

A REPORT
ON
**AUTOMATIC LAND COVER CLASSIFICATION
OF TEMPORAL SATELLITE IMAGES**

BY
Guntaas Singh 2018A7PS0269P
Nisarg Vora 2018A7PS0254P

AT
Indian Institute of Remote Sensing, Dehradun
A Practice School - I station of
BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI

June, 2020

A REPORT
ON
**AUTOMATIC LAND COVER CLASSIFICATION
OF TEMPORAL SATELLITE IMAGES**

BY

ID Number	Name	Branch
2018A7PS0269P	Guntaas Singh	B.E. (Hons.) Computer Science
2018A7PS0254P	Nisarg Vora	B.E. (Hons.) Computer Science

Prepared in the partial fulfillment of the

Practice School - I course

AT

Indian Institute of Remote Sensing, Dehradun

A Practice School - I station of

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI

June, 2020

Acknowledgments

We would sincerely like to thank the Director of Indian Institute of Remote Sensing, Dr. Prakash Chauhan, for giving us an opportunity to work in this organization and gain exposure to corporate work culture, ethics, and etiquette to be followed while working in a professional environment.

We would like to extend our most sincere gratitude to our project in-charge, Dr. Hari Shanker Srivastava, Head of the Programme Planning and Evaluation Group (PPEG) at IIRS, for providing us with the opportunity to work with him on this project, and his guidance and mentorship during the same.

We wish to extend our gratitude to the faculty in charge of the PS-I program at IIRS, Rekha A., Assistant Professor at BITS Pilani - Bangalore Center, for her guidance and advice during the PS-I program, and her helpfulness and responsiveness while addressing all the concerns we raised during the same.

In addition, we would also like to thank the members of the Practice School Division, for their hard work and dedication in operating the PS-I programme remotely to ensure that we have a seamless learning experience.

Guntaas Singh
Nisarg Vora

BIRLA INSTITUTE OF SCIENCE AND TECHNOLOGY
PILANI (RAJASTHAN)
Practice School Division

Station: Indian Institute of Remote Sensing

Centre: Dehradun

Duration: 3 weeks

Date of start: 18th May, 2020

Date of submission: 22nd June, 2020

Title of project: Automatic Land Cover Classification of Temporal Satellite Images

ID Number	Name	Branch
2018A7PS0269P	Guntaas Singh	B.E. (Hons.) Computer Science
2018A7PS0254P	Nisarg Vora	B.E. (Hons.) Computer Science

Name of mentor: Dr. Hari Shanker Srivastava

Designation of mentor: Scientist/Engineer - SG. Group Head, Programme Planning and Evaluation Group (PPEG).

Name of PS faculty: Rekha A.

Key Words: Semantic Segmentation, Support Vector Machines, Convolutional Neural Networks, U-Net Architecture

Project Areas: Remote Sensing, Machine Learning, Deep Learning

Abstract

This project attempts automatic land cover classification of different parts of India into forest, built-up, agricultural land and water bodies using temporal remote sensing data. Data from Agra district, Uttar Pradesh has been used to train different models - k-nearest neighbours, decision trees, support vector machines and convolutional neural networks. These models are then tested on Ahmedabad and Gandhinagar, Gujarat. Google Earth Engine has been used to obtain data from Landsat 8 satellite images. For the purpose of classification, Normalized Difference Vegetation Index (NDVI) values are calculated by masking all other light bands except near-infrared and red light bands. Temporal images with NDVI labels are fed as input to train the models and subsequently, the performance of these models is compared. A convolutional neural network based on the U-Net architecture is found to produce the most accurate results, improving upon traditional machine learning techniques. The models implemented can be used to produce land cover maps for any region, with good accuracy. These can then be used for various applications like natural resource management, urban expansion, etc.

BIRLA INSTITUTE OF SCIENCE AND TECHNOLOGY
PILANI (RAJASTHAN)
Practice School Division
Response Option Sheet

Station: Indian Institute of Remote Sensing

Centre: Dehradun

ID Number	Name	Branch
2018A7PS0269P	Guntaas Singh	B.E. (Hons.) Computer Science
2018A7PS0254P	Nisarg Vora	B.E. (Hons.) Computer Science

Title of project: Automatic Land Cover Classification of Temporal Satellite Images

Usefulness of the project to the on-campus courses of study in various disciplines. Project should be scrutinized keeping in view the following response options. Write Course No. and Course Name against the option under which the project comes.

Code No.	Response Option	Course No.(s) and Name
1	A new course can be designed out of this project.	
2	The project can help modification of the course content of some of the existing Courses	
3	The project can be used directly in some of the existing Compulsory Discipline Courses (CDC)/ Discipline Courses Other than Compulsory (DCOC)/ Emerging Area (EA), etc. Courses	
4	The project can be used in preparatory courses like Analysis and Application Oriented Courses (AAOC)/ Engineering Science (ES)/ Technical Art (TA) and Core Courses.	
5	This project cannot come under any of the above mentioned options as it relates to the professional work of the host organization.	

Signature

Date:

Contents

Acknowledgments	iii
Abstract	iv
1 Introduction	1
1.1 About IIRS	1
1.2 Remote Sensing	1
1.3 Objective: Land Cover Classification	1
1.3.1 Techniques	2
1.3.2 Significance	2
2 Background	4
2.1 Normalized Difference Vegetation Index(NDVI)	4
2.1.1 What is NDVI?	4
2.1.2 Formula	4
2.1.3 Correlation with plant health	5
2.2 Image Classification	6
2.2.1 Pixel Based Approach	6
2.2.2 Object Based Approach	6
2.2.3 Semantic Approach	6
2.3 Deep Learning and Neural Networks	7
2.4 Convolutional Neural Networks	8
2.4.1 Convolutional Layer	8
2.4.2 Pooling Layer	10
2.4.3 Fully Connected Layer	10
2.5 Semantic Segmentation	11
2.6 FCN Architecture	12
2.6.1 Transpose Convolution	12
2.6.2 Skip Connections	13
3 Methodology and Implementation	14
3.1 Software Overview	14
3.1.1 Google Earth Engine	14
3.1.2 Google Colab	14
3.1.3 Python	14
3.1.4 Scikit-learn and Scikit-image	15
3.1.5 TensorFlow	15

3.2	Data Collection	15
3.2.1	Datasets	15
3.2.2	Study Areas	16
3.2.3	Data Preparation	17
3.3	Data Preprocessing	18
3.3.1	Linear Discriminant Analysis	18
3.3.2	Patch generation	20
3.3.3	Data Augmentation	20
3.4	Machine Learning Techniques	21
3.5	Support Vector Machines	21
3.5.1	Types of SVMs	22
3.5.2	Advantages	22
3.5.3	Disadvantages	22
3.5.4	Implementation	23
3.6	Decision Trees	23
3.6.1	Advantages	24
3.6.2	Disadvantages	24
3.6.3	Implementation	24
3.7	K-Nearest Neighbors	25
3.7.1	Overview	25
3.7.2	Advantages	25
3.7.3	Disadvantages	26
3.7.4	Implementation	26
3.8	U-Net Architecture	27
3.8.1	Components	27
3.8.2	Advantages	29
3.8.3	Disadvantages	29
3.8.4	Implementation	29
3.8.5	Training	31
4	Results	33
5	Conclusion	36
5.1	Future Scope	36
Appendix		38
	List of dependencies	38
	Code snippets	38

References	51
List of Abbreviations	53
Glossary	53

List of Figures

1.1	NDVI distribution in 2000 and 2015 in Kathmandu valley, Nepal, derived from MODIS data. Depicts changes in land cover/land use over time.	2
2.1	(1) Without NDVI. (2) With NDVI.	5
2.2	Plant health directly affects reflectance in NIR and RED bands.	5
2.3	A single neuron.	7
2.4	A two layer neural network with fully connected layers.	8
2.5	The convolution operation performed using a 3x3 filter on a 5x5x1 image.	9
2.6	(1) A typical max pooling layer. (2) The max pooling operation.	10
2.7	AlexNet - an influential NN architecture that popularized the use of CNNs and GPUs to accelerate deep learning.	11
2.8	A 3×3 transpose convolution with stride 2 and padding 1. The output gets up-sampled by a factor of 2.	12
2.9	The different variants of the FCN architecture defined in [16], with different skip connections added to the base network.	13
2.10	Different networks based on the FCN architecture.	13
3.1	Different views of the training location - Agra. (1) True colour satellite image (2) False colour infrared image (3) Map of NDVI values (4) Ground truth land cover. [Google Earth Engine]	16
3.2	Masks for different land cover classes for Agra and Mathura region.	18
3.3	Patch generation and data augmentation.	21
3.4	Support Vector Machine	22
3.5	K-Nearest Neighbor Visualization	25
3.6	The U-Net architecture.	27
3.7	Curves for loss and accuracy against number of epochs.	32
4.1	Ground truth and predictions for the training location - Agra and Mathura, Uttar Pradesh	34
4.2	Ground truth and predictions for the testing location - Ahmedabad and Gandhinagar, Gujarat	35

1. Introduction

1.1 About IIRS

Formerly known as Indian Photo-interpretation Institute (IPI), the institute was founded on 21st April, 1966 under the aegis of Survey of India (SOI). It was established in collaboration with the Government of The Netherlands on the pattern of Faculty of Geo-Information Science and Earth Observation (ITC) of the University of Twente, The Netherlands. The original idea of setting up the institute came from India's first Prime Minister - Pandit Jawahar Lal Nehru during his visit to The Netherlands in 1957.

Since its establishment in 1966, IIRS is a key player for training and capacity building in geo-spatial technology and its applications through training, education and research in Southeast Asia. The training, education and capacity building programmes of the institute are designed to meet the requirements of professionals at working levels, fresh graduates, researchers, academia, and decision makers. [11, 12]

1.2 Remote Sensing

Remote sensing is the method of getting information about Earth and its surface without making any physical contact, primarily using satellites. Remote sensing is used in numerous fields, including geography, land surveying and most Earth science disciplines (for example, hydrology, ecology, meteorology, oceanography, glaciology, geology); it also has military, intelligence, commercial, economic, planning, and humanitarian applications.[28] It may be split into "active" remote sensing and "passive" remote sensing. In active remote sensing method, sensors are used to capture the reflection of the signals emitted by the satellite on different objects. In passive remote sensing, the reflection captured by the sensors is mainly due to sunlight. In this project, we are using passive remote sensing data collected by Landsat 8 satellite.

1.3 Objective: Land Cover Classification

Land Cover classification generally refers to the categorization or classification of human activities and natural elements on the landscape within a specific time frame based on established scientific and statistical methods of analysis of appropriate source materials. Land cover refers to the surface cover on the ground like vegetation, urban infrastructure,

water, bare soil etc. Identification of land cover establishes the baseline information for activities like thematic mapping and change detection analysis.[29]

1.3.1 Techniques

Mainly two techniques are used for land cover classification: field survey and remotely sensed imagery analysis. Field survey was the primary method used for land cover classification until a few decades back. However, it was very inaccurate and time-consuming. With developments in technology, the conventional method of field survey which can be inaccurate and time consuming has been replaced by remotely sensed imagery analysis which is highly accurate and covers wider range.

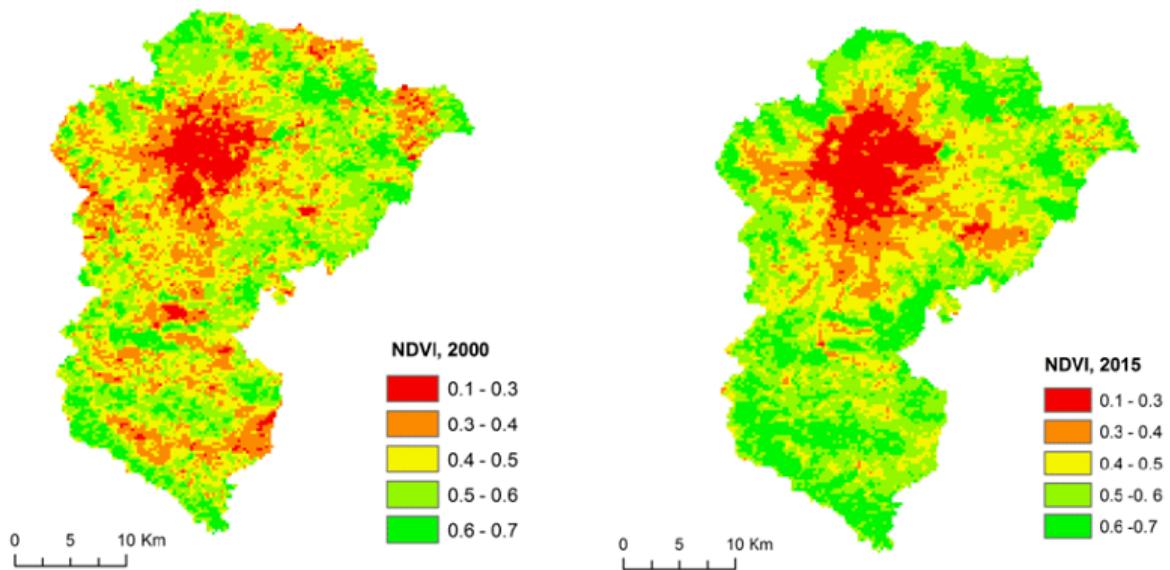


Figure 1.1: NDVI distribution in 2000 and 2015 in Kathmandu valley, Nepal, derived from MODIS data. Depicts changes in land cover/land use over time.

[2]

1.3.2 Significance

Land cover classification data is of great importance in the following applications:

- **Natural Resource Management:** Land cover classification can help policy makers decide how to best manage all the available resources by looking at different terrains. For example, the government can decide which area requires irrigation facilities the most using information about the land cover class distribution.

- **Wildlife habitat protection:** With shrinking habitats of wild animals, land cover classification can provide important data for helping in wildlife preservation and protection.
- **Urban Expansion/encroachment:** Proper planning is required for expansion of urban area. Different land cover features need to be studied to minimize the time and economic cost required for tackling natural obstacles to urban expansion.
- **Damage delineation (Tornadoes, flooding, fire, volcanic):** Live land cover data can provide us with information about where a certain natural disaster (like flood) has occurred/can occur. This can also help us to identify the areas at maximum risk of damage.
- **Target Detection - identification of land strips for use:** Using land cover data, identification of ideal locations for different urban construction becomes easy. For example, one can decide where exactly it is possible to build an airport, bridge etc.

2. Background

2.1 Normalized Difference Vegetation Index(NDVI)

Satellite data has become a very important parameter for judging crop progress, land cover classification etc. However, satellite data can not directly report crop health or classify land cover into different classes. NDVI which is calculated by measuring the reflectance of near-infrared and red light is probably the most important and accurate parameter for farmers to know crop health. NDVI values are so accurate that they can in fact, inform farmers about an upcoming drought upto two weeks in advance. Farmers can then make corresponding decisions to increase irrigation and other crop supplements.

2.1.1 What is NDVI?

Light from the sun falls on Earth in three major light bands: ultraviolet, infrared and red. Most of the ultraviolet light is reflected back by the atmosphere. Healthy green plants absorb most of the visible light for photosynthesis. However, to prevent over-heating and dehydration the light in red band and near-infrared band is reflected back by the plants. The red light is visible to humans, while infrared is not. The NDVI values are calculated using the reflectance in red and near-infrared regions.

2.1.2 Formula

NDVI is calculated in accordance with the formula:

$$NDVI = \frac{NIR - RED}{NIR + RED}$$

where,

NIR = reflection in the near-infrared spectrum

RED = reflection in the red range of the spectrum.

As per the formula above, the density of vegetation (NDVI) at a certain point of the image is equal to the difference in the intensities of reflected light in the red and infrared range divided by the sum of these intensities.

NDVI values range from -1.0 to 1.0, the positive values are usually for plants and vegetation where as the negative values usually indicate presence of snow, water or cloud cover. Values very close to 0 and less than 0.1 are due to presence of urban built ups

rocks etc. Moderate values between 0.2 and 0.3 are due to shrubs and crops whereas large values close to 1.0 represent temperate and tropical forests. These rough trends indicate the relevance of NDVI to the problem of land cover classification.



Figure 2.1: (1) Without NDVI. (2) With NDVI.

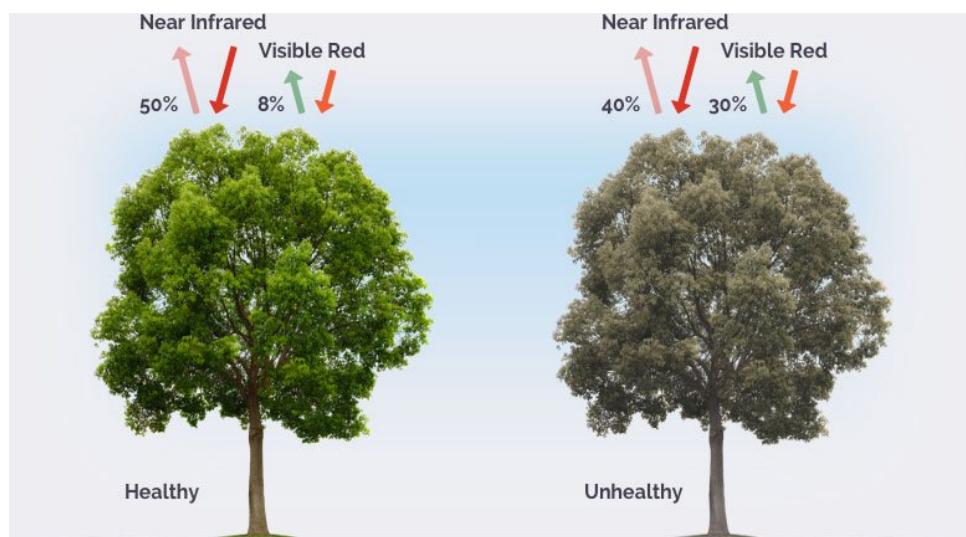


Figure 2.2: Plant health directly affects reflectance in NIR and RED bands.

2.1.3 Correlation with plant health

The amount of chlorophyll acts as a health indicator for the plants. Chlorophyll strongly absorbs visible light and the cell structure reflects near-infrared light. When chlorophyll levels are low, the cell structure gets destroyed and starts absorbing near-infrared light instead of reflecting it. Thus, value of NIR without vegetation decreases and hence the NDVI value becomes smaller.

2.2 Image Classification

Image classification is a standard task in computer vision. In general, the image classification problem involves assigning one label out of a given fixed set of discrete labels to the input image on the basis of its visual content. While this is a trivial task for humans, robust image classification is a big challenge for a machine. To the computer, the image is just a grid of numbers which entirely change in unreliable ways with variations in viewpoint, illumination, occlusion, etc. As a result, there is no obvious algorithm which solves this problem. However, a data driven approach of providing the machine with many examples of each class and use of machine learning techniques has shown to be useful.[15]

There are different ways in which these techniques can be applied for classification of satellite imagery.

2.2.1 Pixel Based Approach

In typical satellite images, pixel sizes are generally similar in size to the objects of interest. Most of the methods for image analysis using remote sensing data work on a per-pixel basis. However, with advances in remote sensing technology, the spatial resolution has become finer than the typical objects of interest, leading to an increase in within-class variability.[13]

2.2.2 Object Based Approach

The term "objects" represents meaningful semantic entities or scene components that are distinguishable in an image.[13] This approach involves the partition of the image into meaningful geographical objects that share relatively homogeneous color, texture, etc.

2.2.3 Semantic Approach

This aims to label each scene image with a specific semantic class. Here, a scene image refers to a local image patch manually extracted from large scale remote sensing images that contain explicit semantic classes.[13]

2.3 Deep Learning and Neural Networks

Application of traditional machine learning techniques requires handcrafted features, developing which demands a considerable amount of engineering skill and domain expertise. This, however, is not true for neural networks, which automatically learn these features from data using a general-purpose learning procedure.[13, 15]

A standard neural network consists of many simple, connected processors called neurons, each producing a sequence of real-valued activations. Input neurons get activated through data applied at the input of the network. Other neurons get activated through weighted connections from previously active neurons. [27] Each neuron can be seen as a single unit applying a non-linear activation function (such as sigmoid, tanh, ReLU) to a linear combination of the input activations to the neuron.[20]

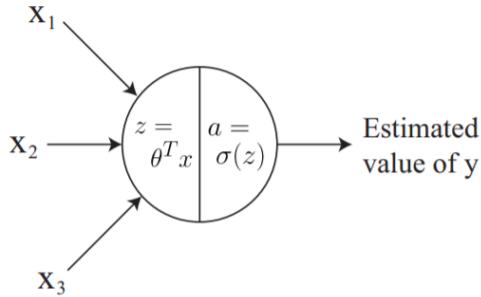


Figure 2.3: A single neuron.
[20]

These single neurons can be stacked so that one neuron passes its output as input into the next neuron. The resulting network of neurons can, hence, consist of several layers of neurons, each with their own learnable weights and biases. Used in conjunction with an appropriate loss function and optimization algorithm, such a network can be used to learn any complex function, if sufficient data is available for training. Forward propagation through the network yields its prediction for a given input. This prediction is compared with the actual class label, and the loss is computed. Backward Propagation is used to compute the gradients of the loss function with respect to the parameters, which are then used by the optimization algorithm to adjust the parameters and minimize the loss over a number of iterations.

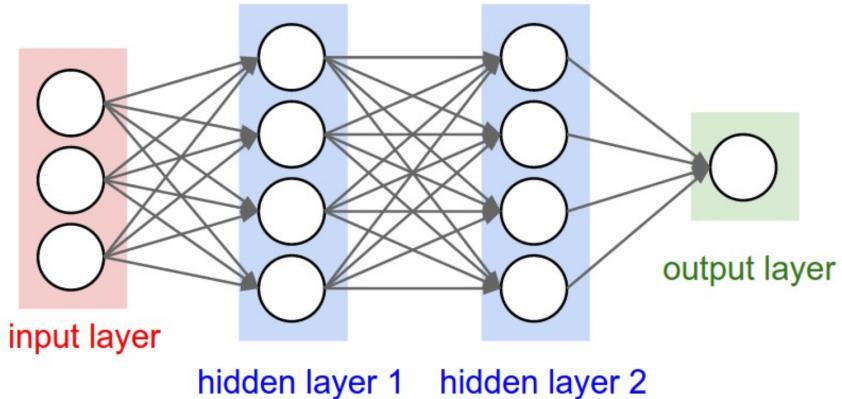


Figure 2.4: A two layer neural network with fully connected layers.

[15]

2.4 Convolutional Neural Networks

Regular neural networks do not scale well to full images. If the input to the neural network is a 200x200 RGB image, the number of weights for each neuron will be $200 \times 200 \times 3 = 120,000$ weights. For large networks, the total number of learnable parameters become very large and lead the model to potentially overfit the training data, unless the training set is adequately large.

A convolutional neural network (CNN) is a sequence of layers. Each layer transforms an input volume (images are represented as a three dimensional matrix) of activations to another with some differentiable function which may or may not have parameters.[15, 21] These layers are of three main types:

2.4.1 Convolutional Layer

This is the core building block for convolutional networks. It is based on the convolution operation on images.

Each convolutional layer of a CNN consists of N kernels or filters of a certain volume of neurons sized $f \times f \times d$, with f being the spatial dimension and d being the number of feature channels of the kernel, which is same as the number of channels in the image at its input (D_i). Every one of these filters is slid through the entire image of size $H_i \times W_i \times D_i$. Convolution refers to the summation of the element-wise dot product of the neurons in each filter with the corresponding values in the input, for each position in which the filter

is aligned with the image. Based on this notion, a convolution with a single filter at each layer results in a two dimensional output of a certain size. [15, 19, 21]

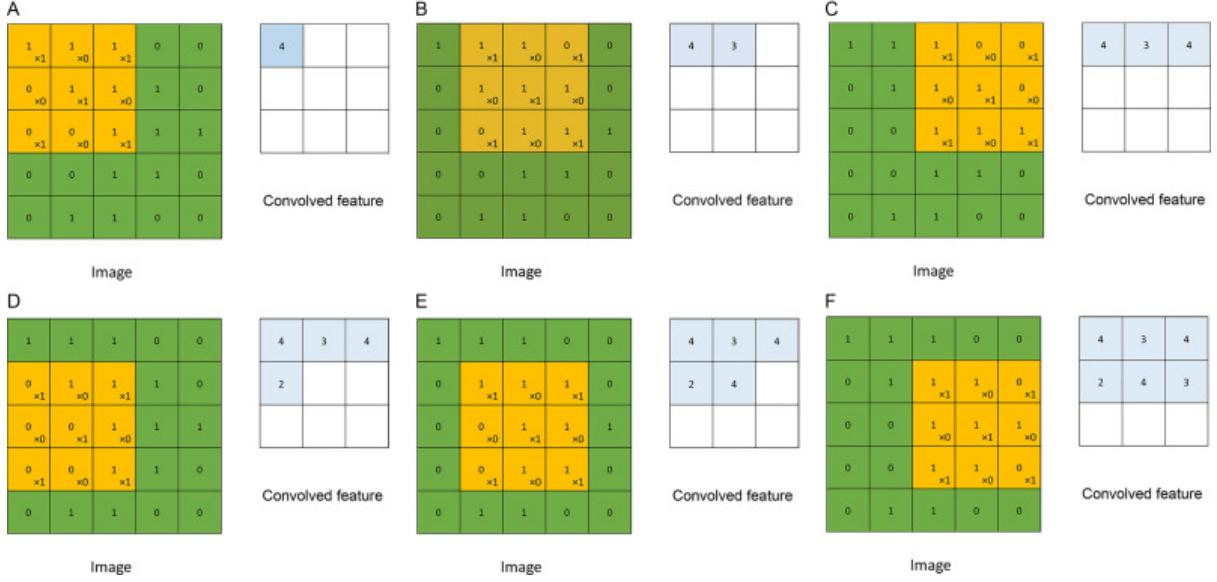


Figure 2.5: The convolution operation performed using a 3x3 filter on a 5x5x1 image.
[5]

The intervals with which the filter moves in each spatial dimension is decided by the **stride** s . In order to prevent undue shrinkage of the volume along the spatial dimension, the image can be padded with pixels along the outer edges. The width of this **padding**, in pixels, is given by another hyper-parameter, p .[21, 19] The convolution operation is repeated for each of the N filters, and the resulting N activation maps are stacked together across the third dimension giving an output volume of dimensions:

$$H_o = \frac{H_i - f + 2p}{s} + 1$$

$$W_o = \frac{W_i - f + 2p}{s} + 1$$

$$D_o = N$$

In a convolutional layer, each neuron is connected to only a local region of the input volume, called the receptive field of that neuron. The extent of connectivity is limited to the filter size along the spatial dimensions, but is full along the depth axis.[15] It should also be noted that all activations belonging to a particular channel in the output volume correspond to a single filter applied on the input volume, and hence depend on the same

shared parameters. Local connectivity and parameter sharing not only help reduce the number of learnable parameters, but also make the CNN good at capturing **translation invariance**. This makes them an ideal choice for the image classification problem.

2.4.2 Pooling Layer

It is common to periodically insert a Pooling layer in-between successive convolutional layers in a CNN. Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting.[15] The most common form of pooling layer in CNN architectures employs filters of size 2×2 with a stride of 2, taking a max over 4 cells of the input image. It is worth noting that while this halves the width and height of the image, the depth remains unaffected as the max operation is applied independently to each channel of the image. This down-sampling process effectively discards 75% of the activations.

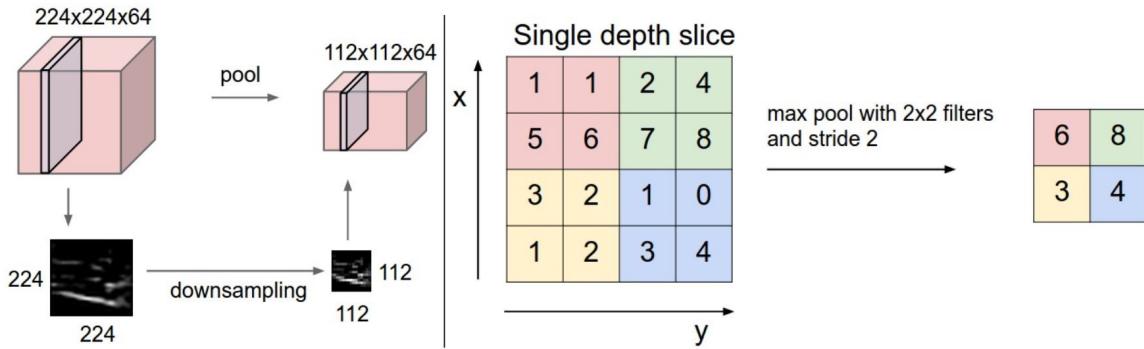


Figure 2.6: (1) A typical max pooling layer. (2) The max pooling operation.
[15]

2.4.3 Fully Connected Layer

Once higher level features are detected from the preceding convolution and pooling layers, a fully connected layer is usually attached at the end of the network. This layer is fully connected to all activations in the previous layer, as in regular neural networks, allowing all the features learned by the network to be taken into account by the output layer.

All of these different types of layers can be stacked together in various ways to form a CNN.

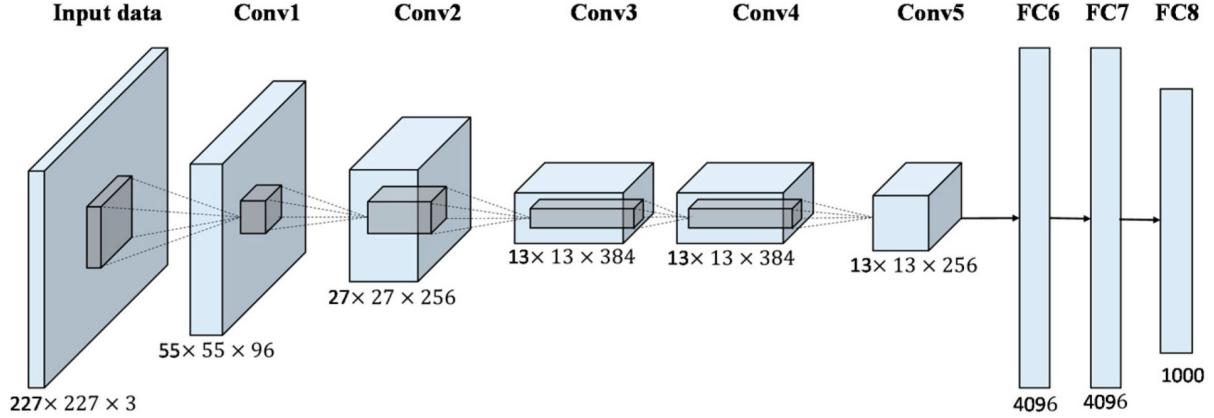


Figure 2.7: AlexNet - an influential NN architecture that popularized the use of CNNs and GPUs to accelerate deep learning.

[14, 10]

2.5 Semantic Segmentation

Typical CNNs used for image classification take fixed-size inputs and produce non-spatial outputs (predicted class labels). However, in case of land cover classification, the objective is not to assign a single class label to a given satellite image, but to divide the given image into segments each corresponding to one class out of a given set of land cover classes. This problem is known as semantic segmentation. There are different ways in which the existing image classification CNNs can be extended to solve this problem.

One approach is to use networks with only convolutional layers. In these networks, zero padding can be used to preserve the spatial dimensions of the input image. However, the large spatial dimensions throughout the network make the model computationally infeasible. [16, 15, 24]

While the problem requires the CNN to produce a spatial output, down-sampling is essential for reducing the need for computational resources to a feasible level. As a result, the most popular approach is to design the network as a combination of an encoder(down-sampling path) and a decoder(up-sampling path).

2.6 FCN Architecture

Fully connected layers have an equivalent representation as a convolutional layer having N filters with dimensions equal to those of the input image. The output of this layer will thus be a volume of dimensions $1 \times 1 \times N$. This simple change allows the same CNN to be applied on images with arbitrary spatial dimensions and classify them in a single pass of forward propagation. This is far more efficient than iterating on different crops and classifying one pixel at a time. This is the basic intuition behind **Fully Convolutional Networks**.[16] For up-sampling the classification maps produced by a FCN, *Long et al.*[16] propose the use of transpose convolution.

2.6.1 Transpose Convolution

Ordinary convolution involves taking a dot product between the filter and the input for every receptive field at a stride of s in the input, and storing the result at a stride of 1 in the output. The resulting output image gets down-sampled by a factor of s . In transpose convolution, we use the values at a stride of 1 in the input and take their scalar product with the filter, storing the resulting matrix at a stride of s in the output. As a result the image gets up-sampled by a factor of s . Any overlapping values in the output are summed together.[16, 15] A stack of layers performing transpose convolution with an appropriate activation function can be used to learn a non-linear up-sampling.[16]

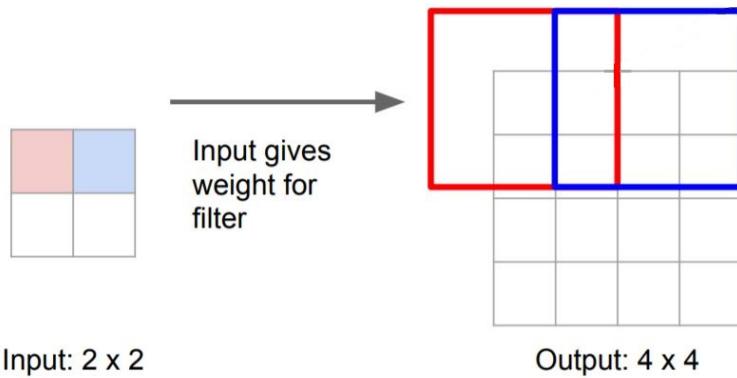


Figure 2.8: A 3×3 transpose convolution with stride 2 and padding 1. The output gets up-sampled by a factor of 2.

[15]

2.6.2 Skip Connections

Semantic segmentation faces an inherent tension between semantics and location. Deep layers capture local information and resolve the semantic information that the image contains. Global information available to shallower layers can be used to capture the location. This motivates the idea of adding skip connections from shallow layers to the final prediction layer. Combining the information captured by shallow and deep layers at the time of up-sampling lets the model make local predictions that take into account global structure and semantics. To combine activations with different spatial dimensions, they are first up-sampled individually as required using transpose convolutions and then summed up. The result is then up-sampled by a final layer, so that the output spatial dimensions match those of the input image. The addition of skip connections produces a finer and more detailed segmentation map.[16]

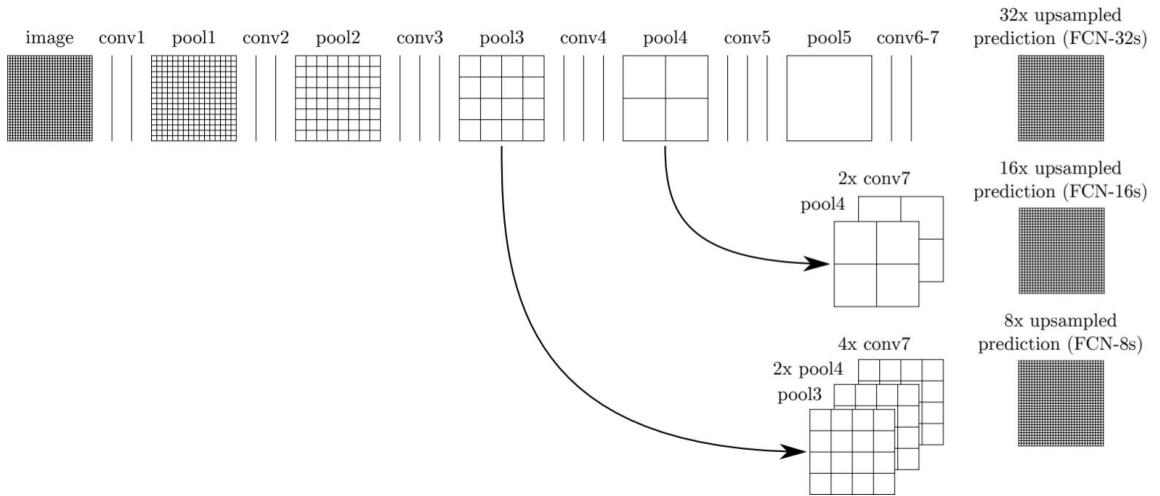


Figure 2.9: The different variants of the FCN architecture defined in [16], with different skip connections added to the base network.

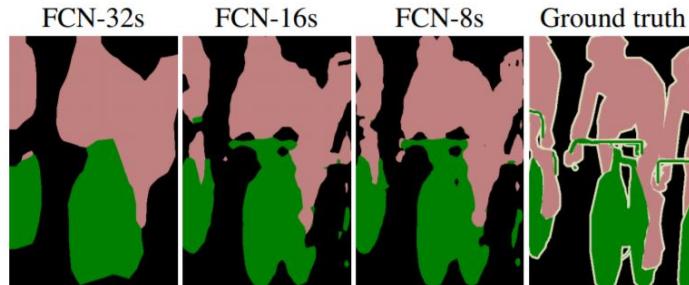


Figure 2.10: Different networks based on the FCN architecture.
[16]

3. Methodology and Implementation

3.1 Software Overview

3.1.1 Google Earth Engine

Google Earth Engine is a computing platform that allows users to run geo-spatial analysis on Google's cloud infrastructure. It provides several ways to interact with the platform. The Code Editor is a web-based Integrated Development Environment (IDE) for writing and running scripts. The Explorer is a lightweight web app for exploring the data catalog and running simple analyses. The client libraries provide Python and JavaScript wrappers around the web Application Programming Interface (API).[9] In our project, the **JavaScript API** was used to visualize, prepare, process, and export satellite imagery and data.

3.1.2 Google Colab

Google Colab is a hosted Jupyter notebook service that provides a web-based development environment for writing and executing arbitrary Python code within a browser. It is particularly well suited for machine learning and data analysis applications. It requires no setup to use, enables quick and easy collaboration and provides free access to computing resources like GPUs.[6] These computational resources are of great importance for training any classifier as they can significantly reduce training time. Since applying learning techniques is a very empirical process, the ability to iterate quickly is indispensable.

3.1.3 Python

Python is an interpreted, high-level, general purpose programming language. It has an easy to learn syntax that emphasizes code readability. It supports multiple programming paradigms and has a comprehensive standard library. It also has a well-developed ecosystem of libraries for scientific computing and data science such as **NumPy**, **Pandas** and **Matplotlib**. These libraries provide data structures and methods for handling large amount of data efficiently and for high-performance computation. In addition to these libraries, in our project, we also used **Pillow** (Python Imaging Library) for image processing and **GDAL** (Geospatial Data Abstraction Library) for handling images in the **GeoTIFF** file format.

3.1.4 Scikit-learn and Scikit-image

Scikit-learn is a free software machine learning library for Python. It features various algorithms for supervised and unsupervised learning tasks such as classification, regression and clustering. It heavily depends on the NumPy library for high-performance linear algebra. In our project, it was used for implementing various models based on machine learning algorithms such as support vector machines, k-nearest neighbours and decision tree classification. for data pre-processing and augmentation. The Scikit-image library was used for augmentation of training data.[23]

3.1.5 TensorFlow

TensorFlow is an end-to-end open source platform for deep learning. It provides high-level APIs based on the Keras API standard which enable quick and easy implementation of deep neural networks. It also provides a library of different optimizers, losses and metrics for training the network. Modules for handling and pre-processing different types of data are also available. It can also be used to implement complex networks with non-linear topology. We implemented such a network based on the U-Net Architecture using the **Keras Functional API**.[1]

3.2 Data Collection

3.2.1 Datasets

Landsat 8

Time series data for NDVI was collected from the *Landsat 8 Collection 1 Tier 1 8-Day NDVI Composite* product. The data is collected by Landsat 8 - an American Earth Observation satellite launched in 2013 under Landsat, a joint program of the United States Geological Survey (USGS) and National Aeronautics and Space Administration (NASA). The satellite images the entire Earth's surface at a spatial resolution of 30m, once every two weeks and records multi-spectral and thermal data. The Operational Land Imager (OLI) collects passive remote sensing data from nine spectral bands including the Near Infrared and Visible bands which we intend to use for the project. The data has been atmospherically corrected in addition to standard geometric adjustments and geo-referencing. Information from the thermal infrared sensor is also used to mask clouds. The imagery, hence, provides a clear view of land. The high temporal resolution of 8 days makes the dataset suitable for the purpose of our project.[8, 30]

Copernicus Global Land Cover dataset

The *Copernicus Global Land Service (CGLS) - LC100 collection 2* is earmarked as a component of the Land service to operate a multi-purpose service component that provides a series of bio-geophysical products on the status and evolution of land surface at global scale. Derived from the PROBA-V 100m time series data for the year of 2015, this dynamic land cover map at 100 m spatial resolution provides a primary land cover scheme. Next to these discrete classes, the product also includes continuous field layers for all basic land cover classes that provide proportional estimates for vegetation/ground cover for the land cover types.[7]

3.2.2 Study Areas

For training our classifiers, we used images of Agra, Mathura and the surrounding region, Uttar Pradesh. This location was chosen as it has a relatively even distribution of different land cover classes. Additionally, first-hand knowledge of the land cover of the region was provided by the mentor which helped us refine the data. For testing, images of Ahmedabad and Gandhinagar, Gujarat were used because of similar reasons.

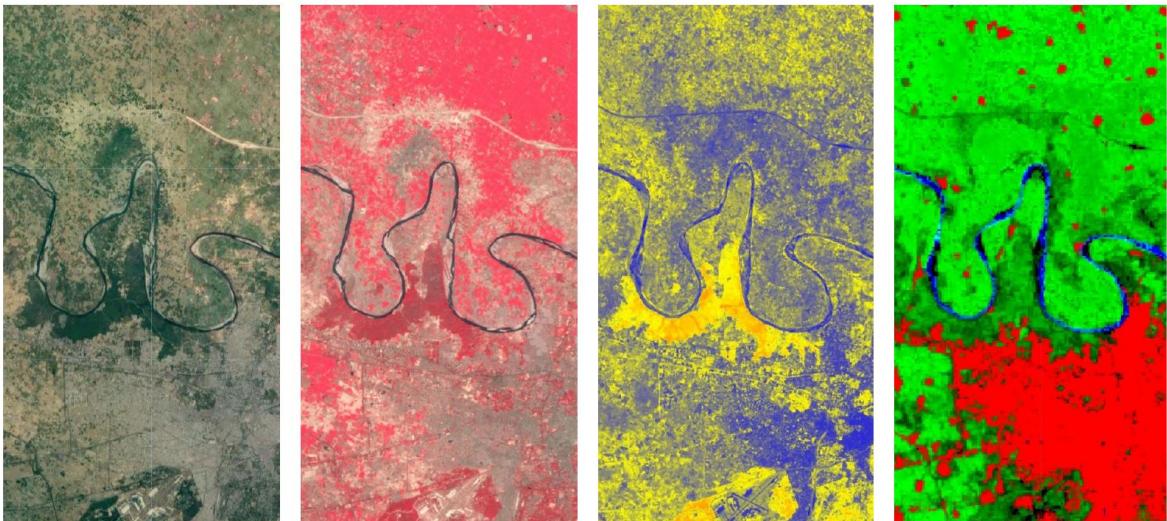


Figure 3.1: Different views of the training location - Agra. (1) True colour satellite image (2) False colour infrared image (3) Map of NDVI values (4) Ground truth land cover. [Google Earth Engine]

3.2.3 Data Preparation

All the data required was collected and prepared using Google Earth Engine Javascript API.

NDVI time series data

The Landsat 8 NDVI Composite product was first imported as an ImageCollection (a class defined in the API, representing a set of images). The extents of the images were constrained to the required region - Agra and Mathura. The collection was then filtered for bimonthly periods (January to February, March to April, and so on). While a higher temporal resolution is desirable, it would not be possible to mask all cloudy regions using the limited number of images in a smaller duration. After filtering, the median NDVI value was computed for every pixel in each bimonthly period. The six images thus achieved were stacked together, producing a single six-band image.

Ground truth land cover data

The Copernicus Global Land Cover dataset has different "coverfraction" bands which provide proportional estimates for different land cover classes classes. Appropriate thresholds were estimated for each of these bands through an iterative process of comparison with land cover maps available via ISRO's Bhuvan platform. The masks obtained by applying these thresholds were concatenated to arrive at a mutually exclusive and exhaustive classification of the study area. The maps obtained were found to be have satisfactory accuracy but suffered from the low spatial resolution of the original dataset - 100m. The project aims to produce finer land cover maps by operating on satellite imagery with a spatial resolution of 30m. The land cover classes used are:

1. Built-up land, including urban and rural areas
2. Cultivated and managed vegetation
3. Water bodies
4. Forest land

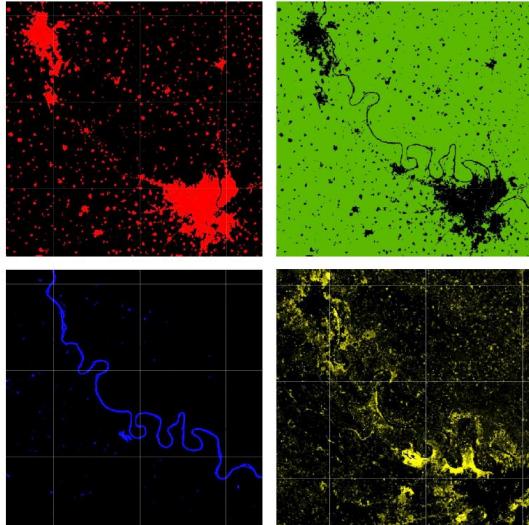


Figure 3.2: Masks for different land cover classes for Agra and Mathura region.

3.3 Data Preprocessing

3.3.1 Linear Discriminant Analysis

Linear discriminant analysis (LDA) is widely used in statistics, pattern recognition, and machine learning to find a linear combination of features that separates two or more classes of objects or events. It is a generalization of Fisher's linear discriminant- which can only be used for two class problems. Principal Component Analysis(PCA) is another dimensionality reduction technique which is closely related to LDA. Both attempt to find linear combinations of variables that explain the data. LDA tries to maximize the separation between the classes whereas PCA does not consider any class differences. Thus, LDA is preferred over PCA for classification tasks.

Consider a set of observations \vec{x} (also called features, attributes, variables or measurements) for each sample of an object or event with known class y . This set of samples is called the training set. The classification problem is then to find a good predictor for the class y of any sample of the same distribution (not necessarily from the training set) given only an observation \vec{x} . [18] $(\vec{\mu}_0, \Sigma_0)$ and $(\vec{\mu}_1, \Sigma_1)$ are mean and co-variance features , respectively.

LDA instead makes the additional simplifying homoscedasticity assumption (i.e. that the class covariances are identical, so $(\Sigma_0 = \Sigma_1 = \Sigma)$ and that the covariances have full

rank. In this case, several terms cancel:

$$\vec{x}^T \Sigma_0^{-1} \vec{x} = \vec{x}^T \Sigma_1^{-1} \vec{x}$$

$$\vec{x}^T \Sigma_i^{-1} \vec{\mu}_i = \vec{\mu}_i^T \Sigma_i^{-1} \vec{x}$$

because Σ_i is Hermitian and the above decision criterion becomes a threshold on the dot product $\vec{w} \cdot \vec{x} > c$ for some threshold constant c , where

$$\vec{w} = \Sigma^{-1}(\vec{\mu}_1 - \vec{\mu}_0)$$

$$c = \vec{w} \cdot \frac{1}{2}(\vec{\mu}_1 + \vec{\mu}_0)$$

This means that the criterion of an input \vec{x} being in a class y is purely a function of this linear combination of the known observations.

It is often useful to see this conclusion in geometrical terms: the criterion of an input \vec{x} being in a class y is purely a function of projection of multidimensional-space point \vec{x} onto vector \vec{w} (thus, we only consider its direction). In other words, the observation belongs to y if corresponding \vec{x} is located on a certain side of a hyperplane perpendicular to \vec{w} . The location of the plane is defined by the threshold c .[18]

Application

During the sowing season the agricultural land is mostly barren. Therefore, the classifier may tend to classify it as wasteland if the classification is made solely based on NDVI values. Similarly, in dry seasons, dry rivers might get classified as barren land too. To prevent this, we use a data preprocessing method- Linear Discriminant Analysis.

Linear discriminant analysis also worked as a dimensionality reduction technique for this project which drastically reduced the processing time and made results more accurate. The accuracy of the results increased from 78 per cent without LDA to almost 97 per cent after preprocessing with LDA method. Since NDVI values were calculated bi-monthly, the data initially had 6 features i.e. 6 dimensional. As the data points were classified into 4 different classes, LDA reduced the number of features to 3 i.e. 3 dimensional.

	Jan-Feb	Mar-Apr	May-Jun	Jul-Aug	Sep-Oct	Nov-Dec
Pixel 1	0.38634	0.23694	0.14535	0.27368	0.45237	0.15393
Pixel 2	0.42746	0.25465	0.15213	0.28335	0.42710	0.12429
Pixel 3	0.41443	0.25145	0.15098	0.28209	0.42312	0.12484

Table 3.1: Pixel-wise bi-monthly data before applying Linear Discriminant Analysis

	LDA1	LDA2	LDA3
Pixel 1	0.30820489	0.19799044	0.10593896
Pixel 2	0.10530831	0.31693233	0.2344135
Pixel 3	0.21845982	0.30482929	0.5671233

Table 3.2: Pixel-wise bi-monthly data after applying Linear Discriminant Analysis

3.3.2 Patch generation

Since, the neural network we implemented expects inputs to have dimensions of 256×256 , the satellite images extracted cannot be used directly. To this end, a patch generation system was implemented. The system takes images of any width and height as input and produces cropped image patches of the required dimensions using a given stride. The patches obtained cover the input image completely, irrespective of its dimensions. A stride smaller than the size can also be used to produce overlapping patches. A reconstruction system was also implemented to restitch the patches produced as output by the neural network, with a provision for handling overlapping patches.

3.3.3 Data Augmentation

Deep neural networks typically require a large amount of training data. It may not be possible for the network to perform well unless the training data provides a representative sample of the distribution of different types of land cover in different areas. Therefore, to augment the size of the training dataset, every patch was rotated by 90° thrice. This allowed us to train the network on four times as much data. It is expected to have helped the network to deal with spatial variations better. Since, machine learning algorithms do not require as much data and consume flattened data in a tabular format, this augmentation was not done for those models.

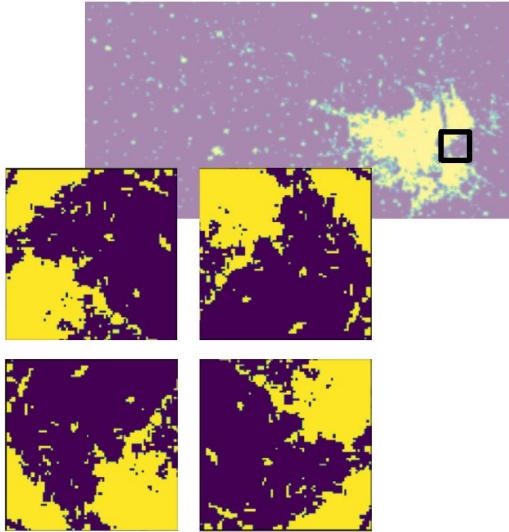


Figure 3.3: Patch generation and data augmentation.

3.4 Machine Learning Techniques

The project aims to achieve automatic land cover classification. To do so, it becomes extremely important to compare accuracy of different models and consider the best out of them. Supervised learning is the machine learning task of learning a function that maps an input to an output on the basis of training data.[26, 17] The training data is the set of input-output example pairs used to train the classifier. If enough data is available then the trained model usually identifies the underlying data patterns and is then able to make predictions on previously unknown input. Supervised learning, however, faces the problem of overfitting when model is made very flexible in order to fit the underlying data and underfitting when the model is too rigid to fit the underlying data.

3.5 Support Vector Machines

A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples. In two dimensional space, this hyperplane is a line dividing a plane into two parts where in each class lay on either side.[22]

3.5.1 Types of SVMs

There are two kinds of support vector machines: Hard Margin Support Vector Machines and Soft Margin Support Vector Machines. Hard Margin Support Vector machines do not allow misclassification of data points, tend to overfit the data and are thus sensitive to outliers. Soft margin support vector machines allow slight misclassification of data points and penalize for each misclassification. For this project, soft margin support vector machines have been used to classify the data points.

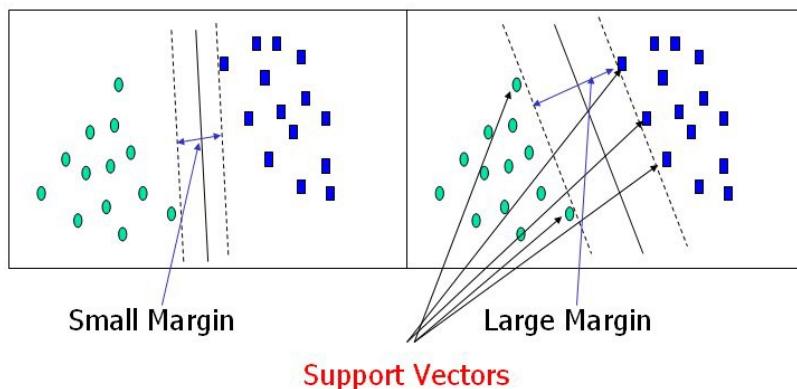


Figure 3.4: Support Vector Machine

3.5.2 Advantages

The advantages of using support vector machines are:

- Support vectors can deal with high dimensional data easily with the help of kernel transformations.
- They are only dependent on some part of the training data known as the support vectors which makes them very memory efficient.
- Can be used for a variety of tasks due to availability of a large number of kernel functions for decision making.
- Fast and memory efficient implementations in various languages is available via different libraries.

3.5.3 Disadvantages

The disadvantages of using support vector machines are:

- They do not take the spatial orientation of the pixels into account while learning.
- With wrong choice of kernel, support vector machines tend to overfit the data.
- Hard margin support vector machines are very sensitive to outliers and hence proper care should be taken while using them.

3.5.4 Implementation

After maximizing the separation between classes and minimizing the variance within classes, we have used support vector machines to find hyper-planes separating different classes. Soft margin support vector machines were used for this project. While finding the dividing hyper-plane, soft margin support vector machines allowed some mis-classification with penalty for each mis-classification.

Class weight were balanced for each class. This means that the penalty for mis-classification of an area like river is higher because its presence is lesser. One vs One method was used for multi-class classification therefore for classifying a point all the classes were taken into account- two at a time and then the point was classified to a class getting maximum votes. Linear Kernel was preferred after trying other kernels due to its higher accuracy. For internal calculations of maxima and minima, dual was used of Lagrange's Multipliers were used. The support vector machine was implemented from sci-kit learn library.

3.6 Decision Trees

A decision tree is a decision support tool that uses a tree-like model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm that only contains conditional control statements. Decision trees are commonly used in operations research, specifically in decision analysis, to help identify a strategy most likely to reach a goal, but are also a popular tool in machine learning. [4] A decision tree is a flowchart-like structure in which each internal node represents a "test" on an attribute (e.g. whether a coin flip comes up heads or tails), each branch represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes). The paths from root to leaf represent classification rules. Decision trees can also be used for calculating conditional probabilities.

3.6.1 Advantages

- Since decision trees are very intuitive and are frequently used in real life they are easy to understand.
- Decision trees can provide important insights like costs, probabilities and alternatives.
- Worst and best case values can easily be identified from the leaves of decision trees.

3.6.2 Disadvantages

- Decision trees are not very stable and can not be used with frequently changing data as a small change in data can lead to very large changes in the decision tree structure.
- The simple learning technique used by decision trees can sometimes be really inaccurate and hence random forests are used instead. However, random forests become very complex to understand as compared to decision trees.
- Information gain in decision trees is biased. More levels generally imply more information gain.

3.6.3 Implementation

NDVI values have different ranges for different classes, however when analysis is done over a period of time, things become complex. Due to such a range wise underlying structure it decision trees can make a good model for classification as they also have an if else type structure. The criterion used for choosing the node to split data at every subsequent path is Gini. Gini is calculated as per the given formula:

$$Gini = 1 - \sum_{i=1}^C (p_i)^2$$

Entropy is another impurity calculation method, however Gini was used as model based on Gini calculations gave higher accuracy.

3.7 K-Nearest Neighbors

3.7.1 Overview

The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other. In KNN, training examples are multi-dimensional feature vectors. Training process in KNN is very efficient as only the training points and their labels need to be stored. The number of classes K , to be used for classification can be chosen by the user. There is no specific method that can predict the ideal k for maximizing the accuracy and hence several values of K need to be tried. An unlabelled instance is assigned to a class which is most frequent among the k nearest training points to that example. Notice in the image above that most of the time, more similar data

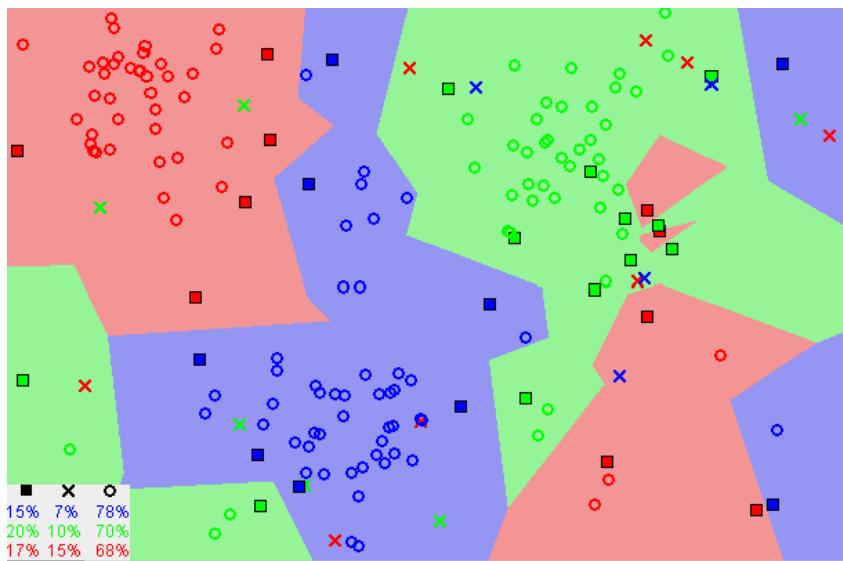


Figure 3.5: K-Nearest Neighbor Visualization

points are closer. The KNN algorithm hinges on this assumption being true enough for the algorithm to be useful. KNN captures the idea of similarity (sometimes called distance, proximity, or closeness) with some mathematics we might have learned in our childhood— calculating the distance between points on a graph. There are other ways of calculating distance, and one way might be preferable depending on the problem we are solving. However, the straight-line distance (also called the Euclidean distance) is a popular and familiar choice.

3.7.2 Advantages

- Extremely easy to implement.

- Can be trained on a large set of data unlike other classifiers like SVM.
- KNN is an ideal choice for certain classification tasks like identification of children etc. involving similarity in different features.
- Training KNN is a very fast processes as only the training examples and their labels need to be stored.

3.7.3 Disadvantages

- Compared to other classifiers KNN is very naive.
- Classifications errors can occur when the data is skewed i.e when there are more examples of one class than another.
- KNNs need sufficient training data for classification with proper accuracy.

3.7.4 Implementation

K-Nearest Neighbors classifies a point to a class in which maximum of its closest neighbors belong to. Over a period of time, pixels belonging to the same classes tend to have similar features and hence they are closer. Thus, K-Nearest Neighbors can be a very accurate method to classify the pixels. The distance calculated here is the simple Euclidean Distance. Its formula is as follows:

$$d(p, q) = d(q, p) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2} = \sqrt{\sum_{i=1}^n (q_i - p_i)^2}$$

The vote of each class point has uniform weight-age.

3.8 U-Net Architecture

Proposed by Ronneberger *et al.*[24], the U-Net architecture builds upon and extends the FCN architecture for more precise segmentation using a smaller training dataset. While the architecture was designed particularly for bio-medical segmentation applications, it has shown to work well for other domains too.

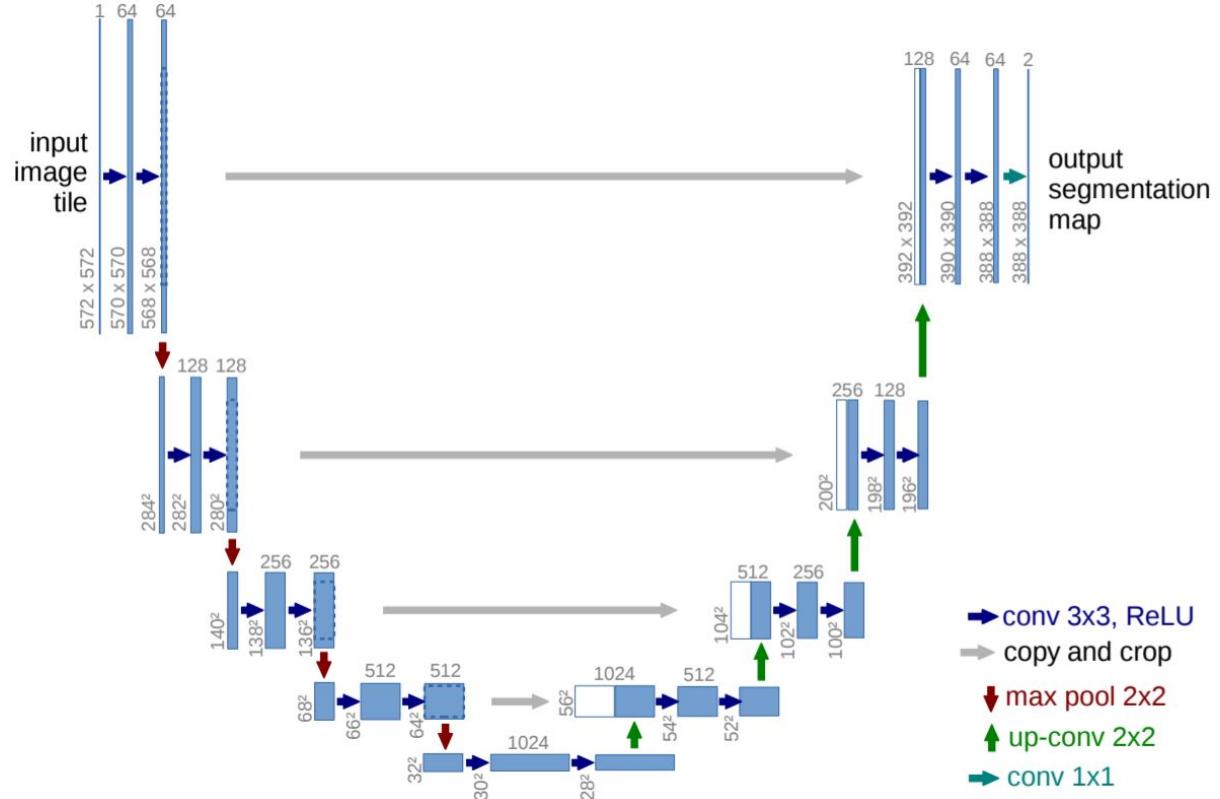


Figure 3.6: The U-Net architecture.
[24]

3.8.1 Components

The U-Net architecture can be divided into three major parts:

1. The down-sampling path
2. The bottleneck
3. The up-sampling path

Down-sampling Path

It is made up of four identical blocks. Each block encloses the following sequence of layers:

1. Convolutional layer ($f = 3, s = 1, p = 0$)
2. Convolutional layer ($f = 3, s = 1, p = 0$)
3. Max Pooling layer ($f = 2, s = 2$)

At each down-sampling step (max-pooling), the number of channels is doubled, starting with 64 channels in the first block. The down-sampling path captures not only features relevant to classification, but also contextual information about the location of different segments.

Bottleneck

It is made up of 2 layers:

1. Convolutional layer ($f = 3, s = 1, p = 0$)
2. Convolutional layer ($f = 3, s = 1, p = 0$)

Up-sampling Path

It is made up of four identical blocks. Each block encloses the following sequence of layers:

1. Up-sampling transpose convolution layer ($f = 2, s = 2$)
2. Concatenation with appropriately cropped feature map from down-sampling path
3. Convolutional layer ($f = 3, s = 1, p = 0$)
4. Convolutional layer ($f = 3, s = 1, p = 0$)

At each up-sampling step (transpose convolution), the number of channels is halved. The output volume after up-sampling is concatenated with the corresponding volume from the down-sampling path after adequate cropping. The introduction of these skip connections allows the propagation of context information about the localization from the down-sampling path to the up-sampling path which already carries semantic information. This helps the network to spread out activations correctly when up-sampling and lends the network its U-shape.

3.8.2 Advantages

The U-Net architecture presented above has several features which lend it better performance than the FCN architecture.

- In the FCN architecture, the single up-sampling layer is added at the end of the CNN and has limited number of channels (equal to the number of class labels). The U-Net architecture makes use of multiple up-sampling layers (each with multiple feature channels) in the up-sampling path, interspersed with blocks of convolutional layers. This provides for the opportunity to add more skip connections and so, to fuse contextual and semantic information better, at the time of up-sampling.
- Concatenation is used for combining information from skip connections, as opposed to the summing operation used in the FCN architecture. This helps to retain more information from both sets of activations, without polluting either.
- The U-Net architecture is also particularly well suited to applications where labelled training data is not available in abundance. This makes it more attractive for our project on account of the limited computational resources and time available for training the network.

3.8.3 Disadvantages

- Convolutional neural networks do not take temporal variations directly into account. However, it is hoped that providing the network with training data from different time periods should allow it to capture some of the temporal trends.
- The performance of a deep neural network greatly depends on the amount of training data available. Data augmentation and regularization can help in alleviating this problem.

3.8.4 Implementation

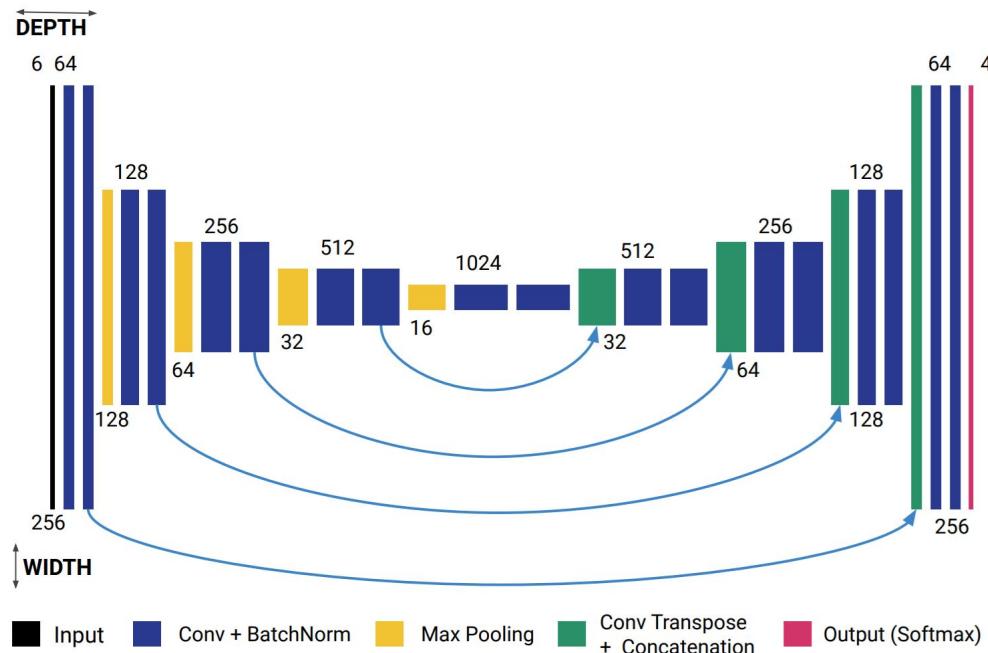
A convolutional neural network based on the above architecture was implemented using TensorFlow in Python. However, some changes were made to enhance the performance of the model.

1. Convolutional layers down-sample the image, leading to shrinkage of the image's spatial dimensions. This is particularly troublesome for skip-connections as concatenation of volumes at the time of up-sampling requires that the width and height

of the volumes match. Using **same padding** before applying the convolution operation prevents this down-sizing and solves the problem. This also enhances the symmetry in the network, with the output layer having the same spatial dimensions as the input image, reducing the overhead of image processing which would otherwise be required for training the network.

2. As a neural network is trained, the distribution of each layer's inputs changes continuously, as the parameters of previous layers change. This slows down learning and necessitates careful parameter initialization. This problem of internal covariate shift can be addressed by normalizing layer inputs to a learnable mean and variance, stabilizing the distribution of the activations. **Batch Normalization** allows us to use higher learning rates and makes the network more robust to the choice of hyper-parameters. Additionally, it also has a regularizing effect and helps prevent overfitting.
3. The shape of image accepted as input by the network was fixed to 256×256 . A patch size of 256 was found to provide sufficient spatial context to the network and fit well with the computational constraints.

The model implemented has 31,051,332 parameters, of which, 31,039,556 are trainable. The arrangement of different layers and shapes of output volumes can be seen in the figure below.



3.8.5 Training

Loss Function

Since the neural network implemented by us uses softmax activation in the output layer, **categorical cross entropy** was used as the loss function. This is the natural choice for any multi-class classification problem where the target labels are one-hot encoded.[3] The original paper by *Ronneberger et al.*[24] proposes the use of a pixel-wise loss weight to achieve better biomedical segmentation. However, this particular scheme was found to be irrelevant for the current project domain. The categorical cross entropy loss is computed as follows:

$$CE = -\lg \frac{\exp s_p}{\sum_j^C \exp s_j}$$

Optimizer

Adam (Adaptive Moment Estimation) is an optimization algorithm designed specifically for deep neural networks that uses adaptive learning rates for each parameter.[25] It computes exponentially weighted averages of gradients and squared gradients.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

where m_0 and v_0 are zero-initialized and g_t represents the gradient of the cost function with respect to parameter θ at step t . These are used for updating the parameters after every step of back-propagation.

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{v_t} + \epsilon} m_t$$

Adam reduces the oscillations in gradients associated with mini-batch gradient descent. The smoother loss curve allows for a larger learning rate and hence, quicker convergence. In limited experiments, Adam was found to produce better results for our network as compared to other optimizers such as stochastic gradient descent.

Hyper-parameters

We first trained the network with typical values for hyper-parameters on a small training set. A variance problem was observed as the accuracy for testing set was low. Regularization was not found to improve performance. However, providing the network with more data through augmentation was found to help considerably. Batch normalization

makes the network robust to the choice of hyper-parameters and so produced significant performance gains. On the whole, training the network for 70 epochs with a learning rate of 0.005 and default values for the other hyper-parameters was found to produce the best results.

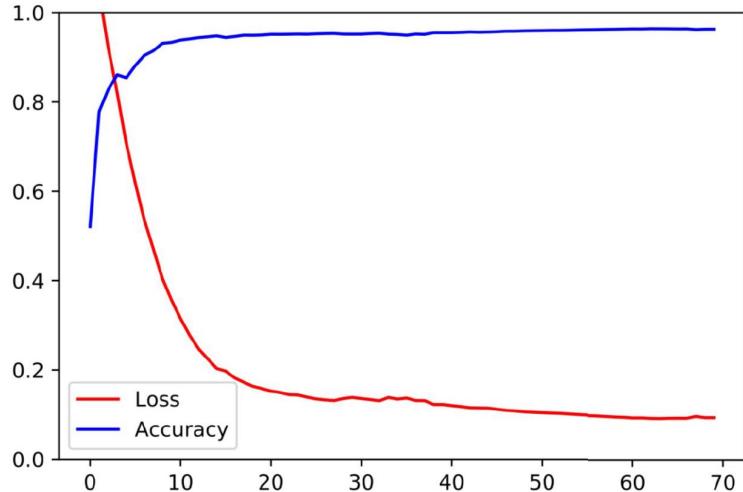


Figure 3.7: Curves for loss and accuracy against number of epochs.

4. Results

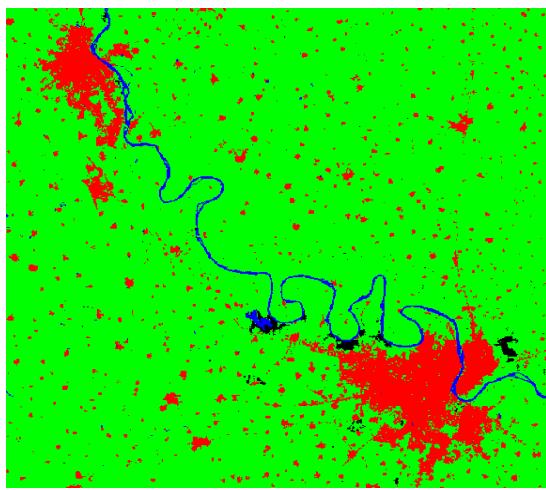
All four models described in the previous section were evaluated on the testing location - Ahmedabad and Gandhinagar, Gujarat. The region has sufficiently different characteristics from the training location to provide a good estimate of the quality of classification for any general region. Particularly, the seasonal river flow and large green areas in the cities make the region harder to classify correctly for any system. **Categorical accuracy** was chosen as the metric for evaluating the performance of the models as it is the natural choice for a multi-class classification problem.

Compared to other classifiers, support vector machines were found to be better at identifying forest areas. Certain urban areas like Fatehpur Sikri, Achhnera and Kiraoli were also very prominent in the output in case of SVM, but were found to be missing in most of the other classifiers. Support vector machines, however, mis-classified the small villages and towns as agricultural areas.

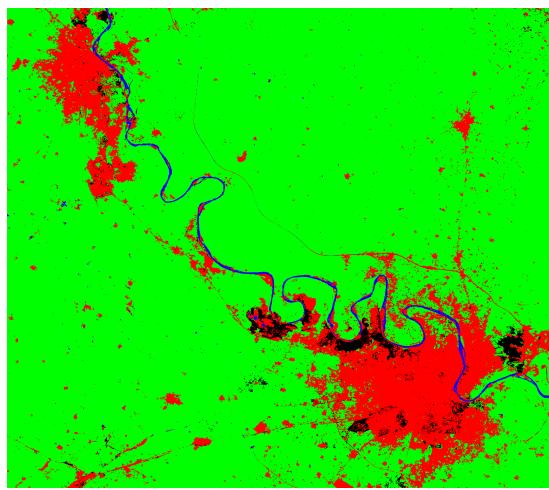
Decision trees had very high training accuracy. However, they were found to have lower accuracy on the testing dataset, which indicates overfitting. K-nearest neighbors had the lowest testing accuracy among all the classifiers. This might be mainly because this model uses a very naive approach of comparing the distance among the pixels for classification. K-nearest neighbors also mis-classified the seasonal river in the Ahmedabad District as urban area leading to lower accuracy.

Convolutional neural networks were best at identifying Keethal lake which was missed by other classifiers. Among all the classifiers, convolutional neural networks had the highest test accuracy and produced a much smoother output. This can be attributed to the fact that they take spatial variations directly into account, as opposed to the pixel-wise classification scheme used by the other models. Hence, we can conclude that out of all the classifiers, convolutional neural networks were best suited for the task of automatic land cover classification from temporal satellite images.

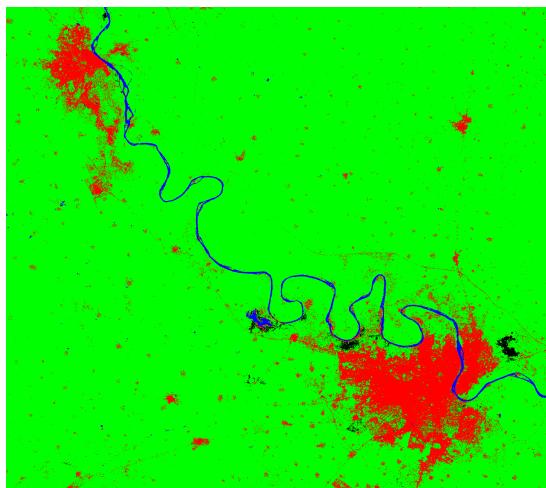
Model	Training Accuracy	Testing Accuracy
K-Nearest Neighbours	93.52%	75.53%
Decision Trees	98.18%	82.71%
Support Vector Machines	89.63%	87.37%
CNN (U-Net architecture)	95.06%	89.31%



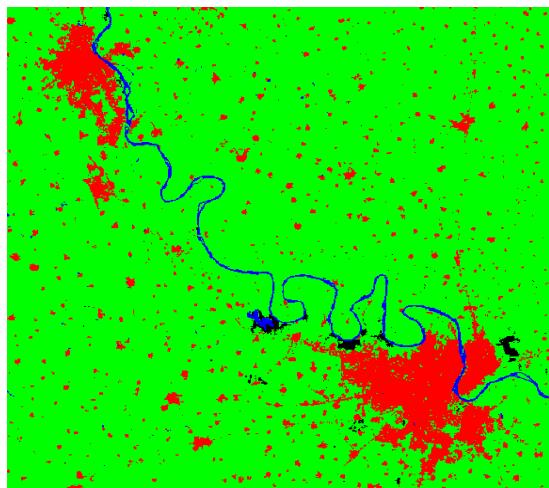
(a) Ground truth



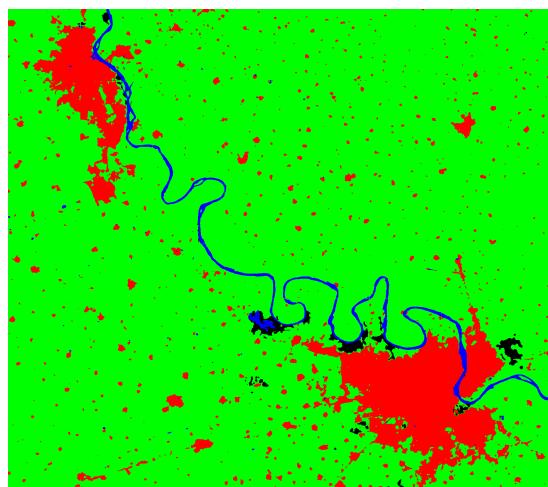
(b) Support Vector Machines



(c) K-Nearest Neighbors

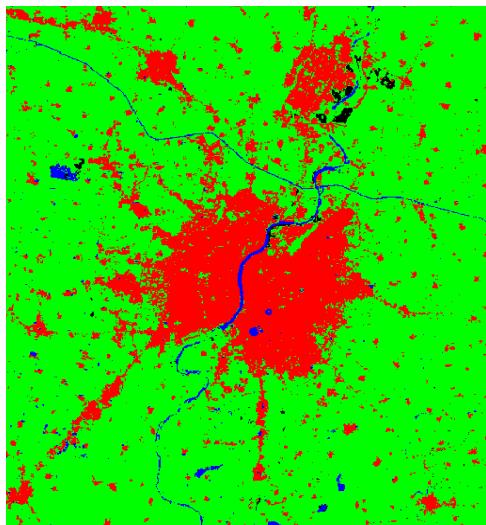


(d) Decision Tree

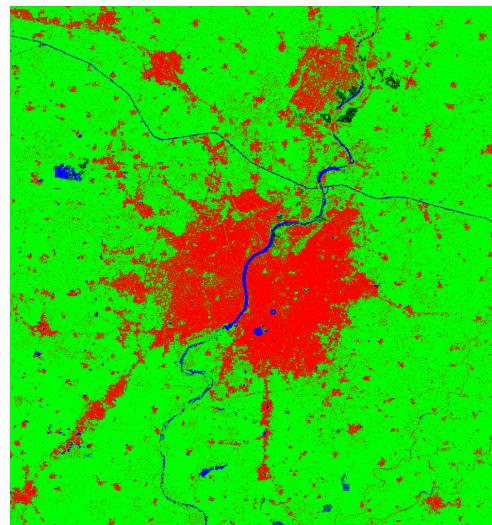


(e) U-Net

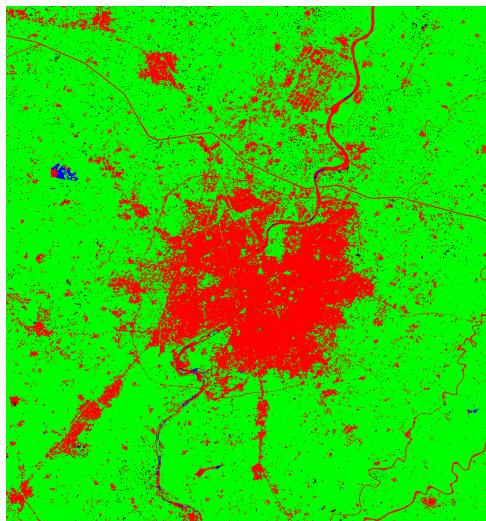
Figure 4.1: Ground truth and predictions for the **training location - Agra and Mathura, Uttar Pradesh**.



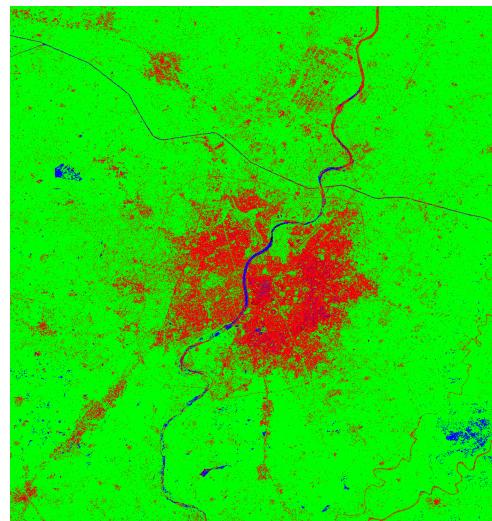
(a) Ground truth



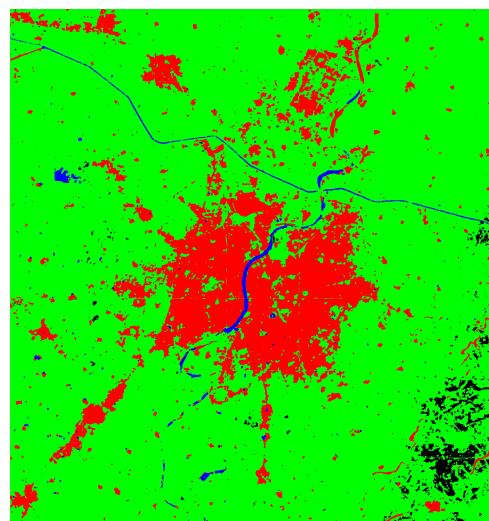
(b) Support Vector Machines



(c) K-Nearest Neighbors



(d) Decision Tree



(e) U-Net

Figure 4.2: Ground truth and predictions for the **testing location - Ahmedabad and Gandhinagar, Gujarat**.

5. Conclusion

Information of land cover classification is very crucial for applications like natural resource management, wildlife habitat protection, urban expansion, damage delineation, target land detection etc. With developments in technology, conventional method of field survey which is both inaccurate and time consuming has been replaced by remotely sensed imagery analysis which is highly accurate and covers wider range. Indian Space Research Organisation (ISRO) and Indian Institute of Remote Sensing (IIRS), Dehradun play an important role in providing information for these applications. Modern statistical and computing techniques like machine learning and deep learning minimize the human intervention in the task of analysis and classification, thus making the outcome very accurate and the process very agile.

The project completed herewith explores the applicability of machine learning and deep learning techniques to the problem of land cover classification and lays out the advantages and disadvantages of different approaches. Traditional machine learning models like support vector machines, decision trees, etc. provide a method for pixel-wise classification based on learning from the temporal variations in the NDVI time series. On the other hand, Convolutional neural networks, as implemented in our project, make use of both spatial and temporal variations for segmentation. However, they also tend to require considerably more training data to produce good results. The U-Net architecture resolves this problem to a considerable extent, especially when trained with some custom modifications on an augmented dataset. The high accuracy attained with limited training is a testament to the success of skip-connection based architectures for semantic segmentation. Meanwhile, traditional machine learning techniques were also found to produce good results, highlighting their relevance to image classification tasks.

5.1 Future Scope

The model has currently been trained on a small dataset consisting of satellite images of Agra and Mathura. Thus, the model has currently learnt the seasonal and crop variations in the Agra District area only. With proper training of seasonal and crop variations of an area, the model would be able to classify at other test locations as well with higher accuracy.

The classification task consisted of only four classes- urban, rural, forest and water-bodies. Models can also be extended to classify areas into other classes like wasteland,

arid-land, rural areas etc. Since these classes cannot be reliably accommodated in any of the existing four classes considered for the project, they increase the intra-class variability making the classification difficult and error-prone. Expanding the scope of the project to more land cover classes should, hence, help improve performance.

In case of convolutional neural networks, the classification was found to be particularly noisy and erroneous close to the boundaries of the image. This can be attributed to the lack of context at the corners and edges, due to less surrounding pixels. Some pre-processing techniques may be useful for artificially increasing the extent of the images to solve this problem.

If the above changes further increase the accuracy of the model then, the classification data can be extremely useful in tasks like natural resource management, urban expansion etc. This will also help in reducing the overheads in field survey method which is still being used in many parts of India.

Appendix

List of dependencies

Package	Version
python	3.6.9
numpy	1.18.5
pandas	1.0.4
matplotlib	3.2.1
pillow	7.0.0
gdal	2.2.2
scikit-image	0.16.2
scikit-learn	0.22.2.post1
tensorflow	2.2.0

Code snippets

Data Preparation

NDVI time series data

```
// Bounds of training and testing locations, marked using GEE Explorer
var agraReg = ee.Geometry.Polygon(
  [[[77.59987614966131, 27.551723242916378],
    [77.59987614966131, 27.092973428952195],
    [78.11760686255194, 27.092973428952195],
    [78.11760686255194, 27.551723242916378]]], null, false);

var ahmeda = ee.Geometry.Polygon(
  [[[72.35056899794004, 23.29854363055665],
    [72.35056899794004, 22.801931268739057],
    [72.81062148817442, 22.801931268739057],
    [72.81062148817442, 23.29854363055665]]], null, false);

// Import dataset, filter by region
var dataset = ee.ImageCollection('LANDSAT/LC08/C01/T1_8DAY_NDVI')
  .filterBounds(agraReg);

// Initialize image as single band, add bands corresponding to other periods
var image = dataset
  .filterDate('2015-01-01', '2015-02-28')
```

```

        .median().rename('n1');

image = image.addBands(dataset.filterDate('2015-03-01',
→ '2015-04-30').median().rename('n2'))
        .addBands(dataset.filterDate('2015-05-01',
→ '2015-06-30').median().rename('n3'))
        .addBands(dataset.filterDate('2015-08-01',
→ '2015-10-31').median().rename('n4'))
        .addBands(dataset.filterDate('2015-09-01',
→ '2015-10-31').median().rename('n5'))
        .addBands(dataset.filterDate('2015-11-01',
→ '2015-12-31').median().rename('n6'));

// Visualize image by displaying on map
Map.addLayer(image, {min:0.0, max:1.0}, 'NDVI');

// Export image to Google Drive
Export.image.toDrive({
  image:image,
  description:"NDVI_agra",
  region:agraReg,
  crs:"EPSG:4326",
  scale:30
});

```

Ground truth land cover data

```

function genClasses(image) {
  // select relevant bands
  var dc = image.select('discrete_classification');
  var tc = image.select('tree-coverfraction');
  var uc = image.select('urban-coverfraction');
  // water, if seasonal coverfraction > 25 or permanent coverfraction > 0
  var water = (image.select('water-seasonal-coverfraction').gt(25))
    .or(image.select('water-permanent-coverfraction').gt(0))
    .rename('water');
  // forest, if tree-coverfraction > 30 and not water
  var forest = tc.gt(30).and(water.not()).rename('forest');
  // buildup land, if urban-coverfraction > 40
  var built = uc.gt(40).and(water.not()).and(forest.not()).rename('urban');
  // agricultural land, if none of the above
  var agro = built.not().and(water.not()).and(forest.not()).rename('crop');
  // create band with labels for every class
}

```

```

var classes = built.multiply(1)
    .add(agro.multiply(2))
    .add(water.multiply(3))
    .add(forest.multiply(4))
    .rename('classes');

// add all bands to image
return image.addBands(built)
    .addBands(agro)
    .addBands(water)
    .addBands(forest)
    .addBands(classes);
}

```

Machine Learning techniques

Data Pre-processing

```

image = gdal.Open('/content/drive/My Drive/PS/data/Agra/NDVI_ahmeda_v2.tif')
ndvi = image.ReadAsArray()image = gdal.Open('/content/drive/My
↪  Drive/PS/data/Agra/NDVI_ahmeda_v2.tif')
ndvi = image.ReadAsArray()
ndvi_flat = ndvi.reshape(ndvi.shape[0], ndvi.shape[1] * ndvi.shape[2]).T
print(ndvi_flat.shape)
image = gdal.Open('/content/drive/My
↪  Drive/PS/data/Ahmedabad/LC_ahmeda_v1_labels.tif')
lc = image.ReadAsArray()
lc = lc - 1
lc_flat = lc.reshape(1, lc.shape[0] * lc.shape[1]).T
a=np.arange(1,ndvi_flat.shape[0]+1)
print(a)
X=pd.DataFrame(data=ndvi_flat[0:,0:],      # values
...           index=a,      # 1st column as index
...           columns=np.array(["month1",
↪  "month2","month3","month4","month5","month6"]))
X=preprocessing.scale(X)
Y=pd.DataFrame(data=lc_flat[0:,0:],
               index=a,
               columns=np.array(["class"]))
sk_lda=LinearDiscriminantAnalysis()
X_lda=sk_lda.fit_transform(X,Y)

```

Support Vector Machines

```
X_train, X_test, Y_train, Y_test=train_test_split(X_lda,Y,
→ test_size=0.2,random_state=43)
print(X_train.shape,X_lda.shape)
clf=svm.SVC(C=1.0, class_weight='balanced', dual=True, fit_intercept=True,
    loss='squared_hinge', max_iter=1000,
    multi_class='ovo', penalty='l2', random_state=None, tol=0.0001,
    verbose=0)
temp=clf.predict(X_lda)
c=confusion_matrix(Y,temp)
print(sum(c[i][i] for i in range(c.shape[0]))/X_lda.shape[0])
final=temp.reshape(lc.shape[0],lc.shape[1])
```

Decision Tree Classifier

```
clf2=tree.DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None,
→ criterion='gini',
    max_depth=None, max_features=None, max_leaf_nodes=None,
    min_impurity_decrease=0.0, min_impurity_split=None,
    min_samples_leaf=1, min_samples_split=2,
    min_weight_fraction_leaf=0.0, presort='deprecated',
    random_state=None, splitter='best')
clf2.fit(X_train,Y_train)\\
```

K-Nearest Neighbours Classifier

```
clf3=KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
    metric_params=None, n_jobs=None, n_neighbors=9, p=2,
    weights='uniform')
clf3.fit(X_train,Y_train)
c=confusion_matrix(Y,temp)
print(sum(c[i][i] for i in range(c.shape[0]))/X_lda.shape[0])
final=temp.reshape(lc.shape[0],lc.shape[1])
```

Convolutional Neural Network: U-Net Architecture

Patch generation and reconstruction

```
def get_patches(img_arr, size=256, stride=256, full_coverage=True):
    # check parameters
```

```

if size % stride != 0:
    raise ValueError("size % stride must be equal to 0.")

if img_arr.ndim != 3:
    raise ValueError("img_arr must be a 3 dimensional array with shape
                    (height, width, depth).")

# number of patches to be created
i_max = (img_arr.shape[0] - size) // stride + 1
j_max = (img_arr.shape[1] - size) // stride + 1
# extra patches created to completely cover image
extra_0 = int((img_arr.shape[0] - size) % stride != 0)
extra_1 = int((img_arr.shape[1] - size) % stride != 0)

if(not full_coverage):
    extra_0 = 0
    extra_1 = 0
patches_list = []

for i in range(i_max):
    for j in range(j_max):
        patches_list.append(
            img_arr[i * stride : i * stride + size, j * stride : j *
                    stride + size]
        )
    if(extra_1):
        patches_list.append(
            img_arr[i * stride : i * stride + size, img_arr.shape[1] -
                    size : img_arr.shape[1]]
        )
    if(extra_0):
        for j in range(j_max):
            patches_list.append(
                img_arr[img_arr.shape[0] - size : img_arr.shape[0], j * stride
                    : j * stride + size]
            )
    if(extra_1):
        patches_list.append(
            img_arr[img_arr.shape[0] - size : img_arr.shape[0],
                    img_arr.shape[1] - size : img_arr.shape[1]]
        )

```

```

    return np.array(patches_list)

def reconstruct_from_patches(img_arr, org_img_size, size, stride,
                           full_coverage=True):
    # check parameters
    if type(org_img_size) is not tuple:
        raise ValueError("org_image_size must be a tuple - (height, width).")

    if img_arr.ndim != 4:
        raise ValueError("img_arr must be a 4 dimensional array of shape
                         (num_patches, size, size, depth).")

    # number of patches along each dimension
    num_patches = img_arr.shape[0]
    print(num_patches)
    print(img_arr.shape)
    depth = img_arr.shape[3]
    i_max = (org_img_size[0] - size) // stride + 1
    j_max = (org_img_size[1] - size) // stride + 1
    # extra patches created for full coverage
    extra_0 = int((org_img_size[0] - size) % stride != 0)
    extra_1 = int((org_img_size[1] - size) % stride != 0)
    if(not full_coverage):
        extra_0 = 0
        extra_1 = 0

    # initialize image with zeros
    image = np.zeros(
        (org_img_size[0], org_img_size[1], depth), dtype=img_arr[0].dtype
    )
    patch_no = 0

    for i in range(i_max):
        for j in range(j_max):
            image[
                i * stride : i * stride + size,
                j * stride : j * stride + size,
            ] = img_arr[patch_no, :, :]
            patch_no += 1
    if(extra_1):

```

```

        image[
            i * stride : i * stride + size,
            org_img_size[1] - size : org_img_size[1]
        ] = img_arr[patch_no, :, :]
        patch_no += 1

    if(extra_0):
        for j in range(j_max):
            image[
                org_img_size[0] - size : org_img_size[0],
                j * stride : j * stride + size,
            ] = img_arr[patch_no, :, :]
            patch_no += 1

    if(extra_1):
        image[
            org_img_size[0] - size : org_img_size[0],
            org_img_size[1] - size : org_img_size[1]
        ] = img_arr[patch_no, :, :]
        patch_no += 1

return image

```

Setting up the model

```

# HYPERPARAMETERS
p = {
    'SIZE' : 256, 'STRIDE' : 256, 'FULL_COVER' : True,
    'NUM_BANDS' : 6, 'NUM_CLASSES' : 4,
    'SAME_PADDING' : True, 'BATCH_NORM' : True,
    'DROPOUT' : False, 'DROPOUT_RATE': 0.0,
    'EPOCHS' : 70, 'BATCH_SIZE' : 8, 'LEARNING_RATE' : 0.005,
    'BETA_1' : 0.99, 'BETA_2' : 0.999, 'EPSILON' : 1e-07
}
def unet(p):
    # Padding
    if(p["SAME_PADDING"]):
        pad = 'same'
    else:
        pad = 'valid'
    # Batch Norm
    batch_norm = p['BATCH_NORM']

```

```

# Dropout
dropout = p['DROPOUT']
drop_rate = p['DROPOUT_RATE']

# Input
input_shape = (p['SIZE'], p['SIZE'], p['NUM_BANDS'])
input = Input(shape=input_shape)
x = input

# Encoder (Down-sampling path)
filters = 64
downLayers = []
for i in range(4):
    x = Conv2D(filters, (3, 3), strides=(1, 1), padding='pad',
               activation='relu', use_bias=not batch_norm)(x)
    if(batch_norm):
        x = BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001)(x)
    x = Conv2D(filters, (3, 3), strides=(1, 1), padding='pad',
               activation='relu', use_bias=not batch_norm)(x)
    if(batch_norm):
        x = BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001)(x)
    downLayers.append(x)
    x = MaxPool2D(pool_size=(2, 2), strides=(2, 2), padding='pad')(x)
    filters *= 2

# Bottleneck
x = Conv2D(filters, (3, 3), strides=(1, 1), padding='pad',
               activation='relu', use_bias=not batch_norm)(x)
if(batch_norm):
    x = BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001)(x)
if(dropout):
    x = Dropout(rate=drop_rate)(x)
x = Conv2D(filters, (3, 3), strides=(1, 1), padding='pad',
               activation='relu', use_bias=not batch_norm)(x)
if(batch_norm):
    x = BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001)(x)
if(dropout):
    x = Dropout(rate=drop_rate)(x)
downLayers.reverse()

# Decoder (Up-sampling path)

```

```

for dlayer in downLayers:
    filters /= 2
    x = Conv2DTranspose(filters, (2, 2), strides=(2, 2), padding=pad)(x)
    # Skip Connection
    cropDim = calcCropDims(int_shape(x), int_shape(dlayer))
    if(not p['SAME_PADDING']):
        crop = Cropping2D(cropDim)(dlayer)
    else:
        crop = dlayer
    x = Concatenate(axis=-1)([x, crop])
    x = Conv2D(filters, (3, 3), strides=(1, 1), padding=pad,
               activation='relu', use_bias=not batch_norm)(x)
    if(batch_norm):
        x = BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001)(x)
    x = Conv2D(filters, (3, 3), strides=(1, 1), padding=pad,
               activation='relu', use_bias=not batch_norm)(x)
    if(batch_norm):
        x = BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001)(x)

# Output
output = Conv2D(p['NUM_CLASSES'], (1, 1), strides=(1, 1), padding=pad,
               activation='softmax')(x)

model = tf.keras.Model(input, output, name='unet')

return model

# Create instance of model
model = unet(p)

# Compile with choice of optimizer, loss function and metrics for evaluation
model.compile(loss=tf.keras.losses.categorical_crossentropy,
               optimizer=tf.keras.optimizers.Adam(
                   learning_rate=p['LEARNING_RATE'], beta_1=p['BETA_1'],
                   beta_2=p['BETA_2'], epsilon=p['EPSILON']
               ), metrics=[tf.keras.metrics.CategoricalAccuracy()])
)

```

Loading datasets

```
# Training data
train_x = gdal.Open('/content/drive/My
→ Drive/PS/data/earth_ndvi_6band/NDVI_v2.tif')
train_x = np.array(train_x.ReadAsArray()).moveaxis(train_x, 0, -1)
train_shape = train_x.shape[:2]
train_x = get_patches(train_x, p['SIZE'], p['STRIDE'], p['FULL_COVER'])
train_y = gdal.Open('/content/drive/My
→ Drive/PS/data/earth_landcover/LC_final_v5.tif')
train_y = np.array(train_y.ReadAsArray()).moveaxis(train_y, 0, -1)
assert(train_y.shape[:2] == train_shape)
train_y = get_patches(train_y, p['SIZE'], p['STRIDE'], p['FULL_COVER'])
```

Data Augmentation

```
train_x_aug = []
train_y_aug = []
for i in range(train_x.shape[0]):
    train_x_aug.append(train_x[i])
    train_y_aug.append(train_y[i])
    img_x = train_x[i]
    img_y = train_y[i]
    for j in range(3):
        img_x = skt.rotate(img_x, 90)
        train_x_aug.append(img_x)
        img_y = skt.rotate(img_y, 90)
        train_y_aug.append(img_y)
train_x_aug = np.array(train_x_aug)
train_y_aug = np.array(train_y_aug)
```

Training the model

```
# Train model and record history
model_history = model.fit(train_x_aug, train_y_aug,
→ batch_size=p['BATCH_SIZE'], epochs=p['EPOCHS'])

def plotTrainingHistory(hist, EPOCHS):
    loss = hist['loss']
    cat_acc = hist['categorical_accuracy']
    epochs = range(EPOCHS)
```

```

plt.figure()
plt.plot(epochs, loss, 'r', label='Loss')
plt.plot(epochs, cat_acc, 'b', label='Accuracy')
plt.legend()
plt.show()

# Plot loss curve and accuracy against epochs
plotTrainingHistory(model_history.history, p['EPOCHS'])

```

Evaluating the model

```

def oneHotFromActivations(testset):
    """
    For vector at each pixel, sets element corresponding to predicted class to
    → 1, rest to 0.
    """
    for t in range(testset.shape[0]):
        for h in range(testset.shape[1]):
            for w in range(testset.shape[2]):
                maxcell = np.max(testset[t][h][w])
                for cell in range(testset.shape[3]):
                    if(testset[t][h][w][cell] == maxcell):
                        testset[t][h][w][cell] = 1
                    else:
                        testset[t][h][w][cell] = 0
    return testset

def viewPredictions(pred, img_shape, stride, size):
    """
    Takes output of model.predict().
    One hot encodes the predictions, restitches patches to get original image,
    → returns a RGB visual representation of predictions.
    """
    pred = oneHotFromActivations(pred)
    pred = np.int32(pred) * 255
    pred = np.moveaxis(pred, -1, 0)[:3]
    pred = np.moveaxis(pred, 0, -1)
    return reconstruct_from_patches(pred, img_shape, stride, size)

def iou(preds, labels):

```

```

"""
Calculates Intersection over Union for predictions, comparing with ground
→ truth data.

Returns a dict containing class-wise IoUs, mean IoU, and categorical
→ accuracy.

"""

h, w = preds.shape[1:3]
num_classes = preds.shape[0]

intersection = []
union = []

for c in range(num_classes):
    intersection_c = np.sum((np.logical_and(preds[c] == 1, labels[c] ==
    → 1)))
    intersection.append(intersection_c)
for c in range(num_classes):
    union_c = np.sum((np.logical_or(preds[c] == 1, labels[c] == 1)))
    union.append(union_c)

intersection = np.array(intersection)
union = np.array(union)
return (intersection / union, intersection.sum() / (h * w))

def testPredictions(preds, labels, img_shape, size, stride):
    """
    preds as output by model.predict.
    labels as output by gdal, cast to a numpy array.
    img_shape must be a tuple - (height, width).
    Evaluates the predictions using accuracy and IoU.

    """

    pred_img = oneHotFromActivations(preds)
    pred_img = reconstruct_from_patches(pred_img, img_shape, size, stride)
    pred_img = np.moveaxis(pred_img, -1, 0)
    ious, accuracy = iou(pred_img, labels)
    return {
        'class_iou': ious,
        'mean_iou': ious.mean(),
        'accuracy': accuracy
    }

```

```
# Visualize predictions
test_pred = model.predict(test_x)
test_pred_img = viewPredictions(test_pred, test_shape, p['SIZE'], p['STRIDE'])
plt.imshow(test_pred_img)

# Evaluate predictions
test_y = gdal.Open('/content/drive/My
↪ Drive/PS/data/earth_landcover/LC_ahmeda.tif')
test_y = np.array(test_y.ReadAsArray())
print(testPredictions(test_pred, test_y, test_shape, p['SIZE'], p['STRIDE']))
```

References

- [1] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: <https://www.tensorflow.org/>.
- [2] Binod Baniya et al. “A Review of Green Roofs to Mitigate Urban Heat Island and Kathmandu Valley in Nepal”. In: 6 (Nov. 2018), pp. 137–152. DOI: 10.12691/aees-6-4-5.
- [3] Raúl Gómez Bruballa. *Understanding Categorical Cross-Entropy Loss, Binary Cross-Entropy Loss, Softmax Loss, Logistic Loss, Focal Loss and all those confusing names*.
- [4] *Decision Trees-A simple way to visualize a decision*. URL: <https://medium.com/greyatom/decision-trees-a-simple-way-to-visualize-a-decision> (visited on 06/21/2020).
- [5] Ehsan Fathi and Babak Maleki Shoja. “Deep neural networks for natural language processing”. In: *Handbook of statistics*. Vol. 38. Elsevier, 2018, pp. 229–316. (Visited on 06/02/2020).
- [6] Google. *Google Colaboratory - frequently asked questions*. URL: <https://research.google.com/colaboratory/faq.html> (visited on 06/20/2020).
- [7] Google. *Google Earth Engine - Copernicus Global Land Cover*. URL: https://developers.google.com/earth-engine/datasets/catalog/COPERNICUS_Landcover_100m_Proba-V_Global (visited on 06/05/2020).
- [8] Google. *Google Earth Engine - Landsat 8*. URL: https://developers.google.com/earth-engine/datasets/catalog/LANDSAT_LC08_C01_T1_SR (visited on 06/05/2020).
- [9] Google. *Google Earth Engine - Platform Overview*. URL: <https://earthengine.google.com/platform/> (visited on 06/04/2020).
- [10] Xiaobing Han et al. “Pre-trained alexnet architecture with pyramid pooling and supervision for high spatial resolution remote sensing image scene classification”. In: *Remote Sensing* 9.8 (2017), p. 848.
- [11] *History—Indian Institute of Remote Sensing*. URL: <https://www.iirs.gov.in/historyandobjectives> (visited on 06/01/2020).
- [12] *Indian Institute of Remote Sensing*. URL: <https://www.iirs.gov.in/institute-profile> (visited on 06/01/2020).

- [13] Manideep Kolla, Aniket Mandle, and Apoorva Kumar. *Eye in the sky*. GitHub Page. 2018. URL: <https://github.com/manideep2510/eye-in-the-sky> (visited on 06/01/2020).
- [14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.
- [15] Fei-Fei Li, Justin Johnson, and Serena Yeung. *cs231n - Lecture Slides*. 2017. URL: <http://cs231n.stanford.edu/slides/2017/> (visited on 06/01/2020).
- [16] Jonathan Long, Evan Shelhamer, and Trevor Darrell. “Fully convolutional networks for semantic segmentation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 3431–3440.
- [17] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. In: *Foundations of Machine Learning*, 2012.
- [18] Shivaprakash Muruganandham. *Modern Applied Statistics with S*. 2016.
- [19] Shivaprakash Muruganandham. *Semantic segmentation of satellite images using deep learning*. 2016.
- [20] Andrew Ng and Kian Katanforoosh. *cs229 - Lecture Notes*. 2018. URL: <http://cs229.stanford.edu/notes/> (visited on 06/01/2020).
- [21] Andrew Ng, Kian Katanforoosh, and Younes Bensouda Mourri. *Convolutional Neural Networks*. 2017. (Visited on 06/02/2020).
- [22] Savan Patel. *Chapter 2: SVM (Support Vector Machines)*. 2017. (Visited on 06/05/2020).
- [23] Fabian Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *CoRR* abs/1201.0490 (2012). arXiv: 1201 . 0490. URL: <http://arxiv.org/abs/1201.0490>.
- [24] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-net: Convolutional networks for biomedical image segmentation”. In: *International Conference on Medical image computing and computer-assisted intervention*. Springer. 2015, pp. 234–241.
- [25] Sebastian Ruder. *An overview of gradient descent optimization algorithms*. URL: <https://ruder.io/optimizing-gradient-descent/index.html#adam> (visited on 06/21/2020).
- [26] Stuart J. Russell and Peter Norvig. In: *Artificial Intelligence: A Modern Approach, Third Edition*. 2010.
- [27] Jürgen Schmidhuber. “Deep learning in neural networks: An overview”. In: *Neural networks* 61 (2015), pp. 85–117.

- [28] Robert A. Schowengerdt. In: *Remote sensing: models and methods for image processing* (3rd ed.). 2007.
- [29] Yashwant Singh. *Significance of land cover/land use*. URL: <https://www.satpalda.com/blogs/significance-of-land-use-land-cover-lulc-maps> (visited on 06/05/2020).
- [30] United States Geological Survey. *Landsat 8*. URL: <https://www.usgs.gov/land-resources/nli/landsat/landsat-8> (visited on 06/05/2020).

List of Abbreviations

CNN	Convolutional Neural Networks
ET	Evapotranspiration
FCN	Fully Convolutional Neural Networks
GEE	Google Earth Engine
GPU	Graphical Processing Unit
IIRS	Indian Institute of Remote Sensing
NDVI	Normalized Difference Vegetation Index
NIR	Near InfraRed
NN	Neural Networks.
SVM	Support Vector Machines
KNN	K-Nearest Neighbours

Glossary

Chlorophyll green photosynthetic pigment found in plants

Convolution mathematical way of combining two signals to form a third signal

Epoch one complete pass through the entire training set

Kernel Kernel functions.

Pixel a minute area of illumination on a display screen, one of many from which an image is composed.

Pooling process of extracting features from image output of a convolution layer

Semantic Segmentation the process of associating each pixel of an image with a class label

Temporal relating to time