Plot the Predicted image
```

Fig 8: Ground Truth Image

This is the ground truth image which has been used for training layer by layer and also gives information for different layers. The testing is done by comparing our output with this ground truth image.

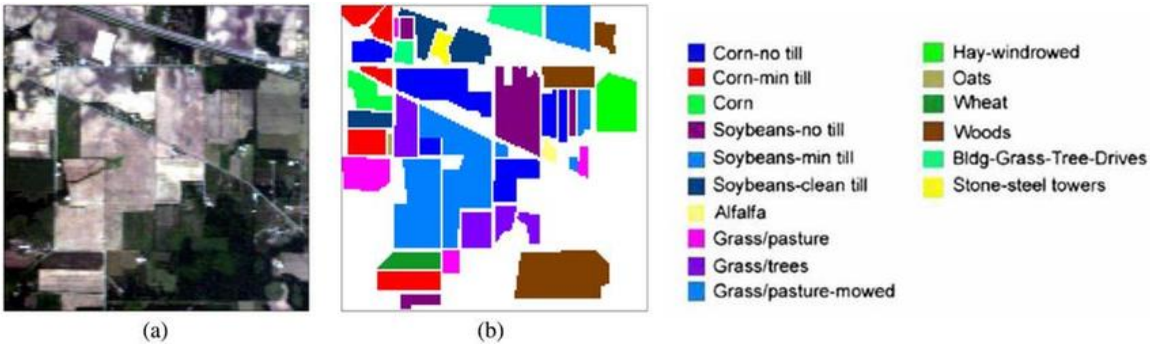


Fig 9: The different Labels with respect to the colour assigned

Ground truth classes for the Indian Pines scene and their respective samples number

#	Class	Samples
1	Alfalfa	46
2	Corn-notill	1428
3	Corn-mintill	830
4	Corn	237
5	Grass-pasture	483
6	Grass-trees	730
7	Grass-pasture-mowed	28
8	Hay-windrowed	478
9	Oats	20
10	Soybean-notill	972
11	Soybean-mintill	2455

12	Soybean-clean	593
13	Wheat	205
14	Woods	1265
15	Buildings-Grass-Trees-Drives	386
16	Stone-Steel-Towers	93

Table 1: Ground truth classes for the Indian Pines scene and their respective samples number

In [13]:
```
# Plot the Predicted image
predict_image = spectral.imshow(classes = outputs.astype(int),figsize =(5,5))
```

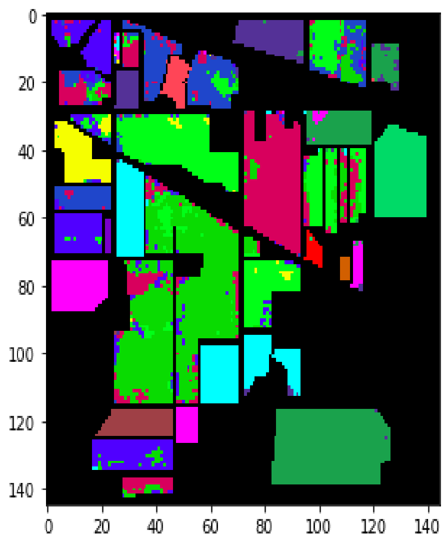


Fig 10: Final Classified Image

CHAPTER8

CONCLUSION AND FUTURE SCOPE

8.1 Conclusion

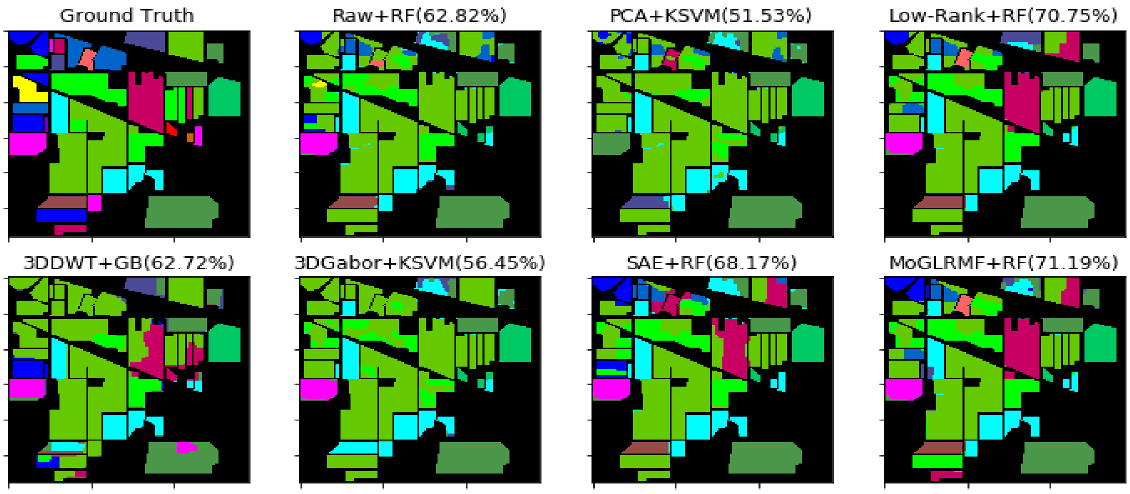


Fig 11: Comparing Previous Works

Comparing the previous work

We proposed a new methodology for efficiently classifying a hyperspectral image which uses deep learning with the implementation of tensor flow.

The proposed method is empirically shown to be faster since it is pre-trained already and cheaper because there is no need of a GPU farm.

It also avoid over fitting.

Other methods were mostly manual and consumed lot of time.

We also came across convolutional neural network which makes the task of image classification feasible with automatic feature extraction.

Hence, after comparison of our work with other related works we came to a conclusion that our model is performing better with a higher accuracy and meets our problem statement goals.

8.2 Future Scope

The future scope of the project might be putting the classification into real time usage

- Yield estimation in wheat - Hyperspectral remote sensing was used to help predict yield in wheat as a function of fertilizer concentration.
- Food Analysis- Resonon's hyperspectral imaging systems are used in food research and industry to identify defects, characterize product quality, and locate contaminants.
- Cooked Food- Subtle color changes associated with food quality can readily be identified using hyperspectral imaging.
- Environmental Monitoring- Hyperspectral imaging is used to track forest health, water quality, and surface contamination.
- Further improvement in this project could lead to more accurate results.

Machine Learning and different techniques created new systems to spot patterns which the human brain is not capable of, and since finance is quantitative, to start with, it's laborious not to notice traction. Financial corporations have conjointly endowed heavily in AI in the past, and many others are starting to investigate and implement the financial applications of machine learning (ML) and deep learning to their operations. The high emotionalism of the crypto market ecosystem has already become a topic of study by developers who are attempting to come up with an Al-based solution to increase profit returns. One of the first steps taken in this area was the creation of models that use a neural network to make cryptocurrency valuation predictions. Another way crypto trading is being influenced by AI and ML is through the analysis of sentiments. Sentiment analysis is the processing of enormous volumes of information from various sources like articles, blogs, comments, social media posts, even video transcription to work out the market's "feelings" regarding a topic — to determine if it is positive, neutral or negative. Neural networks endlessly supply increased accuracy. Neural networks make predictions associated with crypto markets remarkably faster. Their nature is to crunch information of cryptocurrency exchange rates constantly. Which are then used to forecast market movements by minutes, hours and days. Fundamental analysis is employed by both cryptocurrency and stock traders.With Artificial Intelligence, all industries, whether informational, technical or operational will become interdependent and interconnected.

REFERENCES

- *Hierarchical Multi-Scale Convolutional Neural Networks for Hyperspectral Image Classification, Simin Li, Xueyu Zhu, Jie Bao*
- *International Journal of Pure and Applied Mathematics Volume 101 No. 5 2015, 809-829*
- *Wang, Yi & Duan, Hexiang. (2018). Classification of Hyperspectral Images by SVM Using a Composite Kernel by Employing Spectral, Spatial and*

Hierarchical Structure Information. Remote Sensing. 10. 26. 10.3390/rs10030441.

- *Fang, B.; Li, Y.; Zhang, H.; Chan, J.C.-W. Semi-Supervised Deep Learning Classification for Hyperspectral Image Based on Dual-Strategy Sample Selection. Remote Sens. 2018, 10, 574.*
- *Z. Zhao, P. Zheng, S. Xu and X. Wu, "Object Detection With Deep Learning: A Review," in IEEE Transactions on Neural Networks and Learning Systems, vol. 30, no. 11, pp. 3212-3232, Nov. 2019, doi: 10.1109/TNNLS.2018.2876865.*