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: 6th 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 guide: Dr. Hari Shanker Srivastava
Designation: 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, urban, agricultural land and water bodies which can be used for various applications like natural resource management, urban expansion etc. Data from Agra district, Uttar Pradesh has been used to train Support Vector Machines and Convolutional Neural Networks which are then tested on Gandhinagar district, 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 the models and subsequently, the accuracy of these models is reported.

Contents

List of Figures

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. [9, 10]

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.[24] 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.[25]

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.

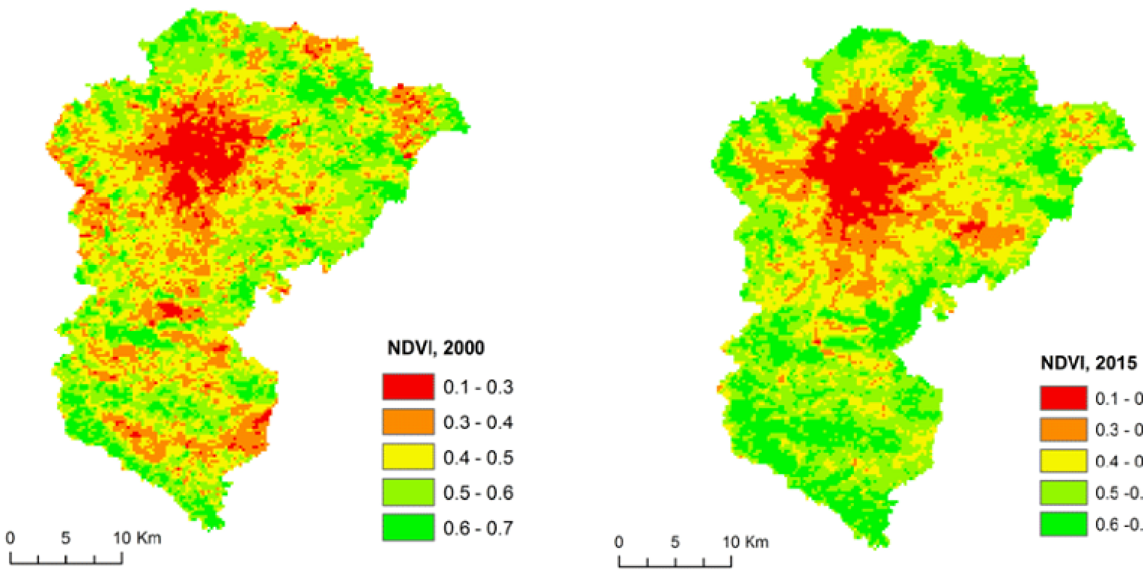


Figure 1.1: NDVI distribution in 2000 and 2015 in Kathmandu valley, Nepal, derived from MODIS data. Depicts changes in land cover/land use over time.
[2]

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.

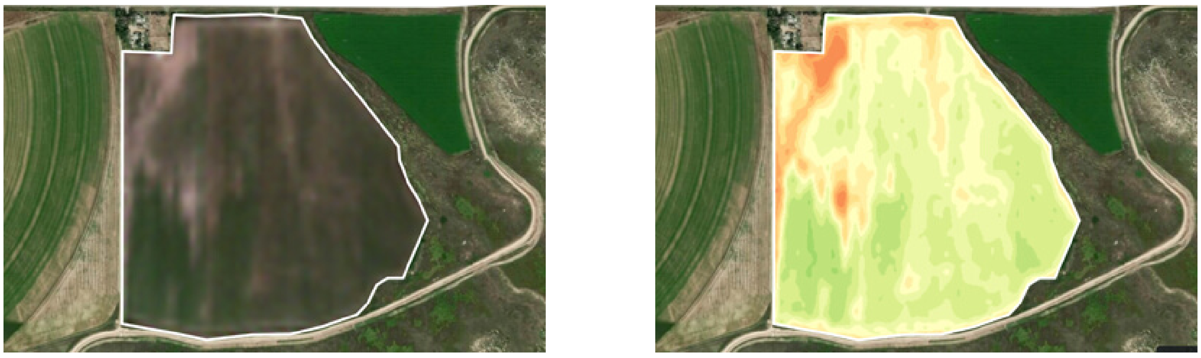


Figure 2.1: (1) Without NDVI. (2) With NDVI.

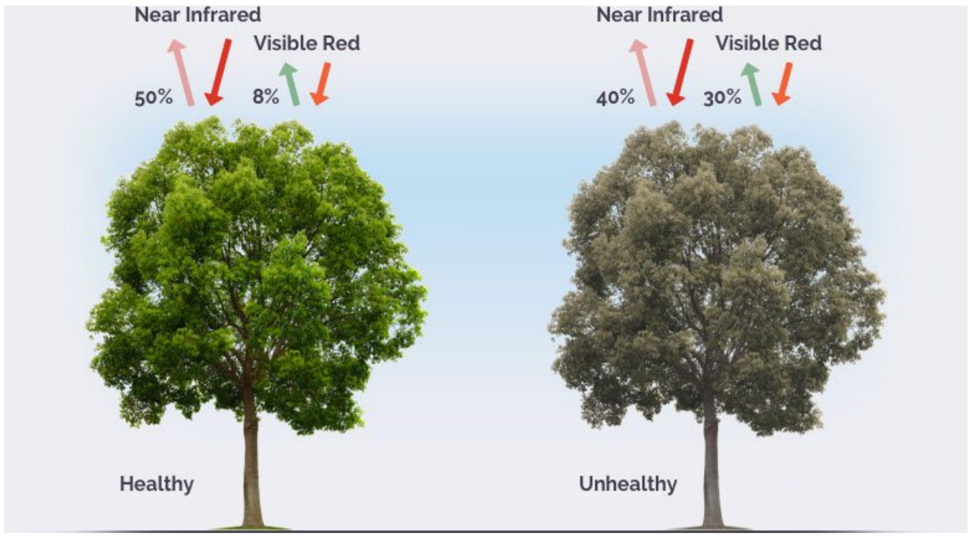


Figure 2.2: Plant health directly affects reflectance in NIR and RED bands.

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.[13]

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.[11]

2.2.2 Object Based Approach

The term "objects" represents meaningful semantic entities or scene components that are distinguishable in an image.[11] 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.[11]

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.[11, 13]

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 he network. Other neurons get activated through weighted connections from previously active neurons. [23] 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.[17]

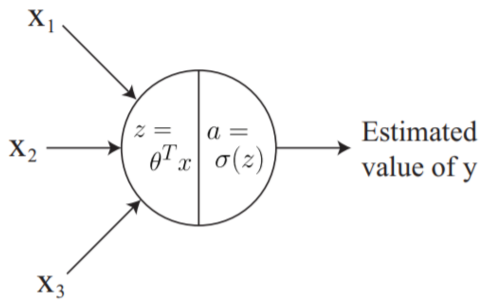


Figure 2.3: A single neuron.
[17]

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.

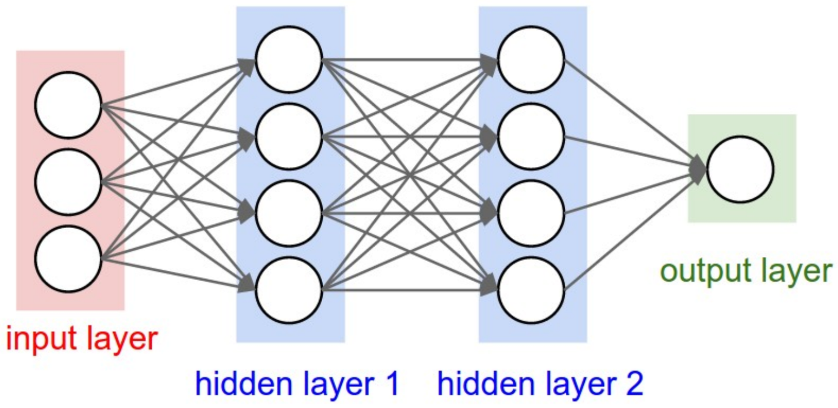


Figure 2.4: A two layer neural network with fully connected layers. [13]

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*200*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 layers 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.[13, 18] 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. [13, 16, 18]

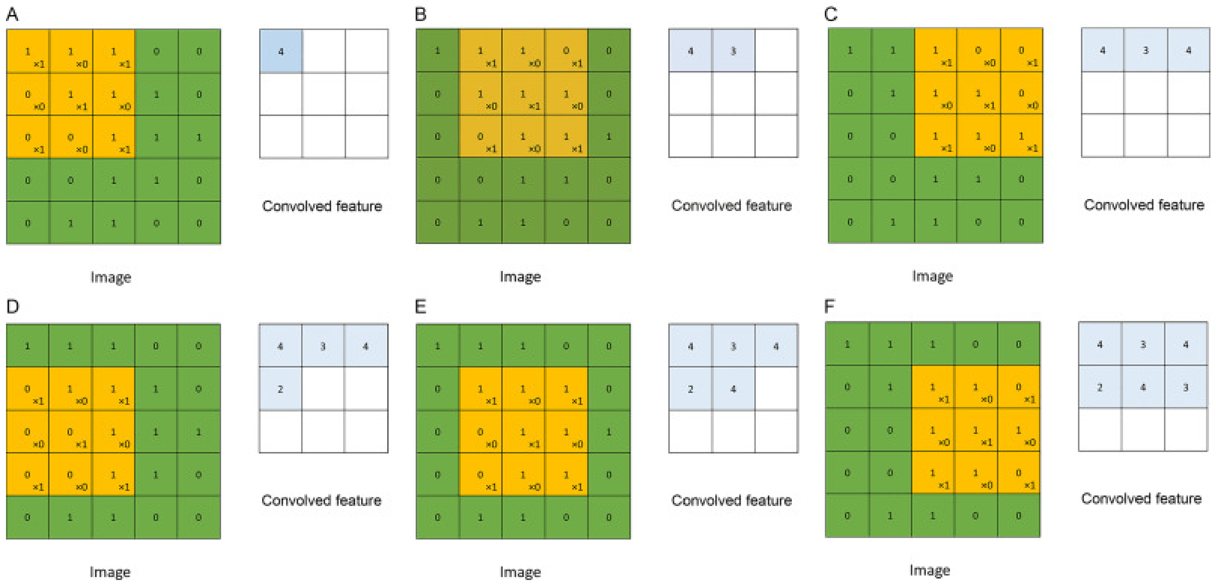


Figure 2.5: The convolution operation performed using a 3x3 filter on a 5x5x1 image. [3]

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.[18, 16] 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.[13] 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.[13] 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.

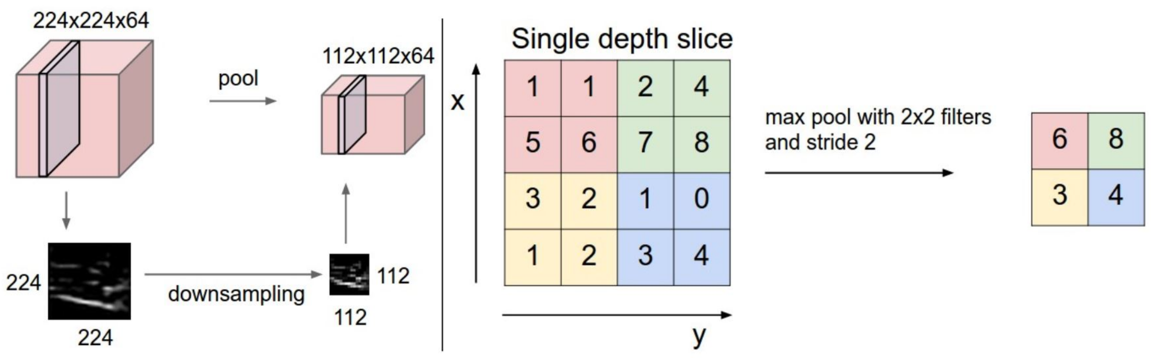


Figure 2.6: (1) A typical max pooling layer. (2) The max pooling operation. [13]

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.

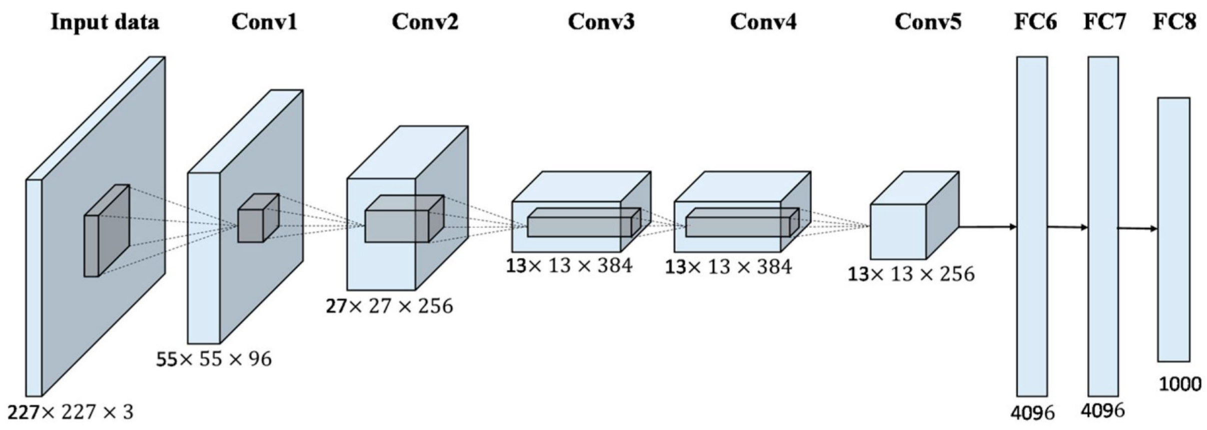


Figure 2.7: AlexNet - an influential NN architecture that popularized the use of CNNs and GPUs to accelerate deep learning.
[12, 8]

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. [14, 13, 21]

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**.[14] For up-sampling the classification maps produced by a FCN, *Long et al.*[14] 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.[14, 13] A stack of layers performing transpose convolution with an appropriate activation function can be used to learn a non-linear up-sampling.[14]

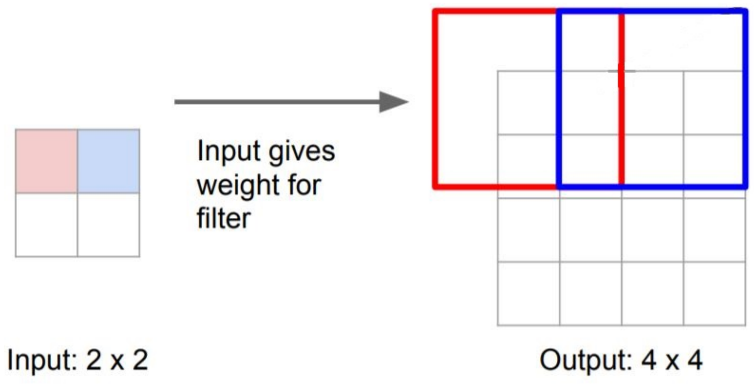


Figure 2.8: A 3×3 transpose convolution with stride 2 and padding 1. The output gets up-sampled by a factor of 2.
[13]

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.[14]

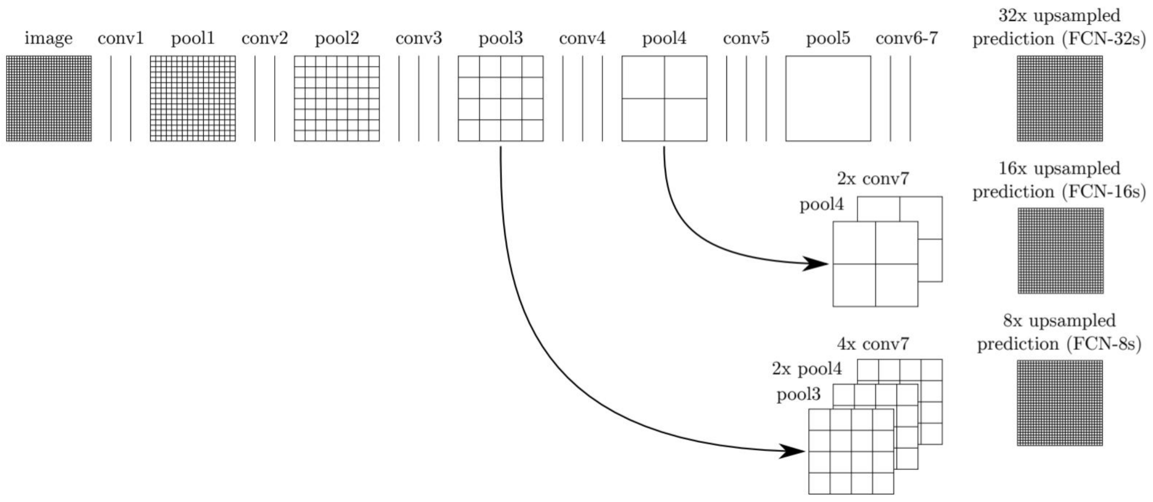


image
conv1
pool1
conv2
pool2
conv3
pool3
conv4
pool4
conv5
pool5
conv6-7
32x upsampled prediction (FCN-32s)
2x conv7
pool4
16x upsampled prediction (FCN-16s)
4x conv7
2x pool4
pool3
8x upsampled prediction (FCN-8s)

Figure 2.9: The different variants of the FCN architecture defined in [14], with different skip connections added to the base network.

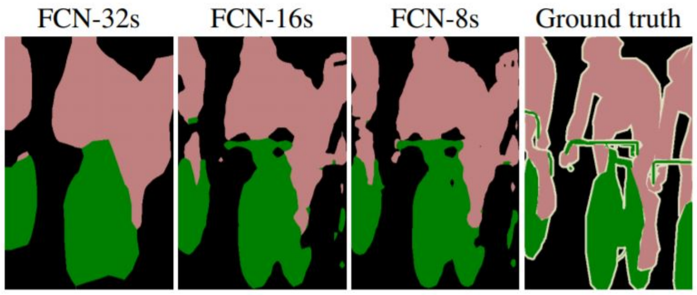


Figure 2.10: Different networks based on the FCN architecture.
[14]

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).[7] 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.[4] 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.[20]

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.[6, 26]

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.[5]

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.

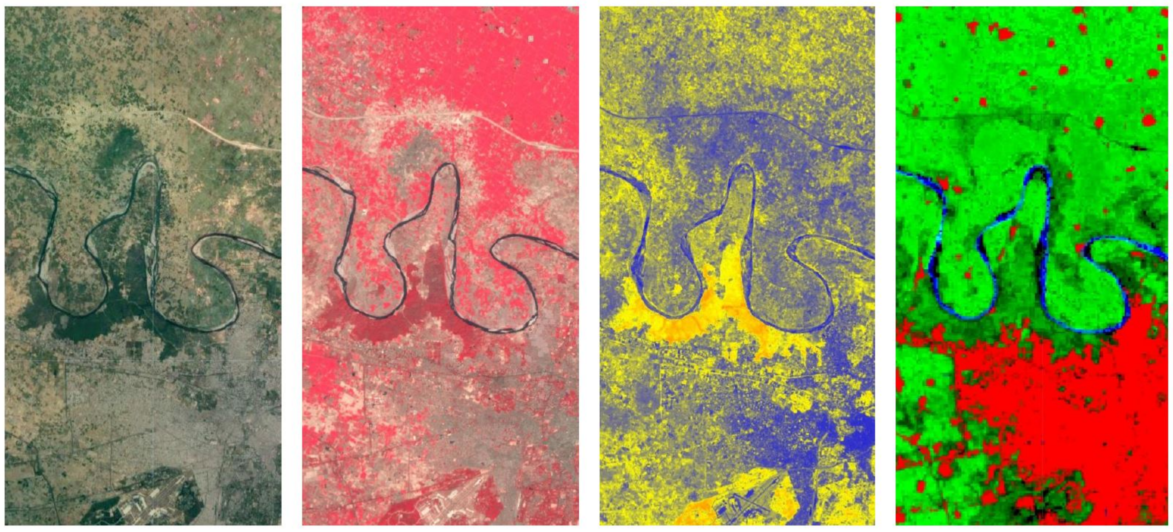


Figure 3.1: Different views of the training location - Agra. (1) True colour satellite image (2) False colour infrared image (3) Map of NDVI values (4) Ground truth land cover. [Google Earth Engine]

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

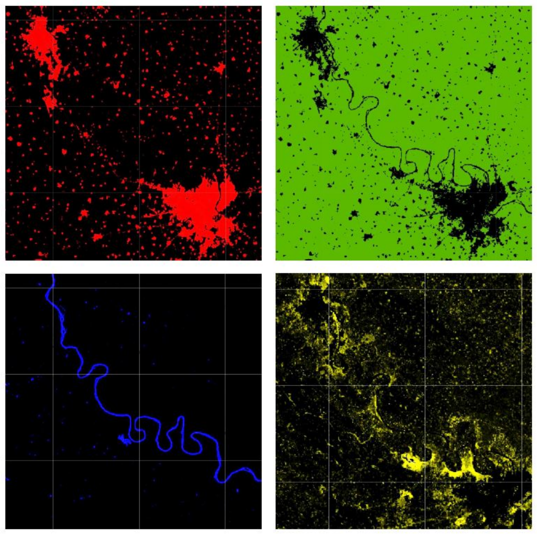


Figure 3.2: Masks for different land cover classes for Agra and Mathura region.

3.3 Data Preprocessing

3.4 Linear Discriminant Analysis

Linear discriminant analysis (LDA) is a generalization of Fisher's linear discriminant(which is only applicable to 2 class problems), a method used in statistics, pattern recognition, and machine learning to find a linear combination of features that characterizes or separates two or more classes of objects or events. LDA is also closely related to principal component analysis (PCA)-which is another dimensionality reduction approach. They both look for linear combinations of variables which best explain the data. LDA explicitly attempts to model the difference between the classes of data by maximizing between class separation and minimizing within class separation. PCA, in contrast, does not take into account any difference in class which makes LDA better 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 $\vec{x}$.

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 yy 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.

3.4.1 Project Specifics

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.

3.4.2 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 produces as output by the neural network, with a provision for handling overlapping patches.

3.4.3 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.

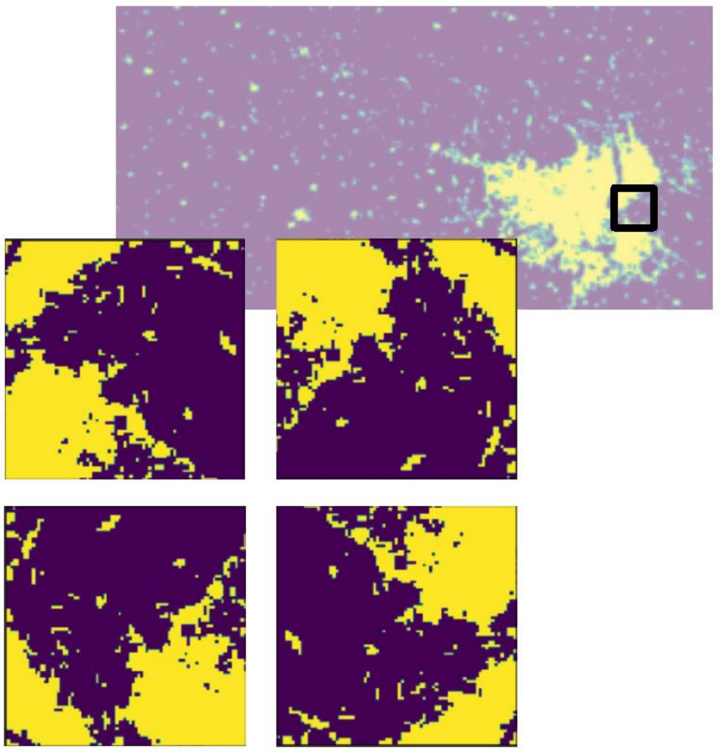


Figure 3.3: Patch generation and data augmentation.

3.5 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.

3.5.1 Support Vector Machines

3.5.2 Overview

Supervised learning is the machine learning task of learning a function that maps an input to an output on the basis of training data.[22, 15] 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 to rigid to fit the underlying data.

3.5.3 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.[19]

3.5.4 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.

3.5.5 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.

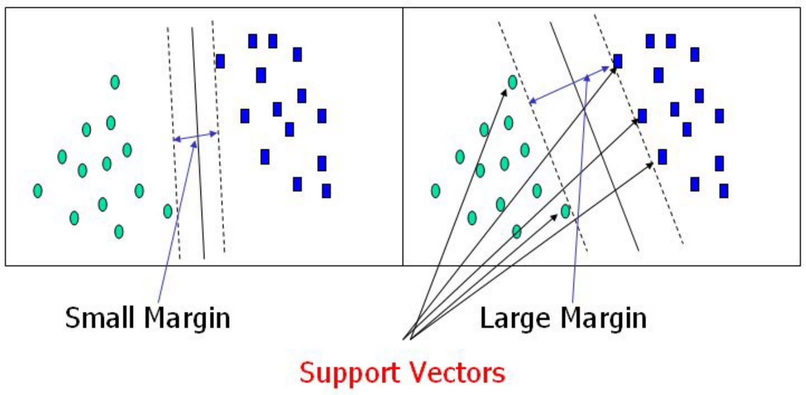


Figure 3.4: Support Vector Machine

- 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.

3.5.6 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 to overfit the data.
- Hard margin support vector machines are very sensitive to outliers and hence proper care should be taken while using them.

3.5.7 Project Specifics

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.

3.5.8 Decision Trees

3.5.9 Overview

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. 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.

In decision analysis, a decision tree and the closely related influence diagram are used as a visual and analytical decision support tool, where the expected values (or expected utility) of competing alternatives are calculated.

A decision tree consists of three types of nodes:

Decision nodes – typically represented by squares Chance nodes – typically represented by circles End nodes – typically represented by triangles Decision trees are commonly used in operations research and operations management. If, in practice, decisions have to be taken online with no recall under incomplete knowledge, a decision tree should be paralleled by a probability model as a best choice model or online selection model algorithm. Another use of decision trees is as a descriptive means for calculating conditional probabilities.

3.5.10 Advantages

- Are simple to understand and interpret. People are able to understand decision tree models after a brief explanation.

- Have value even with little hard data. Important insights can be generated based on experts describing a situation (its alternatives, probabilities, and costs) and their preferences for outcomes.

- Help determine worst, best and expected values for different scenarios.
- Use a white box model. If a given result is provided by a model.

3.5.11 Disadvantages

- They are unstable, meaning that a small change in the data can lead to a large change in the structure of the optimal decision tree.
- They are often relatively inaccurate. Many other predictors perform better with similar data. This can be remedied by replacing a single decision tree with a random forest of decision trees, but a random forest is not as easy to interpret as a single decision tree.
- For data including categorical variables with different number of levels, information gain in decision trees is biased in favor of those attributes with more levels.

3.5.12 Project Specifics

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: Entropy is another

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

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

3.5.13 K-Nearest Neighbors

3.5.14 Overview

The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other. Notice in the image above that most of the time,

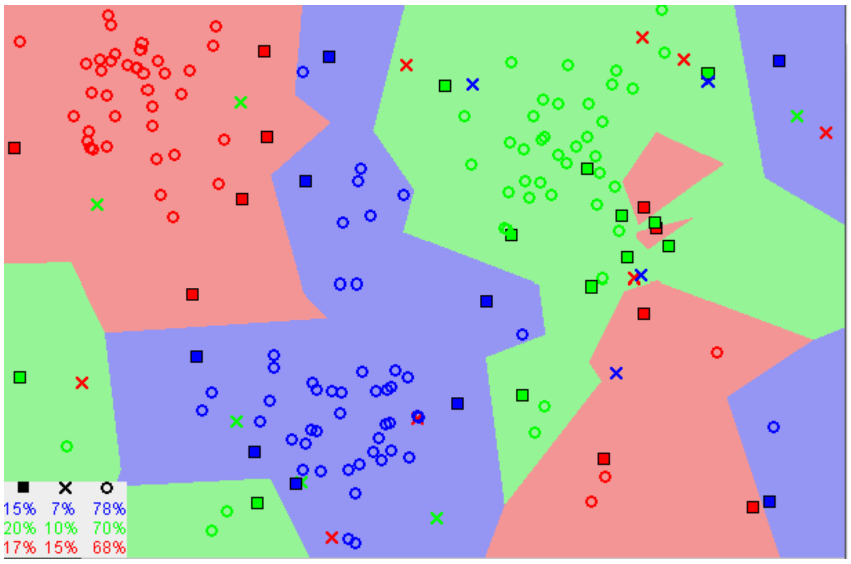


Figure 3.5: K-Nearest Neighbor Visualization

similar data points are close to each other. 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.

3.5.15 Project Specifics

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: The vote of each class point has uniform weightage.

3.6 U-Net Architecture

Proposed by *Ronneberger et al.*[21], 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.

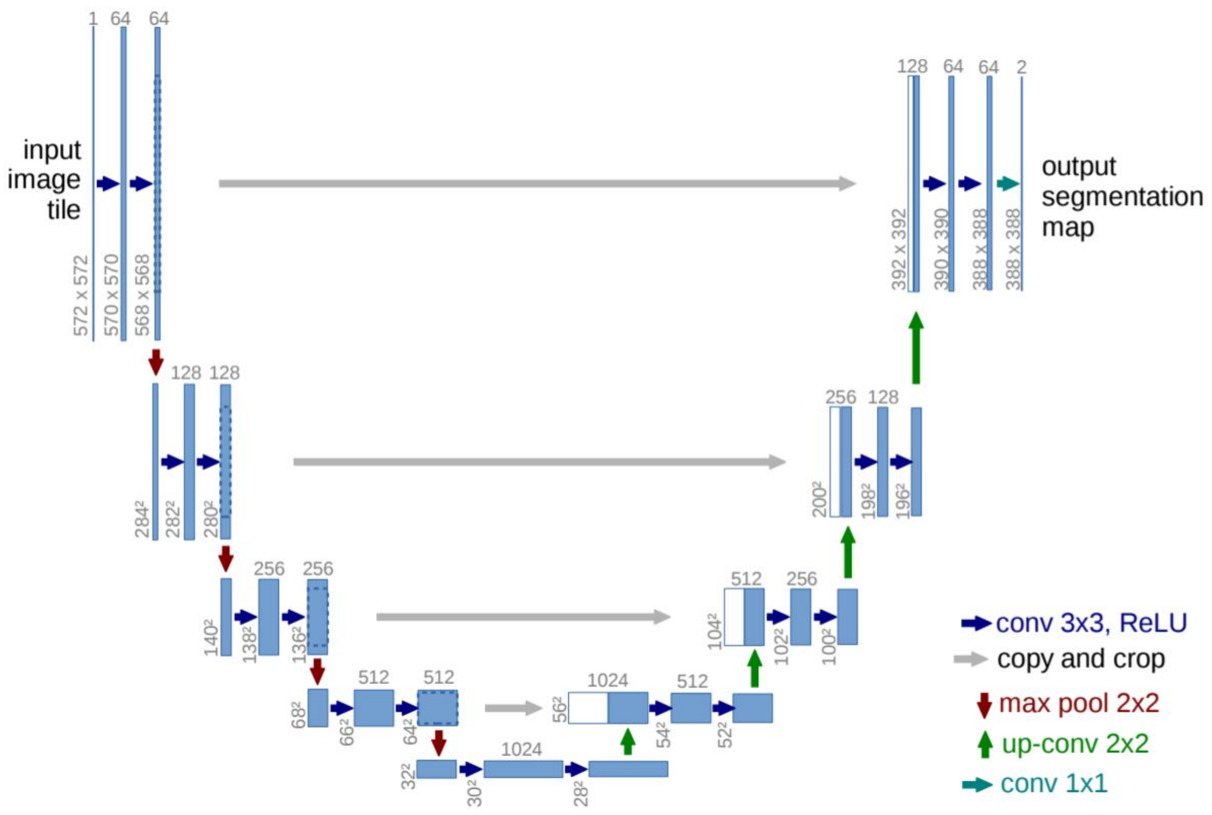


Figure 3.6: The U-Net architecture.
[21]

3.6.1 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.

3.6.2 Implementation

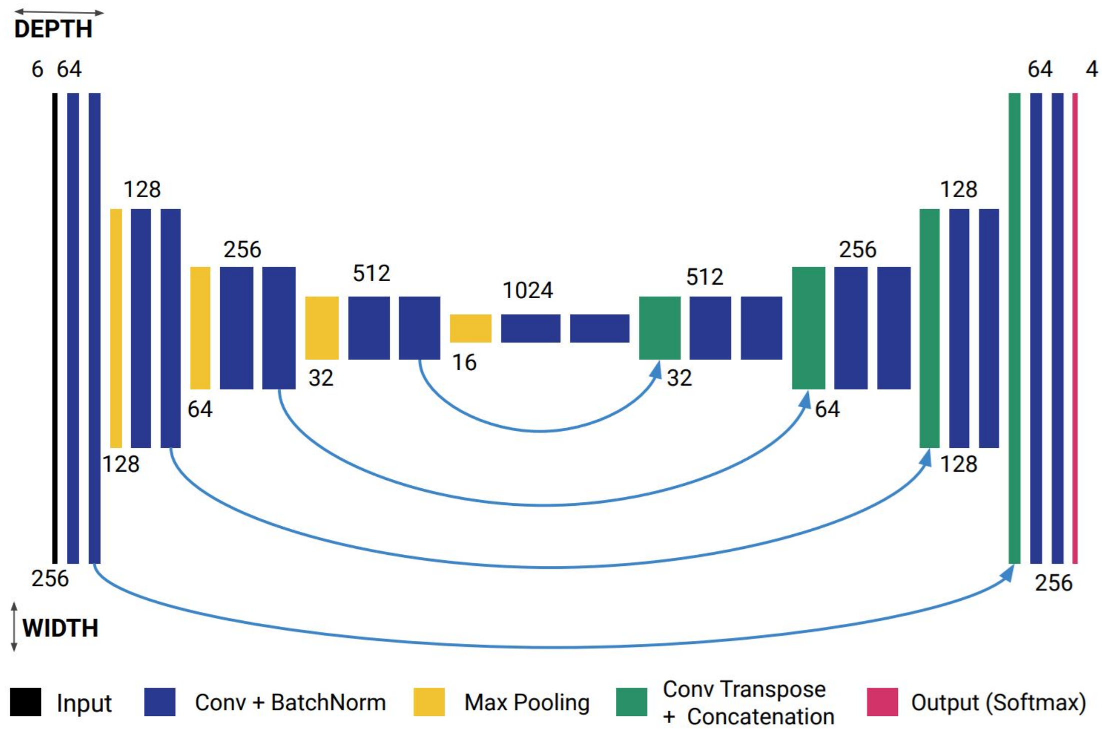
A convolutional neural network based on the above architecture was implemented using TensorFlow in Python. However, some changes were made to enhance the performance of the model.

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. A more detailed plot can be found in the Appendix.



3.6.3 Training

Loss Function

Optimizer

Hyper-parameters

3.6.4 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. We, hence, propose to use this CNN architecture for the problem of land cover classification.

3.6.5 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. We propose using techniques like data augmentation and transfer learning so as to get better results even with our small training set.

4. Results

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 seeks to compare the applicability of machine learning and deep learning techniques to the problem of land cover classification. Support Vector Machines are traditional machine learning models that 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 - a class of deep neural networks, use the spatial variations in the infrared and visible bands of the satellite image for segmentation. The two, hence represent very different approaches for solving the problem. However, the project is still at a very nascent stage, and any comment regarding the comparative performance of these two approaches cannot be made yet.

5.1 Future Scope

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

Preparing 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
});
```

NDVI time series 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');
  // builtup 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);
}

// Import dataset, filter by region
var lc = ee.ImageCollection("COPERNICUS/Landcover/100m/Proba-V/Global")
           .filterBounds(agraReg)
           .first();

// Generate classes
lc = genClasses(lc);

// Visualize image by displaying on map
var visParams = {min:0, max:4, bands:['classes'], palette:['white',
↪  'red', 'lawngreen', 'blue', 'black']};
Map.addLayer(lc, visParams, "LC");
```

```
// Export one hot encoded image to Google Drive
Export.image.toDrive({
  image:lc.select(['built', 'agro', 'water', 'forest']),
  description:"LC_agra",
  region:agraReg,
  crs:"EPSG:4326",
  scale:30
});
// Export labelled image to Google Drive
Export.image.toDrive({
  image:lc.select(['classes']),
  description:"LC_agra_labels",
  region:agraReg,
  crs:"EPSG:4326",
  scale:30
});
```

Convolutional Neural Network: U-Net Architecture

Import statements

```
import numpy as np
import gdal
from google.colab import files
from matplotlib import pyplot as plt
from PIL import Image
import skimage.transform as skt
import tensorflow as tf
from tensorflow.keras.layers import (
    Input,
    Conv2D,
    MaxPool2D,
    BatchNormalization,
    Dropout,
    Conv2DTranspose,
    Cropping2D,
    Concatenate
)
```

```
from tensorflow.keras.backend import int_shape
```

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
```

```python
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' : 20,
  'BATCH_SIZE' : 8,
  'LEARNING_RATE' : 0.001,
  '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())
train_x = np.moveaxis(train_x, 0, -1)
train_shape = train_x.shape[:2]
train_x_img = train_x
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())
train_y = np.moveaxis(train_y, 0, -1)
assert(train_y.shape[:2] == train_shape)
train_y_img = train_y
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'])
```

Evaluting 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_ious': 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']))
```

Sample for code

```
# CODE GOES HERE, PLS COPY AND LEAVE HERE
```

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] 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).

[4] Google. *Google Colaboratory - frequently asked questions*. URL: `https://research.google.com/colaboratory/faq.html` (visited on 06/20/2020).

[5] 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).

[6] 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).

[7] Google. *Google Earth Engine - Platform Overview*. URL: `https://earthengine.google.com/platform/` (visited on 06/04/2020).

[8] 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.

[9] *History—Indian Institute of Remote Sensing*. URL: `https://www.iirs.gov.in/historyandobjectives` (visited on 06/01/2020).

[10] *Indian Institute of Remote Sensing*. URL: `https://www.iirs.gov.in/institute-profile` (visited on 06/01/2020).

[11] 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).

[12] 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.

[13] Fei-Fei Li, Justin Johnson, and Serena Yeung. *cs231n - Lecture Slides*. 2017. URL: http://cs231n.stanford.edu/slides/2017/ (visited on 06/01/2020).

[14] 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.

[15] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. In: *Foundations of Machine Learning,* 2012.

[16] Shivaprakash Muruganandham. *Semantic segmentation of satellite images using deep learning*. 2016.

[17] Andrew Ng and Kian Katanforoosh. *cs229 - Lecture Notes*. 2018. URL: http://cs229.stanford.edu/notes/ (visited on 06/01/2020).

[18] Andrew Ng, Kian Katanforoosh, and Younes Bensouda Mourri. *Convolutional Neural Networks*. 2017. (Visited on 06/02/2020).

[19] Savan Patel. *Chapter 2: SVM (Support Vector Machines)*. 2017. (Visited on 06/05/2020).

[20] 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.

[21] 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.

[22] Stuart J. Russell and Peter Norvig. In: *Artificial Intelligence: A Modern Approach, Third Edition*. 2010.

[23] Jürgen Schmidhuber. "Deep learning in neural networks: An overview". In: *Neural networks* 61 (2015), pp. 85–117.

[24] Robert A. Schowengerdt. In: *Remote sensing: models and methods for image processing (3rd ed.)*. 2007.

[25] 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).

[26] 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 Intitute of Remote Sensing
NDVI	Normalised Difference Vegetation Index
NIR	Near InfraRed
NN	Neural Networks.
SVM	Support Vector Machines

Glossary

Chlorophyll Green photosynthetic pigment found in plants
Convolution mathematical way of combining two signals to form a third signal
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