Norwegian University
of Life Sciences

Master's Thesis 2020 30 ECTS
Faculty of Science and Technology

Road Edge Detection using Hyperspectral and LiDAR data based on Machine and Deep Learning

Rabin Senchuri
Data Science

Acknowledgement

First of all, I would like to thank my thesis supervisor Prof. Ingunn Burud of the Faculty of Science and Technology at the University of Life Sciences of Norway. The door to Prof. Burud 's office was always open whenever I had a question about my research. She constantly allowed this paper to be my own work, but she kept me in the right direction anytime she felt I needed it.

This thesis would not have been possible without the support and guidance of my co-supervisor, Agnieszka Kuras, of the Faculty of Science and Technology at the University of Life Sciences of Norway. She has been an influential person who assisted me in my work and in the implementation of the project.I am very grateful for her involvement and the hours she spent reviewing my thesis.

I would also like to thank the experts from Terratec AS who have been involved in the data acquisition and preparation of this research. This thesis would not have been possible without their passionate support and feedback.

I would like to acknowledge Mrs. Rojina Senchuri at Pokhara University as the second reader of this thesis, and I am gratefully indebted to her for her very valuable comments on this thesis.

Finally, I must express my sincere appreciation to my brother for providing me with unfailing guidance and constant motivation during my years of study and through the process of researching and writing this thesis. Without him, this achievement would not have been possible.

Ås, August, 2020
Rabin Senchuri

Abstract

The mapping and classification of urban areas play a crucial role in the planning of the city and monitoring of various natural hazards that could occur in the future. It is also necessary for evaluating large-scale changes that occur in urban areas like building and road construction. Road edge detection is one of the key aspects of urban planning, as it is necessary to monitor street water levels and to monitor floods and landslides. High spatial and spectral resolution remotely sensed images called hyperspectral images are required to classify urban areas as roads and non-roads. In this study, LiDAR data collected from aircraft were used in combination with hyperspectral images to increase the accuracy of the classification. Hyperspectral and LiDAR data were collected from the Sandvika area using a hyperspectral and LiDAR sensor mounted on the aircraft. The data collected were pre-processed and corrected atmospherically. The classification was performed using machine learning and deep learning algorithms. Finally, the road edges were extracted using Canny Edge Detection Algorithm.

For this analysis, the LiDAR-hyperspectral image fusion approach was used and the fusion was pixel-wise based on hyperspectral and LiDAR data features. Hyperspectral features were derived using the Principal Component Analysis and Normalized Difference Vegetation Index, and the LiDAR features were based on the Normalized Digital Surface and Intensity Model. The classification models used for this study were Support Vector Machine, Random Forest and Convolutional Neural Network. Efficiency of each model was evaluated and optimization was performed in order to obtain the best model. Random Forest outperformed both SVM and CNN on the data classification.

Similarly, there were two types of data; the radiance and the atmospherically corrected data. Atmospherically corrected data is called reflectance data. The classification was performed on both the data and the performance of the radiance data (88% f1-score) was higher than the reflectance data. The approach demonstrated here can be widely applied in the classification and monitoring of urban areas experiencing major change.

List of Abbreviations

ADALINE Adaptive Linear Neuron

ANN Artificial Neural Networks

API Application Programming Interface

BRDF Bidirectional Reflectance Distribution Function

BSQ Band-Sequential Format

CNN Convolutional Neural Network

CSV Comma Separated Values

DL Deep Learning

DN Digital Number

DSM Digital Surface Model

DTM Digital Terrain Model

EMR Electromagnetic Radiation

GNSS Global Navigation Satellite System

GPS Global Positioning System

ID3 Iterative Dichotomiser 3

IMU Inertial Measurement Unit

INS Inertial Navigation System

LiDAR Light Detection and Ranging

LUT Look-up Table

MNF Minimum Noise Fraction

MSE Mean Squared Error

nDSM Normalized Digital Surface Model

NDVI Normalized Difference Vegetation Index

NIR Near Infrared

PCA Principal Component Analysis

ReLU Rectified Linear Units

RF Random Forest

ROI Region of Interest

SGD Stochastic Gradient Decent

SVM Support Vector Machine

SWIR Short Wave Infrared

VNIR Visible and Near Infrared

Table of Contents

List of Figures

List of Tables

Introduction

1.1 Background

Urban environments are becoming more complex due to rapid construction activities. New roads, buildings, and various structures are being constructed and forests and vegetation are being turned down. Therefore, to keep track of these and to identify specific features and objects in urban environments, airborne hyperspectral imaging is used [Grätzer, 2007]. The urban area object classification and detection has been a popular subject for research with the advancement of hyperspectral remote sensing technology.

Hyperspectral remote sensors generate hyperspectral 3D images containing both spatial and spectral information by capturing digital images in hundreds of continuous narrow spectral channels spanning from visible to infrared wavelengths. This rich hyperspectral image feature has been used in a variety of applications by geographers, foresters, environmentalists, geologists, and urban planners. Geographer uses hyperspectral images to study the natural environment of the Earth and its relation with human society and foresters to track different kinds of trees and plants in the forest. Likewise, the environmentalist detects landslides and other natural hazards by using hyperspectral imaging technology [Li et al., 2017]. Urban planners use hyperspectral images to map the city's landscape and make decisions on building and other construction work based on hyperspectral images.

The main concern of this study is the extraction of road boundaries from urban areas. Detection of road edges is essential for monitoring landslides and floods. More recently, road extraction using remote sensing technology has been used for traffic management, urban planning, and GPS monitoring. Several studies have been done to develop a method for extracting roads from remotely sensed images. [Wang et al., 2019] applied Hough Transform to the hyperspectral image to isolate the road and then used NDVI to isolate and remove vegetation such as trees and shrubs along the pavement. [Song and Civco, 2004] has specified a two-step road extraction method where in the first step, the image is categorized into two classes, road, and non-road, and in the second step, the categorized road class is segmented using a region-growing technique, and finally, thresholding is used to extract the road centerline. Another approach to road extraction using residual learning and U-Net is explained in [Zhang et al., 2018]. The advantage of using residual units is that it enhances the training of deep neural networks and the skip connection within network helps the information flow and design of a network with few parameters and therefore improves performance.

Although high-resolution hyperspectral data has been important for classification techniques, in some cases, the spatial resolution of hyperspectral data is not sufficient to separate complex classes present in urban environments due to the presence of mixed pixel [Bioucas-Dias et al., 2012]. The presence of mixed pixels complicates the classification process considerably, so that data from other sources, such as LiDAR, is used to boost the classification result. LiDAR data provides information on the elevation of the Earth's surface object and, in this study, multi-wavelength LiDAR data was used to add intensity information for each LiDAR point. Thus, a combination of hyperspectral and LiDAR data has been used in this analysis to extract the road edges [Khodadadzadeh et al., 2015].

In this study, road edges are extracted using a fusion of hyperspectral and LiDAR data. Supervised machine learning algorithms such as Support Vector Machine (SVM) and Random Forest (RF) and Deep Learning models such as Convolutional Neural Network (CNN) are used to classify remotely sensed images. Then, the road edges are extracted from classified image using Canny Edge Detection algorithm. The integration of hyperspectral and LiDAR data is performed using a pixel-wise process, where the hyperspectral derived features are combined with the LiDAR derived features. The normalized digital surface model (nDSM) feature derived from LiDAR data provides useful information in the spatial sense, while the hyperspectral data, abundant in spectral information, provides continuous spectral signatures for each pixel which can be used for classification purposes [Khodadadzadeh et al., 2015]. The motivation of the analysis is based on three comparisons of the results of the classification. First, the outcomes of the Machine Learning and Deep Learning models are compared. Likewise, radiance and reflectance hyperspectral image classification results are contrasted. Finally, the results obtained from the classification of the fused and the individual data are compared.

1.2 Purpose of Study

The primary purpose of this study is to extract road edges from remotely sensed images using both hyperspectral and LiDAR data. Both machine and deep learning algorithms are used for the classification of road and non-road pixels. Some of the analysis carried out in this study are listed below:

- Compare the classification performance of radiance and reflectance data.
- Compare the performance of the machine learning and deep learning models.
- Compare the efficiency of road extraction with hyperspectral, LiDAR and fused datasets.

1.3 Layout and structure

The layout of the thesis is defined as below:

Chapter 1: Introduction

- This chapter explains the previous road extraction research and our method of doing this. It also describes the purpose of the analysis.

Chapter 2: Theory

- The theoretical and working principles of the methods used in this analysis are discussed in detail in this chapter.

Chapter 3: Method

- This chapter explains the process of data acquisition, data preparation, pre-processing, extraction of features, classification and evaluation.

Chapter 4: Result and Discussion

- This chapter explores the results of the experiment and their comparison. The results of the radiance and the reflectance data are compared. Similarly, the output of a classification algorithm such as SVM, RF, and CNN is contrasted. And finally, the hyperspectral, LiDAR, and fused dataset results are analyzed and compared.

Chapter 5: Conclusion

- This chapter summarizes the results of the experiment and evaluates the accuracy of result. It also describes additional research work that can be done to improve the outcome.

Theory

2.1 Remote Sensing

Remote Sensing refers to the method of monitoring and evaluation of information of Earth surface by measuring the reflected or emitted radiation from aircraft, satellite, or in the laboratory without making physical contact with it. Many remote sensing systems have been developed to generate data offering a wide range of spatial, spectral, and temporal features [Schowengerdt, 2007]. In this thesis, Electromagnetic Radiation (EMR) extending from visible to shortwave infrared regions is used for remote sensing of Earth surface materials.

In this study, remote sensing sensors are mounted on the aircraft. The aircraft is operated over the study area and different materials are detected. These detected signals are processed to produce images. The following principle is implemented, as defined in [Rees, 2013]:

- Emission of EMR from a light source, Sun.
- Emitted EMR interact with the atmosphere until it reaches the target.
- Then the incident energy gets reflected, absorbed, scattered, and transmitted from the ground material.
- The reflected and scattered radiation again interacts with the atmosphere on the way back to the sensor.
- The radiation collected by the sensor is then converted to a digital image using imaging spectroscopy for further analysis.

2.1.1 Electromagnetic Radiation

Planck's quantum theory states that all material can absorb and emit electromagnetic radiation only in 'chunks' of energy, quanta E, and that these are proportional to the frequency of that radiation $E = h\vartheta$, where h is Planck's constant ($6.62606957(29) \times 10^{34} Js$), ϑ is the frequency, and E is energy of an electromagnetic wave. According to this theory, absorption and emission have nothing to do with the physical reality of the radiation itself. However, Albert Einstein reinterpreted Planck's quantum hypothesis and used it to explain the photoelectric effect, in which shining light on certain materials can eject electrons from the material. This phenomenon of illuminating the target material with some source of energy is implemented in remote sensing. This energy is

called Electromagnetic Radiation. EMR consists of electric, and magnetic fields and it follows the wave theory. The magnitude of electric field varies in the direction perpendicular to the direction of propagation of radiation, and magnetic field is perpendicular to the direction of electric field as shown in figure 2.1 [Manolakis et al., 2016].

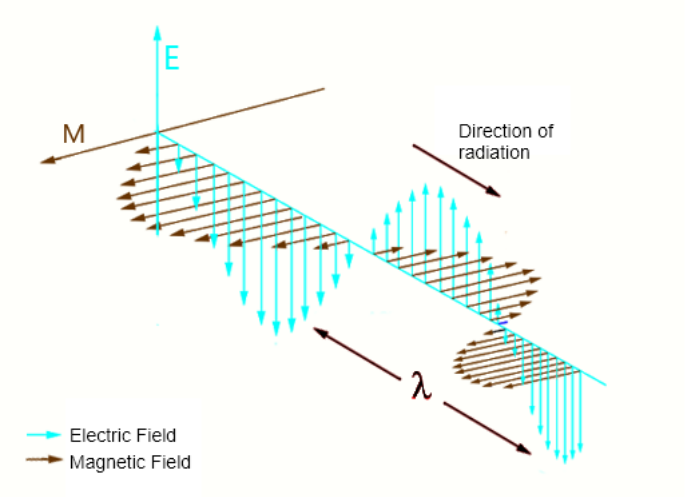


Figure 2.1: The electric and magnetic field that are perpendicular to each other and both oscillating perpendicularly in the direction of propagation. Image modified from [Wang, 1986]

EMR is characterized by its wavelength (λ), and frequency (ϑ) and their relation is defined as:

$$c = \lambda\vartheta \tag{2.1}$$

where c is the velocity of light ($2.99792458 \times 10^8 m/s$). The wavelength is the distance between wave crests as shown in figure 2.1 and frequency is the number of wave cycle passing through a fixed point per unit time [Wang, 1986]. Based on wavelength and frequency, EMR can be categorized from shorter wavelengths to longer wavelengths ranges. This is shown in the figure 2.2.

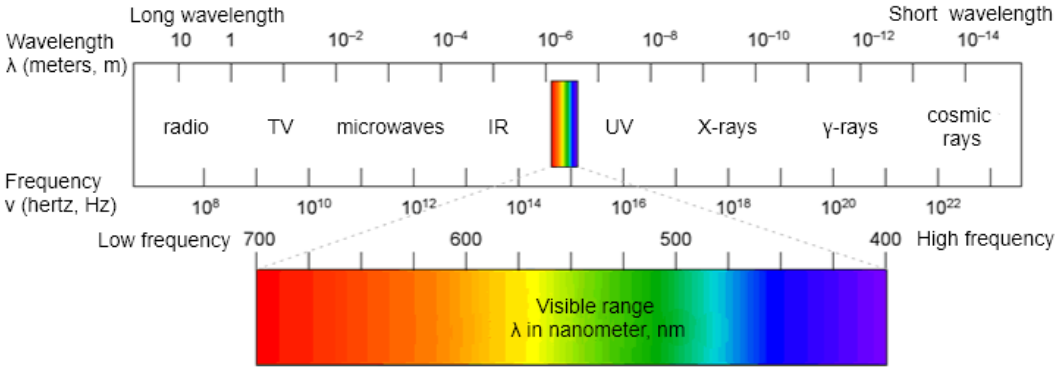


Figure 2.2: Electromagnetic Spectrum. In this study, visible spectrum (400 - 700 nm), near-infrared region(700 - 1000 nm) and shortwave infrared region (1000 - 2500 nm) are used. Image from [Oh et al., 2016]

In Figure 2.2, EMR with the longest wavelength and lowest frequency are on the left side and are related to radio and microwaves. In contrast, EMR with the shortest

wavelength and the highest frequency, related to cosmic and X-rays, are on the right side. Similarly, the visible spectrum occupies the center spectrum (400 - 700 nm). Only, the visible spectrum can be perceived by the human eye when reflected from the target. Other spectrum, that is not visible by human eyes, can be detected by remote sensing devices and used to identify the target [Tro, 2018]. In this study, the visible and near-infrared (VNIR) extending from 400 to 1000 nm wavelength range and shortwave infrared (SWIR) extending from 1000 to 2500 nm wavelength range are used.

2.1.2 Interaction of EMR with atmosphere and surface materials

The EMR traveling towards the target interacts with particles and gases of Earth's atmosphere and undergoes scattering, transmission, and absorption as shown in figure 2.3. This scattering and absorption of EMR depend upon the radiation wavelength and type of atmospheric particles. The EMR is absorbed by various atmospheric molecules such as ozone (O_3), water vapor (H_2O) and carbon dioxide (CO_2) as shown in figure 2.4 [Kerle et al., 2004].

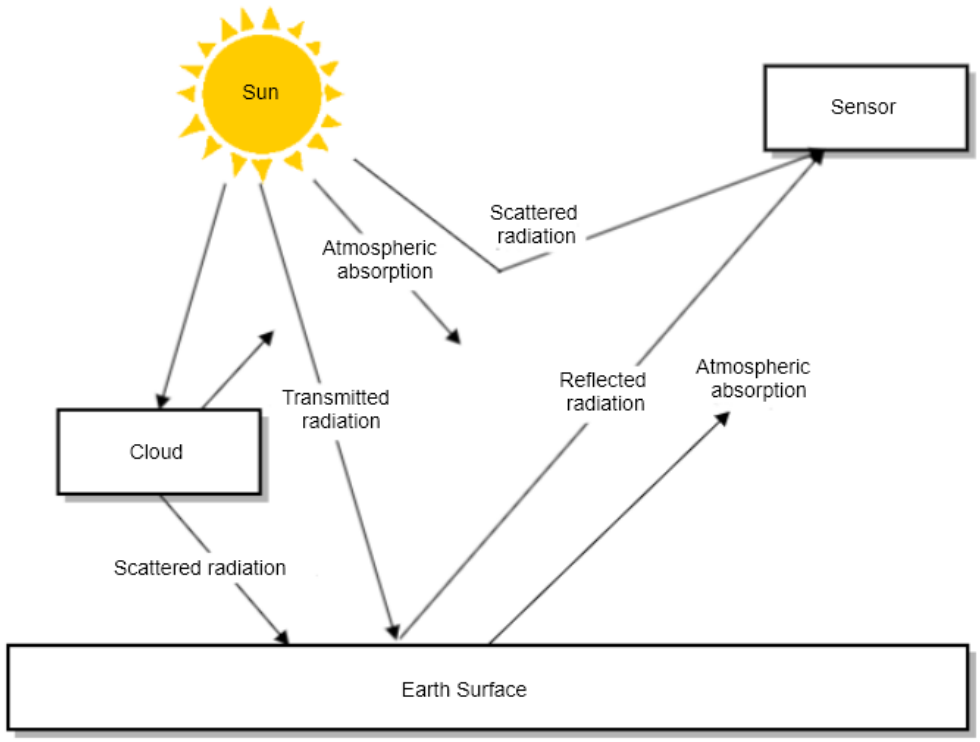


Figure 2.3: EMR interaction in the atmosphere and at the Earth's surface.

The transmitted EMR reaches the surface of the Earth where it interacts with the target and is absorbed, transmitted, and reflected again. Among these, the measurement of the reflected radiation is the area of interest for remote sensing [Joseph, 2005]. The proportion of reflected, absorbed, and transmitted energy depends on the wavelength and type of material.

The leaves absorb the radiation in the red and blue light and reflect the green light so

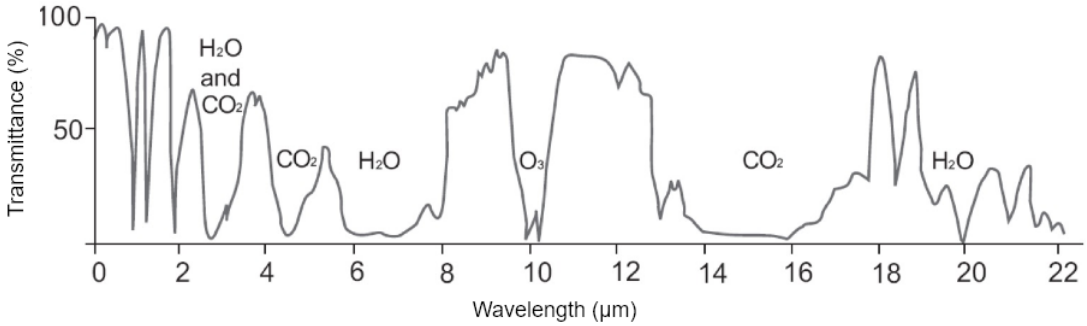


Figure 2.4: EMR absorption by various atmospheric molecules. It can be inferred from the image that many of the wavelengths are not useful for remote sensing of Earth's surface. The most common ranges are between 0.4 and 2 μm, between 3 and 4 μm and between 8 and 14 μm. Image from [Kerle et al., 2004]

that they appear green. A healthy leaf reflects more near-infrared wavelengths. The plant's health can be determined by assessing the reflected near-infrared wavelength. Likewise, water appears blue when it reflects shorter-visible wavelengths and absorbs longer visible and near-infrared wavelengths [Joseph, 2005]. This shows that surface materials are sensitive to different wavelengths of radiation and can, therefore, be used to differentiate different surface materials.

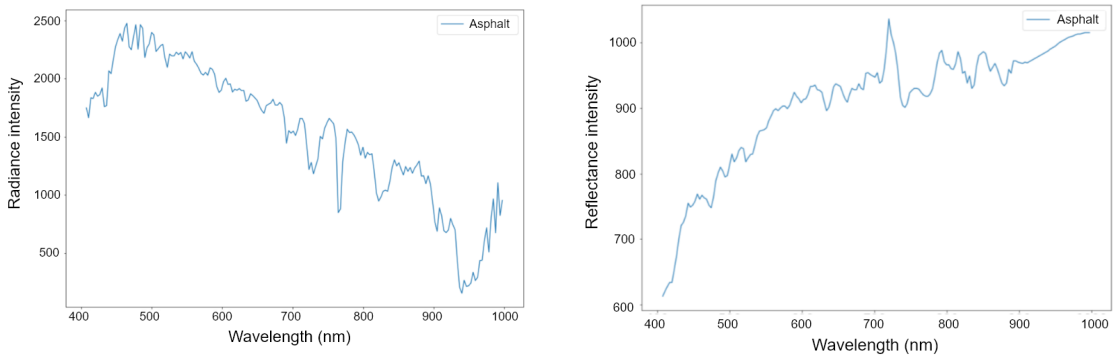
2.1.3 Radiance

The EMR reflected from the target undergoes various atmospheric interactions and is finally detected by the sensor. The sensor records the energy as an array of numbers called Digital Number (DN), representing the brightness of each area [Shippert, 2013]. The size of the area being detected by the sensor depends upon the distance between the target and the sensor. The smallest area that can be detected by a sensor is called a spatial resolution of that sensor [Joseph, 2005].

For any quantitative analysis, the DN values are calibrated into meaningful, quantitative values called radiance. Radiance includes all reflected radiation from the surface material, neighboring pixels, and atmosphere. So, the radiance image has the effects of illumination, atmospheric transmission, sensor characteristic, and it varies in time due to which unique material information cannot be obtained at different takes. Thus, the radiance data has to be converted to reflectance data by performing atmospheric correction [Geospatials, 2013]. The reflectance image obtained after atmospheric correction delivers more repeatable results than radiance image. The comparison of radiance and reflectance spectrum is shown in figure 2.5.

2.1.4 Reflectance

Reflectance is the ratio of the EMR incident on the material surface to the EMR reflected off the material surface [Manolakis, 2005]. Since the radiance data consists of atmospheric effects, estimating the reflectance spectrum from the radiance spectrum is an important step in most hyperspectral image analysis applications. To obtain a reflectance spectrum, an atmospheric correction has to be done [Geospatials, 2013]. The process of atmospheric correction is discussed later in chapter 3.2.2. Figure 2.6 shows the reflectance image and radiance image of the same area.



(a) Spectra of asphalt of radiance image

(b) Spectra of asphalt of reflectance image

Figure 2.5: Asphalt spectra of (a)radiance and (b)reflectance image. The asphalt radiance spectra peak at about 500 nm due to the effects of solar irradiance (solar spectrum peaks at about 500 nm) and atmospheric gases.

Figure 2.6: Left: Radiance Image, Right: Reflectance Image. Both images are of the same place but the radiance and reflectance image looks different. The radiance image is a bit brighter and clear as compared to the reflectance image. Some of the parts of the radiance image are overexposed, whereas the reflectance image is well exposed.

2.2 Hyperspectral Imaging

Hyperspectral imaging is an emerging technique that integrates conventional imaging and optical spectroscopy. Conventional optical imaging obtain images of high spectral and spatial resolution. The first two dimensions of the hyperspectral image cube represent spatial information, and the remaining dimension represents the spectral information. The spatial dimension represents the shape and position of the hyperspectral image, while the spectral dimension represents the number of bands [Vasefi et al., 2016]. These are acquired in such a way that each pixel of the image contains almost a continuous spectral information [Venugopal et al., 2015]. It is different from multispectral images as multispectral images have more than one but less than 20 spectral bands, whereas hyperspectral images have hundreds of spectral bands [Vasefi et al., 2016].

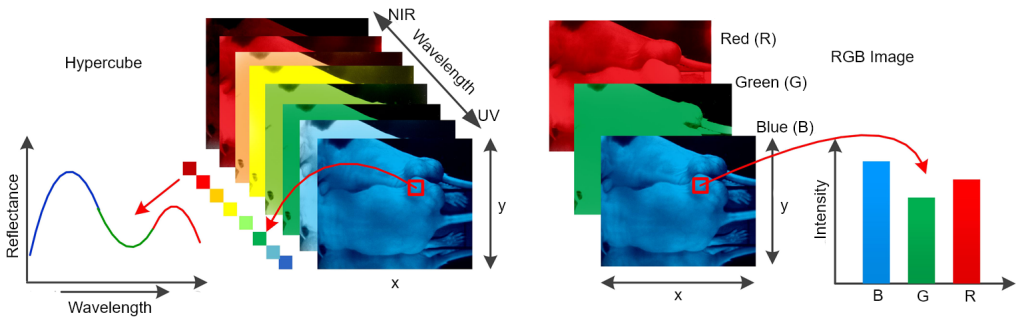


Figure 2.7: Comparison of hyperspectral and multispectral image. Left: Continuous spectra of hyperspectral image pixel. Right: Discrete spectra of multispectral image pixel. Image from [Lu and Fei, 2014].

Hyperspectral image datasets are composed of hundreds of spectral bands in the range that extends beyond the visible range and contains absorption, reflectance, and fluorescence spectrum data for each image pixel. Such spectrum data can be used to determine the particular spectral signature of surface materials so that they can be well separated from each other [Shafri et al., 2012].

Hyperspectral imaging is widely researched field and has many applications in the field of geology, medicine, urban planning, and quality assessment of different materials. In the beginning, hyperspectral imaging was used in the field of remote sensing, where hyperspectral images of distant surface materials were generated and analysis were done to obtain different classification maps. [Feng and Sun, 2012] used hyperspectral techniques like near-infrared hyperspectral imaging, fluorescence hyperspectral imaging, Raman hyperspectral imaging, and their combinations for food safety surveillance. A model was developed for differentiating varieties of commodity maize seeds using hyperspectral imaging in visible and near-infrared region, which is shown in [Zhang et al., 2012]. [Heiden et al., 2012] used hyperspectral imaging for the classification of urban areas based on structure type to assess the ecological situation in the context of urban planning.

2.3 Normalized Difference Vegetation Index

The Normalized Difference Vegetation Index (NDVI) is an approach of evaluating if a land surface contains live green vegetation or not. For this, NDVI uses visible and near-infrared bands of the electromagnetic spectrum. The NDVI algorithm subtracts the red band from near-infrared and is divided by the sum of red and near-infrared bands [Roderick et al., 1996].

$$NDVI = \frac{NIR - RED}{NIR + RED} \tag{2.2}$$

NDVI has wide application in the field of remote sensing. It is used to estimate healthy and unhealthy vegetation index, ground cover proportion, plant photosynthesis activity, and the amount of biomass. NDVI is highly sensitive to vegetation and relatively insensitive to soil and atmosphere [Chen et al., 2004]. Also, early stress in plants can be detected using NDVI and hyperspectral images [Behmann et al., 2014]. Figure 2.8 shows the comparison of the spectral signature of a healthy and an unhealthy plant.

Healthy vegetation absorbs more blue and red light energy to fuel photosynthesis and creates chlorophyll. Plants with more chlorophyll reflect more radiation in the NIR region. [Kokaly, 2001] demonstrated spectroscopic nitrogen concentration estimates from the reflectance spectra of dried plant samples using NDVI. [Valor and Caselles, 1996] related the emissivity to NDVI and used it to measure the emissivity of the vegetation-covered area and thus map the land surface emissivity.

2.4 Hyperspectral Image Cube Unfolding

A hyperspectral cube is a three-dimensional image cube. It has two spatial dimensions and one spectral dimension. Before applying any statistical modeling to a hyperspectral

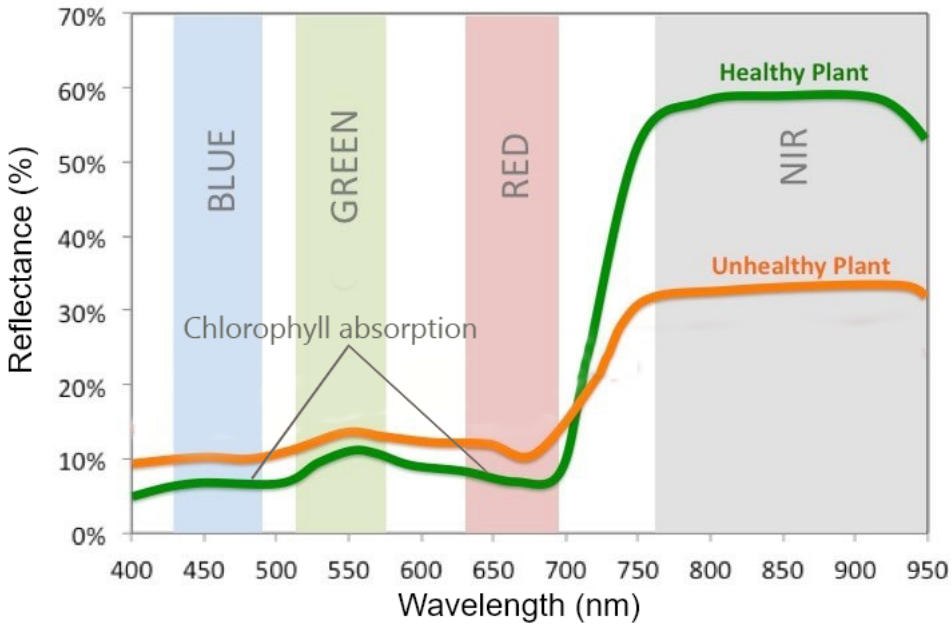


Figure 2.8: Comparison of the spectral signature of a healthy and unhealthful plant. The healthy plant spectral signature shown in the green line plot has more reflection in the NIR region than the unhealthy plant spectral signature shown in the orange line plot. Modified figure from [Shilo, 2018].

image, it must be converted into two-dimensional data consisting of spatial dimensions along rows and spectral band values along columns. This is done by unfolding the hypercube from three dimensions to two dimensions. To unfold a hypercube, each pixel spectrum is stacked, one at the top of the other to create a two-dimensional matrix [Gowen, 2014]. The hyperspectral image unfolding process is shown in Figure 2.9.

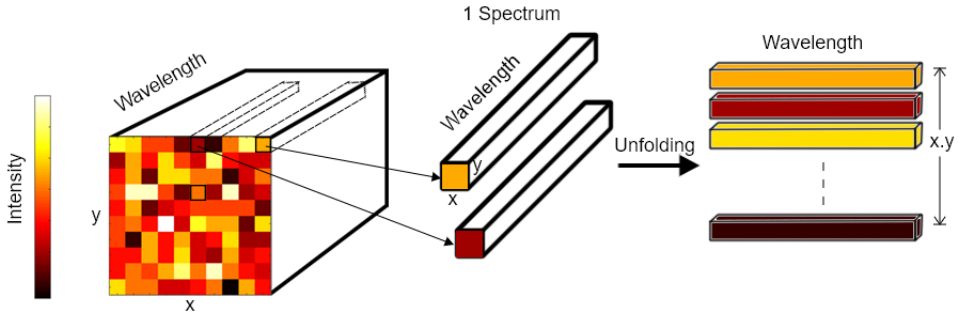


Figure 2.9: Hyperspectral image cube unfolding. Each pixel spectrum is represented by different colors in three dimensional image cube shown in left. To unfold, each color pixel spectra are stacked on top of one another to form a two dimensional matrix.

2.5 Dimensionality reduction Methods

One of the challenging aspects of hyperspectral imaging is the storage and analysis of high volume datasets. Data reduction and compression are necessary for the effective handling of hyperspectral image data. Here compression mainly refers to the processing of spatial domain, and the dimensionality reduction mainly refers to process of reducing the number of spectral bands. At the same time, an important task in dimensionality

reduction is to reduce the redundancy in spectra and spatial information without loosing the valuable details and maximizing the variance of the data [Burger and Gowen, 2011]. One of the most commonly used techniques for dimension reduction is Principal Components Analysis (PCA).

2.5.1 Principal Component Analysis

Principal Component Analysis (PCA) is a linear transformation that is used for dimensional reduction, data compression, feature extraction, data visualization, and various multivariate data analysis [Wold et al., 1987]. PCA focuses mainly on concentrating the spectral variance contained in broad bands of the hyperspectral image to a lower number of principal components. Principal components are the linear combination of original bands. The main goal of the PCA is to re-project data in the direction of high variance and thus, after the PCA, valuable information is obtained in the first few components and other information is discarded as noise. [Licciardi and Chanussot, 2018]. The two key terms used while evaluating PCA are eigenvalues and eigenvectors. The eigenvalues refer to the variance among the data and eigenvectors refer to the corresponding principal component. Principal component with high eigenvalues has high image information and low noise. The scree plot of eigenvalues is analysed to identify the components that have a high variance.

Algebraically, the implementation of PCA can be shown with a dataset matrix X of size $n \times p$, where n is the number of rows and p is the number of columns of matrix. x_1, x_2,...., x_p are variables in each row. Here, the main goal is to find the linear combination of the columns of matrix X with a maximum variance which is given by $\sum_{i=1}^{p} a_i x_i = Xa$, where a is the constant vector [Jolliffe and Cadima, 2016]. The variance of such linear combination is given by equation 2.3 [Jolliffe and Cadima, 2016].

$$var(Xa) = a^T Sa \tag{2.3}$$

where S is defined as the sample covariance matrix of the dataset. The sample covariance matrix is computed by first computing sample mean of data variables and reprojecting them into mean-deviation form. Sample mean (M) of the data is computed by $\frac{1}{p}(x_1 + x_2 + ... + x_p)$ [Lay, 2016]. The sample mean is the center of the scatter plot 2.10a. When the sample mean is subtracted form the data in scatter plot 2.10a, the resulting data is in mean deviation form as shown in plot 2.10b [Lay, 2016]. The new data (B) in mean deviation form is evaluated as;

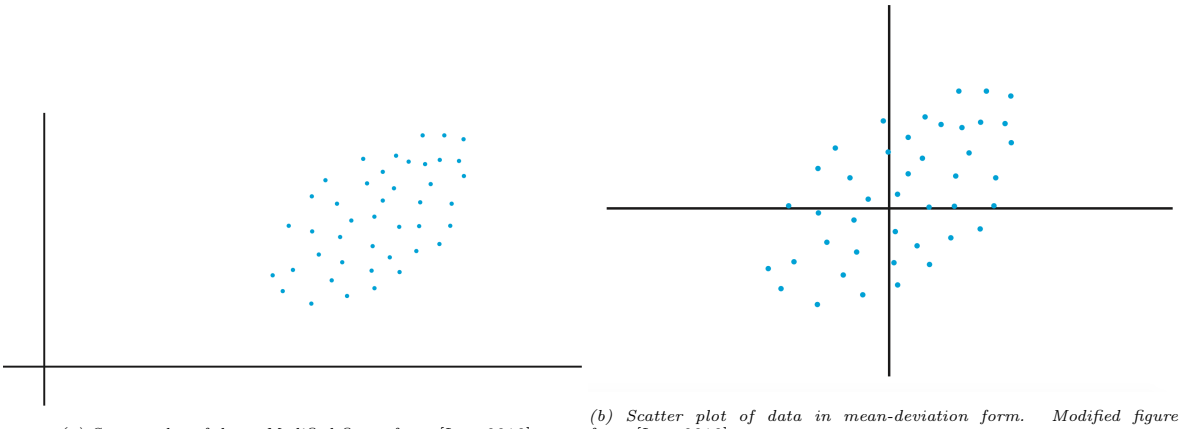
$$y_k = x_k - M, \quad where\ k = 1, 2, ..., p$$

$$B = [y_1\ y_2\ ...\ y_p]$$

The sample covariance matrix (S) is computed by

$$S = \frac{1}{p-1} BB^T$$

where B^T is the transpose of matrix B. In PCA, the main focus is to maximize the variance which is equivalent to obtaining maximum value of quadratic form $a^T Sa$ in equation 2.3. This is achieved by evaluating $a^T Sa - \lambda(a^T a - 1)$, where λ is the eigenvalue.



(a) Scatter plot of data. Modified figure from [Lay, 2016]

(b) Scatter plot of data in mean-deviation form. Modified figure from [Lay, 2016]

For unit vector $a^T a = 1$ and differentiating $a^T Sa - \lambda(a^T a - 1)$ with respect to a, equation 2.4 is obtained [Jolliffe and Cadima, 2016].

$$Sa = \lambda a \tag{2.4}$$

where a is the unit eigenvector, and λ is the corresponding eigenvalue of the covariance matrix S. The first principal component is the eigenvector corresponding to the largest eigenvalue of S, the second principal component is the eigenvector corresponding to the second largest eigenvalue [Lay, 2016].

In order to apply PCA on the hypercube, it has to be first unfolded to two dimensional matrix. Then PCA can be applied on the unfolded hypercube to obtain eigenvectors and eigenvalues [A et al., 2017].

In figure 2.11, PCA is applied to the unfolded hyperspectral image to obtain a number of principal components. Loadings and scores vectors are then obtained from principal components [Pisapia et al., 2018].

PCA is applied by using singular value decomposition(SVD) on the data matrix [Burger and Gowen, 2011]. It is represented as:

$$X = USV^T \tag{2.5}$$

However, for huge highly correlated datasets like hyperspectral image, this is computationally inefficient. So, transformation form of this is used which is represented as,

$$X^T = VSV^T \tag{2.6}$$

The equation above is used to find the loading vectors V. Once the loading vectors V is evaluated, score vectors T can be computed as;

$$XV = USV^T V = US = T \tag{2.7}$$

2.6 Spectral Signature

Although airborne hyperspectral images have a high spectral resolution, they have a low spatial resolution and, with low spatial resolution, it is difficult to distinguish very small materials in the image. In order to solve this, spectral measurements are done

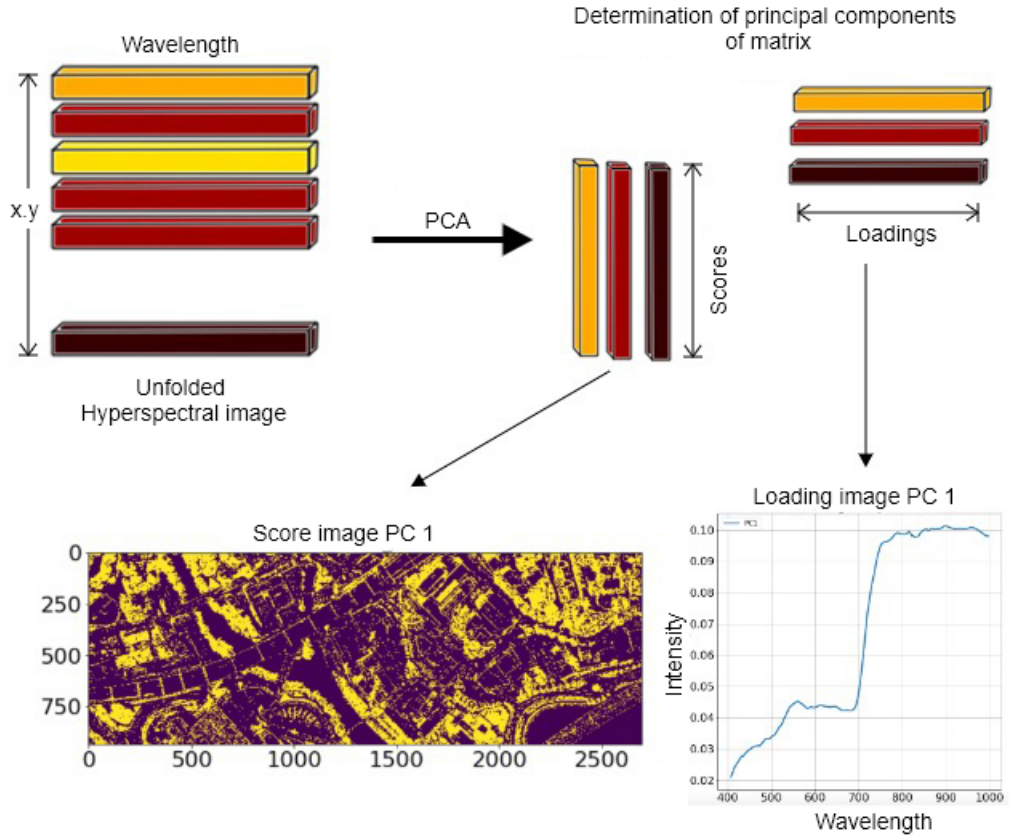


Figure 2.11: PCA on unfolded hyperspectral image (Image taken from study area)

to generate spectral signatures of various land covering materials such as vegetation, asphalt, roofs, soil, rock, and more [Schowengerdt, 2007]. The spectral signature of hyperspectral imaging is a graphical plot of reflectance/radiance and wavelength/wave number. The motivation in this is that different types of material can be distinguished based on their physical properties and chemical composition. That means every material has a characteristic property of absorption and reflectance.As the hyperspectral image has a wide spectral range, it is therefore possible to produce distinct spectral signatures for different surface materials [Charles et al., 2010].

In theory, each material has a unique spectral signature that can be used to distinguish it from the others. In practice, however, material spectra are influenced by different factors, such as natural variation of materials, changes in atmospheric conditions and water absorption. This means that even the same material can exhibit different spectral properties in a different environment.

2.7 LiDAR Data

Light Detection and Ranging (LiDAR) is an optical remote sensing technique that uses laser light to sample earth surface material to produce highly accurate three dimensional measurements [ArcGis, 2018]. LiDAR is a widely used technology for topographical land mapping as it helps to identify land surface elevation. The tools used for creating LiDAR data are LiDAR sensor, laser scanner system, Global Positioning System (GPS),

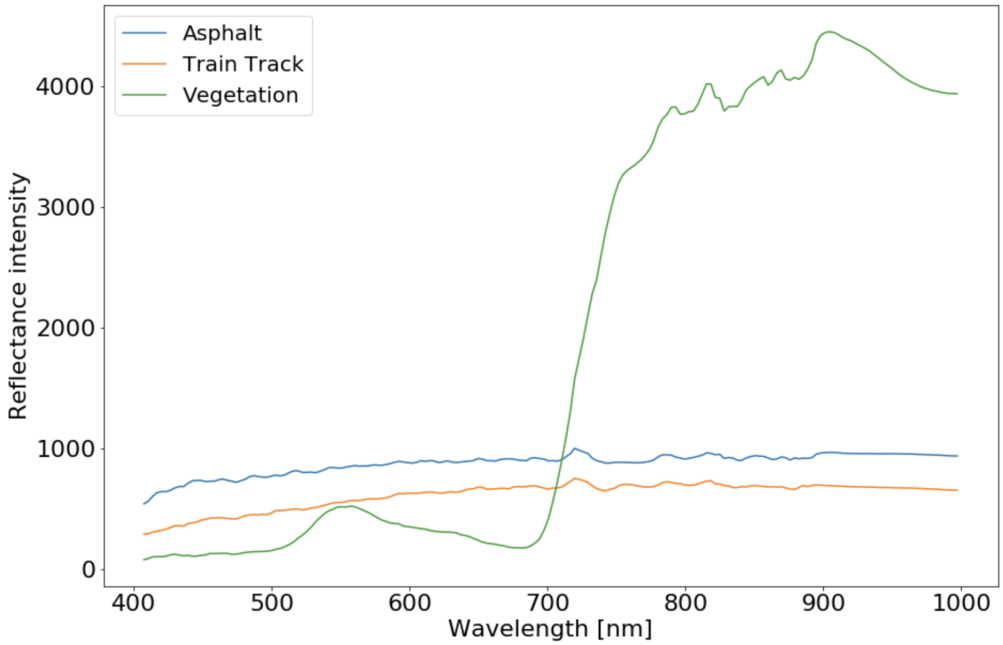


Figure 2.12: Spectral Signature of asphalt, train track, and vegetation. The spectral signature of vegetation is very unique as compared to asphalt and train track.

and Inertial Navigation System (INS). The hyperspectral sensor is a passive sensor as it is dependent on Sun or other light sources for creating images. In contrast, the LiDAR sensor is an active sensor that transmits a laser beam towards the target for performing measurements. The working of LiDAR is shown in figure 2.13. The LiDAR scanner mounted on the aircraft, fires thousands of pulses per second towards the surface. The pulse gets reflected from the target and is detected by the LiDAR sensor. The time at which the laser beam was emitted and detected by the sensor after reflection is recorded and the distance between the LiDAR sensor and the target is evaluated as [Davis, 2012]:

$$Distance = \frac{Speed\ of\ Light * Recorded\ Time}{2} \tag{2.8}$$

The data collected is therefore combined with the location information obtained from the GPS and the INS to generate the georeferenced three-dimensional coordinates (x, y, and z) of the target. The x and y coordinates are the location, and the z coordinate is the elevation details of the target. The GPS provides the precise location of the LiDAR sensor, while the INS provides the precise orientation of the laser scanner. In this process, several points from various materials are collected. Aftermath, these readings are analyzed using a number of methods to obtain highly precise three dimensional georeferenced coordinates points [Davis, 2012]. These points are referred to as LiDAR point clouds.

LiDAR data have extensive applications in remote sensing. [Andersen et al., 2005] used LiDAR point clouds to estimate three-dimensional forest structure over extensive areas and used regression analysis to build a model for estimating forest canopy fuel parameters using LiDAR data. Non-ground materials such as buildings and vehicles are removed from the image to obtain a digital terrain model(DTM) that can be used for flood modeling and landslide prediction [Zhang et al., 2003].

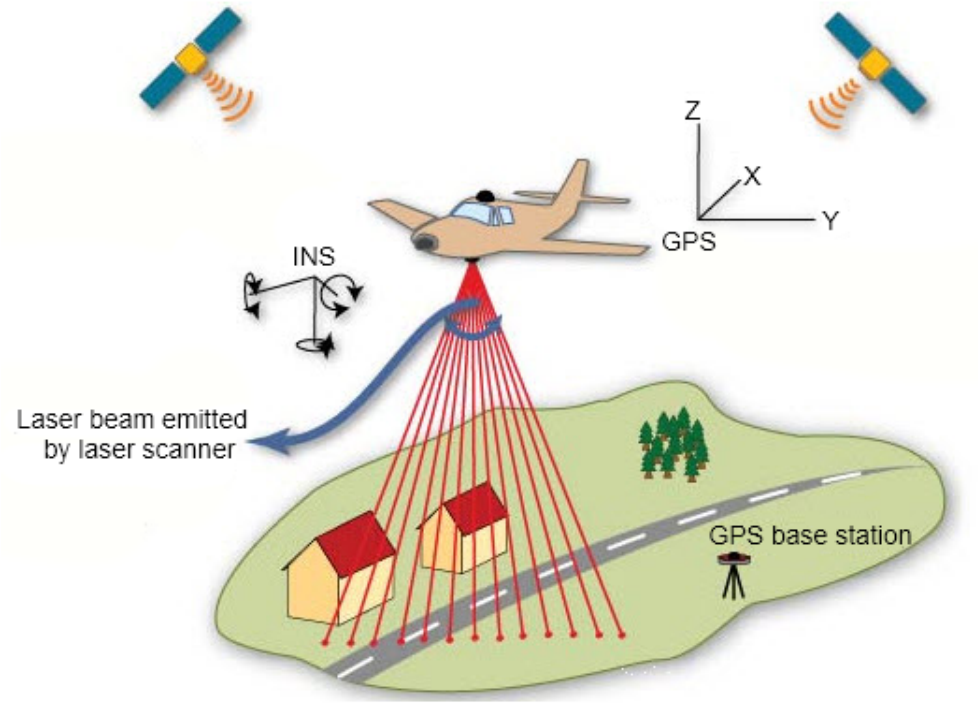


Figure 2.13: Working of LiDAR. The travel time of laser beam to and from the target material is recorded which in combination with positional information obtained from GPS and INS is used to evaluate georeferenced x,y,z coordinates of that material. Modified figure from [Fruchart, 2011]

2.7.1 LiDAR Returns

When the laser beams are fired from the LiDAR system, they fall over various surface material. Moreover, during their travel towards the ground surface, they get reflected multiple times [ArcGis, 2018]. LiDAR sensor receives one or many returns of a single laser beam. The names of those returns are first, second, third, fourth, and many more. The first return is the most significant one so it is used to directly compare hyperspectral and LiDAR data. It has information of surface material like treetops, building roofs, and many more [Andersen et al., 2005].

The intermediate returns include information of other elevation materials and vegetation, while the last return has information about the bare surface of Earth. The first return may have information of Earth's ground surface if there is only one return of a particular laser beam. Likewise, the last return does not always have information of the ground material. It also has information of other materials when the laser beam gets reflected from that material and is unable to pass through it to reach the ground surface [Davis, 2012]. For instance, some buildings have a structure that cannot be penetrated by the laser beam. In such a case, the last return of the beam is the roof of the building, rather than the ground surface. Figure 2.14 shows multiple LiDAR returns from the tree and their corresponding intensities. We can see that first return is related to top of the tree while intermediate return corresponds to different parts of trees, and finally, the last return corresponds to the ground surface.

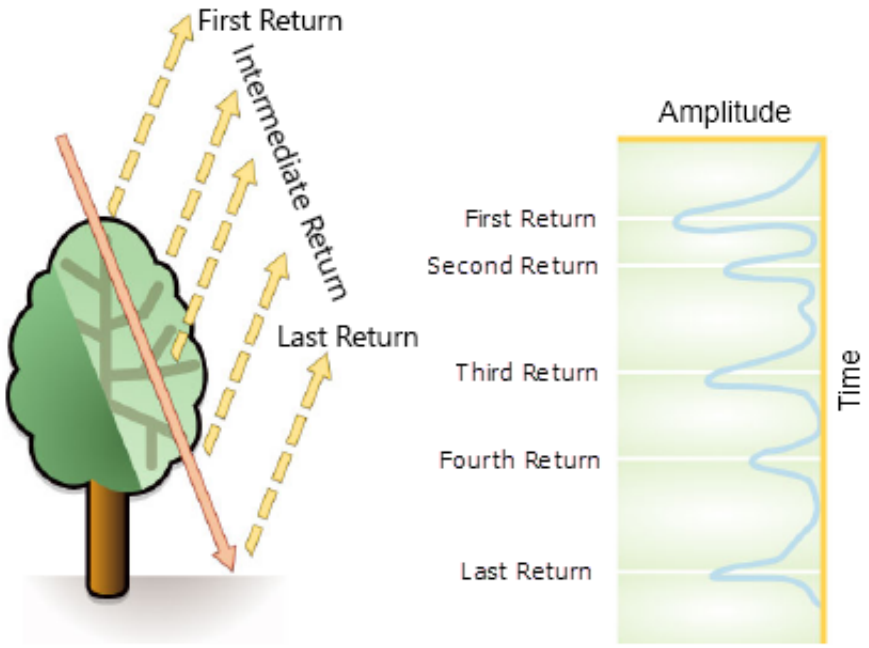


Figure 2.14: Left: LiDAR Return and Right: LiDAR return intensities. Modified figure from [ArcGis, 2018]

2.7.2 Digital Surface Model

Digital Surface Model (DSM) represents all the natural and built features on the Earth's surface [Khosravipour et al., 2015]. DSM obtained from LiDAR data has many applications in forest analysis like tree height measurement, monitoring forest regeneration, biomass, and wildfire risk management [Morsdorf et al., 2004]. For all these applications, the first step is to generate the DSM from respective LiDAR point clouds. DSM is generated by using high elevation value that is measured by using first LiDAR return [Khosravipour et al., 2016].

2.7.3 Digital Elevation Model

Digital Elevation Model (DEM) represents the bare surface of Earth. When the non-ground points such as built (power lines, buildings, and towers) and natural (trees and other types of vegetation) are filtered out of the DSM, a smooth DEM is obtained. In order to produce DEM from LiDAR data, LiDAR points must first be classified as ground and non-ground (natural and built features) points [Irwan Hariyono and Windiastuti, 2018]. Then non-ground points are filtered out to obtain a smooth DEM. DEM has applications in the field of floor modelling and landslide prediction.

2.7.4 Normalized Digital Surface Model

The Normalized Digital Surface Model (nDSM) represents the distance between the ground and the top of the target material. In other words, it calculates the true height of topographical features on the Earth's surface. The nDSM is evaluated by subtraction

of DEM from the DSM [Geography, 2013].

$$nDSM = DSM - DEM \tag{2.9}$$

2.7.5 LiDAR Intensity Model

Intensity is the ratio of reflected light to the emitted light. A LiDAR intensity model is the measure of the strength of the reflecting laser beam that generates the LiDAR point cloud. This intensity model relies on the material's reflective properties, which vary for different materials. Thus, lidar intensity values can be used for object detection and as classification feature [Song et al., 2002].

2.8 Machine Learning Algorithm

Machine learning is a technology which enables computers to learn from experience automatically without being programmed explicitly over time. The goal is to create computer programs that can access data and learn on their own [Pyle and San José, 2015]. Machine learning is computer-driven programming, which means that it is based on algorithms that can learn from computers without human intervention [Mitchell et al., 1997]. The basic working of the machine learning algorithm is shown in figure 2.15. There are several machine learning algorithms that categorized as a supervised and unsupervised as shown in table 2.1.

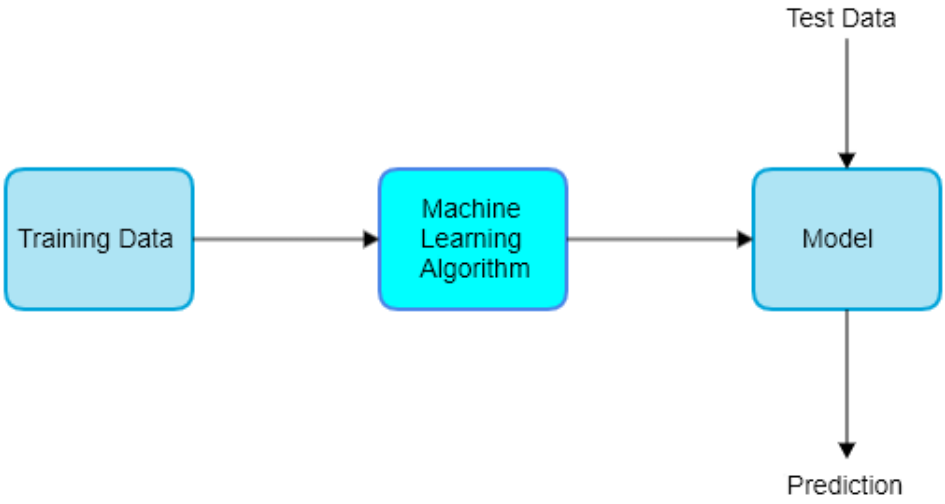


Figure 2.15: Flowchart of machine learning algorithm implementation.

Supervised machine learning algorithms are based on the use of past data learning to predict future events. Labeled data are fed into a supervised machine learning algorithm that learns the pattern from those data and thus generates data models for predicting other unknown data. Labeled data are data with target values that the machine learning algorithm is aiming to predict. Generating labeled data is one of the most challenging tasks in any machine learning and deep learning project [Cloudfactory, 2013]. For this study, the data is labeled pixel-wise in which each pixel of the hyperspectral image is labeled as road, house, forest, train track, or water.

There are certain situations in which labeled training datasets are not available to train machine learning algorithms. For such instances, unsupervised machine learning algorithms are used. An unsupervised machine learning algorithm explores unlabeled datasets to identify the hidden pattern within it and then classifies it on the basis of the pattern.

Semi-supervised machine learning algorithms are a combination of supervised and unsupervised algorithms. It means that both labeled and unlabeled datasets are used for learning process. A semi-supervised machine learning algorithm is used to increase the efficiency of supervised learning using unlabeled datasets when labeled datasets are too limited or too expensive [Zhu and Goldberg, 2009].

Table 2.1: Unsupervised and supervised machine learning algorithms.

Unsupervised	Supervised
	Random Forest
PCA	Decision Trees
K-means	Support Vector Machine
	Logistic Regression

All of these machine learning algorithms consist of representation, evaluation, and optimization components. The representation component consists of a range of classifiers. The evaluation component includes a set of core functions, and optimization involves several parameter optimization techniques to find the most effective classifiers. Table 2.2 shows different examples of three components of the machine learning algorithm [Domingos, 2012].

Table 2.2: Three components of machine learning algorithm [Domingos, 2012]

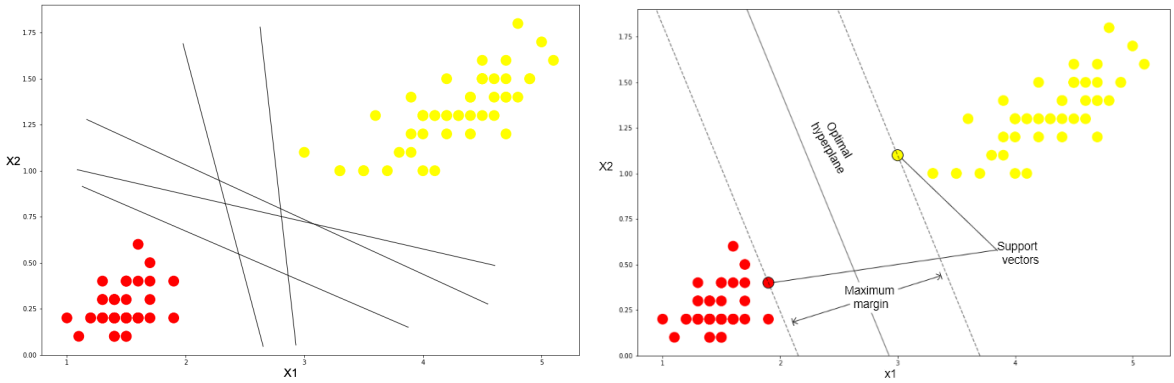
Representation	Evaluation	Optimization
Instances		Combinational Optimization
K-nearest neighbour	Accuracy score	Greedy search
Support Vector Machine	Precision and recall	Beam search
Hyperplanes	Squared error	Branch-and-bound
Naive Bayes	Likelihood	Continuous Optimization
Logistic Regression	Cost	Gradient Descent
Decision Trees	F1-Score	Linear Programming
Random Forest	Cohen-Kappa Score	Quadratic Programming
Neural Network		

Although machine learning algorithms are used in many applications, there is some limitations. One of the main problems is overfitting while running a machine learning algorithm. Overfitting is the condition in which the machine learning algorithm generalizes well with the training data but does not generalize well with the validation data.

2.8.1 Support Vector Machine (SVM)

SVM is a supervised machine learning algorithm which finds the optimal hyperplane separating the data points in multidimensional space [Gandhi, 2018]. The hyperplane

that separates the classes is called the decision boundary. Initially SVM was configured for binary classification [Hsu and Lin, 2002]. Multi-class classification with SVM is possible by creating and integrating multiple binary classifiers [Hsu and Lin, 2002]. Figure 2.16 shows all possible hyperplanes that separate the two classes of data. The main objective of SVM is to maximize the perpendicular distance between the hyperplane and the nearest data points of each class. These data points closest to the optimal hyperplane are called Support Vectors(SV) and the perpendicular distance is called Margin [Gandhi, 2018].



(a) All possible hyperplanes that separate two classes of data

(b) Optimal hyperplane that separates the two classes of data

Figure 2.16: a) All possible hyperplanes that separate two classes of data points and b) Optimal hyperplane that separates the two classes in such a way that the perpendicular distance between the decision boundary and the closest data points of each class is maximum.

The number of dimensions of the hyperplane depends on the number of features in the data. Hyperplane is just a line if the number of features is two and is a two or more dimensional plane if the number of features is three or more [Gandhi, 2018].

In any training of the machine learning algorithm, a good choice of parameters plays an important role. Some of the crucial parameter for adjusting and calibrating SVM algorithms are kernel, regularization, and gamma.

Kernel

SVM works by transforming data into multidimensional space to make it more separable. The task of mapping data from the original input feature space to a multidimensional space is performed with the help of the kernel. There are different types of the kernel, and the choice depends on the nature of the dataset. Some of the most commonly used kernels are linear, polynomial, RBF, and sigmoid [Chang and Lin, 2011].

Regularization

The regularization (C) parameter prevents the misclassification of the training data. The margin of the hyperplane will be smaller for a large regularization value which results in a better classification of the training samples. However, with a small regularization value, the margins are larger, resulting in higher amount of misclassification [Patel, 2018] as shown in Figure 2.17. The cost of using high regularization is expensive.

(a) Low regularization value *(b) High regularization value*

Figure 2.17: a)Low regularization value corresponds to large margin and there are misclassification of data points. b) High regularization value corresponds to small margin which results in better classification. Figure from [Patel, 2018]

2.8.2 Decision Tree Algorithm

Decision Tree is a commonly used supervised machine learning algorithm that divides a dataset based on a particular condition. The decision tree consists of the root, the decision and the leaf node, where the attributes in the root and the decision node ask questions and leaves are the answer to the question and the decision rules are based on if-else statements [Chauhan, 2019].

The decision tree classifies the data by sorting it down from root to leaf node, as shown in Figure 2.18. It is constructed using Iterative Dichotomiser 3 (ID3) algorithm. ID3 algorithm constructs decision trees using a top-down, greedy (select best feature), and iterative approach that follows the following steps [Sakkaf, 2019]:

- Select best attribute A as the NODE
- descendants are created for each value of NODE A
- Sort the descendants of the nodes
- STOP if perfectly classified else iterate over the new leaf nodes

2.8.3 Random Forest Algorithm

Random Forest (RF) is an ensemble learning approach in which many classifier results are aggregated to produce a final output that has a good generalization error and is less vulnerable to overfitting [Liaw et al., 2002]. It is a collection of decision trees in which the outcomes of each decision tree is merged as one final result. In other words, the decision trees are the building blocks of the RF. Decision trees have the downside of being vulnerable to overfitting when the tree is very large. Yet this overfitting problem is mitigated by the use of the RF algorithm [Liaw et al., 2002].

The training of RF algorithm is based on a bagging method where the combination of learning methods increases output. The decision trees are generated using a random collection of variables and random samples from the training dataset. The prediction of each decision tree is estimated, and the best prediction is evaluated based on the vote [Mutanga et al., 2012]. Figure 2.19 shows how the results of the decision tree are combined to form the final output of the random forest.

In random forest, the only parameter to choose is the number of trees. The performance of the random forest is better with a large number of trees at the expense of increased computational costs [Mutanga et al., 2012].

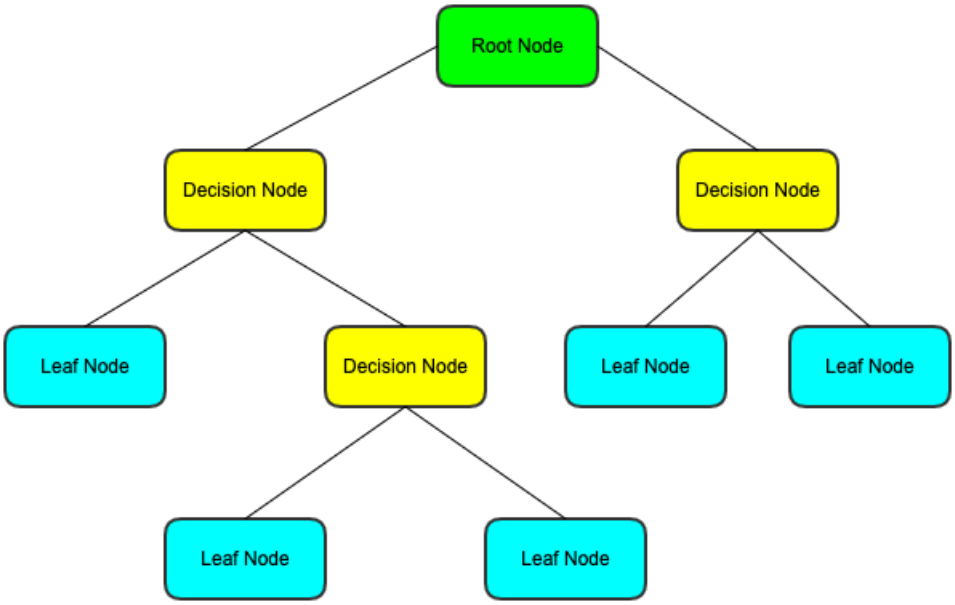


Figure 2.18: Decision Tree. Root Node consists of all data and can be divided into two or more sub-nodes (decision or leaf node). Decision node is a sub-node which can be further divided into sub-nodes. Leaf node is a node that cannot be further separated and includes prediction.

2.9 Artificial Neural Networks (ANN)

The fundamental idea behind Artificial Neural Network's research is the theory and paradigm of how human brains function to solve complex problems. Warren McCulloch and Walter Pitt researched and explained how the neural network operated back in the 1940 [McCulloch and Pitts, 1943]. But, it was implemented after a decade. Researchers and machine learning enthusiasts discontinued researching the neural network because they were unable to find a way to train a multi-layer neural network. In 1986, D.E. Rumelhart, G.E. Hinton, and R.J. Williams were involved in the rediscovery and popularization of the back-propagation algorithm to train a neural network more effectively [Raschka and Mirjalili, 2019]. This, in turn, revoked the interest in more research in neural networks.

ANNs are composed of several layers of single layered neural networks. Adaptive Linear Neuron(Adaline) is a single layered neural network whose key feature is shown in figure 2.20. This algorithm is used to perform a binary classification on a gradient descent method. Gradient descent algorithm is used to update the weight of each epoch using the following rule [Raschka and Mirjalili, 2019]:

$$w := w + \delta w, \; where \; w \; is \; the \; weight \; of \; the \; layer \; and \; \delta w = \eta \nabla J(w) \tag{2.10}$$

Thus, from figure 2.20, the linear combination of the weights connecting the input to the output is the net input(z) which is given by equation 2.11.

$$z = \sum_j w_j x_j = w^T x \tag{2.11}$$

Multi-layer neural network are those composed by connecting multiple single neural network. Figure 2.21 shows a three layer multi-layer neural network. Here all the layers

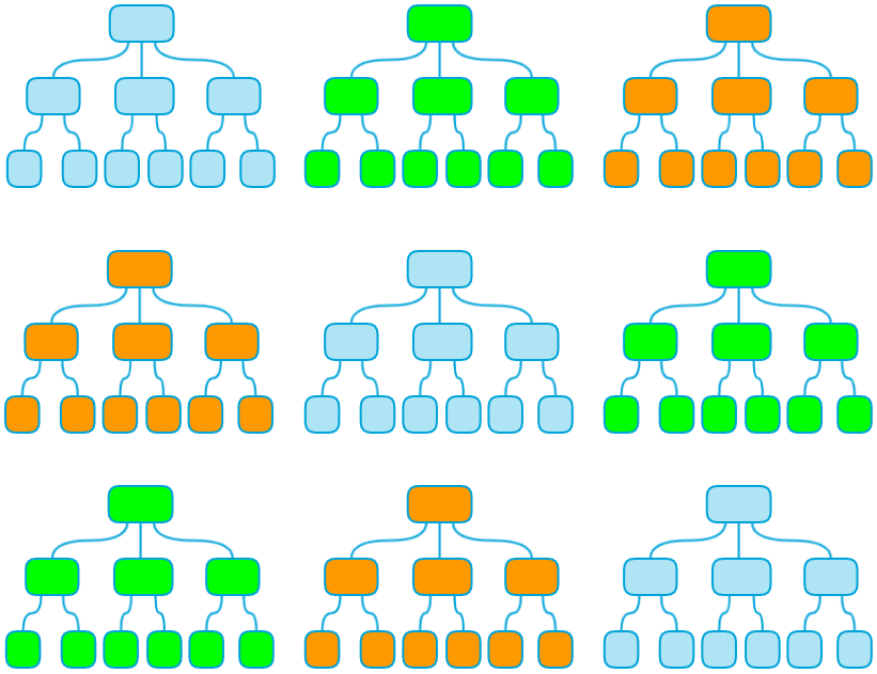


Figure 2.19: Random Forest composed of 9 decision trees. Each decision tree predicts either 0 or 1. Six of the decision tree predicts the result as 1 whereas three decision predicts 0. The final prediction of the random forest is formed by the majority voting so the output prediction is 1.

are fully connected to each other. And if a multi-layer neural network has more than one hidden layer then it is deep neural network.

The multi-layer neural network consists of two parts: forward and backward propagation. In forward propagation, data are propagated from input to output layer leaning the features in input data. Then the output is calculated and compared with the known value to evaluate the error. The minimization of this error is done using back-propagation [Ho et al., 1992]. In back-propagation, the weight adjustment is done by finding derivative for each weight in the network and thus the model is updated. This procedure is repeated for multiple epochs to get the best prediction.

Neural network hyperparameters are the number of layers and the number of neurons. The hyperparameter values are adjusted by cross-validation technique. Deep neural networks are good at processing data but they suffer vanishing gradient problem which will be discussed in later section [Raschka and Mirjalili, 2019].

2.9.1 Convolutional Neural Network (CNN)

Convolutional Neural Network is a deep learning algorithm that takes the image as input, assigns weight and bias to different objects in the image, and thus differentiates those objects from each other. CNN has an architecture similar to that of neuron integration in the visual cortex of the human brain [Raschka and Mirjalili, 2019]. Images are well classified using CNN so that it has gained popularity in the field of computer vision. In the multi-layer perceptron, the vector features are extracted from the image and are fully connected to the hidden layer. The spatial information of the image is not used. However, on CNN, the input layer is connected to the feature map using the receptive fields. Receptive fields are overlapping windows that are passed from

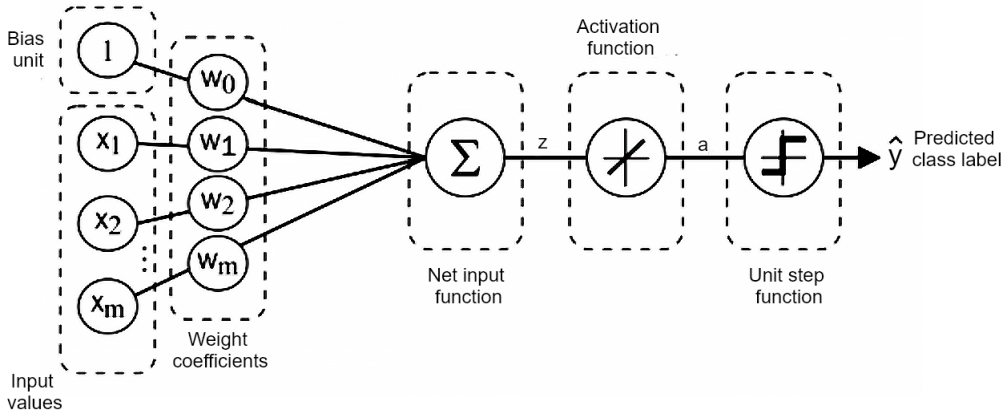


Figure 2.20: Working of adaptive linear neuron algorithm. Figure from [Raschka and Mirjalili, 2019]

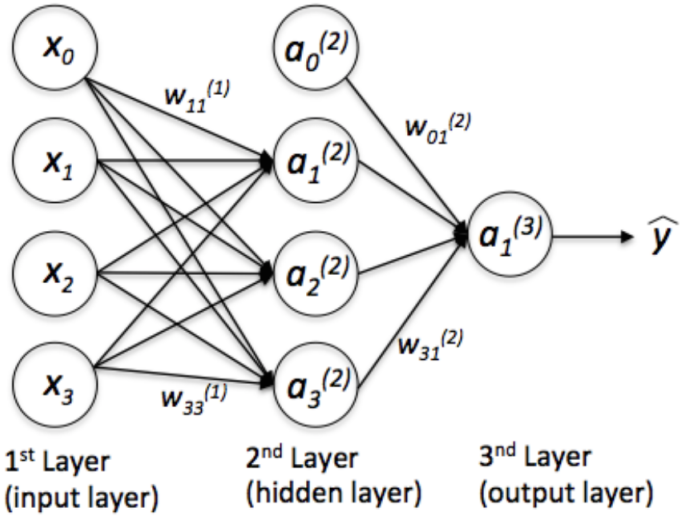


Figure 2.21: Multi-layer neural network with 3 layers of neurons which are input, hidden, and output layer. Figure from [Raschka and Mirjalili, 2019]

pixels to pixels of an input image to create a feature map, and this process is called convolution [Raschka and Mirjalili, 2019].

Unlike regular neural networks, CNN layers arrange neurons in three dimensions: width, height, and depth. For instance, the shape of each image in this study is $931 \times 2400 \times 186$ (width, height, depth). Here, the depth refers to the image channels. CNN consists of three main layers; the convolutional layer, the pooling layer, and the fully connected layer [Saha, 2018]. These three main layers are stacked to form the CNN, as shown in figure 2.22.

Convolutional layer

Convolutional layer is the first layer of CNN, which extracts the features of the input image. In this layer, the filters are convoluted over the pixels of the image. The region of the image where the filter is convoluted is called the receptive field [Saha, 2018]. The depth of the filter must be the same as the depth of the input image. While the filter is convoluted over the image, the values in the filter are multiplied by the original pixel values of the image, and the multiplication is summarized as shown in equation

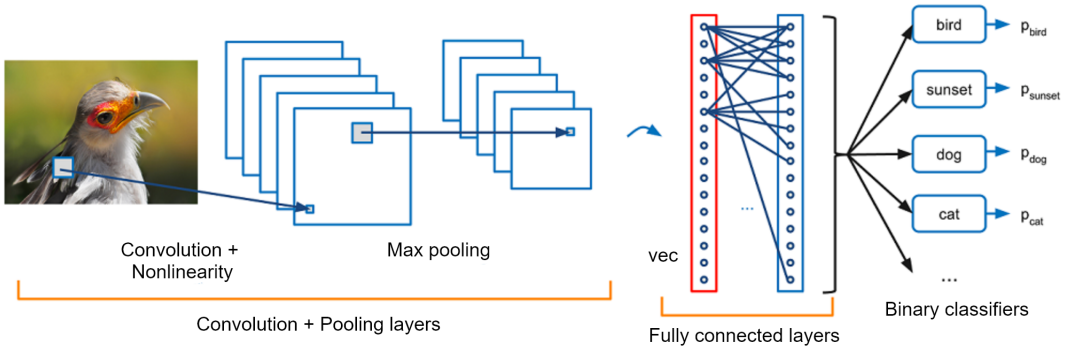


Figure 2.22: Convolution neural network with three layers. Figure from [Raschka and Mirjalili, 2019]

2.12.

$$Y = X \times W \implies Y[i,j] = \sum_{k1=-\infty}^{+\infty} \sum_{k2=-\infty}^{+\infty} X[i-k1, j-k2]w[k1,k2] \tag{2.12}$$

where X is input image pixel value, W is filter value, and Y is the new image pixel value.

The output of equation 2.12 is a single number. This process is repeated by sliding the filter across the image, depending on the scale of the strides. The number of pixels shifts over the input image is called strides [Raschka and Mirjalili, 2019]. When the stride is one, the filters are convoluted by one pixel at a time. An array of numbers is obtained at the end of the process and is used to generate a feature map shown in the figure 2.23.

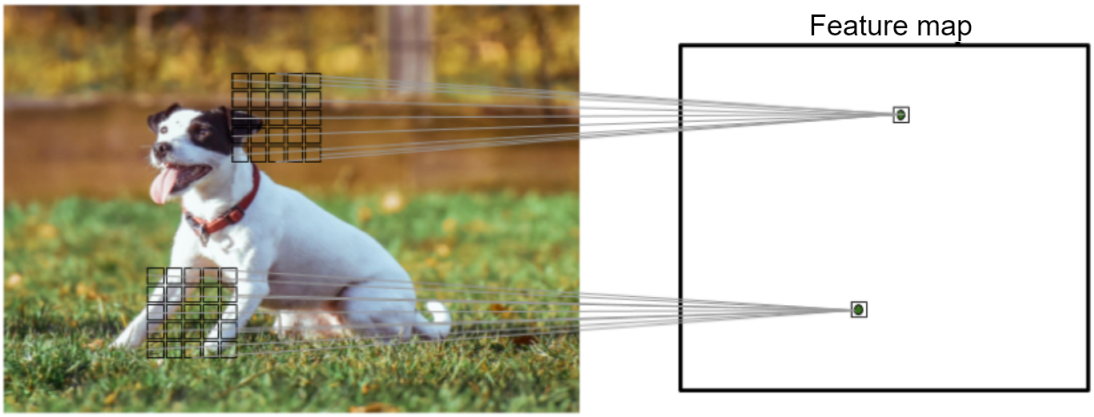


Figure 2.23: Connecting input image to feature maps using receptive fields. Figure from [Raschka and Mirjalili, 2019]

Similarly, the filter used to convolve the image does not fit the input image perfectly. In this case, there are two options: the first alternative is to remove the portion of the image, and the second option is to pad the image with zeros. The second option is better, and it is called padding [Saha, 2018].

The simple implementation of convolution using equation 2.12 in a 3×3 portion of an image with 1 padding using 3×3 filter is shown in figure 2.25

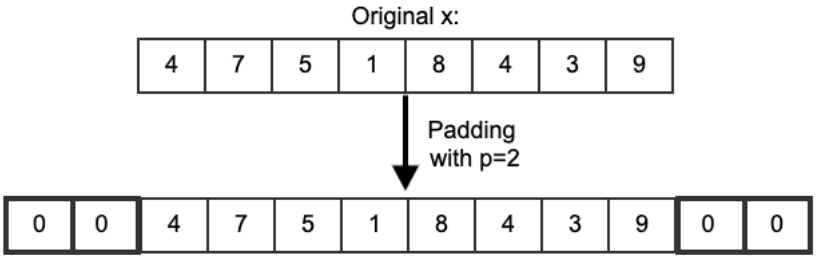


Figure 2.24: Original image matrix padded with two zeros on left and right side. Here the padding size is 2.

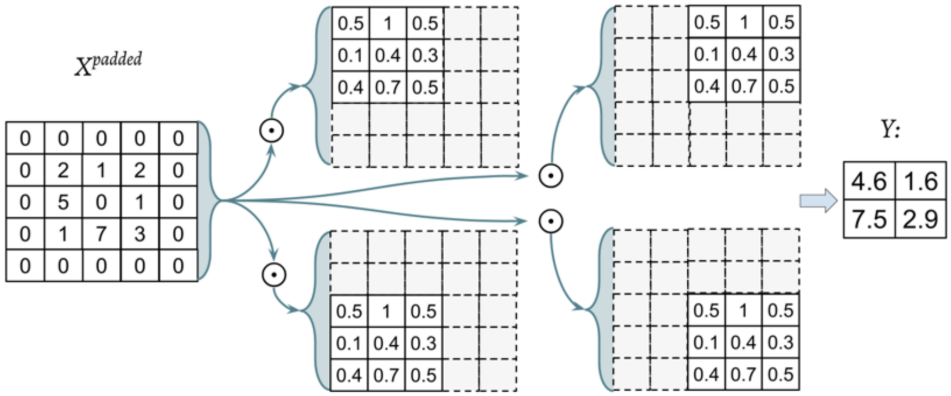


Figure 2.25: Convolution over 3×3 part of an image with 1 padding using 3×3 filter. Figure from [Saha, 2018]

Pooling layer

Pooling layers reduce the dimensionality of each feature map, while retaining important image information. It does this for higher computational efficiency and reduced over-fitting. There are several forms of spatial pooling, such as max pooling, mean pooling, and sum pooling [Saha, 2018].

The largest element is taken from the feature map in max pooling. Max pooling also works as a noise suppressant that eliminates noise activation. In case of mean pooling, the mean of all the elements of the feature map is taken. In the same way, the sum of all the features of the feature map is taken in case of sum pooling. This is shown in figure 2.26.

Fully Connected Layer

The fully connected layer is the end layer of the network that outputs the result. The output of either the convolutional layer or the pooling layer is fed into the fully connected layer, which outputs the n-dimensional vector where N is the number of classes. [Saha, 2018] The values in the n-dimensional vector represent the probability of a specific class. This is shown in figure 2.22.

Training Convolutional Neural Network

The training of CNN is based on the back-propagation. Back-propagation is divided into four steps: forward pass, loss function, backward pass, and weight update. In the forward pass, the training image is passed through the entire network, and the output is evaluated. Since the training data consists of training labels, the loss is calculated

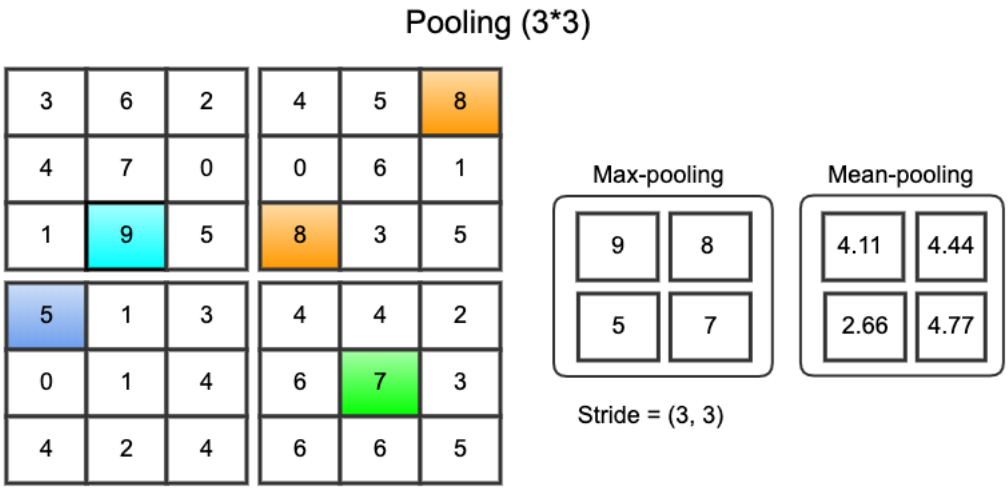


Figure 2.26: Max pooling and mean pooling of the feature map

using the loss function. There are several ways to implement the loss function, and one common method is Mean Squared Error (MSE) defined by equation 2.21

$$loss = \sum \frac{1}{2(true\ label - output)} \tag{2.13}$$

In the first few epochs of training, the loss is very high. The goal here is to reduce the loss by reaching the point where the prediction label is the same as the training label. This is achieved with the help of a backward pass. In the backward pass, the weight that corresponds to most of the loss is evaluated, and the adjustment is made by taking its derivative so that the loss decreases. After that, the weight of the filters is updated in the opposite direction of the gradient.

2.9.2 Activation Functions

An artificial neural network calculates the weighted sum of its data, applies bias, and determines whether or not it should be discharged [SHARMA, 2017]. This decision on whether or not the weighted sum should be discharged, i.e, whether or not the information should be passed on, is made with the help of the activation function. There are generally two types of activation functions; linear and non-linear activation functions. A linear activation function is a simple activation function where no transformation is applied, as shown in equation 2.14 and a network consisting of a linear activation function is easier to train. However, with the linear activation function, the complex structure of the data can not be learned [SHARMA, 2017].

$$\phi_{linear}(z) = z \tag{2.14}$$

Non-linear functions are those with transformation applied and can be used to learn the complex structure of the data. Some non-linear activation functions are discussed below:

Sigmoid Activation Function

Sigmoid activation function is a non-linear activation function that determines the probability of whether a neuron discharge or not. This implies that the output of the

sigmoid activation function is between 0 and 1 so that the high negative numbers are set to zero, and the high positive number is set to one [SHARMA, 2017]. The net input (z) is given as:

$$z = \sum_j w_j x_j = w^T x \tag{2.15}$$

where w is the weight of layer and x is the input to the layer. The sigmoid activation function for the net input is evaluated as:

$$\phi_{logistic}(z) = \frac{1}{1 + e^{-z}} \tag{2.16}$$

where $\phi_{logistic}(z)$ is the sigmoid activation function for the net input z.

When the graph of the sigmoid function is plotted, it is an S-shape curve, as shown in figure 2.27. The function shown in equation 2.16 is differentiable. This mean slope of the sigmoid curve can be calculated. The drawback of using the sigmoid activation function is that it suffers vanishing gradient problem in which the derivative of the function for the net input significantly reduces as z from equation 2.15 increases [SHARMA, 2017]. This, in turn, makes learning weight very slow during the training phase.

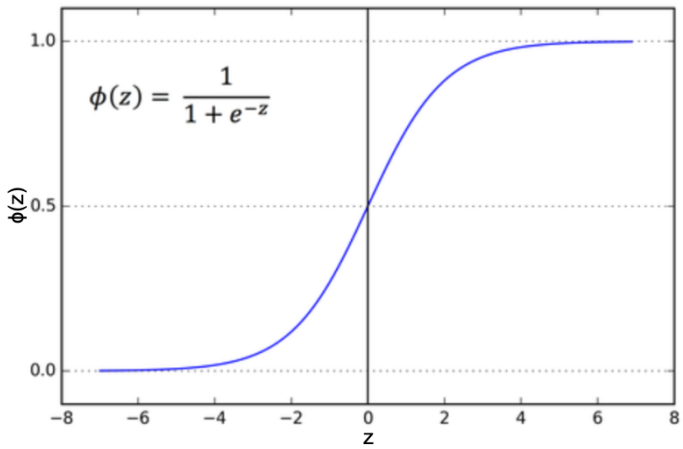


Figure 2.27: Plot of sigmoid activation function. It is a S-shape curve which has the value between 0 and 1. As the value of z gets smaller, the value of sigmoid function is closer to 0 and as z gets larger, the value of sigmoid function is closer to 1.

Rectified Linear Units (ReLU)

Rectified Linear Units is a non-linear activation function used for learning complex neural network function. It is defined as:

$$\phi(z) = max(0, z) \tag{2.17}$$

If the net input from equation 2.11 is less than 0, then the activation is 0 and 1 if the net input is more than 0. Also, the derivative of the ReLU with respect to net input is always equal to 1 for any value of net input [SHARMA, 2017]. This means that ReLU prevents and rectifies the vanishing gradient problem and is appropriate for deep neural networks. The ReLU plot is shown in 2.28

The downside to using ReLU is the dying ReLU problem. In ReLU, if the net input is less than 0 the gradient is 0. When all input is 0 for a neuron, then the gradient in that

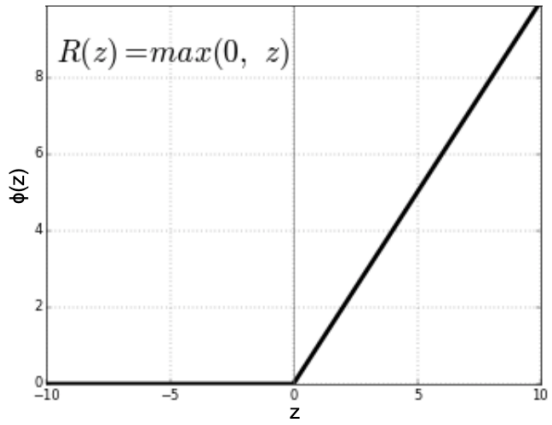


Figure 2.28: Plot of ReLU activation function. The value of ReLU function is 0 if net input is less than 0 and is 1 for all input values more than 0. As z gets larger, the value of sigmoid function is closer to 1.

area will also be 0, because of which weights will not be adjusted during training. Such neurons would therefore avoid reacting to a variation in input [SHARMA, 2017].

Leaky ReLU

The issue of dying of ReLU is mitigated by the use of leaky ReLU. The leaky ReLU achieves so by increasing the range of ReLU functions. When z is less than zero, the output of the leaky ReLU is a small, non-zero, constant gradient instead of 0 gradient [SHARMA, 2017].

Softmax Activation Function

ReLU is an activation function that can only be used for hidden units, but softmax is an activation function that can be used with the output layer. It is used to measure the probability distribution of different classes of input data for a classification problem. The sum of the probabilities is equal to 1 [SHARMA, 2017]. The mathematical representation of the softmax function is shown in equation 2.18.

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad for j = 1, ..., K \tag{2.18}$$

where $\sigma(z)_j$ is the softmax activation function of the net input z.

2.9.3 Optimization Algorithm

In Neural Network, training is based on Back-Propagation method in which the errors obtained from the output layer are propagated in a reverse direction to the other layers. Such errors are used to determine the gradient of the loss function in relation to network weights. The main goal is to find the weights of network for which the loss is minimum. Thus, the gradient obtained is fed to the optimization algorithm that uses it to adjust weights to minimize loss function [Li, 2017]. Adaptive Moment Estimation (Adam) is an optimization algorithm which uses momentum and adaptive learning rates to converge the network faster.

2.10 Accuracy Assessment and Evaluation

Accuracy assessment and evaluation is an important task in any implementation of machine and deep learning algorithm. This is a comparison of the prediction made by the machine and the deep learning algorithm and the true label of the input data. They are also called metrics. There are various metrics for accessing accuracy, and the choice of metrics determines the efficiency of a machine and a deep learning algorithm.

2.10.1 Confusion Matrix

Confusion Matrix determines the overall performance of the classification model using a matrix in which each row represents the true class. The table of the confusion matrix consists of prediction on the x-axis and accuracy outcome on the y-axis. The number of accurate and incorrect predictions and their class-based count values are summarized. The diagonal of the confusion matrix represents the samples that were correctly predicted and all other values outside the diagonal are incorrectly predicted. The confusion matrix is shown in the figure 2.29.

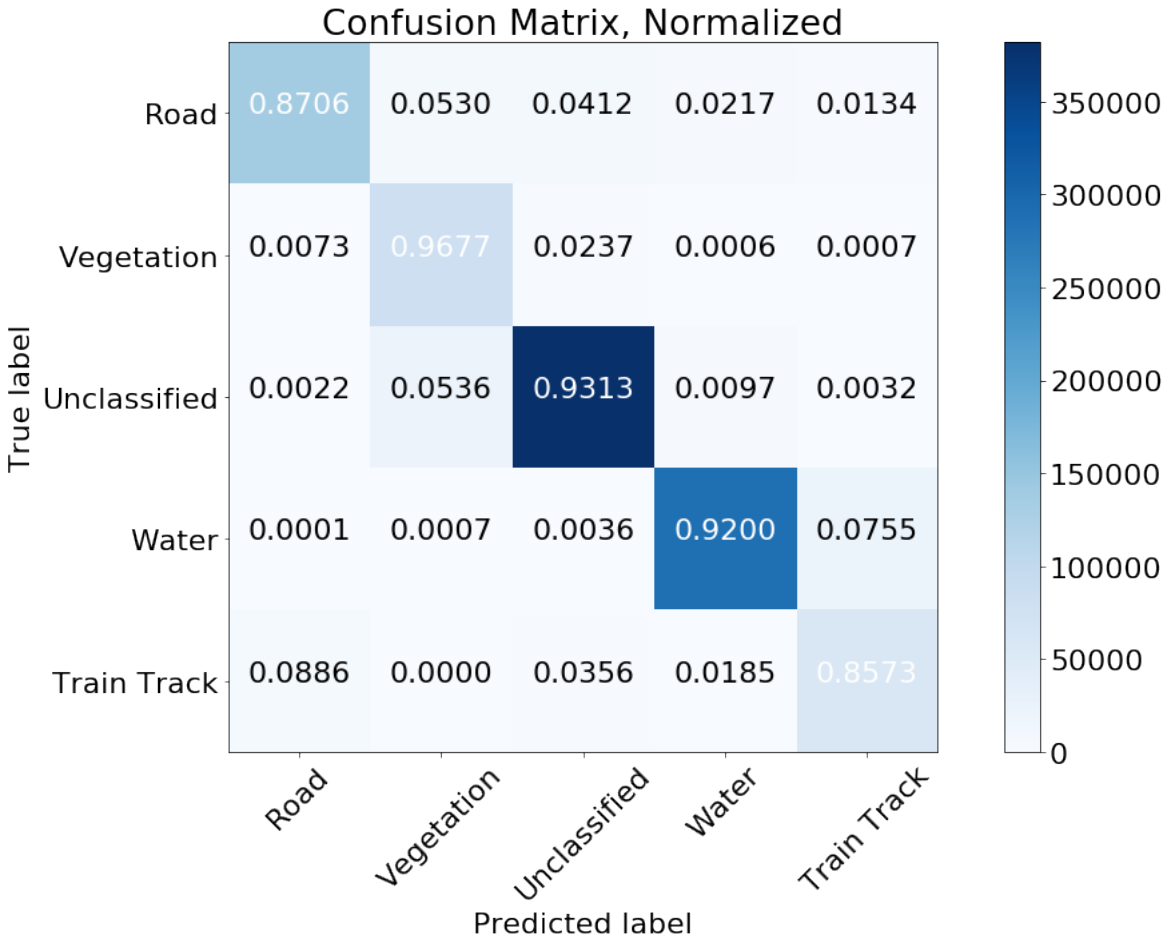


Figure 2.29: Confusion Matrix

2.10.2 Classification Accuracy Score

Classification accuracy, also referred to as accuracy is the ratio of the number of correct predictions to the total number of predictions made. It is the most powerful statistical metric and is misleading when there is unequal number of samples in each class [Mishra, 2018].

$$accuracy = \frac{number\ of\ correct\ prediction}{total\ number\ of\ prediction\ made} \tag{2.19}$$

2.10.3 Precision

Samples that are correctly predicted fall within the True Positive and True Negative groups, while those that are incorrectly predicted are False Positive and False Negative. For every machine learning implementation, the aim is to minimize false positive and false negatives. This is shown in table 2.3

Table 2.3: Table showing the actual and predicted class along with true positive, true negative, false positive and false negative

		Predicted Class	
		Class = Yes	Class = No
Actual Class	Class = Yes	True Positive	False Negative
	Class = No	False Positive	True Negative

Precision is the ratio of the number of correct positive results to the total number of positive results predicted by the classifier. It tries to show whether or not the predicted positive is actually correct. It helps when the cost of false positives is high.

$$precision = \frac{true\ positives}{true\ positives + false\ positives} \tag{2.20}$$

2.10.4 Recall

Th recall is the ratio of the number of correct positive findings to the total number of all relevant samples in the class. Similarly, recall helps when the cost of false negatives is high.

$$recall = \frac{true\ positives}{true\ positives + false\ negatives} \tag{2.21}$$

Both precision and recall must be scrutinized to fully evaluate the effectiveness of a machine learning model. Also, while improving the precision, recall is reduced and vice-versa.

2.10.5 F1-Score

F1-Score is the computation of the accuracy of the model using the weighted mean of precision and recall. A good F1-Score means low false positives and low false negatives. F1-Score values are between 0 and 1. The model is perfect when the F1-Score is 1 and the model fails when the F1-Score is 0. Equation 2.22 indicates the formula for determining the F1-score using precision and recall.

$$F1 - Score = 2 \times \frac{precision \times recall}{precision + recall} \tag{2.22}$$

2.10.6 Cohen's Kappa

When using a machine learning algorithm, there are problems with accuracy assessment and evaluation due to a multi-class and unbalanced class. In this situation, metrics such as classification accuracy, precision, recall do not provide a better assessment of the performance of the classifier. So, Cohen's kappa is used in such situation as it can handle both multi-class and unbalanced class issues well and provide efficient classification performance evaluation. Cohen's kappa compares the accuracy observed with the predicted accuracy [Vieira et al., 2010]. The Kappa score is the number between -1, and 1 and the scores above 0.8 are considered to be good.

$$Cohen'sKappa(k) = 1 - \frac{1 - p_o}{1 - p_e} \tag{2.23}$$

where p_o is the observed accuracy and p_e is the expected accuracy.

2.10.7 Loss Function

Th loss function is a method of determining how well a particular algorithm learns the data. The loss function is used while training the neural network. As the gap between prediction and actual values continues to expand, the loss function also increases. Some optimization function is then used to learn better and thus reduce the loss function. Therefore, in any neural network training, the main aim is to reduce the loss function [Parmar, 2018]. During the evaluation of the model error, the loss function must be chosen, and there are different types of loss function depending on the nature of the machine learning algorithm.

Binary Cross-Entropy

Binary Cross-Entropy is a loss function that is used for a binary classification problem where the target values are in the set 0 , 1. Cross-entropy evaluates a score that summarizes the mean difference between the actual and the predicted probability of a certain class. The score is reduced using various optimization functions. Binary Cross-Entropy evaluation is shown by equation 2.24:

$$D_{binary}(y', y) = -(ylog(y') + (1 - y)log(1 - y')) \tag{2.24}$$

where D_{binary} is the binary cross-entropy score, y is the actual target value and y' is the predicted target value.

Categorical Cross-Entropy

Categorical cross-entropy is used in case of multi-class classification. It is given by equation 2.25:

$$D_{categorical}(y', y) = -\sum_{i=1}^{k} y_i log(y'_i) \tag{2.25}$$

where $D_{categorical}$ is the categorical cross-entropy score, y is the actual target value and y' is the predicted target value.

2.11 Edge Detection

Edge detection is one of many essential parts of computer vision systems. Edge detection helps in efficient analysis of the image by reducing the data size and at the same time, preserving the important information in the data. Edges are the significant local changes in the image intensity occurring on the boundary between two different regions in an image. The main goal of the edge detection technique is to extract the pixels in the image at which the intensity changes sharply [Canny, 1986]. There are many edge detection algorithm, in this study, Canny Edge Detection algorithm is used.

2.11.1 Canny Edge Detection

Canny edge detection is a method of edge detection that uses a multi-stage process to detect edges in the image. This uses a Gaussian derivative filter to measure the strength of the gradients. Following this, the smooth image is obtained. The possible edges are then thinned down to 1-pixel curves by eliminating non-maximum pixels of gradient magnitude and, eventually, edge pixels are retained using hysteresis thresholding on the gradient magnitude [Maitra, 2019]. In the hysteresis thresholding, the pixels above the high threshold are considered to be edges and included in the edge pixels. Those pixels between the low and high thresholds lying on the same line as high threshold pixels are also included in the edge pixels. Pixels with a small threshold are removed from the edge points.

The Canny edge detection algorithm has three parameters that needs to be optimized. These are gaussian width, low and high level of the threshold for the hysteresis thresholding. The steps in this are as follows [Irwan Hariyono and Windiastuti, 2018]:

1. First, the image is smoothed using Gaussian blur to remove the noise in the image.
2. The edges are marked where the gradient of the image has magnitude.
3. The direction of the edge is calculated.
4. The local maxima are only marked as edges
5. Finally, the edges are determined using hysteresis thresholding by suppressing all the edges that are not connected to certain edges points.

Method

3.1 Data Acquisition

In this study, hyperspectral and LiDAR images from multiple places are analyzed and processed for the extraction of valuable information. Airborne hyperspectral images and LiDAR data have been produced and processed by Terratec AS. The image data is the property of the Municipality of Bærum with the aim of using it for different data analysis and mapping.

Figure 3.1: This image represent the airborne hyperspectral data collection by Terratec AS. Figure from [Terratec, 2019]

Hyperspectral images were acquired in August 2019 with two HySpex sensors mounted in a gyro frame of the aircraft. Th airplane was laid at a maximum opening angle of 16 degrees for the HySpex SWIR-384 sensor as shown in figure 3.3. The data collected by HySpex sensors is in DN format (describe pixel values). The acquired image data is encoded in a geospatial domain using the Band Sequential (BSQ) process. BSQ file format stores images as a flat binary file with separate metadata that includes header information. In BSQ format, each image band is stored as an individual file, making it simpler to read and process [The Global Land Covering Facility, 2006]. The DN file format is converted to the radiance data using the HySpex RAD software. The hyperspectral image is the radiance data with 0.3 m resolution for the HySpex VNIR-

1800 sensor and 0.7 m resolution for the HySpex SWIR-384 sensor. The data from the HySpex VNIR-1800 sensor consists of images in the visible and near-infrared (400 - 1000 mm) spectral range, and the data from the HySpex SWIR-384 sensor consists of images in the short-wave infrared (1000 - 2500 mm) wavelength region [Terratec, 2019].

Airborne images are geo-referenced and ortho-rectified. Georeferencing is the process of giving the image location coordinates, and ortho-rectification is the process of eliminating the distortion in the image. Image georeferencing is performed using position and timing data obtained from the Global Navigation Satellite System (GNSS). Similarly, ortho-rectification of hyperspectral data is done using PARGE software based on a 30 cm digital elevation model obtained using the Leica ALS70 laser scanner. Nearest neighbor interpolation is used for georeferencing and location correction along with a transformation in the WGS84 UTM32 coordinate system [Terratec, 2019].

Hyperspectral images were taken from three locations; Sandvika, Honeføss and Hamar with a total area coverage of 37.4 km^2. Information on the collection of hyperspectral data by Terratec AS is given in table 3.1. There are a total of 14 flight line images over the Sandvika area. Each of the flight line images covers from west to east, as shown in figure 3.2. The corresponding mosaic version of the flight line images was created by bilinear interpolation of the ortho-rectified flight strips. Moreover, only 8^{th} and 9^{th} flight line images were used in this analysis [Terratec, 2019].

Table 3.1: Description of hyperspectral data collection by Terratec AS [Terratec, 2019]

Coverage number	Stripe number	Operator	Date
41161	1 - 15	Magnus Nilsson	24.08.2019
41162	1 - 8	Ainar Härm	03.08.2019
41162	1 - 17	Andrei Tanasescu	28.07.2019

LiDAR data for each location is collected together with hyperspectral images. It is recorded using the Reigl VQ-1560i laser scanner with an integrated Inertial Measurement Unit (IMU) mounted on the Gyro mount, as shown in 3.3. LiDAR data varies from hyperspectral images in that it offers dense, detailed and precise 3D coverage of both the object and the ground surface. Also, the LiDAR sensor is an active sensor with a maximum of three wavelengths. It provides information on the elevation of urban areas and thus distinguishes objects based on elevation information [Zhong et al., 2017].

3.1.1 Hyperspectral Data

Hyperspectral images were collected using two HySpex sensors mounted on the gyro frame of the aircraft.

HySpex System

Hyperspectral system consist of two sensors: HySpex VNIR-1800 sensor and HySpex SWIR-384 sensor. These sensors are manufactured by Norsk Elektro Optikk AS, a Norwegian company that produce airborne and ground-based hyperspectral imaging systems. HySpex sensors detect the solar radiance reflected on Earth's surface in a

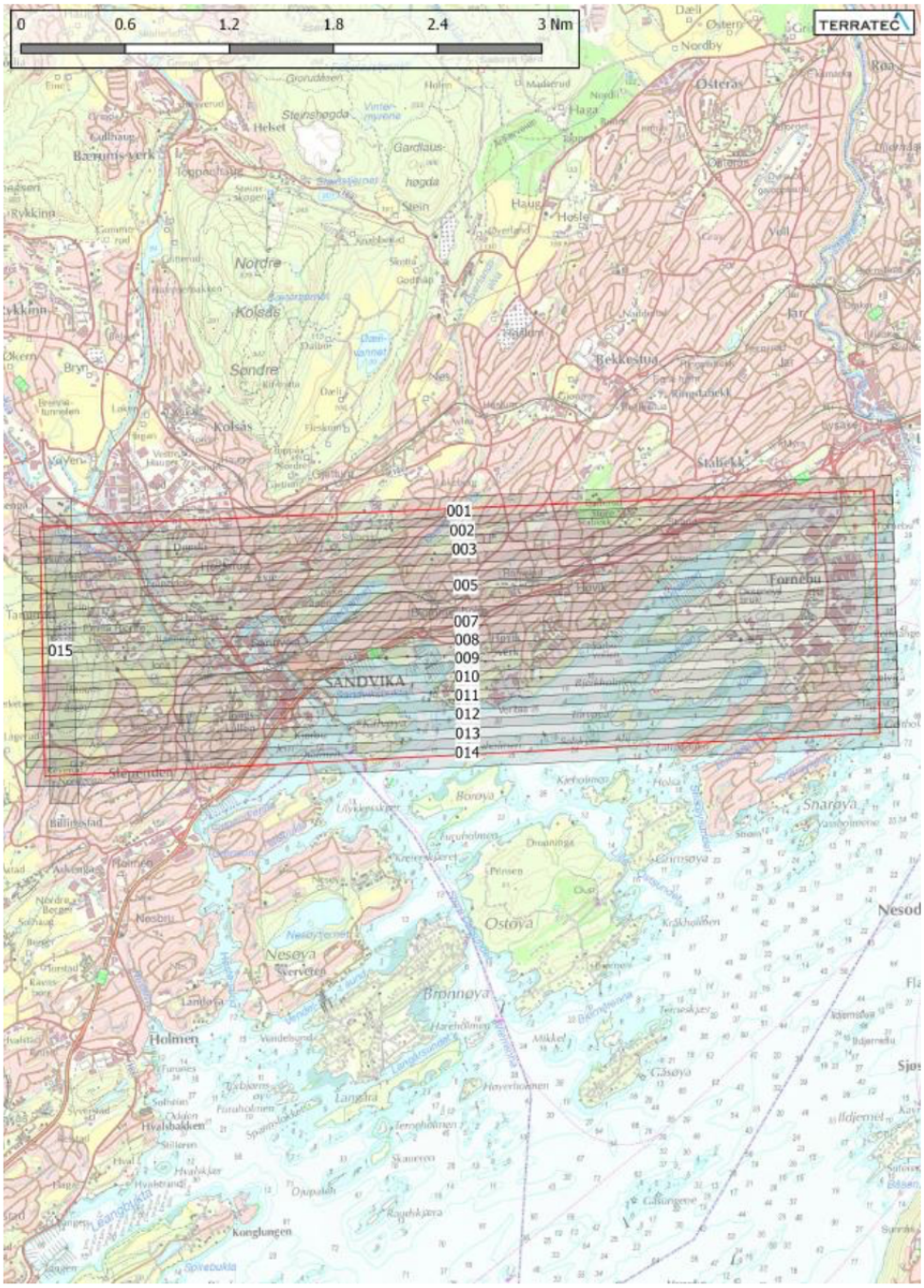


Figure 3.2: Data collection over coverage number 41161. This coverage number occupies the area of Sandvika. 15 flightlines images were obtained. Figure from [Terratec, 2019]

wavelength range from 400 nm to 2500 nm. The HySpex VNIR-1800 sensor captures the image in the visible and near-infrared region from 400 nm to 1000 nm wavelength range with 186 spectral channels, while the HySpex SWIR-384 sensor captures the image in the short-wave infrared region from 930 nm to 2500 nm with 288 spectral channels. During the acquisition of airborne data using HySpex sensor, both sensors are used in combination with INS and GPS, which are the navigation device used to geo-reference images based on the position and altitude of the camera [AS, 2019].

Both of these sensors are based on the pushbroom scanning principle, as shown in figure

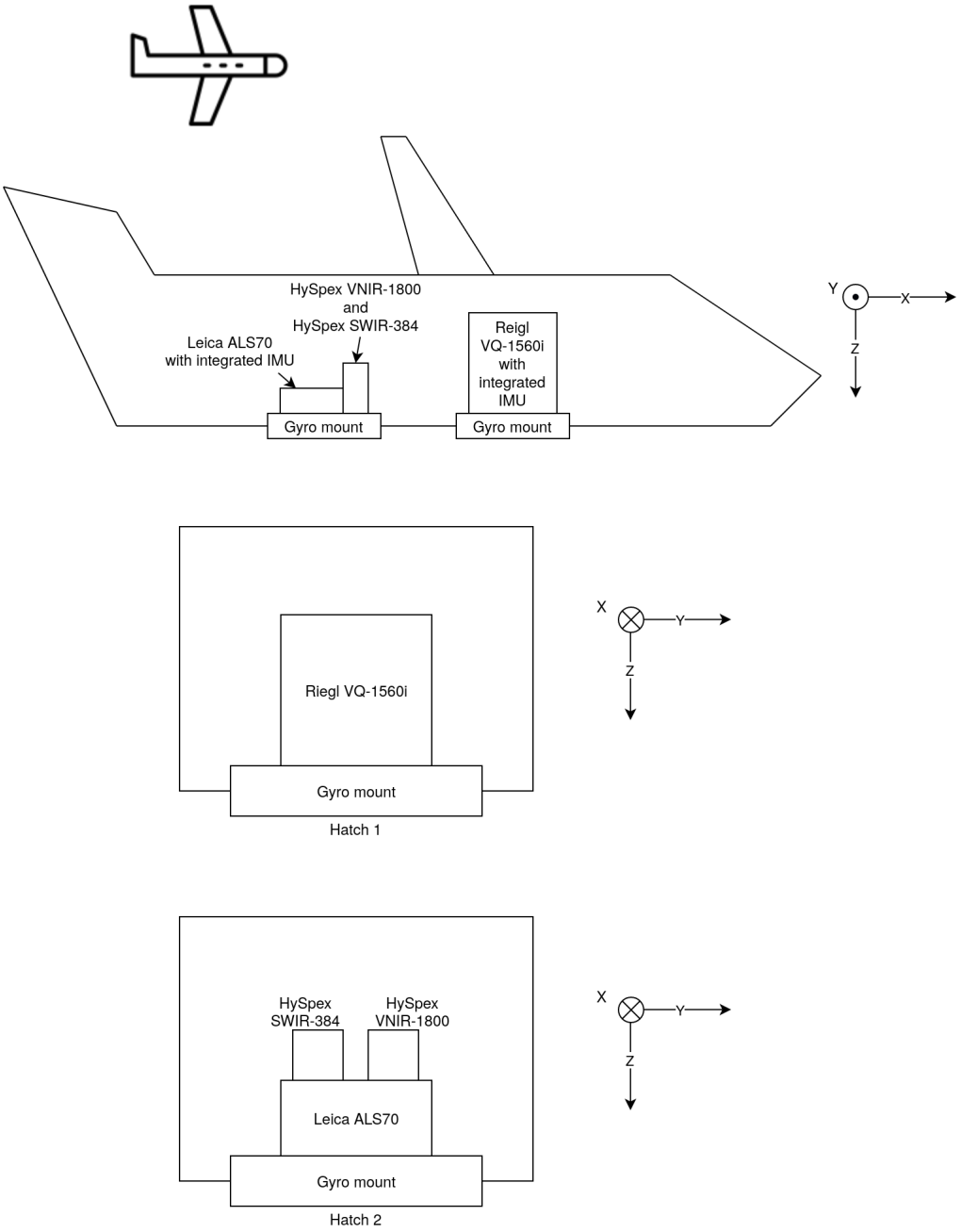


Figure 3.3: LiDAR sensor and Hyperspectral sensor setup. Figure from [Terratec, 2019]

3.4. The incoming light passes through the optics and focuses on the slit through the focusing mirror. Then the light passes through the collimating mirror to the dispersive element. The two focusing and collimating mirrors are aspherical mirrors that avoid the phenomenon of spherical and chromatic aberration while at the same time reducing stray light. The dispersive factor separates different wavelengths and focuses them on the detector array using different lens optics, as shown in figure 3.4. Slice of a two-dimensional hyperspectral image can be obtained from a detector array with spectral details in one direction and a spatial one in the other direction [AS, 2019]. Only one section of the scene is taken at a time, and the pushbroom scan is used to get a two-dimensional spatial representation of the scene. The scan is achieved by installing the sensor in an aircraft and capturing the images in pushbroom scanning mode as shown

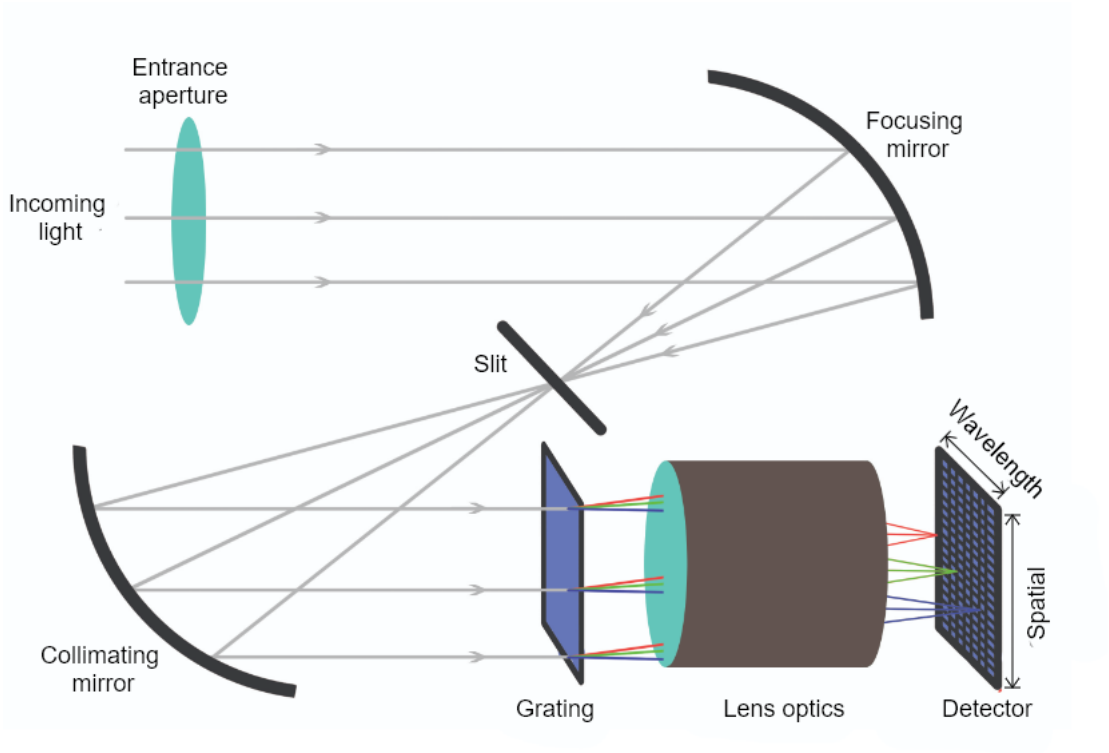


Figure 3.4: Working principle of HySpex Sensor (Pushbroom Principle). Modified figure from [AS, 2019]

in figure 3.3.

The specification of HySpex VNIR-1800 and SWIR-384 sensor is shown in table 3.2

Table 3.2: System specification of HySpex VNIR-1800 and SWIR-384 sensor [Terratec, 2019]

Specification	VNIR-1800	SWIR-384
Spectral Range	400 - 1000 nm	930 - 2500 nm
Spatial pixels	1800	384
Spectral channels	186	288
FOV	17 degree	16 degree
Pixel FOV across/along	0.16/0.32 mrad	0.73/0.73 mrad
Bit resolution	16 bit	16 bit
Dynamic range	2000	7500
Noise Floor	2.4 e-	150 e-
Max speed	260 fps	400 fps

3.1.2 Software and programs

QGIS

QGIS is a professional user-friendly Open Source Geographic Information System (GIS). The initial goal of a QGIS project was to provide a GIS data-viewer, but now it has evolved a lot with functionality such as vector creation, classification based on machine learning algorithms, LiDAR tools, and many more [QGIS, 2020]. QGIS supports raster

and vector data formats along with support for other formats with the help of different plugins. QGIS has been extensively used in this project. First and foremost, QGIS is used to view the large hyperspectral images and clip the part of the image that is important for further analysis. Different image processing is done using a raster calculator. Also, vector tools are used to generate a mask from the images and thus apply the mask to the raster [QGIS, 2020]. Similarly, large hyperspectral images are resized and cropped using QGIS raster tools so that they can be easily accessed and further processed using Python.

ENVI

ENVI is an image analysis software used to analyze GIS, remote sensing, and derive useful information from images in order to make better decisions. ENVI was developed by Harris Geospatial Solutions, Inc [Harris Geospatial Solution, 2020].The image processing kit of ENVI consists of specialized spectral, geometric correction, radar analysis, raster, and vector processing tools. ENVI is frequently used in this study for visualizing and processing hyperspectral images. Also, as pixel-wise classification is used in this study, image class labels are generated using the ENVI Region Of Interest (ROI) tool. It is used to represent and construct spectral libraries. Similarly, VNIR and SWIR images are also combined using ENVI [Harris Geospatial Solution, 2020].

LAStools

LAStools is a system used to process LiDAR point clouds. Since LiDAR data is used in this project to mask different portions of hyperspectral images, LAStools is used for this purpose. It consists of a variety of tools to display, classify, convert, filter, rasterize, triangulate, crop, and polygonize LiDAR data [rapidlasso GmbH, 2020]. DSM and DEM are obtained using LAStools which are then processed using the QGIS raster calculator to obtain nDSM.

3.1.3 Programming Languages and tools

Python

Python is a programming language that consists of several packages and libraries for reading, writing, modifying, and analyzing data. Python is user-friendly and easy to learn and understand. It is commonly used in the area of data analysis, and machine learning. In this study, python 3 is used for reading, visualizing, and analyzing hyperspectral data. It is achieved in combination with different libraries and modules. The python scripts that are used in this study are included in the appendix **??**.

Spectral Python

Spectral Python (SPy) is a free open source python module for processing hyperspectral data which has a function for reading, visualizing, processing, and classifying hyperspectral images. In this project, spectral python is used for reading the ENVI header files. Also, PCA of hyperspectral images is evaluated using spectral python.

Pandas

Pandas is an open-source data structure and data analysis tools for Python programming language. It is used for processing numpy array data, and csv files. In this project, pandas is used to convert numpy array of hyperspectral data into a pandas dataframe. Pandas dataframe is a two-dimensional tabular data that contains labeled axes (rows and column). Different arithmetic and logical operations can be performed on the dataframe using pandas.

Scikit-Learn

Scikit-learn is an open-source and easy to use machine learning library for python programming language. It is simple and efficient tools for predictive data analysis. Scikit-learn is used for classification, regression, clustering, dimensionality reduction, model selection, and preprocessing of hyperspectral data. In this project, Scikit-learn is used to implement supervised classification such as RF and SVD. Also, it is used to implement dimensionality reduction with PCA.

Keras

Keras is a high-level Application Programming Interface (API) for performing neural network task which is written in python and runs on top of TensorFlow, CNTK or Theano. It has supports for both convolution neural networks, recurrent networks and also the combination of two. One of the advantage of using Keras is that it is very fast and can run seamlessly on CPU and GPU. In this project, Keras is used to implement image segmentation using CNN.

3.1.4 File formats

Band Sequential

BSQ format is a popular standard image format that stores images as a flat binary file with the header information in a separate metadata file [The Global Land Covering Facility, 2006]. In BSQ format, each line of data is immediately followed by the next line in the same spectral band so that each image band is stored as an individual file and the user can access and manipulate the single image band [Geospatials, 2019].

TIFF

TIFF is a widely used image format for raster images. TIFF files are often referred to as GeoTIFF in the sense that the file contains the geographical information of the image. Image compression can be maintained using the TIFF file format, along with the preservation of relevant image information.

3.1.5 Shape File

Shape File is usually used to save the vector file. It is used to store geometric position, shape, and attribute of the geographical feature of the image in the form of circles, lines, or polygons. Severals shape files are generated in this project to mask different portions

of the hyperspectral image. The nDSM shape file is created to mask the buildings from the image. It is stored as .shp, .shx, .dbf, and .prj files [Archis, 2019].

3.2 Preprocessing of hyperspectral data

3.2.1 Hyperspectral data types

The data obtained from Terratec AS consist of three types of hyperspectral flight line images. The first data, original DN dataset, is processed to radiance data, and this radiance data is atmospherically corrected (described in chapter 3.2.2) to obtain a reflectance data. Spectral artifacts are still present in the atmospherically corrected data due to their high spectral resolution. To remove these artifacts, spectral polishing is used. Spectral polishing is the process used to eliminate noise and calibration anomalies from atmospherically corrected data in the spectral domain.

The radiometric variation algorithm is used for spectral polishing of the reflectance image which is performed in ATCOR-4. The soil pixels are first masked using the vegetation index ($0 < NDVI < 0.33$). Spectral polishing removes the spikes without disrupting the spectral shape. The average reflectance spectrum is then measured and smoothed over all soil pixels with a 5-channel filter, except for atmospheric water vapor regions where linear interpolation is applied. The ratio between the filtered and the original spectrum of soils is the spectral polishing function applied to all pixels of the image. Figure 3.5 shows the spectra of the raw reflectance image and spectra of polished reflectance image after the application of spectral polishing.

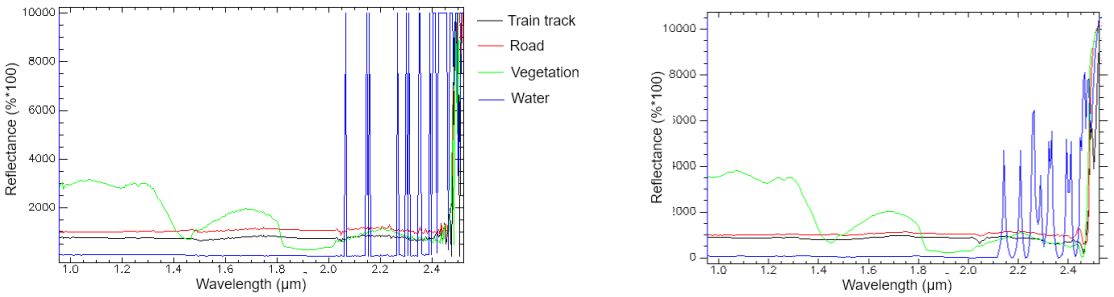


Figure 3.5: Left: Different spectra of atmospheric corrected reflectance image. Right: Different spectra of reflectance image after spectra polishing

3.2.2 Atmospheric correction

Hyperspectral data obtained from Terratec AS are radiance datasets. Radiance consists of all reflected radiation from the surface material, neighboring pixels, and the atmosphere, resulting in lighting and atmospheric transmitting effects. The electromagnetic energy reflected from the material is absorbed by the atmosphere on the way to the sensor, which degrades the quality of the spectra. It also causes more uncertainty and error because a mathematical analysis cannot be done and spectral libraries cannot be used. The atmospheric correction must, therefore, be performed on the radiance data in order to generate the corresponding reflectance data. Factors such as solar illumination, sensor geometry, and terrain information must be considered when achieving the objective of atmospheric correction, which is to reduce the effect of the atmosphere.

Atmospheric correction is performed using ATCOR software and the ATCOR used for airborne imagery is ATCOR-4 [Richter, 2018]. ATCOR-4 performs atmospheric correction and estimates the surface reflectance. ATCOR uses the AFRL MODTRAN code to determine the atmospheric look-up table (LUT) database and the parameters for atmospheric correction were setup manually.

Atmospheric Parameters

Visibility and optical thickness are two parameters that are used to characterize the atmosphere. The estimate of maximum horizontal distance that a human eye can see a dark object in a bright sky is the visibility which is given by equation 3.1.

$$VIS = \frac{1}{\beta} \ln \frac{1}{0.02} \tag{3.1}$$

where β is the extinction coefficient. Similarly, the optical thickness (δ) is the product of the coefficient of extinction (β) and the vertical length (x) of the path from the sea level to the space given as:

$$\delta = \beta \times x \tag{3.2}$$

For the path from sea level to space, two factors, molecules and aerosols, play an important role in changing the optical thickness from equation 3.2 to 3.3. This means that the optical thickness is due to the molecular scattering, aerosol, and molecular absorption [Richter and Schläpfer, 2002]. The optical thickness is unpredictable and is difficult to eliminate during atmospheric correction, so this parameter is kept constant.

$$\delta = \delta(molecular\ scattering) + \delta(aerosol) + \delta(molecular\ absorption) \tag{3.3}$$

Based on these measures, the important atmospheric parameters that vary in space and time are aerosol type and water vapor. The aerosol type includes the absorption and dispersion characteristics of the aerosol particles. The type of aerosol can be measured from a geographical location. The aerosol types provided by ATCOR are rural, urban, maritime, and desert [Richter, 2018].

Generally, the water vapor components are about 920 nm to 960 nm wavelength range. But if the sensor does not have this wavelength range, as in the case of Landsat TM, seasons (summer/winter) are used to estimate the water vapor components [Richter, 2018].

Radiation Components

Two components of radiation in the solar wavelength region are the path radiance and the reflected radiance. Path radiance(L_{path}) is a scattered photon without ground interaction. Reflected radiation consist of two elements, one of which is reflected from a target material pixel and the other from a neighborhood pixel [Richter, 2018]. Equation 3.4 displays the total radiance signal of L. The radiance components are shown in figure 3.6.

$$L = L_{path} + L_{pixel\ reflected} + L_{neighbour\ reflected} \tag{3.4}$$

The radiation reflected from the target material pixel is the main concern, so the measurement and deletion of path and neighborhood pixel reflection is the main task of atmospheric correction [Richter, 2018].

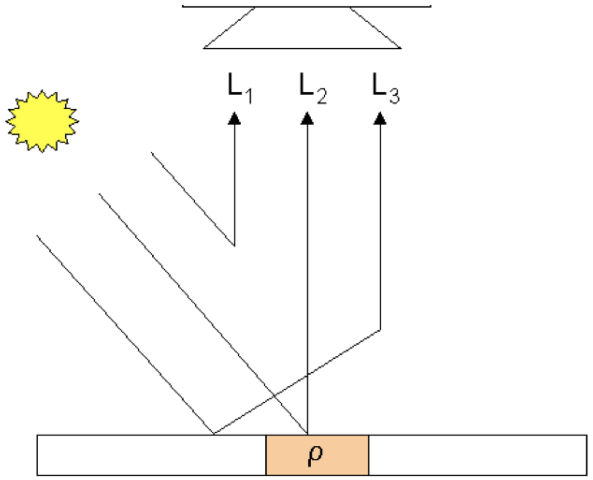


Figure 3.6: Three components of radiation. L_1: path radiance, L_2: Pixel reflected, L_3: neighbourhood of pixel reflected. Figure from [Richter, 2018]

BRDF correction

The viewing angle and the geometry of solar illumination influence the reflectance of the target material. This effect is called Bidirectional Reflectance Distribution Function (BRDF). BRDF effects generally occur as across-track brightness gradients after atmospheric correction is applied to hyperspectral data. These effects are strong in rough terrain with slopes facing the Sun and other sections away from the Sun [Richter, 2018]. ATCOR provides a method for BRDF effect correction by identifying a common anisotropy correction factor using both the surface cover type characterization and the per-pixel observation angle [Richter, 2018]. The overall process of atmospheric correction to obtain reflectance image is shown in figure 3.7.

3.2.3 Generation of Region of Interest

The flight line images obtained from Terratec AS were huge and were computationally inefficient to work with. Therefore, the first task was to align all the images on the flight line and select the ROI for training, testing, and validation. To obtain ROI, flight line images of the Sandvika region are opened on QGIS. Then, with the aid of raster extraction by extent tool, many sections of the images are extracted and saved along with the header file. These extracted images cover a certain portion of the flight line images and have interesting information that meets the requirements of this study. This is shown in figure 3.8.

3.2.4 Masking elevated objects

Airborne hyperspectral images cover large areas of land and consist of many artificial and natural materials. As in this study, the main objective is to extract the edges of the road in such a way that the main emphasis is on the non-elevated part of the image. Also, the spectral signatures of various land cover materials such as roads and rooftops are identical, making it more difficult to distinguish and identify. So, to avoid this misclassification, the buildings are masked from the image with a mask generated from LiDAR data. This is done in QGIS and LAStools and the process is shown in figure 3.9.

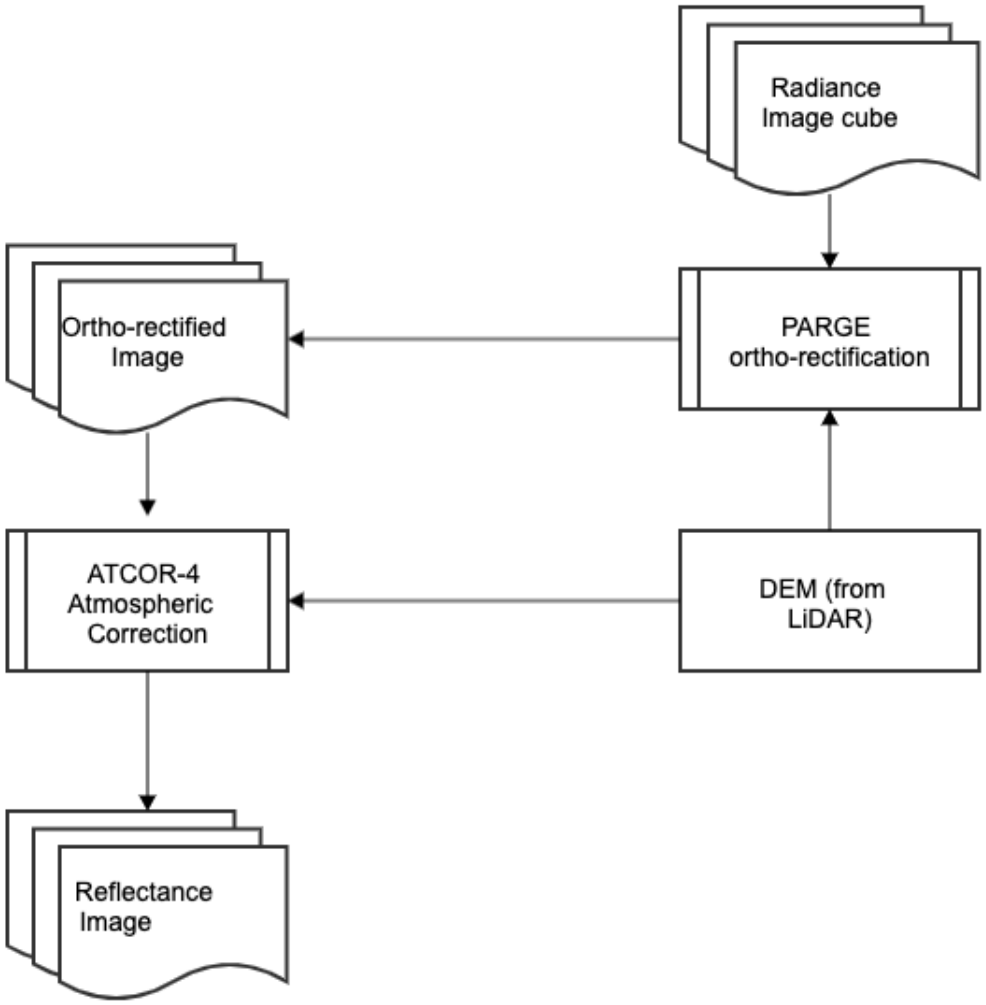


Figure 3.7: Atmospheric correction and generation of reflectance image using ATCOR-4 and BRDF correction. Modified figure from [Richter and Schläpfer, 2002]

The main purpose of masking is to eliminate all items that are elevated above 1 m in height, except vegetation. So, a mask layer is first generated using LiDAR data. Using LAStools, DSM, and DEM are generated using the first and last return of the lidar points. The nDSM raster is then obtained by subtracting the DEM from the DSM. As this nDSM raster consists of vegetation such as trees so that NDVI, a raster obtained from the hyperspectral image using equation 2.2, is subtracted from nDSM to produce nDSM raster without vegetation. Then the mask raster for all artifacts above 1m is obtained by the threshold. Finally, masking is performed using the hyperspectral image and mask raster in QGIS to generate final image cube without elevated objects.

3.2.5 VNIR and SWIR image stacking

VNIR images respond more to color changes. This implies that in VNIR images, the same objects of different colors are classified as different objects. Also, materials of different chemical compositions with the same color are classified as the same objects in VNIR. Likewise, the SWIR images used in this project have a very low spatial resolution,

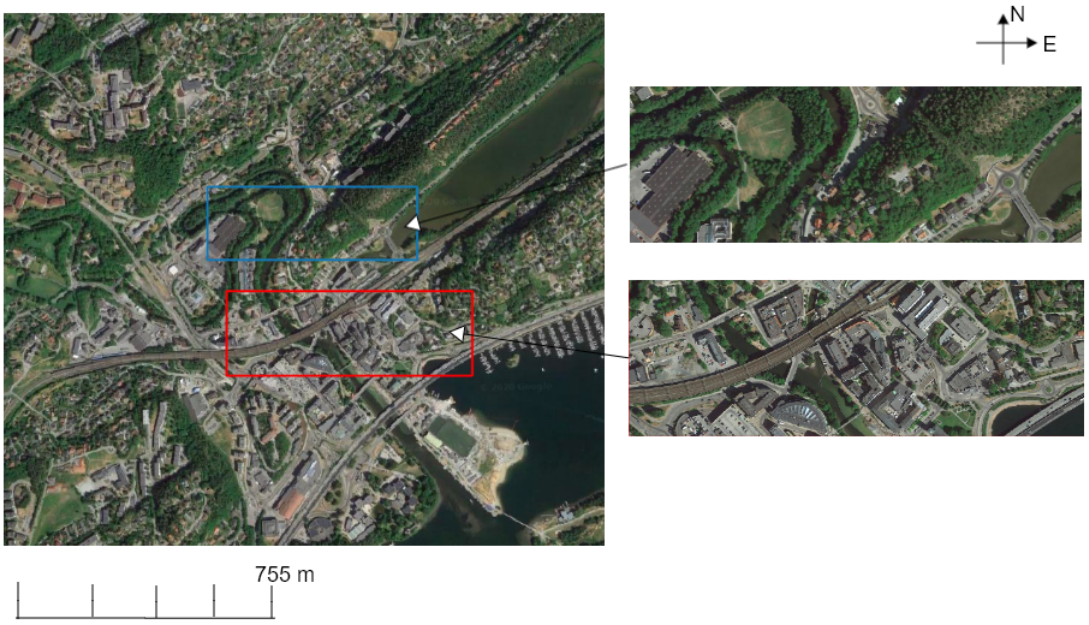


Figure 3.8: Lower right is training area and upper right is validation area.

which prevents a good classification of small surface materials. It was therefore decided to combine both the VNIR and the SWIR image. This is done using the ENVI layer stacking tool. The process flow is shown in figure 3.10

Since the VNIR image has a spatial resolution of 0.3 m and the SWIR image has a spatial resolution of 0.7 m, the VNIR image is re-sampled using the nearest neighbor resampling method from 0.3 m to 0.7 m. Finally, the SWIR and the re-sampled VNIR image are combined using geospatial data of the image.

3.2.6 Training class generation

The supervised classification is used in this project for the classification of hyperspectral images. The first step in any supervised classification is to generate labels for each pixel of the hyperspectral image. Since each pixel in the image refers to specific material, the pixel-wise supervised classification is enforced. For this reason, training classes are created at the pixel level. This is done in ENVI, where polygons are created for each image object along with their labels. Polygons are created from different sections of the image for each class. Four classes are generated from the hyperspectral image in this study. These classes are roads, vegetation, train tracks, and water. The pixel-wise labeling of the study area is shown in figure 3.11.

There should be a balance in the number of pixels for each class in every supervised classification. This means that the number of pixels in each class should be equal. Table 3.3 indicates the number of pixels in each class.

3.3 Feature Extraction

Hyperspectral images, consisting of hundreds of narrow, contiguous spectral bands with a high spectral resolution, allow the retrieval of continuous spectral characteristic curves

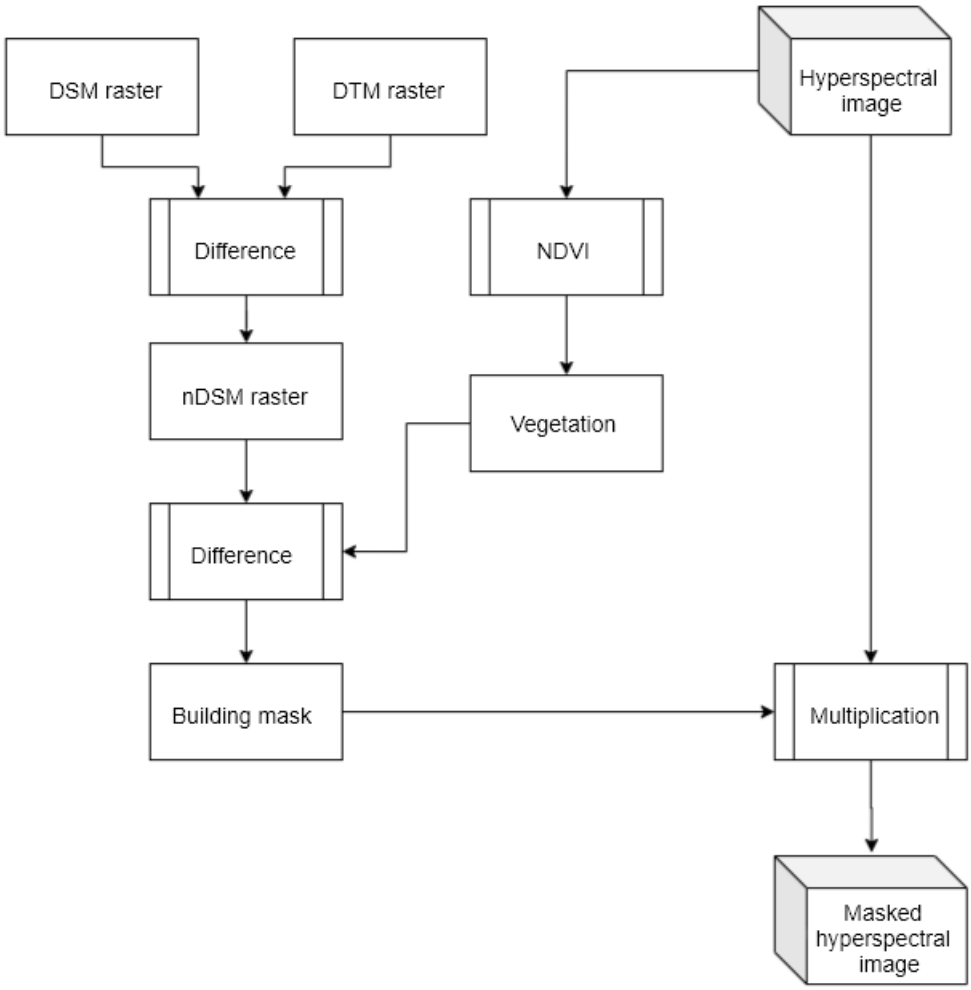


Figure 3.9: Masking hyperspectral image to remove elevated materials

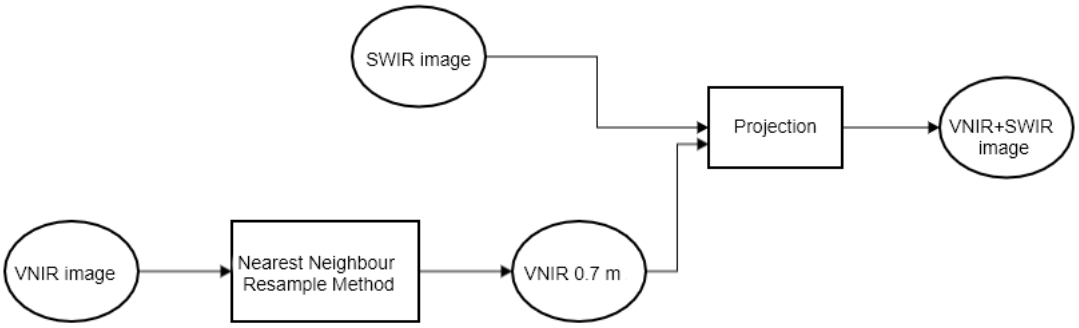


Figure 3.10: VNIR and SWIR image stacking workflow

and therefore serve as a powerful tool for the classification of Earth's surface material [Ma et al., 2013]. Not all of the hypercube bands provide the same proportion of information as the neighborhood bands are redundant, strongly correlated and subject to the Hughes phenomenon [Ma et al., 2013], where the precision of classification increases gradually when the number of spectral bands increases, but declines signifi-

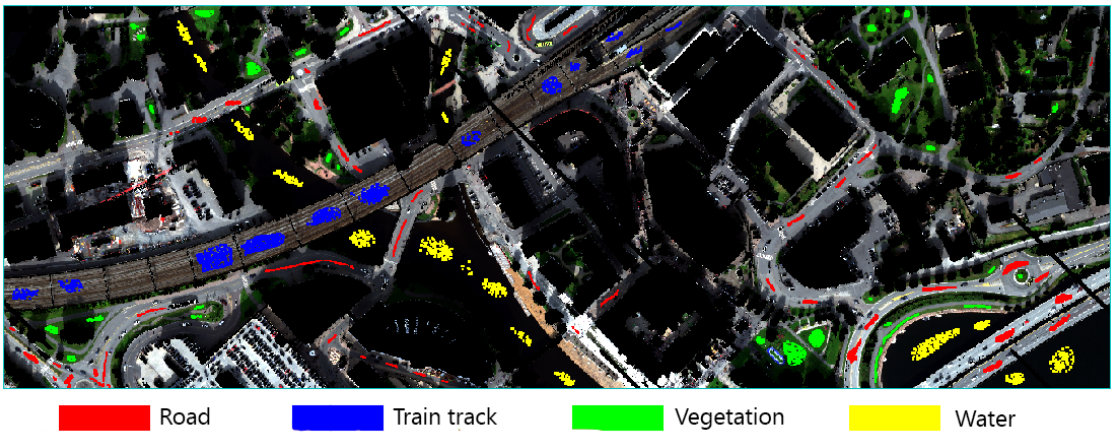


Figure 3.11: Hyperspectral image with class labels

Table 3.3: Pixel count of each class

Class	Pixel Count
Road	14774
Train track	15036
Vegetation	12050
Water	14837

cantly as the band number reaches a certain high value [Ma et al., 2013]. The presence of large spectral bands, also called curse of dimensionality, significantly affects the ability of classifiers to generalize, leading to catastrophic predictive results [Cao et al., 2016]. The objective of the feature extraction is to select certain bands or a combination of bands from the full band of hyperspectral image to overcome the Hughes phenomenon in the classification. Various methods for feature extraction exist and, in this study, techniques such as PCA and NDVI are applied to the hyperspectral image to extract important features [Uddin et al., 2017].

3.3.1 Principal Component Analysis

Neighboring bands of hyperspectral images are highly correlated and provide the same information about the target. PCA is used to remove this correlation between bands. PCA is an unsupervised feature extraction method that performs a linear band combination so that a new set of uncorrelated bands is generated, each providing unique information of the target object [Rodarmel and Shan, 2002].

In this analysis, PCA is implemented in python using scikit-learn package. One of the essential tasks of implementing the PCA is to select the number of principal components and find the components that have high variance. 50 principal components are taken and scree plot is drawn to find the principal components that have dominant variance. The variance of the first six principal components are shown in the figure 3.12

As shown in figure 3.12, the information content continues to decline with an increase in the number of PCA components and only the first 3 components contain approximately 99% of the total variance. The score image of the first six principal components are also shown in figure 3.13 - 3.15.

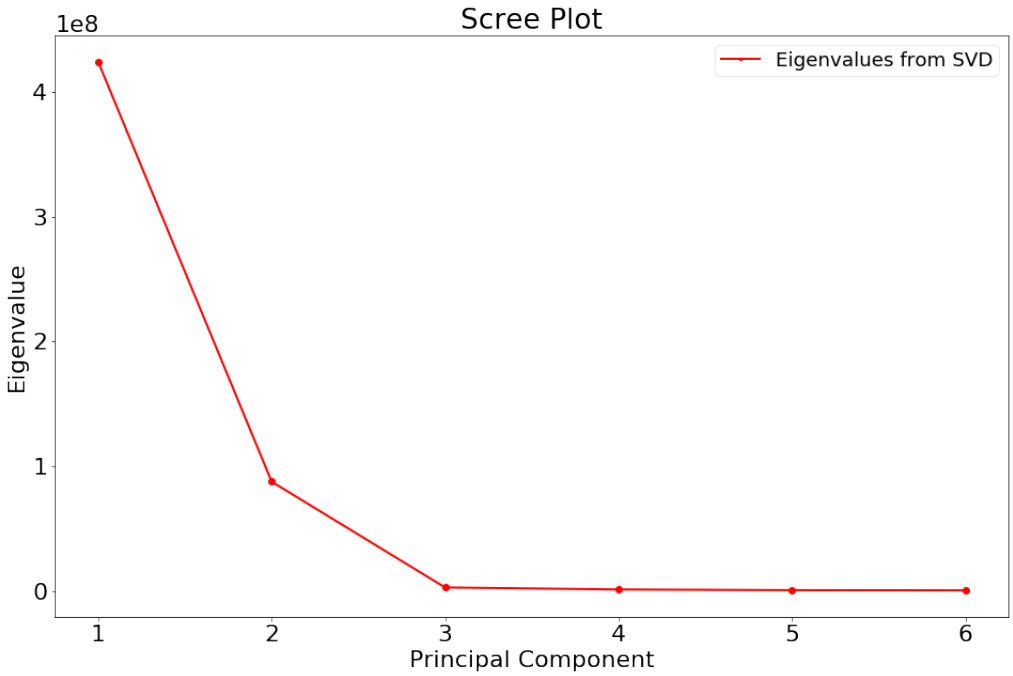


Figure 3.12: Scree plot of first 6 principal components and their respective variance

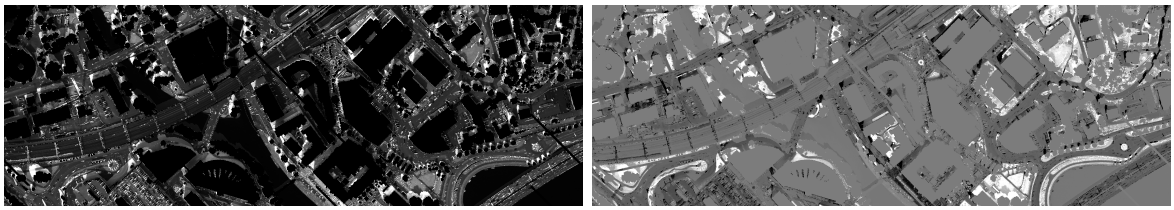


Figure 3.13: Score image of left: PC1 and right: PC2

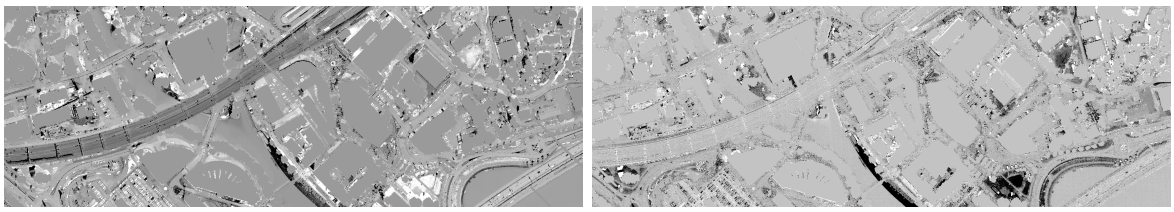


Figure 3.14: Score image of left: PC3 and right: PC4

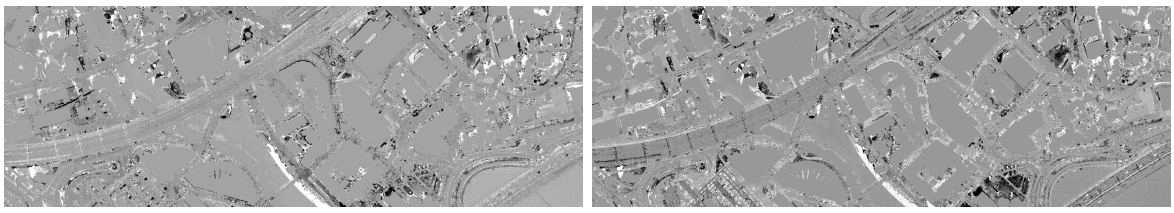


Figure 3.15: Score image of left: PC5 and right: PC6

Similarly, from image 3.13 - 3.15, the first three components contain most of the information. Most of the details and objects are visible in the score image of the first principal component (left figure 3.13). Vegetation is well separated and visible in the second principal component score image (right figure 3.13) and train tracks and roads are seen in the third principal component score image (left figure 3.14). The remaining fourth , fifth, and sixth main components consist of some signal which can be seen in

the figure 3.15. However, the decision was made to include only the first three principal components as classification features. PCA is used separately for VNIR, SWIR and stacked (VNIR+SWIR) images. The number and variation of the principal components is different for these images.

3.3.2 Normalized Difference Vegetation Index(NDVI)

NDVI algorithm is implemented in Python where the red band is subtracted from near-infrared and is divided by the sum of the red and near-infrared bands as shown in the equation 2.2. In this study, band 76 is chosen as a red band and band 106 is used as a near-infrared band for the NDVI evaluation. The output of NDVI is used as a feature for the classification procedure. The resulting image after the NDVI operation is shown in figure 3.16

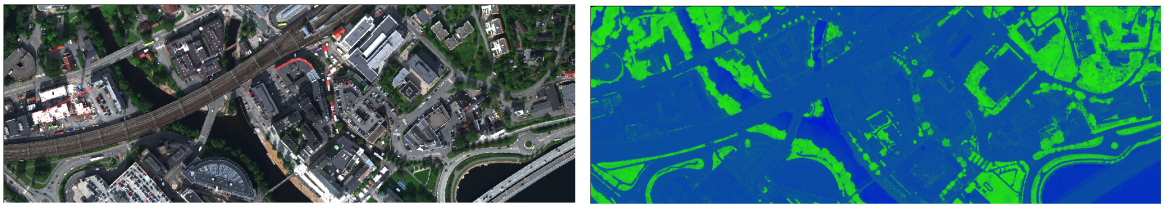


Figure 3.16: Left: Original image, Right: NDVI image showing vegetation (green color represents vegetation)

3.3.3 Normalized Digital Surface Model

In hyperspectral images, different materials have identical spectral signatures, which make it difficult to differentiate between these materials. The nDSM derived from LiDAR data can be used to identify these materials. Objects such as roads and pavements, with similar spectra but different elevations, can be identified using nDSM data.

In this project, the main objective is to extract the edge of the roads and for this, the location of the curbs or road boundaries is crucial which can be identified using fused version of LiDAR and hyperspectral data. So, nDSM obtained from LiDAR data is included as one of the feature for supervised classification. The method is shown in figure 3.17.

3.3.4 LiDAR intensity

LiDAR intensity is generated from the intensity values of the first return of the LiDAR data in LAStools. The intensity values is different for different materials so this is used as one of the features for classification of the target materials.

3.4 Classification based on Machine Learning Algorithms

In this analysis, the hyperspectral image classification is based on a supervised machine learning algorithm. The data values are first normalized to a standard scale without distorting the variations in the ranges of their values. The result after normalization is

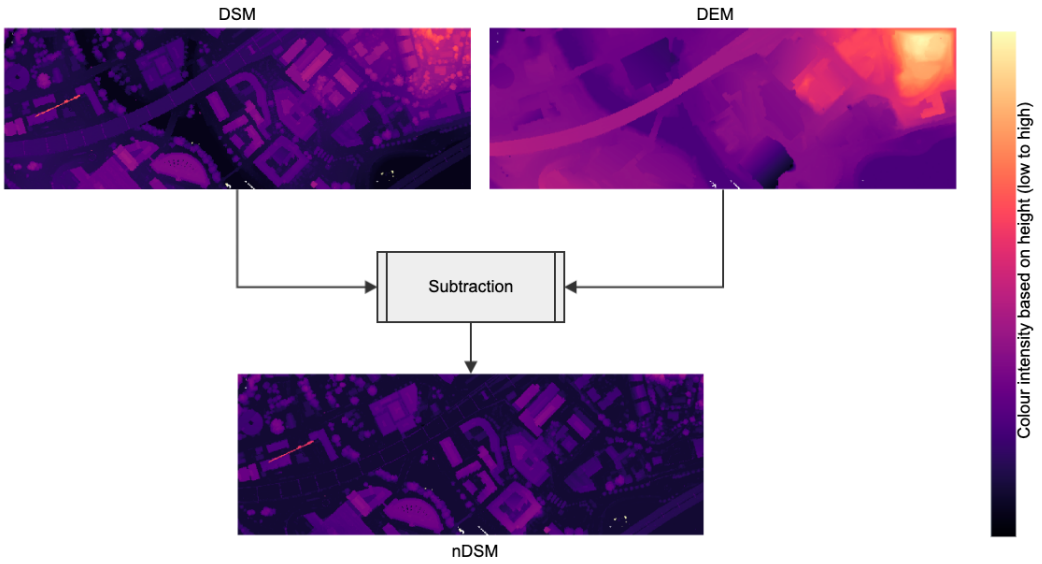


Figure 3.17: nDSM generation by subtracting DSM from DEM derived from LiDAR point cloud.

a dataset with zero mean and unit standard deviation [Jaitley, 2018]. Mathematically, the normalization is evaluated as follows:

$$\tilde{X}[:,i] = \frac{X[:,i] - \mu_i}{\sigma_i}$$

$$\mu_i = \frac{1}{N} * \sum_{k=1}^{N} X[k,i]$$

$$\sigma_i = \sqrt{\frac{1}{N-1} * \sum_{k=1}^{N} (X[k,i] - \mu_i)^2}$$

where μ is Mean, σ is Standard Deviation, i is the number of column in the dataset, N is total number of rows in the dataset, X is the original data value, $\tilde{X}$ is the normalized data value

After the dataset is normalized, it is divided into training and test dataset using train-test-split library of Scikit-Learn as shown in figure 3.18. The training set contains known target value and it is used to train the model and the test set is used to test the model's prediction on this unknown test dataset.

3.4.1 Classification based on Support Vector Machine

SVM is a method of classification based on the statistical information of hyperspectral images [Cortes and Vapnik, 1995]. It locates the optimal hyperplane between classes to split them in a new high-dimensional feature space, taking into account only the training samples that lie at the edge of the class distribution known as support vectors. The CSV file of the hyperspectral image and corresponding class label are generated in ENVI. The CSV files are then read in python for using pandas and further processing is done.

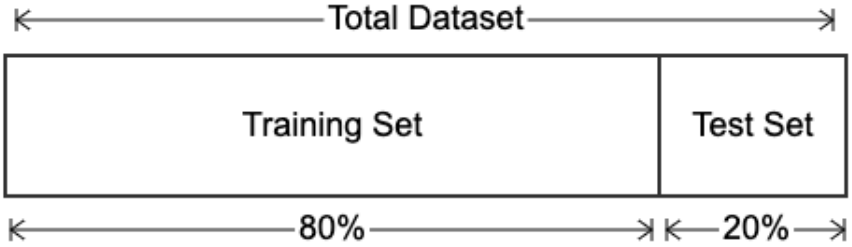


Figure 3.18: Splitting dataset into training (80%) and test (20%) set

Hyperparameter Optimization

A machine learning model is defined as a mathematical model, with a number of parameters to learn from the data. But there are certain parameters called hyperparameters that can not be explicitly learned and must be identified by manually on the basis of some hit and trial process before training begins [GeeksforGeeks, 2018]. A model can have several hyperparameters and the main task in hyperparameter optimization is to find the combination of the parameter which results in high performance of the model [GeeksforGeeks, 2018]. SVM has several hyperparameters, including regularization (C), kernel, and gamma. In python, tuning of hyperparameters is done using Scikit-Learn GridSearchCV library. In GridSearchCV, the model uses many values for each hyperparameter to train it and find the combination of the hyperparameter values which yields the best prediction. For SVM model, best performance is achieved with C=1000, gamma=1 and rbf kernel. The validation curves for SVM using different values of C and gamma (γ) is shown in figure 3.19.

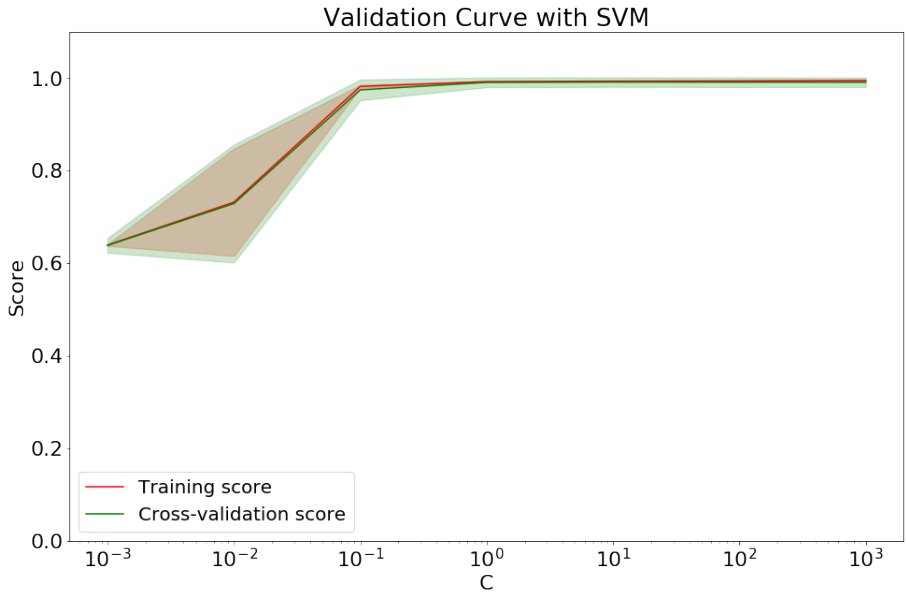


Figure 3.19: Training and validation score of SVM for different values of hyperparameter C. In the plot, it can seen that performance increases with the increase in regularization (C) value

Training

SVM model training is conducted using datasets from southern Sandvika area. The model is trained with both the radiance data and the reflectance dataset separately. Also

it is trained separately for hyperspectral data, LiDAR data, and fused (hyperspectral + LiDAR) data. A total of 13 SVM models were generated, 6 from SWIR, VNIR, and stacked radiance data, 6 from SWIR, VNIR, and stacked reflectance data, and one from LiDAR. The first 3 principal components, NDVI, LiDAR intensity, and nDSM data were used as training features for radiance and reflectance VNIR images. Similarly, the first 6 principal components, LiDAR intensity, and nDSM data were used as training features for radiance and reflectance SWIR images. Likewise, the first 7 principal components, NDVI, LiDAR intensity, and nDSM data were used as training features for the radiance and reflectance stacked image. The first 7 principal components were selected for stacked image because they included about 99.9% of the image total variance.

Validation and Accuracy Assessment

The SVM model performance is assessed using validation data from the northern part of the Sandvika area. The validation dataset shows how well the training has worked. If the performance of the training differs significantly from that of the test, this suggests overfitting or under-fitting. The accuracy of the model is assessed and visualised using confusion matrix, classification report, kappa, and F1 score.

3.4.2 Classification based on Random Forest

RF is a collection of decision trees where classification is based on majority voting of the trees. The script of the classification using RF is shown in appendix A.

Hyperparameter Optimization

Similar to SVM, RF has several hyperparameters such as number of trees, depth of the tree, number of feature and criterion. GridSearchCV function of Scikit-learn library is used to tune the hyperparameter of RF. The best performance is achieved with criterior = "entropy", max-depth = 70, max-features="auto" and n-estimator=80. The validation curve for RF using different values of n-estimator is shown in figure 3.20.

Also cross-validation is also run on the dataset and learning curve is plotted as shown in figures 3.20 and 3.21 respectively. Learning curve shows the performance of model as the training data increases and check whether the model is suffering from bias or variance problem. It determines whether there is need to increase the training dataset or to work on the model. In the plot 3.21, it can be inferred that the RF model is neither suffering from bias nor variance problem.

Training

Similar to SVM, RF model is trained using datasets from southern Sandvika area. The model is trained with both the radiance data and the reflectance dataset separately. Also it is trained separately for hyperspectral data, LiDAR data, and fused (hyperspectral+Lidar) data. A total of 13 RF models were generated, 6 from SWIR, VNIR, and stacked radiance data, 6 from SWIR, VNIR, and stacked reflectance data, and one from LiDAR. The first 3 principal components, NDVI, LiDAR intensity, and nDSM data were used as training features for radiance and reflectance VNIR images. Similarly, the first 6 principal components, LiDAR intensity, and nDSM data were used as

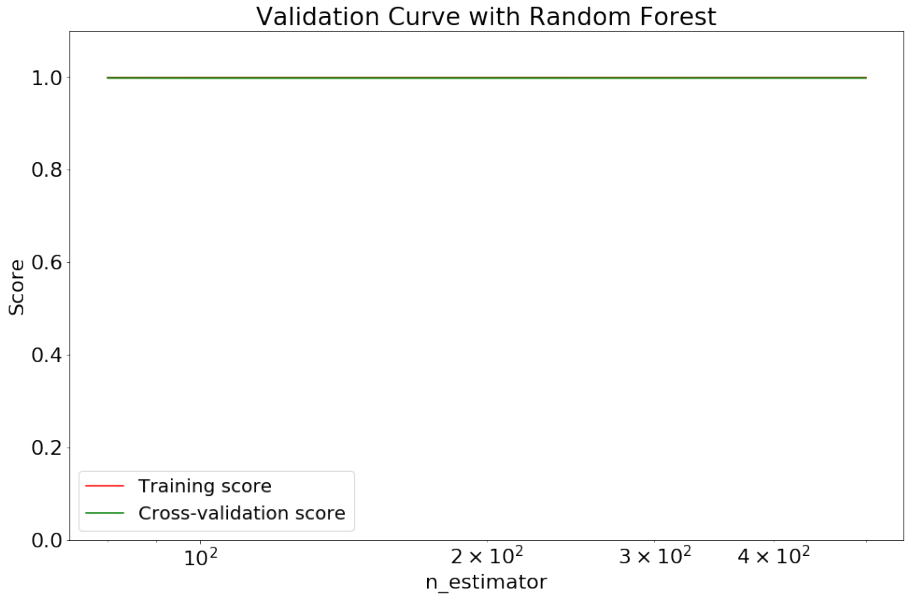


Figure 3.20: Training and validation score of SVM for different values of hyperparameter n-estimator. In the plot, it can seen that performance remains same with the increase in n-estimator value

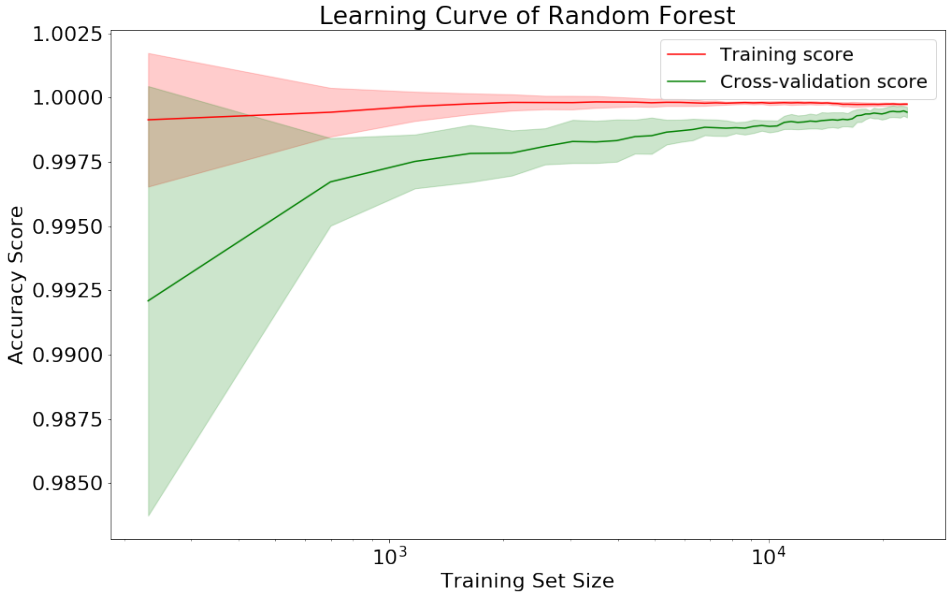


Figure 3.21: Training and cross-validation score of RF for different training set size.

training features for radiance and reflectance SWIR images. Likewise, the first 7 main components, NDVI, LiDAR intensity, and nDSM data were used as training features for the radiance and reflectance stacked image.

Validation and Accuracy Assessment

The RF model performance is assessed using validation data from the northern part of the Sandvika area. The validation dataset shows how well the training has worked. If the performance of the training differs significantly from that of the test, this suggests overfitting or under-fitting. The accuracy of the model is assessed and visualised using confusion matrix, classification report, kappa, and F1 score.

3.5 Classification based on Deep Learning

Deep Learning (DL) models hierarchically learn the features of input data with a very deep neural network, typically deeper than three layers. The network is configured layer-wise first by unsupervised training, and then balanced in a supervised manner. High-level features (objects and big shapes in the image) are learned from low-level features (points and lines learned via the filter of convolution). DL models contribute to gradually more abstract and complicated features at higher levels and these complex features are immutable to the changes the dataset has undergone [Gogineni and Chaturvedi, 2019].

Unlike a machine learning algorithm, DL models can find complex features in the hyperspectral images automatically, avoiding complicated and time-consuming feature engineering processes [Yang et al., 2018]. CNN is one of the widely used DL architecture in the field of image segmentation which considers spatial correlation among pixels. Some of the DL architecture used in this study for hyperspectral image segmentation are discussed in later sections.

3.5.1 CNN Models

In CNN, hyperspectral image is passed through a series of convolution layers with filters, pooling, fully connected layers, and then apply softmax function to classify the objects in image with probabilistic values between 0 and 1. In this study, two different CNN architectures are used as shown in figure 3.22. The two CNN models differ in the number of convolution layers, filer size, and number of dense layers.

In Figure 3.22, the model contains $10 \times 9 \times 9$ input shape. The convolution layer with 30 filters and 3×3 filter kernel sizes is applied to each image window (window size is equal to the input shape). Each filter transforms a part of the image which is defined by kernel size using kernel filter. Additional convolution layers of different filter sizes and kernel sizes are applied to the output of the previous layer. The output of the final convolution layer is three-dimensional, so this output is passed through a flattened layer to convert it into a single dimensional array. The flatten layer output is forwarded to a dense layer with some neuron units and ReLU activation function, which sets all activation values below zero to zero.

Dropout is used to drop a portion of the network randomly, which causes the model to learn features in a distributed manner. This technique also enhances generalization and minimizes overfitting. The dense output layer has neuron units equal to the number of classes and softmax activation functions. This output layer outputs the probability distribution for each class and the sum of those values is equal to 1. The same structure applies to the second model.

After the layers are added to the model, loss function, score function, and optimizer algorithm are defined. Loss function defines how poorly the model performs on the labelled dataset by evaluating the error rate between actual labels and predicted labels. Since, the dataset consists of more than two classes (categorical classification) so "categorical_crossentropy" loss function is used.

The proper choice of optimization algorithm is very important in CNN. Stochastic Gradient Decent (SGD) optimization algorithm is used in this model. SGD consist of differ-

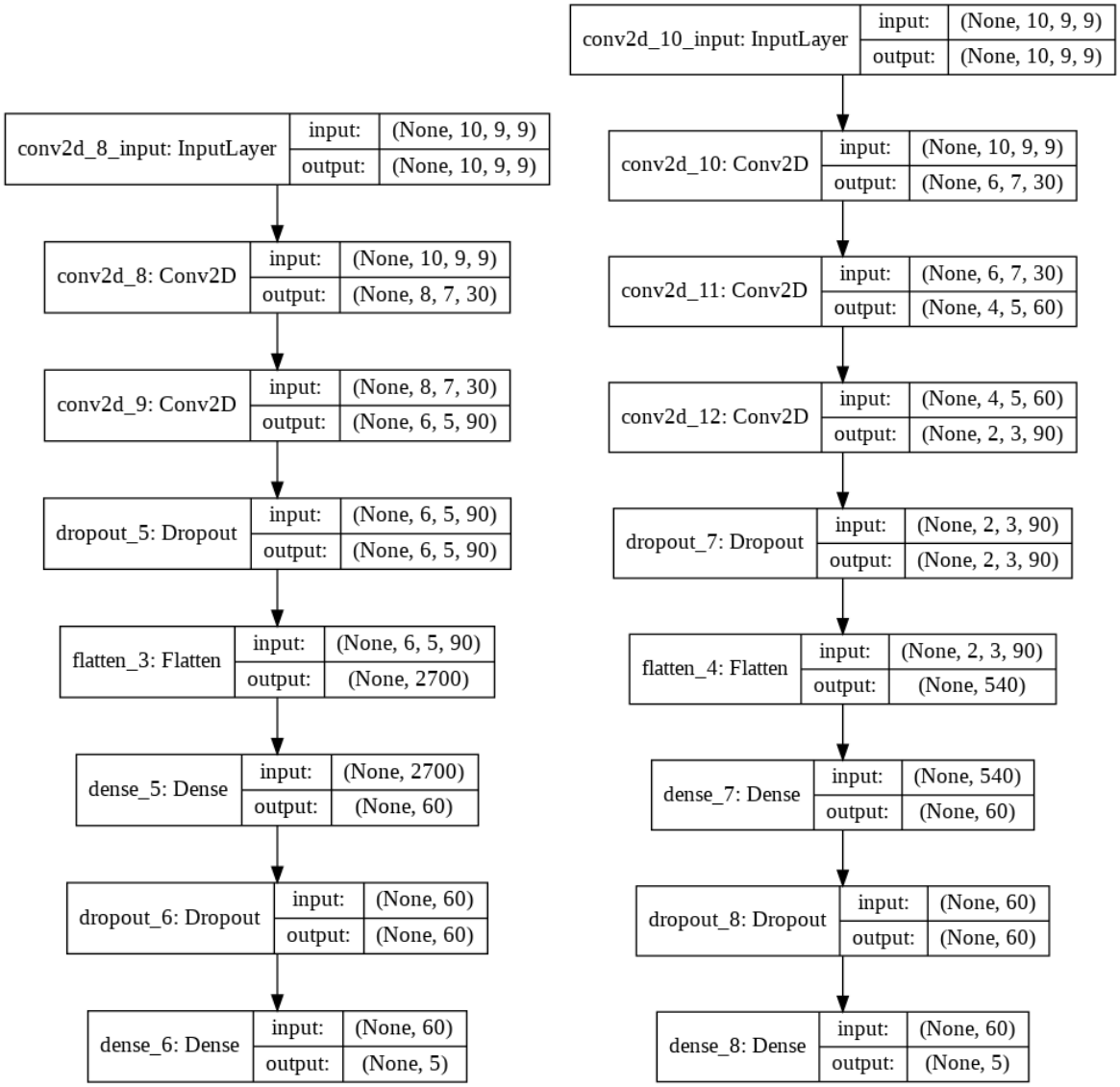


Figure 3.22: Left: CNN model 1 with two convolution layer, Right: CNN model 2 with 3 convolution layer. The plot shows the input and output shape in each layer.

ent hyperparameters like learning rate, decay, momentum, and nesterov. These hyperparameter are optimized using cross-validation until the best performance is achieved. Similarly accuracy metric is used as a score function. The script of the classification using CNN is shown in appendix B.

Training and validation

Before training CNN models, hyperspectral image data are processed and converted to numpy array (.npy) format. The hyperspectral image data are split into patches (windows) of 9 pixels size. The initial form of the dataset is three-dimensional ($length \times height \times bands$) but after splitting the image into small patches, its shape changes to four-dimensional ($number\,of\,patches \times length \times height \times bands$). Then labelled dataset is divided into two sets of train and test, where training comprises of 70% and test set comprises of 30% of the dataset. In the training set, the weak classes are oversampled to balance the dataset. Since the CNN training requires a large amount of data thus, data augmentation is applied to the training set. The data augmentation generates a new

version of pre-existing image patches by rotating, zooming, and flipping them.

The architecture of the CNN model is determined after processing the dataset and the model is modified iteratively after each training based on the model's loss and accuracy plot. The training of the models is done in Keras. The shape of training data is $number\ of\ patches \times length \times height \times bands$. Both augmented and original training data are used to train the CNN. Since training a DL network requires tons of data, it requires more computer memory to load and use all data at once to train a DL network. Thus batch processing is used to process datasets in batch instead of loading all data at once. Computer memory can be used more effectively by batch processing. In this study, all the training was performed on a laptop with 8 GB of memory. Also during training, the model is saved at its best training epoch to avoid overfitting.

3.6 Road Edge Detection

After the classification map is obtained using a machine and a deep learning algorithm, road edges are extracted using Canny Edge Detection. Canny Edge Detection is implemented using python skimage package. It involves the following process:

Noise Reduction

The noise in the image is reduced by smoothing it with the Gaussian Blur filter. It is achieved using the Gaussian Kernel Image Convolution technique. The size of the kernel depends on it and the small kernel means the blur is less visible.

Gradient Calculation

The edge strength and direction are calculated by measuring the gradient of the image using edge detectors such as Sobel filters. After this, the image edges are obtained.

Non-Maximum Suppression

After gradient evaluation, the obtained edges have a mixed shape, so that a non-maximum suppression is carried out in order to thin out the edges. In this, the algorithm goes through all the points in the gradient matrix and finds the pixels with the highest value in the direction of the edge.

Double Threshold

It mainly involves the determination of three types of pixels: strong, weak and non-relevant. High threshold is used to evaluate strong pixels and low threshold is used to classify non-relevant pixels. All pixels with an strength between high and low threshold are weak pixels. Hysteresis is used to transform weak pixels to strong pixels based on the availability of a strong pixel around it.

Generally, the road edges determined using Canny Edge Detection are thin. Multiple dilation is applied on the extracted edges to make it more thick and visible (see figure 3.23).

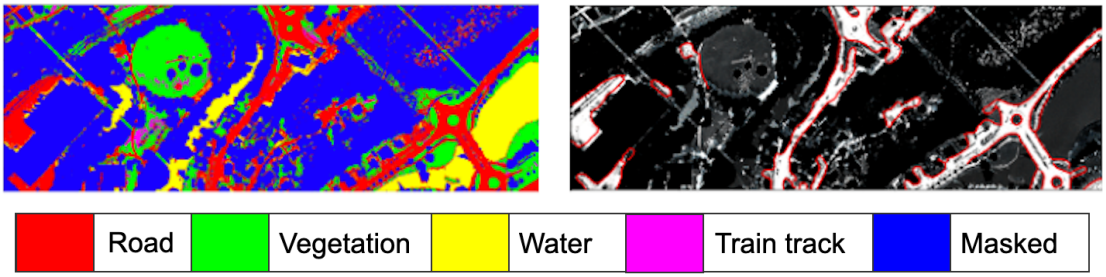


Figure 3.23: Left: Classification map obtained using RF Algorithm. Right: Road Extraction using Canny Edge Detection and Dilation.

Results and Discussion

4.1 Data Description

Many experiments were carried out and different classification models were engineered. The performance of these models were evaluated for radiance, reflectance, LiDAR, and pixel level fused (hyperspectral and LiDAR) datasets. Finally, comparison of the performance of each model was done for each type of dataset. This chapter includes the results of data preprocessing, model architecture, model performance, and comparison.

The city of Sandvika, Norway was selected as a case study. The southern part is used as training area and northern part of the city is used as testing area, see figure 3.8. The area consists of artificial construction such as buildings, roads, and railways.

4.2 Experimental Results

The hyperspectral image and LiDAR data were classified using SVM, RF, and CNN model which were trained using training dataset to achieve maximum accuracy. The trained model was then used to classify the test data. The classification was done in radiance, reflectance of hyperspectral data and LiDAR data separately and classification map was obtained. The performance were evaluated using performance metrics such as Overall Accuracy (OA), F1-score, cohen-kappa score and confusion matrix. The radiance and reflectance hyperspectral images were classified and compared. The one that performed better was fused with the LiDAR data and again the classification was done and the accuracy was compared with the individual data. Also, Canny Edge detection algorithm was used to extract the road from a classified image map. Similarly, the effectiveness of SVM, RF, and CNN in the classification of hyperspectral images was also compared.

4.2.1 Effects of PCA on classification

The first experiment was conducted to access the difference in overall accuracy and computation time for the classification of hyperspectral data using original high-dimensional data and reduced PCA data. The first 3 principal components are used. Performance based on different metrics and computational time of the RF and SVM on the original hyperspectral image and dimensional reduced dataset using PCA is evaluated and compared in table 4.1.

Table 4.1: Comparison of classification accuracy, training and testing time of non-PCA and PCA modified hyperspectral image based on SVM and Random Forest

	RF		SVM	
	Without PCA	With PCA	Without PCA	PCA
Accuracy	0.91	0.91	0.85	0.89
F1-score	0.84	0.83	0.75	0.81
Kappa Score	0.90	0.89	0.85	0.84
Training Time (seconds)	0.63	0.44	0.95	0.10
Testing Time (seconds)	5.37	2.80	73.49	3.86

From table 4.1, it can be observed that training and testing time decreases by using PCA modified hyperspectral data. This is because PCA reduces high dimensional hyperspectral data to a few principal components and less data size means less computation time. In the case of RF, the performance is almost the same for both data, but the precision of the classification is higher for the PCA transformed data in the case of SVM.

4.2.2 Classification based on SVM

The classification was done in the radiance, reflectance, and LiDAR data. Both radiance and reflectance data consist of SWIR and VNIR images which are classified using SVM. First, the classification results of all the types of data using SVM, RF and CNN were summarised and then the best results from each dataset were extracted and compared with each other. Similarly, best results based on classification models were also compared.

The available training dataset were divided into 80% training and 20% validation data. Before splitting the data, they are normalised so that their mean is 0 and variance is 1. Determination of regularization (C) and gamma (γ) parameters is crucial while training SVM model. Values of these hyperparameters are determined using cross-validation. For the reflectance dataset, the best values for the parameters C were 10 and 1000 for the VNIR and SWIR images , respectively. γ values for VNIR and SWIR were found to be 0.1 and 1, respectively. The influence of different parameter values on overall accuracy and computation time of classification is shown in table 4.2. These values were evaluated experimentally based on the available training samples.

Table 4.2: Analysis of overall classification accuracy and the computation time with different SVM parameter values.

	Parameter Value	Overall Accuracy [%]	Computation Time [s]
SVM	C = 100, $\gamma = 1$	97.4	5.68
SVM	C= 10, $\gamma = 0.1$	97.9	5.43

The result of the classification of the test image using SVM based on overall accuracy (OA), F1-score (F1), cohen-kappa score (KS), and computation time (CT) for radiance, reflectance, LiDAR, and radiance fused data is summarised in table 4.3.

As the SWIR image had low spatial resolution, the result of its classification was less than 50%. Therefore, the outcome of SWIR classification was excluded from table 4.3. The F1-Score achieved for individual VNIR radiance (82%) and reflectance data

Table 4.3: Comparison of classification accuracy and computation time (CT) of different data classified using SVM

	Radiance		Reflectance		LiDAR	Radiance fused	
	VNIR	Stacked	VNIR	Stacked		VNIR	Stacked
OA	0.84	0.51	0.91	0.58	0.65	0.84	0.77
F1	0.82	0.45	0.77	0.45	0.47	0.82	0.54
KS	0.78	0.35	0.88	0.60	0.54	0.77	0.69
CT	22.41	80.51	5.70	5.32	9.50	27.71	72.30

(77%) with the SVM classifier was better than those achieved with their respective Stacked (SWIR + VNIR) data (below 55%) because the stacked images were obtained by combining VNIR and SWIR images with low resolution. This experiment shows that the classification accuracy is high when the image has better resolution. Similarly, the result of LiDAR data classification was low compared to the result of Fused (Hyperspectral Image + LiDAR) data. The OA of Stacked data increased from 58 % to 77 % when LiDAR data was used in combination with it. This shows that a better classification map is obtained when elevation-based information is used in combination with hyperspectral images.

The output of the classification shown in figure 4.1 illustrates four land cover classes. Through visual analysis of classification results shown in VNIR radiance images 4.1B and 4.1D, it can be seen that SVM is unable to differentiate between water and road classes so some of the water pixels is labeled as road class. But in case of VNIR reflectance images 4.1F and 4.1J, there is no misclassification of water pixels. This shows that radiance image, not atmospherically corrected, has the effect of water absorption around 900 nm wavelength range, which leads to some pixels of water being classified as other classes.

4.2.3 Classification based on Random Forest

In this experiment, the classification was performed on radiance, reflectance, LiDAR, and a combination of LiDAR and hyperspectral data (fused dataset) using RF. Table 4.5 shows the OA score, the F1 score, the cohen-kappa score, and the computation time for the classification of the different data.

Table 4.4: Analysis of overall classification accuracy and the computation time with different Random Forest parameter values. Here 'n_estimator' is the number of tress in the forest, 'criterion' is the function to measure the quality of split, and 'max_depth' is the maximum depth of the tree

	Parameter Value	Accuracy [%]	Computation Time [s]
Random Forest	n_estimator = 50, criterion = 'entropy', max_depth = 82	92	4.52
Random Forest	n_estimator = 70, criterion = 'gini', max_depth = 90	93	6.23
Random Forest	n_estimator = 200, criterion = 'gini', max_depth = 150	93	17.7

During the classification based on the RF, it is important to determine the values of the parameters, such as the number of trees, the feature to assess the quality of the tree

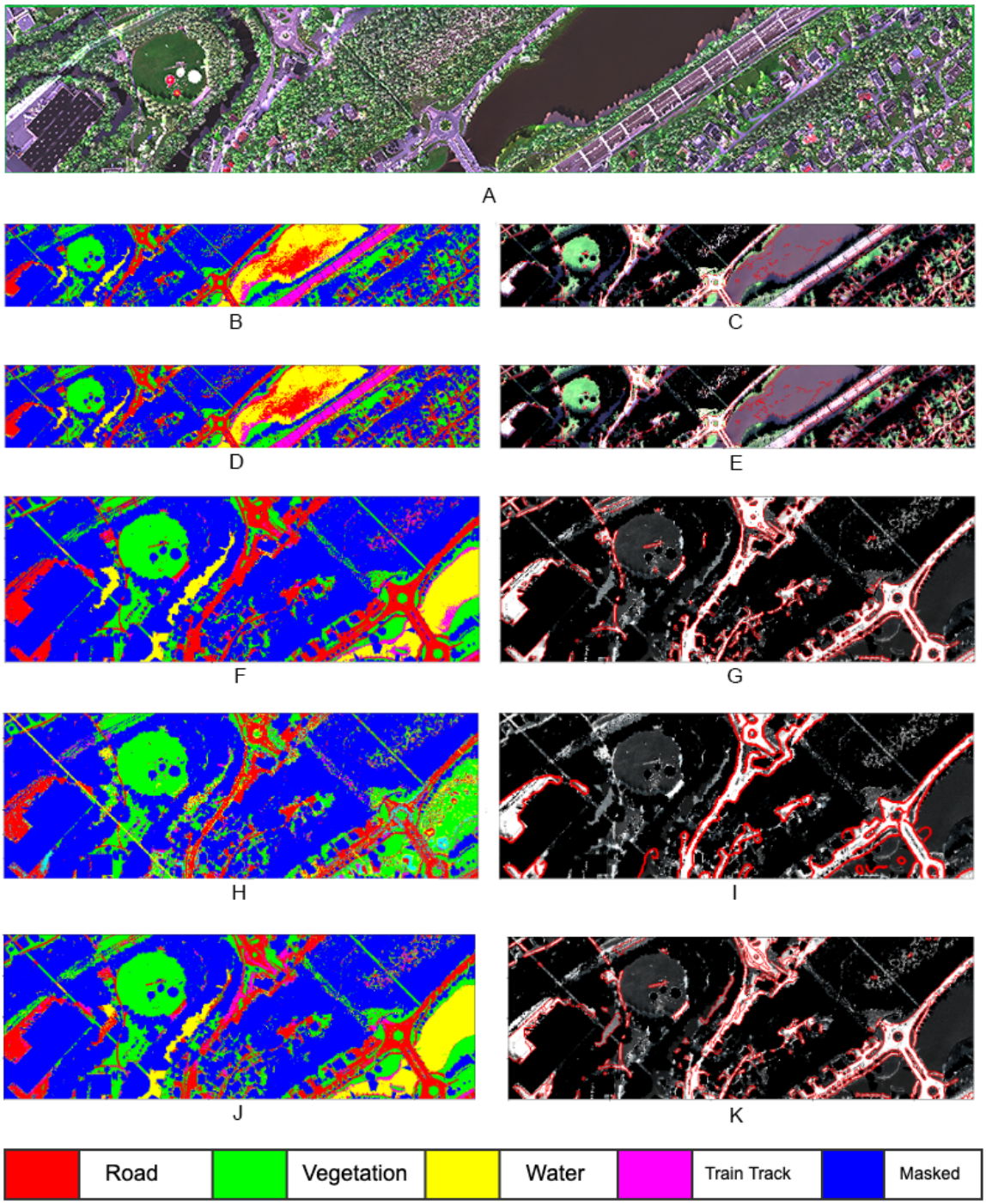


Figure 4.1: Classification results using SVM. Top: Image A is the RGB image. Left: Image B and D are classification map of VNIR and fused radiance data, respectively. Image F, and J are classification map of VNIR and fused reflectance data, respectively, and Image H is classification map of Lidar Data. Right: Image C, E, G, I and K shows the respective road edges

split and the maximum depth of the tree. Table 4.4 shows a comparison of classification accuracy and computation time with the different values of these parameters. The best result is achieved by using 70 trees with a maximum depth of 90 and the 'gini' criterion.

From the table 4.5, it can be seen that excellent OA is achieved for fused (VNIR + LiDAR) data (93%). The classification result of SWIR data is also increased from 52%

Table 4.5: Comparison of classification accuracy and computation time of different type of data classified using Random Forest

	Radiance		Reflectance		LiDAR	Reflectance Fused	
	VNIR	SWIR	VNIR	SWIR		VNIR	SWIR
OA	92	52	93	90	71	93	90
F1	88	28	77	73	60	78	73
KS	88	35	89	85	61	90	85
CT	8.53	1.26	4.53	53.70	8.7	4.76	4.57

to 85% when LiDAR data is combined with the original SWIR data. This shows that LiDAR elevation and intensity data improve the accuracy of classification when used in conjunction with hyperspectral data.

The classification output shown in figure 4.2 shows four land cover classes. Similar to SVM, the results of the classification shown in VNIR radiance images 4.2B and 4.2D illustrate that RF could not distinguish between water and road classes so that some water pixels are marked as road class. Yet there is no misclassification of water pixels in the case of VNIR reflectance images 4.2F and 4.2J. This shows that the radiance image, which has not been atmospherically corrected, has the effect of water absorption around 900 nm wavelength range, which leads to misclassification of some pixels of water as other classes. Similarly, for the fused (VNIR + LiDAR) image shown in figure 4.2M, road extraction is better than for the results of the individual hyperspectral and LiDAR datasets shown in figure 4.2C, 4.2E, 4.2G, 4.2I which have many road misclassifications. The number of pixels well classified and labeled can be seen in the confusing matrix shown in figure

4.2.4 Classification based on CNN

The classification was performed on radiance, reflectance, LiDAR, and combination of LiDAR and hyperspectral data using Convolution Neural Network. Overall Accuracy (OA), F1-score and kappa-score were used to assess the classification performance of the model.

In the experiment, PCA was applied on the training data and reduce the spectral dimension for VNIR from 186 to 10. It was decided to used first ten principal components because CNN performed low with fewer principal components. Then from the resultant data, $9 \times 9 \times 10$ cubes were extracted to compute spatial-spectral features. The network contained two convolution layers, two dropout layers, and two dense layers. The two convolution layers contained ten 3×3 kernels and thirty 3×3 kernels, respectively. The network consists of one fully connected layer and one classification layer with softmax activation function which outputs the probabilities of classes. The architecture of the CNN is shown in figure 3.22. The CNN was trained over 50 iterations with a batch size of 100 samples. The results of the classification are listed on table 4.6 and the visualisation result is shown in figure 4.3.

The CNN model obtained the best result, with an overall accuracy of 83% for VNIR reflectance image which is 3% higher than the fused radiance result (80%). It is clear from figure 4.3 that the both radiance and reflectance data suffered from misclassification problems, such as train (pink) being misclassified as road (red). Also, in classification

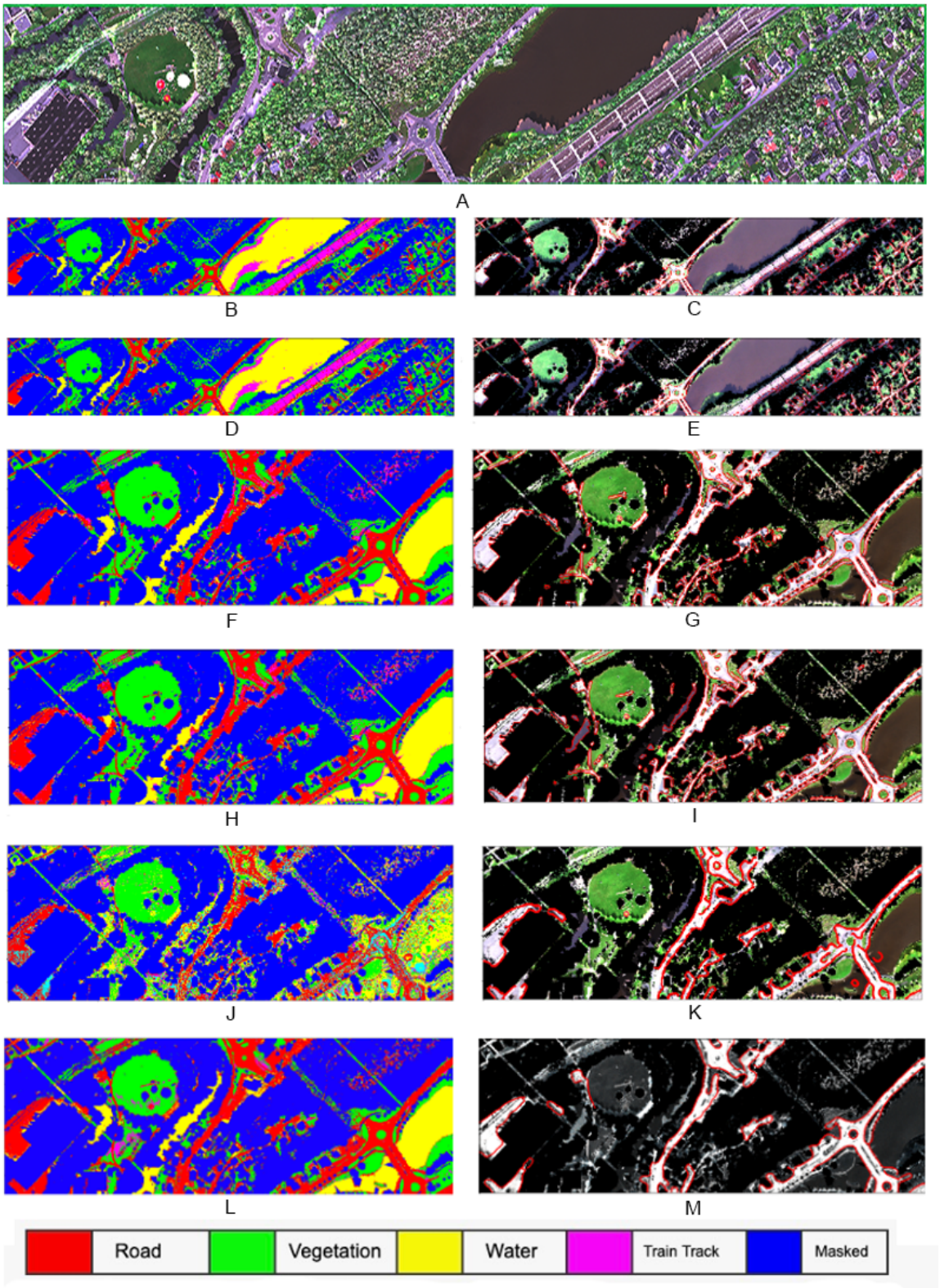


Figure 4.2: Classification results with Random Forest Algorithm. Top: Image A is the RGB image. Left: Image B and D are classification map of VNIR and fused radiance data, respectively. Image F, and H are classification map of VNIR and fused reflectance data, respectively, and Image J and L is classification map of Lidar Data and Stack fused data. Right: Image C, E, G, I, K and M shows the respective road edges.

map of radiance VNIR data, figure 4.3G, there are misclassification of water (yellow) as train track (pink) pixels. However, comparatively, reflectance data achieved less

Table 4.6: Comparison of classification accuracy of different type of data classified using CNN

	Radiance		Reflectance			
	VNIR	VNIR+LiDAR	VNIR	SWIR	Stacked	VNIR + LiDAR
OA	64	80	83	77	81	79
F1	52	60	61	60	57	58
KS	54	73.4	79	73.3	83.8	67.5

misclassification overall.

Influence of different Spatial Size of the data

The CNN model was tested with different size of training samples by extracting spectral-spatial features with different size: $5 \times 5 \times C$, $7 \times 7 \times C$ and $9 \times 9 \times C$, where C is the number of spectral bands. The number of spectral bands is achieved by applying PCA on the original data. CNN model was trained using each type of data and accuracy was evaluated on the validation data. The evaluation result shown in figure 4.4 which demonstrates that the best performance was achieved with a size of $7 \times 7 \times C$ for extracting the spectral-spatial features. This indicates that the target pixel and adjacent neighbor pixels belonging to the same class are covered by 7×7 spatial size. The spatial-spectral feature derived using information in the neighborhood area tends to decrease intra-class variance and thus improves the accuracy of classification. However, the choice of large spatial size can contain noise data so that the accuracy of the classification decreases.

4.3 Classification result comparison

In this section, the results of the classification are compared based on the type of data and method of classification algorithm used. First, the classification results of the radiance and the reflectance were compared and the dataset that performed better was used for classification using other algorithms. Similarly, individual classification result from hyperspectral data, LiDAR data, and fused data are compared and analysed. Lastly, the accuracy result from SVM, RF, and CNN are compared and the best performing model is determined.

4.3.1 Radiance and Reflectance Results

Both the radiance and reflectance hyperspectral images had separate images in SWIR and VNIR wavelength range. So the comparison was performed based on classification results of VNIR and SWIR data. Tables 4.3, 4.5, and 4.6 shows the comparison of classification result of VNIR and SWIR radiance and reflectance data using SVM, RF, and CNN, respectively. It can be seen that the classification result of radiance data is better as compared to reflectance data. The confusion matrix of the classification of radiance and reflectance data based on RF is shown in figure 4.5 and 4.6, respectively.

Although the OA of the classification of the reflectance data is higher than that of the radiance data, the F1-score is higher for the radiance data. It can also be seen from the confusion matrix 4.5 that the train tracks are well classified in the radiance data as opposed to the reflectance data. In the case of reflectance results, more than 50% of the

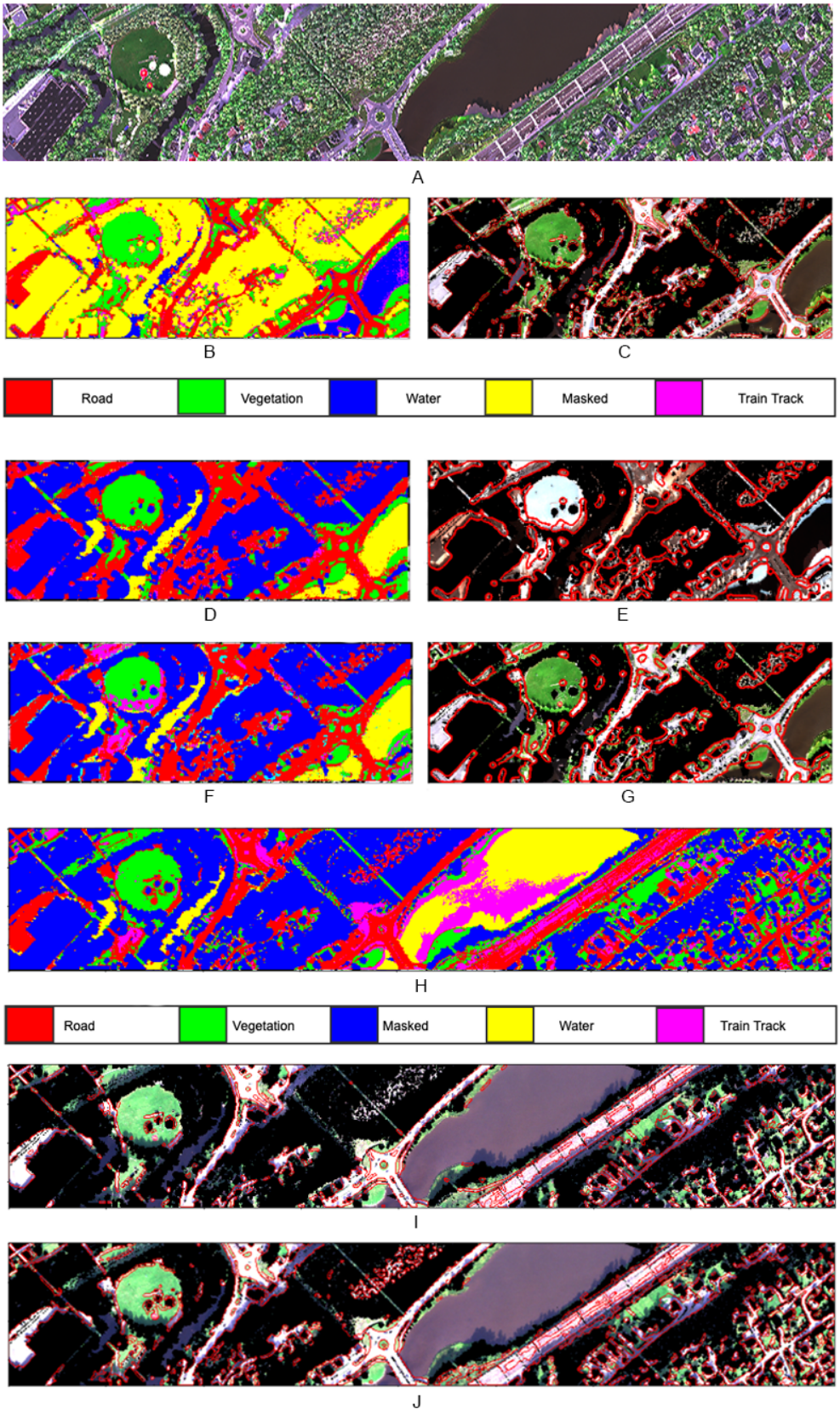


Figure 4.3: Classification results with CNN. Top: Image A is the RGB image. Images B, D and F are classification map of VNIR, SWIR and Fused reflectance data, respectively. Image H is classification map of fused (VNIR+LiDAR) radiance data, respectively, and Images I and J are road map of VNIR and fused radiance data, respectively. Images C, E, and G show the road edges of VNIR, SWIR and Fused reflectance data, respectively.

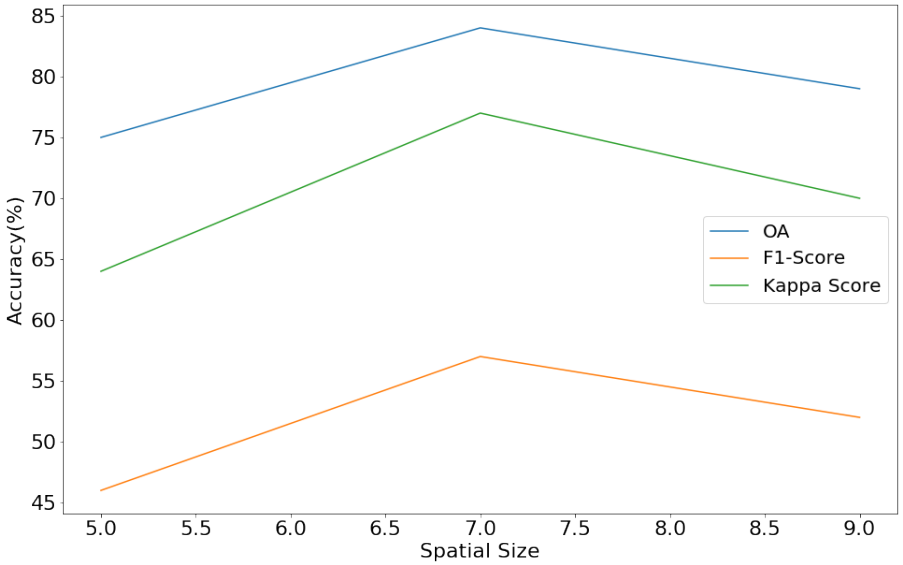


Figure 4.4: Influence of different Spatial size of the data

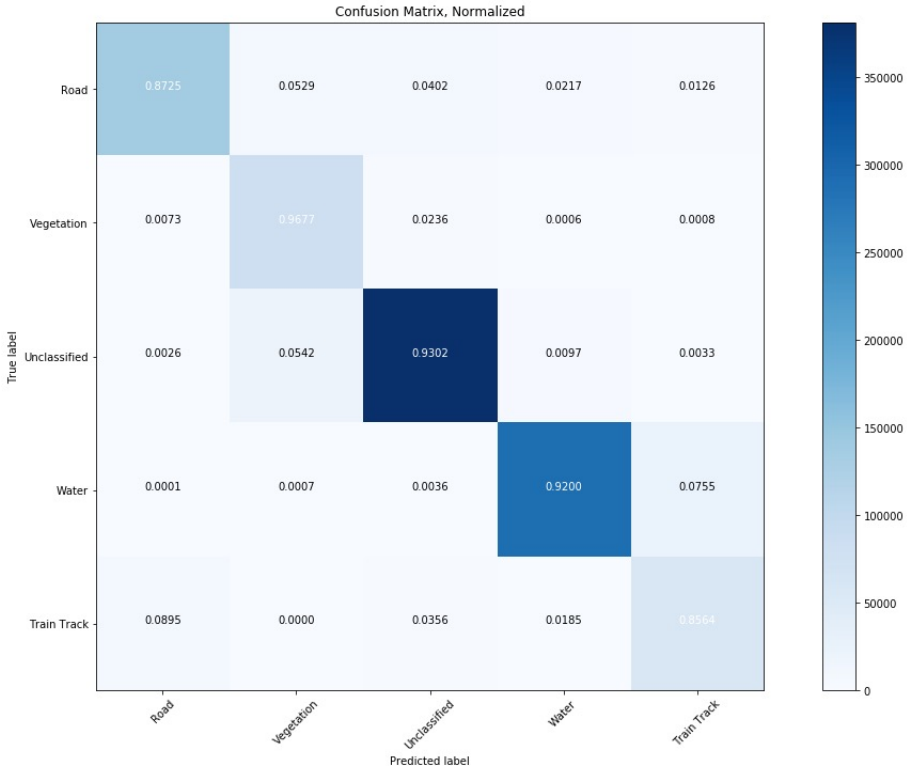


Figure 4.5: Confusion matrix of VNIR radiance data classification based on RF

train track class is classified as a road class. This may be because many of the ground materials had mixed pixels due to the atmospheric correction, due to which the road and train tracks display similar spectral signature in the case of a reflectance image. However, about 8% of the water pixels in the radiance data are classified as a train track class which is not the case in the reflectance data (see figure 4.2C and 4.2E). This is due to the presence of water absorption around 900 - 1000 nm wavelength of the VNIR radiance data. As we are more concerned about the classification of roads and train tracks, so in this case, the results of the classification of radiance data outperformed the reflectance data. [Hoffbeck and Landgrebe, 1994] performed atmospheric correction of

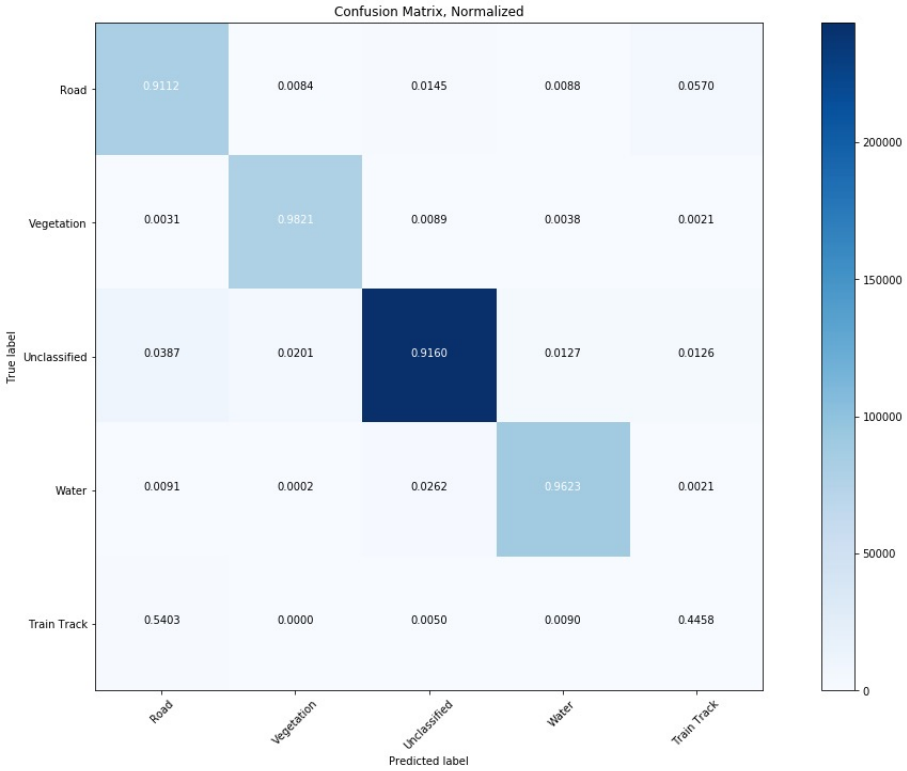


Figure 4.6: Confusion matrix of VNIR reflectance data classification based on RF

the radiance data using ATREM and log residual method and classified the transformed data and the radiance data using Gaussian Maximum Likelihood. The average accuracy of the classification of the reflectance data was found to be lower than that of the radiance data. This finding is identical to the outcomes of our study. Therefore, it can be concluded that the SVM, RF, and CNN classifications are insensitive to changes caused by solar and atmospheric artifacts, and there is no need to attempt to correct them.

4.3.2 Hyperspectral and LiDAR data Results

The hyperspectral data and the LiDAR data were classified separately using SVM, RF, and CNN. The results of the classification is shown in table 4.7.

Table 4.7: Comparison of performance of RF on hyperspectral image (VNIR radiance and reflectance) with LiDAR data.

	Radiance	Reflectance	LiDAR
OA	92	93	71
F1	88	77	60
KS	88	89	61
CT	8.53	4.53	53.70

The classification image of the hyperspectral and the LiDAR data using RF is shown in figure 4.7. Also, their confusion matrix is shown in figure 4.8. In the figure 4.7, different classes are misclassified in the LiDAR classification map. The water pixels are mostly labeled as road and train track class. The LiDAR data consist of two features: elevation and intensity. This indicates that the two features have not been able to generalize each

class well. Also most train track pixels are classified as road class (see right confusion matrix 4.8) because of the similarity of road and train track elevation. However, in the case of a hyperspectral image, the road and train tracks are well separated (see left confusion matrix 4.8) which can be seen in figure 4.7A.

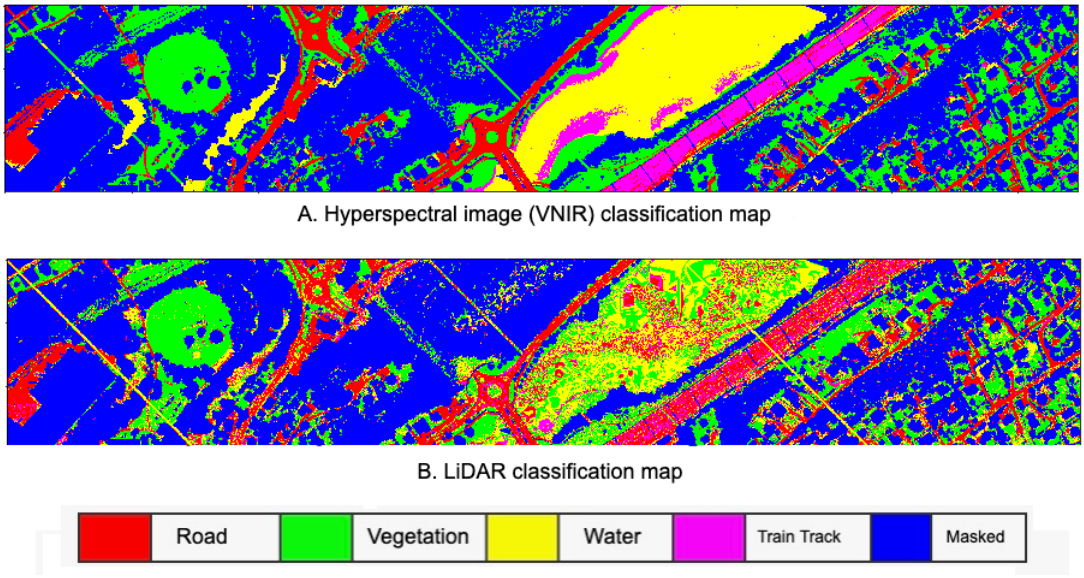


Figure 4.7: Classification map of A. Hyperspectral Image, B. LiDAR data based on RF.

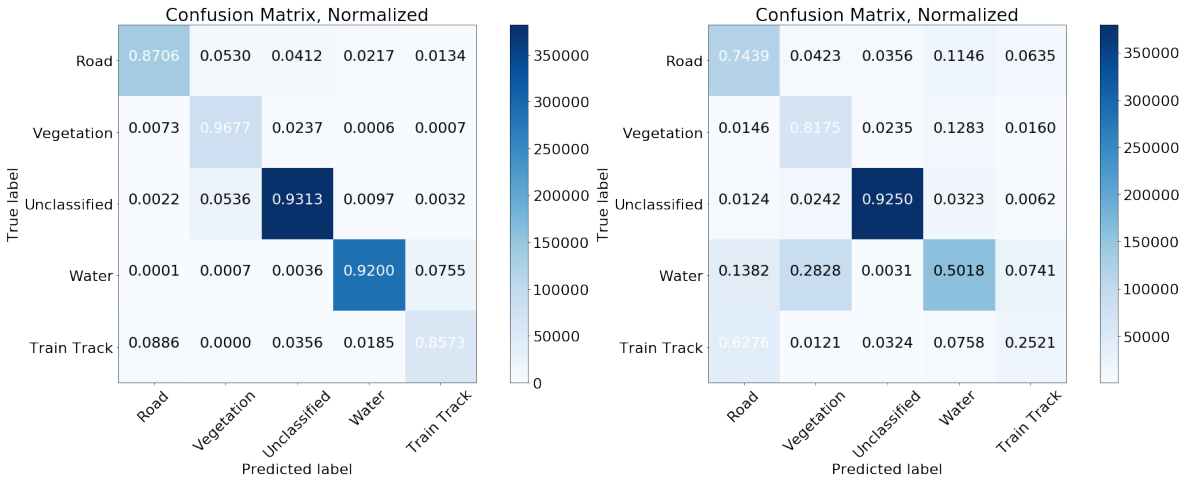


Figure 4.8: Left: Confusion matrix of VNIR radiance data classification based on RF. Right: Confusion matrix of LiDAR data classification based on RF

4.3.3 HSI and Fused data Result

In this section, the hyperspectral image is combined with the LiDAR data and thus the classification result of the fused data is compared with the result of hyperspectral data. Table 4.8 shows the performance comparison of RF in classifying fused and hyperspectral data, separately.

The road edge detection is improved when LiDAR data features are included with the hyperspectral data features (see figure 4.9). There are many misclassifications of road when only hyperspectral data is used as shown in figure 4.9A. This shows that the

Table 4.8: Comparison of classification accuracy and computation time of hyperspectral data and fused data using RF.

	Radiance		Fused		Reflectance		Fused	
	VNIR	SWIR	VNIR	SWIR	VNIR	SWIR	VNIR	SWIR
OA	92	79	92	81	93	90	93	90
F1	88	52	88	59	77	73	79	73
KS	88.5	70	88.5	72	89	85.5	90	85.5
CT	37.93	1.26	47.64	1.33	5.16	0.81	5.82	4.55

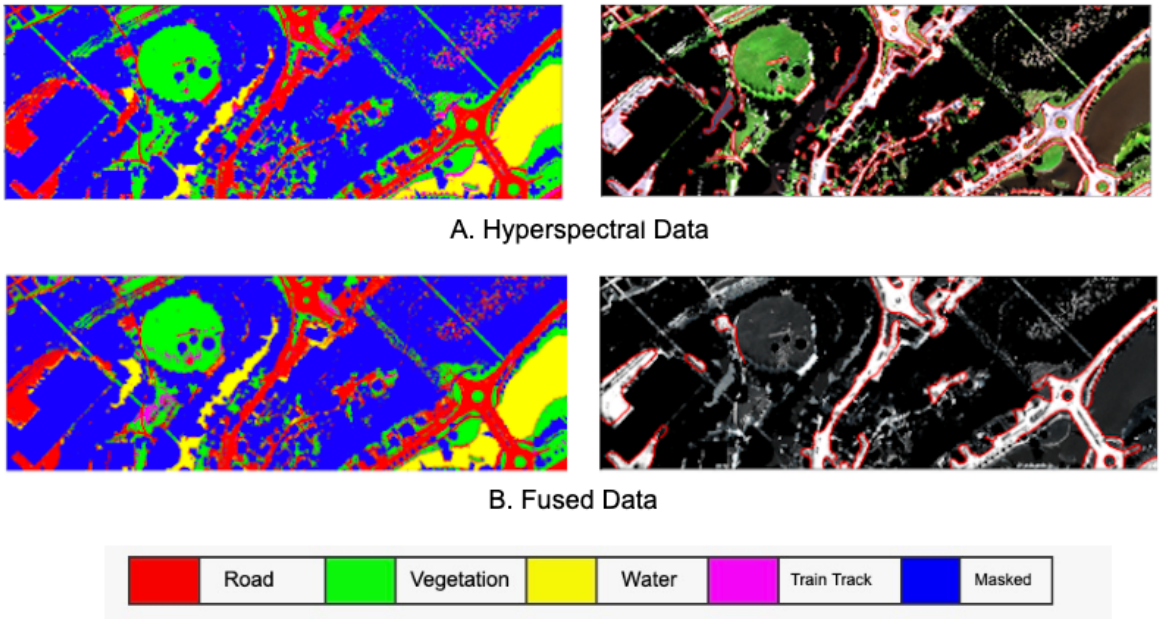


Figure 4.9: Result comparison of hyperspectral and fused data classification map and road map.

elevation feature of the LiDAR was able to separate the roads and other classes based on the height.

From the table 4.8, it can be seen that the OA did not change when LiDAR was merged to the hyperspectral image. However, the F1 score changed from 77% to 79% for VNIR reflectance results. Classification accuracy of the road class improves from 88% to 92% when LiDAR data is combined with hyperspectral data which can be seen in the confusion matrix 4.10. Also in the case of a SWIR radiance image, the F1 score changed from 52% to 59% when LiDAR data is combined with it. This shows that the two features of LiDAR data, elevation and intensity, provide more detail in generalizing classes and increase the accuracy of classification. Both the reflectance data and the reflectance fused data performed poor when classifying the train track class. Most of the train track pixels have been classified as road class. This is because the train track pixels in the reflectance image consist of mixed pixels, including pixels with spectral properties similar to that of the road. However, in case of radiance and radiance fused image, the train track pixels were classified well which can be seen in confusion matrix 4.5.

4.3.4 SVM, RF and CNN Result

In this section, the results of SVM, RF, and CNN are compared for the different data shown in table 4.9. RF outperformed SVM and CNN on the basis of OA, F1-score

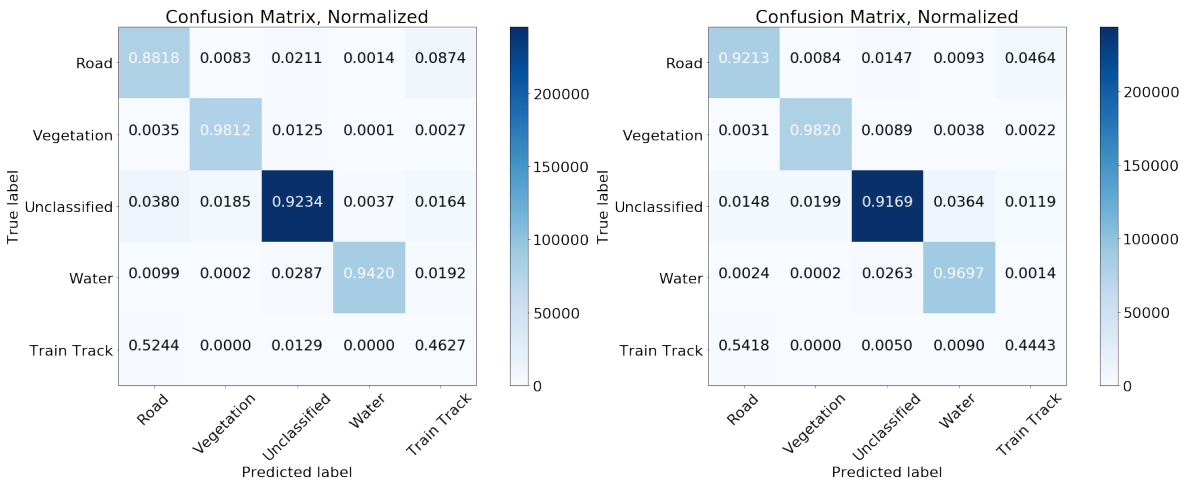


Figure 4.10: Left: Confusion matrix of reflectance hyperspectral data classification based on RF. Right: Confusion matrix of reflectance hyperspectral and LiDAR fused data classification based on RF

and Kappa coefficient. CNN performed the worst of all with only 80% accuracy for the radiance dataset. CNN's poor performance was because it needs a lot of data to perform better than machine learning models. There were two models of CNN and the first one, which was less complex, performed better. This shows that the less complex deep learning model is better suited to finding important data patterns. CNN's accuracy for both the reflectance and radiance data was poor when compared to the RF and SVM result. However, the OA of CNN for radiance data (80%) was better than reflectance data (77%). This is because CNN is dependent on extracting complex features from the data and the reflectance data lost valuable information during the atmospheric correction. This indicates that some of the information of the atmosphere is useful for identifying different classes.

Table 4.9: Comparison of classification accuracy and computation time of Hyperspectral data and fused data using RF.

	SVM		RF		CNN	
	Radiance	Reflectance	Radiance	Reflectance	Radiance	Reflectance
OA	83	92	92	93	80	77
F1	82	77	88	79	60	57
KS	77.60	88.5	88.39	90.4	73.48	67.5
CT	27.71	5.52	47.64	5.82	450.06	321.5

The RF model is capable of generalizing classes well, and so is the SVM model. The kappa coefficient of RF classification of radiance data is 88.4% which is about 11% more than that of SVM and about 14% mode than that of CNN. Figure 4.11 displays the classification map of the fused radiance data classified by SVM, RF and CNN. The classification map obtained by the RF model (see figure 4.11B) is better than the other two classification maps. The train track groups are not well delineated by CNN (see figure 4.11C). It has been classified as a road class. This shows that CNN is unable to distinguish between road and train track pixels.

The findings in [Raczko and Zagajewski, 2017] shows that ANNs outperform both SVM and RF. This is in contrast to the findings presented in this thesis. Similarly, results in [Sabat-Tomala et al., 2020] show that SVM is able to achieve high accuracy in one class while RF does better in classifying another class. This is close to the outcome of

our research where RF is able to generalize well water class.

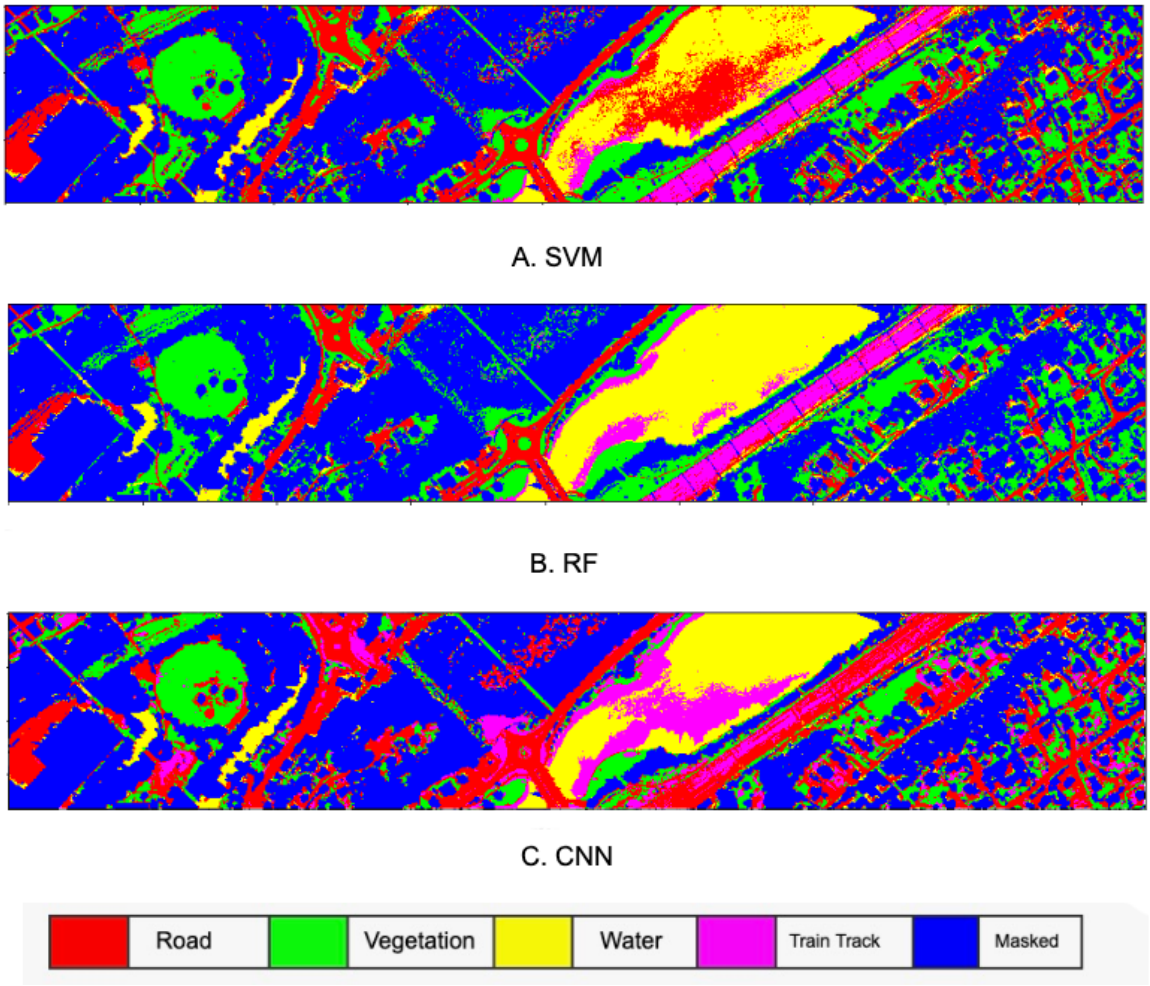


Figure 4.11: Result comparison of SVM, RF and CNN on fused radiance dataset.

Conclusion

Hyperspectral and LiDAR data are used in this analysis to extract road edges based on SVM, RF, and CNN models. Both hyperspectral and LiDAR data was obtained from the Sandvika region of Norway. The data is split into training and testing data. The features derived from the hyperspectral image using PCA and NDVI are combined with the LiDAR elevation and intensity features. These features are used as input to machine and deep learning models. The RF classification result is higher than the SVM and CNN classification results. CNN performed low as the algorithm was overfitted and there was less data available to train the model. The F1 score and kappa coefficient of the classification is shown in table 5.1.

The results of the experiment show that the overall accuracy of the classification improved from 79% to 81% when LiDAR data is combined with SWIR hyperspectral data. Classes have been well classified when LiDAR elevation and intensity features are used. Similarly, the classification algorithm better classifies the radiance data than the reflectance data. Train tracks and roads were well generalized when radiance data was used. It indicates that some of the essential data was lost during the atmospheric correction of the radiance data, as a result of which the reflectance data can not separate the road and train track class well. The reflectance data also consists of mixed pixels, due to which the train pixel exhibited spectral properties similar to those of the road class.

The SVM model, simpler than CNN, performed better than CNN in the classification of hyperspectral and LiDAR data. It can be shown that the simpler model appears to work better than the more complicated models. The most important aspect of SVM is its ability to generalize well from a small number of training data. This can achieve fair precision even with smaller samples of training data which are not possible with CNN. However, there are drawbacks to SVM, such as the selection of main SVM parameters such as the kernel function. Choosing a small value for the kernel width parameter would result in overfitting, whereas the large kernel width values would result in oversmoothing.

Table 5.1: Comparison of SVM, RF, and CNN classification result

	SVM	RF	CNN
F1-Score	0.82	0.88	0.60
Kappa Coefficient	0.77	0.88	0.73

The RF model, the simplest of all models, outperformed both CNN and SVM in the classification of hyperspectral and LiDAR data. The RF classifier is less sensitive than other machine learning classifiers, such as SVM, to the quality of training samples and to overfitting due to the large number of decision trees generated by the random selection of a number of training samples. The RF classifier has been shown to be suitable for the classification of hyperspectral data where the curse of dimensionality and closely correlated data present major challenges. Another advantage of the RF classifier is that it needs only two parameters to be optimized, while the SVM needs a number of parameters to be configured.

The CNN model, the most advanced of the three models, performed poorly compared to the other two models. It focused on capturing the small details of the images and was only able to generalize the training data well. The validation data could not be generalized well, so many of the classes were misclassified.

The conclusion of this study is that, the radiance data performed better than the reflectance data and similarly the simpler model, RF, outperformed all other models in classifying and extracting the road edges. This shows that investing time and resources in atmospheric correction of the radiance data is not effective as the precision of the classification of the radiance data is higher than that of the reflectance data. Similarly, the addition of LiDAR data to hyperspectral data did not significantly change the accuracy of the classification, so spending an enormous amount of money and time on the acquisition of LiDAR data is not worth much.

5.1 Further Work

Many limitations were addressed during this project, including limited data size, LiDAR and hyperspectral data fusion method and failure to extract roads under trees and buildings. Therefore, some of the additional work that can be done to mitigate these limitations is described in this section. The poor performance of the CNN model was attributed to the small size of the training data so that more data from different areas of the study area could be used to train the model. In this study, the pixel-wise fusion of hyperspectral and LiDAR data was used and the classification performance improved slightly. More advanced object-based hyperspectral and LiDAR data fusion can be used to enhance classification performance. In the same way, classes can be further divided into sub-classes, for example, the road class can be divided into pavements and other sub-classes based on road construction materials. In addition, several validation data from various parts of the research area can be used to evaluate the model. Other advanced classification and segmentation algorithms can be implemented. Furthermore, effective method can be used to extract roads under the trees using the combination of hyperspectral and LiDAR data.

Bibliography

[A et al., 2017] A, A. G., C, P. O., M, T., P, J. C., J, M. F., and G, D. (2017). Hyperspectral imaging combined with principal component analysis for bruise damage detection on white mushrooms (Agaricus bisporus). *Journal of Chemometrics*, 22(3-4):259–267.

[Andersen et al., 2005] Andersen, H.-E., McGaughey, R. J., and Reutebuch, S. E. (2005). Estimating forest canopy fuel parameters using lidar data. *Remote sensing of Environment*, 94(4):441–449.

[ArcGis, 2018] ArcGis (2018). What is lidar data.

[Archis, 2019] Archis (2019). Shapefiles.

[AS, 2019] AS, N. E. O. (2019). Hyspex imaging spectrometer user's manual.

[Behmann et al., 2014] Behmann, J., Steinrücken, J., and Plümer, L. (2014). Detection of early plant stress responses in hyperspectral images. *ISPRS Journal of Photogrammetry and Remote Sensing*, 93:98 – 111.

[Bioucas-Dias et al., 2012] Bioucas-Dias, J. M., Plaza, A., Dobigeon, N., Parente, M., Du, Q., Gader, P., and Chanussot, J. (2012). Hyperspectral unmixing overview: Geometrical, statistical, and sparse regression-based approaches. *IEEE journal of selected topics in applied earth observations and remote sensing*, 5(2):354–379.

[Burger and Gowen, 2011] Burger, J. and Gowen, A. (2011). Data handling in hyperspectral image analysis. *Chemometrics and Intelligent Laboratory Systems*, 108(1):13 – 22. Analytical Platforms for Providing and Handling Massive Chemical Data.

[Canny, 1986] Canny, J. (1986). A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*, (6):679–698.

[Cao et al., 2016] Cao, J., Chen, Z., and Wang, B. (2016). Deep convolutional networks with superpixel segmentation for hyperspectral image classification. In *2016 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, pages 3310–3313. IEEE.

[Chang and Lin, 2011] Chang, C.-C. and Lin, C.-J. (2011). Libsvm: A library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)*, 2(3):1–27.

[Charles et al., 2010] Charles, A., Olshausen, B., and Rozell, C. J. (2010). Sparse coding for spectral signatures in hyperspectral images. In *2010 Conference Record*

of the Forty Fourth Asilomar Conference on Signals, Systems and Computers, pages 191–195.

[Chauhan, 2019] Chauhan, N. S. (2019). Decision tree algorithm — explained.

[Chen et al., 2004] Chen, J., Jönsson, P., Tamura, M., Gu, Z., Matsushita, B., and Eklundh, L. (2004). A simple method for reconstructing a high-quality ndvi time-series data set based on the savitzky–golay filter. *Remote Sensing of Environment*, 91(3):332 – 344.

[Cloudfactory, 2013] Cloudfactory (2013). The ultimate guide to data labeling for machine learning.

[Cortes and Vapnik, 1995] Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3):273–297.

[Davis, 2012] Davis, O. (2012). Processing and working with lidar data in arcgis: A practical guide for archaeologists.

[Domingos, 2012] Domingos, P. (2012). A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87.

[Feng and Sun, 2012] Feng, Y.-Z. and Sun, D.-W. (2012). Application of hyperspectral imaging in food safety inspection and control: a review. *Critical reviews in food science and nutrition*, 52(11):1039–1058.

[Fruchart, 2011] Fruchart, C. (2011). Analyse spatiale et temporelle des paysages de la forêt de chailluz (besançon, doubs).

[Gandhi, 2018] Gandhi, R. (2018). Support vector machine — introduction to machine learning algorithms.

[GeeksforGeeks, 2018] GeeksforGeeks (2018). Svm hyperparameter tuning using gridsearchcv — ml.

[Geography, 2013] Geography, G. (2013). Dem, dsm dtm differences – a look at elevation models in gis.

[Geospatials, 2013] Geospatials, H. (2013). Digital number, radiance, and reflectance.

[Geospatials, 2019] Geospatials, H. (2019). Supported file types.

[Gogineni and Chaturvedi, 2019] Gogineni, R. and Chaturvedi, A. (2019). Hyperspectral image classification. In *Processing and Analysis of Hyperspectral Data*. IntechOpen.

[Gowen, 2014] Gowen, A. (2014). Near infrared hyperspectral image analysis using r. part 2: working with hypercubes. *NIR news*, 25(3):17–19.

[Grätzer, 2007] Grätzer, G. (2007). *More Math Into*. Springer.

[Harris Geospatial Solution, 2020] Harris Geospatial Solution, I. (2020). Envi user guide.

[Heiden et al., 2012] Heiden, U., Heldens, W., Roessner, S., Segl, K., Esch, T., and Mueller, A. (2012). Urban structure type characterization using hyperspectral remote sensing and height information. *Landscape and urban Planning*, 105(4):361–375.

[Ho et al., 1992] Ho, K.-L., Hsu, Y.-Y., and Yang, C.-C. (1992). Short term load forecasting using a multilayer neural network with an adaptive learning algorithm. *IEEE Transactions on Power Systems*, 7(1):141–149.

[Hoffbeck and Landgrebe, 1994] Hoffbeck, J. P. and Landgrebe, D. A. (1994). Effect of radiance-to-reflectance transformation and atmosphere removal on maximum likelihood classification accuracy of high-dimensional remote sensing data. In *Proceedings of IGARSS'94-1994 IEEE International Geoscience and Remote Sensing Symposium*, volume 4, pages 2538–2540. IEEE.

[Hsu and Lin, 2002] Hsu, C.-W. and Lin, C.-J. (2002). A comparison of methods for multiclass support vector machines. *IEEE transactions on Neural Networks*, 13(2):415–425.

[Irwan Hariyono and Windiastuti, 2018] Irwan Hariyono, M. and Windiastuti, R. (2018). Classification of lidar data to generate digital terrain model.

[Jaitley, 2018] Jaitley, U. (2018). Why data normalization is necessary for machine learning models.

[Jolliffe and Cadima, 2016] Jolliffe, I. T. and Cadima, J. (2016). Principal component analysis: a review and recent developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374(2065):20150202.

[Joseph, 2005] Joseph, G. (2005). *Fundamentals of remote sensing*. Universities Press.

[Kerle et al., 2004] Kerle, N., Janssen, L. L., and Huurneman, G. C. (2004). Principles of remote sensing. *ITC, Educational textbook series*, 2:250.

[Khodadadzadeh et al., 2015] Khodadadzadeh, M., Li, J., Prasad, S., and Plaza, A. (2015). Fusion of hyperspectral and lidar remote sensing data using multiple feature learning. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 8(6):2971–2983.

[Khosravipour et al., 2016] Khosravipour, A., Skidmore, A. K., and Isenburg, M. (2016). Generating spike-free digital surface models using lidar raw point clouds: A new approach for forestry applications. *International journal of applied earth observation and geoinformation*, 52:104–114.

[Khosravipour et al., 2015] Khosravipour, A., Skidmore, A. K., Isenburg, M., and Wang, T. (2015). Development of an algorithm to generate pit-free digital surface models from lidar. In *SilviLaser 2015 (14th: 2015)*, pages 247–249. International Society for Photogrammetry and Remote Sensing ISPRS.

[Kokaly, 2001] Kokaly, R. F. (2001). Investigating a physical basis for spectroscopic estimates of leaf nitrogen concentration. *Remote Sensing of Environment*, 75(2):153 – 161.

[Lay, 2016] Lay, D. C. (2016). Linear algebra and its applications 5th edition.

[Li, 2017] Li, P. (2017). Optimization algorithms for deep learning.

[Li et al., 2017] Li, Y., Zhang, H., and Shen, Q. (2017). Spectral–spatial classification of hyperspectral imagery with 3d convolutional neural network. *Remote Sensing*, 9(1):67.

[Liaw et al., 2002] Liaw, A., Wiener, M., et al. (2002). Classification and regression by randomforest. *R news*, 2(3):18–22.

[Licciardi and Chanussot, 2018] Licciardi, G. and Chanussot, J. (2018). Spectral transformation based on nonlinear principal component analysis for dimensionality reduction of hyperspectral images. *European Journal of Remote Sensing*, 51(1):375–390.

[Lu and Fei, 2014] Lu, G. and Fei, B. (2014). Medical hyperspectral imaging: a review. *Journal of biomedical optics*, 19(1):010901.

[Ma et al., 2013] Ma, W., Gong, C., Hu, Y., Meng, P., and Xu, F. (2013). The hughes phenomenon in hyperspectral classification based on the ground spectrum of grasslands in the region around qinghai lake. In *International Symposium on Photoelectronic Detection and Imaging 2013: Imaging Spectrometer Technologies and Applications*, volume 8910, page 89101G. International Society for Optics and Photonics.

[Maitra, 2019] Maitra, S. (2019). What canny edge detection algorithm is all about?

[Manolakis, 2005] Manolakis, D. G. (2005). Taxonomy of detection algorithms for hyperspectral imaging applications. *Optical engineering*, 44(6):066403.

[Manolakis et al., 2016] Manolakis, D. G., Lockwood, R. B., and Cooley, T. W. (2016). *Introduction*, page 1–35. Cambridge University Press.

[McCulloch and Pitts, 1943] McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133.

[Mishra, 2018] Mishra, A. (2018). Metrics to evaluate your machine learning algorithm.

[Mitchell et al., 1997] Mitchell, T. M. et al. (1997). Machine learning.

[Morsdorf et al., 2004] Morsdorf, F., Meier, E., Kötz, B., Itten, K. I., Dobbertin, M., and Allgöwer, B. (2004). Lidar-based geometric reconstruction of boreal type forest stands at single tree level for forest and wildland fire management. *Remote sensing of environment*, 92(3):353–362.

[Mutanga et al., 2012] Mutanga, O., Adam, E., and Cho, M. A. (2012). High density biomass estimation for wetland vegetation using worldview-2 imagery and random forest regression algorithm. *International Journal of Applied Earth Observation and Geoinformation*, 18:399 – 406.

[Oh et al., 2016] Oh, S. W., Brown, M. S., Pollefeys, M., and Kim, S. J. (2016). Do it yourself hyperspectral imaging with everyday digital cameras. In *CVPR*, pages 2461–2469.

[Parmar, 2018] Parmar, R. (2018). Common loss functions in machine learning.

[Patel, 2018] Patel, S. (2018). Chapter 2 : Svm (support vector machine).

[Pisapia et al., 2018] Pisapia, C., Jamme, F., Duponchel, L., and Ménez, B. (2018). Tracking hidden organic carbon in rocks using chemometrics and hyperspectral imaging. *Scientific reports*, 8(1):1–14.

[Pyle and San José, 2015] Pyle, D. and San José, C. (2015). An executive's guide to machine learning. *McKinsey Quarterly*, 3:44–53.

[QGIS, 2020] QGIS (2020). Qgis user guide.

[Raczko and Zagajewski, 2017] Raczko, E. and Zagajewski, B. (2017). Comparison of support vector machine, random forest and neural network classifiers for tree species classification on airborne hyperspectral apex images. *European Journal of Remote Sensing*, 50(1):144–154.

[rapidlasso GmbH, 2020] rapidlasso GmbH (2020). Lastools user guide.

[Raschka and Mirjalili, 2019] Raschka, S. and Mirjalili, V. (2019). *Python Machine Learning, 3rd Ed.* Packt Publishing, Birmingham, UK, 3 edition.

[Rees, 2013] Rees, W. G. (2013). *Physical principles of remote sensing*. Cambridge university press.

[Richter, 2018] Richter, Schläpfer, D. (2018). Atmospheric/topographic correction for airborne imagery.

[Richter and Schläpfer, 2002] Richter, R. and Schläpfer, D. (2002). Geo-atmospheric processing of airborne imaging spectrometry data. part 2: atmospheric/topographic correction. *International Journal of Remote Sensing*, 23(13):2631–2649.

[Rodarmel and Shan, 2002] Rodarmel, C. and Shan, J. (2002). Principal component analysis for hyperspectral image classification. *Surveying and Land Information Science*, 62(2):115–122.

[Roderick et al., 1996] Roderick, M., Smith, R., and Lodwick, G. (1996). Calibrating long-term avhrr-derived ndvi imagery. *Remote Sensing of Environment*, 58(1):1 – 12.

[Sabat-Tomala et al., 2020] Sabat-Tomala, A., Raczko, E., and Zagajewski, B. (2020). Comparison of support vector machine and random forest algorithms for invasive and expansive species classification using airborne hyperspectral data. *Remote Sensing*, 12(3):516.

[Saha, 2018] Saha, S. (2018). A comprehensive guide to convolutional neural networks — the eli5 way.

[Sakkaf, 2019] Sakkaf, Y. (2019). Decision trees: Id3 algorithm explained.

[Schowengerdt, 2007] Schowengerdt, R. A. (2007). Chapter 1 - the nature of remote sensing. In Schowengerdt, R. A., editor, *Remote Sensing (Third Edition)*, pages 1 – X. Academic Press, Burlington, third edition edition.

[Shafri et al., 2012] Shafri, H. Z., Taherzadeh, E., Mansor, S., and Ashurov, R. (2012). Hyperspectral remote sensing of urban areas: an overview of techniques and applications. *Research Journal of Applied Sciences, Engineering and Technology*, 4(11):1557–1565.

[SHARMA, 2017] SHARMA, S. (2017). Activation functions in neural networks.

[Shilo, 2018] Shilo, R. (2018). What's the matter with ndvi? you are!

[Shippert, 2013] Shippert, P. (2013). Digital number, radiance, and reflectance.

[Song et al., 2002] Song, J.-H., Han, S.-H., Yu, K. Y., and Kim, Y.-I. (2002). Assessing the possibility of land-cover classification using lidar intensity data. *Interna-*

tional archives of photogrammetry remote sensing and spatial information sciences, 34(3/B):259–262.

[Song and Civco, 2004] Song, M. and Civco, D. (2004). Road extraction using svm and image segmentation. *Photogrammetric Engineering & Remote Sensing*, 70(12):1365–1371.

[Terratec, 2019] Terratec (2019). Data collection report and processing of hyperspectral data.

[The Global Land Covering Facility, 2006] The Global Land Covering Facility, U. o. M. (2006). File format guide.

[Tro, 2018] Tro, N. J. (2018). *Chemistry: Structure and Properties*, page 36–116. Pearson.

[Uddin et al., 2017] Uddin, M., Mamun, M., and Hossain, M. (2017). Feature extraction for hyperspectral image classification. In *2017 IEEE Region 10 Humanitarian Technology Conference (R10-HTC)*, pages 379–382. IEEE.

[Valor and Caselles, 1996] Valor, E. and Caselles, V. (1996). Mapping land surface emissivity from ndvi: Application to european, african, and south american areas. *Remote Sensing of Environment*, 57(3):167 – 184.

[Vasefi et al., 2016] Vasefi, F., MacKinnon, N., and Farkas, D. (2016). Chapter 16 - hyperspectral and multispectral imaging in dermatology. In Hamblin, M. R., Avci, P., and Gupta, G. K., editors, *Imaging in Dermatology*, pages 187 – 201. Academic Press, Boston.

[Venugopal et al., 2015] Venugopal, V., Fang, Q., and Intes, X. (2015). 1 - multimodal diffuse optical imaging for biomedical applications. In Meglinski, I., editor, *Biophotonics for Medical Applications*, Woodhead Publishing Series in Biomaterials, pages 3 – 24. Woodhead Publishing.

[Vieira et al., 2010] Vieira, S. M., Kaymak, U., and Sousa, J. M. C. (2010). Cohen's kappa coefficient as a performance measure for feature selection. In *International Conference on Fuzzy Systems*, pages 1–8.

[Wang et al., 2019] Wang, C., Guo, P., Wang, H., and Yang, F. (2019). Urban trunk roads extraction using hough transform and ndvi in airborne hyperspectral remote sensing images. In *Journal of Physics: Conference Series*, volume 1237, page 032030. IOP Publishing.

[Wang, 1986] Wang, W.-C. (1986). Electromagnetic wave theory. *Google Scholar*.

[Wold et al., 1987] Wold, S., Esbensen, K., and Geladi, P. (1987). Principal component analysis. *Chemometrics and Intelligent Laboratory Systems*, 2(1):37 – 52. Proceedings of the Multivariate Statistical Workshop for Geologists and Geochemists.

[Yang et al., 2018] Yang, X., Ye, Y., Li, X., Lau, R. Y., Zhang, X., and Huang, X. (2018). Hyperspectral image classification with deep learning models. *IEEE Transactions on Geoscience and Remote Sensing*, 56(9):5408–5423.

[Zhang et al., 2003] Zhang, K., Chen, S.-C., Whitman, D., Shyu, M.-L., Yan, J., and Zhang, C. (2003). A progressive morphological filter for removing nonground mea-

surements from airborne lidar data. *IEEE transactions on geoscience and remote sensing*, 41(4):872–882.

[Zhang et al., 2012] Zhang, X., Liu, F., He, Y., and Li, X. (2012). Application of hyperspectral imaging and chemometric calibrations for variety discrimination of maize seeds. *Sensors*, 12(12):17234–17246.

[Zhang et al., 2018] Zhang, Z., Liu, Q., and Wang, Y. (2018). Road extraction by deep residual u-net. *IEEE Geoscience and Remote Sensing Letters*, 15(5):749–753.

[Zhong et al., 2017] Zhong, Y., Cao, Q., Zhao, J., Ma, A., Zhao, B., and Zhang, L. (2017). Optimal decision fusion for urban land-use/land-cover classification based on adaptive differential evolution using hyperspectral and lidar data. *Remote Sensing*, 9(8):868.

[Zhu and Goldberg, 2009] Zhu, X. and Goldberg, A. B. (2009). Introduction to semi-supervised learning. *Synthesis lectures on artificial intelligence and machine learning*, 3(1):1–130.

Appendices

Training and Validation with Random Forest

```
#!/usr/bin/env python
# coding: utf-8


# Importing necessary packages and libraries
from spectral import *
import spectral.io.envi as envi
import pandas as pd
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import MinMaxScaler
import numpy as np
import matplotlib.pyplot as plt
from scipy import ndimage as ndi
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report
from sklearn.metrics import cohen_kappa_score
from sklearn.metrics import confusion_matrix
import matplotlib
import time
from sklearn.metrics import plot_confusion_matrix
plt.rcParams.update({'font.size': 22})


# Function definition
# Function to load image, header file and label data
# returns envi image object, numpy image array and numpy labels array
def loadDataset(header_file, data_file, label_file):
    image = envi.open(header_file, data_file)
    data = image.load()
    label_data = Image.open(label_file)
    labels = np.array(label_data)
    return image, data, labels
```

```
42
43
44 # Function to apply PCA on data X
45 # returns pc reduced image as numpy array.
46 def applyPCA(X):
47     pc = principal_components(X)
48     pc_0999 = pc.reduce(fraction=0.999)
49     img_pc = pc_0999.transform(X)
50     return pc_0999, img_pc
51
52
53 # Function apply NDVI
54 #returns ndvi value numpy array
55 def applyNDVI(X):
56     vi = ndvi(X, 76, 105)
57     data_ndvi = np.nan_to_num(vi)
58     return data_ndvi
59
60
61 # Function to combine two pandas dataframe
62 # returns concatenated dataframe
63 def concatPcaNdvi(pc, ndvi):
64     df_pca = pd.DataFrame(pc.reshape(-1, pc.shape[2]))
65     string = "PC"
66     df_pca.columns = [string + str(n) for n in range(1, pc.shape[2]+1)
       ]
67     df_ndvi = pd.DataFrame(ndvi.reshape(-1, 1))
68     df_ndvi.columns = ["NDVI"]
69     main_df = pd.concat([df_pca, df_ndvi], axis=1, sort=False)
70     return main_df
71
72
73 # Function to combine two data and target value
74 # returns concatenated dataframe
75 def concatDataClass(data, target):
76     df_class = pd.DataFrame(target.reshape(-1, 1))
77     string = "Target"
78     df_class.columns = [string]
79     df = pd.concat([data, df_class], axis=1, sort=False)
80     df_masked = df[(df[['Target']] != 0).all(axis=1)]
81     X = df_masked.drop(columns=['Target']).values
82     y = df_masked['Target'].values
83     return df, X, y
84
85
86 # Visualize image
87 def showImage(img):
88     view = imshow(img, (76, 46, 21), stretch=((0.0, 0.9), (0.0, 0.9),
       (0.0, 0.9)), figsize=(16, 16))
89
90
91 # Function to Standarise data so that mean is 0 and standard deviation
       is 1
92 # returns scaler model and scaled data.
93 def standariseData(X):
94     scaler = StandardScaler()
95     scaler.fit(X)
96     X_scaled = scaler.transform(X)
```

```
97      return scaler, X_scaled
98
99
100 # Function to split the data into 80% train and 20% test set
101 # return train and test data and labels
102 def splitTrainTestSet(X, y, testRatio = 0.2):
103     X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=testRatio, random_state=345,
104                                                         stratify=y)
105     return X_train, X_test, y_train, y_test
106
107
108 # Function to optimise the parameter of classification algorithm
109 # returns the best parameters
110 def parameterOptimization(X_train, y_train):
111     n_estimators = [int(x) for x in np.linspace(start=50, stop=300,
    num=5)]
112     max_features = ['auto', 'sqrt', 'log2']
113     max_depth = [int(x) for x in np.linspace(start=30, stop=100, num=
    5)]
114     criterion = ['gini', 'entropy']
115     param_grid = {
116     'n_estimators': n_estimators,
117     'max_features': max_features,
118     'max_depth' : max_depth,
119     'criterion' : criterion
120 }
121     grid = GridSearchCV(RandomForestClassifier(), param_grid, refit =
    True, verbose = 3, n_jobs=-1, cv=3)
122     grid.fit(X_train, y_train)
123     return grid.best_estimator_
124
125
126 # Function to estimate the model accuracy using train and test data
127 def pretrain_model(estimator, X_train, y_train, X_test, y_test):
128     cl = estimator
129     start = time.time()
130     cl.fit(X_train, y_train)
131     stop = time.time()
132     print(f"Training time: {stop - start}s")
133     pred = cl.predict(X_test)
134     accuracy_score(y_test, pred)
135     print(f"The accuracy score: {accuracy_score(y_test, pred)}")
136
137
138 # Function to train the model with all the data
139 # reutn trained model
140 def train_model(estimator, X_scaled, y):
141     cl = estimator
142     start = time.time()
143     cl.fit(X_scaled, y)
144     stop = time.time()
145     print(f"Training time: {stop - start}s")
146     return cl
147
148
149 # Function to predict the unknown data by trained model
150 # return predicted data as numpy array.
```

```
151 def predict_image(img, cl, scaler, dataframe):
152     for column in dataframe.columns:
153         if column == "Target":
154             dataframe = dataframe.drop(columns=['Target']).values
155     X_data_scaled = scaler.transform(dataframe)
156     start = time.time()
157     y_pred = cl.predict(X_data_scaled)
158     stop = time.time()
159     print(f"Testing time:{stop-start}s")
160     predicted_hsi_mask = y_pred.reshape(img.shape[0], img.shape[1])
161     return predicted_hsi_mask
162 
163 
164 # Visualization Functions
165 # Visualise image with labels
166 def visualize_result(img, predicted_map):
167     view = imshow(img, (76, 46, 21), stretch=((0.0, 0.9), (0.0, 0.9),
    (0.0, 0.9)),figsize=(16, 16), classes=predicted_map)
168     view.set_display_mode('overlay')
169     view.class_alpha = 1
170     view.show_data
171 
172 
173 # Function to Extract road edges using Canny Edge Detection
174 # return road edges
175 def extract_road_edges(road):
176     from skimage.feature import canny
177     from skimage.viewer import ImageViewer
178     from skimage import io
179     from skimage import img_as_uint
180     edges = canny(
181         image=road,
182         sigma=5.5,
183         low_threshold=0.1,
184         high_threshold=0.3,
185     )
186     from skimage.morphology import binary_dilation
187     edge = binary_dilation(edges, selem=None, out=None)
188     edge = binary_dilation(edge, selem=None, out=None)
189     road_edges = edge.astype(int)
190     return road_edges
191 
192 
193 # Function to plot Scree Plot of PCA
194 def screePlot(pc, X_pca):
195     eigvals = pc.eigenvalues
196     num_vars = X_pca.shape[2]
197 
198 
199     fig = plt.figure(figsize=(16, 10))
200     sing_vals = np.arange(num_vars) + 1
201     plt.plot(sing_vals, eigvals, 'ro-', linewidth=2)
202     plt.title('Scree Plot')
203     plt.xlabel('Principal Component')
204     plt.ylabel('Eigenvalue')
205     leg = plt.legend(['Eigenvalues from SVD'], loc='best', borderpad
    =0.3,
206                      shadow=False, prop=matplotlib.font_manager.
```

```
    FontProperties(size='small'),
207                        markerscale=0.4)
208     leg.get_frame().set_alpha(0.4)
209     leg.set_draggable(state=True)
210     plt.show()
211
212
213 # Function to evaluate performance using accuracy score, F1-score, and
        cohen kappa coefficient
214 def performance_evaluation(y_val, y_pred):
215     print(f"Accuracy Score: {accuracy_score(y_val, y_pred)}")
216     target_names = ['Road', 'Vegetation', 'Unclassified', 'Water', '
    Train Track']
217     print(classification_report(y_val, y_pred, target_names=
    target_names))
218     print(f"Cohen Kappa Score: {cohen_kappa_score(y_val, y_pred)}")
219
220
221 # Function to plot confusion matrix
222 def plot_confusion_matrix(cm,
223                           target_names,
224                           title='Confusion matrix',
225                           cmap=None,
226                           normalize=True):
227     """
228     given a sklearn confusion matrix (cm), make a nice plot
229
230     Arguments
231     ---------
232     cm:           confusion matrix from sklearn.metrics.
    confusion_matrix
233
234     target_names: given classification classes such as [0, 1, 2]
235                   the class names, for example: ['high', 'medium', '
    low']
236
237     title:        the text to display at the top of the matrix
238
239     cmap:         the gradient of the values displayed from matplotlib
    .pyplot.cm
240                   see http://matplotlib.org/examples/color/
    colormaps_reference.html
241                   plt.get_cmap('jet') or plt.cm.Blues
242
243     normalize:    If False, plot the raw numbers
244                   If True, plot the proportions
245
246     Usage
247     -----
248     plot_confusion_matrix(cm           = cm,                  #
    confusion matrix created by
249                                                              #
    sklearn.metrics.confusion_matrix
250                           normalize    = True,                # show
    proportions
251                           target_names = y_labels_vals,       # list
    of names of the classes
252                           title        = best_estimator_name) # title
```

```
    of graph
253
254     Citiation
255     ---------
256     http://scikit-learn.org/stable/auto_examples/model_selection/
    plot_confusion_matrix.html
257
258     """
259     import matplotlib.pyplot as plt
260     import numpy as np
261     import itertools
262
263     accuracy = np.trace(cm) / float(np.sum(cm))
264     misclass = 1 - accuracy
265
266     if cmap is None:
267         cmap = plt.get_cmap('Blues')
268
269     plt.figure(figsize=(16, 10))
270     plt.imshow(cm, interpolation='nearest', cmap=cmap)
271     plt.title(title)
272     plt.colorbar()
273
274     if target_names is not None:
275         tick_marks = np.arange(len(target_names))
276         plt.xticks(tick_marks, target_names, rotation=45)
277         plt.yticks(tick_marks, target_names)
278
279     if normalize:
280         cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
281
282
283     thresh = cm.max() / 1.5 if normalize else cm.max() / 2
284     for i, j in itertools.product(range(cm.shape[0]), range(cm.shape
    [1])):
285         if normalize:
286             plt.text(j, i, "{:0.4f}".format(cm[i, j]),
287                      horizontalalignment="center",
288                      color="white" if cm[i, j] > thresh else "black")
289         else:
290             plt.text(j, i, "{:,}".format(cm[i, j]),
291                      horizontalalignment="center",
292                      color="white" if cm[i, j] > thresh else "black")
293
294
295     plt.tight_layout()
296     plt.ylim([4.5, -.5])
297     plt.ylabel('True label')
298     plt.xlabel('Predicted label')
299     plt.show()
300
301 # Function to plot the spectral signature of image pixels
302 def plotSpectra(a, t, v):
303     wavelength=envi.read_envi_header('../VNIR_carlb.hdr')['wavelength'
    ]
304     ww = [float(i) for i in wavelength]
305     #to plot a few spectra
306     plt.figure(figsize=(15,10))
```

```
307     plt.xlabel('Wavelength [nm]')
308     plt.ylabel('Radiance intensity')
309     plt.plot(ww,a, label= "Asphalt")
310     plt.plot(ww,t, label = "Train Track")
311     plt.plot(ww,v, label = "Vegetation")
312     plt.legend(prop={'size': 20})
313     plt.show()
314
315
316
317 # Hyperspectral data
318 # Training
319 image, img, label =  loadDataset('../../hyperImage/reflectance/09
        _VNIR_ROAD_MASKED.hdr',
320                                  '../../hyperImage/reflectance/09
        _VNIR_ROAD_MASKED',
321                          '../../hyperImage/reflectance/roi/class4.tif'
        )
322
323
324 pc, X_pca = applyPCA(img) # Applying PCA on the image
325 screePlot(pc, X_pca)  # Plotting scree plot
326 X_pca = X_pca[:,:,:3]
327 ndvi = applyNDVI(img) #Applying NDVI to image
328 main_df = concatPcaNdvi(X_pca, ndvi)
329
330
331 df, X, y = concatDataClass(main_df, label)  # Concatenate dataframe
        and labels
332 visualize_result(img, label)
333 scaler, X_scaled = standariseData(X)  # Transform data to standard
        scale
334
335 #Split data into training and test set
336 X_train, X_test, y_train, y_test = train_test_split(X_scaled, y)
337 estimator = parameterOptimization(X_train, y_train) # Hyperparameter
        optimization
338
339
340 pretrain_model(estimator, X_train, y_train, X_test, y_test)
341
342 # train the model with all the training data
343 cl = train_model(estimator, X_scaled, y)
344 prediction_map = predict_image(img, cl, scaler, df)
345 visualize_result(img, prediction_map)
346 road = (prediction_map == 1).astype(int)
347 road_edges = extract_road_edges(road)
348 visualize_result(img, road_edges)
349
350
351 # Hyperspectral Validation data
352 val_image, val_img, val_label = loadDataset('../../hyperImage/
        reflectance/vnir/08_clippped_road.hdr',
353                                             '../../hyperImage/
        reflectance/vnir/08_clippped_road',
354                                             '../../hyperImage/
        reflectance/vnir/class_val_ref.tif')
355 showImage(val_img)
```

```
visualize_result(val_img, val_label)

pc, X_pca_val = applyPCA(val_img)
screePlot(pc, X_pca_val)
X_pca = X_pca_val[:,:,:3]
ndvi_val = applyNDVI(val_img)
main_df_val = concatPcaNdvi(X_pca, ndvi_val)


#main_df_val = pd.read_csv('../../hyperImage/csv/features/
    Ref_Vnir_HSI_feature_val')
predicted_map_val = predict_image(val_img, cl, scaler, main_df_val)
visualize_result(val_img, predicted_map_val)
road = (predicted_map_val == 1).astype(int)
road_edges = extract_road_edges(road)
visualize_result(val_img, road_edges)

df_val, X, y =  concatDataClass(main_df_val, val_label)
df_hsi_val_masked = df_val[(df_val[['Target']] != 0).all(axis=1)]
X_val = df_hsi_val_masked.drop(columns=['Target']).values
y_val = df_hsi_val_masked['Target'].values
X_data_scaled_val = scaler.transform(X_val)
y_pred = cl.predict(X_data_scaled_val)

performance_evaluation(y_val, y_pred)
cm = confusion_matrix(y_val, y_pred)
plot_confusion_matrix(cm,
                      normalize    = True,
                      target_names = ['Road', 'Vegetation', '
    Unclassified', 'Water', 'Train Track'],
                      title        = "Confusion Matrix, Normalized")


# LiDAR Data

#import lidar nDSM and intensity data
l_intensity_data = Image.open('../../hyperImage/reflectance/roi/
    masked_intensity.tif')
l_intensity = np.array(l_intensity_data)
df_intensity = pd.DataFrame(l_intensity.reshape(-1, 1))
string = "Lidar Intensity"
df_intensity.columns = [string]

l_dsm_data = Image.open('../../hyperImage/reflectance/roi/masked_nDSM.
    tif')
l_dsm = np.array(l_dsm_data)
df_dsm = pd.DataFrame(l_dsm.reshape(-1, 1))
string = "Lidar nDSM"
df_dsm.columns = [string]

gt_data = Image.open('../../hyperImage/reflectance/roi/class4.tif')
gt = np.array(gt_data)
df_class = pd.DataFrame(gt.reshape(-1, 1))
df_class.columns = ["Target"]
df_lidar = pd.concat([df_intensity, df_dsm, df_class], axis=1, sort=
    False)
df_lidar_masked =  df_lidar[(df_lidar[['Target']] != 0).all(axis=1)]
X = df_lidar_masked.drop(columns=['Target']).values
```

```
409 y = df_lidar_masked['Target'].values
410
411 scaler1, X_scaled = standariseData(X)
412 X_train, X_test, y_train, y_test = splitTrainTestSet(X_scaled, y, 0.2)
413 pretrain_model(estimator, X_train, y_train, X_test, y_test)
414 cl1 = train_model(estimator, X_scaled, y)
415
416 X_lidar_data = df_lidar.drop(columns=['Target']).values
417 S_lidar_data_scaled = scaler1.transform(X_lidar_data)
418 y_pred_lidar = cl1.predict(S_lidar_data_scaled)
419 predicted_lidar_map = y_pred_lidar.reshape(l_dsm.shape[0], l_dsm.shape
    [1])
420 road = (predicted_lidar_map == 1).astype(int)
421 road_edges = extract_road_edges(road)
422 visualize_result(img, predicted_lidar_map)
423 visualize_result(img, road_edges)
424
425
426 # LiDAR Validation
427 df_lidar_val = pd.read_csv('../../hyperImage/csv/features/
    Ref_Vnir_lidar_feature_val')
428 X_data_scaled = scaler1.transform(df_lidar_val)
429 val_pred_full = cl1.predict(X_data_scaled)
430 predicted_lidar_map_val = val_pred_full.reshape(val_image.shape[0],
    val_image.shape[1])
431
432 road = (predicted_lidar_map_val == 1).astype(int)
433 road_edges = extract_road_edges(road)
434 visualize_result(val_img, predicted_lidar_map_val)
435 visualize_result(val_img, road_edges)
436
437 df_lidar_main_val, X, y =  concatDataClass(df_lidar_val, val_label)
438 df_lidar_val_masked = df_lidar_main_val[(df_lidar_main_val[['Target']]
     != 0).all(axis=1)]
439 X_val = df_lidar_val_masked.drop(columns=['Target']).values
440 y_val = df_lidar_val_masked['Target'].values
441 X_data_scaled_val = scaler1.transform(X_val)
442 y_pred = cl1.predict(X_data_scaled_val)
443
444 performance_evaluation(y_val, y_pred)
445 cm = confusion_matrix(y_val, y_pred)
446 plot_confusion_matrix(cm,
447                       normalize    = True,
448                       target_names = ['Road', 'Vegetation', '
    Unclassified', 'Water', 'Train Track'],
449                       title        = "Confusion Matrix, Normalized")
450
451
452 # HSI + LiDAR (Fused Data)
453
454 df_lid_hsi = pd.concat([main_df,df_lidar], axis=1, sort=False)
455 df_masked =  df_lid_hsi[(df_lid_hsi[['Target']] != 0).all(axis=1)]
456 X = df_masked.drop(columns=['Target']).values
457 y = df_masked['Target'].values
458
459 scaler2, X_scaled = standariseData(X)
460 X_train, X_test, y_train, y_test = train_test_split(X_scaled, y)
461 pretrain_model(estimator, X_train, y_train, X_test, y_test)
```

```
cl2 = train_model(estimator, X_scaled, y)
prediction_map = predict_image(img, cl2, scaler2, df_lid_hsi)
visualize_result(img, prediction_map)
road = (prediction_map == 1).astype(int)
road_edges = extract_road_edges(road)
visualize_result(img, road_edges)

performance_evaluation(y_val, y_pred)
cm = confusion_matrix(y_val, y_pred)
plot_confusion_matrix(cm,
                      normalize    = True,
                      target_names = ['Road', 'Vegetation', '
    Unclassified', 'Water', 'Train Track'],
                      title        = "Confusion Matrix, Normalized")


# HSI + LIDAR (Fused) Validation data
df_lid_hsi_val = pd.concat([main_df_val, df_lidar_val], axis=1, sort=
    False)
predicted_hsi_lidar_map_val = predict_image(val_img, cl2, scaler2,
    df_lid_hsi_val)
visualize_result(val_img, predicted_hsi_lidar_map_val)
road = (predicted_hsi_lidar_map_val == 1).astype(int)
road_edges = extract_road_edges(road)
visualize_result(val_img, road_edges)

df_val, X, y =  concatDataClass(df_lid_hsi_val, val_label)
df_hsi_val_masked = df_val[(df_val[['Target']] != 0).all(axis=1)]
X_val = df_hsi_val_masked.drop(columns=['Target']).values
y_val = df_hsi_val_masked['Target'].values
X_data_scaled_val = scaler2.transform(X_val)
y_pred = cl2.predict(X_data_scaled_val)

performance_evaluation(y_val, y_pred)
cm = confusion_matrix(y_val, y_pred)
plot_confusion_matrix(cm,
                      normalize    = True,
                      target_names = ['Road', 'Vegetation', '
    Unclassified', 'Water', 'Train Track'],
                      title        = "Confusion Matrix, Normalized")
```

Training and Validation with CNN

B.1 Data Preprocessing

```
#!/usr/bin/env python
# coding: utf-8

# ## Importing necessary packages and libraries

# In[5]:


import numpy as np
from sklearn.decomposition import PCA
import scipy.io as sio
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
import os
import random
from random import shuffle
from skimage.transform import rotate
import scipy.ndimage
from spectral import *
import spectral.io.envi as envi
from PIL import Image
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib
from pickle import dump


# ## Function Definition

# In[6]:


# Function to load image, header file and label data
# returns envi numpy image array and numpy labels array
def loadDataset(header_file, data_file, label_file):
    image = envi.open(header_file, data_file)
    data = image.load()
    label_data = Image.open(label_file)
    labels = np.array(label_data)
    return data, labels

```

```
42 # Function to split the data into 80% train and 20% test set
43 # return train and test data and labels
44 def splitTrainTestSet(X, y, testRatio=0.10):
45     X_train, X_test, y_train, y_test = train_test_split(X, y,
       test_size=testRatio, random_state=345,
46                                                         stratify=y)
47     return X_train, X_test, y_train, y_test
48
49 # Function to oversample weak classes when the class is in unbalanced
       condition
50 # return new data with balanced class
51 def oversampleWeakClasses(X, y):
52     uniqueLabels, labelCounts = np.unique(y, return_counts=True)
53     maxCount = np.max(labelCounts)
54     labelInverseRatios = maxCount / labelCounts
55     # repeat for every label and concat
56     newX = X[y == uniqueLabels[0], :, :, :].repeat(round(
       labelInverseRatios[0]), axis=0)
57     newY = y[y == uniqueLabels[0]].repeat(round(labelInverseRatios[0])
       , axis=0)
58     for label, labelInverseRatio in zip(uniqueLabels[1:],
       labelInverseRatios[1:]):
59         cX = X[y== label,:,:,:].repeat(round(labelInverseRatio), axis
       =0)
60         cY = y[y == label].repeat(round(labelInverseRatio), axis=0)
61         newX = np.concatenate((newX, cX))
62         newY = np.concatenate((newY, cY))
63     np.random.seed(seed=42)
64     rand_perm = np.random.permutation(newY.shape[0])
65     newX = newX[rand_perm, :, :, :]
66     newY = newY[rand_perm]
67     return newX, newY
68
69 # Function to Standarise data so that mean is 0 and standard deviation
       is 1
70 # returns scaler model and scaled data.
71 def standartizeData(X):
72     newX = np.reshape(X, (-1, X.shape[2]))
73     scaler = preprocessing.StandardScaler().fit(newX)
74     newX = scaler.transform(newX)
75     newX = np.reshape(newX, (X.shape[0],X.shape[1],X.shape[2]))
76     return newX, scaler
77
78 # Function to apply PCA on data X
79 # returns pc reduced image as numpy array.
80 def applyPCA(X, numComponents=75):
81     newX = np.reshape(X, (-1, X.shape[2]))
82     pca = PCA(n_components=numComponents, whiten=True)
83     newX = pca.fit_transform(newX)
84     newX = np.reshape(newX, (X.shape[0],X.shape[1], numComponents))
85     return newX, pca
86
87 # Function to plot Scree Plot of PCA
88 def screePlot(pc, X_pca):
89     eigvals = pc.explained_variance_ratio_
90     num_vars = X_pca.shape[2]
91
92
```

```
93      fig = plt.figure(figsize=(16, 10))
94      sing_vals = np.arange(num_vars) + 1
95      plt.plot(sing_vals, eigvals, 'ro-', linewidth=2)
96      plt.title('Scree Plot')
97      plt.xlabel('Principal Component')
98      plt.ylabel('Eigenvalue')
99      leg = plt.legend(['Eigenvalues from SVD'], loc='best', borderpad
    =0.3,
100                       shadow=False, prop=matplotlib.font_manager.
    FontProperties(size='small'),
101                       markerscale=0.4)
102     leg.get_frame().set_alpha(0.4)
103     leg.set_draggable(state=True)
104     plt.show()
105
106 # Function to pad the data with zero
107 # return new padded data
108 def padWithZeros(X, margin=2):
109     newX = np.zeros((X.shape[0] + 2 * margin, X.shape[1] + 2* margin,
    X.shape[2]))
110     x_offset = margin
111     y_offset = margin
112     newX[x_offset:X.shape[0] + x_offset, y_offset:X.shape[1] +
    y_offset, :] = X
113     return newX
114
115 # Function to split images into small patches of sizw = WindowSize
116 # return patched data and labels
117 def createPatches(X, y, windowSize=5, removeZeroLabels = True):
118     margin = int((windowSize - 1) / 2)
119     zeroPaddedX = padWithZeros(X, margin=margin)
120     # split patches
121     patchesData = np.zeros((X.shape[0] * X.shape[1], windowSize,
    windowSize, X.shape[2]))
122     patchesLabels = np.zeros((X.shape[0] * X.shape[1]))
123     patchIndex = 0
124     for r in range(margin, zeroPaddedX.shape[0] - margin):
125         for c in range(margin, zeroPaddedX.shape[1] - margin):
126             patch = zeroPaddedX[r - margin:r + margin + 1, c - margin:
    c + margin + 1]
127             patchesData[patchIndex, :, :, :] = patch
128             patchesLabels[patchIndex] = y[r-margin, c-margin]
129             patchIndex = patchIndex + 1
130     if removeZeroLabels:
131         patchesData = patchesData[patchesLabels>0,:,:,:]
132         patchesLabels = patchesLabels[patchesLabels>0]
133         patchesLabels -= 1
134     return patchesData, patchesLabels
135
136 def createPatches_val(X, windowSize=5):
137     margin = int((windowSize - 1) / 2)
138     zeroPaddedX = padWithZeros(X, margin=margin)
139     # split patches
140     patchesData = np.zeros((X.shape[0] * X.shape[1], windowSize,
    windowSize, X.shape[2]))
141     patchIndex = 0
142     for r in range(margin, zeroPaddedX.shape[0] - margin):
143         for c in range(margin, zeroPaddedX.shape[1] - margin):
```

```
144             patch = zeroPaddedX[r - margin:r + margin + 1, c - margin:
    c + margin + 1]
145             patchesData[patchIndex, :, :, :] = patch
146             patchIndex = patchIndex + 1
147     return patchesData
148 
149 # Function for Data Augmentation
150 # return Augmented data
151 def AugmentData(X_train):
152     for i in range(int(X_train.shape[0]/2)):
153         patch = X_train[i,:,:,:]
154         num = random.randint(0,2)
155         if (num == 0):
156 
157             flipped_patch = np.flipud(patch)
158         if (num == 1):
159 
160             flipped_patch = np.fliplr(patch)
161         if (num == 2):
162 
163             no = random.randrange(-180,180,30)
164             flipped_patch = scipy.ndimage.interpolation.rotate(patch,
    no,axes=(1, 0),
165                                                                reshape
    =False, output=None, order=3, mode='constant', cval=0.0, prefilter=
    False)
166 
167 
168     patch2 = flipped_patch
169     X_train[i,:,:,:] = patch2
170 
171     return X_train
172 
173 
174 # Function to save the pateches
175 def savePreprocessedData(X_Patches, y_Patches, windowSize,
    wasPCAapplied = False, numPCAComponents = 0, testRatio = 0.25):
176     if wasPCAapplied:
177         with open("training/X_Patches_" + str(windowSize) + "PCA" +
    str(numPCAComponents) + "testRatio" + str(testRatio) + ".npy", 'bw'
    ) as outfile:
178             np.save(outfile, X_Patches)
179         with open("training/y_Patches_" + str(windowSize) + "PCA" +
    str(numPCAComponents) + "testRatio" + str(testRatio) + ".npy", 'bw'
    ) as outfile:
180             np.save(outfile, y_Patches)
181         # with open("validation/X_vals_Patches_" + str(windowSize) + "
    PCA" + str(numPCAComponents) + "testRatio" + str(testRatio) + ".npy
    ", 'bw') as outfile:
182             #   np.save(outfile, X_vals_Patches)
183         #with open("validation/y_vals_Patches_" + str(windowSize) + "
    PCA" + str(numPCAComponents) + "testRatio" + str(testRatio) + ".npy
    ", 'bw') as outfile:
184             #  np.save(outfile, y_vals_Patches)
185     else:
186         with open("../preprocessedData/XtrainWindowSize" + str(
    windowSize) + ".npy", 'bw') as outfile:
187             np.save(outfile, X_trainPatches)
```

```
188         with open("../preprocessedData/XtestWindowSize" + str(
    windowSize) + ".npy", 'bw') as outfile:
189             np.save(outfile, X_testPatches)
190         with open("../preprocessedData/ytrainWindowSize" + str(
    windowSize) + ".npy", 'bw') as outfile:
191             np.save(outfile, y_trainPatches)
192         #with open("../preprocessedData/ytestWindowSize" + str(
    windowSize) + ".npy", 'bw') as outfile:
193             np.save(outfile, y_testPatches)
194
195
196
197
198 # In[7]:
199
200
201 # Load the Global values (windowSize, numPCAcomponents, testRatio)
    from the text file global_variables.txt
202 myFile = open('global_variables.txt', 'r')
203 file = myFile.readlines()[:]
204
205
206 for line in file:
207
208     if line[0:3] == "win":
209
210         ds = line.find('=')
211         windowSize = int(line[ds+1:-1],10)
212
213     elif line[0:3] == "num":
214
215         ds = line.find('=')
216         numPCAcomponents = int(line[ds+2:-1],10)
217
218     else:
219
220         ds = line.find('=')
221         testRatio = float(line[ds+1:])
222
223
224 # In[10]:
225
226
227 X, y = loadDataset('../../hyperImage/reflectance/09_VNIR_ROAD_MASKED.
    hdr',
228                                     '../../hyperImage/reflectance/09
    _VNIR_ROAD_MASKED',
229                            '../../hyperImage/reflectance/roi/class4.tif'
    )
230 X, pca = applyPCA(X, numPCAcomponents)
231
232 X, scaler = standartizeData(X)
233 X.shape
234 XPatches, yPatches = createPatches(X, y, windowSize=windowSize)
235 #X_train, X_test, y_train, y_test = splitTrainTestSet(XPatches,
    yPatches, testRatio)
236 X_train, y_train = oversampleWeakClasses(XPatches, yPatches)
237 X_train = AugmentData(X_train)
```

```
savePreprocessedData(X_train, y_train, windowSize = windowSize,
                     wasPCAapplied=True, numPCAComponents =
    numPCAcomponents,testRatio = testRatio)
dump(scaler, open('scaler/scaler3.pkl', 'wb'))
```

B.2 Training CNN model

```
# -*- coding: utf-8 -*-
"""TrainingCNN.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1uK06Kcph-
    Z9kj51HWVWU8IFlvT0UQq9A
"""



"""# Importing necessary packages and libraries"""

!pip install spectral
import numpy as np
import scipy
import os
from keras.models import Sequential
from keras.layers import Input, Dense, Dropout, Flatten,
    BatchNormalization, Concatenate, Conv2DTranspose
from keras.layers import Conv2D, MaxPooling2D
from keras.optimizers import SGD
from keras import backend as K
import argparse
K.image_data_format()
from keras.utils import np_utils
import matplotlib.pyplot as plt
from keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
import h5py
from keras.models import load_model

"""# Connecting to google drive"""

from google.colab import drive
drive.mount('/content/drive')

"""# Load the Global values (windowSize, numPCAcomponents, testRatio)
    from the text file global_variables.txt"""

myFile = open('/content/drive/My Drive/thesis/global_variables.txt', '
    r')
file = myFile.readlines()[:]

for line in file:
    if line[0:3] == "win":
        ds = line.find('=')
        windowSize = int(line[ds+1:-1],10)

```

```
47      elif line[0:3] == "num":
48          ds = line.find('=')
49          numPCAcomponents = int(line[ds+2:-1],10)
50
51      else:
52          ds = line.find('=')
53          testRatio = float(line[ds+1:])
54
55  """# Function Definition"""
56
57  def splitTrainTestSet(X, y, testRatio=0.10):
58      X_train, X_test, y_train, y_test = train_test_split(X, y,
        test_size=testRatio, random_state=345,
59                                                          stratify=y)
60      return X_train, X_test, y_train, y_test
61
62  # Loading training datasets
63  train_X = np.load("/content/drive/My Drive/thesis/Hyperimage/
        Reflectance/VNIR/spatialSize/X_Patches_" + str(windowSize) + "PCA"
        + str(numPCAcomponents) + "testRatio" + str(testRatio)  + ".npy")
64  train_Y = np.load("/content/drive/My Drive/thesis/Hyperimage/
        Reflectance/VNIR/spatialSize/y_Patches_" + str(windowSize) + "PCA"
        + str(numPCAcomponents) + "testRatio" + str(testRatio) + ".npy")
65
66  #Splitting the data into train and test set
67  X_train, X_test, y_train, y_test = splitTrainTestSet(train_X, train_Y,
        testRatio)
68
69  # Reshape into (numberofsamples, channels, height, width)
70  train_X = np.reshape(train_X, (train_X.shape[0],train_X.shape[3],
        train_X.shape[1], train_X.shape[2]))
71
72  # convert class labels to on-hot encoding
73  train_Y = np_utils.to_categorical(train_Y)
74
75  # Reshape into (numberofsamples, channels, height, width)
76  X_train = np.reshape(X_train, (X_train.shape[0],X_train.shape[3],
        X_train.shape[1], X_train.shape[2]))
77  # convert class labels to on-hot encoding
78  y_train = np_utils.to_categorical(y_train)
79  # Reshape into (numberofsamples, channels, height, width)
80  X_test = np.reshape(X_test, (X_test.shape[0],X_test.shape[3], X_test.
        shape[1], X_test.shape[2]))
81  # convert class labels to on-hot encoding
82  y_test = np_utils.to_categorical(y_test)
83
84  input_shape = train_X[0].shape
85  C1 = 3*2
86
87  # Define the model 1
88  def model_1():
89    model = Sequential()
90
91    model.add(Conv2D(C1, (3, 3), activation='relu', input_shape=
        input_shape))
92    model.add(Conv2D(3*C1, (3, 3), activation='relu'))
93    model.add(Dropout(0.25))
94
```

```
95    model.add(Flatten())
96    model.add(Dense(6*numPCAcomponents, activation='relu'))
97    model.add(Dropout(0.5))
98    model.add(Dense(5, activation='softmax'))
99    sgd = SGD(lr=0.0001, decay=1e-6, momentum=0.9, nesterov=True)
100   model.compile(loss='categorical_crossentropy', optimizer='adam',
       metrics=['accuracy'])
101   model.summary()
102   return model
103
104 # Define the model 2
105 def model_2():
106   model = Sequential()
107
108   model.add(Conv2D(C1, (5, 3), activation='relu', input_shape=
       input_shape, data_format='channels_first'))
109   model.add(Conv2D(2*C1, (3, 3), activation='relu'))
110   model.add(Conv2D(3*C1, (3, 3), activation='relu'))
111   model.add(Dropout(0.25))
112
113   model.add(Flatten())
114   model.add(Dense(6*numPCAcomponents, activation='relu'))
115   model.add(Dropout(0.5))
116   model.add(Dense(5, activation='softmax'))
117   sgd = SGD(lr=0.0001, decay=1e-6, momentum=0.9, nesterov=True)
118   model.compile(loss='categorical_crossentropy', optimizer=sgd,
       metrics=['accuracy'])
119   model.summary()
120   return model
121
122 # Train the model and validate with test data
123 model = model_1()
124 history = model.fit(X_train, y_train, batch_size=100, epochs=10,
      validation_data=(X_test, y_test))
125
126 score = model.evaluate(X_test, y_test, verbose=0)
127 print(f'Test loss: {score[0]} / Test accuracy: {score[1]}')
128
129 # Visualize history
130 # Plot history: Loss
131 plt.plot(history.history['val_loss'])
132 plt.title('Validation loss history')
133 plt.ylabel('Loss value')
134 plt.xlabel('No. epoch')
135 plt.show()
136
137 # Model Accuracy plot
138 plt.rcParams.update({'font.size':22})
139 plt.figure(figsize=(15, 10))
140 plt.plot(history.history['accuracy'])
141 plt.plot(history.history['val_accuracy'])
142 plt.title('model accuracy')
143 plt.ylabel('accuracy')
144 plt.xlabel('epoch')
145 plt.legend(['train', 'test'], loc='upper left')
146 plt.show()
147
148 # summarize history for loss
```

```
plt.figure(figsize=(15, 10))
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

# Train the model with all training data
model.fit(train_X, train_Y, batch_size=100, epochs=20)

# Save the model
model.save('/content/drive/My Drive/thesis/training/my_model1_refVNIR'
     + str(windowSize) + 'PCA' + str(numPCAcomponents) + "testRatio" +
    str(testRatio) + '.h5')

# Train second model and save the model
model1 = model_2()
history = model1.fit(train_X, train_Y, batch_size=100, epochs=10,
    validation_data=(X_test, y_test))
model1.save('/content/drive/My Drive/thesis/training/
    my_model2_rad_VNIR1' + str(windowSize) + 'PCA' + str(
    numPCAcomponents) + "testRatio" + str(testRatio) + '.h5')
```

B.3 Testing model, validation and visualization

```
# -*- coding: utf-8 -*-
"""Reflectance VNIR testing and Visualization.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1
    RwPXhzAM_BvPQoTvp7CiEKIYvOsa4NQb
"""

# Commented out IPython magic to ensure Python compatibility.
# Import the necessary libraries
!pip install spectral
from sklearn.decomposition import PCA
import os
import scipy.io as sio
import numpy as np
from keras.models import load_model
from keras.utils import np_utils
from sklearn.metrics import classification_report, confusion_matrix,
    cohen_kappa_score
import itertools
import spectral
import matplotlib
# %matplotlib inline
from spectral import *
import spectral.io.envi as envi
from PIL import Image
import pandas as pd
from pickle import load
```

```
29
30 from google.colab import drive
31 drive.mount('/content/drive')
32
33 # Load the Global values (windowSize, numPCAcomponents, testRatio)
    from the text file global_variables.txt
34 myFile = open('/content/drive/My Drive/thesis/global_variables.txt', '
    r')
35 file = myFile.readlines()[:]
36
37
38 for line in file:
39
40     if line[0:3] == "win":
41
42         ds = line.find('=')
43         windowSize = int(line[ds+1:-1],10)
44
45     elif line[0:3] == "num":
46
47         ds = line.find('=')
48         numPCAcomponents = int(line[ds+2:-1],10)
49
50     else:
51
52         ds = line.find('=')
53         testRatio = float(line[ds+1:])
54
55 def loadDataset():
56     image = envi.open('/content/drive/My Drive/thesis/Hyperimage/
    Reflectance/VNIR/09_VNIR_ROAD_MASKED.hdr',
57                       '/content/drive/My Drive/thesis/Hyperimage/
    Reflectance/VNIR/09_VNIR_ROAD_MASKED')
58     data = image.load()
59     label_data = Image.open('/content/drive/My Drive/thesis/Hyperimage
    /Reflectance/VNIR/class4.tif')
60     labels = np.array(label_data)
61     return data, labels
62
63
64 def reports (X_test,y_test):
65     Y_pred = model.predict(X_test)
66     y_pred = np.argmax(Y_pred, axis=1)
67     #target_names = ['Road', 'Pavement', 'Vegetation', 'Train Track',
    'Water'
68      #           ,'Train Track', 'Shadows']
69     target_names = ['Road', 'Vegetation', 'Unclassified', 'Water'
70                 ,'Train Track']
71
72     classification = classification_report(np.argmax(y_test, axis=1),
    y_pred, target_names=target_names)
73     confusion = confusion_matrix(np.argmax(y_test, axis=1), y_pred)
74     score = model.evaluate(X_test, y_test, batch_size=32)
75     kappa = cohen_kappa_score(np.argmax(y_test, axis=1), y_pred)
76     Test_Loss =  score[0]*100
77     Test_accuracy = score[1]*100
78
79     return classification, confusion, Test_Loss, Test_accuracy, kappa
```

```
80
81
82 def applyPCA(X, numPCAcomponents=75):
83     newX = np.reshape(X, (-1, X.shape[2]))
84     pca = PCA(n_components=numPCAcomponents, whiten=True)
85     newX = pca.fit_transform(newX)
86     newX = np.reshape(newX, (X.shape[0],X.shape[1], numPCAcomponents))
87     return newX, pca
88
89 def Patch(data,height_index,width_index):
90     #transpose_array = data.transpose((2,0,1))
91     #print transpose_array.shape
92     height_slice = slice(height_index, height_index+PATCH_SIZE)
93     width_slice = slice(width_index, width_index+PATCH_SIZE)
94     patch = data[height_slice, width_slice, :]
95
96     return patch
97
98 def standartizeData(X, scaler):
99     newX = np.reshape(X, (-1, X.shape[2]))
100    newX = scaler.transform(newX)
101    newX = np.reshape(newX, (X.shape[0],X.shape[1],X.shape[2]))
102    return newX
103
104 def saveClassifiedData(classes):
105    with open("/content/drive/My Drive/thesis/result/radiance/VNIR/
    training_map_REF_vnir"+ ".npy", 'bw') as outfile:
106            np.save(outfile, classes)
107 def visualize_result(img, predicted_map):
108    view = imshow(img, (1, 2, 3), stretch=((0.0, 0.9), (0.0, 0.9),
    (0.0, 0.9)),figsize=(16, 16), classes=predicted_map)
109    view.set_display_mode('overlay')
110    view.class_alpha = 1
111    view.show_data
112
113 def extract_road_edges(road):
114    from skimage.feature import canny
115    from skimage.viewer import ImageViewer
116    from skimage import io
117    from skimage import img_as_uint
118    edges = canny(
119        image=road,
120        sigma=5.5,
121        low_threshold=0.1,
122        high_threshold=0.3,
123    )
124    #viewer = ImageViewer(edges)
125    #viewer.show()
126    from skimage.morphology import binary_dilation
127    edge = binary_dilation(edges, selem=None, out=None)
128    edge = binary_dilation(edge, selem=None, out=None)
129    #edge = binary_dilation(edge, selem=None, out=None)
130    road_edges = edge.astype(int)
131    return road_edges
132
133 X_test = np.load("/content/drive/My Drive/thesis/Hyperimage/
    Reflectance/VNIR/X_Patches_" + str(windowSize) + "PCA" + str(
    numPCAcomponents) + "testRatio" + str(testRatio) + ".npy")
```

```
y_test = np.load("/content/drive/My Drive/thesis/Hyperimage/Reflectance/VNIR/y_Patches_" + str(windowSize) + "PCA" + str(numPCAcomponents) + "testRatio" + str(testRatio) + ".npy")
X_test  = np.reshape(X_test, (X_test.shape[0], X_test.shape[3], X_test.shape[1], X_test.shape[2]))
y_test = np_utils.to_categorical(y_test)
model = load_model('/content/drive/My Drive/thesis/training/my_model1_refVNIR' + str(windowSize) + 'PCA' + str(numPCAcomponents) + "testRatio" + str(testRatio) + '.h5')
scaler = load(open('/content/drive/My Drive/thesis/Hyperimage/Reflectance/VNIR/spatialSize/scaler3.pkl', 'rb'))
X, y = loadDataset()
X,pca = applyPCA(X,numPCAcomponents)
height = y.shape[0]
width = y.shape[1]
PATCH_SIZE = windowSize
numPCAcomponents = numPCAcomponents
outputs = np.zeros((height,width))
for i in range(height-PATCH_SIZE+1):
    for j in range(width-PATCH_SIZE+1):
        image_patch=Patch(X,i,j)
        #print (image_patch.shape)
        X_test_image = image_patch.reshape(1,image_patch.shape[2], image_patch.shape[0],image_patch.shape[1]).astype('float32')
        prediction = (model.predict_classes(X_test_image))
        outputs[int(i+PATCH_SIZE/2)][int(j+PATCH_SIZE/2)] = prediction +1

predict_images = imshow(classes = outputs.astype(int),figsize =(16,16))

classes = outputs.astype(int)
visualize_result(X, classes)

saveClassifiedData(classes)

val_image = envi.open('/content/drive/My Drive/thesis/Hyperimage/Reflectance/VNIR/validation/08_clippped_road.hdr', '/content/drive/My Drive/thesis/Hyperimage/Reflectance/VNIR/validation/08_clippped_road')
val_img = val_image.load()
val_img.shape

X_val,pca_val = applyPCA(val_img,numPCAcomponents)
X_val = standartizeData(X_val, scaler)

height = X_val.shape[0]
width = X_val.shape[1]
PATCH_SIZE = windowSize
numPCAcomponents = numPCAcomponents
outputs = np.zeros((height,width))
for i in range(height-PATCH_SIZE+1):
    for j in range(width-PATCH_SIZE+1):
        image_patch=Patch(X_val,i,j)
        #print (image_patch.shape)
        X_test_image = image_patch.reshape(1,image_patch.shape[2], image_patch.shape[0],image_patch.shape[1]).astype('float32')
        prediction = (model.predict_classes(X_test_image))
```

```
179         outputs[int(i+PATCH_SIZE/2)][int(j+PATCH_SIZE/2)] = prediction
    +1
180
181 predict_images = imshow(classes = outputs.astype(int),figsize =(16,16)
    )
182 classes = outputs.astype(int)
183 visualize_result(X_val, classes)
184 def saveClassifiedData(classes):
185     with open("/content/drive/My Drive/thesis/result/reflectance/VNIR/
    val_map_REF_vnir11"+ ".npy", 'bw') as outfile:
186             np.save(outfile, classes)
187 saveClassifiedData(classes)
```

Norges miljø- og biovitenskapelige universitet
Noregs miljø- og biovitskapelege universitet
Norwegian University of Life Sciences

Postboks 5003
NO-1432 Ås
Norway