Improving LULC Map Production via Semantic Segmentation and Unsupervised Domain Adaptation

by

Rostyslav-Mykola Tsenov

A Thesis submitted to the Faculty of Graduate Studies of
The University of Winnipeg
in partial fulfillment of the requirements of the degree of

Master of Science

Department of Applied Computer Science
University of Winnipeg
Winnipeg, Manitoba, Canada

Improving LULC Map Production via Semantic Segmentation and Unsupervised Domain Adaptation

by

Rostyslav-Mykola Tsenov

Supervisory Committee

Dr. C. J. Henry, Surpervisor
(Department of Applied Computer Science)

Dr. S. Ramanna, Member
(Department of Applied Computer Science)

Dr. J. J. Li, External Member
(Department of Geography and Environmental Management, University of Waterloo)

Abstract

In recent years, a lot of remote sensing problems benefited from the improvements made in deep learning. In particular, deep learning semantic segmentation algorithms have provided improved frameworks for the automated production of land-use and land-cover (LULC) map generation. Automation of LULC map production can significantly increase its production frequency, which provides a great benefit to areas such as natural resource management, wildlife habitat protection, urban expansion, damage delineation, *etc.* In this thesis, many different convolutional neural networks (CNN) were examined in combination with various state-of-the-art semantic segmentation methods and extensions to improve the accuracy of predicted LULC maps. Most of the experiments were carried out using Landsat 5/7 and Landsat 8 satellite images. Additionally, unsupervised domain adaption (UDA) architectures were explored to transfer knowledge extracted from a labelled Landsat 8 dataset to unlabelled Sentinel-2 satellite images. The performance of various CNN and extension combinations were carefully assessed, where VGGNet with an output stride of 4, and modified U-Net architecture provided the best results. Additionally, an expanded analysis of the generated LULC maps for various sensors was provided. The contributions of this thesis are accurate automated LULC maps predictions that achieved 92.4% of accuracy using deep neural networks; production of the model trained on the larger area, which is six times the size from the previous work, for both 8-bit Landsat 5/7, and 16-bit Landsat 8 sensors; and generation of the network architecture to produce LULC maps for the unlabelled 12-bit Sentinel 2 data with the knowledge extracted from the labelled Landsat 8 data.

Keywords: Land use and land cover (LULC), deep learning (DL), semantic segmentation, convolutional neural network (CNN), remote sensing, satellite images.

Acknowledgment

I would like to extend my deepest gratitude to my supervisor Dr. Christopher Henry for the provided encouragement, patience, and invaluable contribution throughout the duration of my Master's program. I also cannot begin to express my thanks to Drs. Christopher Storie and Joni Storie (University of Winnipeg, Geography Dept.), whose help for my thesis cannot be overestimated, they were pivotal in the remote sensing, data acquisition, preprocessing, and analysis. I am also grateful to Dr. Yangjun Chen, my first supervisor, for allowing me to start the study at the University of Winnipeg.

I would like to express my deepest appreciation to my supervisory committee, Dr. Sheela Ramanna and Dr. Jonathan Li for the constructive criticism and helpful advice. I wish to thank all the people whose assistance was a milestone in the completion of this project, especially my former instructor Dr. Simon Liao, and all the valued members of the Applied Computer Science Department and the Faculty of Graduate Studies.

Thanks should also go to Manitoba Hydro and Mitacs for support and funding of this work, which would not have been successful without it. I would like to acknowledge the assistance and help of every person I met during my journey at the University of Winnipeg and in the Mitacs program.

Lastly, I would also like to extend my gratitude to my family and friends for the irreplaceable support throughout the study.

Dedication

In dedication to my family.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Land use and land cover maps (LULC) are products generated from raw satellite imagery where each pixel of the image is assigned to some sort of class or label (*e.g.* water, forest, road, cropland, *etc.*). Land use refers to the purpose the land serves, like agriculture, wildlife habitat or any other synthetic area, and land cover refers to the surface cover on the ground, like water, forest, urban infrastructures, vegetation. LULC maps are crucial in areas such as natural resource management, wildlife habitat protection, urban expansion, damage delineation, *etc.* [1]. In the beginning, LULC maps were generated manually using semi-automated techniques. In general LULC maps are produced by combining different colour channels (also known as *bands*) of a given satellite image (called the *composition*). There are a huge number of different sensors that exist and are used in remote sensing. Examples include Landsat, Sentinel, RADAR, LiDAR, Quickbird, Worldview, and GeoEye. And each sensor's data structure might differ. For example, the number of bands, spatial resolution (square area on the ground in meters each pixel represents), or spectral resolution (the magnitude of light energy emitted by an object on the ground measured at a particular wavelength). Conventional methods of generating LULC maps take a lot of time and effort to produce adequate results. Therefore, automation of LULC map production can remarkably contribute to the mentioned areas by increasing the production frequency of LULC maps. Currently, there are plenty of different automated ways to create LULC maps by using advanced algorithms, machine learning tools [2], or deep learning (DL) [3], which started growing in popularity in the last decade. DL found immense success in remote sensing by providing fast and consistent human-like performance and solving numerous problems that had not been solved by conventional methods [3]. Also, before the DL, LULC maps were developed using supervised, semi-supervised, and unsupervised methods and algorithms (K-means Clustering, Decision Tree and Maximum Likelihood Classier) [2]. However, these methods are error-prone, lack consistency, and require a substantial amount

of data and user input to provide satisfactory results.

DL has become a hot topic in recent years, and new groundbreaking papers are getting released every few months, where examples include image processing [4], computer vision [5], and object detection [6]. Most problems in deep learning applications are solved using convolutional neural networks (CNN), which is a kind of neural network (NN) that produces results equal to or better than humans for certain applications. A CNN is currently considered as a state-of-the-art method for computer vision tasks, such as image classification and object recognition [7]. Remote sensing related tasks have also benefited from DL in the last couple of years [3] as their tasks are very similar to the ones solved by CNNs.

The main problem considered in this thesis is LULC map generation, which directly maps to the semantic segmentation computer vision problem. To resolve this task, [8] [9] introduced an *encoder-decoder* architecture, which extracts features from the images using CNN in the form of an encoder and then decodes those features back to the original size using a reversed CNN. Also, encoder-decoder networks were implemented in remote sensing [10] [11] [12] [13] [14], where they reached exceptional success by providing accurate and consistent results. In this thesis, we aim to improve LULC map production. This work takes two forms. First, we improve the results presented in [15] by incorporating new and modified CNNs [16] [17] [18] [19] to the existing architecture, and adding more advanced model and architectural extensions [20] [21]. For this problem, we used 8-bit data from Landsat 5/7, and 16-bit data from Landsat 8 sensors, where both of them have a spatial and spatial resolution of 30m x 30m per pixel. Data from these two sensors have a similar structure, and they all have red, green, blue (RGB) and multiple infrared bands.

Next, we investigate the production of LULC maps for scenarios where a new sensor is considered, but where there is no labelled data. For instance, the results presented in the first part of this thesis are quite powerful, but they require a large amount of labelled data, which is not always possible or practical to produce. One way to resolve this is to transfer knowledge from existing labelled datasets to similar unlabelled ones, which was the main motivation for creating *unsupervised domain adaption* (UDA) learning frameworks [22]. It is this approach that was used UDA to generate LULC maps on sensors without corresponding labelled data. For this problem, we used 16-bit data from Landsat 8 sensor with a spatial resolution of 30m x 30m per pixel, and 12-bit data from Sentinel-2 sensors with a spatial resolution of 10m x 10m per pixel for RGB bands and 20m x 20m for infrared bands.

1.1 Problem Definition

The work in [12][13] modified the fully convolutional network (FCN) [8] and was introduced to generate LULC maps based on Landsat 5/7 data. The work was then improved in [15] by introducing context module extension [23] and adversarial training [24], which reached a global accuracy of 90.46%. In this thesis, we aim to improve the performance of the model by introducing new state-of-the-art models, extensions and architectural designs. Additionally, we consider training models on the North American Land Change Monitoring System (NALCMS)[1] dataset that covers six times the area from the previous experiments, and we consider both 8-bit Landsat 5/7 and 16-bit Landsat 8 data. Moreover, a UDA architecture is introduced to transfer knowledge extracted from a labelled Landsat 8 dataset, and apply it on an unlabeled Sentinel-2 dataset to generate LULC.

1.2 Proposed Approach

To create a LULC map using DL, an encoder-decoder based network should be used. Work in this thesis is built upon [15], where we similarly used the same encoders (VGGNet [25], Googlenet [26], ResNet [27]) with the FCN-8 based structure [8]. New Xception [17] CNN was introduced, and the ResNet model was modified to use a full-preactivation design [16]. Additionally, U-Net [19] and Feedbackward [18] decoders were implemented for the VGGNet encoder, and then a generic version of the models that performed best was designed to fit all encoders. To improve the performance of predictions even further, many model and architectural extensions were introduced.

- Context module - This extension is placed at the end of the network, and it aggregates multi-scale contextual information of the prediction [23].
- Output stride (OS) - This extension represents the desirable size of the ratio the output feature map of the encoder [28].
- Atrous spatial pyramid pooling (ASPP) - This extension is placed in between encoder and decoder and aims to capture objects and context of the features at multiple scales [29].
- Deeplab - This is an architectural extension that focuses on using output stride and ASPP model extensions with a custom decoder design [20].

[1] http://www.cec.org/north-american-land-change-monitoring-system/

- Generative adversarial network (GAN) - This is an architectural extension that introduces an additional NN called discriminator, which tries to distinguish what images were generated by the encoder-decoder network [24].
- Progressive GAN - This is an architectural design that is an advanced type of GAN, which gradually increases the size of the output images, thus providing more stability to predictions [21].

Throughout an experiment, the best model variation of CNNs and extensions were observed, and then the product for Landsat 5/7 and Landsat 8 was generated. The performance of the mentioned products is assessed and compared. Lastly, the UDA learning framework [30] was used to generate a map product from Sentinel-2 by transferring knowledge using a conditional generative adversarial network [31].

1.3 Contribution of the Thesis

The contribution of this thesis is described below.

- Achieving 92.4% on NALCMS on the southern extent of Manitoba dataset by modifying encoder-decoder network structure, introducing new state-of-the-art model extensions, and architectural designs.
- Generating and comparing the performance between networks trained on a small dataset (equivalent to the size of datasets used in previous work) as well as a dataset six times the size.
- Analyzing networks trained on the 8-bit (Landsat 5/7) and 16-bit (Landsat 8) datasets.
- Generating an architectural design for a network that can produce LULC maps from unlabelled Sentinel-2 data using only labelled Landsat 8 data.

1.4 Organization of the Thesis

The remainder of this thesis is organized as follows.

- **Chapter 2** introduces the notion of a NN to the reader and the process of its training. Additionally, the structure of the CNNs used in this work are defined.
- **Chapter 3** reviews work from the DL, remote sensing, and other related fields.

- **Chapter 4** discusses the in-depth structure of CNNs, different model extensions, and architectural designs used in this thesis.
- **Chapter 5** introduces the datasets used to generate the presented results as well as their characteristics. Dataset augmentation and transformation, experimental setup, and implementation details are also discussed.
- **Chapter 6** presents an analysis of the results of different network implementations. This chapter also provides an analysis of the model performance trained on several larger datasets with different characteristics.
- **Chapter 7** sums up the work done in the thesis and discusses possible directions for future experiments.

Chapter 2

Methodology

This section introduces basic and in-depth information about NNs. In the first place, the backbone structure of a NN is explained, like the neurons and what they contain, and the way weights are generated and optimized. Next, DL is discussed by starting with CNN and their uses and then it is discussed in the context of different DL architectures that are commonly labelled deep neural networks (DNN).

2.1 Neural Network

Artificial intelligence (AI) is a field of computer science that is involved in building tools and machines that are capable of completing tasks typically resolved by humans. One of the most successful and popular ways to generate AI tools is to utilize machine learning (ML) algorithms, where ML is a subfield of AI. ML algorithms use statistics to find patterns present in provided data. There are three popular ways to create an algorithm in the field of ML.

- Supervised learning – This is a process where the algorithms are developed using a priori data provided by experts.
- Unsupervised learning – This is a process where the algorithms identify patterns of interest without any input from experts.
- Semi-supervised learning – This learning method is a combination of both supervised and unsupervised learning. The algorithm is developed with a combination of information provided by experts, and data for which nothing is known.

For the rest of this chapter we focus on supervised learning techniques.

Inspiration for the first artificial neural networks (ANN) [32] came from the neuronal structure of the human brain, which consists of billions of connected neurons. A neuron is a specialized nerve cell that receives many signals as input and processes them, and neurons also send signals between the body and the brain. An artificial neuron is called a *perceptron*, as depicted in Fig. 2.1, and an ANN is a large number of connected *perceptrons* (see *e.g.* Fig. 2.2). For the remainder of this thesis, an artificial neuron is simply called a neuron and an ANN is simplified to a NN.

2.1.1 Perceptron

The structure of a perceptron is more simplified than a neuron in an actual human brain [33] (see *e.g.* Fig. 2.1). The input to a perceptron is a vector $\vec{X} = (x_0, x_1, \ldots, x_n)$, and, for each input value x_i, there is a corresponding weight w_i from the vector $\vec{W} = (w_0, w_1, \ldots, w_n)$. A perceptron with input $\vec{X}$ and weights $\vec{W}$ is modeled as

$$y = \sum_{i=0}^{n} w_i * x_i. \tag{2.1}$$

Also, each perceptron has a bias b, which shifts the result from the origin, and therefore gives more flexibility to the output. For example, if there is no bias in the perceptron and the input is $\vec{X} = (0, 0, \ldots, 0)$, the output will be 0 no matter the value of the weight. A perceptron with input $\vec{X}$, weights $\vec{W}$ and bias b is modelled as

$$y = \sum_{i=0}^{n} w_i * x_i + b. \tag{2.2}$$

Lastly, the output is passed through an activation function, which determines whether to activate the neuron or not [34]. There are many different activation functions, but to keep the example simple, a step activation function is shown (also known as a binary activation function). The step activation function is defined as

$$f(y) = \begin{cases} 1, & \text{if } y > 0, \\ 0, & \text{otherwise.} \end{cases} \tag{2.3}$$

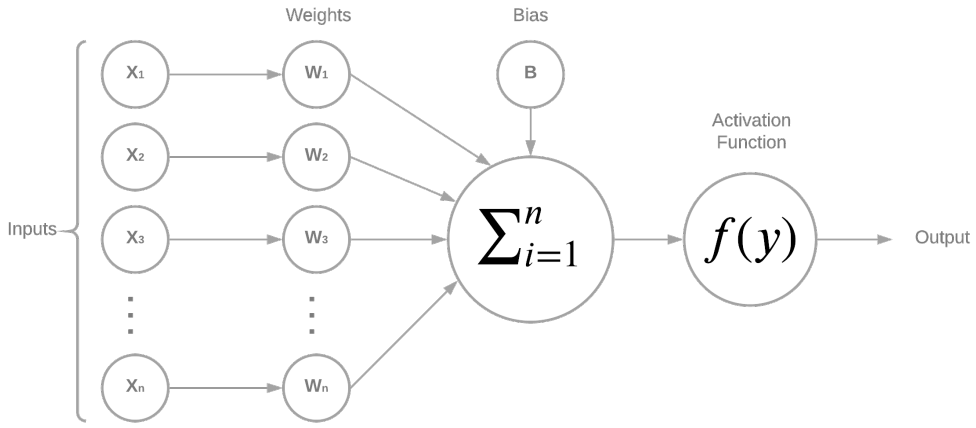


Figure 2.1: Structure of a perceptron.

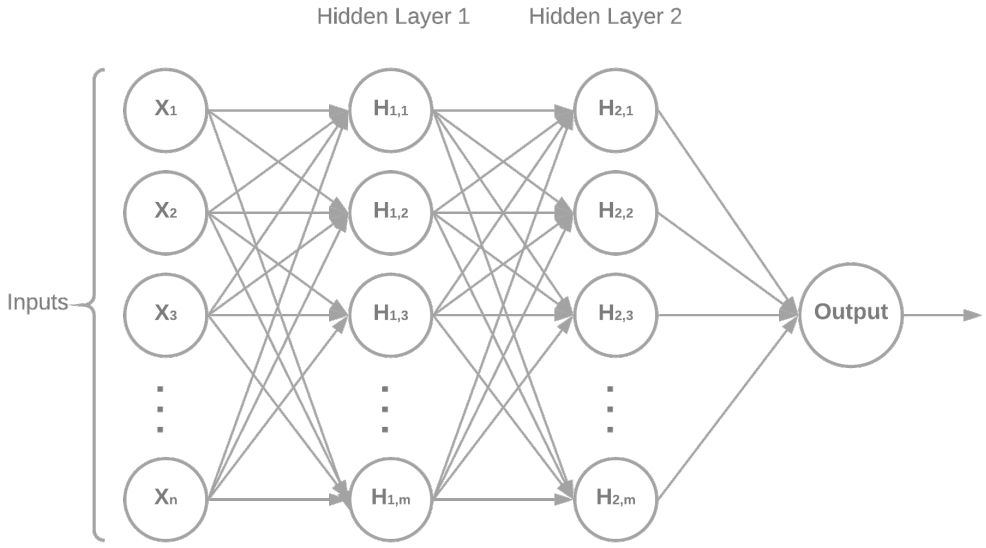


Figure 2.2: Structure of simple neural network with inputs $\vec{X} = (x_0, x_1, \ldots, x_n)$ and 2 hidden layers.

2.1.2 Interpretation of Gradient

A gradient is the collection of partial derivatives of a function $f(\vec{X})$ denoted as $\Delta f(\vec{X})$, where $\vec{X}$ is a vector of inputs [35]. Gradients are highly used in NN algorithms, especially in *backpropagation* and *optimization* algorithms, further described in Sections 2.1.4 and 2.1.7. Imagine that we have a simple multiplication function of two inputs $f(x, y) = xy$, and partial derivatives for both inputs are formulated

as

$$f(x,y) = xy \quad \rightarrow \quad \frac{\partial f}{\partial x} = y, \quad \frac{\partial f}{\partial y} = x. \tag{2.4}$$

Partial derivatives represent the pace of change of a function concerning the variables surrounding an immeasurably small region near a specific point [35], formulated as

$$\frac{\partial f(x)}{\partial x} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}, \tag{2.5}$$

where operator $\frac{d}{dx}$ is applied on the function f and returns the derivative, and h is a number close 0 to represent the function as a straight line. Also, the vector of partial derivatives (gradient) Δf is represented as $\nabla f = [\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}] = [y, x]$.

As another example, considered the composed expression $f(x,y,z) = (x+y)z$. This function can be differentiated by splitting it into two functions $q = x + y$ and $f = qz$. This composed function now becomes an multiplication function $f = qz$ and gradient of this function f is calculated with respect to its inputs x, y, z, as $\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$ [35]. Combining these gradient expressions through multiplication is known as chain rule, which is highly used in backpropagation (see *e.g.* Section 2.1.4).

2.1.3 Loss function

Training a neuron is the process of adjusting the weights of a perceptron to produce specific outputs. In this case, some sort of signal is used to incrementally adjust the weights until the desired output is produced. Typically, this signal is in the form of a loss function that assesses the error produced by the NN. Based on this value, weights are adjusted to minimize the error. This process is further described in Sections 2.1.4 and 2.1.7. There are many different loss functions that can be used for this task. For example, the most common are described as follows.

- Mean squared error (MSE) – This method is commonly used in regression tasks [36]. Calculated for all inputs n, by the average of the squared difference between actual (ground truth) y_i and predicted values $\hat{y}_i$, computed as

$$MSE = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}. \tag{2.6}$$

- Cross-Entropy Loss – This method is commonly used in a large number of classification problems [37]. A separate loss is calculated for each class label, where m is the total number of classes. Loss values increase as the prediction diverges from the actual label, computed as

$$CrossEntropyLoss = -\sum_{i=1}^{m} y_i \ln(\hat{y}_i). \tag{2.7}$$

2.1.4 Backpropagation

NNs are a supervised learning algorithm, which means there is a correct answer for each input, and, based on the loss value, we assess how close the output of the NN is to the corresponding label. For this task, the gradient of the loss function is calculated with respect to all the weights in a NN, which is called backward propagation of errors or *backpropagation* [38]. To describe how it works, imagine that we have a NN consisting of a vector of layers $\vec{L} = (l_0, l_1, \ldots, l_k)$ and for each corresponding layer l_i there are weights $\vec{W} = (w_0, w_1, \ldots, w_k)$ and input $\vec{X} = (x_0, x_1, \ldots, x_k)$ with activation result $\vec{A} = (a_0, a_1, \ldots, a_k)$. For each layer l_i, the output is calculated as

$$a_i = f(w_i * x_i + b_i), \tag{2.8}$$

where f is activation function and b_i is bias of the current layer l_i. Also, if $i \neq 1$, then $x_i = a_{i-1}$. After reaching the last layer k, we compute the loss function C, which gives the output error for the last layer as

$$\delta_k = \frac{\partial C}{\partial a_k} f'(w_k * x_k + b_k). \tag{2.9}$$

Then, the error for each layer is calculated backwards as

$$\delta_i = ((w_{i+1})^T \delta_{i+1}) \odot f'(w_i * x_i + b_i), \tag{2.10}$$

where $(w_{i+1})^T$ are the transposed weights from the layer l_{i+1} and $\odot$ is a dot product. Having errors for each weight in every layer, we can adjust them in the direction of the gradient to reduce the overall error of the network.

2.1.5 Initialization

As discussed above, each perceptron has weights that can be adjusted to produce specific outputs. The process of adjusting the weights of a perceptron is called *training*. These weights need to be initialized prior to training. There are plenty of different initialization techniques, but the simplest ones are *zero*

initialization, which sets each weight value to zero, and *random initialization*, which generates a random value for each weight. However, using these simple initialization techniques can lead to poor results. For example, initializing a weight to a very high value will likely force the activation value to 1. Similarly, an initial value of 0 will cause the activation function to output 0. [39].

To fix the issue described above, "normalized initialization" was introduced by Bengio and Glorot [40], also known as *Xavier initialization*. This method generates values just like random initialization, but distributed in the following range $\left[-\frac{\sqrt{6}}{\sqrt{n_i+n_{i+1}}}, \frac{\sqrt{6}}{\sqrt{n_i+n_{i+1}}}\right]$, where n_i is the number of inputs to the current layer, and n_{i+1} is the number of outputs from the current layer.

2.1.6 Input Normalization

Input normalization is a well-known method used to bring every input value $\vec{X} = (x_0, x_1, \ldots, x_n)$ to a similar range, to omit any significant differences in the input value range. For example, consider the inputs x_1 and x_2 in the intervals [0,1] and [0, 0.01], respectively. In this case, the parameters correlated with each input will also exist on different scales due to the NN combining these inputs through a series of linear combinations and nonlinear activations. A lot of different input normalization methods exist, but the following are used the most often.

- Zero to one normalization – This method casts the input value x_i to the new value z_i in range $[0, 1]$, and is computed as
$$z_i = \frac{x_i - min(X)}{max(X) - min(X)}. \tag{2.11}$$
- Negative one to one normalization – This method is similar to the previous normalization, but casts the input value x_i to the new value z_i in range $[-1, 1]$ instead of $[0, 1]$, and is computed as
$$z_i = 2 * \frac{x_i - min(X)}{max(X) - min(X)} - 1. \tag{2.12}$$
- Mean normalization – Instead of casting the input values to the specific range, this method tries to center them around zero, and is computed as
$$z_i = x_i - mean(X). \tag{2.13}$$

2.1.7 Optimization

Optimization is the process of adjusting NN weights to minimize the error value produced by the loss function. There are a lot of different existing methods for this task, but *gradient descent* is the

most popular optimization algorithm [41]. Gradient descent is an iterative algorithm that focuses on finding a local minimum of a specific function by taking steps in the direction of the negative of the gradient at the given step. Also, there are many gradient descent variants, like *Batch Gradient Descent*, *Stochastic Gradient Descent* and *Mini-Batch Gradient Descent* [42] [43]. Each variation has a trade-off by increasing the accuracy of weights or reducing the time it takes to train.

After the backpropagation step described in the subsection above, we receive derivatives for each layer, and we need to optimize the weights and biases. The optimization of weights and biases can be formulated as

$$\theta_{j+1} = \theta_j - \frac{\eta}{m} \sum_{j=1}^{m} \delta_{i,j} (a_{i-1,j})^T, \tag{2.14}$$

where θ_j are the weights $\vec{W}$ at iteration step j, θ_{j+1} is the adjusted weights at the next iteration step, η is the *learning rate* of the optimization algorithm, and m is the number of the training examples in the current iteration (also referenced as *batch size*). Also, an *epoch* is defined as a full pass of all the training samples through the machine learning algorithm. To make this algorithm work iteratively, repeat Eq. (2.14) n amount of times, where n is the number of iterations.

The optimization algorithm introduced in Eq. (2.14) is the simplest of all, known as stochastic gradient descent (SGD) [42]. There are more advanced and complex optimization algorithms, for example, momentum, root mean square propagation (RMSprop), adaptive gradient algorithm (AdaGrad) and adaptive moment estimation (Adam) [44][45][46]. Currently, *Adam* is the most used optimization algorithm, and the main difference from SGD is in the adaptive per-parameter learning rates and in calculating the exponential average of the gradient and the squared gradient, which decreases the time needed for training.

The Adam method stores an exponentially decaying average of past squared gradients v_j and keeps an exponentially decaying average of past gradients m_j, where j is the current iteration step. v_j and m_j are calculated as

$$m_j = \beta_1 m_{j-1} + (1 - \beta_1)\delta_j, \tag{2.15}$$

$$v_j = \beta_2 v_{j-1} + (1 - \beta_2)\delta_j^2, \tag{2.16}$$

where β_1 and β_2 are values close to 1. Also, the bias-corrected moment estimates are calculated as

$$\hat{m}_j = \frac{m_j}{1 - \beta_1^j}, \tag{2.17}$$

$$\hat{v_j} = \frac{v_j}{1 - \beta_2^j}. \tag{2.18}$$

Then, the parameters are updated as

$$\theta_{j+1} = \theta_j - \eta \frac{\hat{m}_j}{\sqrt{\hat{v_j}} + \epsilon}. \tag{2.19}$$

The most used values for β_1 is 0.9, β_2 is 0.999 and for ϵ is 10^{-8}.

2.2 Convolutional Neural Network

A CNN is very similar to the structure of a regular NN as it also has inputs, trainable weights, and biases. CNNs are usually used to solve computer vision-related tasks, like image classification. The difference is the presence of three-dimensional layers, which have width, height and depth, like in the structure of the regular image, where depth refers to the third dimension of an activation volume and not the depth of a full NN, which can refer the total number of layers in a network [35] (see *e.g.* Fig. 2.3). Input images to the NN leads to some changes, like convolution and dot products used instead of weight multiplication, pooling layers used for reducing the width and height of the layers the deeper they are in the CNN, different activation functions, and much more.

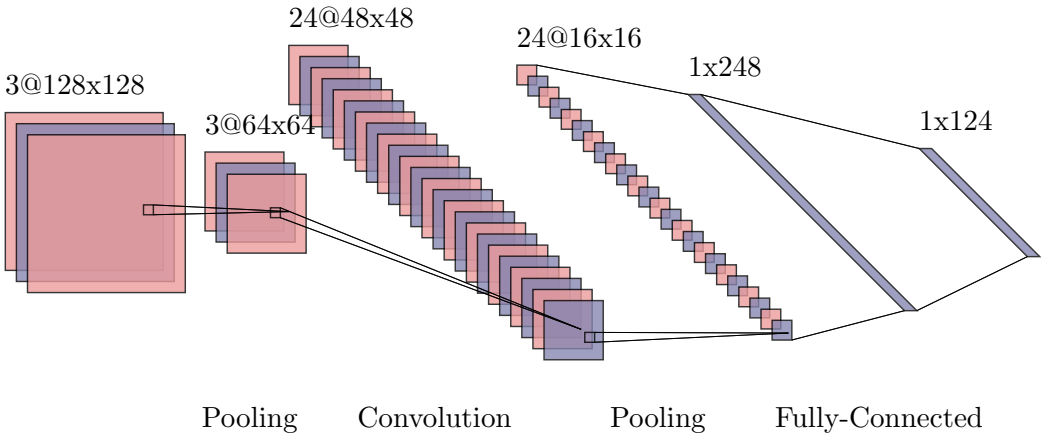


Figure 2.3: Regular CNN structure for 128 x 128 image and 3 colour channels with convolutional layer, pooling layers and fully connected layer.

2.2.1 Convolutional Layer

The convolutional layer is a crucial part of the structure of CNNs and, based on its name, uses the convolution operation [47]. The convolution operation is used a lot in image processing-related tasks

(see *e.g.* Fig. 2.4). CNNs also have weights, but, unlike simple NNs, it stores them in the form of small matrices called *filters*. The size of each is KxK, and each convolution layer consists of D amount of filters. The total amount of weights per convolutional layer can be calculated as $K * K * D$ with D biases. The size of the input to the convolutional layer is WxH, where W is the width, and H is the height. A 3x3 filter size is the most common, but sometimes different sizes are used. Note, that filter dimensions should be smaller than the input. Also, D usually represents the number of activations masks or the depth of the convolutional layer. The convolutional layer does not look at the whole input at one step, it rather looks at input n amount of steps through a small window of the size KxK called the *receptive field*. Define input to the convolutional layer as $\vec{X} = (X_0, X_1, X_2, ..., X_n)$, where $X_i = (x_0, x_1, x_2, ..., x_{K*K})$ are values received from the input through receptive field at step i. The convolution operation is formulated as

$$z_i = X_i * f, \tag{2.20}$$

where z_i is the output feature map and f is the filter. As was mentioned above, multiplication in this layer is replaced with the convolutional operation of filters on input.

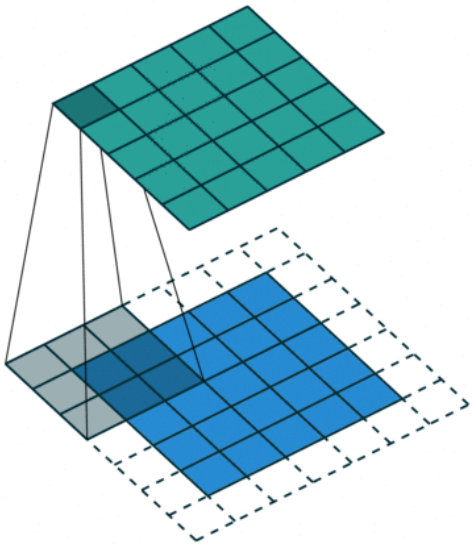


Figure 2.4: Convolution with a filter size of 3x3 and zero-padding $P = 1$ [48], where the blue grid is input, and the green one is the output of the convolutional layer.

Filters usually slide through each input value, but sometimes striding S is used to lower the size of output by skipping pixels. When $S = 1$, the filter will slide through each value, and when $S = 2$ or $S = 4$, the filter skips every next or every two values for each slide. Moreover, applying filters on the images will give smaller output, even when $S = 1$, this happens when applying filters on the border of

the input [35]. To produce an output the same size as the input, zero-padding P is used, which fills the input volume with zeros around the border. The actual output size is calculated as

$$(H_o|W_o) = \frac{((H|W) + 2P - K)}{S} + 1, \tag{2.21}$$

where H_o is the height and W_o is the width of the output features.

Atrous Convolution

There are more than one type of convolution used in CNNs. One example is *atrous convolution*, which is also known as *dilated convolution* [49]. The idea behind dilated convolution is to increase the perimeter of the filter without increasing the number of parameters. This convolution is the same as the regular one but has a dilation rate l, shown in Fig. 2.5. The dilation rate corresponds to the number of spaces inserted between kernel elements. The atrous convolution is calculated as

$$z_i = X_i *_l f. \tag{2.22}$$

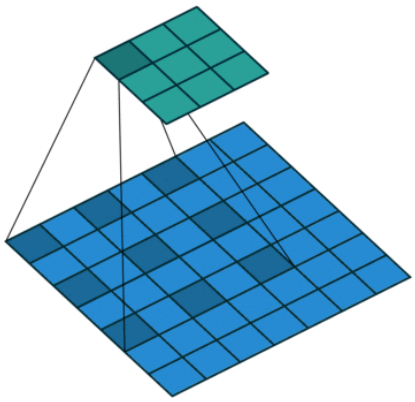


Figure 2.5: Atrous convolution with a filter size of 3x3 and dilation rate $l = 2$ [48], where the blue grid is input, and the green one is the output of the convolutional layer.

Transposed Convolution

Transposed convolution, also known as deconvolution, is a regular convolution operation, but instead of lowering the output size of the features, it tries to recover the original size of the input [50], shown in Fig. 2.6. This convolution is highly used in *autoencoder* and *encoder-decoder* structures [51][8]. The result is achieved by increasing the size of the input by inserting zeros between feature elements, called striding.

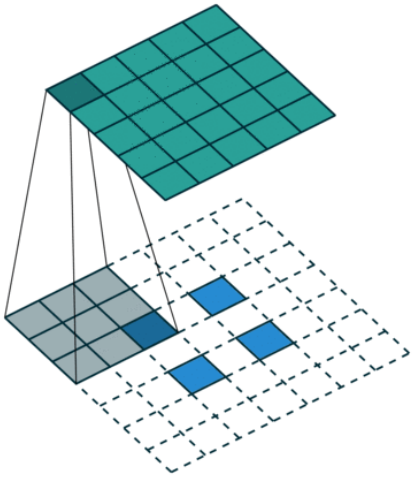


Figure 2.6: Transposed convolution with a filter size of 3x3 and striding rate is equal to 2 [48], where the blue grid is input, and the green one is the output of the convolutional layer.

Depthwise Separable Convolution

Depthwise separable convolution is a recent and slightly different operation than original convolution [17]. Regular convolution executes the channel and spatial-wise calculation in a single operation, while depthwise separable convolution performs two different operations, shown in Fig. 2.7. In the beginning, depthwise convolution is used on each input channel, then pointwise convolution is applied to generate a combination of the results from the depthwise convolution. Where depthwise convolution uses a single convolutional filter for each input channel and tries to keep each channel separate. And pointwise convolution is regular 1x1 convolution, which iterates through every single point.

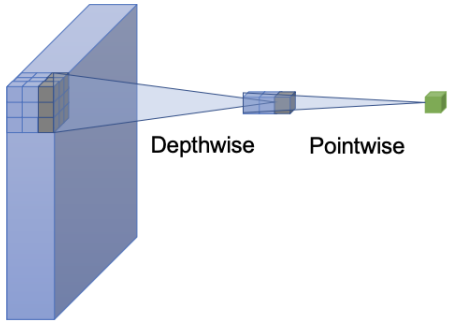


Figure 2.7: Depthwise separable convolution with filter size of 3x3 followed by a pointwise convolution [17], where input is depicted in blue on the left.

2.2.2 Pooling Layer

The pooling layer, also known as downsampling, is usually used in between convolutional layers or entire blocks. The purpose of these layers is to lower the spatial size of the features to reduce the number of parameters in the following layers. Downsampling is similar to the convolution operation, which uses input with dimensions WxHxD, and a filter window of size K and striding S to generate new output. There are many different pooling methods, some of which are the following.

- Max pooling – This method looks at the most active feature in a pooling region X_i in $\vec{X} = (X_0, X_1, X_2, ..., X_n)$. The output size is determined by the striding value S, calculated as
$$z_i = max(X_i). \tag{2.23}$$
- Average pooling – This method takes all values in each region X_i in X. The output size is determined by the striding value S, calculated as
$$z_i = avg(X_i). \tag{2.24}$$
- Global pooling – This method takes all values from the input X and performs max pooling or average pooling. The output from the operation is a single value [52], calculated as
$$z = global\_pool(X). \tag{2.25}$$

2.2.3 Upsampling Layer

The upsampling layer does the opposite of the pooling layer. Instead of lowering the spatial size of the features, it tries to recover the original size and its mostly used in autoencoder and encoder-decoder type architectures [51][8], which mimics the job of transposed convolution. Upsampling is usually in-between the blocks, which is followed by at least one convolutional layer. Most of the time, transposed convolution performs better for the problem of upsampling where the spatial size of the features should be increased. This is due to the presence of trainable parameters, but in other cases, upsampling is a better option as it is not involved in the training and does not have trainable weights and biases, thus decreasing computation time. There are several upsampling layers used in NNs.

- Nearest neighbour upsampling – This method is based on the nearest neighbour interpolation [53], the simplest of the three, when increasing the size of features it recovers missing values by replacing them with the nearest one.

- Bilinear upsampling – This method is based on the bilinear interpolation [54], it determines missing values by replacing them with a weighted average of the four nearest values. The values closer to the missing one has higher weights.

- Bicubic upsampling – This method is based on the bicubic interpolation [55], very similar to the bilinear interpolation, but it determines missing values by replacing them with a weighted average of the sixteen nearest values. As in the bilinear interpolation, the values closer to the missing one have higher weights. This one is the most computationally expensive of the three.

2.2.4 Activation Layer

Activation functions are usually used after applying weights and biases on the input using simple operations, as was described in Subsection 2.1.1 or using convolution. The purpose of the activation function is to help NN learn the complex patterns present in the input by determining what data to activate and pass to the next neuron using a non-linear transformation. If activation functions were not present, the NN would be just a regular linear regression model. Many different activation functions exist, for example, the step function was described in Subsection 2.1.1. However, the following functions are used by the CNNs in this thesis.

- Linear function – The idea behind this method is to generate a signal proportional to the input, by multiplying a constant value c on each value of output y. Linear functions are a better choice than a step function, but they are not usable with backpropagation since all layers of the NN end up producing the same output [34]. The linear function computed as

$$f(y) = cy. \tag{2.26}$$

- Sigmoid function – This method is a non-linear function that focuses on smoothing the gradient of output y and bounds the values between 0 and 1. However, this function is computationally expensive, and gradients can vanish if the values are very high or low [34]. The sigmoid function is computed as

$$f(y) = sigmoid(y) = \frac{1}{1 + e^{-y}}. \tag{2.27}$$

- Tanh function – This method is a non-linear function which is extremely similar to the sigmoid, but it bounds values between -1 and 1 instead of 0 and 1 [34]. The tanh function computed as

$$f(y) = tanh(y) = \frac{2}{1 + e^{-2y}} - 1. \tag{2.28}$$

- Rectified linear unit (ReLU) function – This method is one of the most used activation functions because of its computation efficiency. However, the gradient sometimes becomes zero when y is negative or close to zero, which causes problems with backpropagations. The values are bound between 0 and ∞ [34]. The ReLU function computed as
$$f(y) = max(0, y). \tag{2.29}$$
- Leaky ReLU function – This function is a modified version of the ReLU that includes a hyper-parameter λ. Here, λ gives a little slope for negative values instead of bounding them to 0. Even though this seems like a better version of ReLU, leaky ReLU does not provide consistent results for negative input values [34]. The leaky ReLU function computed as
$$f(y) = max(\lambda y, y). \tag{2.30}$$

2.2.5 Batch Normalization Layer

Sometimes after training a NN on a specific dataset, the model might *overfit*, which means that the trained NN performs well on the dataset used for training but not well on the validation dataset. This is called underfitting, where the model does not generalize to examples outside of the training set [56]. To prevent these problems, a *dropout* layer is used, which drops a unit with connection during training with a specified probability p. However, recently more advanced methods were developed, like the *Batch Normalization* (BN) layer [57], which negates the use of the dropout layer and accelerates the training by fixing the means and variances of the layer input. Consider a mini-batch $\mathcal{B}$ which consists of inputs $\vec{X} = (x_1, x_2, ..., x_m)$, normalized inputs $\vec{\hat{X}} = (\hat{x_1}, \hat{x_2}, ..., \hat{x_m})$, and $\vec{Y} = (y_1, y_2, ..., y_m)$ their linear transformation. Then, mini-batch mean $\mu_{\mathcal{B}}$ and variance $\sigma^2_{\mathcal{B}}$ are calculated as

$$\mu_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m} x_i, \tag{2.31}$$

$$\sigma^2_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2. \tag{2.32}$$

Following normalized inputs, $\hat{x}_i$ is computed as

$$\hat{x}_i = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}}. \tag{2.33}$$

Then, the output scaled and shifted y_i, before passing to the new layer, is calculated as

$$y_i = \gamma\hat{x}_i + \beta = \mathrm{BN}_{\gamma,\beta}(x_i), \tag{2.34}$$

where γ and β are learnable parameters.

2.2.6 Fully Connected Layer

A *fully connected layer* is a layer in a neural network where each neuron in the layer is connected to all outputs from the previous layer. These layers are most often placed at the end of a CNN, where they flatten the output from the previous layer into a one-dimensional array, and typically apply more weights and biases before providing a final classification decision.

2.3 Chapter Summary

This chapter introduced the notion of NNs and their structure. Algorithms and methods that are used for their training are discussed, like loss function, backpropagation, and optimization. Additionally, the structure of CNNs were presented with the methodology for the layers. Layers and their purpose were explained, in particular, the convolutional, downsampling, upsampling, batch normalization, and fully connected layers.

Chapter 3

Literature Review on the Use of Deep Learning in Semantic Segmentation and Remote Sensing

This chapter introduces DL with some highly used architectures designed for different tasks, like successful CNN architectures, which are used for image classification. Then, more advanced DNN architectures are discussed for use in remote sensing and LULC map production, examples include encoder-decoder models. Finally, works of different NN extensions are introduced as well as the UDA architecture.

3.1 Deep Learning

In the last two decades, the popularity of NNs and DL has significantly increased due to a large number of advances in many research areas [58]. Also, improvements in graphics processing unit (GPU) technology were a big part of the success of DL. Models, which previously took months or weeks to train, now took just a few hours or days. This acceleration in training made it possible to create a DNN with large numbers of layers and, together with GPUs, made them accessible to the public. DL surpassed many state-of-the-art methods in fields like speech recognition [59], natural language processing (NLP) [60], image processing [4], computer vision [5], and object detection [6]. In recent years, DL was highly successful in remote sensing [12][13] [15][3][14], especially in fields like image preprocessing and classification. Work in this thesis focuses on LULC, which falls under the classification branch of DL in remote sensing.

3.2 Deep Convolutional Neural Networks

CNNs are a type of DNN designed to solve tasks based on structured arrays, like images or audio. A deep convolutional neural network (DCNN) is a CNN with a large number of convolutional layers. DCNNs mimic the structure of the human visual cortex, where it sequentially uses convolutional layers to recognize more sophisticated shapes and features of the objects [61]. CNNs have proven to be extremely effective and are now a state-of-the-art method for computer vision tasks, such as image classification and object recognition [7]. Also, CNNs has found success in text classification, which is part of the NLP field.

The first CNN that beat other conventional methods in the performance of recognition of handwritten digits was *LeNet* developed by LeCun in 1998 [62]. It was able to classify distorted and rotated images, which were big issues for algorithms at the time. LeNet is a shallow CNN; it has five convolution and pooling layers, two fully-connected layers at the end, and the input to the network is a 32x32 image. However, the model considered each pixel as separate input and completely ignored neighbourhood features. The bottleneck for the improvement of CNNs at that time was the absence of GPU for use during training. Thus, training was restricted to CPUs making it nearly impossible to train deeper models with a large number of parameters in a realistic amount of time.

In 2012, the *AlexNet* [63] model was presented and won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC). It was considered a revolutionary model that first showed NNs could outperform other algorithms in the ILSVRC image recognition and classification tasks. AlexNet was established from the LeNet network by increasing the total amount of layers from five to eight and adding large filters (11x11 and 5x5), depicted in Fig. 3.1. This new architecture was only able to be trained due to the advent of general purpose computing using GPU and servers containing multiple GPU cards. Moreover, AlexNet was one of the first models to fully implement dropout to bypass the *overfitting* problem [56] and add ReLU activation functions to ease the *vanishing gradient problem* [34]. AlexNet had a significant impact on the field and future of CNN models by introducing a new architecture standard with an efficient training approach.

The success of AlexNet accelerated research in the architectural design of CNNs. In 2014, Simonyan presented the *VGGNet* architecture [25], which consisted of 16 or 19 sequential layers making it a lot deeper than AlexNet. To achieve the higher number of layers, the architecture decreased the size of the filters from 11x11 and 5x5 to 3x3 to reduce the computational complexity. Layers are built using stacked convolutions with zero-padding, with the output passing through a max-pool operation with

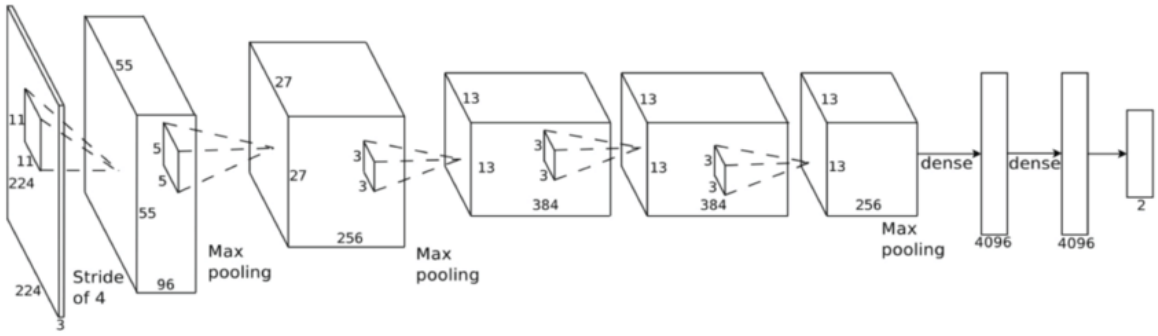


Figure 3.1: An illustration of the AlexNet architecture [63].

stride (2x2). Despite decreasing the number of parameters via smaller filters, the architecture still consists of around a hundred million parameters, which makes it difficult or impossible to train on lower-end devices. However, VGGNet showed extremely positive results in ILSVRC 2014, by placing second in the classification & localization challenge[1].

The winner of the 2014 classification & localization challenge was GoogleNet, also known as the *Inception model* [26]. The idea behind this architecture is to reduce computational complexity and still achieve high accuracy. It accomplished that by replacing convolutional layers with the *inception blocks*, which followed the split, transform and merge design, shown in Fig. 3.2. Each block consisted of 3 different sizes of filters (5x5, 3x3, and 1x1) to capture a wider range of features, especially those that varied in spatial sizes. Moreover, inception blocks have bottleneck layers (1x1 convolution), which regulate computations before each convolution with a large filter. The architecture consists of 9 inception blocks and infrequent max-pooling layers with a (2x2) stride, reaching about 4 million overall parameters. Later, improved versions of this architecture were introduced [57][64], which focused on replacing the (5x5) convolutions with two (3x3) (see *e.g.* Fig. 3.3a), the addition of asymmetric (1xn) and (nx1) convolutions in the inception block (see *e.g.* Fig. 3.3b), the inclusion of batch normalization layers, and the use of RMSProp optimizer. Recently, Francois Chollet introduced the extension of the Inception architecture called Xception [17]. It replaces inception blocks with plain *depthwise separable convolutions*. Inception architectures have performed extremely well in image classification tasks. However, due to a large number of layers and a low number of parameters, this architecture tends to overfit or underfit in some domains.

The ResNet architecture, proposed by He [27], is similar to the VGGNet, but it is around eight times deeper. Also, this design utilizes *residual blocks*, which consist of a ReLU with batch normalization

[1] http://www.image-net.org/challenges/LSVRC/2014/results

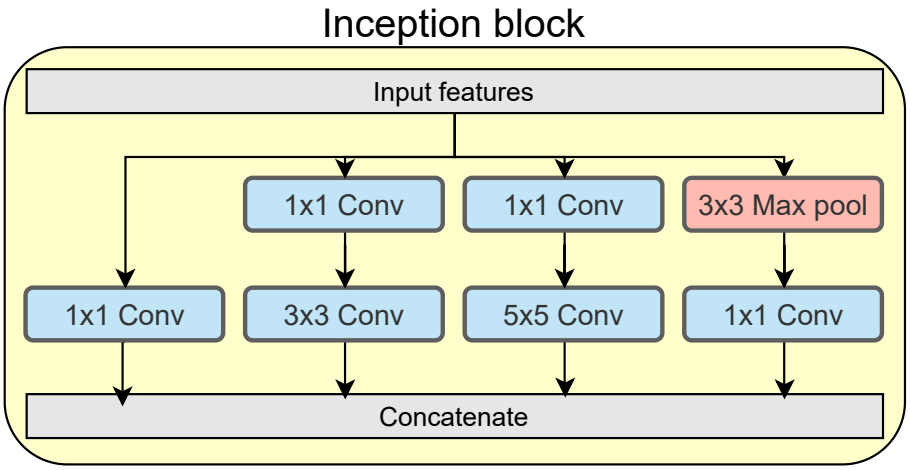
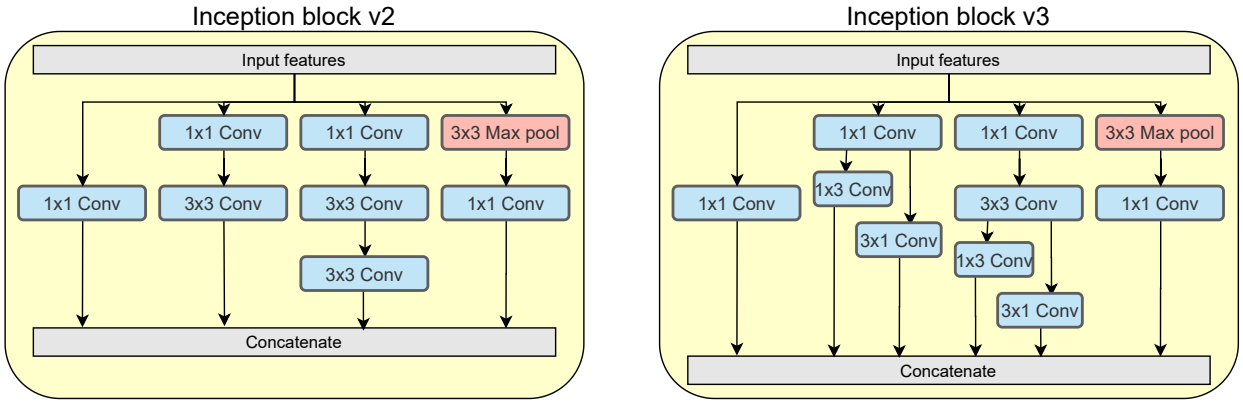


Figure 3.2: The first iteration of the inception block, where blue (1x1) convolutions are bottleneck layers, (3x3) are regular convolutions, and red is a downsampling layer.



(a) **(b)**

Figure 3.3: Improved versions of the inception blocks. (a) is an inception block that replaced (5x5) convolution with two (3x3), and (b) is an inception block that takes advantage of asymmetric (1xn) and (nx1) convolutions.

in between convolution layers. This architecture also uses shortcuts, which are simple elementwise addition of input x and output of the residual block $H(x)$, which means that ResNet trains on residual function $F(x) = H(x) + x$ instead of the regular mapping $H(x) = x$. Naturally, one would think it is effective due to the enormous amount of layers, but its success is mostly due to the shortcuts and residual block design, which helps with the vanishing gradient problem exhibited by CNNs with a large number of layers. Moreover, one study showed that it is possible to improve the performance of ResNet by changing the order of convolution, batch normalization, and ReLU inside the residual blocks [16]. ResNet won the 2015-ILSVRC and currently is one of the best architectures used for image classification tasks.

3.3 Deep Convolutional Encoder-Decoders

There are many computer vision tasks that cannot be solved by CNNs in the original form in which they were introduced (such as object detection or semantic segmentation); however, with architectural structural changes, these networks can be used in these related domains. The goal of object detection is to detect occurrences of objects of a certain class within the input image by drawing borders around the identified objects. While semantic segmentation focuses on the pixel-level classification of the entire scene, where perceptual objects (*i.e.* a specific class) within the image are all assigned the same label. The deep convolutional encoder-decoder architecture was introduced to solve these tasks [8][9] (see *e.g.* Fig. 3.4). The concept of the encoder-decoder is to extract features from the input using an encoder and then decode those features back to the original size using a decoder before classification, where the encoder is a CNN without fully connected and classification layers and the decoder is a mirror of the encoder or a CNN with a different architecture.

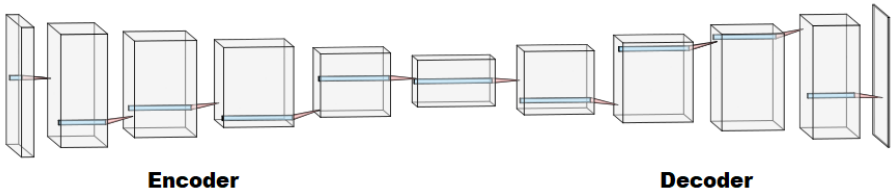


Figure 3.4: Overall structure of a regular encoder-decoder architecture.

One of the first works that used an encoder-decoder network was a FCN presented in 2014 [8]. It uses VGGNet with fully convolutional layers instead of fully connected layers as an encoder, while the decoder is a single interpolation layer that resizes prediction to the input size. Also, this architecture takes advantage of skip connections to recover lost information in downsampling layers, where each skip connection is a *shortcut* from the ResNet architecture [27]. Despite giving promising results, this architecture has difficulties in accurately predicting edges of the objects. In 2016, [29] presented a solution to overcome this issue by combining a conditional random field (CRF) with the final layer of the DCNN. Later on, the CRF was replaced with a domain transform (DT) layer, due to high computational complexity [65].

In 2015, the SegNet architecture was introduced [9] that improved the performance of the FCN by modifying the decoder via mirroring the encoder. In this architecture, the decoder and encoder consist of an equal number of convolutional layers, where the one difference between the encoder and decoder is that the latter uses bilinear upsampling layers instead of downsampling ones. Then, Ronneberger

presented the U-Net architecture [19] that is based on SegNet where the difference is due to the presence of skip connections between encoder and decoder, which improved the accuracy of predictions and helped with the vanishing gradient problem. In 2016, Milletari modified the U-Net model to process volumetric inputs [66] by including residual blocks in between convolutional blocks and replaced 2D convolutions with their 3D counterparts [67][68]. Despite groundbreaking results of the U-Net type architecture, it has an immense number of parameters – more than twice of the regular DCNN designed for simple classification – that slows down the training and makes models unusable on lower-end devices. Some architectures lowered the number of convolutional layers in the decoder to one per upsampling to decrease the number of parameters, which was also done in FCN [8]. Moreover, [18] presented a *feedbackward decoder* for the U-Net model. This model used transposed encoder weights in the decoder thereby lowering the number of parameters by a factor of 2, but this change came with a cost of lower accuracy of predictions.

While most of the papers focused on improving the decoder, Deeplab, introduced in 2014 [69], focused more on modifying the encoder. Deeplab is mainly an FCN architecture with dilated convolutional layers and fully connected CRF placed at the last layers of the network. In 2016 DeeplabV2, was presented [29] that drastically improved the performance of the model by including the first generation of ASPP. ASPP is a collection of four parallel dilated convolutions with different strides (6x6, 12x12, 18x18 and 24x24), and it was designed to try and capture objects and context of the features at multiple scales. DeeplabV3 [28] completely rethought the architecture by replacing the VGGNet-based encoder with ResNet at a fixed OS of 4 and removing the fully connected CRF layer. The OS is also a ratio of the input image size to the output feature map of the encoder. For an OS of 4 and an input image size of 224 x 224 x 3, the output feature size of the encoder is (56x56). Also, the new architecture uses the second generation of ASPP, which places a new layer in between the encoder and decoder consisting of one (1x1) and three (3x3) dilated convolutions with rates (6x6, 12x12, 18x18). DeeplabV3+, presented by Chen [20], extends DeeplabV3 by replacing the ResNet encoder with an Xception CNN. This change drastically reduced the number of parameters in the model and increased the performance. Additionally, DeeplabV3+ has a simple yet effective decoder, which improved the sharpness of predictions on the boundaries. DeeplabV3 lost a lot of information because feature maps were upsampled to the original size using bilinear interpolation. Currently, DeeplabV3+ is a state-of-the-art segmentation network.

3.3.1 Extensions

Throughout the years a lot of different encoder-decoder architectures were developed on the pair with network extensions. Network extensions are plug-in modules, which aim to improve the performance of the deep convolutional encoder-decoders without significant changes to the already existing architecture. *ASPP* and *fixed OS* from DeeplabV3 [28] are considered as network extensions because they can be easily applied to any encoder-decoder based architecture.

In 2015, the context aggregation module [23] was presented to increase the sharpness of predictions by aggregating multi-scale contextual information. This module is plugged into the last layer of the decoder and consists of 8 sequential dilation convolution layers with rates ranging from (1x1) to (16x16), where the size of the feature maps are not changed throughout all layers. The context aggregation module has proven to improve the performance of already existing semantic segmentation networks like FCN [8] with the cost of high computational complexity and a larger number of parameters.

Goodfellow [24] introduced a GAN, which consists of a generator and a discriminator, where the goal of the generator is to produce an image as close to the real one as possible, and the discriminator attempts to distinguish between original and generated images. The work presented in [70] used GAN to boost the performance of NNs, where the encoder-decoder is a generator, and custom binary CNN is a discriminator. GAN have shown great results in semantic segmentation tasks but comes with the cost of training and prediction stability. *Progressive GAN*, introduced in 2017 [21], resolved issues with training stability. The idea behind progressive architecture is to gradually add layers to both generator and discriminator throughout the training. This extension was also used in the paper [71], where the gradual increase of the network layers started in the decoder.

3.4 Unsupervised Domain Adaption

In recent years demand for DNNs in a wide range of fields has increased drastically. However, in some areas, there is either low or no presence of labelled datasets, which restrains the successful use of NNs there. UDA [22] is a semi-supervised learning framework that focuses on resolving this problem by transferring knowledge from existing labelled datasets (source domain) to similar unlabelled ones (target domain).

UDA type architectures started gaining popularity in the last couple of years, and found success in image classification tasks by implementing architectures that consisted of 3 networks [30]: a generator

network, which tries to mimic features from the source domain and apply them on to images from the target domain; a discriminator that is used alongside generator by giving feedback on how well new data mimics features of the target domain; and a regular CNN which trains on enhanced source domain images and their corresponding labels. This architecture provided acceptable results but has difficulties with the training due to the presence of 3 neural networks. One paper showed that it is possible to reach the equivalent accuracy with only one network by modifying the architecture to have two outputs [72]. Both source and target domains are passed through the same CNN, where the only difference is in their prediction. The first output is the categorical prediction of the source domain, and the second output is the prediction of how the target domain images were distorted (flipped, rotated, zoomed in, *etc.*). This architecture proved to be very effective, sometimes outperforming state-of-the-art GAN-based networks.

Currently, the use of UDA on semantic segmentation is a hot topic, due to the absence and difficulties with the generation of labelled datasets. However, there are not a lot of papers published in this sphere yet. Architecture that was introduced in a [73] gained success by following the approach mentioned in [30], but replacing CNN with an encoder-decoder. Later [31] proposed to modify the architecture by embedding a generator with the encoder (conditional generator). At this point, the generator enhances only encoded features instead of the entire image. This approach improved performance and reduced the computational complexity of the architecture. Some work tried to ignore adversarial training [72] by producing semantic masks on the source domain images and introducing reconstruction on the target domain images [74]. This model significantly reduced the computational complexity of the UDA type architectures with no loss in the accuracy of predictions.

3.5 DNNs in Remote Sensing

The success of DL in image processing encouraged researchers to use it in remote sensing. The development of DCNNs was mainly focused on scene classification and LULC map production. One of the first studies for classifying used high-resolution satellite images (UC Merced dataset) [10][11] due to good structural information. At that time, conventional DL algorithms performed poorly on medium-resolution satellite images (10m-30m per pixel) because of a deficiency of such structural information. In 2017, Sharma presented a patch-based CNN [75], which performed effectively on medium-resolution satellite images.

A lot of researchers applied different semantic segmentation methods and techniques for LULC map production that were successful since these problems are very similar to each other. Most of the

modern architectures follow the encoder-decoder structure with some minor changes. One of the first to successfully use semantic segmentation architectures for remote sensing applications were [12][13] [14] [15]. Their solution is based on an FCN structure with different encoders (*i.e.* VGGNet, GoogleNet, ResNet) used on Landsat 5/7 satellite images. Also, they implemented GAN and context module extensions to boost the performance of predictions. Additionally, work presented in 2018 [76] showed that architectures that were successful in semantic segmentation on the real-world images need to be modified to get their full potential in remote sensing. A number of different modifications were made to improve the performance of the architectures on satellite imagery. For example, to keep most of the structural information, a no-downsampling encoder network was developed [77] that used atrous convolution. Some works centred on modifying the decoder by adding symmetric unconvoluted layers [78][79]. These architectures significantly improved the accuracy of generated LULC maps.

Satellite data consists of a large number of colour channels (bands) based on each sensor type. Some sensors can have more than ten bands with different spatial resolutions ranging from 2m to 30m. Using conventional encoder-decoder architecture would be impossible in this case, that's why most of the designed architecture takes advantage of the RGB and infrared bands, which most of the time, share the same spatial resolution. However, one work presented an architecture that used both RGB and near-infrared (NIR) bands with different spectral resolutions by modifying the U-Net model to have two encoders, one for RGB data and another one for corresponding NIR data [80]. Also, some works included reconstruction of the input to boost the performance of prediction. This architecture has a decoder with two parts, the first part creates segmented images and the second one recreates encoded images [81]. These modified networks slightly increased the accuracy and sharpness of predictions.

Work in this thesis is a continuation of V. Alhassan's work [15]. This thesis discusses the usage of CNNs for creating LULC maps using encoder-decoder architectures, where ResNet, VGG-16 and GoogleNet were used as encoders with a combination of FCN based decoder including transposed convolution and skip connections. Also, [15] used two network extensions for improving accuracy: the first one is GAN, which increased overall accuracy by a slight margin; and the second one is the context module which was discussed earlier. The results show that NN can produce good results for the problem of automatic LULC map production.

3.6 Chapter Summary

This chapter discussed related works in the DL field, especially ones that are computer vision-related. The concept of encoder-decoder was introduced with examples of some successful architectures, like FCN [8], and U-Net [19]. Additionally, this chapter addresses some state-of-the-art model and architectural extensions for encoder-decoder networks, like GAN [24], context module [23], and Deeplabv3+ [20]. Then, the UDA learning framework was presented and explained with some prosperous works [31]. Lastly, DL in remote sensing was discussed with some successful examples in the image classification and LULC map generation tasks.

Chapter 4

Neural Network Models

This chapter discusses encoder-decoder architectures, and extensions that were implemented in this thesis. Additionally, CNN models are addressed in-depth, for instance, VGGNet[25], GoogleNet[26], ResNet[27][16], Xception[17] with a combination of different decoders. Moreover, more detailed information is provided regarding model extensions and model modifications that boost the performance of these models. Finally, the UDA architecture with the use of conditional generative adversarial network [31] is explained.

4.1 Encoder-Decoder

This thesis is focused on training NNs on satellite imagery, which, by its structure, differs from regular real-world images of objects. Most CNNs and encoder-decoder networks were trained on images with three colour channels: red, green and blue (RGB), while the results in this thesis were produced by models trained with satellite images (Landsat 5/7, Landsat 8, Sentinel-2) consisting of a combination of six colour channels: RGB, one near-infrared (NIR) and two shortwave-infrared (SWIR). Therefore, the input size of all the model input layers H x W x 3 was changed to H x W x 6, which slightly increases the number of parameters in the first convolutional layer, but not in the following layers. In this thesis, H and W are equal to 224 making the input size 224 x 224 x 6 and output of the network 224 x 224 x 1.

4.2 Encoders

In our experiments, the size of the input to the encoder is always 224 x 224 x 6, while the size of the extracted features is H x W x D, where H is the height, W is the width and D is the depth of the last

convolutional layer in the encoder. A number of different CNNs were used in the role of the encoder in this research: VGGNet[25], GoogleNet[26], ResNet[27][16], Xception[17].

4.2.1 VGGNet

VGGNet, presented in [25], is one of the most successful and highly used models, which took second place in the ILSVRC 2014 classification & localization challenge. It gained its popularity mostly not by its performance, but due to its simple and easy to implement structure. VGGNet consists of 16 or 19 sequential layers, where each convolutional layer uses a (3x3) filter size, followed by a dropout or batch normalization layer, ReLU activation function, and periodic max pooling or convolutional layer with a stride of (2x2). Implemented VGGNet architecture is depicted in Fig. 4.1.

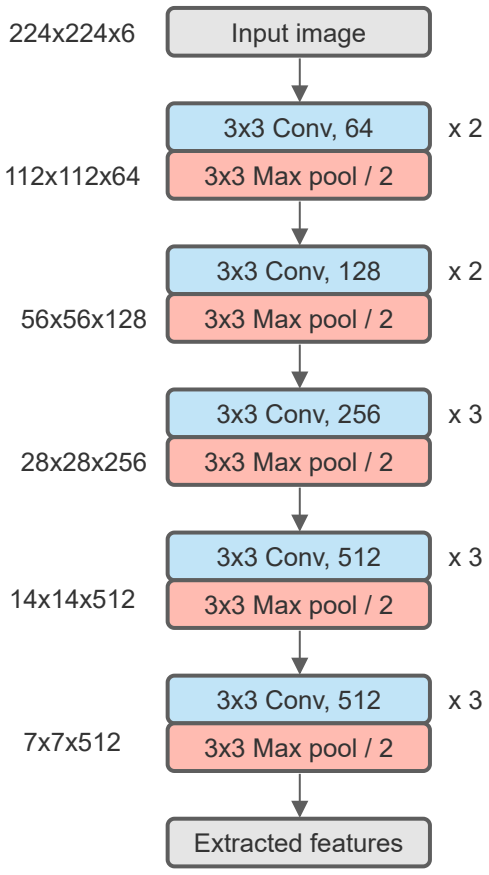


Figure 4.1: Network architecture diagram for an implemented VGGNet type encoder, where each convolutional layer is followed with batch normalization and ReLU activation function.

4.2.2 GoogleNet

GoogleNet was the winner of the 2014-ILSVRC classification & localization challenge. The presented architecture uses a novel structure that tries to improve accuracy and reduce the computational complexity of the model by introducing an inception block design [26]. Shortly after that, improved versions of this architecture were introduced [57] [64], which focused on modification the inception block (see *e.g.* Fig. 3.3). In this thesis, the first iteration of the inception model was used to simplify implementation. Inception block usually consists of 3 parallel convolutions with different filter sizes (5x5, 3x3, 1x1) and a max-pooling layer with striding (2x2). Also, layer inputs are passed through the bottleneck layer, with (1x1) convolutions, to reduce the depth of the input vector to make it more computationally efficient (as shown in Fig. 3.2). A more detailed architecture diagram is depicted in Fig. 4.2.

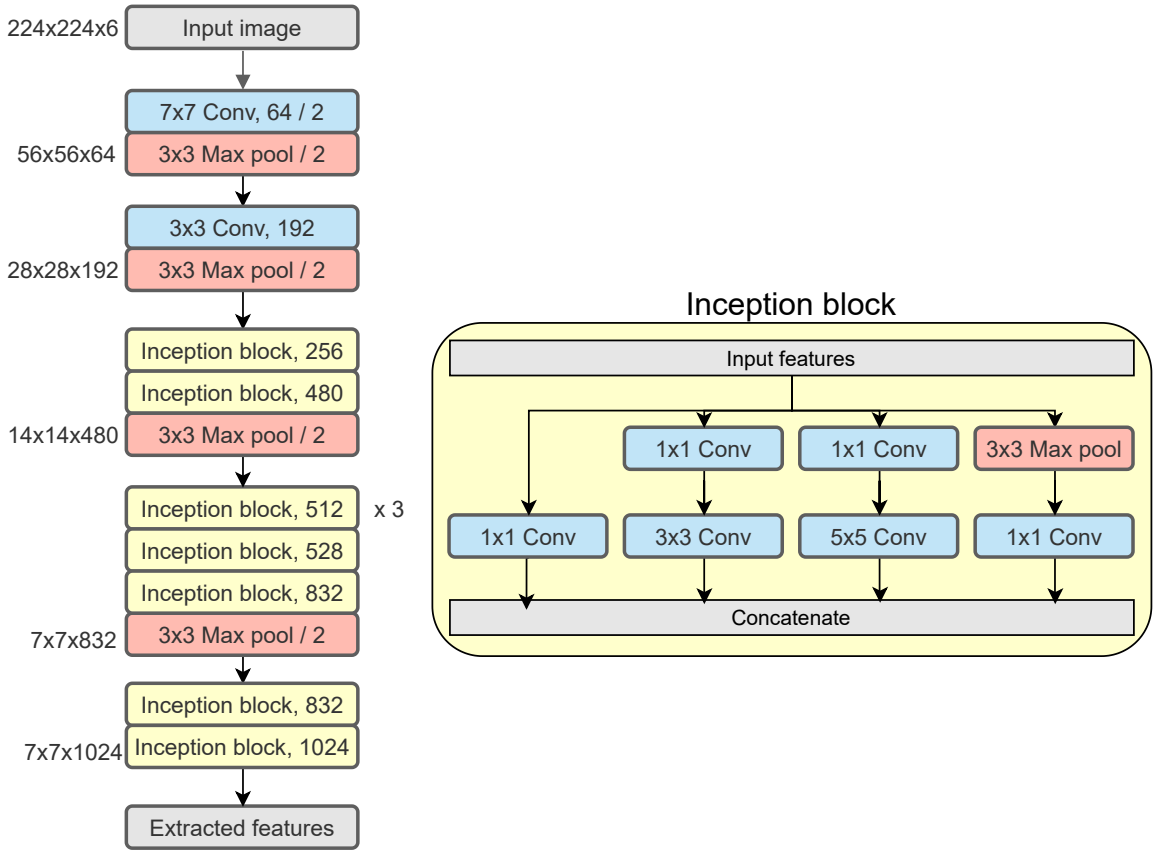


Figure 4.2: Network architecture diagram for an implemented GoogleNet type encoder, where each convolutional layer and inception block is followed with batch normalization and ReLU activation function.

4.2.3 Xception

This work also uses the Xception model [17], which is a modified version of GoogleNet. Currently, this model is one of the best for image classification tasks, and showed excellent results for semantic segmentation when used in the Deeplabv3+ architecture [20]. The Xception model followed the same design goals of GoogleNet, *i.e.* reaching high accuracy with low computational complexity with simple implementation. The Xception structure is similar to GoogleNet but replaces inception blocks with (3x3) depthwise separable convolutions with shortcuts [17]. A more detailed architecture diagram is shown in Fig. 4.3.

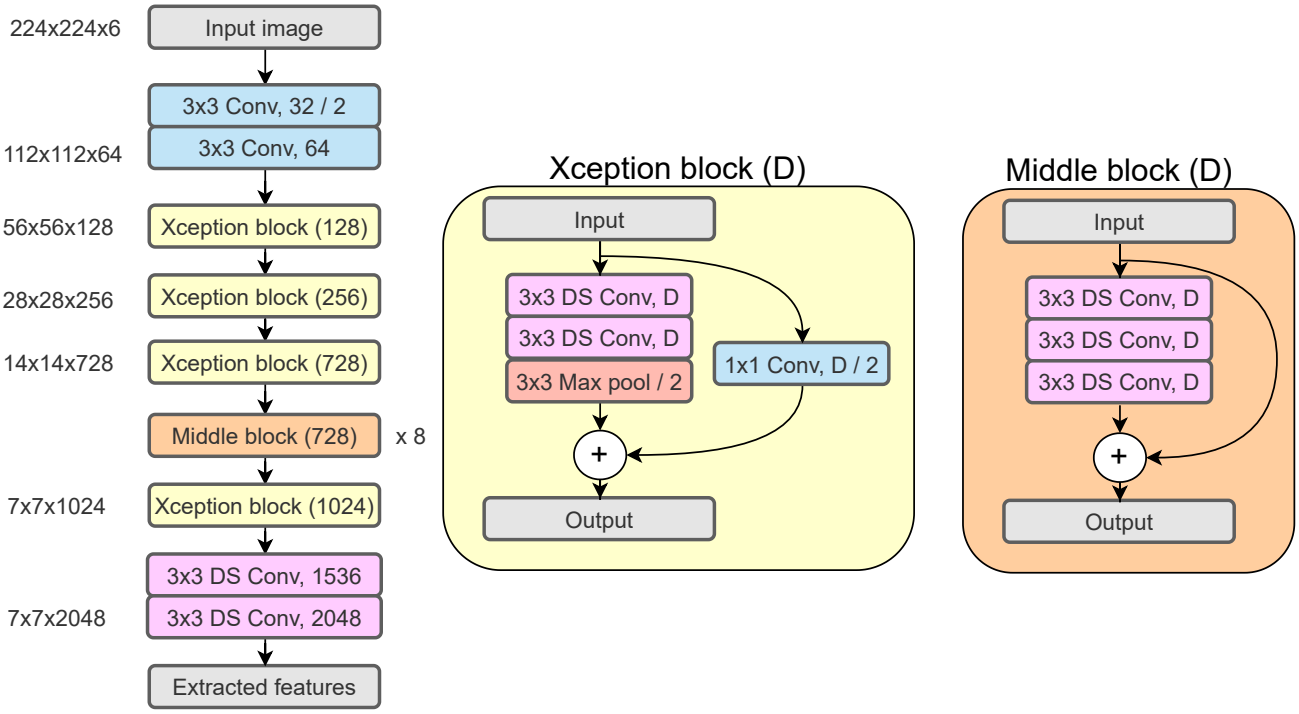
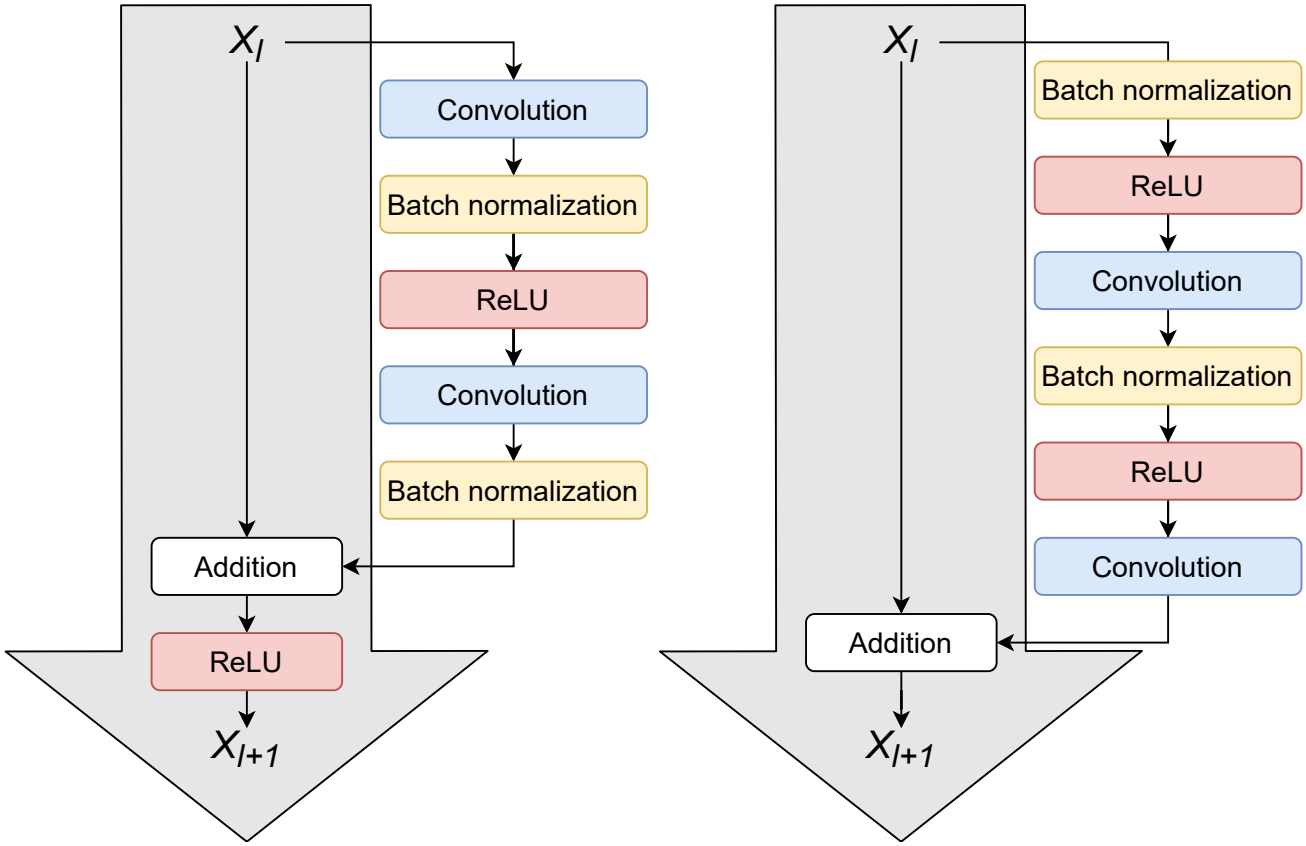


Figure 4.3: Network architecture diagram for an implemented Xception type encoder, where each depthwise separable (DS) convolution is followed with batch normalization and ReLU activation function.

4.2.4 ResNet

ResNet was the winner of 2015-ILSVRC and currently is one of the best architectures used for image classification tasks, due to its residual block design, which allows generating very deep models [27]. ResNet consists of a number of sequential residual blocks, where each block has one (3x3) convolutional layer in between two (1x1) convolutional layers, which are responsible for depth reduction and restoration. Moreover, there is a shortcut before each residual block, which is a simple elementwise addition of

input and output of the residual block, which helps with the vanishing gradient problem [34] and enables deeper models. Also, we used an additional version of this model, which changes the order of convolution, batch normalization and ReLU inside the residual blocks called fully pre-activation ResNet introduced in [16]. The difference between regular and pre-activation residual blocks is depicted in Fig. 4.4. Also, a more detailed architecture diagram of the ResNet model can be seen in Fig. 4.5.



(a) Regular residual block design.

(b) Pre-activation residual block design.

Figure 4.4: Regular and improved residual blocks designs: (a) applies convolution first and then batch normalization with ReLU, and (b) applies batch normalization with ReLU first and then convolution.

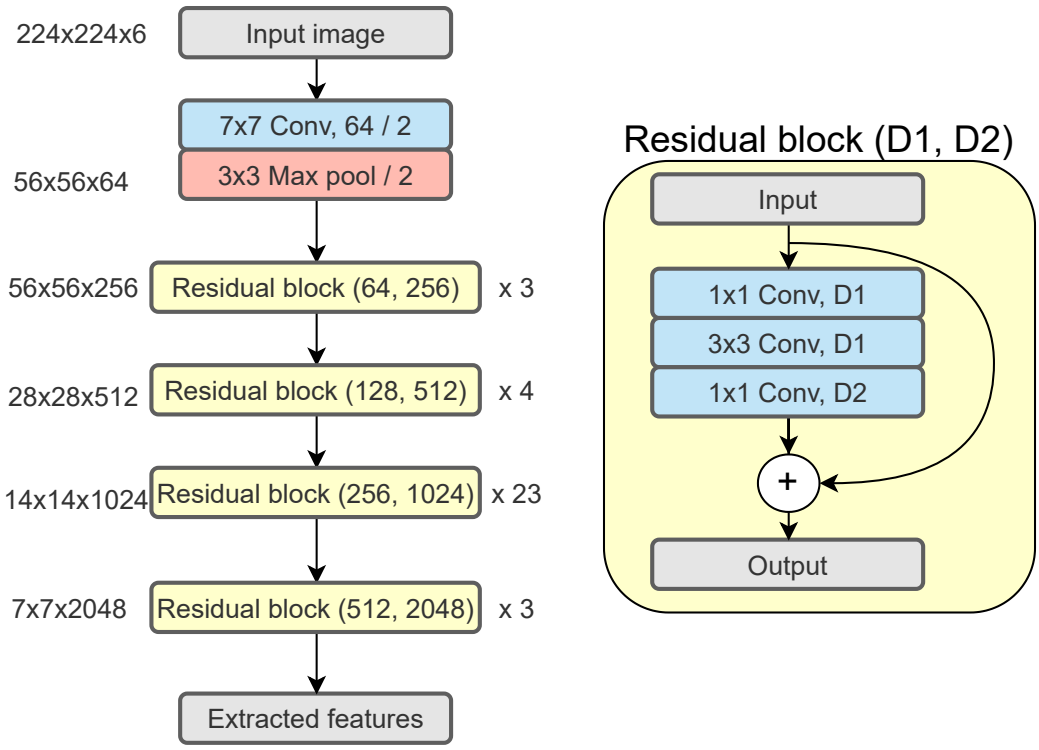


Figure 4.5: Network architecture diagram for an implemented ResNet type encoder.

4.3 Decoders

The decoders are networks that try to produce output with the same dimensions as the original image. A new labelled mask is produced from the low-level features retrieved by the encoder network or some other algorithm. In our experiments, we used the following decoders: FCN-type [8], U-Net [9][19], Feedbackward [18]. Input to the decoders are feature maps generated by an encoder of the size H x W x D, where H is the height, W is the width, and D is the depth of the last convolutional layer in the encoder. The output from the decoder is a generated labelled mask of size 224 x 224 x 1, which has the same height and width as input to the encoder.

4.3.1 FCN

The FCN architecture described in [8] is one of the first successful decoders used for semantic segmentation models. In this work, we implemented an FCN-8 decoder variant, which gradually upsamples extracted features using transposed convolution and merges them with features of the same size from the encoder

layers using shortcuts from ResNet [27], also referred to as *skip connections*. See Fig. 4.6 to get a better understanding of the structure of the FCN decoder.

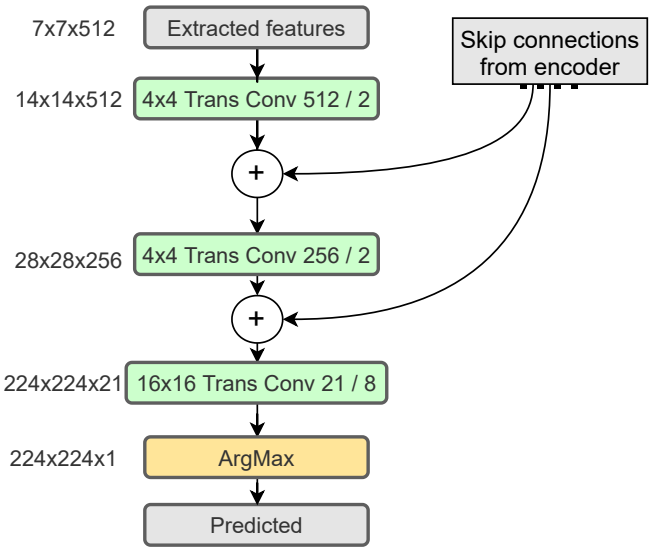


Figure 4.6: Network architecture diagram for an implemented FCN-8 type decoder for VGGNet, which uses transposed convolution on the pair with skip connections from the encoder.

4.3.2 U-Net

The SegNet model introduced in [9] presented a new decoder design, which fully mimics the structure of the encoder, but replaces downsampling layers with upsampling one. Despite great results, it suffered from a vanishing gradient problem due to the depth of the model. Shortly after, an improved version of the same decoder was introduced [19], called U-Net, which implements skip connections to improve the quality of predictions and help with the vanishing gradient problem. In this thesis, we implemented the U-Net decoder only for the VGGNet encoder due to difficulties with implementation. This structure is shown in Fig. 4.7.

4.3.3 Modified U-Net

To make the U-Net type decoder compatible with every encoder it was slightly modified. The overall structure of the model follows the same concept that is presented in Fig. 4.7. However, the number of convolutions was reduced to one per upsampling layer, and complicated block structures (*e.g.* ResNet residual block, GoogleNet inception block) were disregarded, which also significantly decreases the computational complexity of the model.

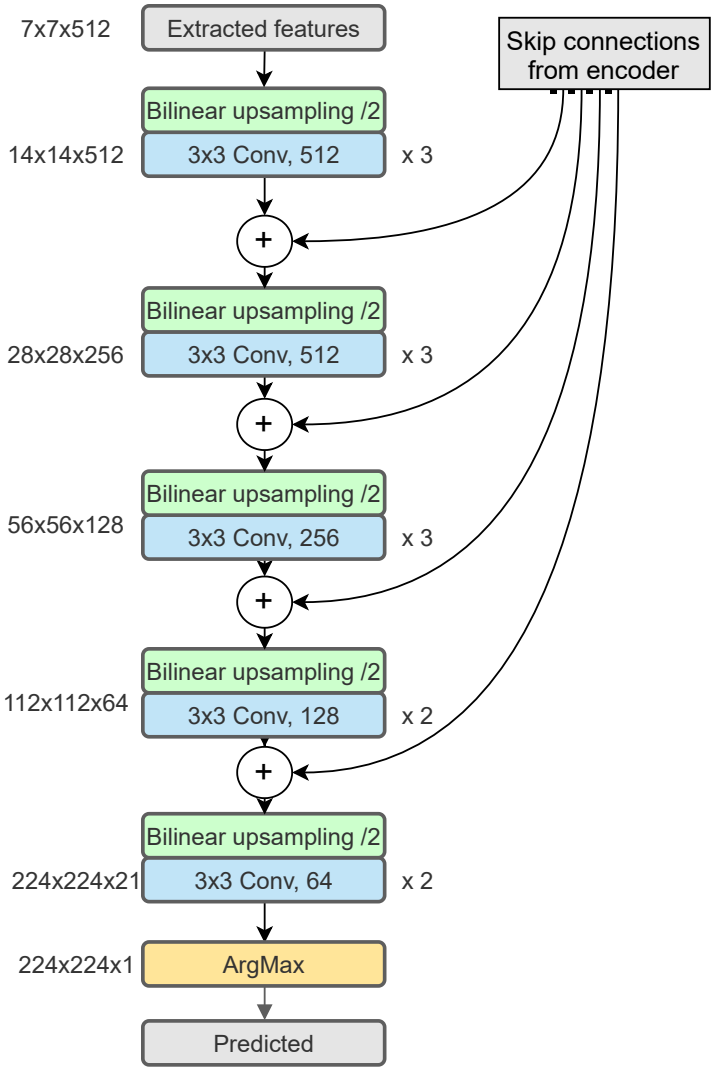


Figure 4.7: Network architecture diagram for an implemented U-Net type decoder for VGGNet, which uses reversed CNN structure on the pair with skip connections from the encoder.

4.3.4 Feedbackward

U-Net type decoders [9][19] have double the amount of parameters of the encoder, making them computationally expensive for training. Recently one work presented a way to resolve this issue by introducing the feedbackward decoder [18], which uses the weights W from the encoder in the decoder by transposing them (*i.e.* using W^T). This approach decreases the time needed for training and the overall size of the model because the same weights are used for encoding and decoding. We implemented this model only for VGGNet due to the complexity of implementation. The feedbackward decoder has an identical structure to the U-Net decoder shown in Fig. 4.7.

4.4 Model Extensions

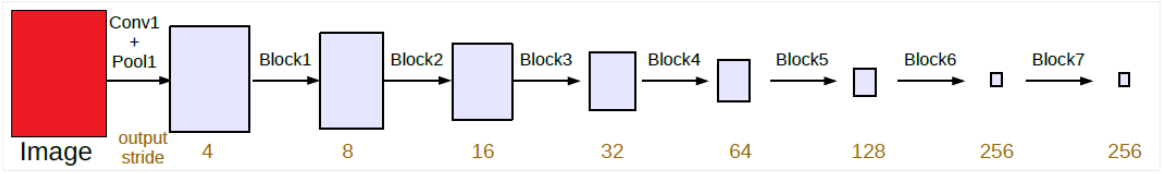
Some model extensions were implemented to improve the accuracy of prediction, where model extensions are plug-in modules that aim to improve the performance of the encoder-decoder networks without significant changes to their architectures. Model extensions usually are layer modifications or blocks that are inserted in between, before or after encoder-decoder architecture.

4.4.1 Layer-Level Modifications

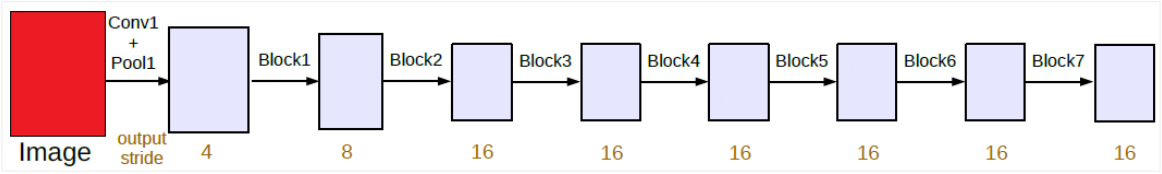
To improve the performance and stability of training, several modifications were made to the model on the layer level. For instance, [57][18] recommends replacing dropout layers and biases with batch normalization to stabilize training and omit overfitting and underfitting. Additionally, [20] showed that it is possible to get a slight boost in performance by using convolutional layers with striding instead of pooling layers and replacing upsampling layers (*e.g.* bilinear upsampling) with transposed convolution.

4.4.2 Output Stride

[28] introduced the OS model extension, which freezes the width and height of features at a certain step in a NN. Also, the OS is usually referred to as a ratio of the input size to the output feature map of the encoder. For example, if the OS is set to 8 or 16 and an input image size is 224 x 224 x 6, the output feature size of the encoder is 28 x 28 x D or 14 x 14 x D respectively, where D is the depth of the last layer of the encoder (see, *e.g.*, Fig. 4.8). In this work, we tested OS of 32, 16, 8, 4 and 2.



(a) Encoder architecture with regular OS.



(b) Encoder architecture with fixed OS at 16.

Figure 4.8: Comparison of encoders with regular OS and fixed OS at 16 [28] .

4.4.3 Atrous Spatial Pyramid Pooling

ASPP is a model extension that was introduced in [29] and then improved in the [28][20]. This extension is inserted in between encoder and decoder and is only usable with the combination of the OS model extension, mentioned in the subsection above, or with a model with a large amount of extracted features. ASPP is a block that captures objects and context of the features at multiple scales, and consists of four parallel dilated convolutions with different strides (6x6, 12x12, 18x18 and 24x24). The structure of ASPP is depicted in Fig. 4.9.

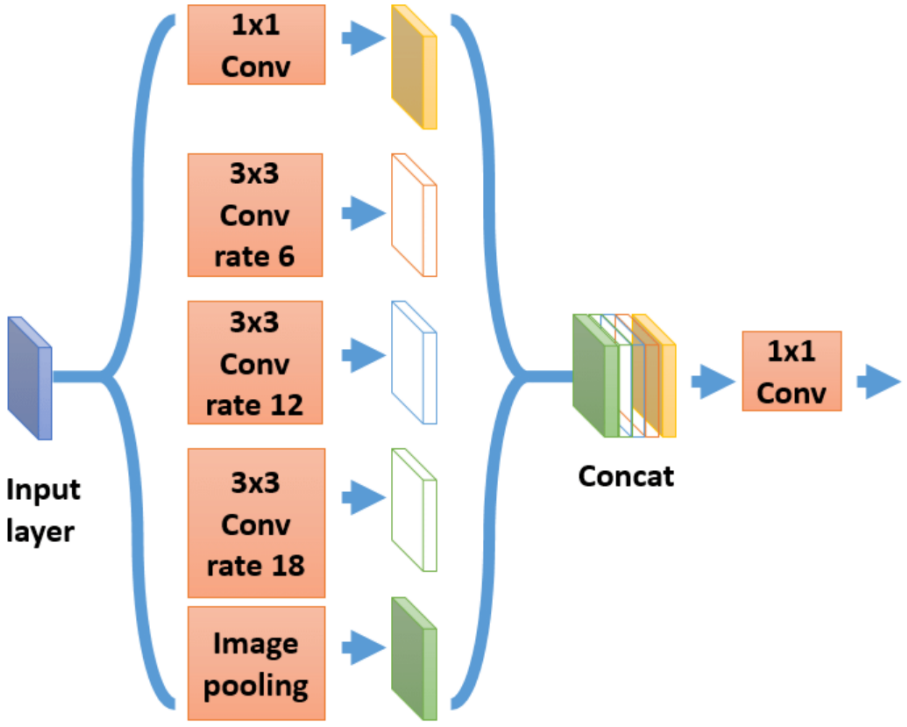


Figure 4.9: Network architecture diagram for an implemented ASPP extension [82].

4.4.4 Context Module

The context module, introduced in [23], aggregates multi-scale contextual information of the prediction to improve its sharpness and accuracy by inserting itself after the decoder and before the argmax operation. This extension consists of stacked dilated convolution at a rate ranging from (1x1) to (16x16). In previous work [15], the use of the context module proven to be extremely effective by increasing the accuracy of predictions by a slight margin in the cost of extremely large computational complexity due to the size of convolutional layers. The structure of this extension is summarised in Fig. 4.10.

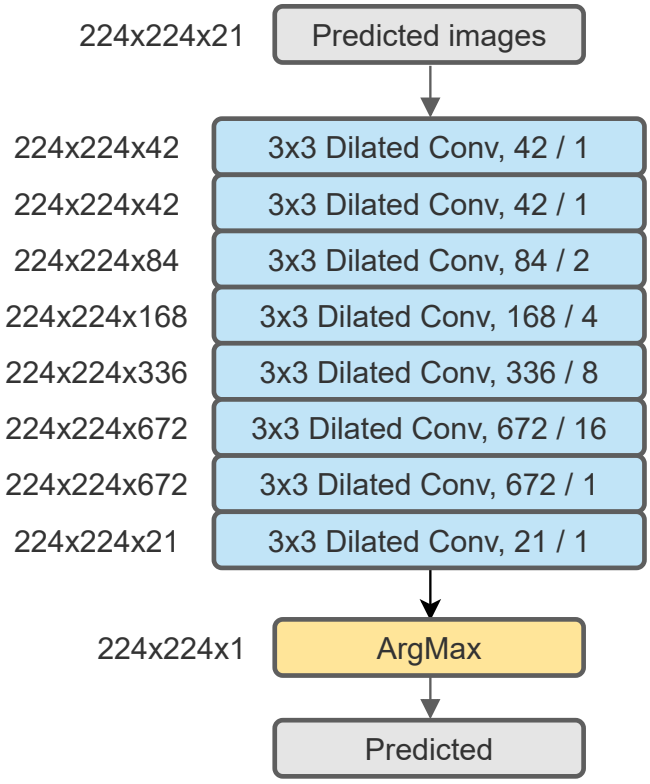


Figure 4.10: Network architecture diagram for an implemented context module extension using dilation convolutions at different rates, which plugs-in after decoder and before argmax/softmax operation.

4.5 Architectures

Over and above to model extensions, different architectural designs were explored and implemented. For example, Deeplab architectures presented in [29][28] are based on encoder-decoder architectures and designed to solve semantic segmentation tasks, but they do not have a decoder in their structure. Also, adding GAN to an encoder-decoder extension to boost the accuracy of prediction was investigated. Moreover, a UDA architecture was implemented that is designed to transfer knowledge from the source domain to the target domain.

4.5.1 Deeplabv3+ Architecture

The state-of-the-art Deeplabv3+ architecture [20] for semantic segmentation of real-world images was implemented in combination with Xception and Resnet models. This architecture uses OS and ASPP model extensions to boost the accuracy of predictions. While older versions of Deeplab [29][28] have just a single bilinear upsampling layer in the place of the decoder, the implemented version uses two

steps. The implemented decoder retains the FCN-8 structure, but transposed convolution is replaced with bilinear upsampling and elementwise addition shortcut is replaced with concatenation (see *e.g.* Fig. 4.11).

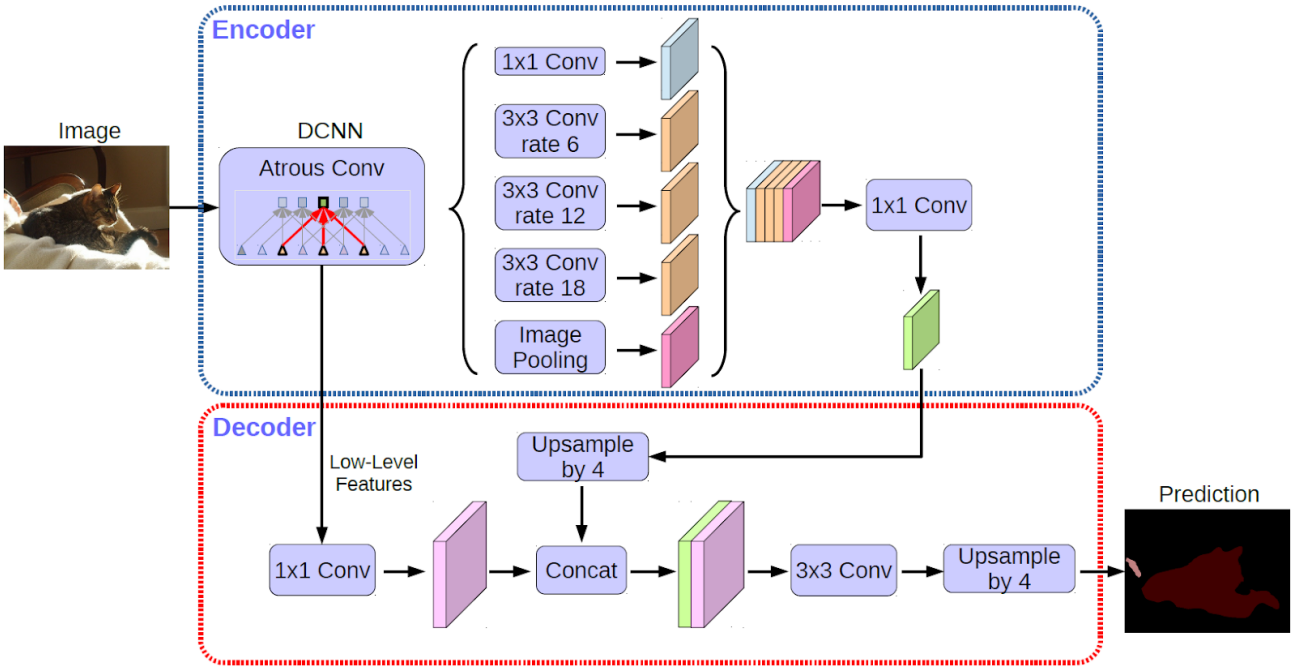


Figure 4.11: Implemented Deeplabv3+ architectural design [20].

4.5.2 Generative Adversarial Network

To further improve the accuracy of predictions, GAN architectures were considered and implemented, as they proved to slightly improve the performance of the models [24][15]. GAN architecture consists of 2 parts: a generator, G, and a discriminator, D, where the generator is an encoder-decoder architecture mentioned before, and the discriminator is a simple CNN, which uses stacked convolutional, batch normalization and leaky ReLU layers [70]. While the generator creates a labelled mask $\hat{y} = G(x)$ from the input x, the discriminator tries to identify between two inputs which one is a ground truth map (*real*) y and which one is generated map (*fake*) $\hat{y}$. In this architecture generator and discriminator are trained independently process is depicted in Fig. 4.12.

Due to changes made to the architecture and the presence of two trainable NNs, the calculation of the loss function must be modified. Firstly, the loss between predicted maps $\hat{y}$ and ground truth maps y for the generator G is calculated using a multi-class cross-entropy (mce) loss function L_{mce} defined by

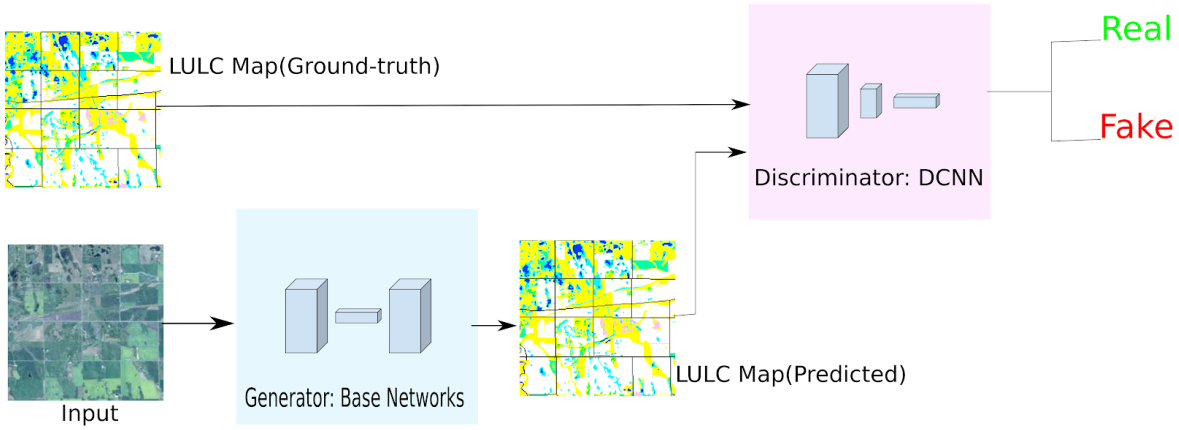


Figure 4.12: Implemented GAN architectural design [15].

Eq. 2.7. Correspondingly, the loss can be calculated for the discriminator D using binary cross-entropy (bce) loss, defined as

$$L_{bce}(z, \hat{z}) = -(z \ln(\hat{z}) + (1 - z) \ln 1 - \hat{z}, \tag{4.1}$$

$$L_D(y, \hat{y}) = \frac{1}{N} \sum_{i=0}^{N} L_{bce}(y, 1) + L_{bce}(\hat{y}, 0), \tag{4.2}$$

where z is a binary probability for prediction and ground truth, $\hat{z}$ is predicted probability in range (0, 1), and N are several labelled masks that are processed in the single step (batch size). Also, we can modify the generator loss by including the discriminator calculation, so the network not only tries to predict maps as close to ground truth as possible but also tries to make them not look like fake images. Thus, the new generator loss is defined as

$$L_G(y, \hat{y}) = \frac{1}{N} \sum_{i=0}^{N} L_{mce}(y, \hat{y}) + \lambda L_{bce}(\hat{y}, 1), \tag{4.3}$$

where λ is a regularization hyper-parameter.

4.5.3 Progressive Architecture

In this thesis, a progressive growing GAN architecture is considered [71], which works with an encoder-decoder type network. The overall training process is identical to regular GAN, but the decoder and discriminator are gradually increased in depth and size of features as training passes. For example, the generator produces predicted maps of size 7 x 7 x 21 at the beginning of the training, and convolutional

blocks are added to the decoder after a certain amount of training steps, which increases the size of the prediction to 14 x 14 x 21. This process repeats until the decoder reaches an original input size of 224 x 224 x 21. Depth of the discriminator increases similarly to the decoder, but convolutional blocks are added at the beginning of the CNN. Ground truth images for loss calculation are downsampled to the size of predictions using the nearest neighbour method [53]. A progressive growing GAN architecture is shown in Fig. 4.13.

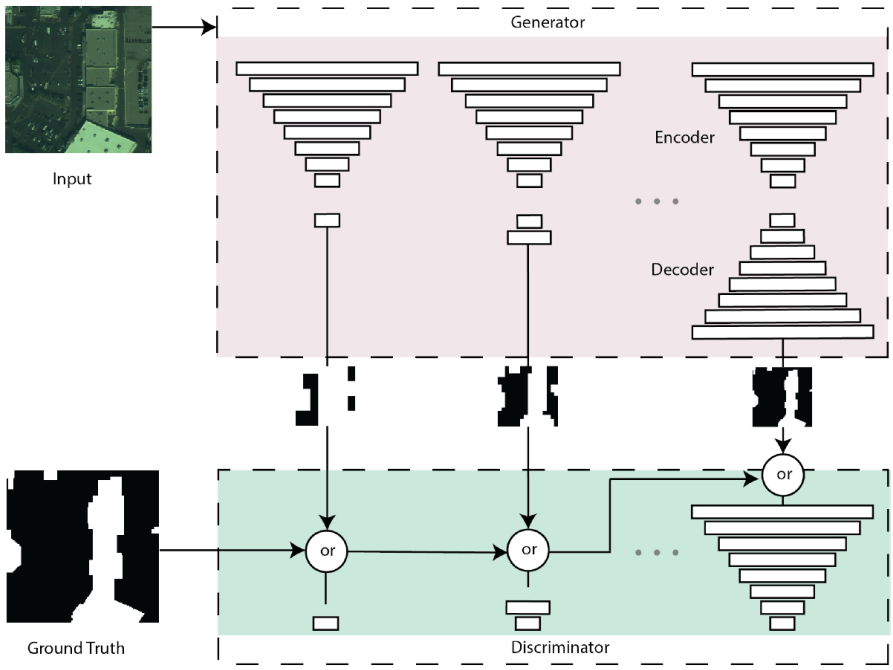


Figure 4.13: Network architecture diagram for an implemented progressive GAN [71].

4.5.4 Structured Domain Adaptation Network

In this thesis, we had to transfer knowledge from the Landsat 8 (30m x 30m) dataset to the Sentinel-2 (20m x 20m) dataset, which is the formulation of the UDA task. To resolve this problem, a structured domain adaptation network was implemented [31], which consists of a regular encoder-decoder network, conditional generator and discriminator. The regular encoder-decoder is based on a VGGNet feature extractor E with U-Net decoder and classifier T and designed to produce a labelled mask from the input images. The conditional generator, G, is formed from B stacked residual blocks, in our implementation $B = 8$, which focus on extracting features from the target domain and applying them to the source domain features with the help of a discriminator. The discriminator, D, follows the same idea that was described in Subsection 4.5.2, but, in this case, the real data are features extracted from the target

domain, and the fake data are features extracted from the source domain and enhanced by a conditional generator G. The general idea behind the UDA structure is to make features extracted from the source domain look like features extracted from the target domain and train an encoder-decoder network on the enhanced source domain data. The structured domain adaptation network is illustrated in Fig. 4.14.

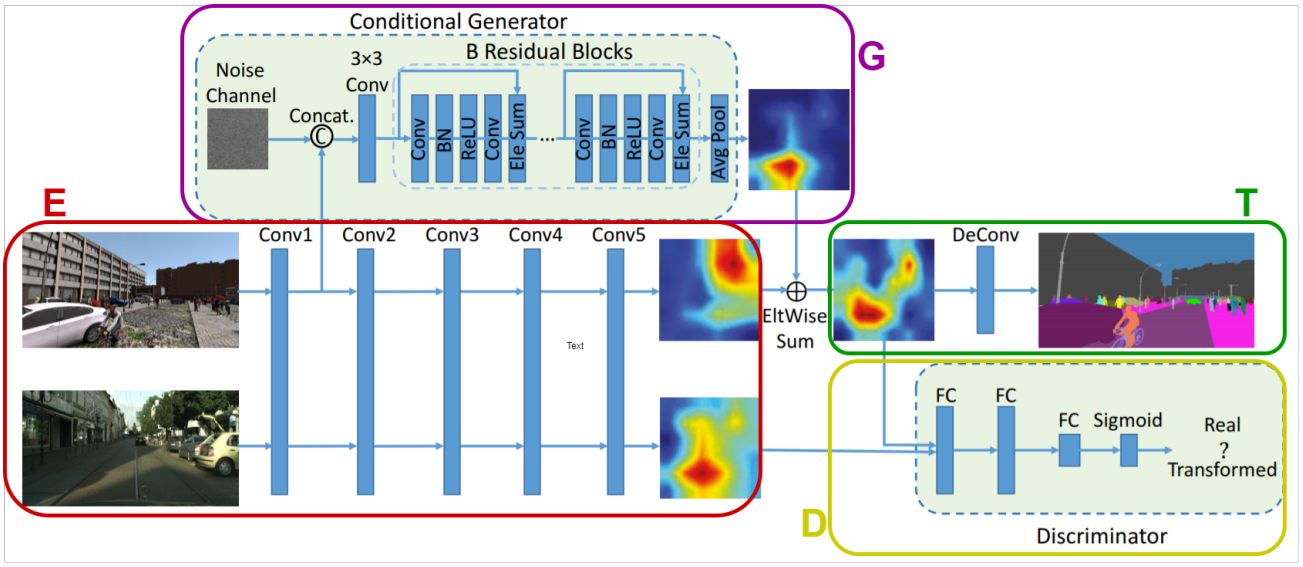


Figure 4.14: Network architecture diagram for an implemented UDA architectural design, where Conv1-Conv5 is an encoder E, DeConv T is a U-Net type decoder, the conditional generator G is the target domain feature extractor, and the discriminator D trains the conditional generator [31].

The loss function was updated for correct UDA training. Define x_s as raw data from the source domain, x_t as raw data from the target domain, and y_s as the corresponding ground truth labels for the raw input x_s. Binary cross-entropy loss L_{bce} was used for calculating the discriminator loss introduced in Eq. 4.1, but ground truth y and predicted masks $\hat{y}$ are replaced with extracted target domain features $E(x_t)$ and enhanced source domain features $E(G(x_s))$, respectively. Additionally, to make the training more stable, the generator loss was updated, which was calculated using predictions of enhanced source domain features $T(E(G(x_s)))$ through the use of predictions based on non-enhanced source domain features $T(E(x_s))$. The new loss function is defined as

$$\frac{1}{N}\sum_{i=0}^{N} \lambda_1 L_{mce}(y_s, T(E(G(x_s)))) + \lambda_2 L_{mce}(y_s, T(E(x_s)) + \lambda_3 L_{bce}(E(G(x_s)), 1), \tag{4.4}$$

where λ_{1-3} is a regularization hyper-parameters.

4.6 Chapter Summary

This chapter discussed the in-depth structure of implemented encoder-decoder architectures, where encoders (VGGNet [25], ResNet [16], *etc.*) and decoders (FCN [8], U-Net [19], *etc.*) were presented separately. Then, a complete structure for each model and architectural extension was presented, like OS, ASPP, GAN, *etc.* Lastly, the considered UDA learning framework that relies on a conditional generative adversarial network [31] was introduced and explained.

Chapter 5

Implementation Details

In this chapter, we introduce our remote sensing datasets (Landsat 5/7, Landsat 8, Sentinel-2), their type, and characteristics. Moreover, we discuss augmentation and transformation preprocessing methods that were used on the datasets before and during the training. Also, a brief introduction to the hardware and software used to generate the results is given.

5.1 Datasets

As was discussed, the overall goal of this work was to develop models to automatically generate LULC maps corresponding to the NALCMS labels, the legend of the classes is shown in Fig. 5.1. Note that not every class was present in the used datasets. Moreover, we wanted to perform this task for data produced by Landsat 5/7, Landsat 8, and Sentinel-2 data. The area we were interested in was the *Lake Winnipeg watershed*. However, this area was too large to use for comparison and contrasting different DL models due to the associated training time required for a dataset of this size. The result is that we used four datasets, which include a smaller dataset encompassing the *southern extent of Manitoba*, as well as the larger Lake Winnipeg dataset. The first provided dataset, called the *southern extent of Manitoba*, was acquired from the Landsat 5/7 (see a red outline in Fig. 5.2 and 5.3), which covers an area of approximately 148,800 km^2. Shortly after that, the *Lake Winnipeg watershed* datasets were provided for both Landsat 5/7 and Landsat 8 sensors, which covers an area of six times the size of the southern extent of Manitoba (see *e.g.* Fig. 5.4). From Fig. A.3 and Fig. A.5, the Landsat 5/7 Lake Winnipeg watershed dataset consists of 3 provinces (Alberta, Saskatchewan, and Manitoba), while Landsat 8 counterpart additionally includes part of Ontario and North Dakota but is missing the northern part of the previous dataset. The size of the Landsat 8 Lake Winnipeg watershed is slightly smaller than the

Landsat 5/7 equivalent, but not for a big margin. Also, an additionally provided dataset of the southern extent of Manitoba was acquired from Sentinel-2 sensors. Most of the experiments were performed on the Landsat 5/7 southern extent of Manitoba to lower the time needed for training. Then, based on the results, the best architecture was chosen, and the models for Landsat 5/7 and 8 were generated using the Lake Winnipeg watershed datasets.

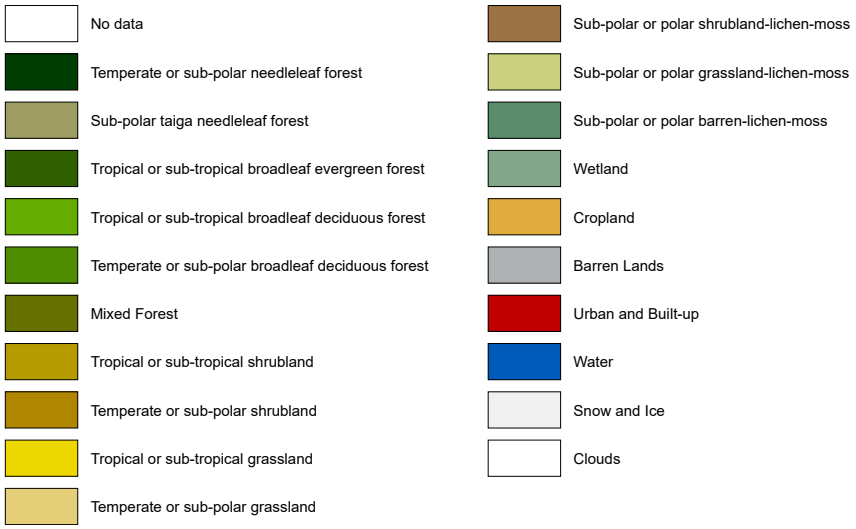


Figure 5.1: NALCMS land-use classes.

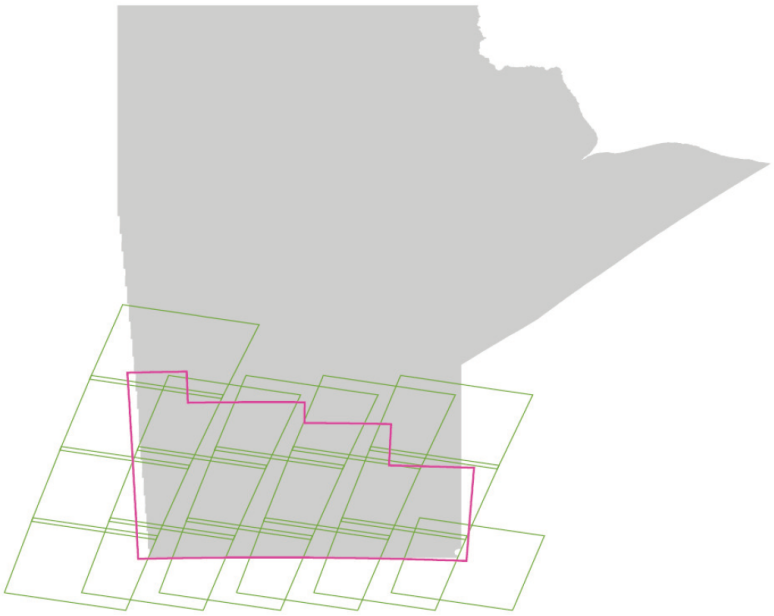


Figure 5.2: Province of Manitoba with the southern extent outlined with red border and the associated Landsat 5/7 scenes that cover this area (green)[15].

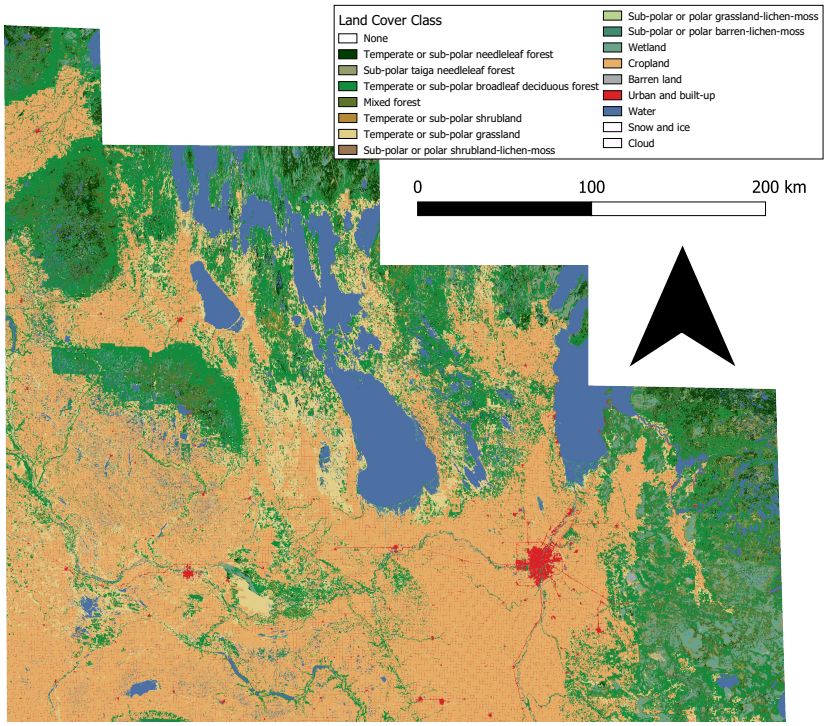


Figure 5.3: NALCMS maps of the southern extent of Mantioba.

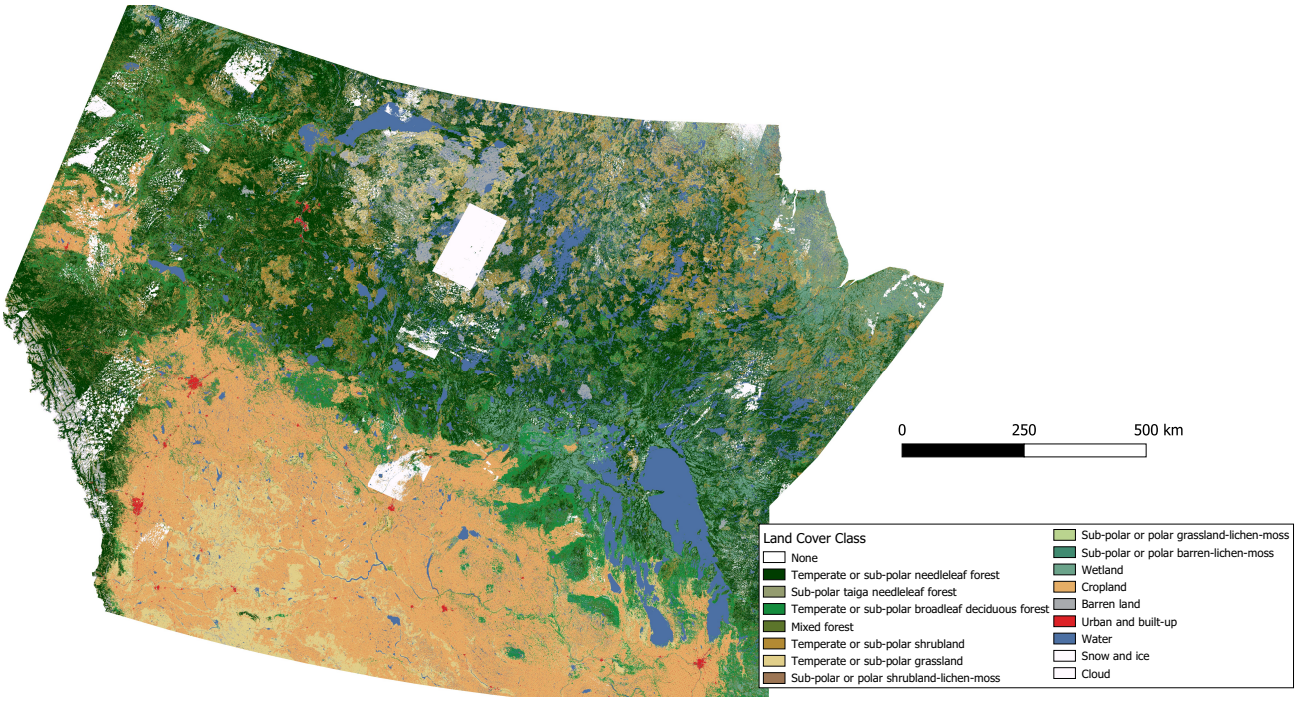


Figure 5.4: NALCMS maps of Lake Winnipeg watershed.

Each of the sensors has its characteristics usually represented in the form of four different types of resolutions [83].

- Spatial resolution – This type of resolution refers to the area of the square on the ground, which each pixel represents.
- Spectral resolution – This type of resolution indicates the ability of a sensor to measure specific wavelengths of the electromagnetic spectrum.
- Temporal resolution – This type of resolution refers to the frequency of a satellite to provide images of the same geographical area.
- Radiometric resolution – This type of resolution describes a satellite's ability to discriminate very slight differences in energy. The finer the radiometric resolution of a sensor, the more sensitive it is to detecting small differences in reflected or emitted energy.

Detailed differences between used sensors can be seen in Table 5.1. Note that we used the same six bands for all three sensors, and *cloud masks* were generated using different methods for Landsat 5/7 and Landsat 8 sensors.

Table 5.1: Landsat 5/7, Landsat 8, and Sentinel-2 sensors comparison.

	Landsat 5/7	**Landsat 8**	**Sentinel-2**
Spectral Bands	Blue, Green, Red, NIR, SWIR1, SWIR2 (see Tables 5.2 & 5.3)		
Spatial Resolution	30m x 30m		10m x 10m and 20m x 20m
Radiometric Resolution	8-bit	16-bit	12-bit
Cloud Mask	*Otsu* thresholding method	QA band	-

5.1.1 Landsat 5/7 and Landsat 8

Landsat 5/7 and Landsat 8 are satellites that collect imagery from the surface of the Earth with a spatial resolution of 30m x 30m. This type of satellite imagery has plenty of bands, each having a different wavelength. We used the datasets with six-band composition, bands and their characteristics are described in Table 5.2. The difference between the satellites is mostly due to the format used to store data. Landsat 5/7 uses 8-bit unsigned integers, while Landsat 8 uses 16-bit. Cloud masks for both

sensors were provided beforehand. Landsat 5/7 datasets labels for the clouds were manually generated using the Otsu thresholding method [84] on the blue band. Alternatively, labels for the clouds were manually generated for Landsat 8 using the quality assessment (QA) band [85].

Table 5.2: Spatial resolution and wavelength range of the Landsat 5/7 and Landsat 8 data.

Spectral region	Wavelength range (nm)	Resolution (m)
Blue (B)	450 – 515	30
Green (G)	525 – 605	30
Red (R)	630 – 690	30
Near Infrared (NIR)	750 – 900	30
Shortwave Infrared (SWIR)	1550 – 1750	30
Shortwave Infrared 2 (SWIR2)	2090 – 2350	30

5.1.2 Sentinel-2

Besides Landsat, the Sentinel-2 dataset of southern Manitoba extent was obtained, where Sentinel-2 is a satellite that acquires high spatial resolution of 10m x 10m to 60m x 60m optical images at 12-bit format and stores it using 16-bit unsigned integers. Sentinel-2 data should follow Landsat 8 structure, thus obliged to have a six-band composition of the same spatial and similar spectral resolutions. However, corresponding Sentinel-2 bands have a different spatial resolution. Therefore, the acquired Sentinel-2 dataset had some bands interpolated to generate a required six-band composition. Downsampling of higher 10m x 10m bands to 20m x 20m instead of upsampling was considered because it introduces less systematic noise during interpolation. The nearest neighbour [53] method was chosen due to its ability to keep original pixel values after interpolation, while the other methods do not. Detailed information of the bands are given in Table 5.3.

Table 5.3: Spatial resolution and wavelength range of the Sentinel-2 data.

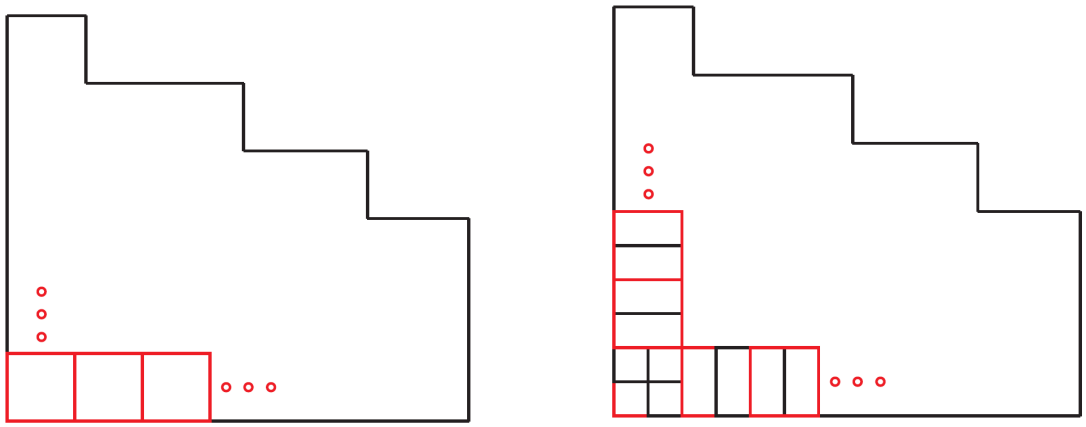
Spectral region	Wavelength range (nm)	Resolution (m)
Blue (B)	458 – 523	10
Green (G)	543 – 578	10
Red (R)	650 – 680	10
Near Infrared (NIR)	785 – 899	10
Shortwave Infrared (SWIR)	1565 – 1655	20
Shortwave Infrared 2 (SWIR2)	2100 – 2280	20

5.2 Data Preprocessing

All satellite data, and corresponding labels, were preprocessed before training due to the size of the inputs. For example, consider the southern extent of Manitoba dataset with a pixel resolution of 14975 x 13331. This image is completely mismatched with the input to the implemented encoder-decoder networks, which requires input image resolution of 224 x 224. Similarly, there would not be enough images of size 14975 x 13331 to train a DL model, and images of this size could not be processed by our computer hardware. Consequently, we used the data augmentation technique called *tiling*, introduced in [12][13][14][15], with some minor improvements.

5.2.1 Tiling

Tilling is the process of splitting a bigger image into small-sized squared ones (see *e.g.* Fig. 5.5a), also referred to as tiles, where a tile is one 224 x 224 image. For example, from the southern extent of Manitoba dataset of size 14975 x 13331 around 7,270 tiles were generated. Moreover, to increase the number of tiles generated from the same image, tilling with 1/2 overlap (shifted tilling) was used (see *e.g.* Fig. 5.5b). Shifted tilling increased the number of tiles produced to 29,100, where 26,190 was used for training and 2,910 was used for validation. Splitting tiles for training and validation was performed randomly with a 90% to 10% ratio, as was done in the previous works [12][13][14][15]. Increasing the validation dataset ratio might provide difficulties with training on the classes with low presence. Additionally, empty tiles were not included in any of the datasets.



(a) Baseline tiling of the map.

(b) Tilling of the map with 1/2 shifting.

Figure 5.5: Illustration of the tilling [15].

The resolution of the Lake Winnipeg watershed satellite image is 64394 x 48859 pixels, which results in approximately 85,400 tiles using the aforementioned augmentation methods. From the generated tiles, 76,800 were used for training and 8,500 for validating. When it comes to this size, random distribution between training and validation datasets is not desirable. Therefore, a pseudo-random distribution was implemented with the use of large blocks (see *e.g.* Fig. 5.6) to ensure that training and validation sets contain all classes and tiles from all locations. Each tile is assigned to one of the blocks, where each block is approximately 10000 x 10000 pixels. Then, all tiles in the block were randomly sampled, ensuring that training and validation sets contain all labels from the current block, leading to better training and more accurate testing results. Table 5.4 provides the number of tiles present in all of the generated datasets mentioned above. Note that the Sentinel-2 dataset has no validation tiles due to the absence of the corresponding labels, and its use in the UDA learning framework.

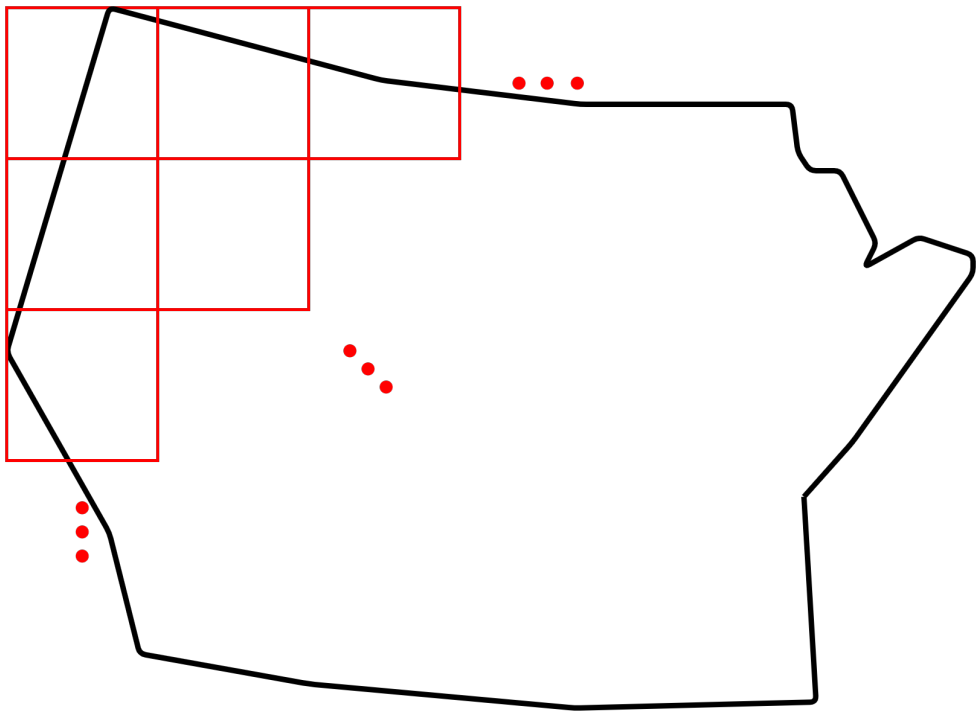


Figure 5.6: Illustration of the blocking, where each red square is 10000 x 10000 pixels.

Table 5.4: Size of the generated datasets.

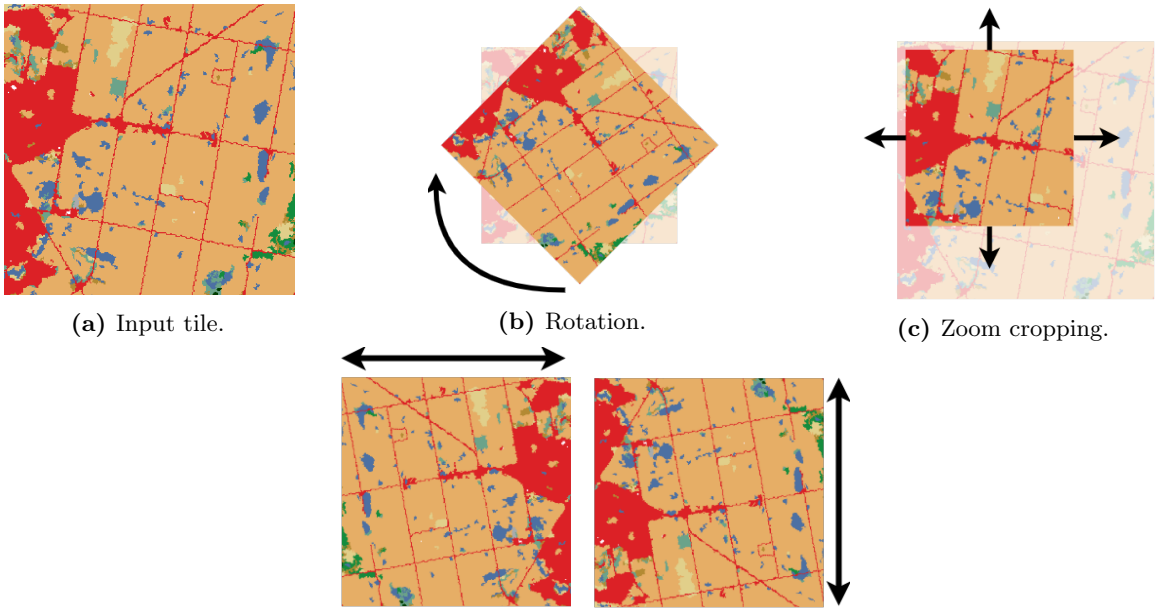
	Landsat 5/7 Southern extent of Manitoba	**Landsat 5/7** Lake Winnipeg watershed	**Landsat 8** Lake Winnipeg watershed	**Sentinel-2** Southern extent of Manitoba
Training images	∼29 100	∼85 400	∼107 900	∼34 412
Validation images	∼2 900	∼8 500	∼12 000	-

5.2.2 Data Transformation

Training DL models often results in either overfitting or underfitting [56] that is caused by a lack of variation in the dataset or bad model architecture. One of the simplest ways to resolve this issue is to shuffle data after every epoch and apply random data transformation on each image [86][87]. Data transformation ensures that tiles are not represented in the same way during training, and it is achieved by randomly applying one or more of the following operations.

- Rotation – This transformation rotates the image by 90 degrees 1-4 times (as shown in Fig. 5.7b).
- Flipping – This transformation flips the image left to right, upside down or both (as shown in Fig. 5.7d).
- Zoom cropping – This transformation upscales image on a scale between 10% and 25%, then crops 224 x 224 image at a random position (as shown in Fig. 5.7c).

Note that, unlike augmentation, transformation is used during training and not before. Also, it does not increase the number of generated tiles.



(a) Input tile. **(b)** Rotation. **(c)** Zoom cropping. **(d)** Flipping.

Figure 5.7: Transformation of the tiles.

5.3 Experimental Setup

Training of the networks was performed using the *TensorFlow* [88] Python library with *Docker* containers[1]. Most of the results were generated using an NVIDIA Digits DevBox[2] containing four Titan X GPUs with 12GB of memory per GPU, 64 GB DDR4 RAM, and a Core i7-5930K 3.5 GHz processor. Table 5.5 describes the time needed to train a well-performing model and the approximate prediction time of the same model on different datasets. The training on the southern extent of Manitoba dataset varied between 5 and 7 days, while the Lake Winnipeg watershed dataset took between 20 and 30 days. Also, the prediction time of the network on the non-augmented southern extent of Manitoba datasets took 8 minutes for the Landsat 5/7 dataset, and 15 minutes for the Sentinel-2 dataset. Additionally, prediction of the Lake Winnipeg watershed datasets took close to 1.5 hours for both sensors. Most of the networks were trained with a batch size of 16 on a model that was distributed across 3 GPUs, leading to a global batch size of 48. However, some network variations used a lower batch size due to computational complexities. As was earlier carried out in [15], training was performed with a learning rate of $\eta = 10^{-4}$ for 100 epochs, then with a learning rate of $\eta = 10^{-5}$ for another 100 epochs using the Adam optimization algorithm [46].

Table 5.5: Comparison of the training and prediction time for each dataset.

	Landsat 5/7 Southern extent of Manitoba	**Landsat 5/7** Lake Winnipeg watershed	**Landsat 8** Lake Winnipeg watershed	**Sentinel-2** Southern extent of Manitoba
Training time	∼5-7 days	∼20 days	∼25 days	∼25 days
Prediction time	∼8-10 minutes	∼1.5-2 hours	∼1-1.5 hours	∼15-20 minutes

5.4 Chapter Summary

This chapter presented an overview of datasets used in this thesis, like the southern extent of Manitoba, and the Lake Winnipeg watershed. Additionally, characteristics of satellite sensors were compared, like Landsat 5/7, Landsat 8, and Sentinel-2. Also, preprocessing methods and techniques were discussed, in particular, tiling, blocking, and transformation. Lastly, an experimental setup was presented, where the system and related software were introduced.

[1] https://www.tensorflow.org/install/docker
[2] https://developer.nvidia.com/devbox

Chapter 6

Experiments, Results, and Analysis

In this chapter, we evaluate and compare the performance of different model variations, which were trained on the Landsat 5/7 southern extent of Manitoba. Then, based on the results, the best model combination was selected to train models using Landsat 5/7 and Landsat 8 Lake Winnipeg watershed datasets. Using a model trained on the Landsat 5/7 dataset will allow us to develop maps from 1984 to 2017, and a model trained on the Landsat 8 dataset will allow us to develop maps starting from 2013 and forward in time. Moreover, the chosen network was also trained with an UDA architecture for the problem of transferring perceptual knowledge from the labelled Landsat 8 dataset (source) to the unlabelled Sentinel-2 dataset (target). This experiment was carried out to see if it is possible to generate accurate LULC maps on the data from the sensor that has no corresponding labels. Lastly, the results of all generated products are discussed and assessed. The overall training process is depicted in Fig. 6.1.

6.1 Evaluation Metrics

The performance of the trained models must be evaluated based on some sort of metrics. One of the most common approaches is to use the loss calculated during training, but the loss function can provide vague, non-intuitive results (especially when models are performing well). Also, comparison of loss values can be meaningless when evaluating different architectural designs, because they often use different loss functions. In this thesis, pixel accuracy, precision, recall, F1-score, and critical success index (CSI)[1] were used to evaluate the results. To begin, pixel accuracy is a metric that calculates the ratio between the number of correctly classified pixels and a total number of pixels in the image, thus

[1] CSI is also known as a threat score.

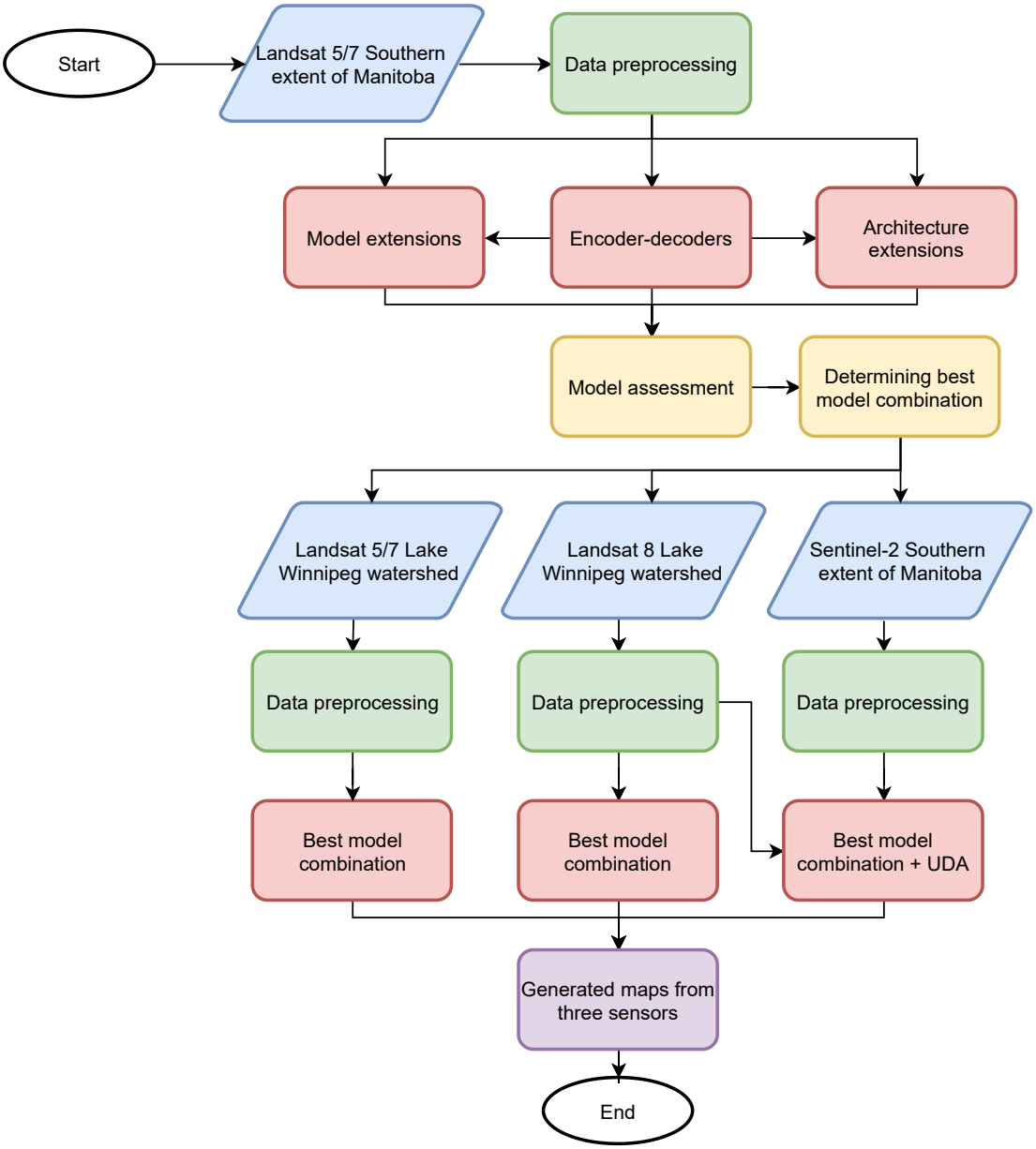


Figure 6.1: Training flowchart, where the blue shapes correspond to data, green shapes correspond to preprocessing, red shapes correspond to models and extensions, yellow shapes correspond to model assessment, and purple shapes correspond to final generated maps.

providing a general assessment of the prediction, which can be easily compared and understood. Define n_{ij} as the number of pixels in the image with ground-truth label i and corresponding predicted label j. Let $t_i = \sum_{j=1}^{C} n_{ij}$ denote the total number of pixels labelled with label i, C is the number of classes, n_{ii} are the number of pixels correctly predicted, and n_{ji} are the number of incorrectly label pixels (with

respect to label i). [15]. The pixel accuracy is defined as

$$Accuracy = \frac{\sum_{i=1}^{C} n_{ii}}{\sum_{i=1}^{C} t_i}. \tag{6.1}$$

Based on the pixel accuracy, the best model combination was then trained on bigger datasets. Next, we will define, precision, recall, F1-score, and CSI. Let class i be labelled a *positive class*, and every other class labelled a *negative class*. Then, the true positive, TP_i, is defined as the number of correctly predicted pixels of the positive class, true negative TN_i as the number of correctly predicted pixels of the negative class, false positive FP_i as the number of incorrectly predicted pixels of the positive class, and false negative FN_i as the number of incorrectly predicted pixels of the negative class. Given these definitions, we can then define the following metrics.

- Precision - This method calculates how precise are the predictions for the positive class [89]. Also, it represents how accurate the model performed for detecting the relevant features. The precision is computed as
$$Precision_i = \frac{TP_i}{TP_i + FP_i}. \tag{6.2}$$

- Recall - This method calculates how many of the actual positives are correctly labelled by the model [89]. Also, it reflects the ability of a model to retrieve all the relevant elements within a dataset. The recall is computed as
$$Recall_i = \frac{TP_i}{TP_i + FN_i}. \tag{6.3}$$

- F1-score - This method provides a method for combining recall and precision to get a single measure by differentially weighting the recall and precision values [90]. The F1-score is computed as
$$F_i = 2 * \frac{Precision_i * Recall_i}{Precision_i + Recall_i}. \tag{6.4}$$

- CSI - This method calculates the worst possible scenario of predictions by including not classified and misclassified positives in the calculation [91]. Also, the calculation of the CSI score is identical to intersection over union (IoU) [92]. The CSI is computed as
$$CSI_i = \frac{TP_i}{TP_i + FP_i + FN_i}. \tag{6.5}$$

6.2 Comparison of Model Variations

All model combinations introduced in Chapter 4 were trained on the southern extent of Manitoba and assessed on the validation dataset. Due to time and complexity limitations, some model variations were ignored. Table 6.1 on the left shows the results of the trained decoders paired with the VGGNet encoder. Based on the results, the modified U-Net decoder was chosen as the main decoder network because it has 3 times fewer parameters than plain U-Net with a cost of 0.04% of pixel accuracy. Moreover, using the modified U-Net decoder allowed for the implementation of more advanced structures elsewhere. Also, the modified U-Net decoder is more generic and is compatible with any encoders. Table 6.1 on the right displays the results of all implemented encoders in combination with the Modified U-Net decoder. VGGNet proved to have the highest pixel accuracy of 90.11%, followed by ResNet at 89.05%.

Table 6.1: Results of the encoder-decoder model variations.

Network	Pixel accuracy
FCN-8	88.28
U-Net	**90.15**
Feedbackward	89.59
Modified U-Net	**90.11**

(a) Results from the trained VGGNet encoder with different decoders.

Network	Pixel accuracy
VGGNet	**90.11**
GoogleNet	83.14
Xception	88.36
ResNet	**89.05**

(b) Results from the trained different encoders with modified U-Net decoder.

Additionally to encoder and decoder experiments, model extensions were also tested. From the previous test VGGNet and ResNet encoders with the modified U-Net decoder were chosen. Table 6.2 shows the results of those two networks trained on OS ranging from 32 to 2, where the OS 32 is the same as the models without an extension. Both models performed the best with the OS 4, where VGGNet and ResNet reached 92.4% and 91.42% pixel accuracy, respectively. The ASPP and the context module model extensions provided a slight performance increase with the networks with the OS 32. However, both networks did not benefit from these extensions with the OS 4. Therefore, ASPP and context module extensions were ignored in further tests.

Lastly, VGGNet and ResNet with and without OS extension were trained on different architectural designs, examples include GAN and progressive GAN (see *e.g.* Table 6.3). Even though GAN provided a slight performance increase for the ResNet model with no model extension, other networks performed identically to the one without, and in some cases slightly worse. Progressive GAN and traditional GAN, did not provide any increase in performance. Based on the results presented below, adversarial training had a lesser or no effect at all on the models that performed well on the given dataset. Also, the ResNet Deeplabv3+ architecture was trained on our dataset, as it is considered a state-of-the-art

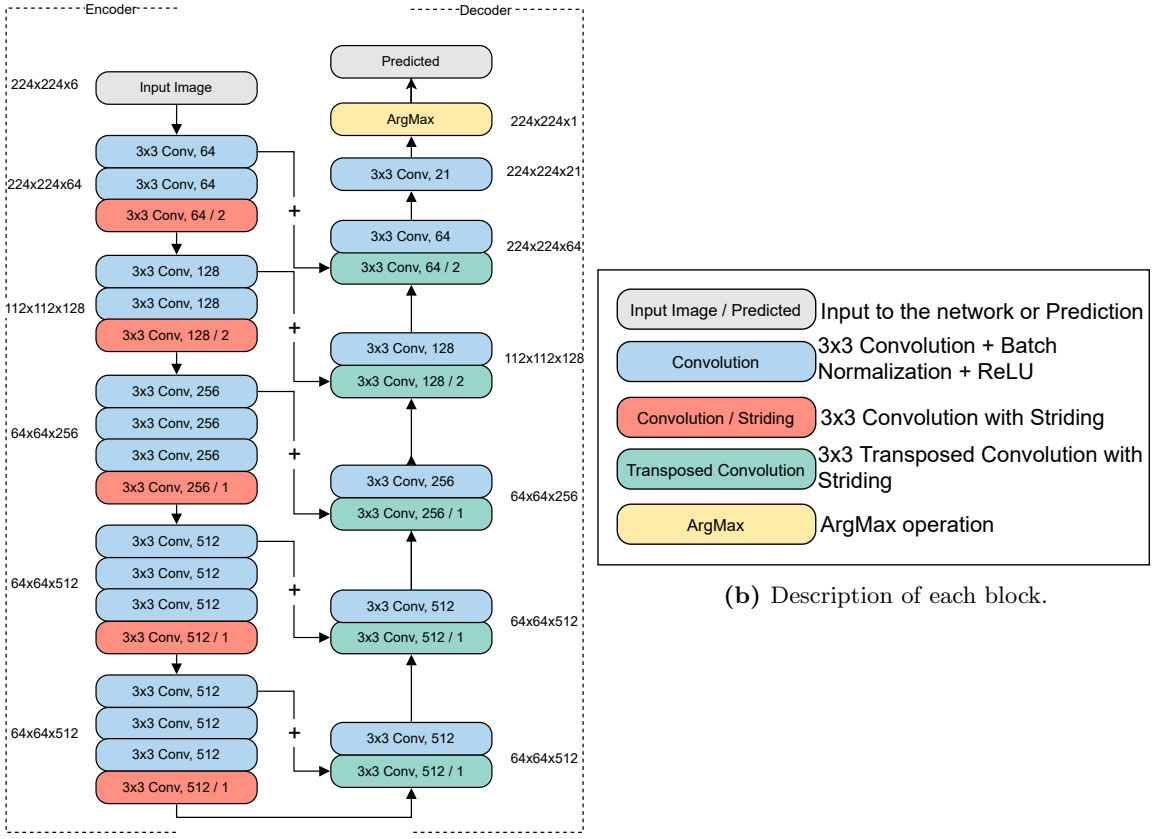
Table 6.2: Pixel accuracy results from the trained networks with extensions.

OS	32	16	8	4	2
Network					
VGGNet	90.11	90.66	91.25	**92.4**	91.01
VGGNet + ASPP	90.72	-	-	92.22	-
VGGNet + context module	90.18	-	-	92.04	-
ResNet	89.05	89.72	90.02	**91.42**	-
ResNet + ASPP	-	-	-	91.37	-
ResNet + context module	-	-	-	90.31	-

semantic segmentation model, but it showed low performance in our dataset reaching 87.43% of pixel accuracy. Based on all experiments, the model with the best performance and lowest complexity was VGGNet with modified U-Net and OS 4, which reached 92.4% on the southern extent of the Manitoba validation dataset. As the pixel accuracy of the model approaches 100% it is getting more difficult to improve its performance by introducing changes and enhancement to the model and architecture. In the papers and our experiments, we can see that extensions, like ASPP, context module and GAN can improve the performance of the model. However, if the model already performs well implementing or even combining them will not provide further benefit, and in some cases even aggravate the results. The best performing network architecture is depicted in Fig. 6.2. Next, the worst results based on pixel accuracy are shown in Fig. 6.3. Similarly, the best results based on pixel accuracy are shown in Fig. 6.4.

Table 6.3: Pixel accuracy results from the trained networks with different architectural designs.

Network \Architectures	Regular	Deeplabv3+	GAN	Progressive GAN
VGGNet	90.11	-	89.99	90.09
VGGNet + OS 4	**92.4**	-	92.37	92.41
ResNet	89.05	-	89.26	-
ResNet + OS 4	91.42	87.43	91.04	-



(a) Structure of the best model.

(b) Description of each block.

Figure 6.2: Network architecture diagram for best model combination: VGGNet encoder with Modified U-Net decoder and OS 4.

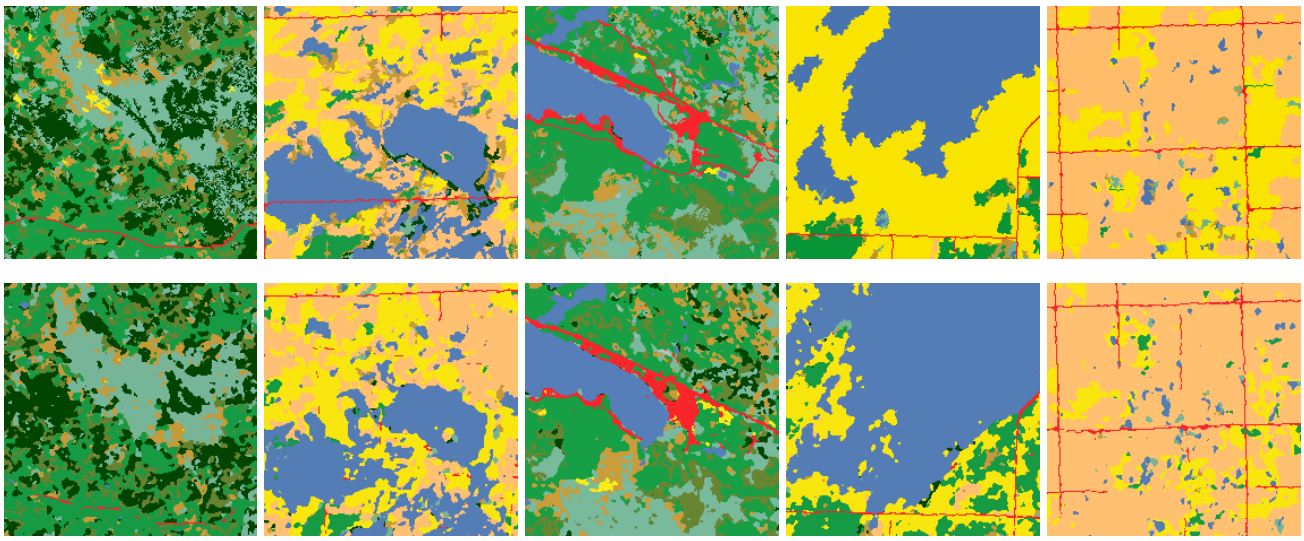


Figure 6.3: Examples of poor results generated by the best model trained on the Landsat 5/7 data from the southern extent of Manitoba. The first row are ground truth tiles, and second row are predicted tiles.

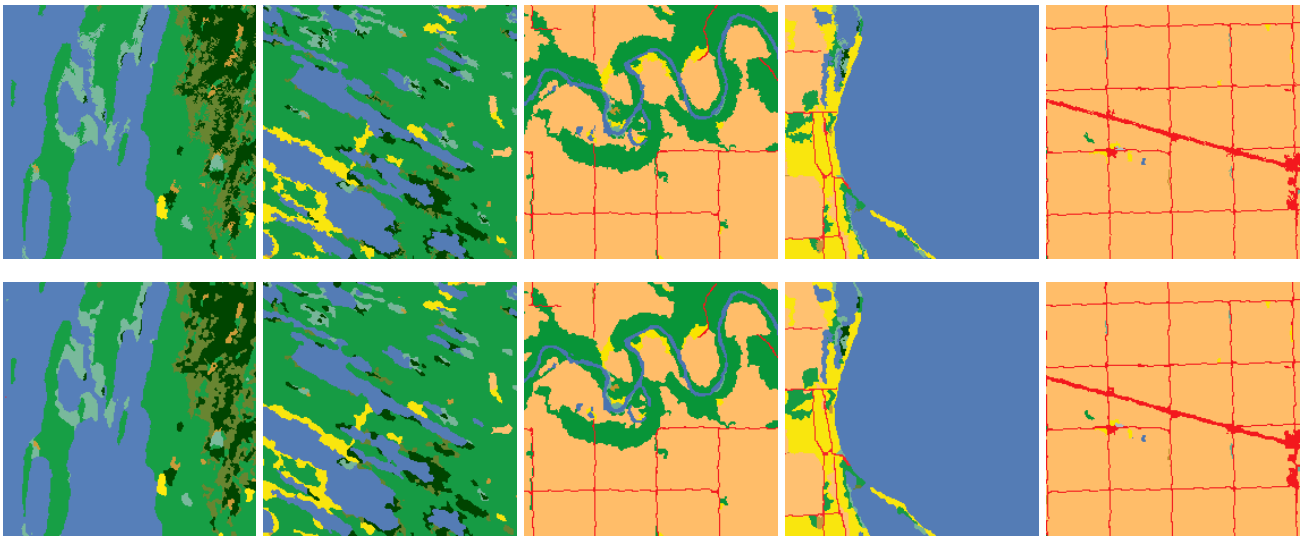


Figure 6.4: Examples of good results generated by the best model trained on the Landsat 5/7 data from the southern extent of Manitoba. The first row are ground truth tiles, and second row are predicted tiles.

6.3 Product Generation

The next step, after determining the model combination that performed the best on the southern extent of Manitoba was to generate *products* trained on Lake Winnipeg watershed datasets, where the product is a trained model. As the result, two models were generated: one was trained on the Landsat 5/7 dataset, which allows us to develop maps going back in time; and the second one was trained on Landsat 8 dataset, which allows us to develop maps going forward in time. Moreover, the chosen network was also trained using UDA architecture to transfer knowledge extracted from Landsat 8 dataset to the Sentinel-2 to develop accurate LULC maps from the sensor that does not have corresponding labels. Lastly, the performance of all models was assessed and presented in the form of tables.

6.3.1 Landsat 5/7

First, the best model was trained with Landsat 5/7 data from the Lake Winnipeg watershed. The number of pixels per class and their percentage is shown in Table 6.4. Note that some classes have a really low presence, *e.g.* 11, 12, 13 (lichen-moss-related classes) and 19 (snow and ice). On the other hand, class 0 (no data) has the most pixels due to the non-rectangular shape of the map (see, *e.g.*, Fig. 5.4). No data labels were mostly ignored in the training as they do not provide any useful information. Full no data labelled tiles were not included in the training and validation datasets, but

tiles with the presence of any other label were included. Table 6.5 on the left represents a per-class assessment of the dataset and on the right shows a global assessment of the predicted map. Additionally, the error matrix, also known as a confusion matrix, is presented in Table 6.6. Based on the results, well-performing classes had a higher percentage of pixels than others, *e.g.* classes 1, 5, 10, 15, 16, 18, 20. While classes with a low percentage of pixels performed poorly, especially 11, 12, 13 and 19 because it is well known that deep learning models need a large number of class examples in order to achieve good performance. Also, forest-related classes with low presence were the most often misclassified with class 1, and the urban and built-up class was misclassified with cropland. Moreover, predicted clouds have extremely high results because they were generated by a thresholding method [84], and the network did not have any difficulties mimicking this method due to it being a straightforward function with sufficient data examples. Overall, performance of the model reached 80.66% pixel accuracy on the validation dataset, where only a few tiles had no data pixels. The pixel accuracy presented in this dataset is lower due to the size and amount of extractable features presented in this dataset. Also, thresholding methods are known to provide inaccurate results. Cloud labels generated by the Otsu thresholding method did not classify some of the clouds, especially transparent ones, and misclassified bright urban areas with the clouds. We suggest that this was one of the factors that influenced the performance of the model. The worst results based on pixel accuracy are shown in Fig. 6.5. Similarly, the best results based on pixel accuracy are shown in Fig. 6.6. The generated maps based on the Landsat 5/7 dataset are presented in the Appendix A in the Fig. A.2, and Fig. A.4 with their corresponding ground truth labels in the Fig. A.1, and Fig. A.3.

Table 6.4: Total number of labels per pixel of the Landsat 5/7 dataset.

Class	Name of class	# of pixels	% of pixels
0	None	1031407726	32.7824
1	Temperate or sub-polar needleleaf forest	488262071	15.5190
2	Sub-polar taiga needleleaf forest	30020347	0.9542
5	Temperate or sub-polar broadleaf deciduous forest	170429106	5.4169
6	Mixed forest	96116915	5.4169
8	Temperate or sub-polar shrubland	183620158	3.0550
10	Temperate or sub-polar grassland	157337200	5.0008
11	Sub-polar or polar shrubland-lichen-moss	4117649	0.1309
12	Sub-polar or polar grassland-lichen-moss	11029343	0.3506
13	Sub-polar or polar barren-lichen-moss	2302774	0.0732
14	Wetland	156313523	4.9683
15	Cropland	432590674	13.7495
16	Barren land	34526845	1.0974
17	Urban and built-up	26771188	0.8509
18	Water	229553602	7.2962
19	Snow and ice	936557	0.0298
20	Cloud	90890767	2.8889

Table 6.5: Assessment of the predicted Landast 5/7 dataset.

Class	CSI	Precision	Recall	F1-score
0	99.76	99.78	99.98	99.88
1	74.99	84.15	87.33	85.71
2	42.78	62.8	57.29	59.92
5	57.41	75.7	70.39	72.95
6	40.78	63.27	53.42	57.93
8	49.67	65.29	67.5	66.38
10	66.33	78.9	80.63	79.76
11	21.68	55.47	26.24	35.63
12	42.36	57.97	61.13	59.51
13	37.76	53.65	56.04	54.82
14	58.08	73.75	73.21	73.48
15	87.84	92.61	94.46	93.53
16	72.2	82.01	85.79	83.85
17	43.61	63.08	58.56	60.74
18	86.36	94.45	90.98	92.68
19	58.75	67.31	82.21	74.02
20	97.58	97.76	99.81	98.78

(a) Per-class assessment.

Accuracy	Accuracy (excluding None)	Accuracy (Validation Dataset)
88.19	82.44	80.66

(b) Pixel accuracy.

	CSI	Precision	Recall	F1-score
Average	57.66	70.44	69.17	69.42
Weighted Average	80.79	88.04	88.19	88.08
Average (excluding None)	55.19	68.72	67.35	67.63
Weighted Average (excluding None)	71.54	82.32	82.44	82.33

(c) Global assessment.

Table 6.6: Confusion matrix of the Landsat 5/7 dataset, where rows represent ground truth labels, and columns represent predicted labels.

	0	1	2	5	6	8	10	11	12	13	14	15	16	17	18	19	20
0	99.98	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.00	0.00	0.00	0.00	0.00	0.00
1	0.00	87.33	0.48	0.77	2.72	3.65	0.45	0.01	0.04	0.00	3.30	0.05	0.15	0.10	0.87	0.00	0.09
2	0.01	9.74	57.29	0.01	0.05	7.64	6.99	0.27	1.73	0.15	12.09	0.00	2.84	0.06	1.00	0.00	0.14
5	0.00	3.42	0.00	70.39	7.92	9.71	1.73	0.04	0.01	0.00	1.58	4.02	0.01	0.45	0.60	0.00	0.11
6	0.00	22.56	0.08	17.75	53.42	3.21	0.22	0.00	0.00	0.00	1.61	0.46	0.01	0.26	0.31	0.00	0.11
8	0.00	11.49	0.68	4.30	0.66	67.49	5.11	0.11	0.55	0.03	6.39	1.81	0.17	0.42	0.70	0.00	0.09
10	0.22	2.22	1.39	0.92	0.11	5.61	80.63	0.04	0.41	0.12	1.08	4.14	1.60	0.64	0.80	0.00	0.05
11	0.17	2.03	1.48	0.34	0.00	14.76	18.03	26.24	28.39	1.39	2.66	0.00	0.32	0.00	3.75	0.01	0.43
12	0.03	2.89	6.72	0.00	0.00	10.48	2.12	1.57	61.13	5.69	3.50	0.03	3.36	0.01	2.07	0.00	0.40
13	0.03	0.54	2.69	0.00	0.00	1.12	0.51	1.70	30.62	56.03	0.73	0.00	2.89	0.00	2.45	0.00	0.68
14	0.03	10.94	1.64	1.70	0.70	7.30	1.13	0.08	0.18	0.01	73.21	1.38	0.26	0.20	1.16	0.00	0.09
15	0.40	0.03	0.00	0.84	0.04	0.33	2.23	0.00	0.00	0.00	0.08	94.46	0.01	1.20	0.26	0.00	0.13
16	0.06	3.51	1.55	0.09	0.02	1.11	3.29	0.04	0.45	0.12	1.04	0.42	85.79	0.41	0.82	1.00	0.28
17	0.22	2.57	0.01	2.49	0.64	1.74	4.51	0.00	0.00	0.00	0.39	28.04	0.14	58.56	0.53	0.00	0.15
18	0.03	2.48	0.16	0.59	0.08	0.79	1.02	0.02	0.08	0.03	0.81	2.34	0.44	0.08	90.98	0.00	0.06
19	0.13	0.31	0.01	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.00	0.02	16.09	0.01	0.16	82.21	1.02
20	0.00	0.04	0.00	0.04	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.05	0.01	0.00	0.02	0.01	99.81

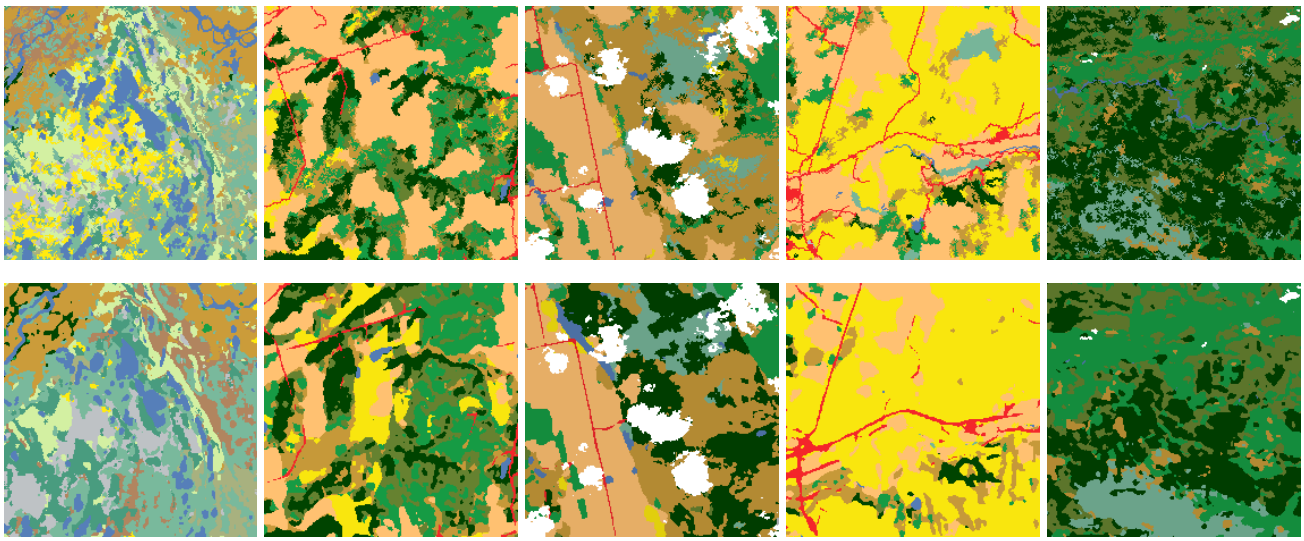


Figure 6.5: Examples of poor results generated by the best model trained on the Landsat 5/7 data from the Lake Winnipeg watershed. The first row are ground truth tiles, and second row are predicted tiles.

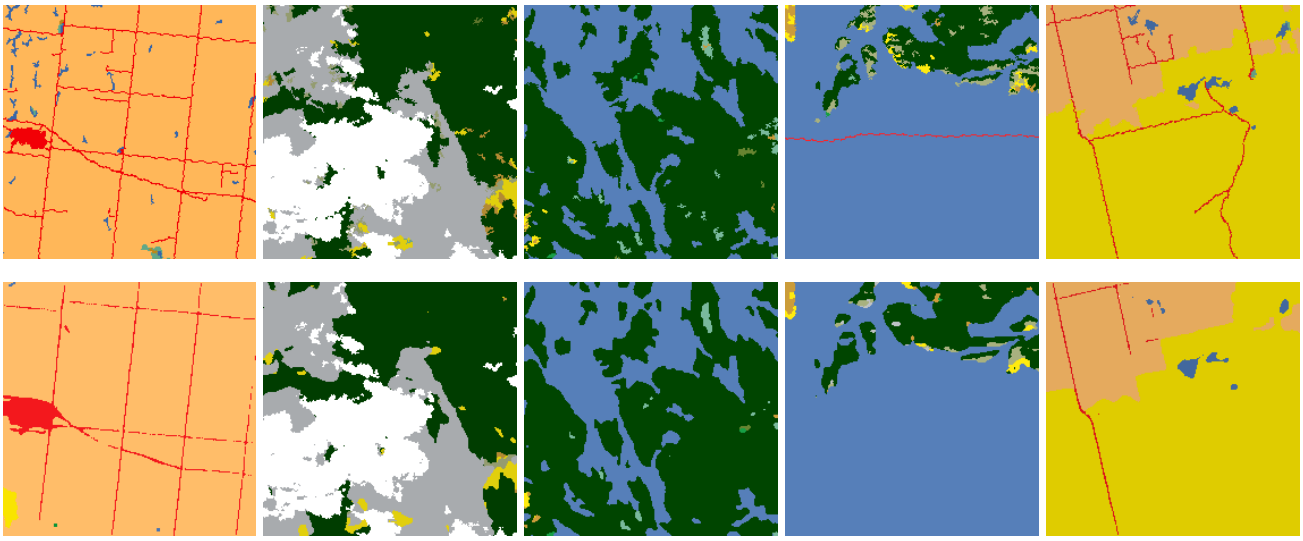


Figure 6.6: Examples of good results generated by the best model trained on the Landsat 5/7 data from the Lake Winnipeg watershed. The first row are ground truth tiles, and second row are predicted tiles.

6.3.2 Landsat 8

Similar to the previous subsection, the best model was trained on the Landsat 8 Lake Winnipeg watershed. In Section 5.1 we described the difference between Landsat 5/7 and Landsat 8 Lake Winnipeg watershed datasets. The number of pixels per class and their percentage is shown in Table 6.7, and the performance of the trained model is presented in Tables 6.8, 6.9. Results provided by the model trained on the Landsat 8 dataset are similar to the Landsat 5/7 model. However, the pixel accuracy of the predicted map increased by 8%, and improvements in accuracy were noticed across most of the classes. These improvements are achieved by providing a 16-bit dataset instead of 8-bit, which made it easier to distinguish features due to a larger range of values. Note that the cloud class accuracy here is slightly lower and on the level with the best-performing classes because the cloud mask for Landsat 8 was generated from the QA band provided by the sensor and not the thresholding method. The worst results based on pixel accuracy are shown in Fig. 6.7. Similarly, the best results based on pixel accuracy are shown in Fig. 6.8. The generated maps based on the Landsat 8 dataset are presented in the Appendix A in the Fig. A.6 with corresponding ground truth map, which is shown in the Fig. A.5.

Table 6.7: Total number of labels per pixel of the Landsat 8 dataset.

Class	Name of class	# of pixels	% of pixels
0	None	1208979489	50.8798
1	Temperate or sub-polar needleleaf forest	101124573	4.2558
2	Sub-polar taiga needleleaf forest	568110	0.0239
5	Temperate or sub-polar broadleaf deciduous forest	106208424	4.4698
6	Mixed forest	52604376	2.2139
8	Temperate or sub-polar shrubland	36149337	1.5213
10	Temperate or sub-polar grassland	98413645	4.1417
11	Sub-polar or polar shrubland-lichen-moss	4270	1.80E-04
12	Sub-polar or polar grassland-lichen-moss	40467	1.70E-03
13	Sub-polar or polar barren-lichen-moss	178	7.49E-06
14	Wetland	89369338	3.7611
15	Cropland	481744136	20.2742
16	Barren land	12135657	0.5107
17	Urban and built-up	26556386	1.1176
18	Water	117838940	4.9592
19	Snow and ice	534825	2.25E-02
20	Cloud	43874765	1.8465

Table 6.8: Assessment of the predicted Landast 8 dataset.

Class	CSI	Precision	Recall	F1-score
0	99.99	99.99	99.99	99.99
1	76.21	84.65	88.44	86.5
2	41.62	67.21	52.22	58.77
5	71.37	83.11	83.48	83.29
6	56.03	74.1	69.67	71.82
8	48.61	70.48	61.04	65.42
10	81.44	90.12	89.43	89.77
11	17.62	54.06	20.73	29.97
12	21.25	59.3	24.87	35.04
13	37.27	66.13	46.07	54.31
14	75.76	85.78	86.64	86.21
15	93.12	95.77	97.11	96.44
16	70.79	84.13	81.7	82.9
17	57.79	74.52	72.01	73.24
18	88.25	95.04	92.51	93.76
19	69.57	86.6	77.96	82.05
20	91.51	94.81	96.33	95.56

(a) Per-class assessment.

Accuracy	Accuracy (excluding None)	Accuracy (Validation Dataset)
95.11	90.06	88.04

(b) Pixel accuracy.

	CSI	Precision	Recall	F1-score
Average	64.60	80.34	72.95	75.59
Weighted Average	91.49	95.04	95.11	95.07
Average (excluding None)	62.39	79.11	71.26	74.07
Weighted Average (excluding None)	82.68	89.92	90.06	89.97

(c) Global assessment.

Table 6.9: Confusion matrix of the Landsat 8 dataset, where rows represent ground truth labels, and columns represent predicted labels.

	0	1	2	5	6	8	10	11	12	13	14	15	16	17	18	19	20
0	100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
1	0.01	88.44	0.04	1.26	3.96	1.18	0.47	0.00	0.00	0.00	2.86	0.13	0.33	0.14	0.83	0.00	0.37
2	0.04	14.72	52.23	0.00	0.09	0.19	5.31	0.01	0.16	0.00	1.71	0.00	24.58	0.02	0.14	0.00	0.79
5	0.01	1.54	0.00	83.48	5.64	2.38	0.91	0.00	0.00	0.00	1.98	2.64	0.08	0.44	0.67	0.00	0.24
6	0.01	12.22	0.01	11.81	69.67	1.13	0.22	0.00	0.00	0.00	2.97	0.40	0.05	0.36	0.85	0.00	0.31
8	0.01	6.26	0.00	10.92	2.29	61.04	6.49	0.00	0.00	0.00	5.36	4.66	0.83	1.03	0.85	0.00	0.26
10	0.01	0.74	0.02	1.02	0.11	2.40	89.43	0.00	0.00	0.00	0.75	3.58	0.41	0.72	0.73	0.00	0.10
11	0.00	32.72	6.11	0.00	1.83	12.81	23.33	20.73	0.02	0.00	0.00	0.00	2.13	0.00	0.14	0.00	0.19
12	0.10	2.56	8.60	0.08	0.01	0.94	16.52	0.04	24.87	0.00	36.40	0.08	8.64	0.13	0.52	0.00	0.52
13	0.00	33.15	0.00	5.06	0.00	14.05	0.00	0.00	0.00	46.07	1.12	0.00	0.00	0.00	0.56	0.00	0.00
14	0.01	3.57	0.00	1.98	1.22	1.23	0.71	0.00	0.00	0.00	86.64	2.92	0.25	0.32	0.87	0.00	0.29
15	0.00	0.01	0.00	0.45	0.02	0.14	0.57	0.00	0.00	0.00	0.38	97.11	0.01	0.87	0.32	0.00	0.11
16	0.06	4.72	0.59	0.99	0.22	2.37	3.30	0.00	0.00	0.00	2.63	0.93	81.70	0.35	0.55	0.35	1.24
17	0.00	0.71	0.00	1.95	1.08	1.00	2.75	0.00	0.00	0.00	1.13	18.65	0.13	72.01	0.39	0.00	0.20
18	0.00	0.67	0.00	0.71	0.27	0.19	0.96	0.00	0.00	0.00	0.76	3.53	0.04	0.11	92.51	0.00	0.27
19	0.38	0.08	0.00	0.00	0.00	0.01	0.04	0.00	0.00	0.00	0.05	0.02	16.84	0.01	0.07	77.96	4.54
20	0.00	0.60	0.00	0.37	0.16	0.07	0.13	0.00	0.00	0.00	0.48	1.01	0.34	0.07	0.40	0.05	96.34

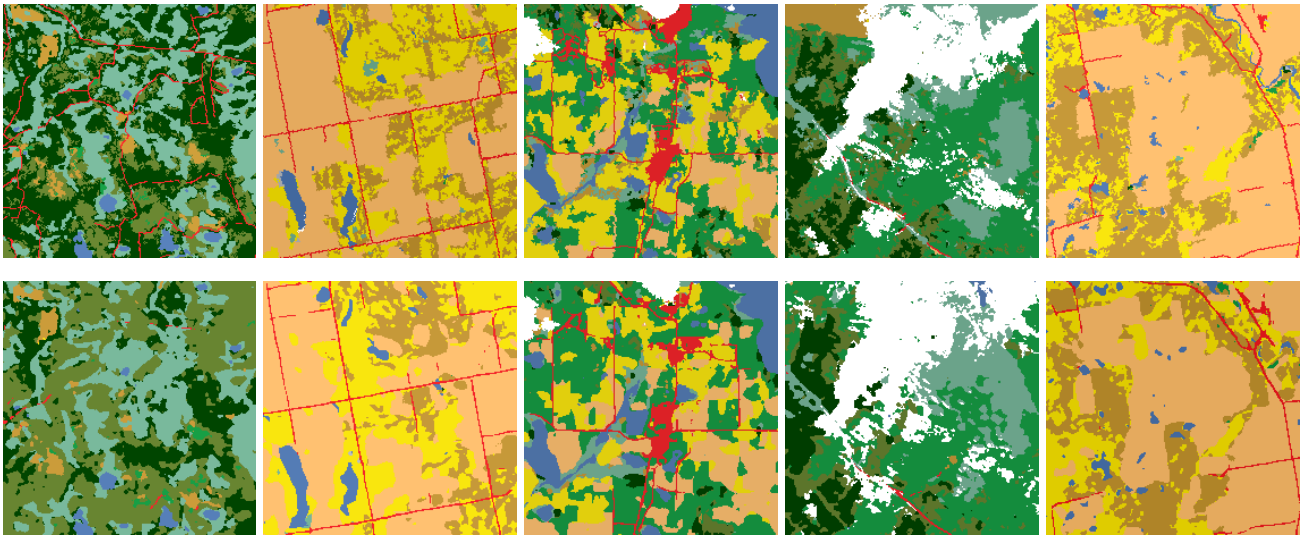


Figure 6.7: Examples of poor results generated by the best model trained on the Landsat 8 data from the Lake Winnipeg watershed. The first row are ground truth tiles, and second row are predicted tiles.

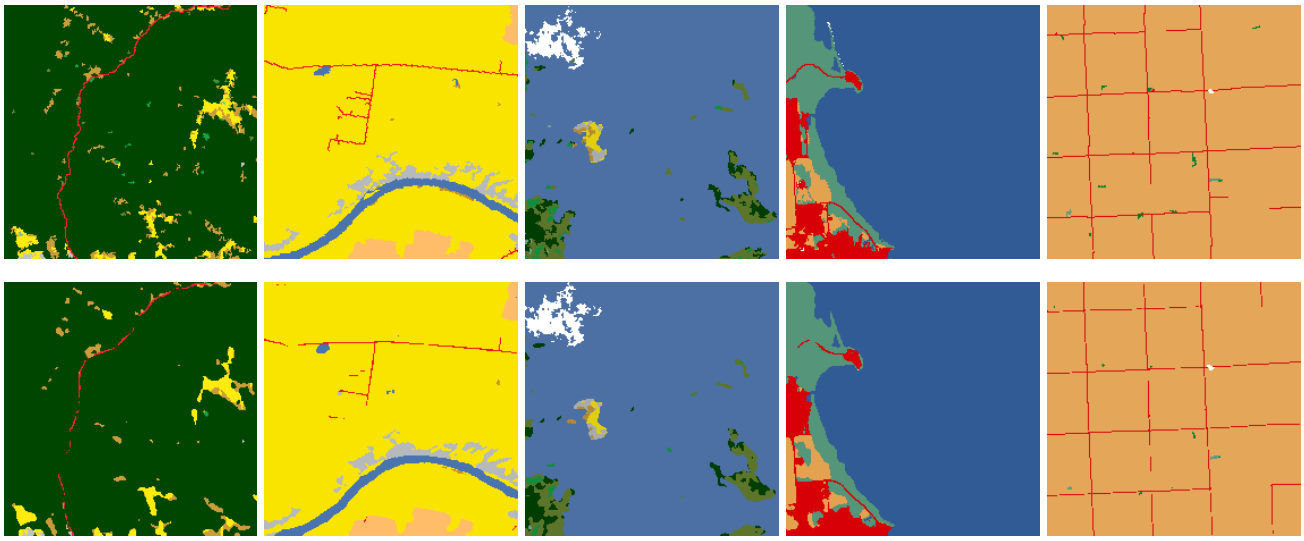


Figure 6.8: Examples of good results generated by the best model trained on the Landsat 8 data from the Lake Winnipeg watershed. The first row are ground truth tiles, and second row are predicted tiles.

6.3.3 Sentinel-2

In the last experiment, the best model was trained using the UDA architecture, where the source domain was the labelled Landsat 8 dataset, and the target domain was the Sentinel-2 dataset. To assess the performance of the model, Landsat 8 NALCMS labels were upscaled to the same resolution as the Sentinel-2 dataset using the nearest neighbour algorithm [53], and, during the evaluation, cloud labels were ignored. Also, the Sentinel-2 dataset was generated from 2018 scenes, while upscaled Landsat 8 NALCMS labels are from the 2017 dataset. This means that the assessment of the model presented below does not fully express the performance of the generated models, and the actual performance can be either worse or better. The pixel distribution of the labels is shown in Table 6.10, and the results provided by the generated model presented in Table 6.11. The model reached 66.72% pixel accuracy and performed relatively well on the cropland and water classes. The worst results based on pixel accuracy are shown in Fig. 6.10. Similarly, the best results based on pixel accuracy are shown in Fig. 6.11. However, generated model misclassified bright large water bodies with clouds, which can be seen in Fig. 6.9. This misclassification could have happened due to higher reflectance on these water bodies making them almost white in the RGB bands, and a big difference in value range between Landsat 8 and Sentinel-2 sensors. Additionally, water from the Sentinel-2 sensor could have resembled clouds to the Conditional Generator part of the UDA architecture from the Landsat 8 sensor, thus making the difference in value range even bigger. To resolve an issue with bright large water bodies, a water mask of the large bodies was generated manually and applied to the predicted map. Then, the

best model was retrained on the corrected dataset (see, *e.g.*, Table 6.12). The newly trained model performed slightly better, and the F1-score of the water class increased by 6%. This experiment has shown that it is possible to transfer extracted knowledge from one sensor and apply them to a sensor with slightly different characteristics. The worst results based on pixel accuracy are shown in Fig. 6.12. Similarly, the best results based on pixel accuracy are shown in Fig. 6.13. The generated maps based on the Sentinel-2 dataset are presented in the Appendix A in the Fig. A.8, and Fig. A.9 with corresponding ground truth labels, which is shown in the Fig. A.7.

Table 6.10: Total number of labels per pixel of the upscaled Landsat 8 dataset.

Class	Name of class	# of pixels	% of pixels
0	None	132129912	27.4273
1	Temperate or sub-polar needleleaf forest	10008201	2.0775
2	Sub-polar taiga needleleaf forest	750	2.00E-04
5	Temperate or sub-polar broadleaf deciduous forest	75579277	15.6886
6	Mixed forest	10328661	2.1440
8	Temperate or sub-polar shrubland	11144587	2.3134
10	Temperate or sub-polar grassland	35604006	7.3906
12	Sub-polar or polar grassland-lichen-moss	1377	2.86E-04
13	Sub-polar or polar barren-lichen-moss	13	2.70E-06
14	Wetland	23585867	4.8959
15	Cropland	134609846	27.9421
16	Barren land	1493685	0.3101
17	Urban and built-up	7763713	1.6116
18	Water	39496305	8.1986

Table 6.11: Assessment of the predicted Sentinel-2 dataset using UDA architecture.

Class	CSI	Precision	Recall	F1-score
0	99.93	99.97	99.97	99.97
1	19.32	43.46	25.82	32.39
2	1.33E-10	100.00	1.33E-10	2.67E-10
5	54.23	75.18	66.06	70.33
6	13.12	43.25	15.85	23.20
8	9.60	21.30	14.88	17.52
10	23.18	39.89	35.62	37.64
12	7.08E-02	2.70	7.26E-02	1.41E-01
13	7.69E-09	100.00	7.69E-09	1.54E-08
14	25.34	40.05	40.83	40.44
15	71.29	78.37	88.75	83.24
16	4.60	21.13	5.55	8.80
17	21.66	39.02	32.76	35.61
18	67.49	87.28	74.85	80.59

(a) Per-class assessment.

Accuracy	Accuracy (excluding None)
75.12	65.72

(b) Pixel accuracy.

	CSI	Precision	Recall	F1-score
Average	29.28	56.54	35.79	37.85
Weighted Average	65.59	76.19	75.12	75.26
Average (excluding None)	23.84	53.2	30.85	33.07
Weighted Average (excluding None)	52.61	67.21	65.72	65.92

(c) Global assessment.

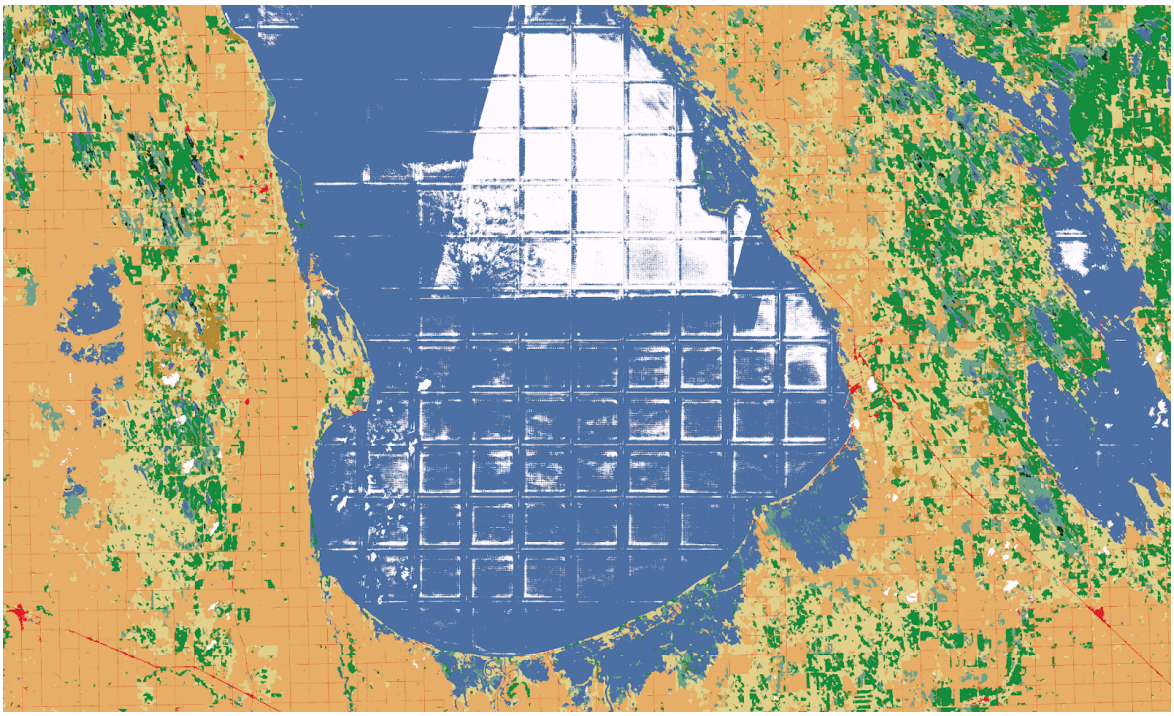


Figure 6.9: Misclassified bright large water bodies by the UDA model.

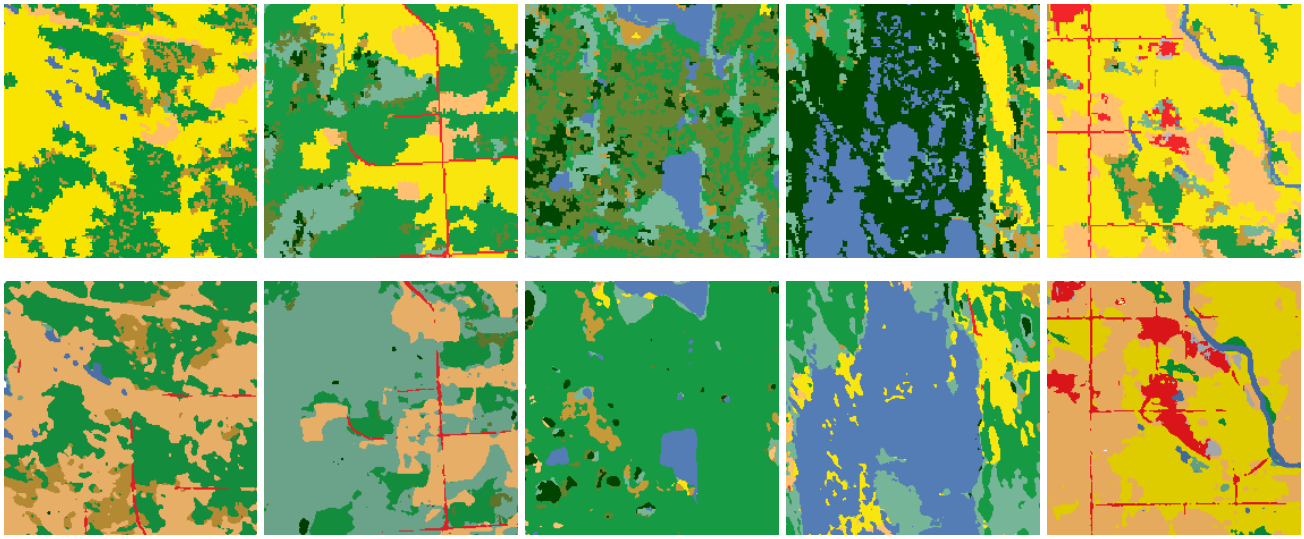


Figure 6.10: Examples of poor results generated by the UDA model trained on the Sentinel-2 southern Manitoba extent. First row is ground truth tiles, and second row is predicted tiles.

Table 6.12: Assessment of the predicted Sentinel-2 dataset using the best model, which was trained on the generated dataset with applied water mask.

Class	CSI	Precision	Recall	F1-score
0	99.93	99.97	99.97	99.97
1	18.14	45.38	23.21	30.71
2	1.33E-10	100.00	1.33E-10	2.67E-10
5	53.92	75.66	65.24	70.07
6	11.53	45.44	13.39	20.68
8	9.54	21.29	14.74	17.42
10	23.20	39.87	35.69	37.66
12	7.26E-11	100.00	7.26E-11	1.45E-10
13	7.69E-09	100.00	7.69E-09	1.54E-08
14	25.97	39.67	42.93	41.24
15	71.29	78.05	89.17	83.24
16	4.16	23.59	4.81	7.99
17	21.26	39.43	31.57	35.07
18	76.19	88.43	84.62	86.48

(a) Per-class assessment.

Accuracy	Accuracy (excluding None)
75.88	66.78

(b) Pixel accuracy.

	CSI	Precision	Recall	F1-score
Average	29.65	64.06	36.09	37.89
Weighted Average	66.22	76.35	75.88	75.64
Average (excluding None)	24.25	61.29	31.18	33.12
Weighted Average (excluding None)	53.48	67.43	66.78	66.44

(c) Global assessment.

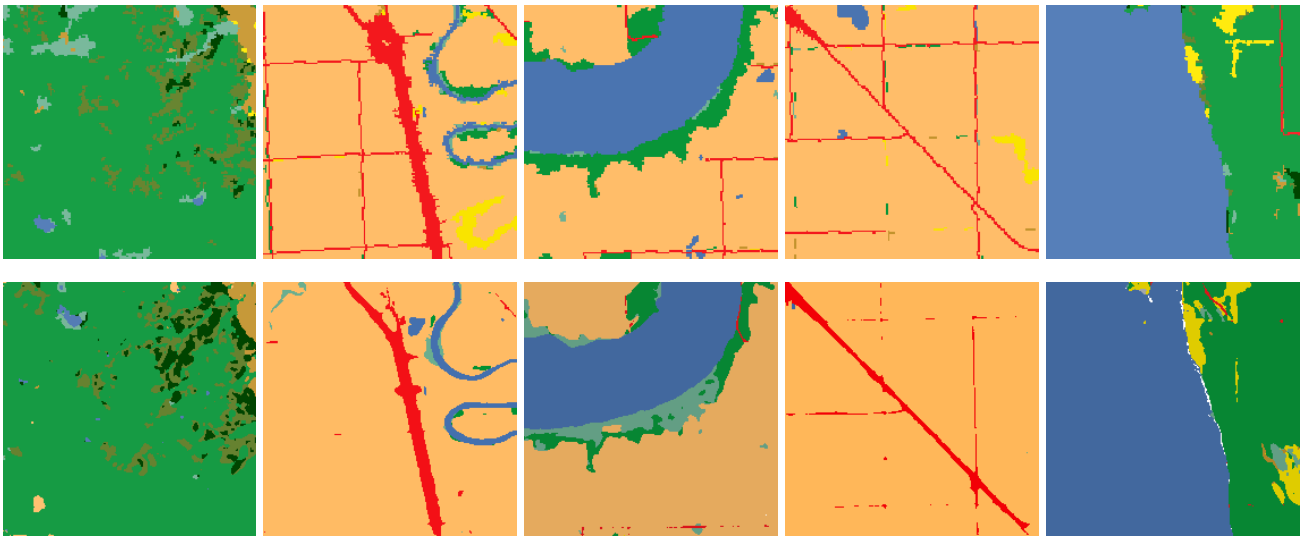


Figure 6.11: Examples of good results generated by the UDA model trained on the Sentinel-2 southern Manitoba extent. First row is ground truth tiles, and second row is predicted tiles.

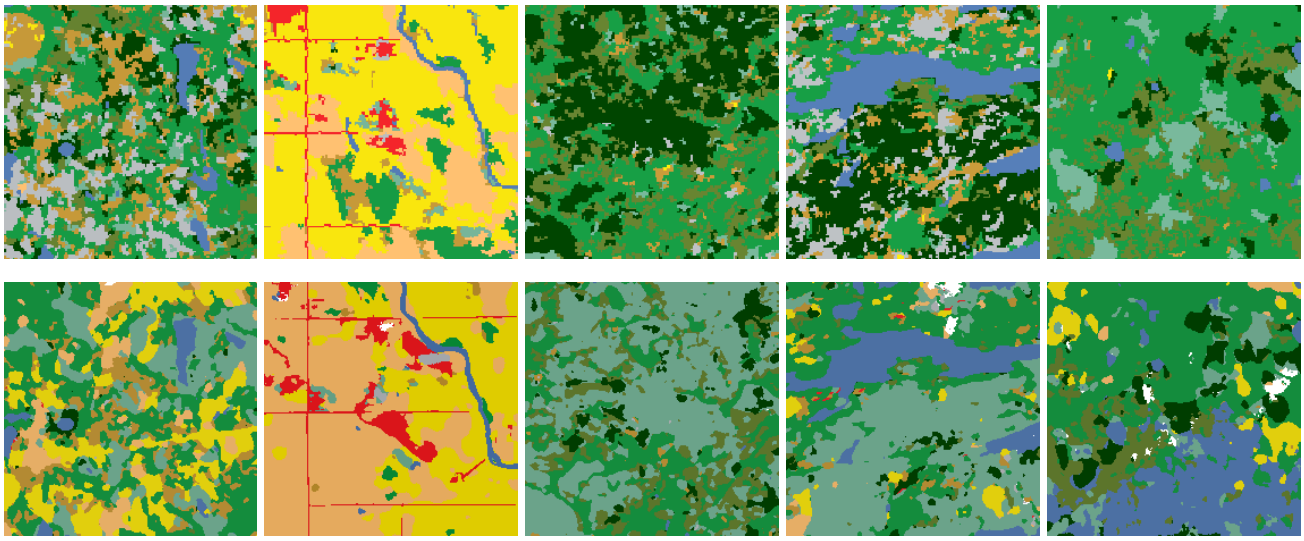


Figure 6.12: Examples of poor results generated by the best model trained on the corrected Sentinel-2 southern Manitoba extent. First row is ground truth tiles, and second row is predicted tiles.

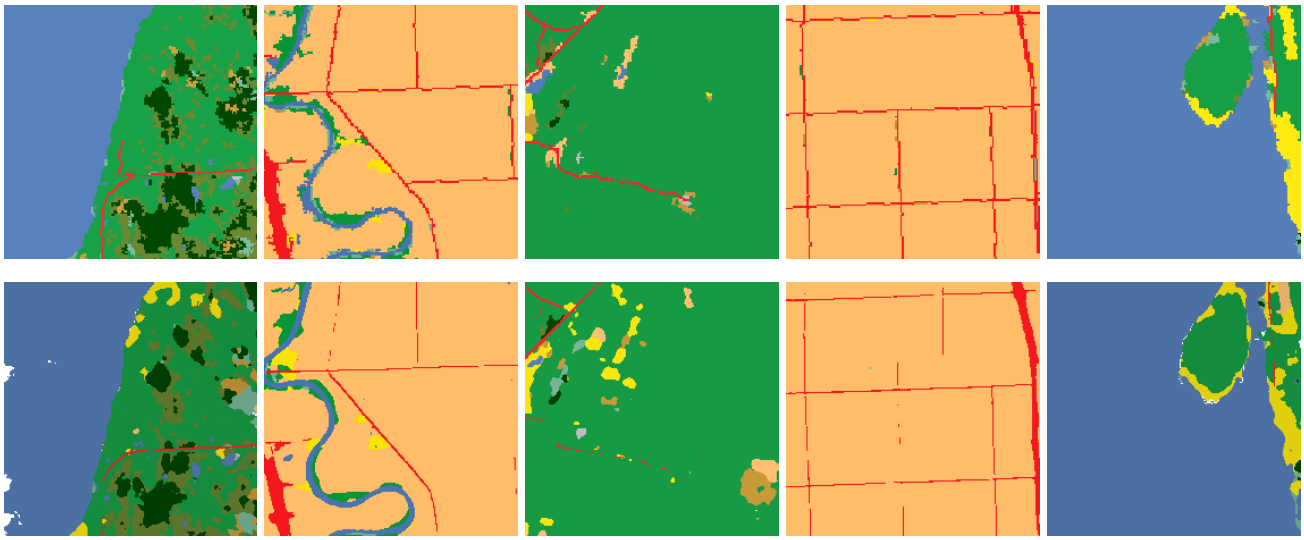


Figure 6.13: Examples of good results generated by the best model trained on the corrected Sentinel-2 southern Manitoba extent. First row is ground truth tiles, and second row is predicted tiles.

6.4 Chapter Summary

This chapter presented the experiments used to find the best encoder-decoder variation. The results have shown that the best model variation is the VGGNet encoder with modified U-Net decoder and OS fixed at 4 that reached 92.4% accuracy. Additionally, this model variation was trained on larger Landsat 5/7 and Landsat 8 datasets that reached 80.66% and 88.04% accuracy, respectively. This shows that with the use of generated models, it's possible to produce fairly accurate LULC maps from mentioned sensors for any year starting from the 1984. Lastly, the UDA learning framework was used to transfer perceptual knowledge from the Landsat 8 dataset to the Sentinel-2 dataset. However, the model had a relatively poor performance reaching 65.72% accuracy.

Chapter 7

Conclusion

In this thesis, we aimed to continue the work presented in [12][13][14] by introducing new and modifying existing NN designs. This was achieved by testing four encoders and decoders, where the best combination was VGGNet with modified U-Net decoder, which reached 90.11% pixel accuracy on the validation dataset. Then, many different extensions were applied to the network, where most of them slightly improved the performance of the network, and only the output stride fixed at 4 pushed prediction accuracy to the 92.4% on the validation dataset. Combining multiple extensions did not provide any improvements. Additionally, different architectural designs were tested in the form of adversarial training and custom encoder-decoder design (Deeplabv3+). Adversarial architecture had a slight improvement on the networks without extensions but did not have any effect on the best variation of the model, while Deeplabv3+ architecture performed poorly compared to other architectures. Based on the results, the better the model performs on the given dataset, the lesser effect adversarial training has. The same pattern can be seen in previous work [15].

Two models were trained on the large Landsat 5/7 and Landsat 8 datasets, separately. The generated Landsat 5/7 LULC map had a lower pixel accuracy of 80.66% due to the difference in size between the southern extent of Manitoba and Lake Winnipeg watershed datasets, the number of extractable features presented in the larger dataset, and the presence of the noise in the form of wrongly classified and not classified clouds. The generated Landsat 8 LULC map had a better pixel accuracy of 88.04% due to the use of 16-bit values instead of 8-bit and more accurate cloud labels. Lastly, the model was trained on the Landsat 8 and Sentinel-2 datasets using UDA architecture. The Sentinel-2 LULC map had a relatively poor performance on most classes except water and cropland. However, these results do not fully represent the performance of the generated model, due to differences in the actual resolution, and years of the developed and ground truth LULC maps, and the fact that cloud labels were ignored

during the evaluation. With the use of generated models, we can produce fairly accurate LULC maps from three sensors for any year starting from the 1984.

This thesis presents a solution for Landsat 5/7, Landsat 8, and Sentinel-2 LULC map generation using the NALCMS dataset. Even so, there are a lot more directions for future work. For example, generating custom encoder-decoder architecture by heavily modifying ResNet and Xception encoders to include many convolutional operations on the original width and height of the input images, thus providing more sharp results, similarly to what was done in the VGGNet. Additionally, modifying the ASPP model extension by altering the dilation rate and order of the convolutional operation, or implementing an alternative counterpart, which will provide better results on the remote sensing datasets. However, one of the most important directions is to increase the performance of the UDA architecture by improving an existing or implementing a different architectural design. For example, including reconstruction of the input to help network extract features from the target dataset more easily, improving histogram alignment between source and target domains [74] or constructing more advanced adversarial training [30] by modifying both generator and discriminator networks. Also, NNs can be trained on high-resolution 11-bit datasets like Quickbird, Worldview 2/3, and GeoEye[1] to provide sharp and more detailed LULC maps. Furthermore, a multi-sensor model can be considered by training a network on multiple datasets from different sensors at once, making a more generic and versatile model, which can be used to develop LULC maps from the different sensors without prior training.

[1] https://www.eo4idi.eu/eo4sd-knowledge-portal/3-remote-sensing-technology/32-platforms/322-digitalglobe

Appendix A

Ground Truth and Generated LULC Maps

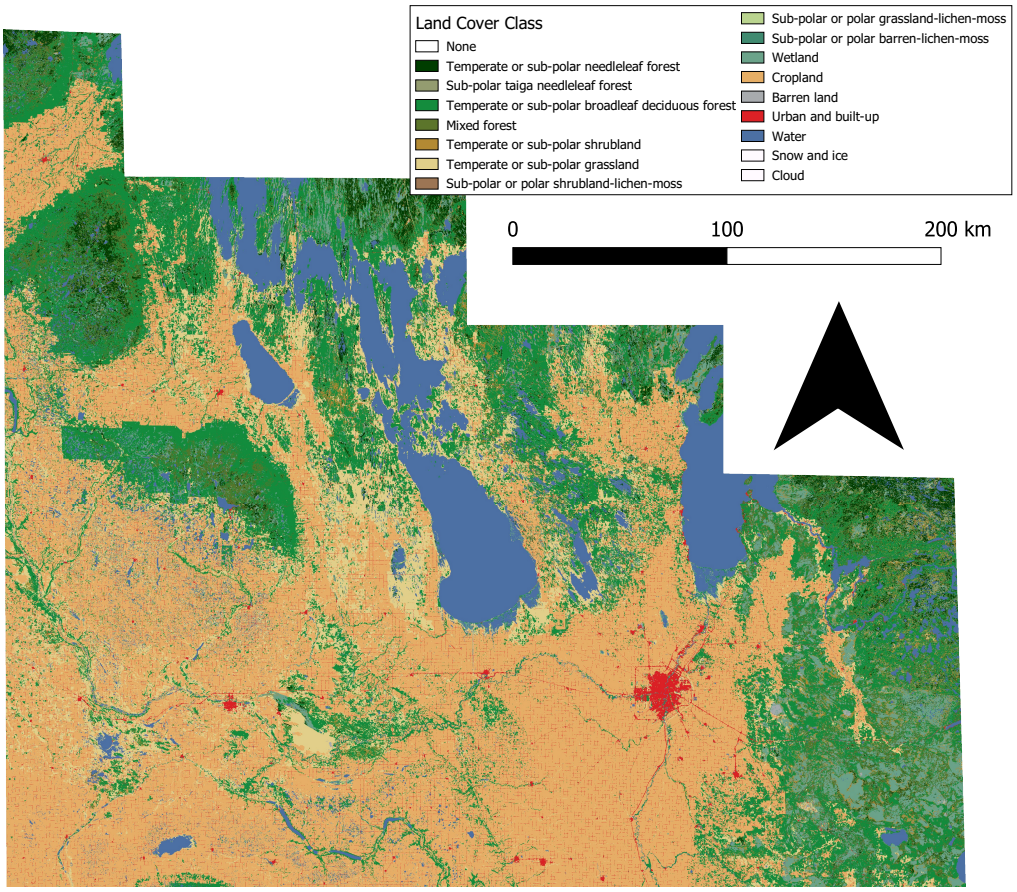


Figure A.1: Ground truth map of the southern extent of Manitoba for Landsat 5/7 dataset.

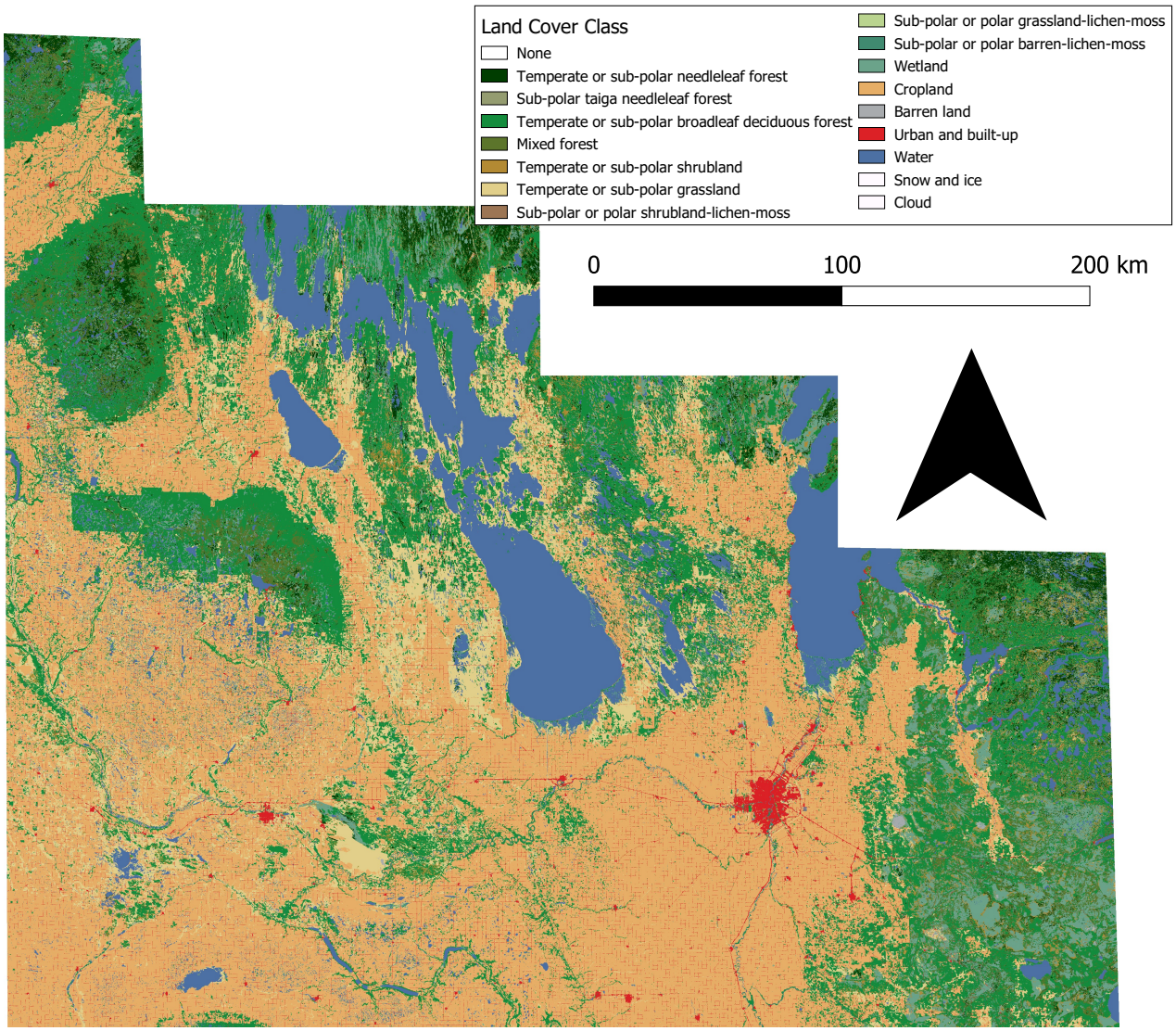


Figure A.2: Predicted map of the southern extent of Manitoba for Landsat 5/7 dataset using best model.

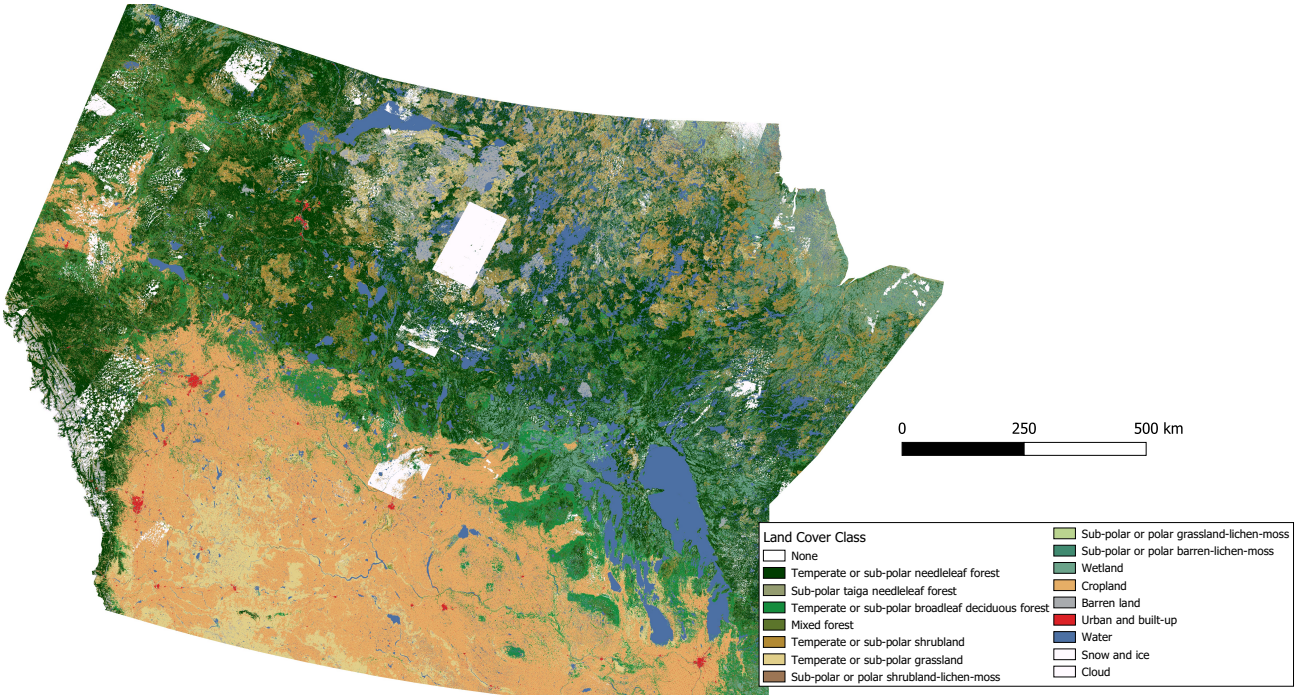


Figure A.3: Ground truth map of the Lake Winnipeg watershed for Landsat 5/7 dataset.

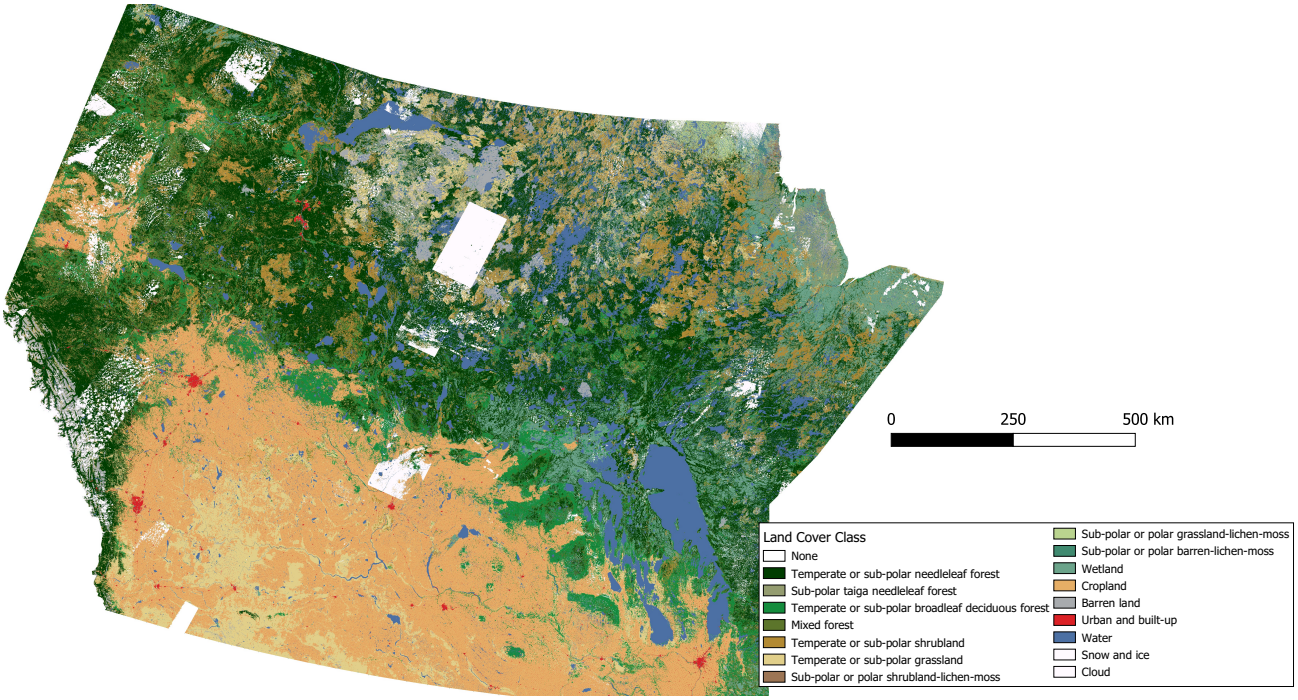


Figure A.4: Predicted map of the Lake Winnipeg watershed for Landsat 5/7 dataset using best model.

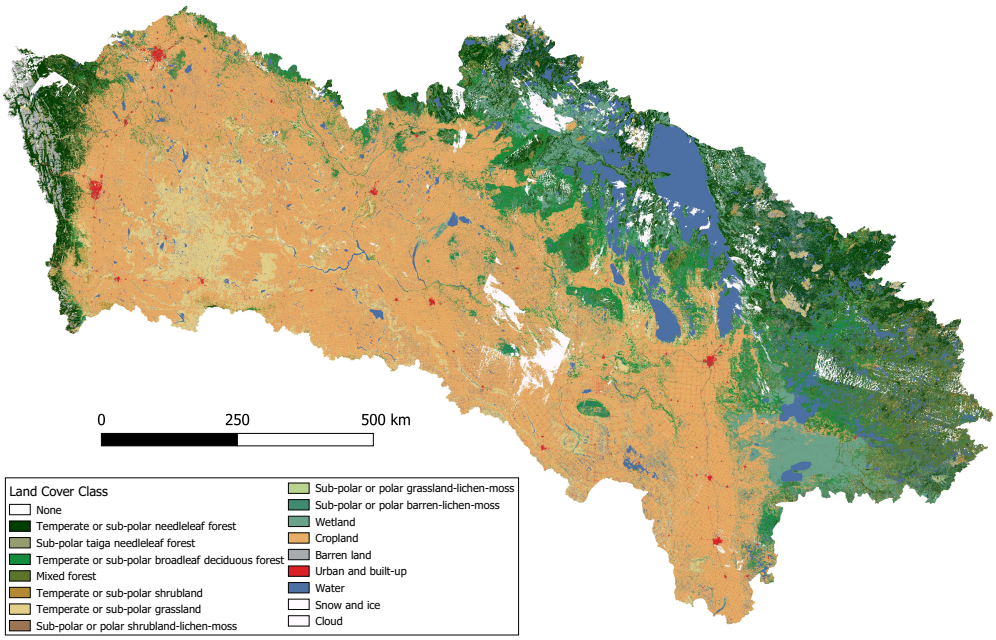


Figure A.5: Ground truth map of the Lake Winnipeg watershed for Landsat 8 dataset.

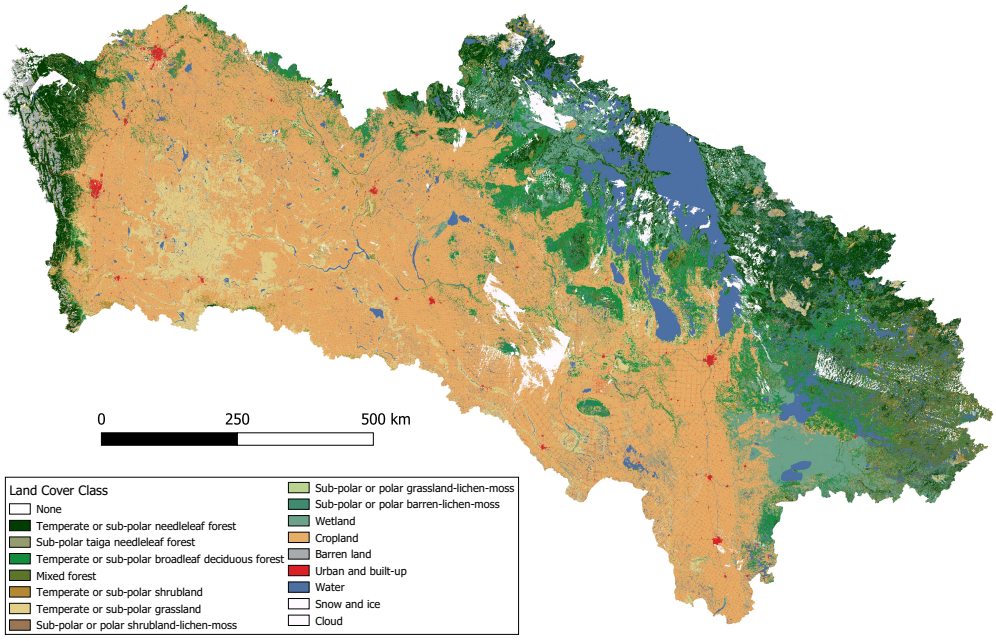


Figure A.6: Predicted map of the Lake Winnipeg watershed for Landsat 8 dataset using best model.

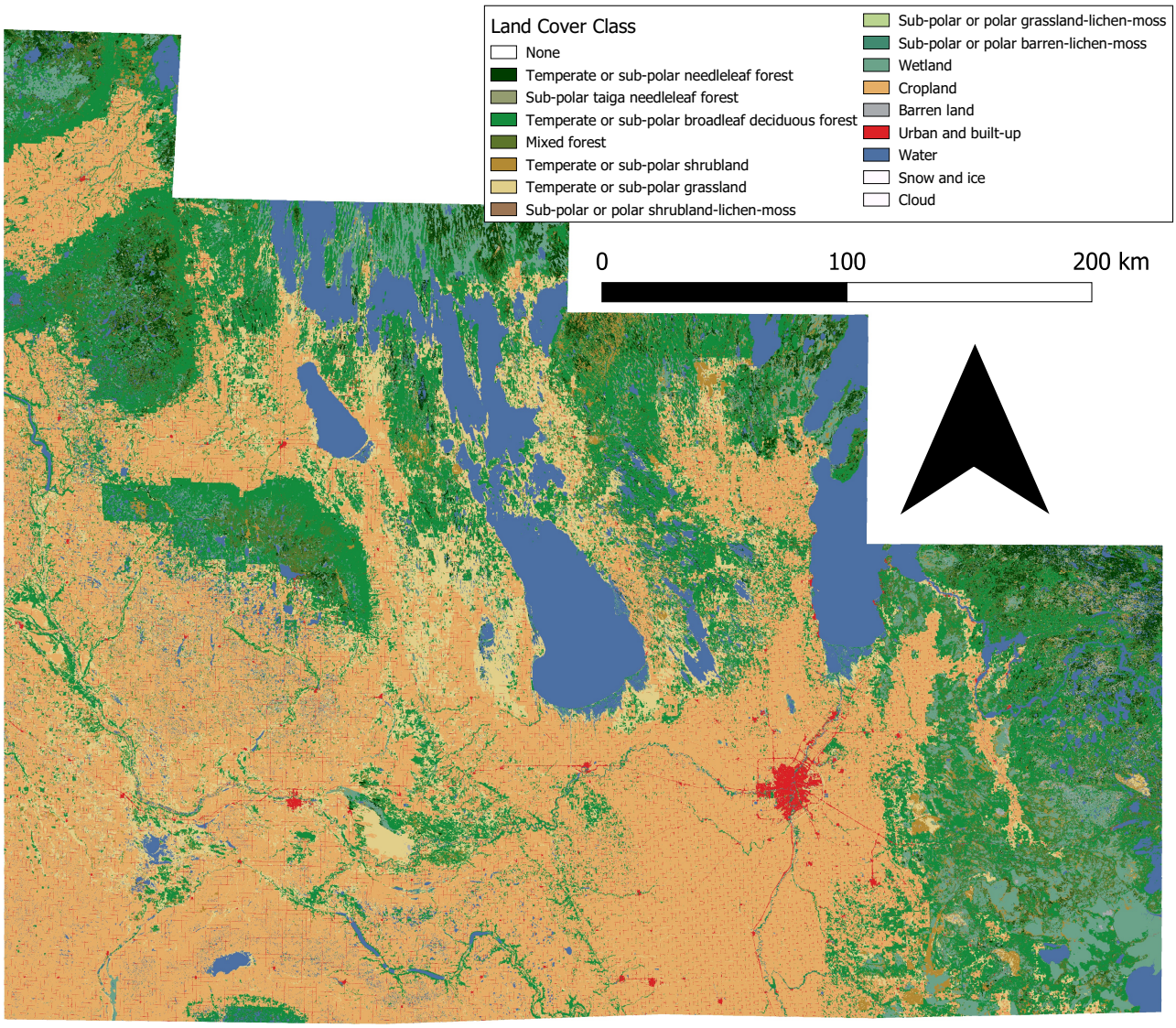


Figure A.7: Ground truth map of the southern extent of Manitoba for Sentinel-2 dataset.

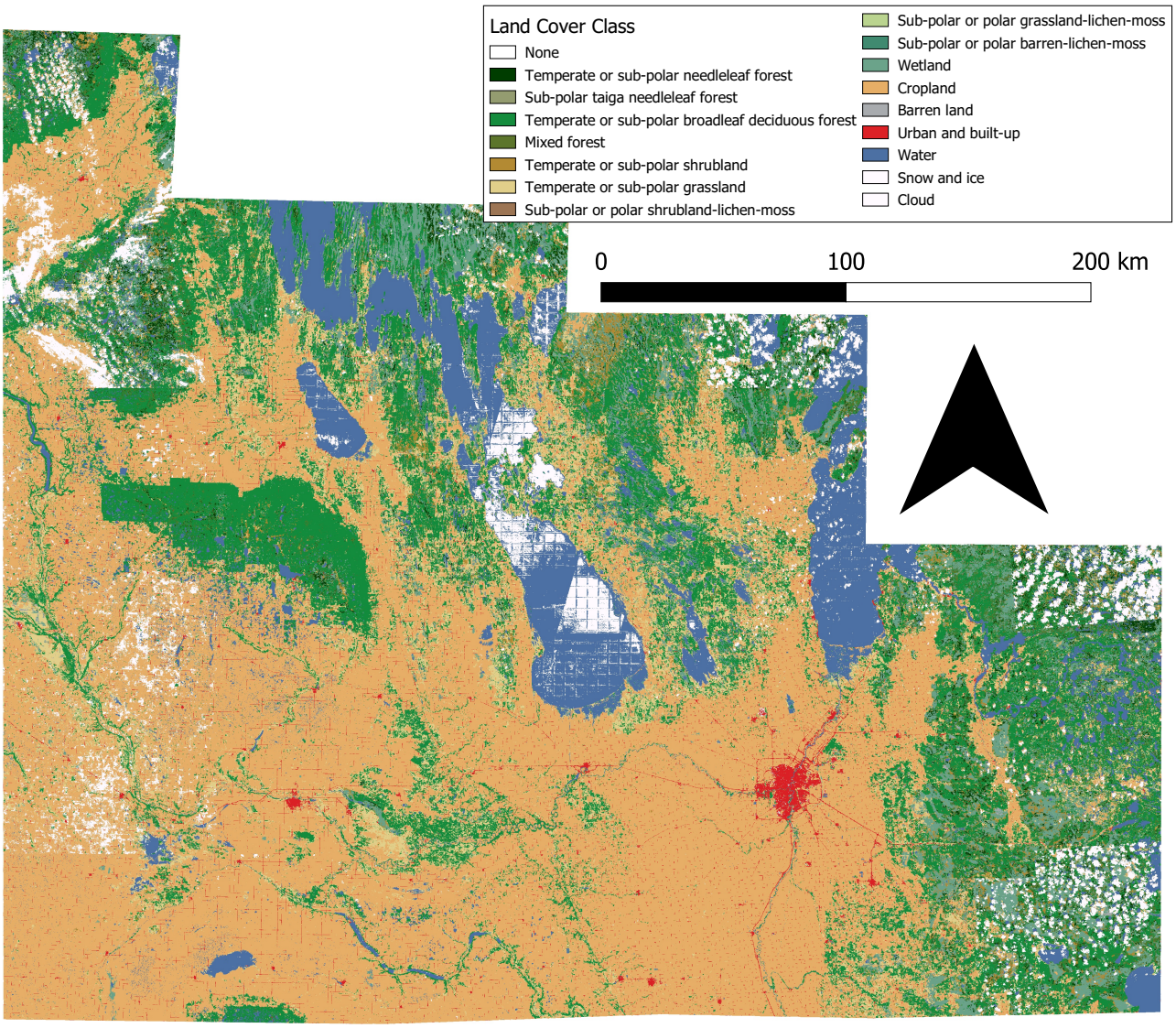


Figure A.8: Predicted map of the southern extent of Manitoba for Sentinel-2 dataset using best model trained on UDA architecture.

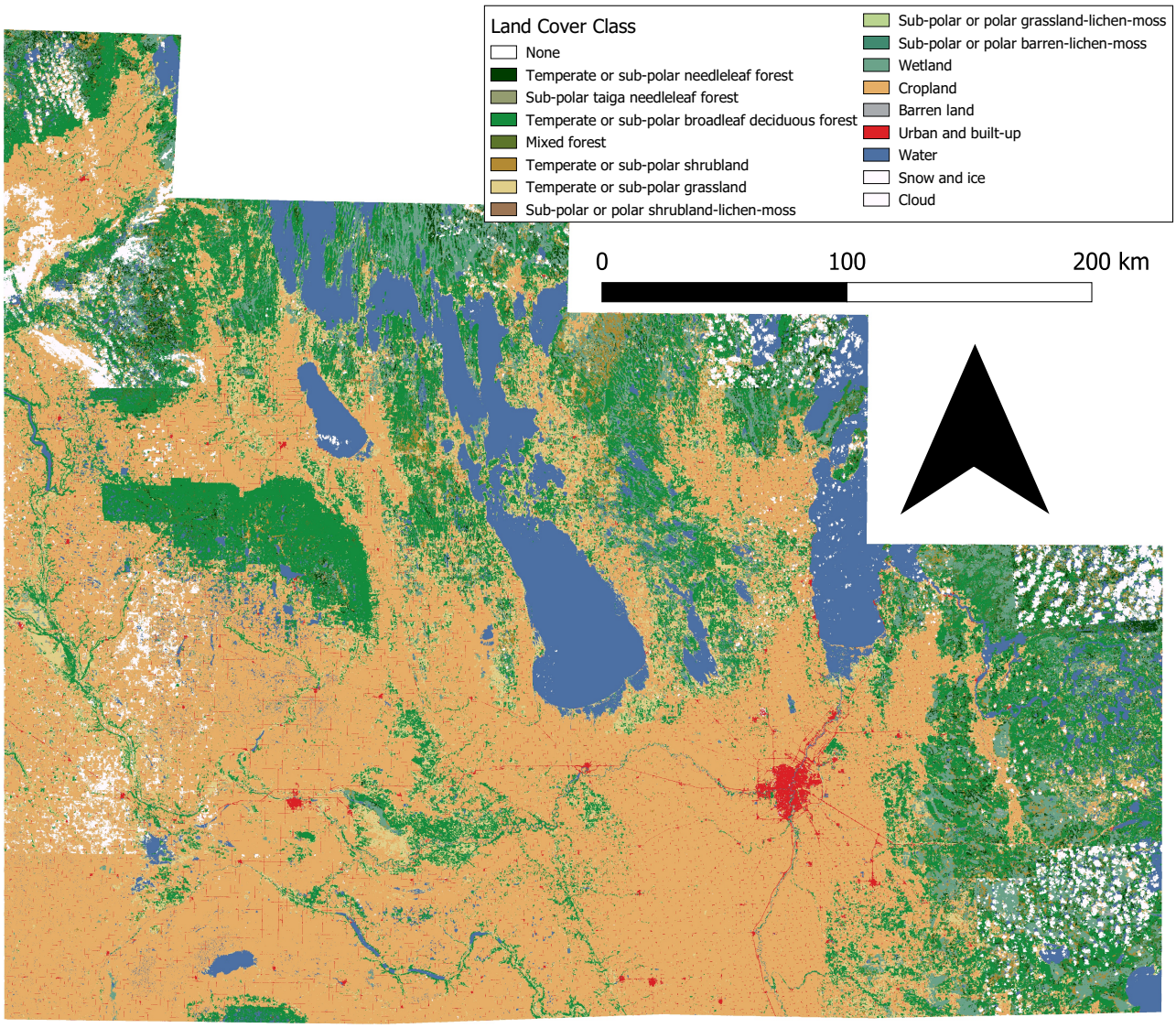


Figure A.9: Predicted map of the southern extent of Manitoba for Sentinel-2 dataset using best model trained on corrected dataset.

Bibliography

[1] P. Treitz and J. Rogan, "Remote sensing for mapping and monitoring land-cover and land-use change," *Progress in Planning*, vol. 61, no. 4, pp. 269–279, 2004.

[2] D. Lu and Q. Weng, "A survey of image classification methods and techniques for improving classification performance," *International Journal of Remote Sensing*, vol. 28, no. 5, pp. 823–870, 2007.

[3] L. Ma, Y. Liu, X. Zhang, Y. Ye, G. Yin, and B. A. Johnson, "Deep learning in remote sensing applications: A meta-analysis and review," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 152, pp. 166–177, 2019.

[4] A. Mayr, G. Klambauer, T. Unterthiner, and S. Hochreiter, "Deeptox: toxicity prediction using deep learning," *Frontiers in Environmental Science*, vol. 3, p. 80, 2016.

[5] F. Milletari, S.-A. Ahmadi, C. Kroll, A. Plate, V. Rozanski, J. Maiostre, J. Levin, O. Dietrich, B. Ertl-Wagner, K. Bötzel *et al.*, "Hough-cnn: deep learning for segmentation of deep brain regions in mri and ultrasound," *Computer Vision and Image Understanding*, vol. 164, pp. 92–102, 2017.

[6] P. Pérez de San Roman, J. Benois-Pineau, J.-P. Domenger, F. Paclet, D. Cataert, and A. de Rugy, "Saliency driven object recognition in egocentric videos with deep cnn: toward application in assistance to neuroprostheses," *Computer Vision and Image Understanding*, vol. 164, pp. 82–91, 2017, deep Learning for Computer Vision.

[7] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi, "A survey of the recent architectures of deep convolutional neural networks," *Artificial Intelligence Review*, vol. 53, no. 8, pp. 5455–5516, 2020.

[8] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 3431–3440.

[9] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 12, pp. 2481–2495, 2017.

[10] F. P. Luus, B. P. Salmon, F. Van den Bergh, and B. T. J. Maharaj, "Multiview deep learning for land-use classification," *IEEE Geoscience and Remote Sensing Letters*, vol. 12, no. 12, pp. 2448–2452, 2015.

[11] J. Wang, C. Luo, H. Huang, H. Zhao, and S. Wang, "Transferring pre-trained deep cnns for remote scene classification with general features learned from linear pca network," *Remote Sensing*, vol. 9, no. 3, p. 225, 2017.

[12] C. J. Henry, C. D. Storie, M. Palaniappan, V. Alhassan, M. Swamy, D. Aleshinloye, A. Curtis, and D. Kim, "Automated lulc map production using deep neural networks," *International Journal of Remote Sensing*, vol. 40, no. 11, pp. 4416–4440, 2019.

[13] C. D. Storie and C. J. Henry, "Deep learning neural networks for land use land cover mapping," in *IGARSS 2018-2018 IEEE International Geoscience and Remote Sensing Symposium*. IEEE, 2018, pp. 3445–3448.

[14] V. Alhassan, C. Henry, S. Ramanna, and C. Storie, "A deep learning framework for land-use/land-cover mapping and analysis using multispectral satellite imagery," *Neural Computing and Applications*, pp. 1–16, 2019.

[15] V. Alhassan, "Automated land use and land cover map production: A deep learning framework," Master's thesis, University of Winnipeg, Winnipeg, Canada, 2018. [Online]. Available: https://winnspace.uwinnipeg.ca/handle/10680/1579

[16] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *Computer Vision – ECCV 2016*, vol. 9908. Cham: Springer International Publishing, 2016, pp. 630–645.

[17] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 1251–1258.

[18] B. Wang, J. Glossner, D. Iancu, and G. N. Gaydadjiev, "Feedbackward decoding for semantic segmentation," *arXiv preprint arXiv:1908.08584*, 2019.

[19] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*. Cham: Springer International Publishing, 2015, pp. 234–241.

[20] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-decoder with atrous separable convolution for semantic image segmentation," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 801–818.

[21] T. Karras, T. Aila, S. Laine, and J. Lehtinen, "Progressive growing of gans for improved quality, stability, and variation," *arXiv preprint arXiv:1710.10196*, 2017.

[22] W.-G. Chang, T. You, S. Seo, S. Kwak, and B. Han, "Domain-specific batch normalization for unsupervised domain adaptation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 7354–7362.

[23] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," *arXiv preprint arXiv:1511.07122*, 2015.

[24] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *arXiv preprint arXiv:1406.2661*, 2014.

[25] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[26] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1–9.

[27] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016, pp. 770–778.

[28] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking atrous convolution for semantic image segmentation," *arXiv preprint arXiv:1706.05587*, 2017.

[29] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 4, pp. 834–848, 2018.

[30] E. Tzeng, J. Hoffman, K. Saenko, and T. Darrell, "Adversarial discriminative domain adaptation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 7167–7176.

[31] W. Hong, Z. Wang, M. Yang, and J. Yuan, "Conditional generative adversarial network for structured domain adaptation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 1335–1344.

[32] J. McCarthy, M. L. Minsky, N. Rochester, and C. E. Shannon, "A proposal for the dartmouth summer research project on artificial intelligence, august 31, 1955," *AI Magazine*, vol. 27, no. 4, pp. 12–12, 2006.

[33] F. Rosenblatt, *The perceptron, a perceiving and recognizing automaton Project Para.* Cornell Aeronautical Laboratory, 1957.

[34] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[35] F. Li and A. Karpathy, "CS231n: Convolutional neural networks for visual recognition," Course Notes, Stanford University, 2015.

[36] C. Sammut and G. I. Webb, Eds., *Mean Squared Error.* Boston, MA: Springer US, 2010, pp. 653–653. [Online]. Available: https://doi.org/10.1007/978-0-387-30164-8_528

[37] Y. N. Wu, *Cross Entropy.* Boston, MA: Springer US, 2014, pp. 154–154. [Online]. Available: https://doi.org/10.1007/978-0-387-31439-6_743

[38] J. L. McClelland, D. E. Rumelhart, P. R. Group *et al.*, *Parallel distributed processing.* MIT Press Cambridge, MA, 1986, vol. 2.

[39] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, Y. W. Teh and M. Titterington, Eds., vol. 9, JMLR Workshop and Conference Proceedings. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 249–256.

[40] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[41] M. D. Binder, N. Hirokawa, and U. Windhorst, Eds., *Gradient Descent*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 1765–1766. [Online]. Available: https://doi.org/10.1007/978-3-540-29678-2_2075

[42] L. Bottou, "Stochastic gradient descent tricks," in *Neural Networks: Tricks of the Trade*. Springer, 2012, pp. 421–436.

[43] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," in *Neural Networks: Tricks of the Trade*. Springer, 2012, pp. 437–478.

[44] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural Networks for Machine Learning*, vol. 4, no. 2, pp. 26–31, 2012.

[45] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. 7, 2011.

[46] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[47] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," *arXiv preprint arXiv:1603.07285*, 2016.

[48] K. Bai, "A comprehensive introduction to different types of convolutions in deep learning," Feb 2019. [Online]. Available: https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215

[49] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," *arXiv preprint arXiv:1511.07122*, 2015.

[50] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 3431–3440.

[51] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *science*, vol. 313, no. 5786, pp. 504–507, 2006.

[52] M. Lin, Q. Chen, and S. Yan, "Network in network," *arXiv preprint arXiv:1312.4400*, 2013.

[53] R. Olivier and C. Hanqiang, "Nearest neighbor value interpolation," *International Journal of Advanced Computer Science and Applications*, vol. 3, no. 4, 2012. [Online]. Available: http://dx.doi.org/10.14569/IJACSA.2012.030405

[54] E. J. Kirkland, *Bilinear Interpolation*. Boston, MA: Springer US, 2010, pp. 261–263. [Online]. Available: https://doi.org/10.1007/978-1-4419-6533-2_12

[55] G. Birkhoff and H. L. Garabedian, "Smooth surface interpolation," *Journal of Mathematics and Physics*, vol. 39, no. 1-4, pp. 258–268, 1960. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/sapm1960391258

[56] H. Zhang, L. Zhang, and Y. Jiang, "Overfitting and underfitting analysis for deep learning based end-to-end communication systems," in *2019 11th International Conference on Wireless Communications and Signal Processing (WCSP)*. IEEE, 2019, pp. 1–6.

[57] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, F. Bach and D. Blei, Eds., vol. 37. Lille, France: PMLR, 07–09 Jul 2015, pp. 448–456.

[58] M. M. YAPICI, A. TEKEREK, and N. TOPALOĞLU, "Literature review of deep learning research areas," *Gazi Mühendislik Bilimleri Dergisi (GMBD)*, vol. 5, no. 3, pp. 188–215, 2019.

[59] Đ. T. Grozdić, S. T. Jovičić, and M. Subotić, "Whispered speech recognition using deep denoising autoencoder," *Engineering Applications of Artificial Intelligence*, vol. 59, pp. 15–22, 2017.

[60] M. Al-Ayyoub, A. Nuseir, K. Alsmearat, Y. Jararweh, and B. Gupta, "Deep learning for arabic nlp: A survey," *Journal of Computational Science*, vol. 26, pp. 522–531, 2018.

[61] J. Kim, O. Sangjun, Y. Kim, and M. Lee, "Convolutional neural network with biologically inspired retinal structure," *Procedia Computer Science*, vol. 88, pp. 145–154, 2016.

[62] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[63] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in Neural Information Processing Systems*, vol. 25, pp. 1097–1105, 2012.

[64] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016, pp. 2818–2826.

[65] L.-C. Chen, J. T. Barron, G. Papandreou, K. Murphy, and A. L. Yuille, "Semantic image segmentation with task-specific edge detection using cnns and a discriminatively trained domain transform," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 4545–4554.

[66] F. Milletari, N. Navab, and S.-A. Ahmadi, "V-net: Fully convolutional neural networks for volumetric medical image segmentation," in *2016 Fourth International Conference on 3D Vision (3DV)*. IEEE, 2016, pp. 565–571.

[67] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 652–660.

[68] J. Huang and S. You, "Point cloud labeling using 3d convolutional neural network," in *2016 23rd International Conference on Pattern Recognition (ICPR)*. IEEE, 2016, pp. 2670–2675.

[69] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Semantic image segmentation with deep convolutional nets and fully connected crfs," *arXiv preprint arXiv:1412.7062*, 2014.

[70] P. Luc, C. Couprie, S. Chintala, and J. Verbeek, "Semantic segmentation using adversarial networks," *arXiv preprint arXiv:1611.08408*, 2016.

[71] E. Collier, K. Duffy, S. Ganguly, G. Madanguit, S. Kalia, G. Shreekant, R. Nemani, A. Michaelis, S. Li, A. Ganguly, and S. Mukhopadhyay, "Progressively growing generative adversarial networks for high resolution semantic segmentation of satellite images," in *2018 IEEE International Conference on Data Mining Workshops (ICDMW)*. IEEE, 2018, pp. 763–769.

[72] Y. Sun, E. Tzeng, T. Darrell, and A. A. Efros, "Unsupervised domain adaptation through self-supervision," *arXiv preprint arXiv:1909.11825*, 2019.

[73] Y. Li, L. Yuan, and N. Vasconcelos, "Bidirectional learning for domain adaptation of semantic segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 6936–6945.

[74] J. Wang, J. Chen, J. Lin, L. Sigal, and C. W. de Silva, "Discriminative feature alignment: Improving transferability of unsupervised domain adaptation by gaussian-guided latent alignment," *Pattern Recognition*, vol. 116, p. 107943, 2021.

[75] A. Sharma, X. Liu, X. Yang, and D. Shi, "A patch-based convolutional neural network for remote sensing image classification," *Neural Networks*, vol. 95, pp. 19–28, 2017.

[76] P. Li, P. Ren, X. Zhang, Q. Wang, X. Zhu, and L. Wang, "Region-wise deep feature representation for remote sensing images," *Remote Sensing*, vol. 10, no. 6, p. 871, 2018.

[77] J. Sherrah, "Fully convolutional networks for dense semantic labelling of high-resolution aerial imagery," *arXiv preprint arXiv:1606.02585*, 2016.

[78] G. Chen, X. Zhang, Q. Wang, F. Dai, Y. Gong, and K. Zhu, "Symmetrical dense-shortcut deep fully convolutional networks for semantic segmentation of very-high-resolution remote sensing images," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 11, no. 5, pp. 1633–1644, 2018.

[79] R. Kemker, C. Salvaggio, and C. Kanan, "Algorithms for semantic segmentation of multispectral remote sensing imagery using deep learning," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 145, pp. 60–77, 2018.

[80] V. Khryashchev, R. Larionov, A. Ostrovskaya, and A. Semenov, "Modification of u-net neural network in the task of multichannel satellite images segmentation," in *2019 IEEE East-West Design & Test Symposium (EWDTS)*. IEEE, 2019, pp. 1–4.

[81] M. Papadomanolaki, K. Karantzalos, and M. Vakalopoulou, "A multi-task deep learning framework coupling semantic segmentation and image reconstruction for very high resolution imagery," in *IGARSS 2019-2019 IEEE International Geoscience and Remote Sensing Symposium*. IEEE, 2019, pp. 1069–1072.

[82] B. Benjdira, K. Ouni, M. M. Al Rahhal, A. Albakr, A. Al-Habib, and E. Mahrous, "Spinal cord segmentation in ultrasound medical imagery," *Applied Sciences*, vol. 10, no. 4, p. 1370, 2020.

[83] D. DiBiase and A. John, "The nature of geographic information," *An Open Geospatial Textbook. URL: https://www. e-education. psu. edu/natureofgeoinfo/c6 _ p12. html (hämtad 2020-04-09)*, 2008.

[84] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, no. 1, pp. 62–66, 1979.

[85] (1999) Landsat collection 1 level-1 quality assessment band. [Online]. Available: https://www.usgs.gov/media/images/landsat-8-quality-assessment-band-attributes-and-possible-values

[86] D. Ho, E. Liang, X. Chen, I. Stoica, and P. Abbeel, "Population based augmentation: Efficient learning of augmentation policy schedules," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 09–15 Jun 2019, pp. 2731–2741.

[87] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of Big Data*, vol. 6, no. 1, pp. 1–48, 2019.

[88] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[89] F. W. Lancaster, *Evaluation of the MEDLARS demand search service.* US Department of Health, Education, and Welfare, Public Health Service, 1968.

[90] C. J. Van Rijsbergen, "A new theoretical framework for information retrieval," in *Acm Sigir Forum*, vol. 21, no. 1-2. ACM New York, NY, USA, 1986, pp. 23–29.

[91] J. T. Schaefer, "The critical success index as an indicator of warning skill," *Weather and Forecasting*, vol. 5, no. 4, pp. 570–575, 1990.

[92] P. J. Roebber, S. L. Bruening, D. M. Schultz, and J. V. Cortinas Jr, "Improving snowfall forecasting by diagnosing snow density," *Weather and Forecasting*, vol. 18, no. 2, pp. 264–287, 2003.