Unsupervised Domain Adaptation using Satellite Images for Significantly Different Infrastructure Objects

by

Mikhail Sokolov

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

Unsupervised Domain Adaptation using Satellite Images for Significantly Different Infrastructure Objects

by

Mikhail Sokolov

Supervisory Committee

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

Dr. J. Storie, Member
(Department of Geography)

Dr. Y. Bouroubi, External Member
(Département de Géomatique Appliquée, Université de Sherbrooke)

Abstract

Deep learning has become one of the most efficient computer vision tools in recent years. The success and variety of deep learning semantic segmentation models inspired scientists in the remote sensing domain to apply them to satellite imagery. Here, these models can produce reliable land use land cover maps in a short amount of time. However, porting these models to new sensors or domains is still limited by the amount of labelled data for training the network. The image labelling process is time-consuming and expensive because it is often manual (or semi-automated) work and requires assigning a label to each pixel in a satellite image. One solution is to apply the semantic segmentation model trained on a domain with known labels to a domain where labels are missing. For this to work, the discrepancy between domains must be narrowed to produce acceptable results. However, in practice, domain discrepancy can be significant. Developing domain adaptation models to bridge this discrepancy is the problem considered in this thesis, and it is important because semantically similar objects can look different from one geographical area to another. Therefore, several state-of-the-art domain adaptations were considered and validated using GeoEye-1 and WorldView-2 satellite imagery. The GeoEye-1 images represented a Canadian land cover, and WorldView-2 represented the African continent; thus, the domain discrepancy was significant. The CyCADA model with adapted noisy labeller showed the highest performance among all the considered models and achieved 32.6% of mean intersection over union, which is 7.5% higher compared to the model without adaptation. The contributions of this thesis are an attempt at domain adaptation across domains with the significant structural discrepancy, structural improvements to the CyCADA and DAugNet models, and quantitative and qualitative analysis of model performance on domain adaptation with significant structural discrepancy.

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

Acknowledgment

I express my deepest gratitude towards my Supervisor Dr. Christopher Henry for the constant help and encouragement from the start of the thesis work. I have been fortunate to have a Supervisor who gave me the freedom to explore on my own and at the same time helped me plan the future work-plan with timely reviews and suggestions wherever required. Thank you for giving me enough extension of time whenever required and having faith in me.

It gives me great pleasure to acknowledge the support of Drs. Joni Storie and Christopher Storie (University of Winnipeg, Geography Dept.), for the mentoring and pleasant company during the entire master's program. Also, special thanks to Rostyslav-Mykola Tsenov, Jamie Cameron, and Victor Alhassan for the helpful discussions and cooperation.

Finally, I would like to thank Manitoba Hydro, Arctic Gateway Group, Mitacs, NSERC, and the Faculty of Graduate Studies for funding this work.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Land use and land cover (LULC) maps are products resulting from satellite imagery that relate each pixel in the satellite image to a specific information class of objects, *e.g.*, vegetation, hydro, road, *etc.* Land use refers to how the land is utilized, such as agriculture, industrial and residential development, *etc.* Land use applications generally serve to monitor the changes in land usage over time. Land cover, in turn, refers to which natural (rivers, forests, snow, *etc.*) or human-made objects (buildings, cars, roads, *etc.*) cover the ground [1], the land cover features that can be detected using reflected energy recorded per pixel while the land use is inferred based on land cover elements.

Governments and commercial organizations involved in land management widely use LULC maps because they can provide up-to-date and accurate information if generated on a regular basis. For example, LULC maps are used for emergency response to help efficiently deploy restoration forces in the territory of massive flooding or landslides. Moreover, demand for up-to-date LULC maps is increasing over the years because the performance of new satellites is also increasing, providing more information and the variation of its use. For instance, the Sentinel-1 synthetic aperture radar (SAR) [2] enabled all-weather land monitoring because its sensors can penetrate the clouds. Furthermore, new constellations that provide data at higher temporal frequencies (*i.e.* shorter revisit periods) and broader area coverage are coming online.

The process of generating LULC maps involves many trained specialists. Before deep learning was introduced, the process of LULC map production was semi-automated. It required a human in the loop, but also made use of existing tools and algorithms [3]. To label one satellite image, a knowledgeable person must visually assess each image pixel and assign a corresponding label. This labour is time-consuming and expensive, and tied with the need for increased temporal frequency of LULC maps, which drove the need for a fully automated solution. This need and the rise of deep learning algorithms [4] led to the development of segmentation algorithms for LULC map generation.

Semantic segmentation is a process of assigning each pixel a semantic label, *e.g.*, cat, dog, background, *etc.* Recent achievements in image classification problems using convolutional neural networks (CNNs) led to significant advances in per-pixel segmentation tasks as well [5, 6, 7]. This, in turn, was used in a wide spectrum of computer vision applications [8, 9]. The success of CNN-based segmentation algorithms led to their use in developing LULC maps, with very good results [10, 11]. However, the process of annotating a large number of images needed for training CNN-based semantic segmentation models could be a crux. For example, the release of each new satellite (and corresponding sensors) usually requires the creation of a new LULC dataset due to spatial and spectral differences in the new sensors. Also, the ability to adapt models trained with labelled data from one domain (called the *source* domain) to another domain (called the *target* domain) would be very useful due to the expense of labelling new datasets. For example, different remote sensing (RS) scene image datasets may be taken from different types of sensors, in different weather conditions, in different geographical areas, and have different resolutions and scales. Consequently, the domain distribution discrepancy may be significant from one dataset to another, which makes models trained on a source data set useless for new target domains.

To tackle this issue, researchers have started investigating and developing domain adaptation (DA) techniques that are used to close the gap between source and target domains. Most common approaches aim to align features across two domains so that a semantic segmentation model can generalize across them [12]. However, feature adaptation in segmentation tasks is a very complicated (compared to classification task) process because the model has to encode a diversity of different visual characteristics such as appearance, shapes and context [13]. Another group of adaptation methods deal with a style transferring task and show outstanding performance when applied to RS datasets [14, 15, 16]; both semantic segmentation and style transferring methods are incorporated in this research.

1.1 Problem Definition

Recently, there has been great success in domain adaptation results [14, 15, 16]. However, they are limited by one critical condition - even though the source and target datasets are generated by different sensors and have a noticeable discrepancy in the representation of the domain, they still are pretty similar with respect to the structural characteristics of the physical objects they represent. In other words, two different sensors acquire images over the same geographical area. The question of domain adaptation when samples of both datasets represent semantically similar objects but with different structures (such as paved roads vs. dirt roads) is poorly studied, and this thesis is focused on exploring this problem.

1.2 Proposed Approach

In order to conduct sophisticated analysis of the topic, the following datasets were chosen as source and target domains. The GeoEye-1 (GE1) [17] represents the source domain dataset, which is a 2-meter spatial resolution multi-band 8-bit set of images taken in spring, summer and fall months across Canada. WorldView-3 (WV3) [18] represents the target domain dataset, which is a 0.31-meter (pansharpened) 16-bit set of images taken in an unknown season over a desert area of the African continent. Corresponding labels matching both datasets are also provided: background, roads, buildings, vegetation, hydro. Samples from both datasets are shown in Fig. 1.1. The discrepancy table (Table 1.1) shows the degree of visual dissimilarity for semantically identical objects. To achieve the goal of the thesis, several state-of-the-art methods were chosen that all have unique traits to distinguish them from each other and make them proper candidates for this task.

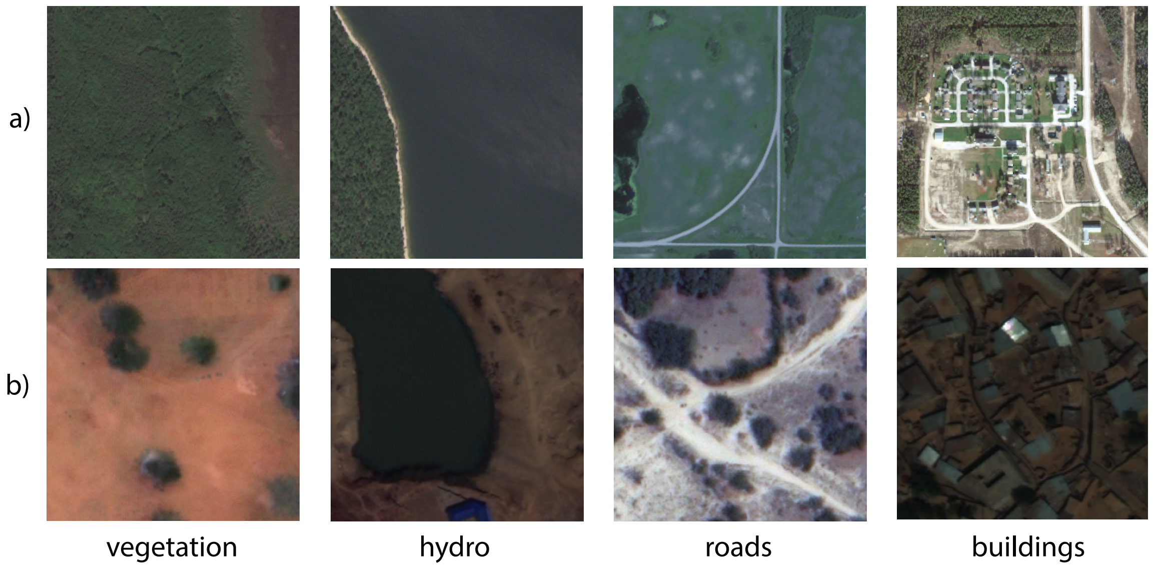


Figure 1.1: Samples from the source and target datasets where row a) represents GE-1 and row b) represents WV-3 imagery.

One of these methods (Method 1) was developed by Tsai *et al.* in [13], where a pixel-level prediction adaptation task was used rather than feature adaptation. The idea of this approach is explained as in

Feature	GeoEye-1	WorldView-3
Sensor characteristics		
Blue	450-510mm	450-510mm
Green	510-580mm	510-580mm
Red	655-690mm	630-690mm
Near IR	780-920mm	770-895mm
Spatial resolution	2m (downsampled from 1.84m)	0.31m (pansharpened)
Radiometric resolution	8bit	16bit
Semantic characteristics		
class: background	mostly agricultural fields with crops or black soil	mainly a mix of soil and sand with colour range from light gray to dark orange
class:vegetation	arrays of dense green forest	sparsely distributed individual trees of round shape
class:hydro	well-represented class of water-bodies having colour from dark green to blue	underrepresented class of waterbodies mostly dark brown colour
class: roads	narrow and straight asphalt	narrow or wide twisty dirt tracks
class: buildings	structured arrays of residential areas with relatively small buildings of square shape	unstructured arrays of constructions of arbitrary shapes randomly located and having mostly iron roofs

Table 1.1: Table of discrepancies between the datasets.

semantic segmentation the output space shares similarities between source and target domains even if the domain shift between them is significant. The similarities, in this case, are spatial layout and local context. To restate the paper's goal, the authors try to adapt the output (segmentation) space to address the pixel-level domain adaptation problem. The proposed method can be implemented using a CNN semantic segmentation model combined with a generative adversarial network [12]. The adversarial part is discarded at the testing stage, and the model can be used on the target domain without additional computations.

Another method (Method 2) was proposed by Zhu *et al.* in [19] and is known as CycleGAN. This method is widely used and found in many applications in different areas because of its proven performance in style transferring tasks [20, 21, 22, 23]. After the model is trained, a segmentation model is trained using target-style source images with corresponding labels.

The third method (Method 3) was proposed by Hoffman *et al.* in [24] and is known as cycle-consistent adversarial domain adaptation (CyCADA). Its main goal is to adapt image representations at feature and pixel levels while enforcing structural consistency locally and globally. Basically, this method is a combination of a feature-level adaptation method [13] and modified CycleGAN. After the style transferring part of the model is trained, a semantic segmentation model is trained, similar to Method 2.

The fourth method (Method 4) was proposed by Tasar *et al.* in [15] as a data augmenter network (DAugNet). It employs the adaptive instance normalization (AdaIN) layer [25] for style transferring and aims to stylize source images to look like target ones. Once the style-transferring part of the model is trained, the classifier is trained similarly to Method 2 and 3. As claimed by the authors of DAugNet, this model trains fast and demands less memory while training.

1.3 Contribution

The major contributions of this project are as follows:

- As far as I am aware, this thesis represents the first attempt at domain adaptation across domains with significant structural discrepancy.
- Improvements for the CyCADA and DAugNet models.
- Quantitative and qualitative analysis of model performance on domain adaptation with significant structural discrepancy.

1.4 Thesis Layout

The rest of this thesis is outlined as follows:

Chapter 2 includes the methodology related to the proposed solution. It generally explains fundamental definitions of artificial neural networks (ANN), convolutional neural networks (CNN) and adversarial learning.

Chapter 3 reviews publications and research work related to this thesis area.

Chapter 4 thoroughly describes the structure and training process of each model considered in this thesis. It also includes a description of proposed improvements for CyCADA and DAugNet.

Chapter 5 provides the implementation details such as custom data preprocessing, training hyperparameters, and system specifications.

Chapter 6 gives per-model results, compares them and discusses the pros and cons of each model.

Chapter 7 summarizes the conducted work and proposes a future plan of research in this direction.

Chapter 2

Methodology

This chapter explains the fundamental notions of neural networks (NN), starting from the concept of a formal neuron and continuing to advanced architectures and mathematical models. First, the structure of a biological neuron in human brains is introduced as the inspiration of an artificial neuron. Also, the definitions of the most common activation functions are provided. Second, the training process of neural networks through updating weights is discussed. Third, more sophisticated combinations of artificial neurons are discussed, particularly convolutional neural networks (CNN), which are widely used in image processing tasks. Specific layers of CNNs are also explored. Lastly, the relatively recent idea of adversarial learning and generative adversarial networks (GANs) is considered at the end of the chapter.

2.1 Neural Network

An artificial neural network (ANN) is an algorithm (or a group of algorithms) where the goal is to learn how to recognize underlying patterns in a given input dataset. The learning process imitates the functionality of the human brain. Depending on the task, the structure of an ANN can be different. However, the structure can also be the same if the inputs change because the ANN can adapt to them. Hence and further, an ANN is referred to as a neural network (NN).

2.1.1 Single-Layer Perceptron

A biological neuron is a sophisticated functional unit of the nervous system [26]. It induces electrical signals and transfers them through the human body's nervous system. Basically, biological neurons perform three operations: receive signals, process the signal, and decide whether to send a processed signal to other neurons or body parts. In 1958 Rosenblatt [27] proposed the idea of an artificial neuron that could mimic basic the functions of the biological neuron. He called it a *perceptron* and its structure is depicted in Fig. 2.1. Even though the perceptron is a significantly simplified version of a biological neuron, it has proven to solve classification problems, which is why it was widely adopted.

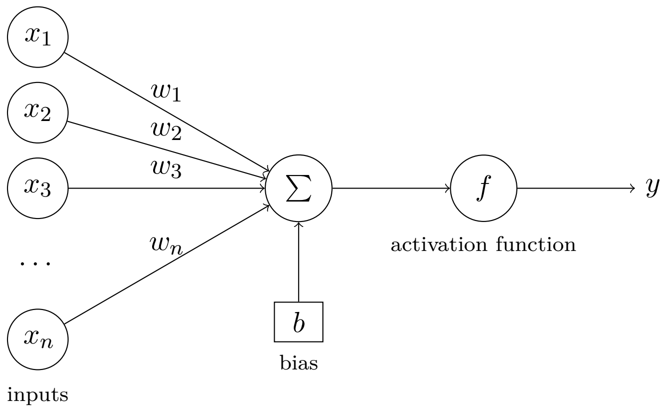


Figure 2.1: Figure depicting a single-layer perceptron (SLP).

We denote a single-layer perceptron (SLP) as a mathematical model consisting of a weighted adder and a non-linear element. In the Fig. 2.1, the vectors $\boldsymbol{x}_i$ and $\boldsymbol{w}_i$ are input signals and weight coefficients, respectively, and the output of an SLP is defined as

$$net = \sum_i w_i x_i + b, \tag{2.1}$$

where *net* is a weighted sum of input signals and b connotes a bias which can shift the weighted sum and is independent to the input values. It is important in cases when, for instance, the input vector $\boldsymbol{x}$ is a zero-vector (*i.e.* all its elements are zero), then the output will be always zero. The bias gives more flexibility to the SLP in this case. The result of *net* is then passed to a non-linear element f:

$$y = f(net), \tag{2.2}$$

where y is an output signal (vector), and f is an activation function, which is discussed in the next section.

2.1.2 Activation Functions

An SLP model without an activation function can do nothing but linearly scale and shift the inputs. In order to solve more sophisticated problems like classification, an extra operation, such as nonlinear activation, is needed. Depending on the task, different types of activation functions are used [28]. Those related to this project (from [28]) are discussed and summarized in this section.

Step Function

In classical perceptron definition the activation function is a step function:

$$y = \begin{cases} 0, & \text{for } net < 0 \\ 1, & \text{for } net \geq 0. \end{cases} \tag{2.3}$$

The step function is discontinuous and non-differentiable at the point of discontinuity. In a perceptron, it is used to imitate the thresholding operation of a biological neuron, *i.e.* it was introduced to either pass the signal further or terminate it.

Logistic Function

The logistic function, also known as a sigmoid or a Fermi function, is one of the most commonly used activation functions in computer science, and it is defined as:

$$y = \frac{1}{1 + e^{-net}}. \tag{2.4}$$

The graph of the function is depicted in Fig. 2.2. It is often used where the input signals are continuous throughout the domain of definition. The activation function itself is continuously differentiable, thus its first derivative is also continuously differentiable. This means calculating the partial derivative of this function is a straightforward task, which is necessary for gradient descent methods (discussed in Section 2.2.2).

The function is symmetric relative to the point (net=0, y=1/2), what makes values $y = 0$ and $y = 1$ tantamount. However, the range of the output values from 0 to 1 are not symmetrical, and this significantly slows the training process [29]. This function is often used when the task is to predict the probability of anything because the outputs of this function are always in the range from 0 to 1.

Hyperbolic Tangent

Another activation function is the hyperbolic tangent which also relates to the class of logistic functions, yet shifted and scaled with respect to the original logistic function. Its graph is depicted in Fig. 2.3 and is given as:

$$y = \frac{e^{net} - e^{-net}}{e^{net} + e^{-net}}. \tag{2.5}$$

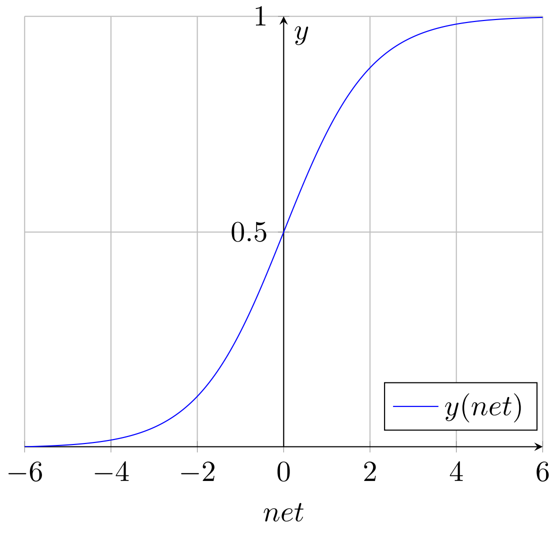


Figure 2.2: Plot depicting the output of a logistic function.

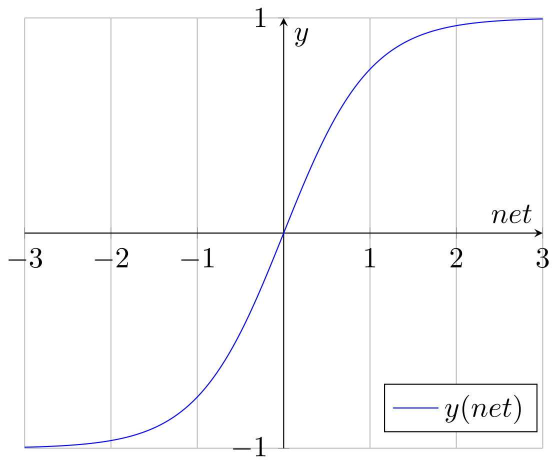


Figure 2.3: Plot depicting the output of a hyperbolic tangent function.

This function is applicable for networks where the input signals are continuous. It is symmetrical relative to the point (net=0, y=0), which is an advantage compared to the classic sigmoid function. The first derivative of the hyperbolic tangent function is continuously differentiable, thus gradient descent methods are applicable for the training process.

Rectified Linear Unit

The Rectified Linear Unit (ReLU) function is a linear activation function which returns 0 if the input value is negative and the value itself otherwise. The graph of the function is depicted in Fig. 2.4 and the formula is:

$$y = max(0, net). \tag{2.6}$$

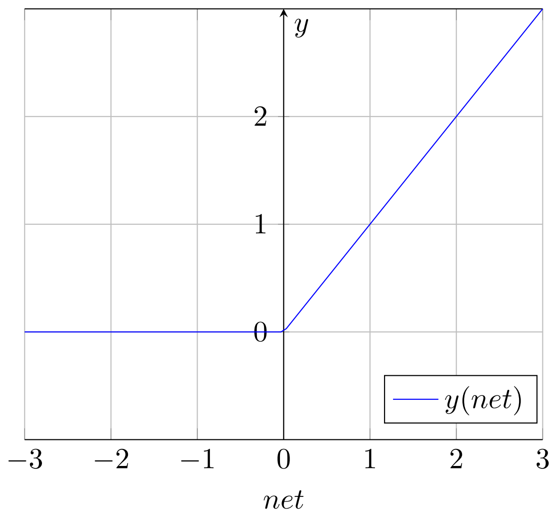


Figure 2.4: Plot depicting the output of a ReLU function.

The ReLU function provides better options for training NNs with a significant number of layers (deep networks) [30, 31]. It is more computationally efficient to calculate compared to the logistic regression functions because it is only a comparison operation. Also, its derivative is faster to compute, which speeds up the gradient descent algorithm discussed in Section 2.2.2.

Leaky Rectified Linear Unit

Even though the ReLU has many advantages and, by far, is the most commonly used activation function in computer vision problems, it has one significant drawback. When the input signal vector has negative values, they turn zero immediately, having no chance to influence the network output. To tackle this issue, the leaky ReLU function was developed. Its graph is depicted in Fig. 2.5 and the formula is:

$$y = \begin{cases} \alpha net, & \text{for } net < 0 \\ net, & \text{for } net \geq 0, \end{cases} \tag{2.7}$$

where α is a parameter also known as a slope. It is usually 0.01 but can vary depending on the task.

2.1.3 SLP Training

The main feature of the perceptron is its ability to be trained in order to classify input vectors. The training process is iterative and is aimed to adjust weights and biases until a certain goal is achieved. Usually, the goal is to minimize some type of function associated with a classification task. The original

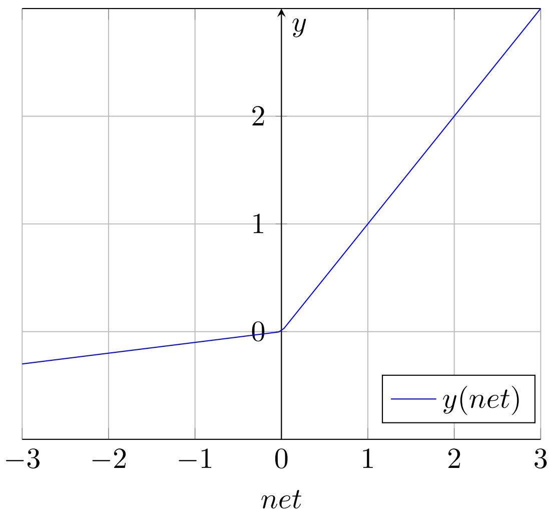


Figure 2.5: Plot depicting the output of a leaky ReLU function.

training algorithm for the perceptron was proposed by Rosenblatt in [27], and is shown in Alg. 2.1.1 [32].

Algorithm 2.1.1 PerceptronTraining(D, MaxIterations)

```
1: $w(1)_i \leftarrow Random(0.5, 0.5)$, for all $i$=1..D.length //initialize weights, D is a training set
2: $b \leftarrow Random(0.5, 0.5)$ //initialize bias
3: for p=1..MaxIterations do
4:     for all $(x, y_d) \in D$ do
5:         $y(p) \leftarrow activation[\sum_{i=0}^{D.length} w_i(p)x_i(p) + b]$ //calculate activation for current iteration
6:         $e(p) \leftarrow y(p) - y_d(p)$ //calculate error, where $y_d(p)$ is desired value
7:         $w_i(p+1) = w_i(p) + \alpha x_i(p)e(p)$ //update weights, $\alpha$ is a learning rate less than 1.
```

Only linearly separable functions can be implemented by an SLP [33]. To overcome this issue, combinations of several SLP into multi-layer structures, known as multi-layer perceptrons (MLP), are applied.

2.1.4 Multi-Layer Perceptron

Perceptrons can be combined together into a network in different ways. The possible basic architecture of multi-layer networks is a multi-layer perceptron (MLP) depicted in Fig. 2.6. This network consists of an arbitrary number of layers, where the neurons from each layer are connected with the neurons of previous and subsequent layers in an “each-to-each" manner, which is known as a fully connected network. The first layer in an MLP is called the input layer, the intermediate layers are called hidden layers, and the very last layer is called the result or output layer. The number of neurons in the layers varies depending on the classification task and network architectures. The MLPs with many (no strict definition) hidden layers are usually called deep neural networks, and those with a small number of hidden layers are called shallow neural networks.

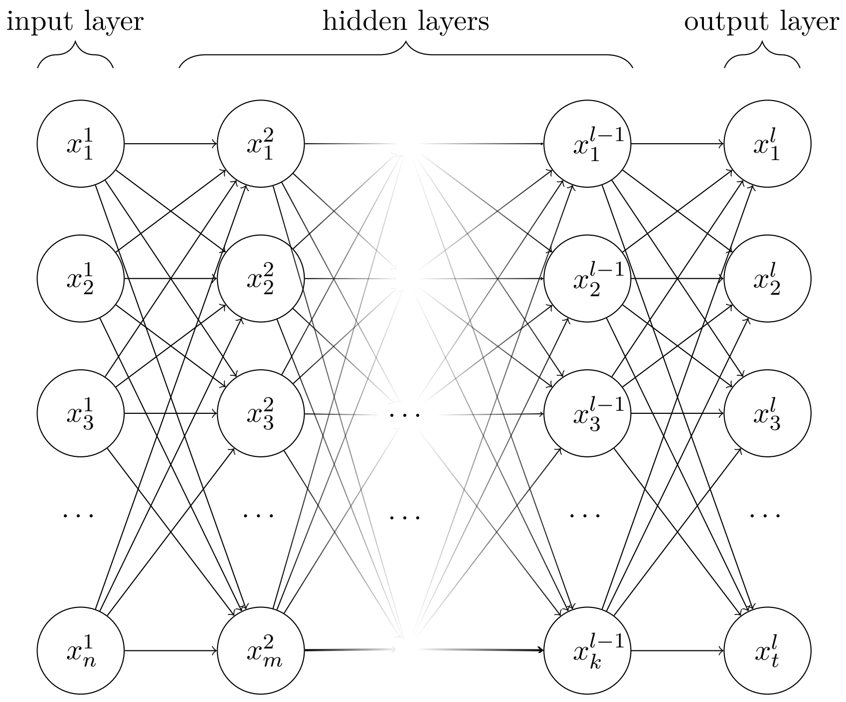


Figure 2.6: A multi-layer perceptron.

The behavior of a MLP can be expressed by the formula:

$$\begin{aligned} &h^{(1)} = \phi^{(1)}(W^{(1)}x + b^{(1)}), \\ &h^{(2)} = \phi^{(2)}(W^{(2)}h^{(1)} + b^{(2)}), \\ &... \\ &y = \phi^{(l)}(W^{(l)}h^{(l-1)} + b^{(l)}), \end{aligned} \tag{2.8}$$

where $\boldsymbol{x}$ is an input vector, $\boldsymbol{h^{(l)}}$ is an activation vector for the lth layer, $\boldsymbol{W^{(l)}}$ is a weight matrix of the lth layer, $\boldsymbol{b^{(l)}}$ is its bias vector, and y is an output signal (vector) of the MLP. As seen from 2.8, the approximation of an arbitrary multidimensional function is achieved by sequential calculation of linear combinations and nonlinear transformations [34, 35, 36].

2.2 Training

In general, the training process for NNs is similar to the perceptron training process described in Section 2.1.3. The goal is to minimize a cost (or loss) function through updating the weights of the model. This approach is systematic and includes the following major steps. First, a loss function is defined and applied to calculate the difference (or error) between the model's outputs and desired target vector. Second, a certain optimization algorithm is applied to minimize (or maximize) the loss function. Third, the weights of the model are updated through the backpropagation algorithm.

2.2.1 Loss Function

In NNs, the loss function is tightly coupled to the task that has to be solved. It describes a multi-dimensional space of all possible solutions of the task, and the ultimate goal is to find a global minimum (or maximum) of this loss function. Generally, this function has to be minimized but there are some problems that aim to maximize a cost function [12]. Thus, any cost function must be defined according to the specific task [37, 38]. Those related to this project (from [37, 38]) are discussed and summarized in this section.

Binary Cross-Entropy

This function is used in binary classification problems where the task is to classify whether the inputs belong to one class or another. Given an NN output vector, $\hat{y}$, and a target vector, y, the binary cross-entropy loss function can be expressed as:

$$L_{bce}(y, \hat{y}) = -y_i \cdot \log \hat{y}_i - (1 - y_i) \cdot \log(1 - \hat{y}_i). \tag{2.9}$$

Cross-Entropy

The cross-entropy loss is also known as a multi-nomial logistic loss is a generalized version of a binary cross-entropy loss. The generalization allows computing the error for multi-class classification tasks. Given an NN output vector, $\hat{y}$, a target vector, y, and the number of classes, n, the loss is defined as:

$$L_{ce}(y, \hat{y}) = -\sum_{i=1}^{n} y_i \cdot \log \hat{y}_i. \tag{2.10}$$

As can be seen from Eq. 2.10, the binary cross-entropy loss can be acquired by using the number of classes $n = 2$.

L1-norm

The L1-norm function is also known as least absolute deviations or least absolute error. It is used to minimize the sum of the absolute differences between the NN outputs $\hat{y}$ and the target values y:

$$L_1(y, \hat{y}) = \sum_{i=1}^{m} |y_i - \hat{y_i}|, \tag{2.11}$$

where m is the number of elements in y.

2.2.2 Optimization

Having the loss function, it is necessary to find its optimum value. The gradient descent method and its variations are commonly-used for deep learning and machine learning model training. After passing the inputs through the NN (forward propagation), the outputs are obtained, and the loss function is used to calculate the error for the given inputs. In order to minimize this error, the multi-variable derivative of the loss function is calculated with respect to all the network parameters that participated in the forward pass. Thus, the slope of the loss function is calculated, which shows the direction and rate of fastest increase, and, therefore, the opposite direction is used to minimize it. In other words, the gradient can be used to iteratively yield a vector that points in the best direction to minimize the loss function. Once the new vector is obtained, the corresponding gradient value is used to adjust the network weights in combination with a certain learning rate. The learning rate is a fractional value that controls how intensive the weights are updated (also known as magnitude). It is a well-known practice to decrease the learning rate over iterations in order to come closer to the global optimum of the loss function. The overall formula for the gradient descent method is expressed as follows:

$$\theta_{i+1} = \theta_i - \lambda \cdot \frac{\partial L(\theta_i)}{\partial \theta_i}, \tag{2.12}$$

where θ_i are parameters that participated in the forward pass at the *ith* iteration, L is a loss function, λ is a learning rate, and θ_{i+1} are new weights for the network which are supposed to encourage the network to predict outputs with less error. The algorithmic representation of the method is shown in Alg. 2.2.1: Several modifications of this method exist. Two modifications known as Adam and Stochastic Gradient Descent (SGD) [39] are used in this thesis.

2.2.3 Backpropagation

Backpropagation is a method of updating the weights in a NN that was first proposed in 1960 and popularized by Rumelhart *et al.* in [40]. The method is designed to recursively adjust weights in the

Algorithm 2.2.1 Gradient Descent(θ_{init}, MaxIterations)

```
1: $\theta_1 = \theta_{init}$ // the network is initialized with initial weights $\theta_{init}$
2: for i=1..MaxIterations do
3:     calculate $d_i = \frac{\partial L(\theta_i)}{\partial \theta_i}$ // calculate partial derivative
4:     calculate $\theta_{i+1} = \theta_i - \lambda \cdot d_i$ // update parameters
5: return $\theta_{i+1}$
```

NN in order to minimize the assigned loss function.

Having neuron value $q_j^l = \sum_{k=1}^m \theta_{jk}^l a_k^{l-1} + b_j^l$ at level l of the network with a weight θ_{jk}^l associated with the k^{th} neuron in the layer $l-1$ to the j^{th} neuron in the l^{th} layer, activation value a_k^{l-1} at the layer level $l-1$ (see Fig. 2.7), bias vector b_j^l, and a loss function $L(\theta_i^l)$, the goal is to calculate $\frac{\partial L}{\partial \theta_i^l}$ in order to find the direction of how parameters θ_i^l and b_j^k must be changed to minimize the cost function. This can be done by applying a chain rule as follows [41]:

$$\frac{\partial L}{\partial \theta_{jk}^l} = \frac{\partial L}{\partial q_j^l} \cdot \frac{\partial q_j^l}{\partial \theta_{jk}^l}. \tag{2.13}$$

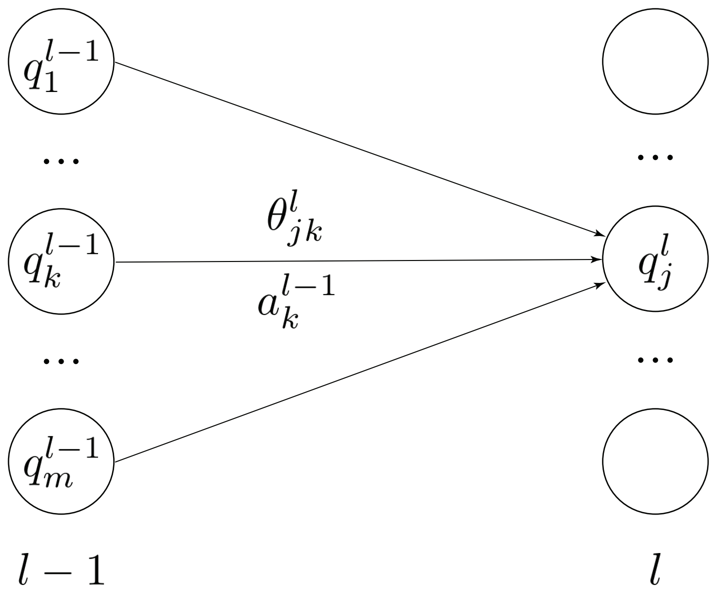


Figure 2.7: Figure depicting connections between neurons in layers $l-1$ and l with associated weights and activation.

Using the definition of q_j^l, Eq. 2.13 can be rewritten as:

$$\frac{\partial L}{\partial \theta_{jk}^l} = \frac{\partial L}{\partial q_j^l} \cdot a_k^{l-1}. \tag{2.14}$$

In order to find a partial derivative with respect to the bias, similar approach can by applied:

$$\frac{\partial L}{\partial b_j^l} = \frac{\partial L}{\partial q_j^l} \cdot \frac{\partial q_j^l}{\partial b_j^l}. \tag{2.15}$$

Using the definition of q_j^l, Eq. 2.15 can be rewritten as:

$$\frac{\partial L}{\partial b_j^l} = \frac{\partial L}{\partial q_j^l} \cdot 1. \tag{2.16}$$

The common part $\frac{\partial L}{\partial q_j^l}$ of Eq. 2.13 and Eq. 2.15 is called “local gradient" or “error of the j_{th} neuron at level l" and defined as:

$$\delta_j^l = \frac{\partial L}{\partial q_j^l}. \tag{2.17}$$

Thus, an equation for the rate of change of the loss function with respect to any weight in the network can be written as:

$$\frac{\partial L}{\partial \theta_{jk}^l} = a_k^{l-1} \delta_j^l. \tag{2.18}$$

Similarly, the rate of change of the loss function with respect to any bias in the network can be found as:

$$\frac{\partial L}{\partial b_j^l} = \delta_j^l. \tag{2.19}$$

Recursively applying a chain rule, local gradient δ can be found at any level of the network:

$$\delta_j^l = ((\theta^{l+1})^T \delta^{l+1}) \odot \sigma'(q^l), \tag{2.20}$$

where $(\theta^{l+1})^T$ is the transpose for the weight matrix θ^{l+1} for the layer $l+1$, $\sigma'(q^l)$ is a derivative of an activation function σ at q^l, and $\odot$ is a Hadamard product [42]. From the Eq. 2.19, it can be seen how the error is propagated through the network, from the output layer to the beginning. The algorithmic representation of the backpropagation method is shown in Alg. 2.2.2.

Algorithm 2.2.2 Backpropagation(θ_{init})

```
1: Set an activation function $a^1$ for the input layer
2: Pass input $x$ through the network and compute $q^l = \theta^l a^{l-1} + b^l$ and activation $a^l = \sigma(q^l)$ in each
   layer.
3: Compute error of the output layer $\delta^{out} = \frac{\partial L}{\partial a^{out}} \odot \sigma'(q^{out})$ where $L$ is a cost function .
4: for $l$=$out - 1...2$ compute:
5:     $\delta_j^l = ((\theta^{l+1})^T \delta^{l+1}) \odot \sigma'(q^l)$
6: return $\frac{\partial L}{\partial \theta_{jk}^l} = a_k^{l-1}\delta_j^l$, $\frac{\partial L}{\partial b_j^l} = \delta_j^l$
```

2.3 Convolutional Neural Network

A convolutional neural network (CNN) is a modified MLP generally used to process images. Even though MLPs are also able to classify images, CNNs outperform them in this area significantly. Firstly and mainly, CNNs processes images “as is" in multi-dimensional format, therefore preserving spatial information of the input signal. Secondly, the layers of CNNs are sparsely connected rather than fully connected as in MLPs; thus, the computations are more efficient with respect to the memory usage and processing time, which will become apparent as the components of a CNN are introduced later in this section.

2.3.1 Convolutional Layer

The main building block of any CNN is a convolutional layer. A convolution [43] is a repeated process of filters applied (the dot product) to the input image with a certain step, known as a stride (see *e.g.* Fig. 2.8). Thus, a convolutional layer performs convolution operations processing input images

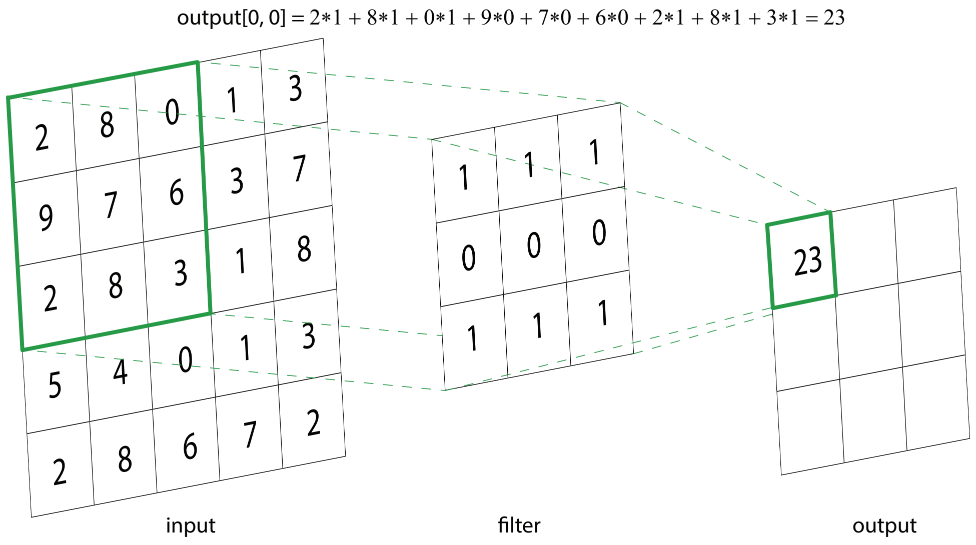


Figure 2.8: Example of a convolutional operation.

into a three-dimensional feature map. Compared to the MLP, CNNs store weights in the form of the small matrices of form $(K \times K)$, also called *filters*. These filters represent a receptive field of the convolutional layer and can be customized to fit the specific needs of the task. Depending on the task, each convolutional layer can be comprised of a D number of filters. Thus, the total number of weights in each convolutional layer can be defined as $K \times K \times D$ plus extra D weights for a bias. The formula for a convolution operation can be defined as:

$$I * h = \sum_{k,l} I(k,l) \cdot h(i-k, j-l), \tag{2.21}$$

where I is an input matrix of size $i \times j$ and h is a filter of size $k \times l$. Therefore, the output of the convolutional layer can be defined as:

$$x_l = f(x^{l-1} * h^l + b^l), \tag{2.22}$$

where x^l is the output feature map of the lth convolutional layer, f is an activation function, b^l is a bias, and $*$ is a convolutional operation for the previous layer output x^{l-1} and kernel h.

There is a set of main hyper-parameters that define the work of the convolutional layer. First, the number of filters in the convolution defines the depth of the output feature map. Second, the size of the kernel specifies the height and width of the convolution window. Third, the above-mentioned stride defines the step of the convolution window along the horizontal and vertical directions. Fourth, the padding is a number of zeros added to the borders of the input image. Assuming the input image has three channels (RGB), resolution $H_{input} \times W_{input}$ (where H_{input}, W_{input} stand for height and width, respectively), number of filters D, size of the kernels (K_{height}, K_{width}), stride S, and padding P (see Fig. 2.9); the formula for the output feature map shape can be expressed as:

$$\begin{aligned} Height_{out} &= \tfrac{(H_{input}+2P-K_{height})}{S} + 1, \\ Width_{out} &= \tfrac{(W_{input}+2P-K_{width})}{S} + 1, \\ Depth_{out} &= D. \end{aligned} \tag{2.23}$$

2.3.2 Pooling Layer

This layer performs a pooling operation on the input feature map. The most common pooling operation in computer vision tasks is maximum pooling [44], which means a maximum value within each patch of size (N, M) is picked and propagated to the next layer. The main goal of this procedure is to highlight the most present feature in the patch. Also, this performs a spatial reduction of the input features, thus, providing more efficiency to the deep network. This is because the pooling operation extracts the most significant features of an image, such as edges, points, *etc.* while reducing the dimensionality and variance of the input features. The work of a maximum pooling layer is depicted in Fig. 2.10. Besides maximum pooling, there are also average and global pooling layers. The average pooling layer, as can be surmised from its name, produces the average of all values in each patch. A global pooling layer takes an input feature map with a shape of $H \times W \times C$, where H is height, W is width, and C is depth

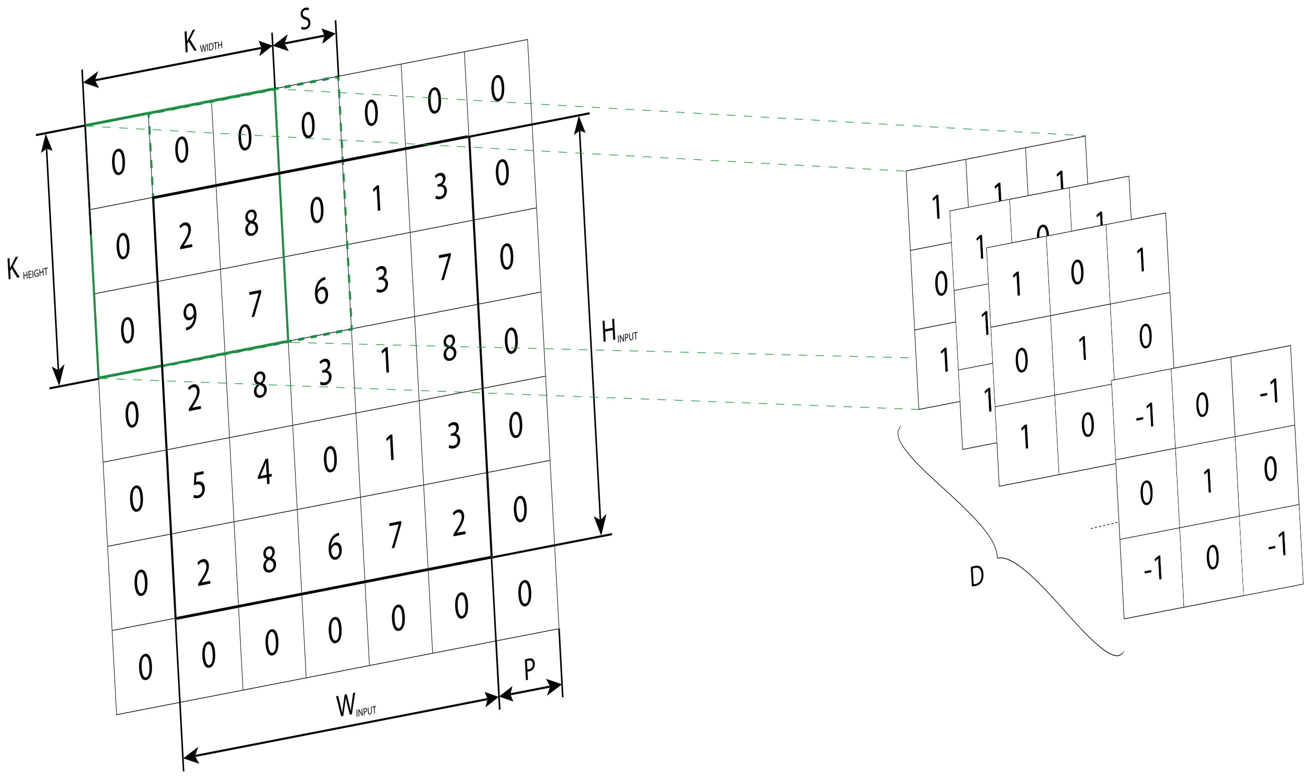


Figure 2.9: The hyperparameters of a convolutional layer.

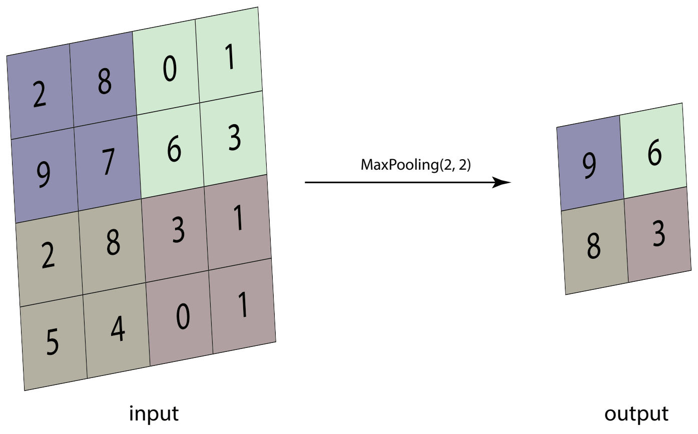


Figure 2.10: Example of a max pooling layer.

(or a number of channels); and reduces it to the shape $1 \times 1 \times C$. Only a single value is left, which is an average or maximum (depending on the type of global pooling layer) of all values in each channel.

2.3.3 Batch Normalization Layer

NNs that can do a good job of classifying new samples not contained in the test set are said to generalize to the problem domain. It often happens that a NN model trained on a specific dataset cannot properly generalize features of the validation dataset, thus having a low performance. This issue is called *overfitting* and can be fixed by introducing a regularization *dropout* layer which drops connections between neurons (according to a predefined probability) during training of the NN. In computer vision, however, a more advanced method was developed, also known as a batch normalization layer. A batch normalization layer enforces regularization and reduces generalization error in complex NNs [29]. During training, the weights of the network layers change; thus, the next output after each training iteration can be significantly different from the previous one. The next layer, which is trained on the distribution of the previous weight, is less effective when seeing a new distribution. Using batch normalization layers stabilizes the training process since the inputs to the next layers are normalized and significantly speeds up the training process [29]. Consider a batch B of inputs $\overrightarrow{X} = (x_1, x_2, \ldots, x_k)$, then the batch mean and variance are calculated as

$$\mu_B = \frac{1}{k}\sum_{1}^{k} x_i, \tag{2.24}$$

$$\sigma_B^2 = \frac{1}{k}\sum_{1}^{k}(x_i - \mu_B)^2. \tag{2.25}$$

After that, the inputs $\overrightarrow{X}$ are normalized through the formula:

$$\hat{x_i} = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}, \tag{2.26}$$

where $\hat{x_i}$ are the normalized inputs and ϵ is a small constant value keeping the denominator safe from zero-division. The normalized inputs are then squashed through a linear function with parameters γ and β, which are the only trainable parameters in the batch normalization layer. The final output of the layer can be expressed as

$$y_i = \gamma\hat{x_i} + \beta, \tag{2.27}$$

where y_i are the outputs of the batch normalization layer.

Another normalization layer which is frequently used in computer vision tasks and in this thesis particularly, is an instance normalization layer. In contrast to a batch normalization layer, the features are standardized within an instance yet not a batch. This is widely used in style transferring tasks [45].

2.3.4 Upsampling Layer

The goal of using this layer is to resize the input feature map using interpolation methods. The most common methods are nearest neighbours [46] and bilinear interpolation [47]. In CNNs, the resize operation is often used to bring the encoded (and downsampled) features back to the original size in order to be able to match them with the target label. Functionally, upsampling layers are also used as an alternative to transposed convolution layers [43]; however, technically, upsampling layers work as an opposite to the pooling operation. The height and the width of the input feature map are re-sized to the needed dimensions and intermediate pixels are populated according to the chosen method. The work of the upsampling layer is depicted in Fig. 2.11.

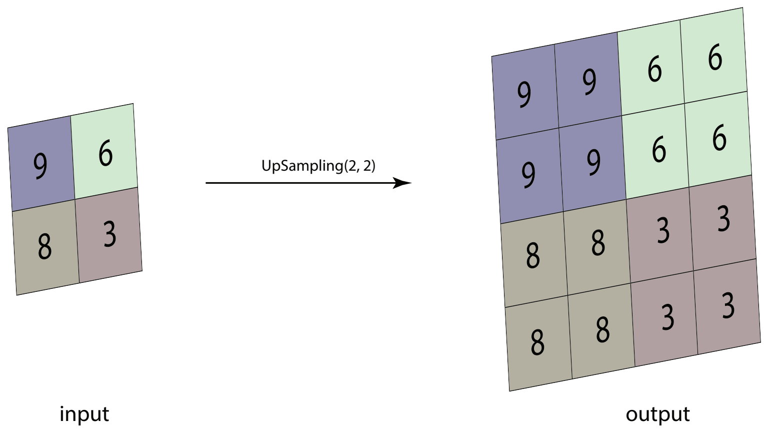


Figure 2.11: Figure depicting an upsampling layer.

2.3.5 Skip Connections

Skip connections are not a layer but a method of transferring features from one part of the network to another. One benefit of this method is that it allows the recovery of spatial information in the latter layers in deep neural networks [48]. Another benefit of using skip connections is that they neutralize the vanishing gradient problem. Assume there are many layers in the NN, and the gradient is propagating back through each layer using the chain rule (see Section 2.2.3). As a result, the earliest layers of the network receive gradient value close to zero because it was multiplied many times by fractional numbers beforehand. Thus, the weights of the layers in the beginning of the network do not or barely update. Before 2014, this was one of the major obstacles in building deeper networks. Skip connections allow the gradient to propagate by skipping the furthest layers and bringing it directly to the earliest layers of the NN.

2.3.6 Adaptive Instance Normalization Layer

The Adaptive Instance Normalization (AdaIN) Layer was first proposed by Huang *et al.* in [25] and was developed for the fast and effective style transferring between images in different domains. Given content input from the source images and style input from target images, the AdaIN layer adjusts the mean and variance of the content input, thus, forcing it be similar to the style input in terms of the distribution. AdaIN does not have trainable parameters, it adaptively calculates affine transformation parameters from the target domain:

$$AdaIN(c, s) = \sigma(s) \left(\frac{c - \mu(c)}{\sigma(c)} \right) + \mu(s), \tag{2.28}$$

where c is a content input, s is a style input, σ returns the variance of the input, and μ returns the mean of the input. As can be seen from the equation, this layer scales content input by $\sigma(s)$ and shifts it by $\mu(s)$.

2.4 Generative Adversarial Networks

A generative adversarial network (GAN) is a network where each part of it contests with each other in form of a zero-sum game [49]. It was first proposed by Goodfellow *et al.* in [12]. The wide use of this kind of model began quite recently, in 2014, and this area is still in active development. Currently, development of GAN-based NNs is driven mostly by style transferring and data augmentation tasks [16, 19, 20, 21].

2.4.1 Generator vs Discriminator

Given statistically different source and target datasets, the generator (G) part of the GAN outputs predictions from source inputs which have the same statistics as the target dataset distribution (*i.e.* the dataset of interest). This is achieved through adversarial learning which is also known as unsupervised learning because there is no predefined class label vector to match predictions with, and, thus, update the weights of the model. Instead, there is a part of this model, called discriminator (D), which is responsible for detecting fake (the outputs from G) and real (from the target distribution) signals.

G is a model which generates fake outputs with statistics close to the target distribution. D, in turn, serves as a data evaluator and decides whether the sample is from the target dataset or it is a fake generated by G. In this game, the goal of the generator is to maximize an error of D (or minimize its own error in faking the samples), *i.e.* to make D decide wrong, and D is aimed to be able to maximize its probability of correct labelling target samples and samples from G. In other words, the task of G and D can be expressed as a value function $V(D, G)$:

$$\min_{G} \max_{D} V(D, G) = \mathbb{E}_x[\log D(x)] + \mathbb{E}_z[\log(1 - D(G(z)))], \tag{2.29}$$

where z represents samples from the source dataset, $D(x)$ represents the probability that x came from the distribution of interest, and $G(z)$ is an output from the generator.

2.4.2 Convolutional Generative Adversarial Networks

Originally, D and G are both MLPs, where G outputs vectors of arbitrary size, yet D outputs only a single probability value. However, in computer vision problems, both models use CNN structures to deal with images as inputs. The most general representation of a fully convolutional generative adversarial network (FCC-GAN)[50] is depicted in Fig. 2.12.

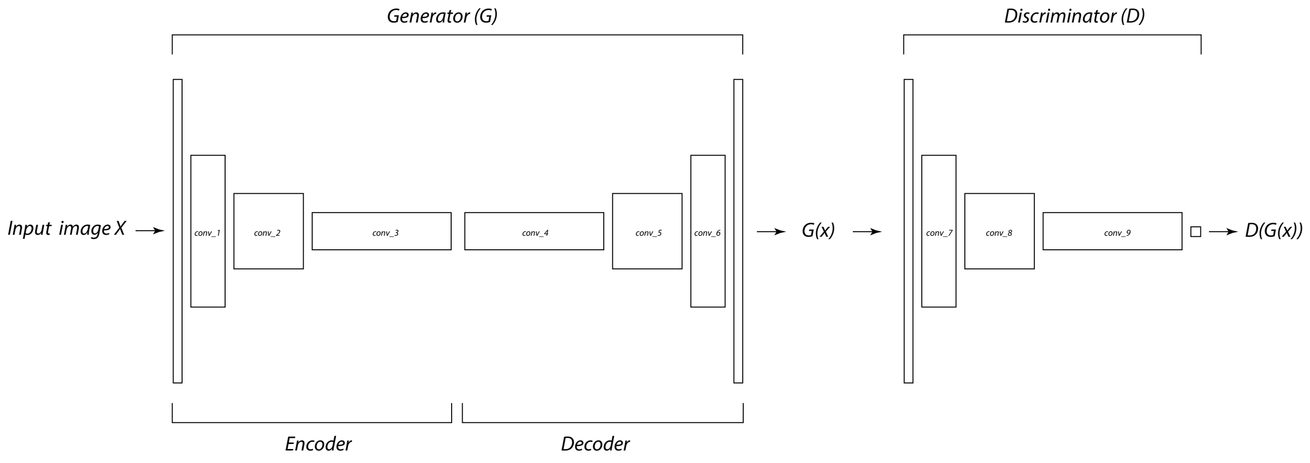


Figure 2.12: Fully Connected and Convolutional GAN.

The G in an FCC-GAN has an encoder-decoder (see Section 2.3) architecture and its last layer usually ends with a hyperbolic tangent activation function to map outputs from -1 to 1, or sigmoid to map from 0 to 1, depending on the type of input data scaling. In other words, the output of G is also an image. The architecture of D in an FCC-GAN is similar to the encoder part of a CNN. The input image is passed through the network, and the lower level features are extracted. Based on the extracted features, a single-filter convolutional layer is applied on top of D and also can be extended by a sigmoid activation function to output probability.

2.4.3 Training of Generative Adversarial Networks

The underlying algorithm used to train GANs is the backpropagation method discussed in Section 2.2.3. However, since the architecture is supposed to use at least two components (discriminator and generator), the training steps are slightly different. The overall algorithmic representation of the GAN training process can be expressed as follows:

Algorithm 2.4.1 GANTraining(D, G, X, T, MaxIterations)

```
1: Initialize weights for the generator $G$ and discriminator $D$.
2: for iter=1...MaxIterations:
3:     Pass source inputs $x \in X$ through the $G$ and get outputs $\hat{y} = G(x)$
4:     Freeze weights of $D$ and pass $\hat{y}$ through $D$ and get probability $p_g$
5:     Compute adversarial loss for the $G$ by calculating how close the $\hat{y}$ is to $T$ according to the $D$.
6:     Update weights of $G$
7:     Unfreeze weights of $D$ and pass $\hat{y}$ and target inputs $t \in T$ trough the network $D$.
8:     Compute adversarial loss for the $D$ by calculating how close the $\hat{y}$ is to $X$ and how close the
   $D(t)$ is to $T$ according to the $D$.
9:     Update weights of $D$
```

Chapter 3

Literature Review

This chapter provides an overview of the most recent developments in image segmentation. Afterwards, state-of-the-art methods of domain adaptation in computer vision are also considered. Finally, unsupervised domain adaptation methods for remote sensing imagery are explored in the last section.

3.1 Deep Learning

There have been almost fifteen years since the advancements in the graphics processing unit (GPU) development made a huge leap by bringing CUDA technology to the market, and parallel calculations using GPUs have outperformed traditional CPU-only systems [51]. This opened previously inaccessible opportunities toward the development of NNs and deep neural networks (DNN) with a significant number of hidden layers and parameters because the operations inherit in NN are parallel in nature [4]. The training (or learning) process of DNNs became reasonable with operations that previously took months or weeks reduced to days or hours. This resulted in the success of AlexNet [30] that paved the way for deep learning applications in areas such as computer vision [52], voice [53] image processing [54], natural language processing [55], big data [56], and others [57]. This project was inspired by the recent application of DNNs developed for performing image semantic segmentation and applied to the fields of remote sensing [10, 58, 59] and domain adaptation [24, 60, 61].

3.2 Semantic Segmentation

As was shown in recent publications, any classification convolutional neural network (CNN) can be transformed into a fully-convolutional network (FCN) for image segmentation [62]. The main idea is that all fully connected layers can be replaced by fully convolutional layers. This, in turn, allowed CNNs developed for classification to be augmented (with more trainable layers) to enable them to produce outputs with dimensions that matched the input, which is vital for image semantic segmentation tasks. Good examples of FCNs are AlexNet [30], VGGNet [63] and ResNet [64] architectures. All of them achieved the best results via the PASCAL VOC Evaluation Server [65] in the year they were developed. Besides replacing fully connected layers, these models are also incorporated skip connections [48] which allowed them to enhance segmentation performance. At the same time, the concept of convolutional *encoder-decoder* appears. The encoder of a CNN is considered a part of the model which maps raw image pixels to a rich representation in the form of multi-dimensional feature vectors. Contrary, the decoder maps these features back to the “raw" representation, thus, producing the model’s output.

Another important innovation that pushed the accuracy of image segmentation tasks forward was the introduction of dilated convolution [5, 66] (also called atrous convolution [67, 68]). This method expands receptor field inputs and, thus, increases the information extracted from the features (rather than reduces the amount of information as is the case in the traditional convolution operation). The one disadvantage of this method, however, is that it suffers from so called “griding" artifacts. The later research by Yu *et al.* in [69] removes these artifacts and improves the segmentation accuracy of the NN.

The encoder part of a NN used in semantic segmentation has also undergone a significant metamorphosis over time. AlexNet [30], developed in 2012, possessed five convolutional blocks and won the ILSVRC 2012 [70] with this design. Two years later, in 2014, the GoogLeNet [71] won ILSVRC 2014 using 22 layers and the “network-in-network" structure, which is a convolutional layer design built on smaller networks. The VGG architecture was also developed in 2014 and had the comparable number of layers - 16 or 19, depending on the modification. The VGG was another model that performed well in the ILSVRC 2014. One year later, in 2015, the ResNet model (with up to 152 layers) was a true breakthrough in the ILSVRC 2015 and MS COCO 2015. It was not only highly accurate but also the deepest among all models developed up to this time due to the introduction of skip connections. Another important architecture widely used in image segmentation was the UNet model, developed in 2015 by Ronneberger *et al.* [72]. This model is an improvement of the FCN proposed by Long *et al.* in [62].

Meanwhile, the remote sensing industry significantly grew each year. New satellites are launched almost every year downstreaming more and more data. To process this data (*i.e.* label each pixel in a satellite image), trained specialists labour was applied, which is a major obstacle to producing products derived from this satellite data at frequent intervals. The success and variety of DL semantic segmentation models inspired scientists in the remote sensing domain to start using these approaches [10, 59, 73]. However, transferring these models to remote sensing data was not straightforward since it required specific attention to features of satellite images. In fact, there were three main hurdles to transferring these techniques, which are described below.

Firstly, each pixel of the satellite image is potentially semantically meaningful. For example, a pixel can represent a background or water body; thus, the pixel-level accuracy of the potential model is a crucial condition. Another feature of satellite images is that they usually have more than three channels. Sometimes the number of channels does not allow the application of traditional frameworks “as is," thus making training “out-of-the-box" methods impossible. Thirdly, the remote sensing images contain satellite-view specific objects such as the tops of trees, rivers, and rooftops. Thus, publicly available image datasets such as ImageNet [74] or PASCAL VOC cannot be used for the segmentation model training.

To address the first pixel-level accuracy related issue, dilated convolutions were proposed by Yu *et al.* [66] in 2016. As discussed above, this method increases the amount of spatial information by integrating the elements at nonadjacent positions in a kernel. In 2018, Xu *et al.* [75] proposed an FCN for buildings segmentation which they called Res-U-Net. This model uses hand-crafted features (such as the normalized differential vegetation index, the normalized digital surface model, *etc.*) and a concept of a guided filter, which was firstly proposed by He *et al.* in [76]. Chen *et al.* [5] proposed DeepLab - a semantic segmentation framework with an atrous spatial pyramid pooling decoder. It delineates objects at different scales by passing the output from the encoder to several convolutional layers with different sampling rates. Cheng *et al.* [77] proposed extended SegNet, which employed an edge-detecting network and a segmentation network. The edge map, generated by the edge-detecting network, is passed to the segmentation network to fine-tune the results. Another edge-related method was proposed by Lu *et al.* [78], where the mixed gradient loss is introduced to control the accuracy of the objects' edge prediction. The Sobel gradient [79] is applied to the inputs and the outputs and the mixed gradient loss is a mean square error between these two gradients. Marmanis *et al.* [80] proposed a fusion-based strategy to improve the pixel-level accuracy. In his work, he proposed a FCN with two separate segmentation networks working in parallel on different types of input data. The first model processes the height information from a digital elevation model, and the second model processes the input colour channels. The outputs from both networks are concatenated and passed to the point-wise convolutional layer to combine the features and generate a final prediction. Achanta *et al.* [81] proposed a post-processing method for increasing the prediction accuracy. This is a superpixel algorithm known as simple linear iterative clustering (SLIC), which is based on k-mean clustering approach to generate superpixels. The idea of using superpixels is that labels of neighbouring pixels are strongly correlated since the objects in the input images are spatially continuous. Thus, if the majority of the nearby pixels were classified as the same class, then the whole cluster is likely to have this class. Another post-processing method to improve the accuracy of predictions was proposed by Chen *et al.* [5] and known as conditional random fields (CRFs). This method recovers object boundaries after the smooth score map of the input image is generated by the deep convolutional neural network (DCNN). The method is based on using the energy function where the unary potential is a class label predicted by the DCNN, and the pairwise potential includes an input image with spatial and contextual information. Sun *et al.* [82] proposed to use a multi-filter CNN and multi-resolution segmentation (MRS) approach to achieve a higher object boundary accuracy. The multi-filter CNN combines and processes LiDAR data and high-resolution optical imagery via a data fusion for semantic labelling. Afterwards, the MRS is used to delimitate object boundaries for the salt-and-pepper artifacts reduction.

To address the second issue, related to the multi-spectral specificity of the remote sensing imagery, the following ideas were proposed. Ghamisi *et al.* [83] proposed using fractional order Darwinian particle swarm optimization (FODPSO) for selecting the most informative bands and then passing them to the final classifier. Zhao *et al.* [84] proposed a spectral-spatial feature-based classification (SSFC) framework. Within this framework, deep learning and dimension reduction are used to extract spectral and spatial features of the input multispectral image. First, the CNN extracts high-level spatial-related features. Then, spatial and spectral features are stacked together, and the fusion feature is extracted. Finally, the multiple-feature-based classifier is trained. Yu *et al.* [85] proposed using 1x1 convolutional layers to extract the hyperspectral image information. Also, an average pooling layer and dropouts with higher rates have been employed to alleviate the overfitting problem. Li *et al.* [86] proposed using a 3D convolutional neural network (3D-CNN) to process multi-band satellite imagery. The idea is that hyperspectral satellite imagery can be treated as a 3D cube. Thus, it is reasonable to use 3D filtering for the simultaneous extraction of the spectral-spatial features. Fang *et al.* [87] proposed using a 3D dense convolutional network with a spectral-wise attention mechanism (MSDN-SA). This method employs 3D dilated convolutions to concurrently extract spatial and spectral information at different scales and densely connects all extracted feature maps with each other.

To deal with the third issue, related to the lack of a sufficient number of training samples and publicly available datasets, domain adaptation (DA) methods were developed. Domain adaptation is a technique of applying a DNN model trained on a dataset where labels are available (source dataset) to a dataset where labels are missing (target dataset). DA resolves the discrepancy between source and target datasets by aligning feature distributions the way a semantic segmentation model is able to successfully classify target samples. The DA algorithms that were developed to overcome the lack of available labels for semantic segmentation model training are considered in the next section.

3.3 Domain Adaptation

Domain adaptation techniques are used to close the gap between source and target domains. They became popular after Ganin *et al.* [60, 88] proposed unsupervised domain adaptation through backpropagation. Since then, many variants and model architectures that fit the domain adaptation task have been proposed. The most common approaches aim to align features across two domains the way a semantic segmentation model can generalize them. For instance, a feature-level adaptation method was proposed by Tzeng *et al.* [89], where he uses so-called domain confusion loss, which directly minimizes the distance between source and target representations, thus initializing domain invariance. Also, there is a classification loss which solves the image classification task. Hong *et al.* [90] proposed using a conditional generative adversarial network for structured domain adaptation. The central part of the proposed model is a conditional generator which is aimed to enhance source domain features to have a similar distribution as the target. The conditional generator consists of several residual blocks. There is also a CNN encoder with five convolutional layers that extracts target features. The target domain features and enhanced source domain features are then passed through the discriminator represented by a multi-layer perceptron. Thus, the domain adaptation task is resolved via adversarial learning. At the same time, the semantic segmentation task solved after the cross-entropy loss is calculated after passing the enhanced source features through the deconvolutional layer. Another feature-level adaptation method was proposed by Tzeng *et al.* [91], which combines adversarial learning with discriminative feature learning. In particular, during the training, the model must learn a discriminative mapping of target images to the source feature space. The model consists of a CNN feature encoder and an FCC discriminator, and the domain adaptation is solved via adversarial learning. The methods mentioned above suffer a common issue - they do not enforce semantic consistency while aligning feature representation of both domains. However, this is crucial when the final goal is the semantic segmentation of target samples. The following pixel-level domain adaptation methods were developed to resolve this issue.

Tsai *et al.* [13] proposed the AdaptSegNet model, where the task is to adapt features in the output space. It is analogous to semantic segmentation, where the output space shares similarities, such as spatial layout and local context, between source and target domains, even if the domain shift between them is significant. The domain adaptation task is solved through adversarial learning, where the semantic segmentation model works as a generator and tries to fool the external discriminator model by producing indistinguishable features in the output space. Additionally, a categorical cross-entropy loss ensures that adapted source features are still properly classified, thus, preserving semantic consistency. Originally based on CycleGAN [19] and feature-level adaptation methods proposed in [60, 91], the CyCADA model was proposed by Hoffman *et al.* [24] to preserve semantic consistency through pixel-level unsupervised domain adaptation. The model uses a noisy labeller, which is a semantic segmentation model trained on source data and applied to target data without adaptation. After that, a pixel-space adaptation process is performed using two generators and two discriminators. The first generator and discriminator work to translate the target domain feature representation to source samples, and the second pair of generator and discriminator work as a whole to translate the source domain feature representation to target samples. The noisy labeller, trained previously, is integrated into the training process and encourages an image to be classified in the same way after translation as it was before translation, according to this classifier. Finally, the semantic segmentation model with feature-level adaptation is trained on source samples stylized as target ones. In this project, however, the noisy labeller was replaced by the pixel-level adaptation model proposed by Tsai *et al.* [13], and the feature-level adaptation at the final stage was removed because the model without it showed better performance. Another pixel-level adaptation method, which is called bidirectional learning, was proposed by Li *et al.* [92]. The authors explore a model where two separate modules cooperate. There is an image-to-image translation model and segmentation adaptation model, both similar to the ones proposed in [24]. The learning process involves two directions: "translation-to-segmentation" and "segmentation-to-translation," and that is why the whole process is a closed-loop cycle. Moreover, a self-supervised learning (SSL) approach is incorporated into a segmentation adaptation module. This way, different from segmentation models trained only on source data, the segmentation adaptation model uses both source and target images for training. Iteratively predicting labels for the target domain, they are considered as the approximation of the ground truth labels. Thus, only those with high probability are used in training over and over again. After the segmentation model is trained on the source and target labels, it is used as the noisy labeller discussed above, thus making the training process looped. Even though these two methods are both focused on the pixel-level adaptation task, where semantic consistency is preserved, the source and target datasets they were trained on are represented by first-view car driving samples. The following pixel-space adaptation methods were applied to the remote sensing imagery and are claimed to be effective.

The whole group of the pixel-level adaptation methods which were validated on remote sensing imagery were proposed by Tasar *et al.* [14, 15, 16]. His first method is called ColorMapGAN and is aimed to linearly shift each source sample band distribution to match the target domain distribution. At first, a U-Net [72] classifier is trained only on source samples and corresponding labels. After that, a ColorMapGAN module is trained to transfer the visual appearance of the source samples to look like target ones. The ColorMapGAN consists of a generator and a discriminator. The generator is an architecturally simple construction that is represented by only scaling and shifting matrices. Thus, each source sample band (red, green, and blue) is passed through the scaling and shifting operations. The output of such transformations is then passed through the discriminator, which is similar to [62]. The discriminator decides how close the transformed source sample is to the target distribution. The generator's goal is to fool the discriminator by faking the source images. It is crucial to notice here that the semantic consistency is preserved during such transformations because there are neither convolutional nor pooling layers. The final step of the training process is to fine-tune the initial classifier with faked source samples and their corresponding labels. The drawback of the ColorMapGAN is that it processes each band separately, which generates slightly noisy outputs. Another method that was developed with application to remote sensing data is called semantically consistent image-to-image translation (SemI2I). The idea of this method is close to [19] but has some specific differences. First, the image-to-image translation module is operating using the adaptive instance normalization (AdaIN) (discussed in Section 2.3.6) layer between the encoder and decoder part. During training, this layer scales and shifts the input source sample feature representation to match the target domain's accumulated mean and variance. Also, the image-to-image part consists of relatively shallow convolutional models; thus, it can be trained quickly. The following losses enforce semantic consistency of the translated source images: cross-reconstruction loss, self-reconstruction loss, and image gradient loss computed for the original source image and its translated to target domain representation copy. Additionally, the authors re-size the low-level features extracted by the first convolutional encoder layer and concatenate them to each deconvolution layer in the corresponding decoder. After the image-to-image part is trained, a semantic segmentation model is trained on translated source images with corresponding labels and validated on the target dataset The last model proposed by this author is called DAugNet and is a simplified (but still effective) version of the previously discussed method. There is only one image-to-image translation part for the source-to-target and target-to-source style direction. There is also only one discriminator which estimates how accurate the style translation was. It can be used to evaluate both direction translations because it has domain-specific output layers. The scaling and shifting of feature representation are performed by constant predefined target-style vectors for the mean and variance, which do not change during the training. The same losses enforce the semantic consistency of the transformed samples as in the previously discussed method. In this project, the low-level features, as in the SemI2I model, are added to the DAugNet model to increase semantic consistency. After the style transferring part is done, the target-like source samples are used for semantic segmentation model training and validation on the target dataset.

Chapter 4

Generating a Model

This chapter discusses details of the proposed models. It first explains the architecture of each model and then the structure of the objective functions assigned. The last section is dedicated to explaining the modifications made to the models to improve their performance.

4.1 Baseline model

A baseline model was used in order to conduct experiments without domain adaptation. It is a pure semantic segmentation model trained on the source (GE-1) images and validated on the target images (WV-3). That is to say that there was no adversarial learning during the model training, and, thus, it was used as a comparison model to those described in the following sections.

As a baseline model, the DeepLab-v2 [5] framework with ResNet101 [64] as a backbone architecture, pre-trained on ImageNet [74] dataset, was used. To enlarge the receptive field, dilated convolution layers were applied to Conv3 and Conv4 layers with dilation of 2 and 4, respectively [66]. Also, the stride parameter in these layers was changed to 1. This decreased the original downsampling rate from 32 to 8. As a decoder and final classifier of the model, Atrous Spatial Pyramid Pooling (ASPP) [5] layer was used. The encoder part of the baseline model is represented in Fig. 4.1, and its decoder part is represented in Fig. 4.2.

One significant difference from the original DeepLab-v2 architecture was in the number of input channels of the first layer of the baseline model. In order to work with multi-band satellite images, the number of input channels was changed from 3 to 4. Since the model was pre-trained, the Red channel pre-trained weights were copied to the newly added fourth Near IR channel.

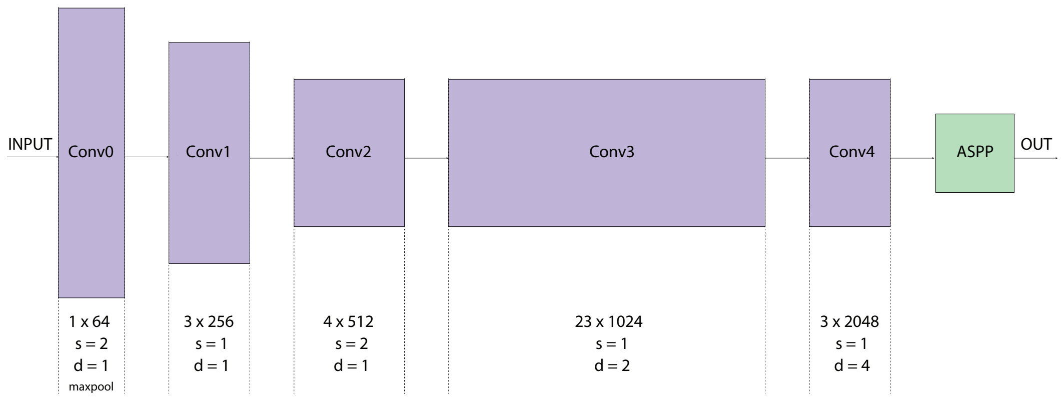


Figure 4.1: Baseline model encoder.

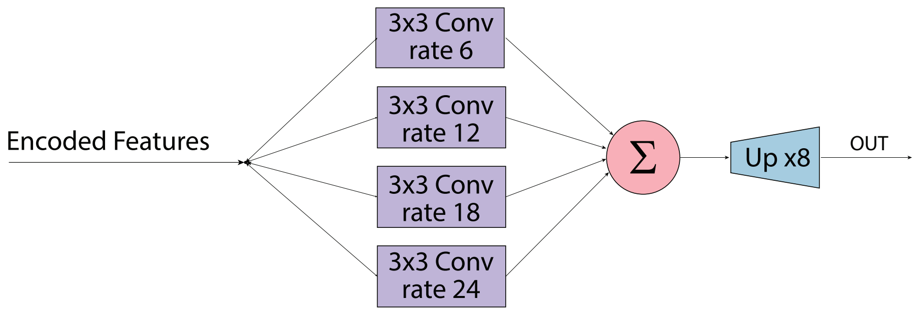


Figure 4.2: Baseline model decoder implemented with an atrous spatial pyramid pooling layer.

The baseline model was used as a segmentation model in Sections 4.3, 4.4, 4.5, after the style-transferring part. It was also used in 4.4 as a noisy labeller. The model was optimized using only one cost function, which is a classic cross-entropy function implemented using the formula provided in Section 2.10.

4.2 AdaptSegmNet

The AdaptSegNet [13] adopts DeepLab-v2 framework as a backbone. It is similar to the baseline model implementation, yet several modifications were made to enforce adversarial learning. Firstly, an extra (in addition to the traditional decoder) ASPP decoder layer was added to the intermediate part of the model, between blocks Conv3 and Conv4. Thus, the features extracted from Conv3 are passed directly to the intermediate decoder, to yield a full size prediction. The backbone model with both decoders composes a generator in terms of the GAN problem. Secondly, the discriminator part of the model is represented by two fully-convolutional discriminators that were used assigned to each of the decoder outputs. Their goal is to decide which domain the output space features. The goal of the generator is to generate indistinguishable features in the output space for the source and target domains. The architecture of the model is depicted in Fig. 4.3.

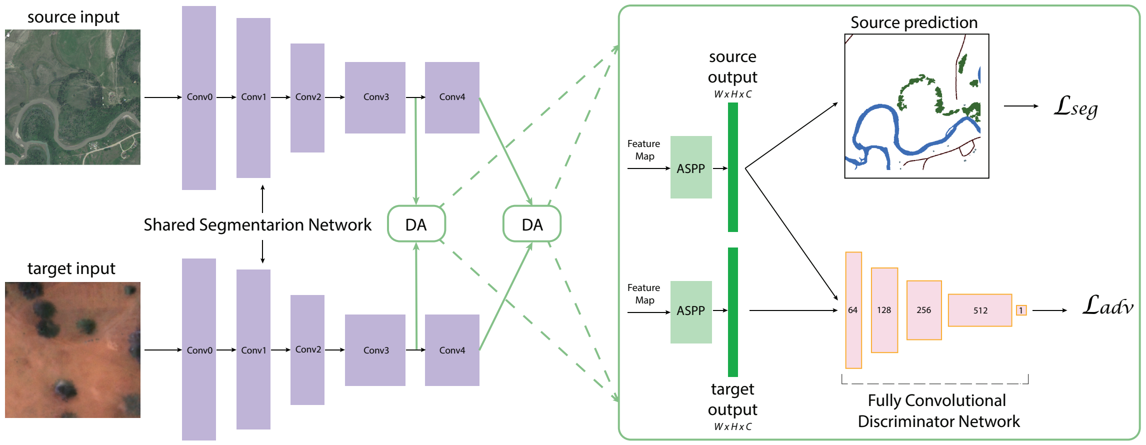


Figure 4.3: AdaptSegNet architecture.

The loss function of the AdaptSegmNet can be expressed as:

$$L(I_s, I_t, Y_s) = L_{seg}(I_s, Y_s) + \lambda^1_{adv} L^1_{adv}(I_t) + \lambda^2_{adv} L^2_{adv}(I_t), \tag{4.1}$$

where I_s and I_t represent the source and target input samples respectively, Y_s are corresponding labels for the source domain images, L_{seg} is a cross-entropy loss of the segmentation task, L^2_{adv} is an adversarial loss of the discriminator for the output after the last convolutional layer, and L^1_{adv} is an adversarial loss of the discriminator for the intermediate output after Conv3 block. λ^1_{adv} and λ^2_{adv} are coefficients assigned to each adversarial loss to control their influence on the final objective.

4.3 CycleGAN

The CycleGAN model used in this project can be split into several individual models - the "image-to-image" style-transferring model and a segmentation model. The segmentation model here is one described in Section 4.1. The style-transferring model, in turn, can be further split into four separate modules. The first module is a "source-to-target" "image-to-image" generator. Its goal is to process the source domain images and output target-like images containing source content. The second module is similar to the first one, except its goal is to process the target domain images and output source-like samples containing target content. Both modules are represented by two convolution layers with kernel size 3 x 3 and stride parameter equal 2, nine residual blocks [64], and two transpose convolution layers with kernel size 3 x 3 and parameter equal 1. The last layer maps decoded features back to the 4-channel blue, green, red, and near-infrared (BGRN) space and then applies a hyperbolic tangent function. Each generator module is accompanied by an individual discriminator, which is used to estimate how well the style-transferring task is performed. For the discriminator networks, so-called LSGANs [93] are used. The overall style-transferring architecture of the CycleGAN is depicted in Fig. 4.4. The segmentation architecture is the same as for the baseline model discussed in Section 4.1.

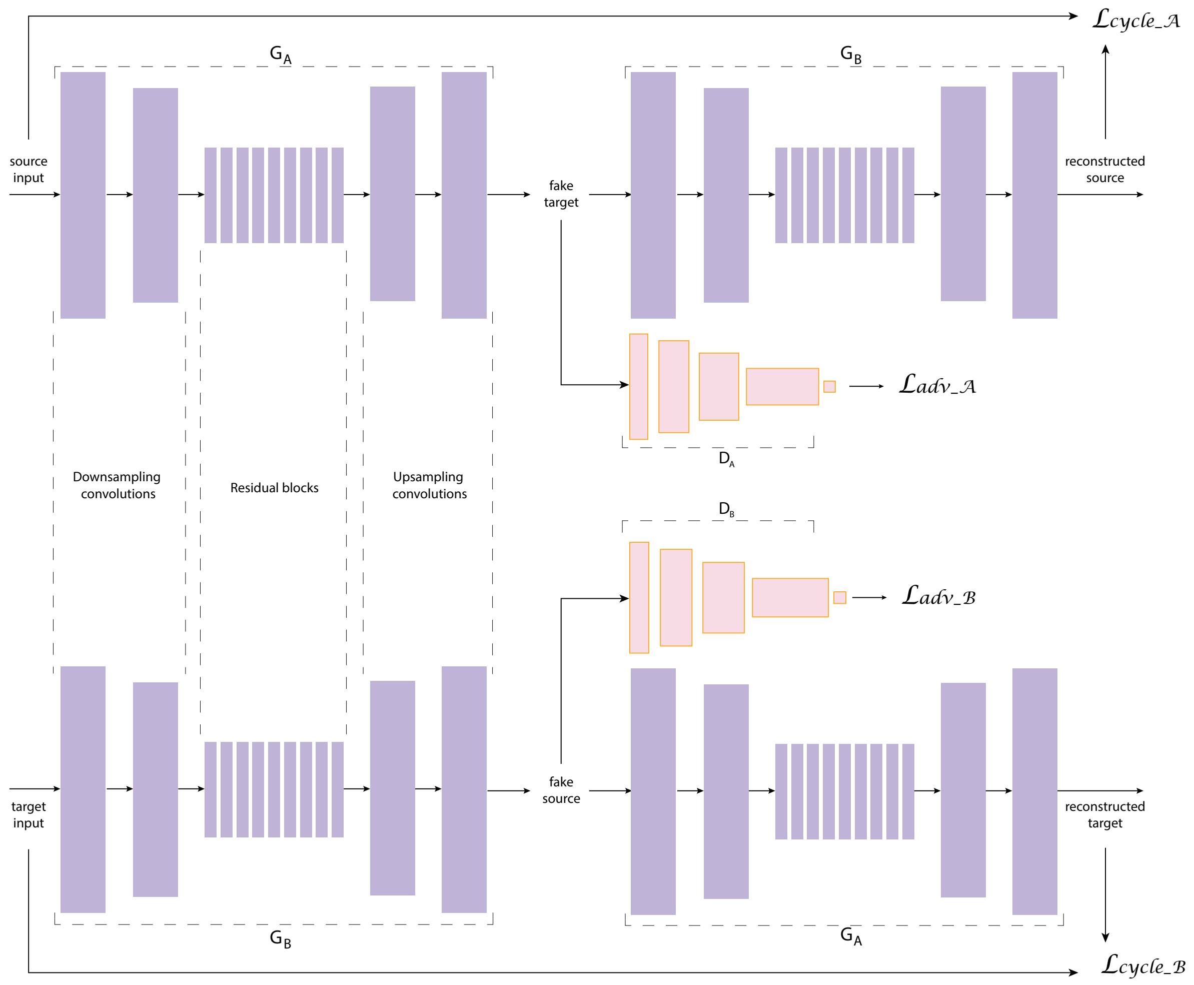


Figure 4.4: Architecture of the CycleGAN style-transferring component. G_A represents the "source-to-target" style generator, and G_B represents the "target-to-source" style generator. D_A is a discriminator judging target-like source images with a corresponding loss L_{adv_A}, and D_B is a discriminator judging source-like images with a corresponding loss L_{adv_B}. L_{cycle_A} is a cycle loss for the source domain, and L_{cycle_B} is a cycle loss for the target domain.

The first stage of the CycleGAN training is about transferring the style of the target domain images to the source ones. After the source images are translated to target domain style, the semantic segmentation model is trained on the translated source images with corresponding labels, and validation is made for the subset of target samples.

The loss function of the CycleGAN model and consequently applied segmentation model can be expressed as:

$$L(I_s, I_t, Y_s) = L_{style}(G_A(I_s), I_t) + L_{seg}(G_A(I_s), Y_s), \tag{4.2}$$

where I_s and I_t represent the source and target input samples respectively, Y_s are corresponding labels for the source domain images, and G_A is a "source-to-target" style transferring model. The segmentation loss L_{seg} is a cross-entropy loss while the style-transferring loss L_{style} can be further expressed as:

$$\begin{aligned} L_{style}(G_A, G_B, D_A, D_B, I_s, I_t) = L_{adv}(G_A(I_s), I_t) + L_{adv}(G_B(I_t), I_s) + \\ L_{cycle}(G_B(G_A(I_s)), I_s) + L_{cycle}(G_A(G_B(I_t)), I_t), \end{aligned} \tag{4.3}$$

where G_B is a "target-to-source" generator, D_A and D_B are discriminators which assess the outputs from G_A and G_B respectively, and L_{cycle} is a reconstruction loss represented by L_1-norm, discussed in Section 2.2.

4.4 CyCADA

Technically, the CyCADA [24] is a significantly improved version of the CycleGAN model, discussed in the previous section. The main improvement is related to the semantic consistency control during the style-transferring phase. In CycleGAN, the goal of the model is to make the source domain distribution close to the target domain distribution without limitation with respect to the semantic component of the samples. For instance, using a CycleGAN, the building in the source sample can be replaced during the style-transferring process by, for example, a bunch of trees or nothing at all. To control this, the so-called noisy labeller was introduced prior to the style-transferring phase. Since the source dataset has related labels, the segmentation model with the same architecture as in Section 4.1 is trained, and its weights are fixed. Using this pre-trained segmentation model f, the labels for the source domain, and the noisy labels for the target domain are generated during the style-transferring model training. This way, the original images from both domains and their style-translated versions are encouraged to have the same labels, thus, be semantically consistent. The style-transferring architecture of CyCADA is depicted in Fig. 4.5. The segmentation architecture is the same as for the baseline model discussed in Section 4.1.

The final loss function can be expressed as:

$$L(I_s, I_t, Y_s) = L_{style}(G_A(I_s), I_t) + L_{seg}(G_A(I_s), Y_s), \tag{4.4}$$

where I_s and I_t represent the source and target input samples respectively, Y_s are corresponding labels for the source domain images, and G_A is a "source-to-target" style-transferring model. The segmentation

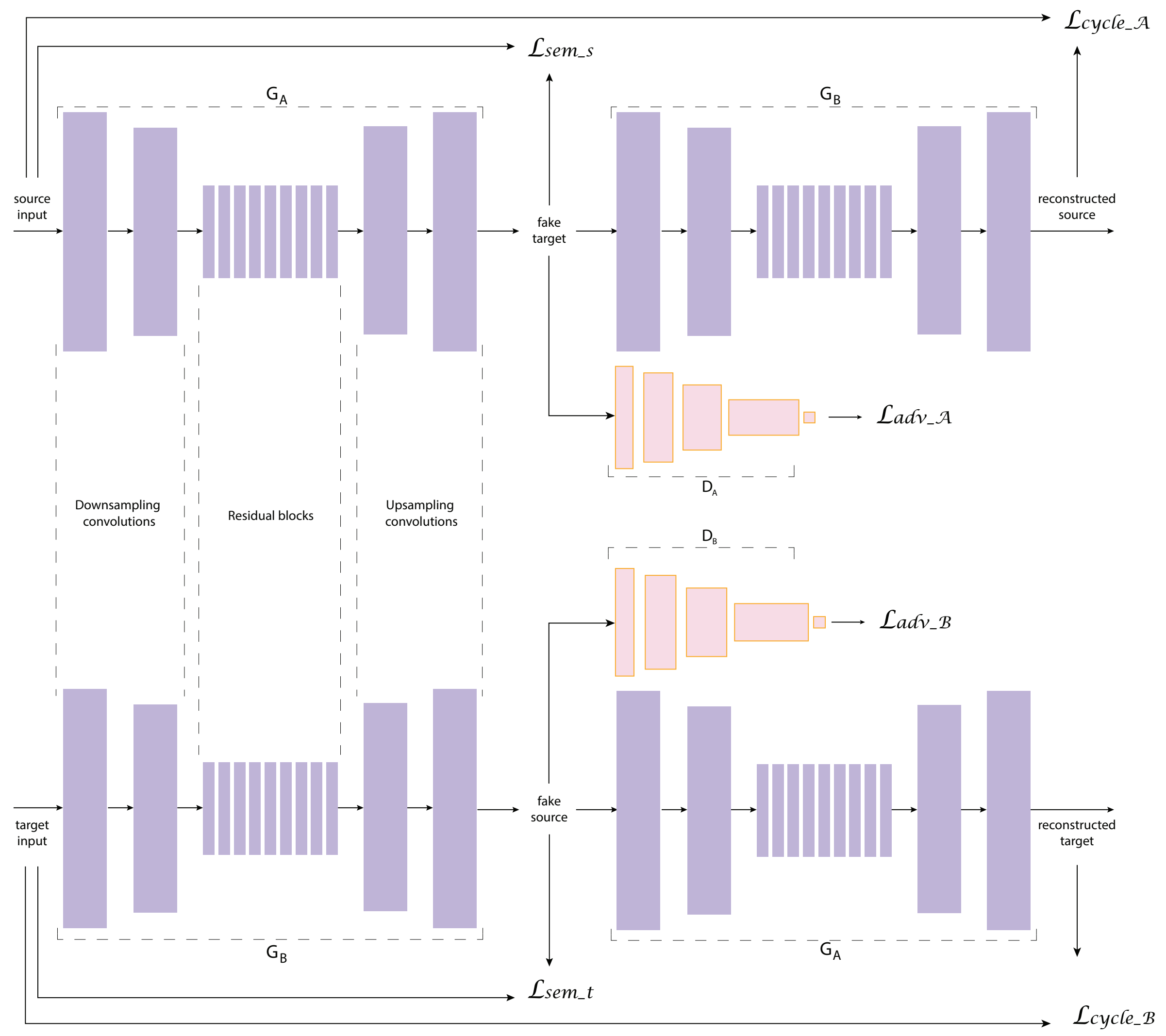


Figure 4.5: Architecture of the CyCADA style-transferring component. G_A represents the "source-to-target" style generator, and G_B represents the "target-to-source" style generator. D_A is a discriminator judging target-like source images with a corresponding loss L_{adv_A}, and D_B is a discriminator judging source-like images with a corresponding loss L_{adv_B}. L_{cycle_A} is a cycle loss for the source domain, and L_{cycle_B} is a cycle loss for the target domain. L_{sem_s} is a semantic consistency loss initiated by a noisy labeller for the source samples, and L_{sem_t} is a semantic consistency loss enforced by a noisy labeller for the target samples.

loss L_{seg} is a cross-entropy loss while the style-transferring loss L_{style} can be further expressed as:

$$\begin{aligned}L_{style}(G_A, G_B, D_A, D_B, I_s, I_t) = L_{adv}(G_A(I_s), I_t) + L_{adv}(G_B(I_t), I_s)+\\ L_{cycle}(G_B(G_A(I_s)), I_s) + L_{cycle}(G_A(G_B(I_t)), I_t)+\\ L_{sem}(f(G_A(I_s)), f(I_s), f(G_B(I_t)), f(I_t)),\end{aligned} \tag{4.5}$$

where G_B is a "target-to-source" generator, D_A and D_B are discriminators which assess the outputs from G_A and G_B respectively, L_{cycle} is a reconstruction loss represented by L_1-norm, discussed in Section

2.2, and L_{seg} is a cross-entropy semantic consistency loss calculated for the predicted labels (produced by the fixed noisy labeller f) labels of the original source and target samples and their style-translated versions.

The original paper [24] supposes using a feature-level adaptation in the segmentation model in the final stage of the model training part. This option was considered during the experiments in this project. However, the overall result of the model without feature-level adaptation at the final stage showed better performance and, therefore, the feature-level adaptation was not used. All other modifications are discussed in Section 4.6.

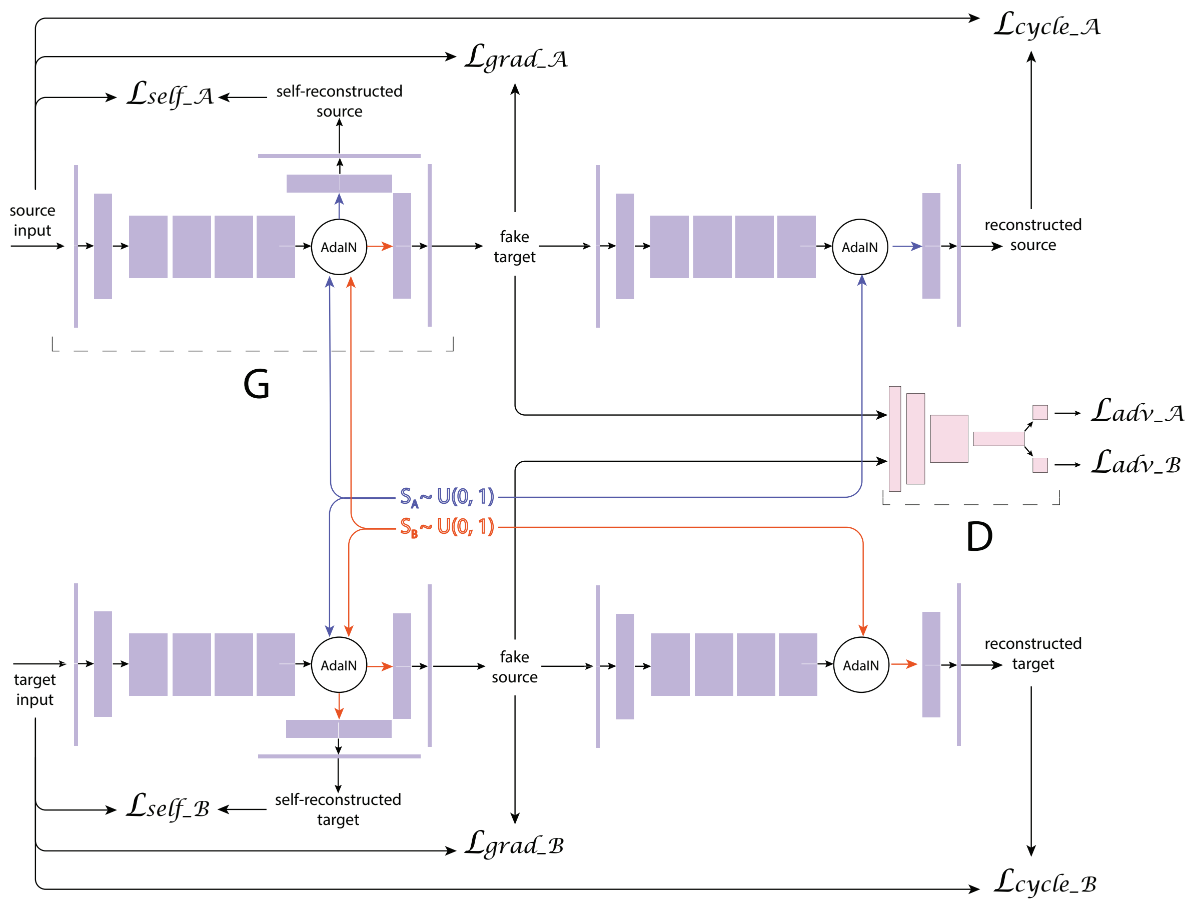
4.5 DAugNet

The data augmenter network (DAugNet) [15] is positioned as a style-transferring model with several advantages compared to the previously considered models. Firstly, it is a shallow network, therefore, it has fewer parameters and can be trained quickly. There is only one generator and one discriminator for both directions ("source-to-target" and "target-to-source") of sample translation. Secondly, there is significant attention to preserving semantic consistency during the style-transferring phase. Among others, the adaptive instance normalization layer (see Section 2.3.6) is employed to transfer style without changing the sample semantically. As can be seen from Fig. 4.6, the data augmentor (the part of the model responsible for style-transferring) consists of six convolutional blocks in the encoder part and two convolutional blocks in the decoder part. The four last blocks in the encoder are residual blocks. Since the original paper does not provide any implementation details for these residual blocks, those described in [64] where chosen as a final variant. The upsampling operation in the decoder is performed by employing UpSampling layers (see Section 2.3.4). The discriminator in this system is represented by a fully-convolutional discriminator described in [50] but with two domain-specific output layers so that it is able to perform an evaluation of both direction translations.

Before training the model, a unique 128-value style code is generated for each domain. This code represents the domain's mean and variance γ, σ by real numbers drawn from a uniform distribution in the range [0, 1]. These codes stay constant and do not change during the training. The style is translated in the AdaIN layer by combining the content embedding from each domain with the opposite style vector.

The final loss function of the training for the DAugNet can be expressed as:

$$L(I_s, I_t, Y_s) = L_{style}(G(I_s), I_t, S_A, S_B) + L_{seg}(G(I_s), Y_s), \quad (4.6)$$



$\mathcal{L}cycle\_\mathcal{A}$
$\mathcal{L}grad\_\mathcal{A}$
$\mathcal{L}self\_\mathcal{A}$
self-reconstructed source
source input
AdaIN
fake target
AdaIN
reconstructed source
G
$\mathcal{L}adv\_\mathcal{A}$
$\mathcal{L}adv\_\mathcal{B}$
S_A~ U(0, 1)
S_B~ U(0, 1)
D
target input
AdaIN
fake source
AdaIN
reconstructed target
$\mathcal{L}self\_\mathcal{B}$
self-reconstructed target
$\mathcal{L}grad\_\mathcal{B}$
$\mathcal{L}cycle\_\mathcal{B}$

Figure 4.6: Architecture of the DAugNet style-transferring component. Style transferring in both directions is performed by a single generator G with a single discriminator D. S_A and S_B represent 128-value style codes of mean and variance, L_{adv_A} and L_{adv_B} are adversarial losses of the discriminator, L_{self_A} and L_{self_B} are selfreconstruction losses, L_{grad_A} and L_{grad_B} are edge losses, and L_{cycle_A} and L_{cycle_B} are reconstruction losses for the source and target domains, where A assigned to source and B assigned to target.

where I_s and I_t represent the source and target input samples respectively, Y_s are corresponding labels for the source domain images, G is a common style-transferring model, and S_A and S_B are style codes for source and target domain, respectively. The segmentation loss L_{seg} is a cross-entropy loss while the style-transferring loss L_{style} can be further expressed as:

$$\begin{aligned}
&L_{style}(G, D, S_A, S_B, I_s, I_t) = L_{adv}(G(I_s), I_t, S_B) + L_{adv}(G(I_t), I_s, S_A)+\\
&L_{cycle}(G(G(I_s, S_A), S_B), I_s) + L_{cycle}(G(G(I_t, S_A), S_B), I_t)+\\
&L_{self}(G(I_s, S_A), I_s) + L_{self}(G(I_t, S_B), I_t)+\\
&L_{grad}(G(I_s, S_B), I_s) + L_{grad}(G(I_t, S_A), I_t),
\end{aligned} \tag{4.7}$$

where L_{adv} is an adversarial loss of the discriminator, L_{cycle} is a reconstruction loss represented by L_1-norm (discussed in Section 2.2), L_{self} is a self-reconstruction loss represented by L_1-norm, L_{grad} is a $L1$-norm loss enforcing semantic consistency for the edges of original and translated samples. The L_{self} is calculated for the input images A and B and their translated versions A' and B' obtained by combining the inputs with its own style code. L_{cycle} loss is calculated for the inputs A and B and their reconstructed versions A'' and B'', where A'' and B'' are the output of the “source-to-target-to-source" and “target-to-source-to-target" operation respectively. L_{grad} is a $L1$-norm calculated for the Sobel gradient [79] of inputs A and B and Sobel gradient of their translated versions $fakeB$ and $fakeA$.

4.6 Model Modifications

4.6.1 CycleGAN

The only modification that was made to the CycleGAN model used in this project was embedding additional self-reconstruction loss (or so-called identity loss). Using generators G_A and G_B (which translate source samples to target style and target samples to source style, respectively), this loss regularizes the generator's behaviour with respect to the input samples taken from the same domain as is the generator. The identity loss function can be expressed as:

$$L_{idt}(G_A, G_B, I_s, I_t) = ||G_A(I_s) - I_s||_1 + ||G_B(I_t) - I_t||_1, \tag{4.8}$$

where I_s are source samples and I_t are target samples. This loss was not included in the original CycleGAN loss function formulation but was considered in a separate chapter “Photo generation from paintings" of the original article [19]. The loss is used to preserve image tint where there was no need to change it. The identity loss was first proposed by Taigman *et al.* in [94].

Another modification that was made and tested with the CycleGAN model was replacing the backbone generator model with a UNet-256 [72] architecture. In this case, 256 means the number of filters in the last convolutional layer of the encoder; thus, UNet-256 is a slightly concise version of the original model. Because the UNet model presumes to have skip connections, the objects in the stylized images had sharper edges compared to the CycleGAN model with the nine ResNet [64] blocks as a backbone. The results are discussed in Section 6.5

4.6.2 CyCADA

Since CyCADA is based on CycleGAN, the same modification of the final objective function was made (see previous Section 4.6.1), and the same replacement of a backbone from convolutional blocks with nine ResNet blocks to UNet-256 was made. The version of CyCADA with the original nine ResNet blocks backbone was not considered in experiments because the UNet-256 version showed higher scores both for CycleGAN and CyCADA. Another change was in the alternation of the noisy labeller, discussed in Section 4.4. Instead of a segmentation model, a model with feature adaptation in the output space (same as the one discussed in Section 4.2) was employed as a noisy labeller. The idea here is that the original noisy labeller does not perform well on the target domain. Thus, its predictions for the target samples are not accurate. Using the AdaptSegNet model as a noisy labeller enforces the predicted labels for both domains to improve. In fact, this noisy labeller alternation increased the overall accuracy of CyCADA, which is discussed in Section 6.6

4.6.3 DAugNet

The DAugNet was improved by implementing the idea of Tasar *et al.* in [16], where they used skipped connections to enforce higher semantic consistency during training. The features extracted from the first convolutional layer are re-sized then concatenated to every deconvolutional layer in the decoder. This way, the decoder has a footprint of the objects taken from the real inputs. Therefore, it forces the decoder to keep the objects in the translated images in the right place. The results of this modification are discussed in Section 6.7.

Chapter 5

Implementation Details

This chapter discusses the implementation details of the project. First, the data preprocessing steps are explored for both domains. Afterwards, training details such as learning rate, batch size, type of optimizer, *etc.* are considered. Finally, system specifications of the training environment are provided.

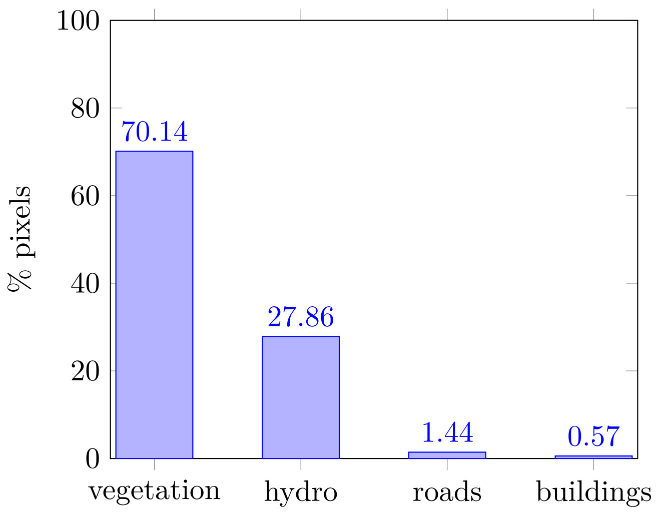
5.1 Data Preprocessing

This section describes the details of the considered training datasets. Then, the preprocessing steps for each domain are explained. Also, auxiliary statistics such as pixel value distribution and label distribution are provided.

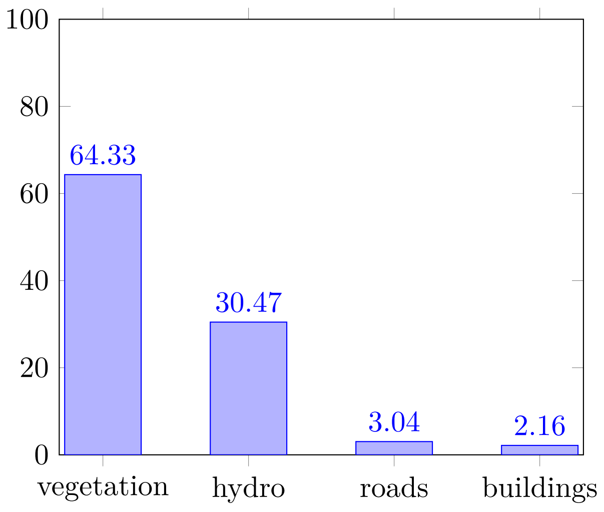
5.1.1 Source Domain

The domain with known labels (source) for this project is represented by the GeoEye-1 (GE1) [17] imagery dataset, which is a 2-meter spatial resolution multi-band 8-bit set of images taken in spring, summer and fall months across Canada. The dataset was provided by Natural Resources Canada [95]. It was initially downsampled from 1.84-meter spatial resolution and converted from 16-bit radiometric resolution to 8-bit representation. The GeoTIFF [96] raster image and corresponding label shapefiles were cropped into samples of size 4 x 256 x 256 and 1 x 256 x 256, respectively, where 4 represents the number of bands (blue, green, red, and near-infrared) in the image file and 1 represent a single band of the corresponding label file which was rasterized and saved in GeoTIFF format.

The labels available for the source domain are vegetation (include only forested regions), hydro, roads, buildings, and background. The annotation of this dataset was conducted automatically; thus, labels do not always perfectly match the objects. However, the quality of the annotation is enough for the baseline model (described in Section 4.1) to achieve high-quality predictions on the validation subset of the source dataset. The results of the oracle source model are provided in Section 6.2. The oracle model is a model trained and validated on the samples from the same domain, *i.e.* this is a model that shows potential segmentation performance for the dataset. Another feature of the dataset is that it is highly unbalanced. As can be seen from Fig. 5.1a, the class buildings is highly underrepresented compared to other classes. To overcome this issue, the dataset was augmented using the following steps. First, each label file of the dataset is opened, and a number of building pixels is calculated. If the number of building pixels in the current sample is non-zero, the sample is rotated 90, 180, 270 degrees, then flipped horizontally and vertically. The modified versions of the sample are then saved as separate files. After these transformations, the building class increased its population that is seen in Fig. 5.1b.



(a) GeoEye-1 (original) label distribution

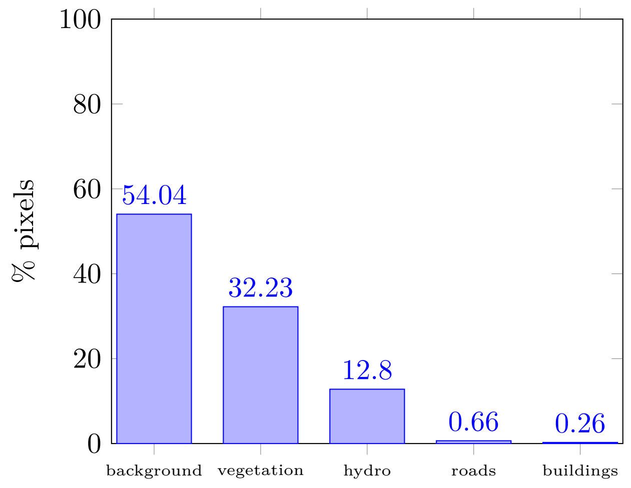


(b) GeoEye-1 (augmented) label distribution

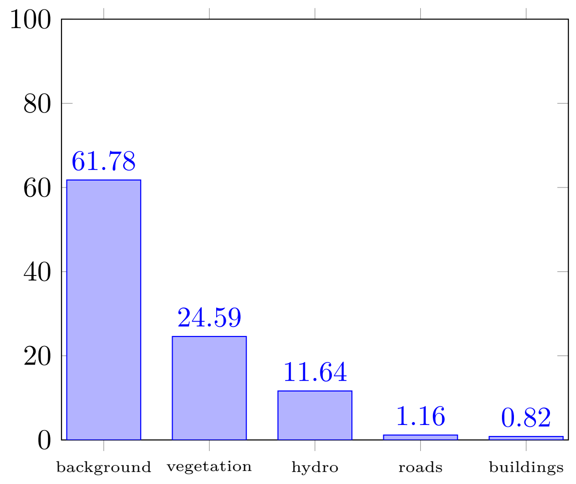
Figure 5.1: Label distribution in source dataset before and after augmentation.

The background class prevails over any other classes, but its importance is not significant. Therefore, it was not shown in Fig. 5.1 to clearly see the difference in distribution among more significant classes. However, the original class distribution and the one of the augmented dataset with the background class included are provided in Fig. 5.2.

Another issue with the source dataset was in its border samples. The label layer does not perfectly match the corresponding image layer. Thus, the areas of the image layer where labels are absent must be masked as "nodata". "Nodata" is a specific class in remote sensing where no information for the current pixel is available. Further, during the model training, the area of "nodata" pixels is ignored by a loss function and therefore, the potential error of computations decreases. The example of this issue



(a) GeoEye-1 (original) label distribution



(b) GeoEye-1 (augmented) label distribution

Figure 5.2: Label distribution in source dataset before and after augmentation including background class.

and the proposed solution is depicted in Fig. 5.3.

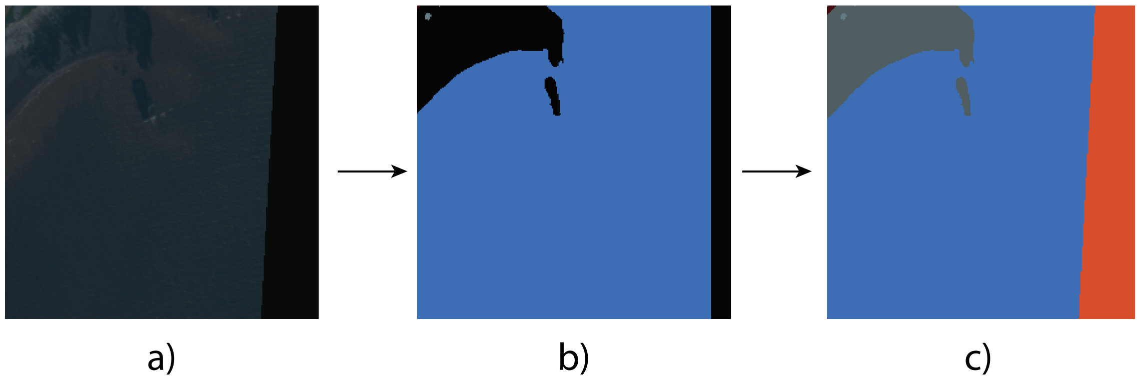


Figure 5.3: This figure shows how the border issue with "nodata" pixels was resolved. a) is an example of an image sample and b) its original label. Black areas in a) and b) represent "nodata" pixels. As can be seen, they do not match. In c) the fixed label where "nodata" area is in red and matches a), and a grey area is the background class.

The total number of the source samples is 8108. For the oracle model (*i.e.* the model that is trained on source data and validated on the source data), the dataset was split into training and validation subsets with the number of samples 6975 and 1133, respectively. Each sample was randomly chosen; thus, the label distribution in both subsets is roughly equal. After all the transformations and augmentation are complete, the final distribution of pixel values for the entire dataset is available as a histogram and depicted in Fig. 5.4.

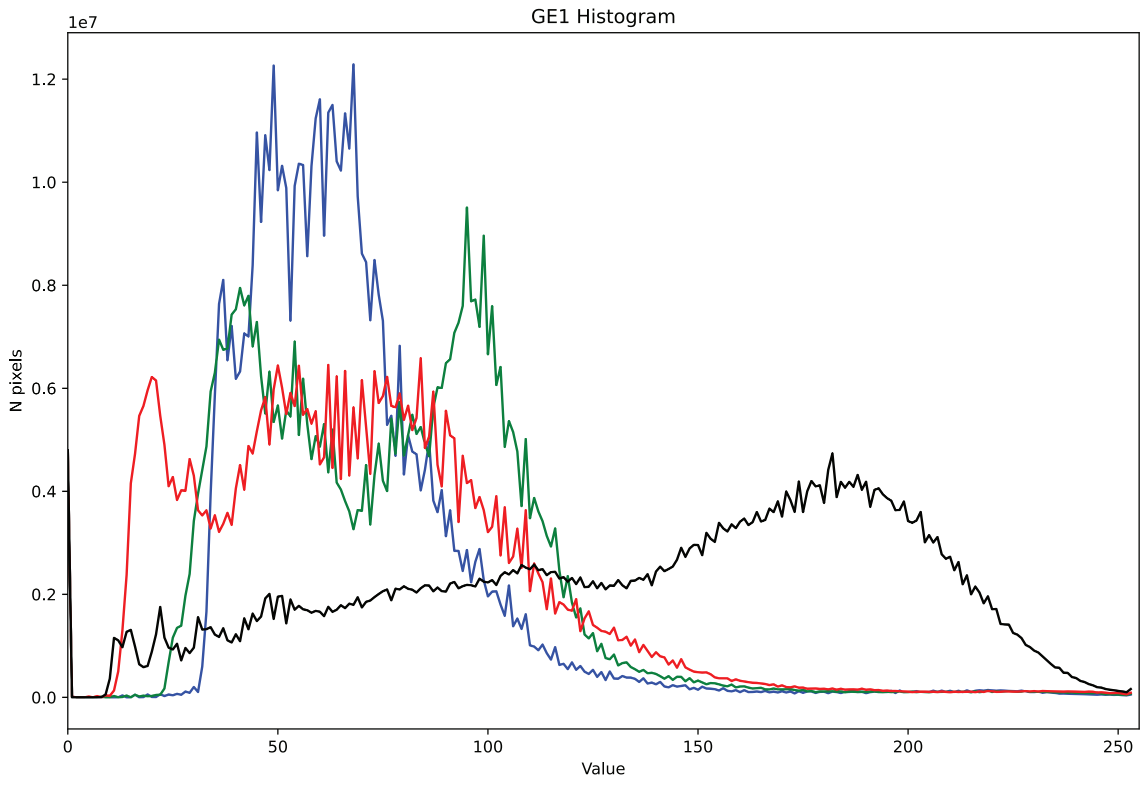


Figure 5.4: The histogram of pixel value distribution of the source domain dataset. The blue, red, green, and black lines represent a number of pixel values in the blue, red, green, and near IR channels of the entire dataset, respectively.

5.1.2 Target Domain

As a target domain, or the where labels do not participate in training the models and which style must be transferred to the source images, the WorldView-3 (WV3) [18] imagery dataset was used. This dataset is a 0.31-meter (pansharpened) 16-bit set of images taken in an unknown season (presumably, summer) over a desert area of the African continent. The dataset is publicly available on the Kaggle [97] competition website and was initially generated by the Defence Science and Technology Laboratory [98]. The original 16-bit radiometric resolution was converted to an 8-bit format. The satellite images in the dataset are not GeoTIFFs; thus, it is not possible to determine over which area they were taken. The following preprocessing steps were made before cropping the original satellite images and corresponding labels for the training dataset. First, the low-resolution (1.24m) colour bands of the images are pansharpened [99] using the corresponding high-resolution (0.31m) grayscale panchromatic band. Second, the labels are generated from the provided well-known text (WKT) [100] files and resized to match the pansharpened image size. Finally, the labels are rasterized and saved in multi-band TIFF format. After that, the images were cropped to samples of size 4 x 256 x 256, where 4 represents the number of bands (blue, green, red, and near-infrared) - same as for the source domain. The labels, in turn, were cropped to samples of size 1 x 256 x 256.

The target domain labels are different from those available for the source domain. It is buildings, miscellaneous manmade structures, roads, tracks (poor/dirt/cart track, footpath/trail), trees (woodland, hedgerows, groups of trees, standalone trees), crops (contour ploughing/cropland, grain (wheat) crops, row (potatoes, turnips) crops), waterways, standing water, large vehicles, and small vehicles. In order to bring them all to match the source dataset, the following label merging was performed and is shown in Table 5.1.

Label Number	GeoEye-1	WorldView-3
0	background	background, crops, vehicles large, vehicles small
1	vegetation	trees
2	hydro	waterways, standing water
3	roads	roads, tracks
4	buildings	buildings, miscellaneous manmade structures

Table 5.1: Labels available for both datasets. Several labels in the WV-3 dataset were merged to match source annotations.

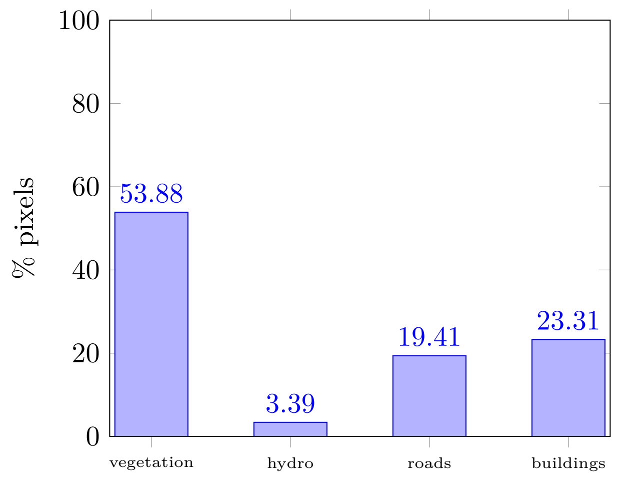
The total number of the target samples is 6400. For the oracle model (*i.e.* the model that is trained on target data and validated on target data), the samples were split into 5507 and 893 samples for the training and validation subsets, respectively. The final label distribution of the entire target dataset is depicted in Fig. 5.5. The pixel distribution for the target domain is depicted in Fig. 5.6.

5.2 Training Details

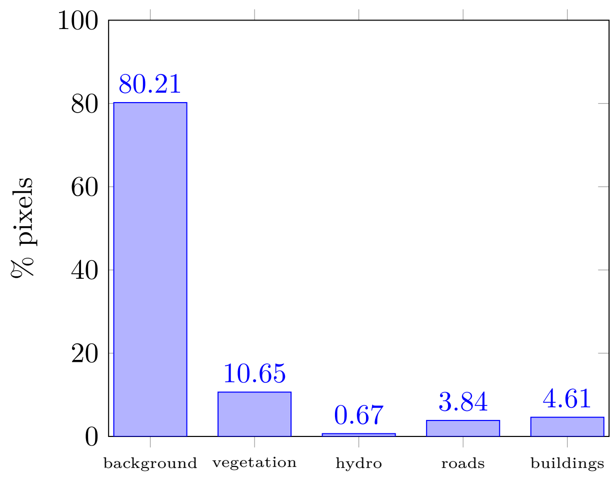
This section provides details on the parameters of each model training. The parameters include an optimizer algorithm chosen for each model, a value of a learning rate, specific training steps, *etc.*

5.2.1 Baseline Model

The baseline model was trained on the source and validated on target samples. The idea was to see how a segmentation model without adversarial learning can perform on the unseen data. Also, the results of



(a) WorldView-3 label distribution without background class.



(b) WorldView-3 label distribution including background class.

Figure 5.5: Label distribution in target dataset with and without background class.

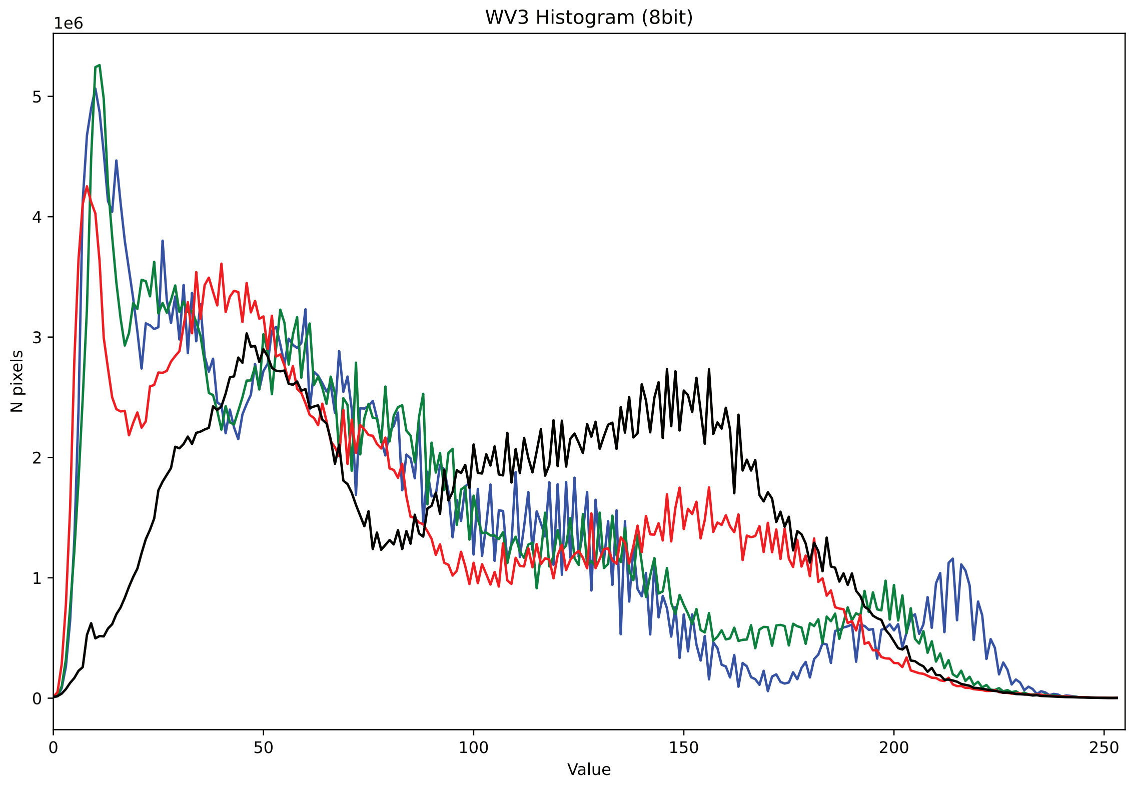


Figure 5.6: The histogram of pixel value distribution of the target domain dataset. The blue, red, green, and black lines represent the number of pixel values in the blue, red, green, and near-infrared channels of the entire dataset, respectively.

this experiment were compared with the results of the models using unsupervised domain adaptation techniques. As an optimizer (discussed in Section 2.2.2), the Adam method was chosen. The initial learning rate and weight decay were set to 1×10^{-4} and 5×10^{-4}, respectively. During training, the learning rate was decreased using polynomial decay with a power of 0.9 as mentioned in [5] with the formula:

$$lr_c = lr_b \times \left(1 - \frac{iter_c}{iter_m}\right)^p, \tag{5.1}$$

where lr_c is the current learning rate, $iter_c$ is a current training iteration, $iter_m$ is a maximum number of training iterations, and p is a decaying power. The model was trained with 16 images in a batch, over 125000 steps. No discriminator network was used here because adversarial learning was intentionally not included in this baseline experiment. The input images from both domains were resized to 256x256 pixels and normalized from -1 to 1. The model was initialized with ImageNet [74] weights.

5.2.2 AdaptSegNet

To train AdaptSegNet, three optimizers were used. The first optimizer was assigned to the generator part of the model, and two other were assigned to both discriminators of the model. For the generator, the Stochastic Gradient Descent (SGD) optimizer with Nesterov acceleration was used, where momentum is 0.9 and the weight decay is 5×10^{-4}. The initial learning rate is set as 2.5×10^{-4} and is decreased using polynomial decay with a power of 0.9. The batch size while training was 16, and 250000 training steps were committed. Each of the two discriminator networks was trained using an Adam optimizer with the initial learning rate as 10^{-4} and the same polynomial decay as the generator network. Such parameters of the Adam optimizer as $beta1$ and $beta2$ are set as 0.9 and 0.99, respectively. The input images were resized to 256x256 pixels and normalized from -1 to 1. The model was initialized with ImageNet weights. After the model was trained, the discriminator part was discarded, and the outputs from the farthest (the one that follows the last encoder convolutional block) decoder are taken. To control the influence of adversarial losses calculated for both discriminators, λ_1 was set to 0.0002, and λ_2 was set to 0.001, where λ_1 and λ_2 are coefficients that multiplies the losses of the discriminator in the middle part of the generator and at the end of it respectively.

5.2.3 CycleGAN

The CycleGAN training consists of two parts - style transferring and semantic segmentation. During the first training part, the following settings were used. One Adam optimizer for both generators was used with parameters $beta1$ equal to 0.5 and $beta2$ equal to 0.999. The initial learning rate was set to 2×10^{-4}

and linearly decayed over training epochs starting from 10. The total number of training was set to 35 (or 141890 steps) with a batch size of 2 images. An Adam optimizer optimized both discriminators with the same hyperparameters as the generators. In order to control the contribution of each loss in the style-transferring training phase, the following coefficients were assigned to each component of the loss. Both adversarial losses had coefficient 1, cycle consistency losses were multiplied by coefficient 10, and identity (self-reconstruction) losses were multiplied by 2. The second part of the training was conducted having the same training parameters discussed in Section 5.2.1. The only difference is that the source images with target style (translated by the “source-to-target" style generator) were used as a training dataset. After that, the model was validated on the target validation dataset.

5.2.4 CyCADA

The CyCADA model training consisted of three steps. First, a noisy labeller was trained on source data to generate latent labels for the target dataset. For this project, the noisy labeller is the AdaptSegNet model, which was trained with the same trained parameters discussed in Section 5.2.2. With the noisy labeller trained, the second style-transferring phase was as follows. The CyCADA style-transferring phase has the same training steps as the CycleGAN model, with training parameters discussed in Section 5.2.3. To control the losses' influence in the final objective of the model, the following coefficients were chosen. Both adversarial losses had coefficient 1, cycle consistency losses were multiplied by coefficient 10, and identity (self-reconstruction) losses were multiplied by 2. The semantic consistency loss had to be accurately fine-tuned because it significantly changed the result of the style translation. Making it higher than 2, resulted in the semantic consistency loss overpowering all other losses, leading to the input source sample not acquiring the style of the target domain (leaving the source sample unchanged). Making this loss lower than 0.25 led to significant artifacts on the translated objects and high semantic inconsistency. Thus, the semantic consistency loss coefficient in the range [0.5, 1.5] showed an optimum performance value during the experiments. The third part of the training is the same as for the baseline model and discussed in Section 5.2.1. The inferences from the “source-to-target" style generator were used as a training dataset, and the results were validated on the target validation dataset.

5.2.5 DAugNet

The DAugNet model was trained in two phases. The first phase was style translation, and the second one was semantic segmentation of the translated source samples. The generator of the model was optimized by the Adam optimizer with parameters $beta1$ and $beta2$ equal to 0.5 and 0.999, respectively.

The initial learning rate was equal to 10^{-4} and was decayed over training steps with the following formula:

$$lr_c = lr_b \times \frac{iter_m - iter_c}{iter_m - iter_d}, \tag{5.2}$$

where lr_c is the current learning rate, $iter_c$ is the current training iteration, $iter_m$ is the maximum number of training iterations, and $iter_d$ is the number of iterations where decaying starts. The total number of training steps was 13000, and the weight decaying step was 7000. The discriminator of the model was optimized using the Adam optimizer with the same parameters as were used for the generator's optimizer. Since the style-transferring part of the objective function has many composing losses, the following coefficients were assigned to each of them. The adversarial losses for the generator (in both directions) are multiplied by 1, the cross-reconstruction loss is multiplied by 5, the self reconstruction loss is multiplied by 5, and the edge loss is multiplied by 10. The parameters of the semantic segmentation model in the last part are the same as discussed in 5.2.1. The only difference is that the source images in the target-like style are used as training data. Afterwards, the model is validated on the target dataset.

5.3 System Specifications

The models were implemented in Python version 3.7.6 in PyTorch framework version 1.8.1 with CUDA 10.1 support. All experiments were conducted on a single NVIDIA RTX A6000 GPU with 48GB of memory, running in the Docker v20.10.8 container with Ubuntu distribution v20.04.3 LTS. Also, the main system specifications include AMD Ryzen Threadripper 3960X 24-Core Processor, 258GB of system memory, and 16.8TB of total storage space.

Chapter 6

Results and Analysis

In this chapter, the thesis results are discussed both quantitatively and qualitatively. First, the evaluation metrics used are described, then the results table is provided. Afterwards, each model's performance is evaluated and compared. Finally, the samples of each model prediction are provided and evaluated.

6.1 Evaluation Metrics

All experiments in this project use mean intersection over union (mIoU) as a validation measure. Mean IoU is widely used in object detection tasks. Consider two segmentations for an object within an image: one the ground truth and one the prediction of a model. Given these two segments, we can find the intersection and union as well as calculate the ratio of pixels in the intersection to those in the union. Then, the mIoU is the average of all these ratios for each class. Assume, p_{ij} is a number of pixels where the ground-truth label is i, and the prediction label is j. A formula for the mIoU can be defined as follows:

$$mIoU = \frac{1}{m}\sum_{i=1}^{m}\frac{p_{ii}}{\sum_{j=1}^{m} p_{ij} + \sum_{j=1}^{m} p_{ji} - p_{ii}}, \tag{6.1}$$

where m is the number of classes in the segmentation task, p_{ii} is the number of correctly predicted pixels, $\sum_{j=1}^{m} p_{ij}$ is the total number of pixels labelled as i, and $\sum_{j=1}^{m} p_{ji}$ is the total number of incorrectly predicted pixels.

Another important metric used in this project is per-class IoU, which reflects the model's performance on each class of the segmentation task. It is crucial because mIoU shows the average performance over all labels, but in reality, some classes like background can represent more than 80 percent of the dataset, which dominate the calculation of the average regardless. The per-class IoU can be acquired from the mIoU formula and expressed as:

$$IoU_{per-class} = \frac{p_c}{\sum_{j=1}^{m} p_{cj} + \sum_{j=1}^{m} p_{jc} - p_c}, \tag{6.2}$$

where p_c is the number of correctly predicted pixels for the given class c, $\sum_{j=1}^{m} p_{cj}$ is the total number of pixels labelled as c, and $\sum_{j=1}^{m} p_{ji}$ is the total number of incorrectly labelled pixels for the given class c.

6.2 Table of Results

The results of the project are composed into a single Table 6.1. The results from the two oracle models are presented first. Recall, an oracle model is one that is trained and validated on the same domain dataset. Then, the result of the baseline model (a segmentation model without adversarial learning trained on the source and validated on target dataset), followed by all the results from the proposed models. The sections that follow discuss each model's performance and compare it with other models.

Model	mIoU	background	vegetation	hydro	roads	buildings
Oracle source	81.12	93.56	90.0	93.14	60.71	68.22
Oracle target	75.75	92.23	69.98	84.12	67.59	64.88
No Adaptation						
Baseline	25.11	76.57	19.93	14.75	0.25	14.03
Unsupervised Domain Adaptation						
AdaptSegNet	26.33	72.19	15.31	43.64	0.52	0.0
CycleGAN (resnet9)	24.41	72.19	20.72	6.63	1.63	17.58
CycleGAN (unet256)	25.16	65.11	20.19	3.01	3.87	33.63
CyCADA (semloss=0.5)	26.82	74.38	14.7	4.61	4.07	36.35
CyCADA (semloss=1.0)	26.2	69.46	19.86	4.66	2.91	34.68
CyCADA (semloss=1.5)	26.89	72.26	19.26	6.6	4.09	32.26
CyCADA+modif (semloss=0.5)	31.44	74.64	17.36	24.99	2.15	38.05
CyCADA+modif (semloss=1.0)	31.15	72.66	16.71	25.16	3.83	37.42
CyCADA+modif (semloss=1.5)	32.6	80.65	17.12	30.71	0.66	33.89
DAugNet	23.78	63.98	19.23	9.5	1.68	24.52
DAugNet+modif	25.69	67.06	22.08	11.25	2.3	25.77

Table 6.1: The table of results for each model. CyCADA+modif refers to the modification of the noisy labeller (discussed in Section 4.6.2), and semloss refers to the semantic consistency loss (discussed in Section 5.2.4). DAugNet+modif refers to the modification of the DAugNet model discussed in Section 4.6.3. resnet9 and unet256 in the CycleGAN results refer to the different backbone architectures and discussed in Section 4.6.1.

6.3 Baseline Model

The baseline model, considered in Section 4.1, showed a poor overall result. The mean IoU for all five classes is 25.11%, a comparable result for all the conducted experiments. This is due to the prevailing background class, which reached 75.19% of per-class IoU, vegetation class that showed 19.93%, and hydro class that showed 14.75% IoU. The latter two are slightly above average among all the models. However, the roads and buildings performed worse than almost all the models with domain adaptation. The roads class reached only 0.16% IoU, which means this class was completely invisible for the model without adaptation. The buildings class showed a better performance of 14.03% IoU, but still, it is only better than the AdaptSegNet's result for this class. Fig. 6.1 shows the normalized confusion matrix for the baseline model.

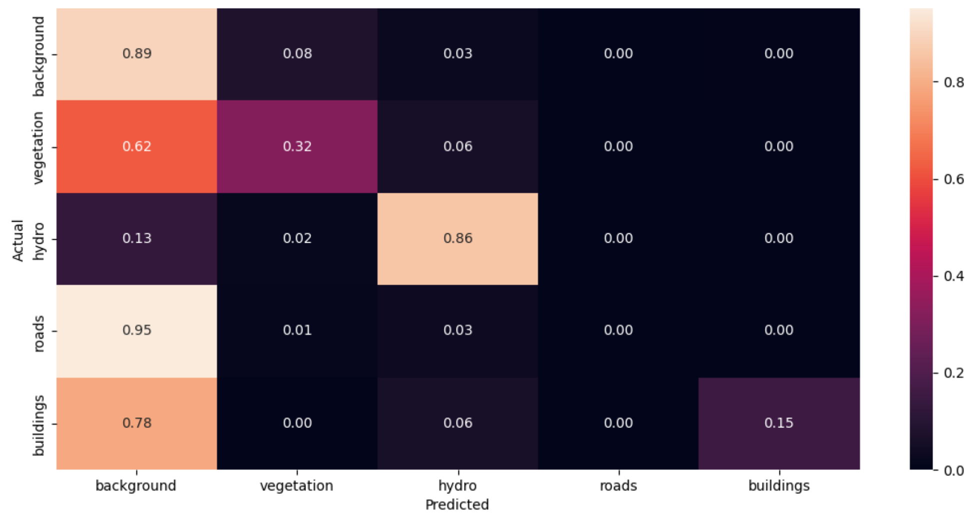


Figure 6.1: Normalized confusion matrix of the WV-3 validation dataset for the baseline model.

6.4 AdaptSegmNet

AdaptSegNet mIoU performance is 1.22% better than the baseline model and reached 26.33%. However, overall, this result is an average of all the experiments. The background class showed 72.19% IoU, while the vegetation class performed slightly worse (15.31% IoU) than average among all the models, including the baseline model for this class. The most significant score was the hydro class - 43.64% IoU, the highest performance among all the models for this class. This can be explained by the fact that the features representing the structure of water bodies in the output space for both models can be efficiently aligned. The same cannot be said about the roads and buildings classes. The model could not generalize the representation of these classes in the output space and ultimately failed the classification task. Roads and buildings showed almost zero scores - 0.52% and 0.0% IoU, respectively. Fig. 6.2 shows the normalized confusion matrix for the AdaptSegNet model.

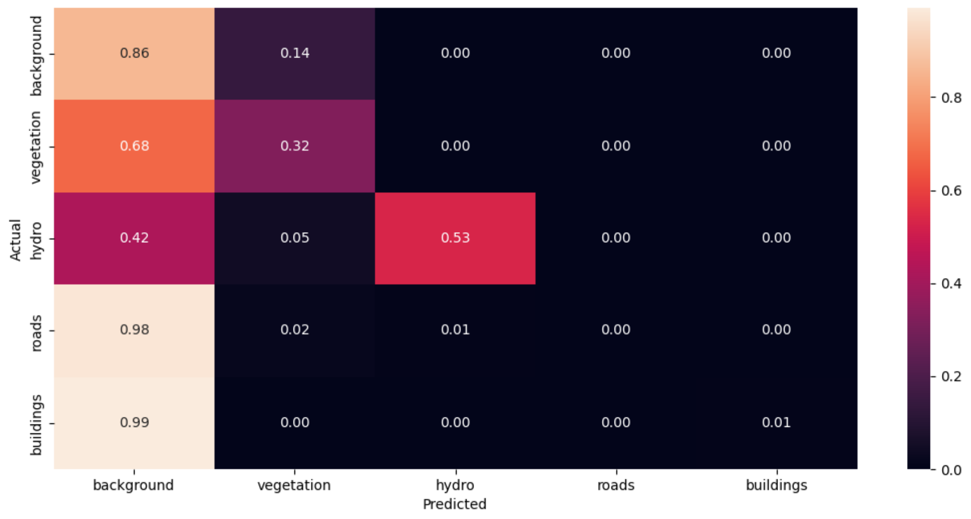


Figure 6.2: Normalized confusion matrix of the WV-3 validation dataset for the AdaptSegNet model.

6.5 CycleGAN

The CycleGAN with ResNet blocks as a backbone showed the lowest mean IoU result (24.41%) among all the trained models, including the baseline model. The background class reached 72.19% IoU, which is close to the bottom border of all results. The vegetation class, however, performed relatively well are reached 20.72% IoU. Such classes as hydro, roads, and buildings performed weakly compared to other models with adaptation (except AdaptSegNet, where the hydro class showed the top performance).

The CycleGAN with unet256 structure as a backbone showed better overall performance than the CycleGAN with ResNet blocks. The mean IoU increased by 0.75% and reached 25.16% IoU. Class buildings doubled their performance and reached 33.63% IoU - this is average among all the style-transferring models. Class roads have also doubled and reached 3.87% IoU but is still far from accurate overall. However, the background and hydro classes decreased the performance and showed 65.11% and

3.01% IoU, respectively. Both results are the lowest among all the models, including the baseline model. This is mainly because the transferred source images are semantically inconsistent and full of artifacts, which led to high misclassification during the segmentation phase of the training.

Since the main part of the CycleGAN training is style transferring, it is important to ensure that it is performed successfully. Fig. 6.3 shows how the source domain pixel value distribution transformed to match the target domain pixel value distribution. This confirms that the adversarial task in the style-transferring training part was completed.

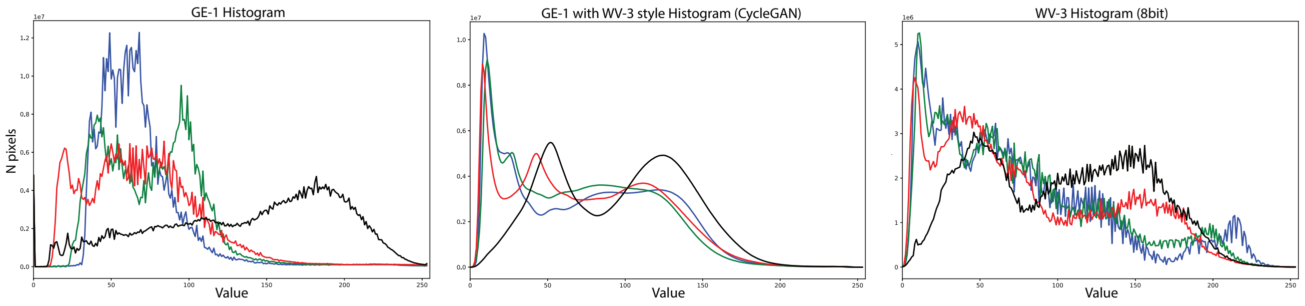


Figure 6.3: Example of the style-transferring operation of the CycleGAN model with unet256 backbone. The left picture represents the original source domain pixel value distribution. In the middle is the pixel value distribution in the source domain with the target style. The right picture represents the original target domain pixel value distribution.

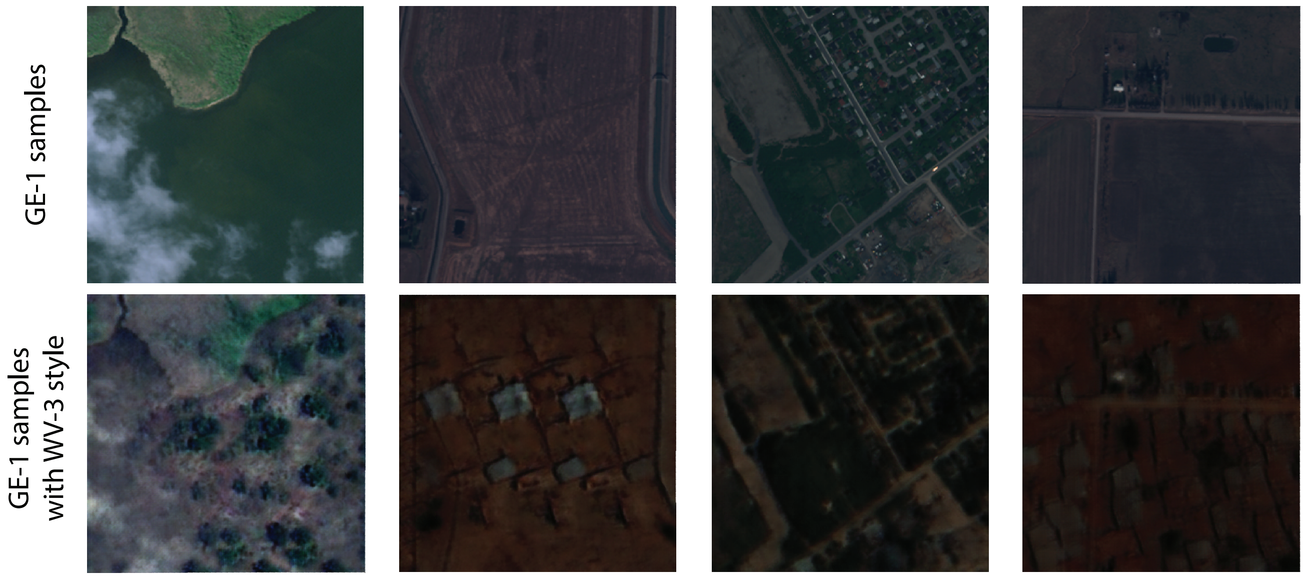


Figure 6.4: Figure depicting samples of the initial GE-1 dataset and their WV-3 stylized version (CycleGAN model with unet256 as a backbone).

However, as can be seen from Fig. 6.4, there are many artifacts in the translated images. For example, the water area is populated with trees, and the field is populated with buildings. This is because the CycleGAN model does not preserve semantic consistency while transferring style. These artifacts create additional obstacles for the segmentation model, that is trained afterward. Fig. 6.5 shows the normalized confusion matrix for the CycleGAN model.

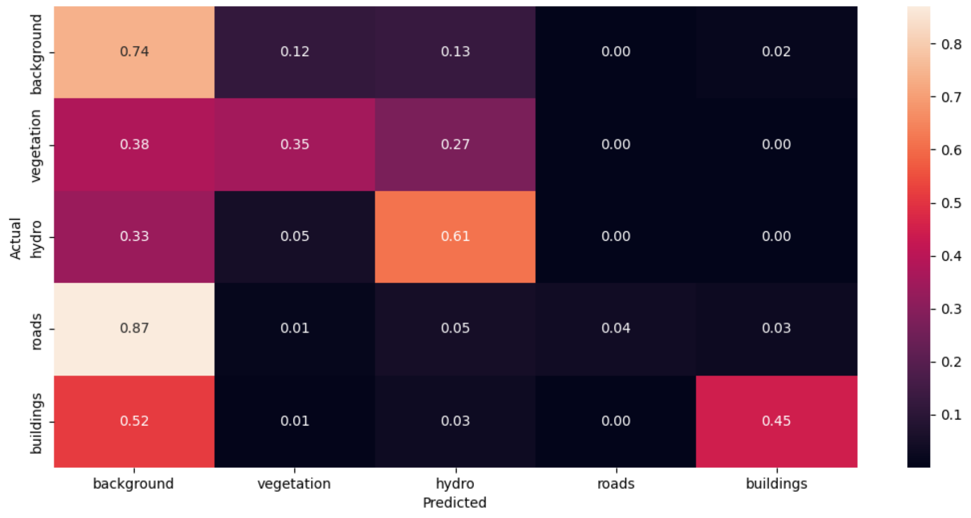


Figure 6.5: Normalized confusion matrix of the WV-3 validation dataset for the CycleGAN model.

6.6 CyCADA

There are more results for the CyCADA model because of the variability of possible combinations in the architecture and the fine-tuning parameters. The experiments were mostly focused on finding the optimum value for the semantic consistency loss coefficient. Also, the results with the proposed adapted noisy labeller are considered.

The CyCADA model with original noisy labeller [24] produces a mean IoU performance of 26.8% and 29.89% for the experiments where semantic consistency loss (semloss) equalled 0.5 and 1.5, respectively. These results are in the upper border of all the experiments and overall better than the CycleGAN and baseline models. Having semloss equal 1, however, significantly decreases the model's performance in the background and roads classes. The vegetation class showed close to the top performance of 19.86% and 19.26% IoU when semloss was equal to 1.0 and 1.5, respectively. The hydro class performed worse than all the models (except CycleGAN), including the baseline model. However, the roads class showed the top IoU score among all the considered models, with 4.07% and 4.09% for the semloss equal 0.5 and

1.5, respectively. The model's performance for the buildings class is relatively good (36.35%, 34.68%, and 32.25% IoU) for all considered semloss values (0.5, 1.0, and 1.5).

Switching the original noisy labeller module to the adapted one allowed a significant increase to both the overall mIoU and per-class performance of the CyCADA model. The mean IoU increased by 5-6% overall compared to all other models, including CyCADA with the original labeller. The maximum mIoU reached was 32.6% mIoU with semloss equal 1.5. The background class also showed top performance among all the models - 80.65% IoU with semloss equal 1.5. However, the vegetation and road classes showed average performance compared to other models. Instead, the hydro class showed top performance among the adaptation models with style-transferring. The highest value of IoU for this class was 30.71% with semloss equal 1.5. The building class also showed performance among all the models - 38.05% IoU with the semloss equal 0.5. CyCADA with modified noisy labeller generally showed the top scores among all other models.

The main part of CyCADA model training is style transferring. An example of it is depicted in Fig. 6.6. As can be seen from the translated pixel value distribution, the adversarial training shifted the pixel value distribution in each band. Compared to the CycleGAN model, the translated version of the source images has fewer artifacts, thus, improving the result of the following segmentation task. Fig. 6.8 shows the normalized confusion matrix for the CyCADA model.

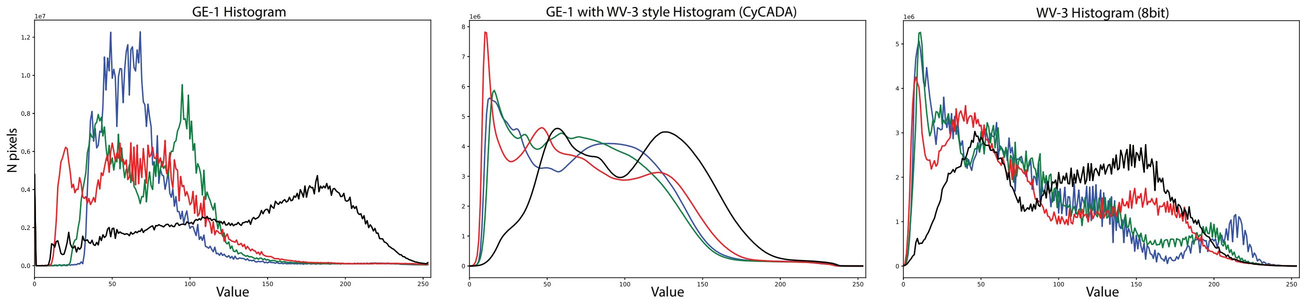


Figure 6.6: Example of the style-transferring operation of the CyCADA model with the original noisy labeller and semantic loss coefficient equal 0.5. The left picture represents the original source domain pixel value distribution. In the middle is the pixel value distribution in the source domain with the target style. The right picture represents the original target domain pixel value distribution.

6.7 DAugNet

The DAugNet model results are represented by two experiments. First, the original model [15] results are considered. Then, the model results with proposed skip connections are discussed.

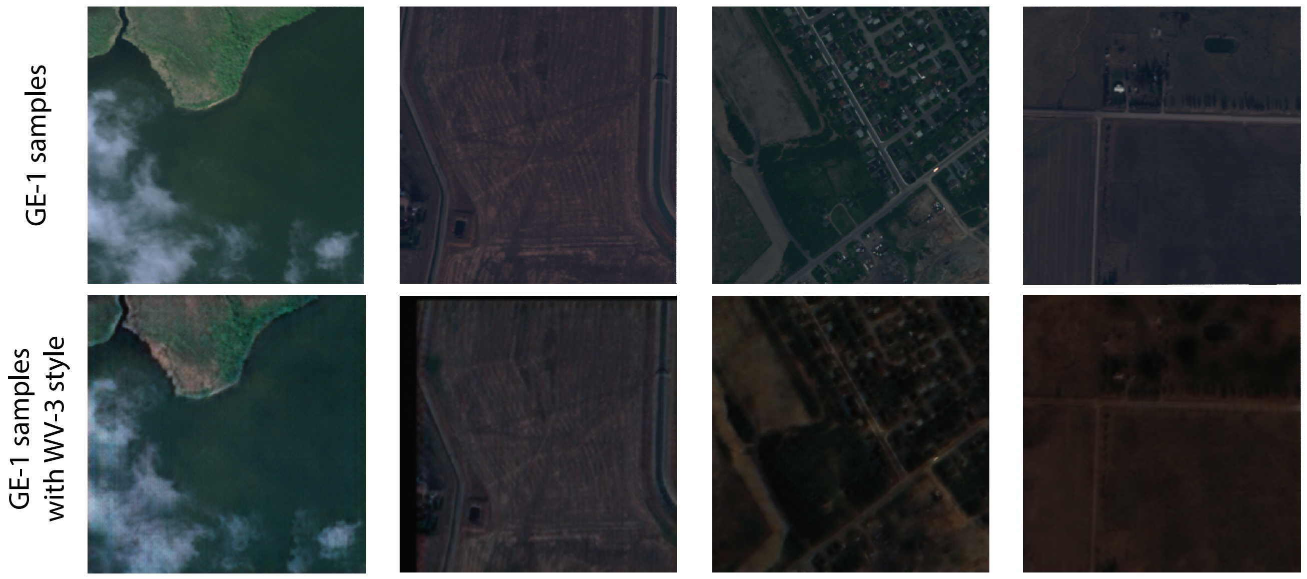


Figure 6.7: Figure depicting samples of the initial GE-1 dataset and their WV-3 stylized version (CyCADA model trained with *semloss* = 1.5 and unet256 as a backbone).

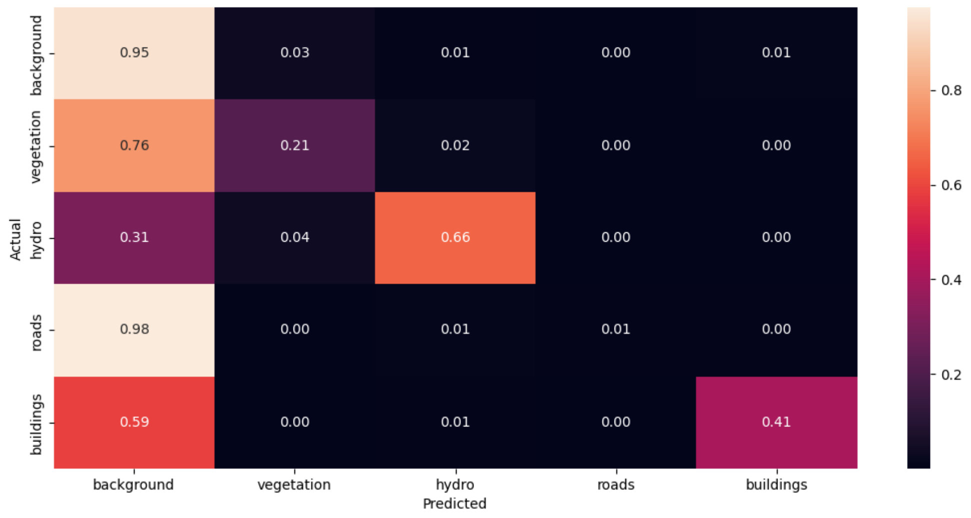


Figure 6.8: Normalized confusion matrix of the WV-3 validation dataset for the CyCADA model.

The original DAugNet model showed a relatively poor mean IoU score - 23.78%. This is mainly because such classes as background (63.98%) and hydro (9.5%) showed close to lower border per-class IoU performance. The road class showed average results among all the models - 1.68% IoU, and the buildings class (24.52%) performed better than the baseline and AdaptSegNet models.

Adding skip connections to the original DAugNet model increased the model mIoU performance by 1.91%. The background class result increased from 63.98% to 67.06%; the vegetation class reached the top score of 22.08% IoU among all the models. All other classes also slightly increased within 2% IoU.

The DAugNet model training consists of two parts - style transferring and semantic segmentation. The success of the first part defines the success of the consequent part and the full training in general. In Fig. 6.9 it is shown that the style of the target domain was transferred to the source domain samples. The translated source sample distribution is similar to the target domain pixel value distribution. Visually, the artifacts or some colour variation are present over the waterbodies. This is because the water surface in the training samples has a green tone, similar to the crops areas, which are considered as background (see the leftmost sample in Fig. 6.10). The land part of the images does not seem to have significant artifacts. Fig. 6.11 shows the normalized confusion matrix for the DAugNet model.

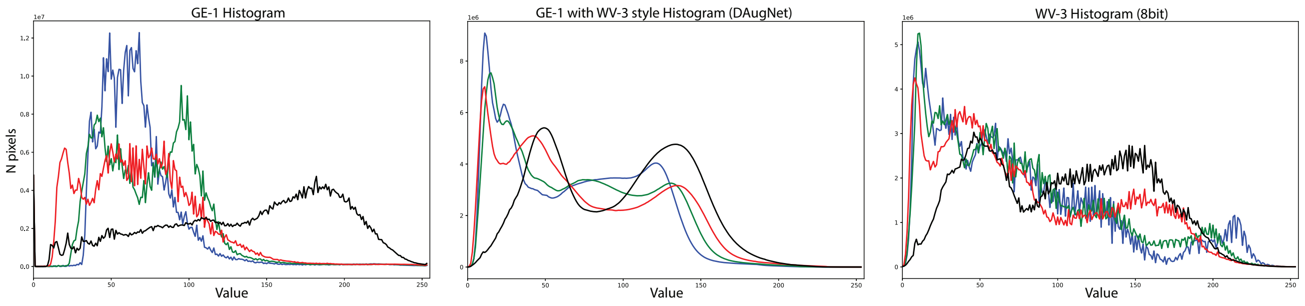


Figure 6.9: Example of the style-transferring operation of the DAugNet model with skip connections. The left picture represents original source domain pixel value distribution. In the middle is pixel value distribution in the source domain with the target style. The right picture represents the original target domain pixel value distribution.

6.8 Samples Results from the Experiments

In this section, the results of the final segmentation part of each model are shown. As can be seen from Fig. 6.12, the baseline model was able to somewhat detect hydro and vegetation classes; however, it also misclassified hydro with the background. Buildings and roads were almost invisible for this model.

The AdaptSegNet seems to generate only two classes as the output - hydro and background. The hydro class was detected relatively accurately according to the samples. The background class replaced all other classes.

The CycleGAN model started to “see" buildings with little accuracy. It also misclassified vegetation class a lot. From the second sample from the bottom, the tree shadow is classified as water, which is

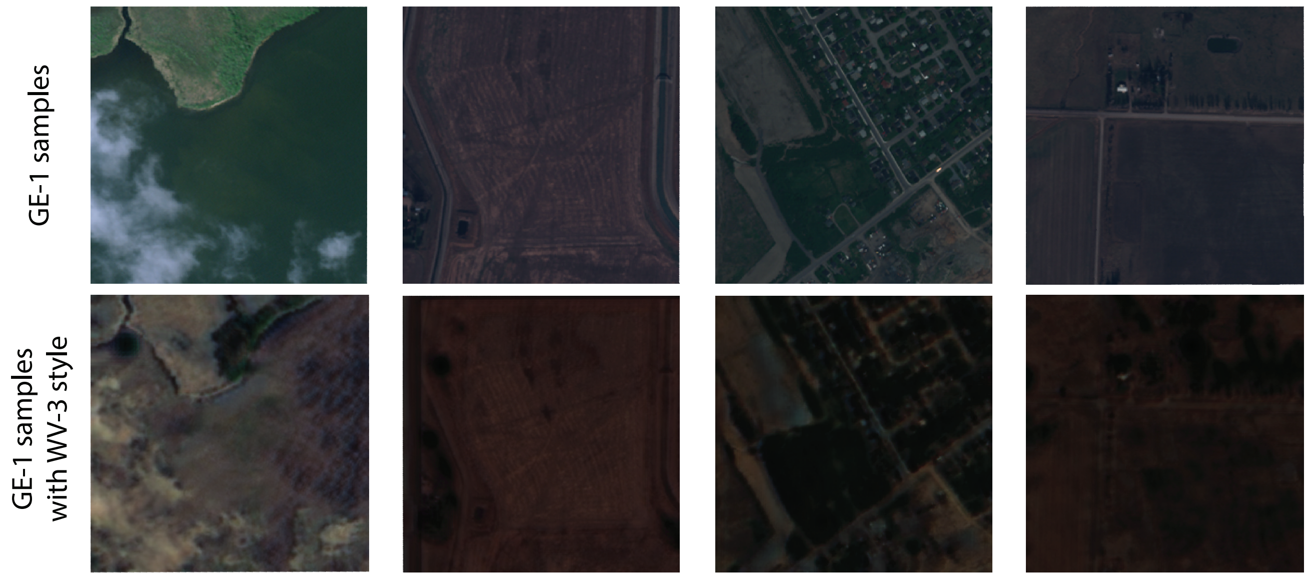


Figure 6.10: Figure depicting samples of the initial GE-1 dataset and their WV-3 stylized version (DAugNet model with skip connections).

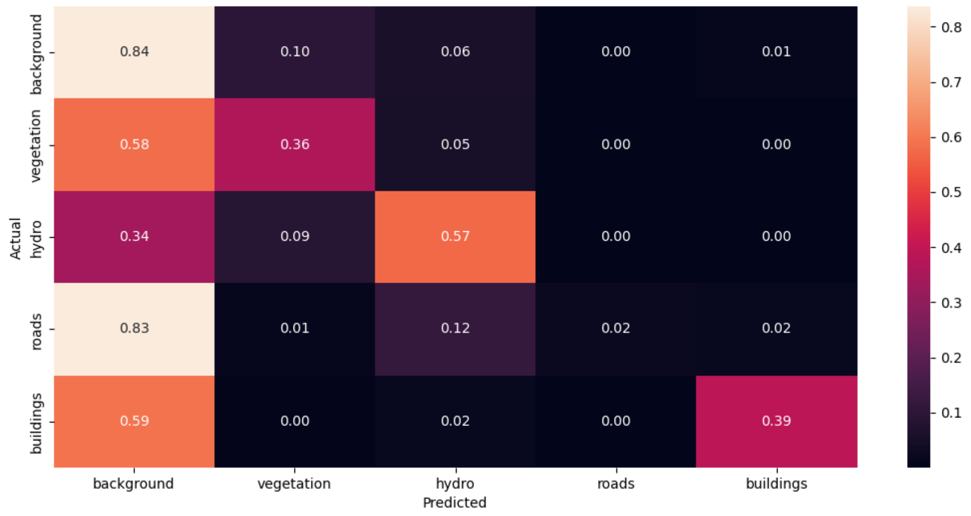


Figure 6.11: Normalized confusion matrix of the WV-3 validation dataset for the DAugNet model.

common in remote sensing. The CycleGAN could partly detect the road (track) in the third from the top sample. The roads is the worse detecting class in this project because all the models misclassify dirt tracks with the background.

The CyCADA model was able to detect roads and buildings partly. In the fourth from the bottom image, the dirt road was misclassified as background, though its contours are noticeable. In some images, vegetation is also was misclassified as hydro.

The DAugNet model was somewhat accurate in buildings and vegetation prediction, having all other classes significantly misclassified. Some background became quite dark after the style-transferring; thus, the model classified them as hydro. The same can be said about the vegetation and a shadow under it which was classified as hydro. In the third from the bottom sample, however, we can see how relatively accurate two buildings are detected and a forest area around them even though it is labelled as a background on validation labels. In Fig. 6.13 the examples with the highest (for the baseline model) mIoU score are shown. The water, vegetation, and background classes are decently detected. The road class, however, is an exclusion. In Fig. 6.14 the examples with the lowest (for the baseline model) mIoU score are shown. There are many misclassifications for the buildings class and for the roads that are completely invisible. In Fig. 6.15 the examples with the highest (for the AdaptSegNet model) mIoU score are shown. The water and vegetation classes for these samples were detected somewhat accurately. The buildings class, represented in the second column, was also detected. The road class, however, was not recognized completely. In Fig. 6.16 the examples with the lowest (for the AdaptSegNet model) mIoU score are shown. As can be seen, all the object classes were misclassified as the background class. In Fig. 6.17 the examples with the highest (for the CycleGAN model) mIoU score are shown. The water, vegetation, and background classes are decently detected. The man-made structure on the second-column sample is misclassified as a road, which is explainable. In Fig. 6.18 the examples with the lowest (for the CycleGAN model) mIoU score are shown. The background area on the third and fourth-column samples is misclassified as vegetation. The ground-truth labels for these samples are probably not accurate. In Fig. 6.19 the examples with the highest (for the CyCADA model) mIoU score are shown. The water, vegetation, buildings and background classes are properly detected. The road class, however, was misclassified as the background class. In Fig. 6.20 the examples with the lowest (for the CyCADA model) mIoU score are shown. All classes were misclassified mostly as the background class. In Fig. 6.21 the examples with the highest (for the DAugNet model) mIoU score are shown. The water, vegetation, and buildings were somewhat accurately identified. However, the road on the second-column sample was misclassified as the water class. In Fig. 6.22 the examples with the lowest (for the DAugNet model) mIoU score are shown. The first-column sample was completely labelled as the water class, probably, because of its intensive blue spectrum appearance.

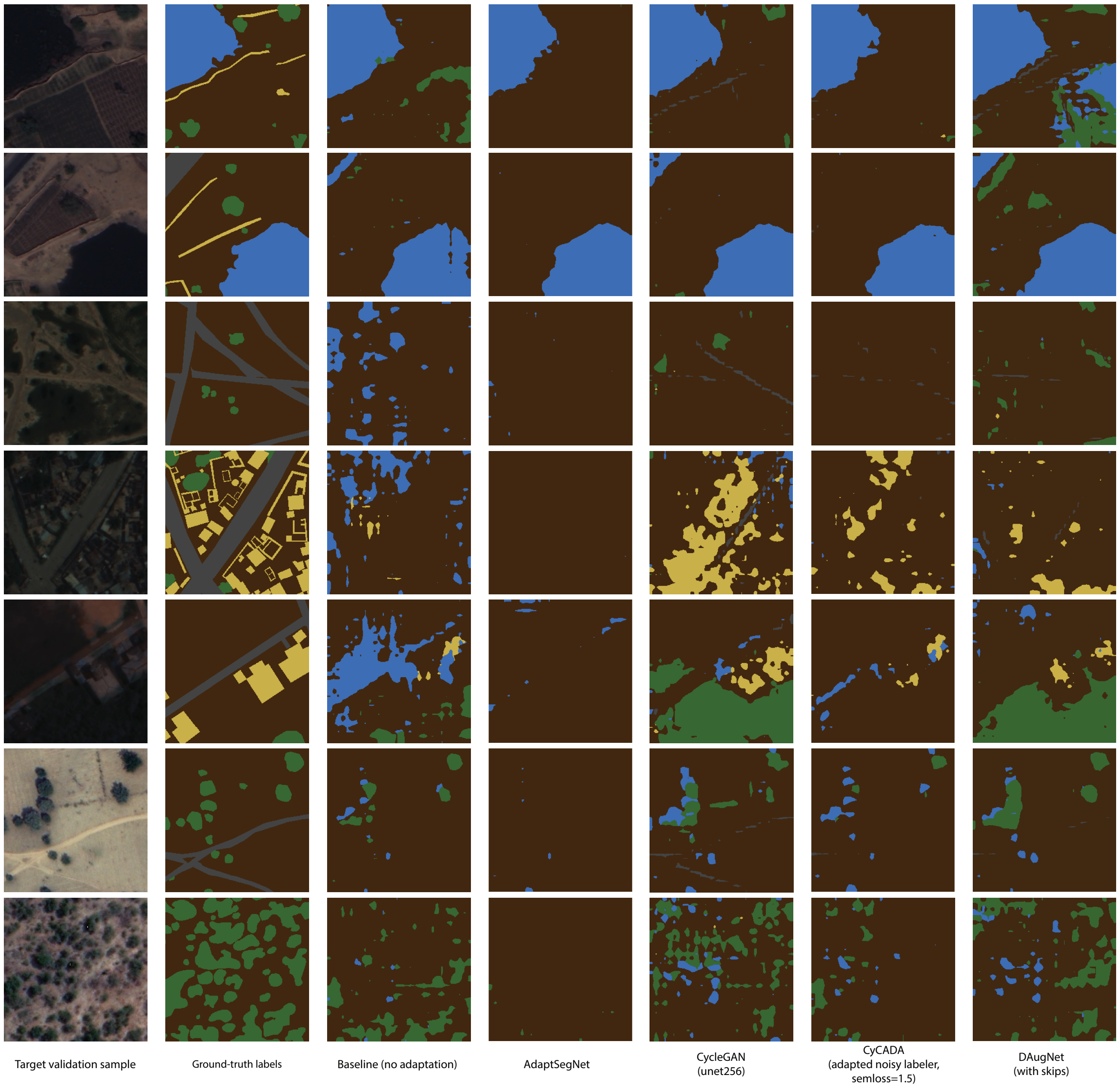


Figure 6.12: Examples of the labels predicted by each model. Brown, green, blue, gray, and yellow colors represent background, vegetation, hydro, roads, and buildings classes, respectively.

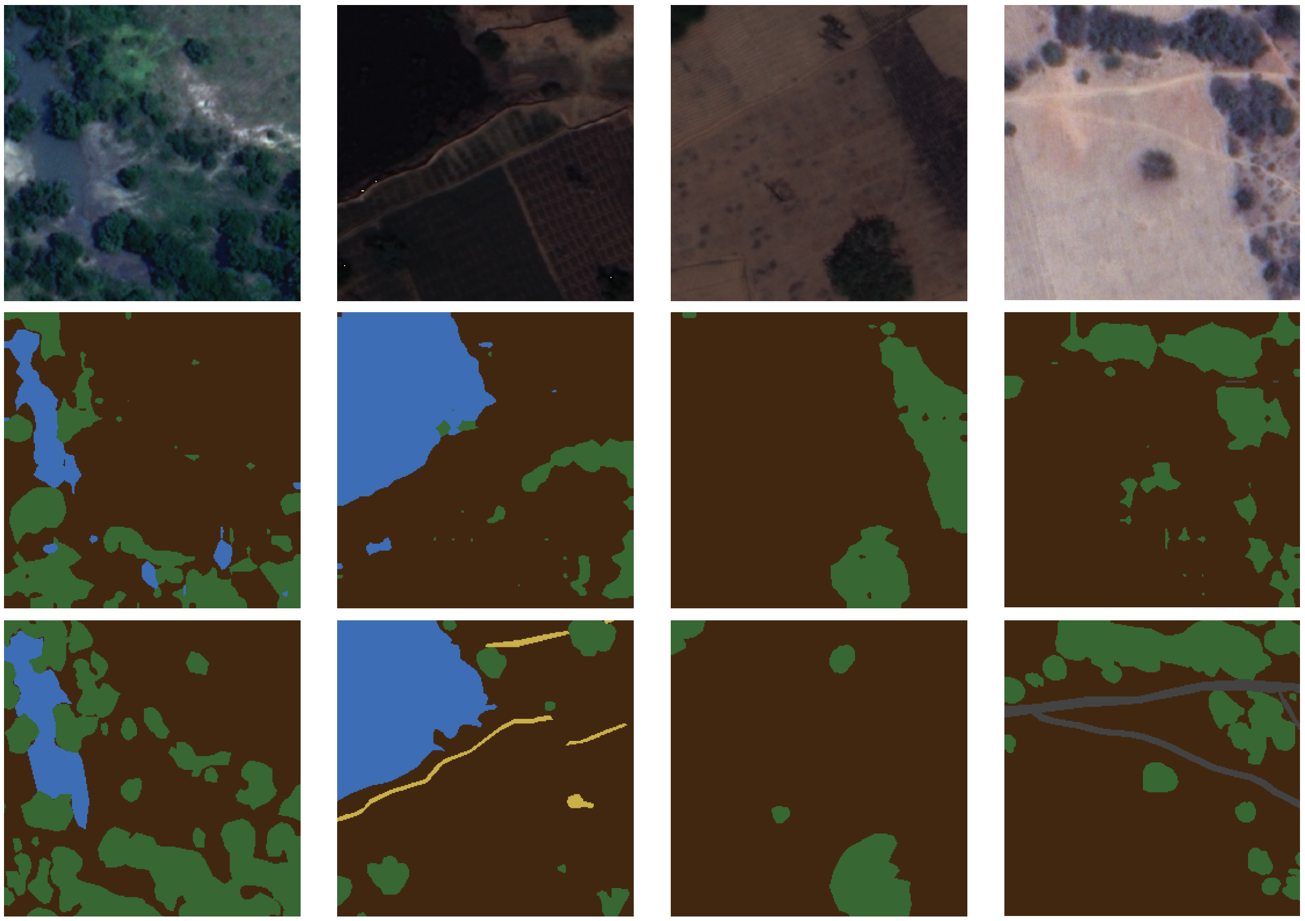


Figure 6.13: Examples of good results predicted by the baseline model. Second row is predicted tiles, and lower row is ground truth tiles.

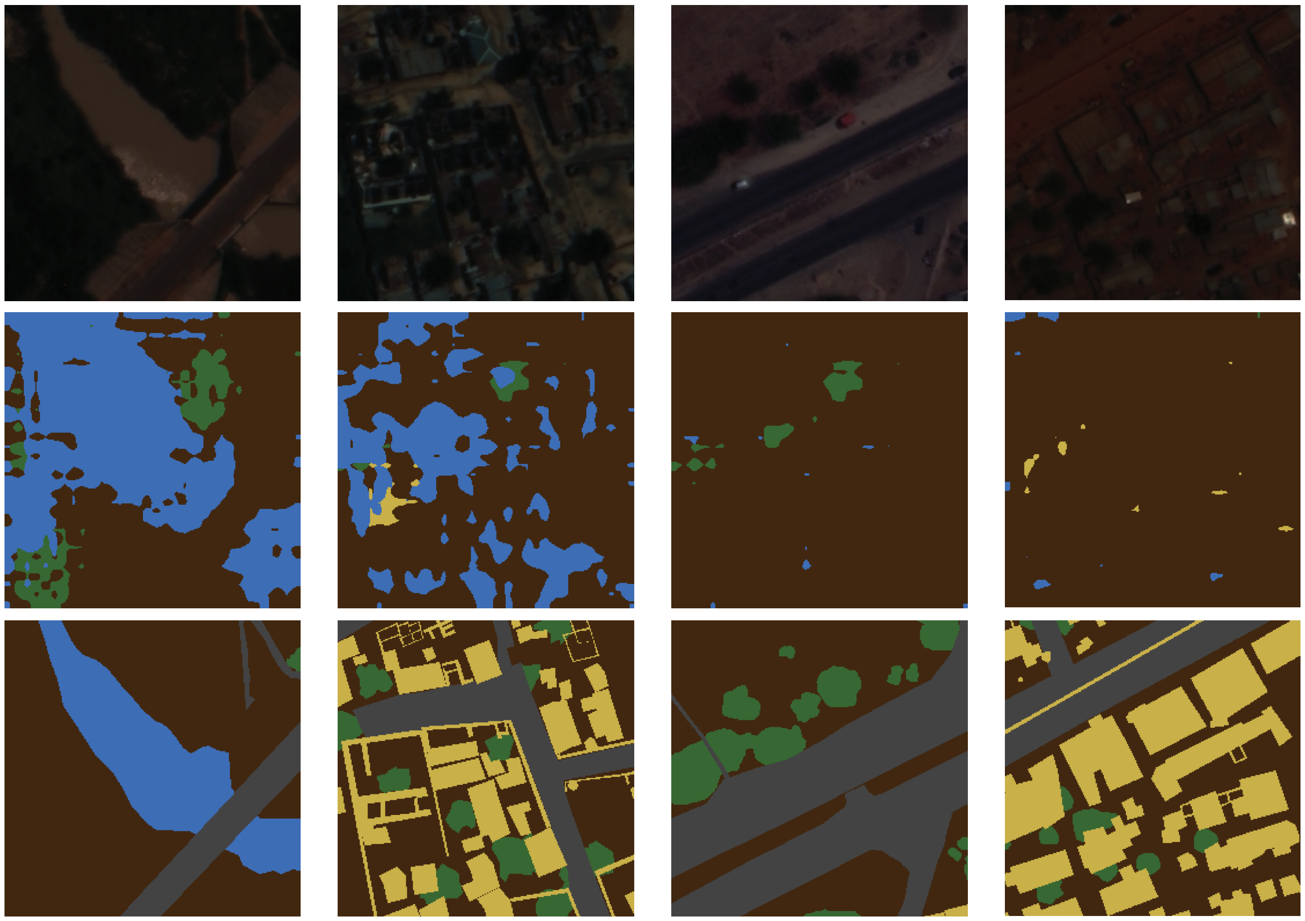


Figure 6.14: Examples of bad results predicted by the baseline model. Second row is predicted tiles, and lower row is ground truth tiles.

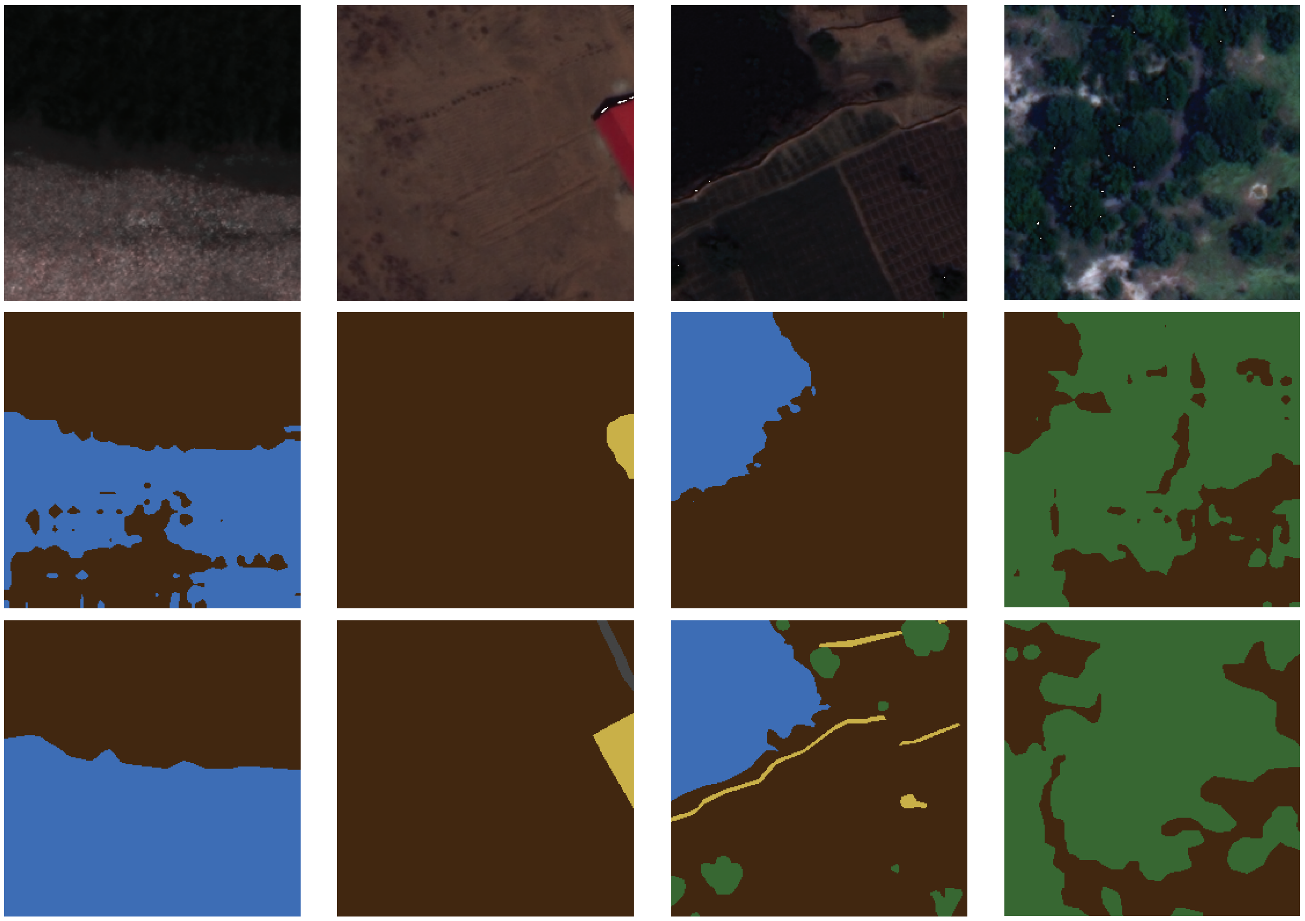


Figure 6.15: Examples of good results predicted by the AdaptSegNet model. Second row is predicted tiles, and lower row is ground truth tiles.

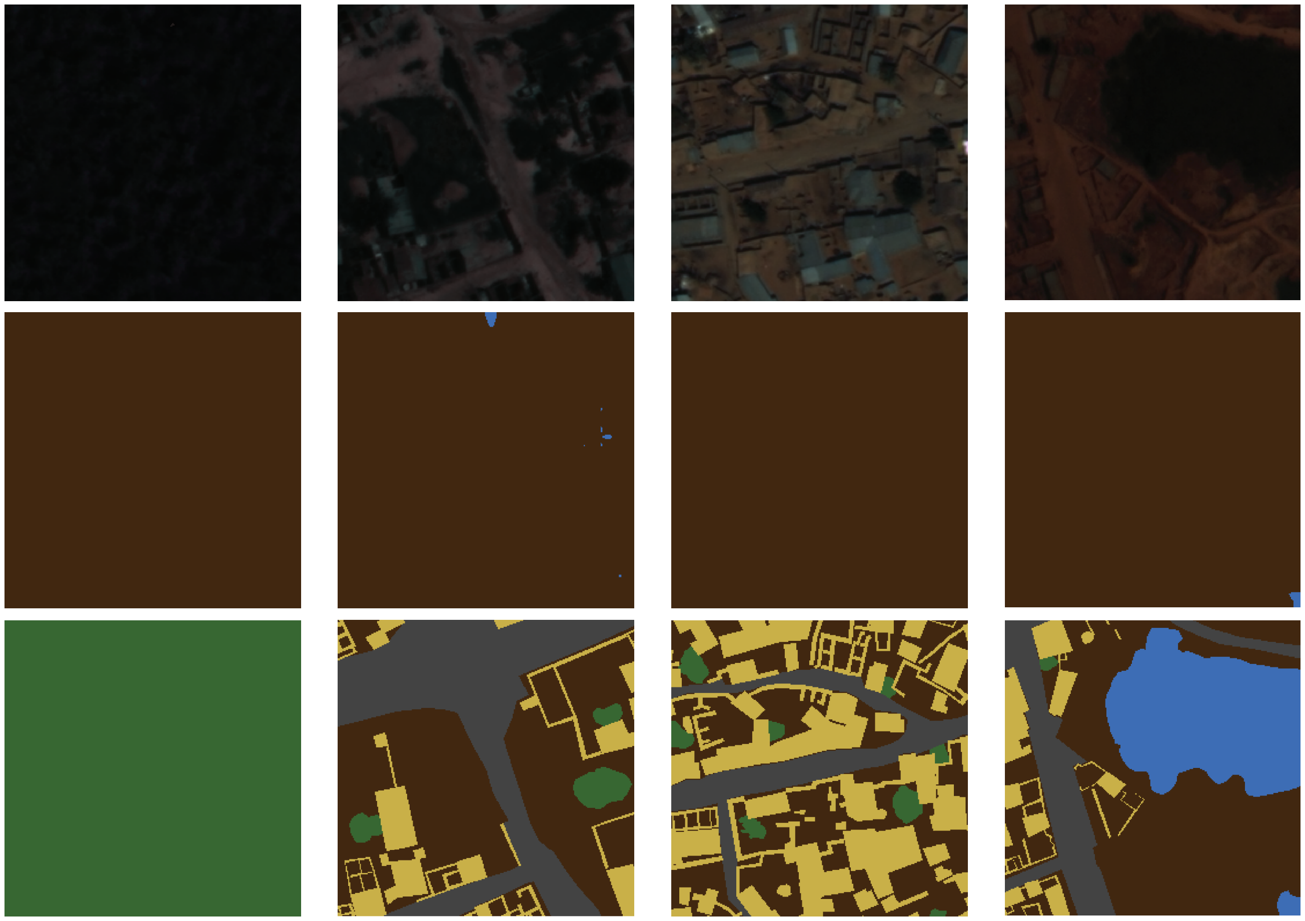


Figure 6.16: Examples of bad results predicted by the AdaptSegNet model. Second row is predicted tiles, and lower row is ground truth tiles.

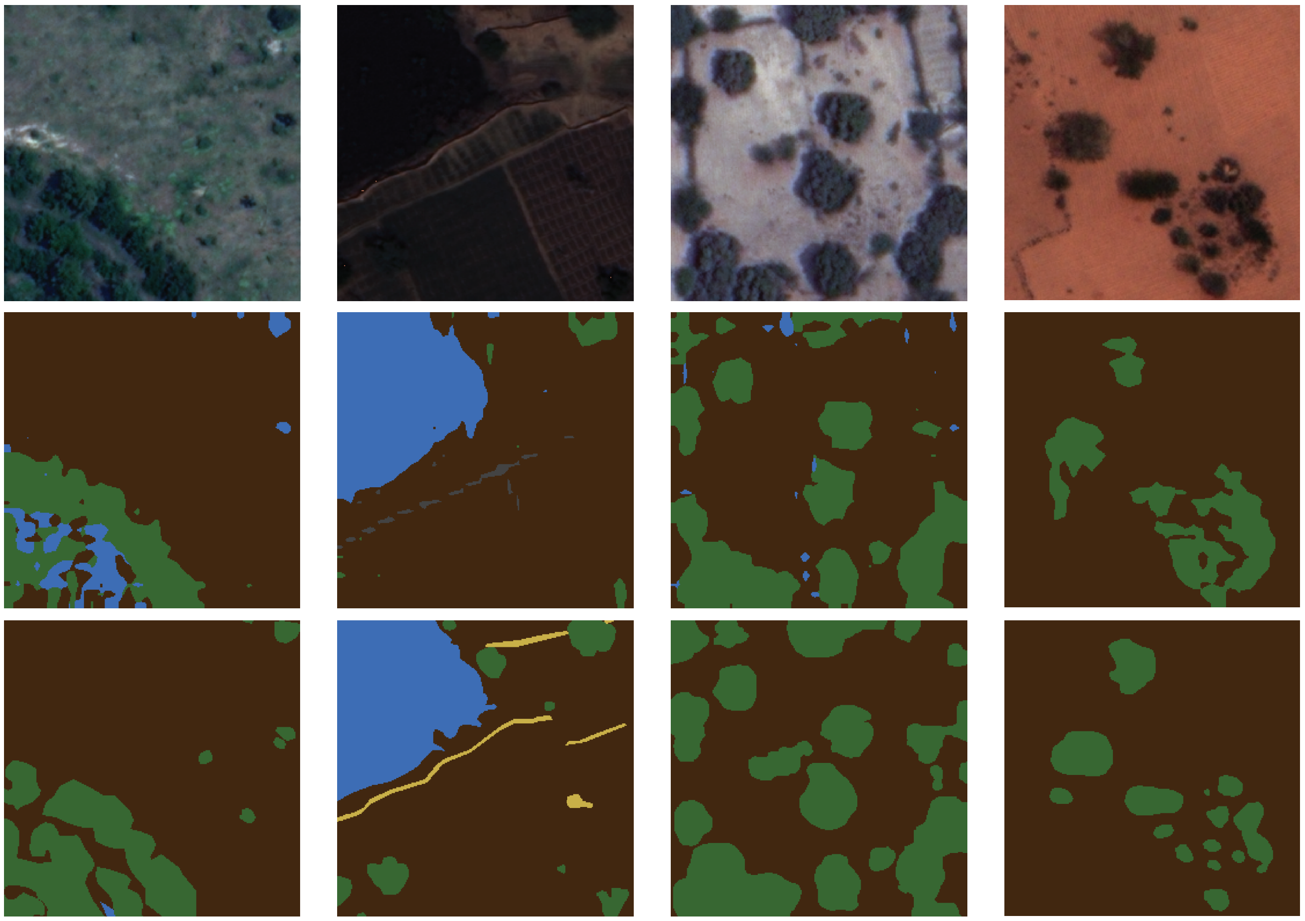


Figure 6.17: Examples of good results predicted by the CycleGAN model. Second row is predicted tiles, and lower row is ground truth tiles.

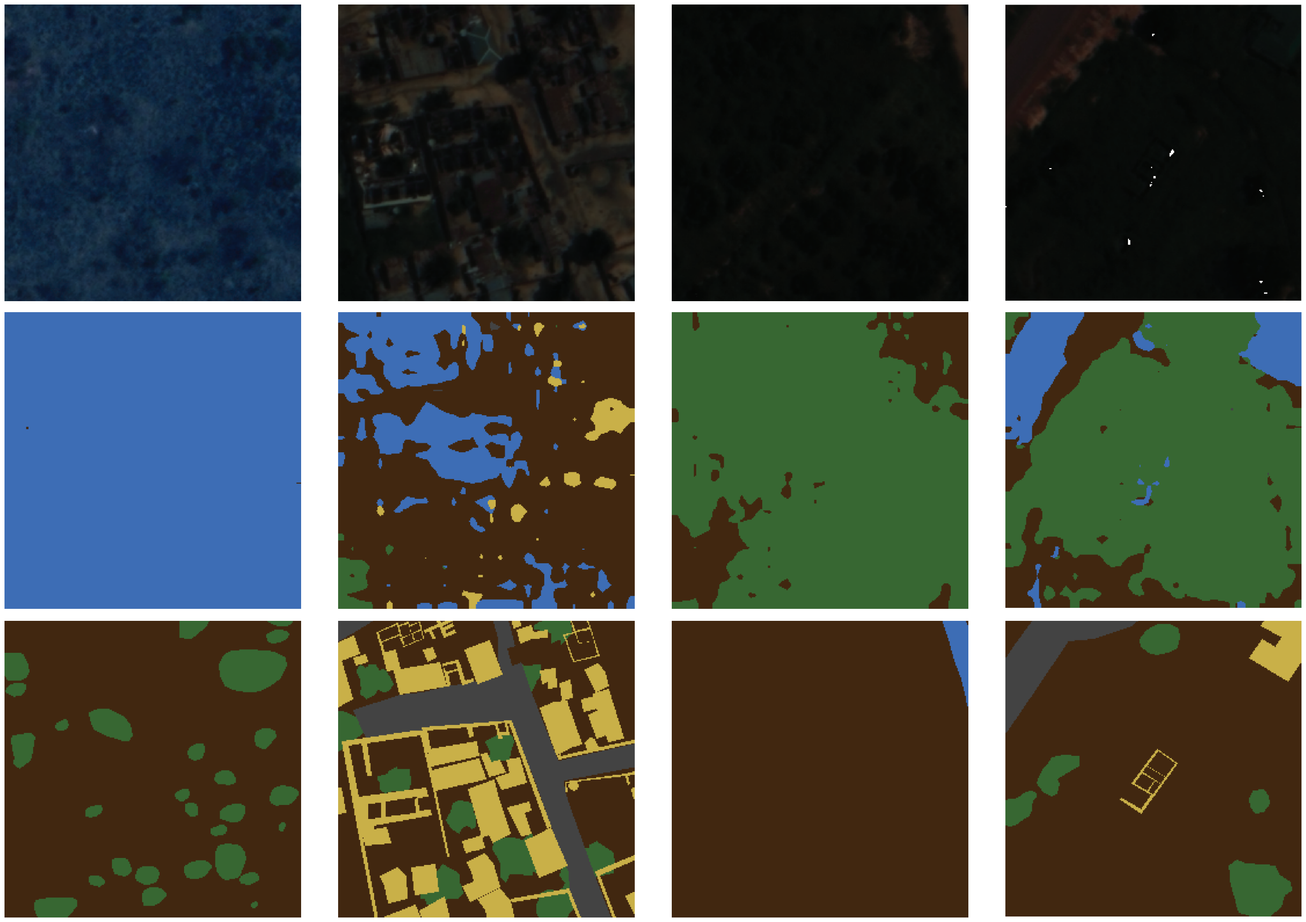


Figure 6.18: Examples of bad results predicted by the CycleGAN model. Second row is predicted tiles, and lower row is ground truth tiles.

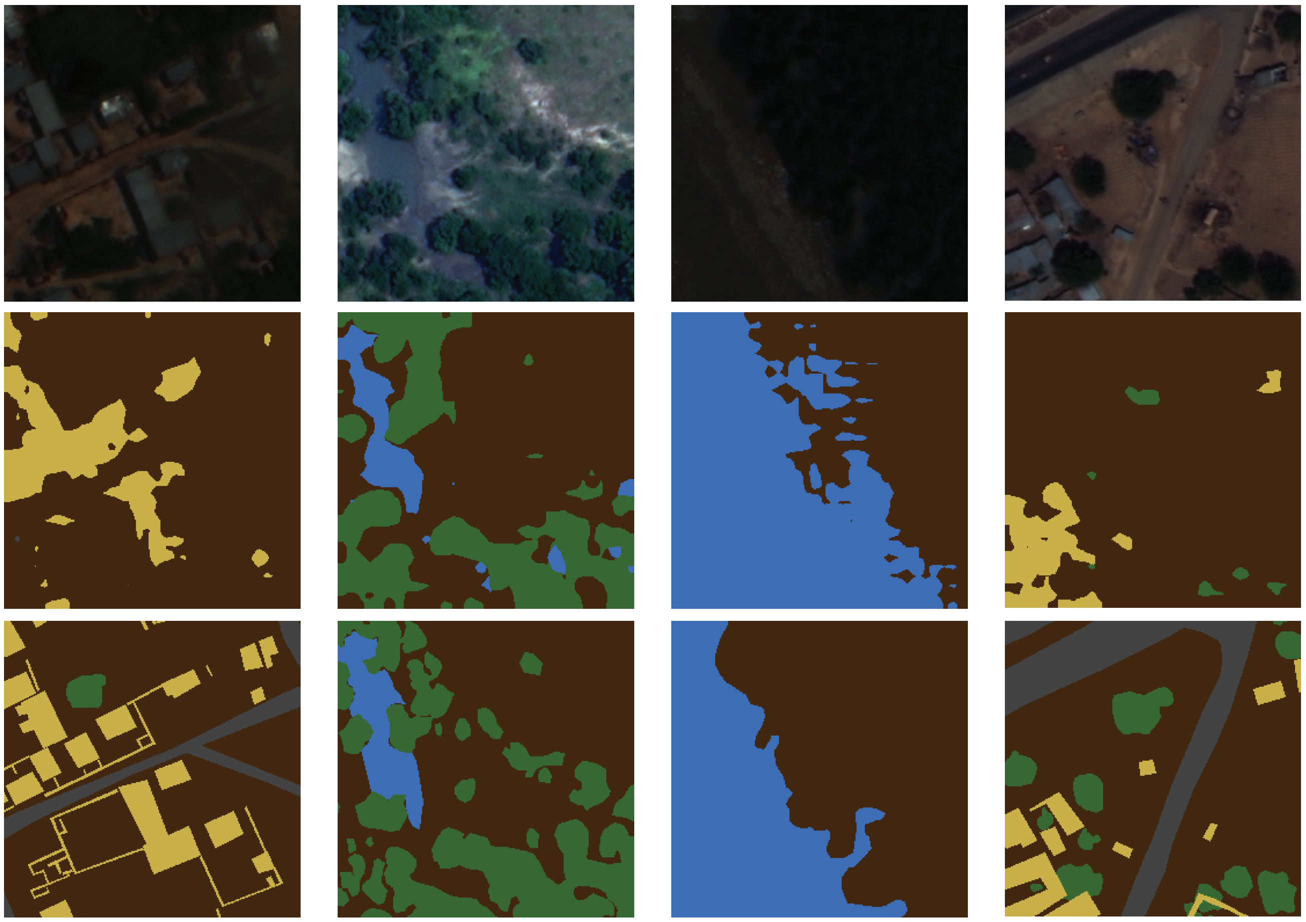


Figure 6.19: Examples of good results predicted by the CyCADA model. Second row is predicted tiles, and lower row is ground truth tiles.

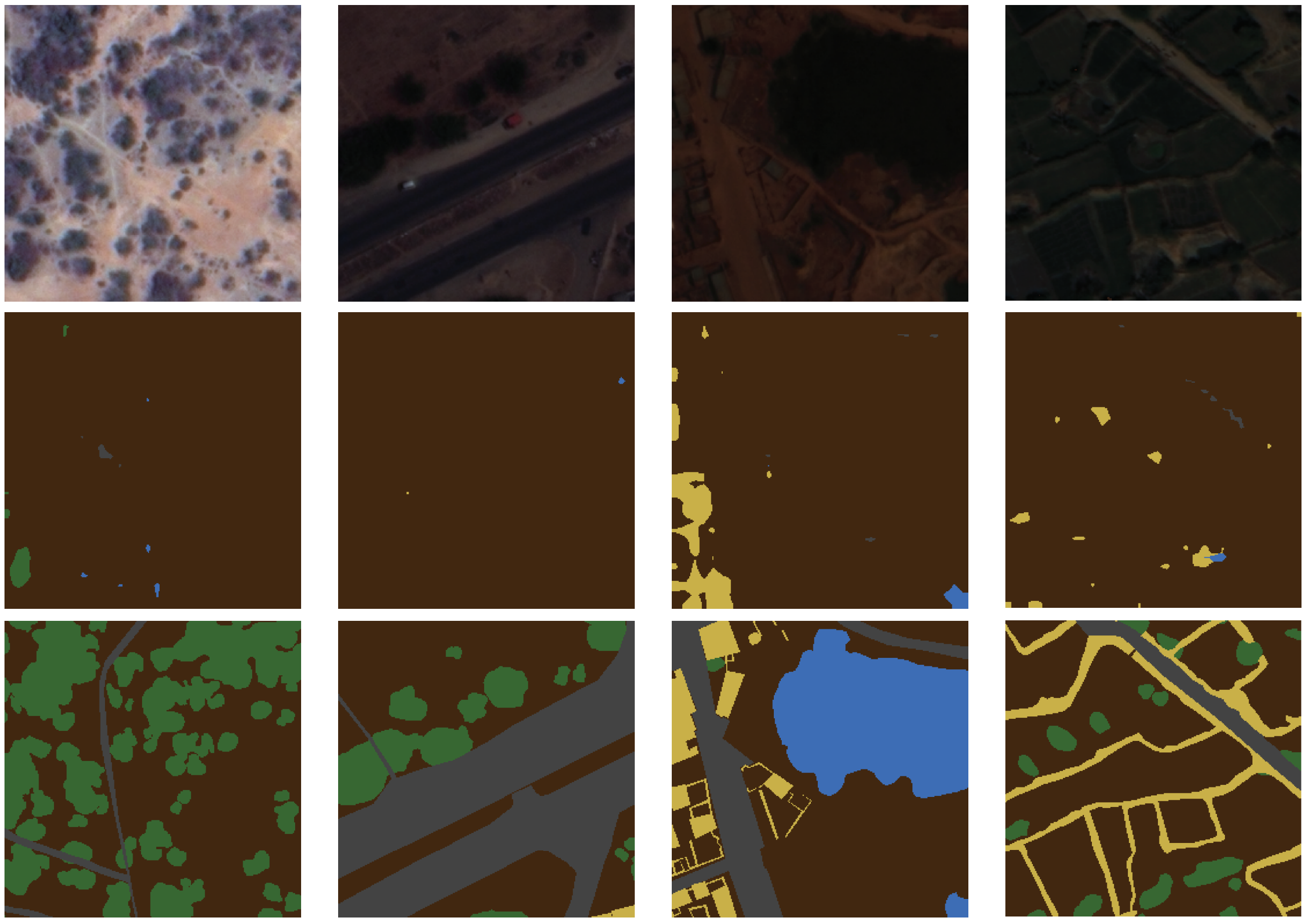


Figure 6.20: Examples of bad results predicted by the CyCADA model. Second row is predicted tiles, and lower row is ground truth tiles.

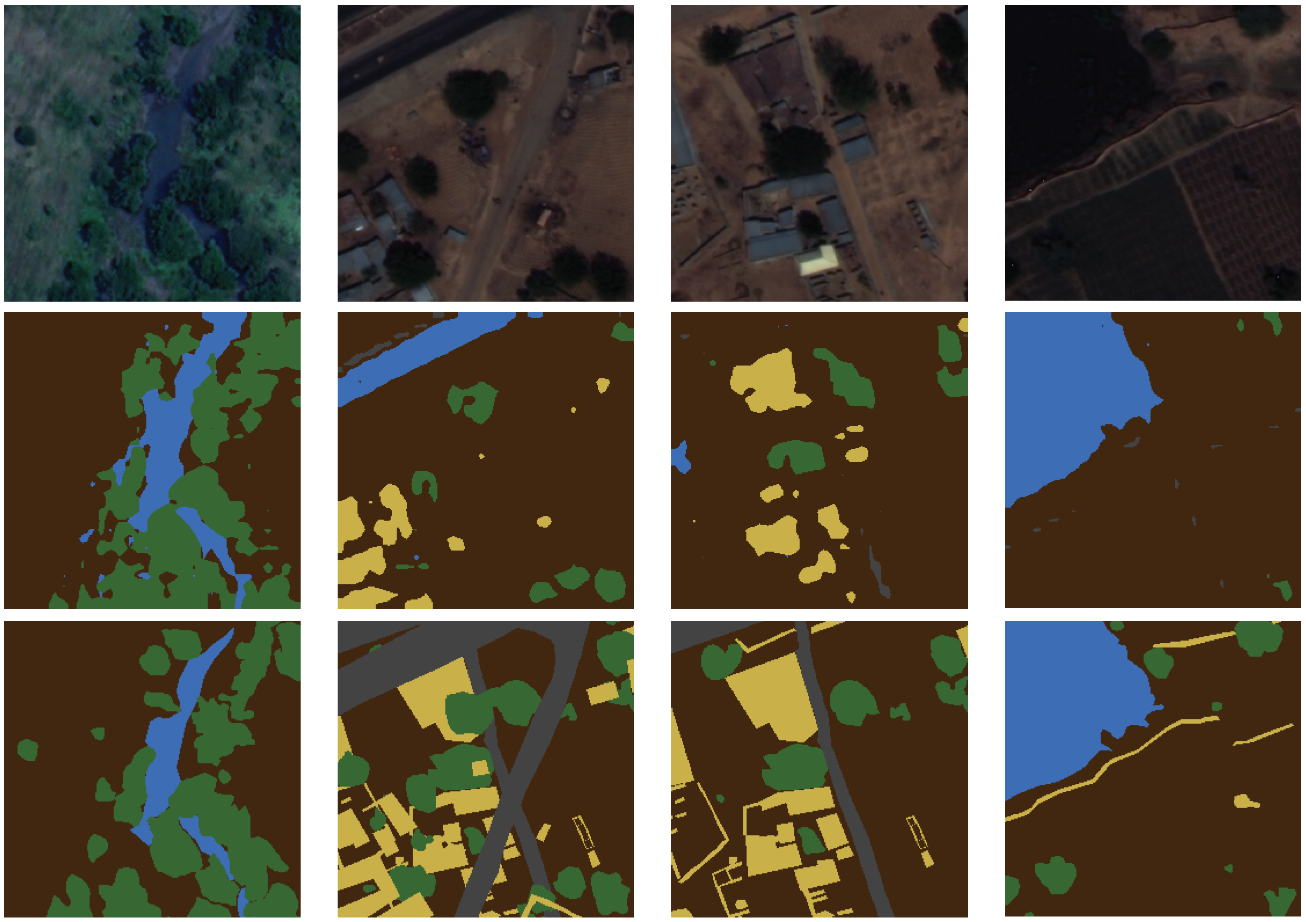


Figure 6.21: Examples of good results predicted by the DAugNet model. Second row is predicted tiles, and lower row is ground truth tiles..

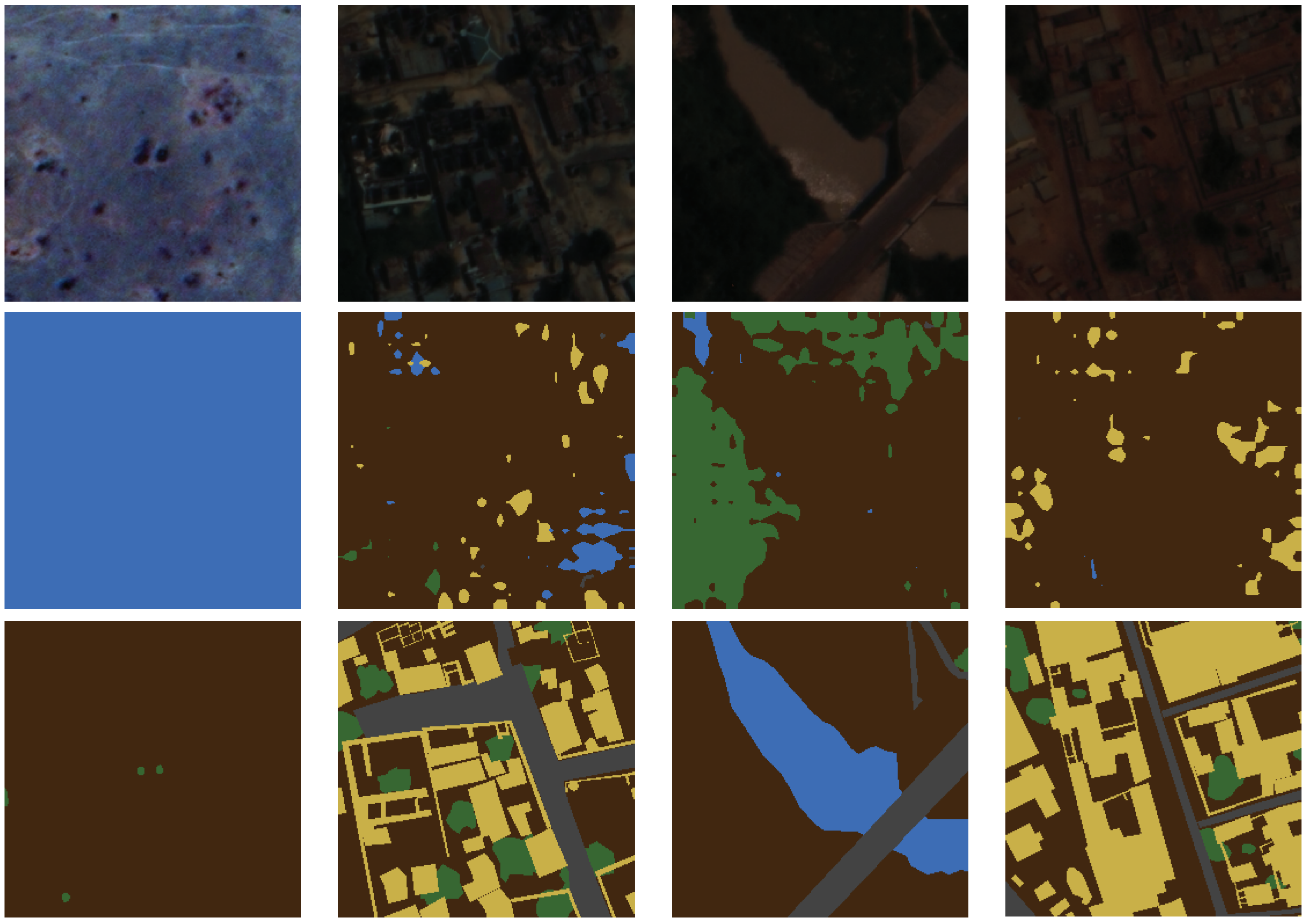


Figure 6.22: Examples of bad results predicted by the DAugNet model. Second row is predicted tiles, and lower row is ground truth tiles.

Chapter 7

Conclusion

The main goal of this thesis was to investigate the performance of state-of-the-art unsupervised adaptation models in remote sensing where the discrepancy between domains is significant. Models such as AdaptSegNet, CycleGAN, CyCADA, and DAugNet were implemented and evaluated. The DAugNet model was improved by incorporating skip connections between the first convolutional block and each decoder block, which increased the overall mIoU accuracy by 1.91% and reached the maximum IoU score for the vegetation class (22.08%) among all the models. Additionally, the CyCADA model was enhanced by introducing an adapted noisy labeller rather than the one described in the original paper. This improvement allowed the CyCADA to outperform all other models and reach the maximum mIoU score of 32.6%. The AdaptSegNet model was not considered successful overall but showed the highest performance in the hydro class (43.64%). The CycleGAN showed the lowest performance mainly because it does not maintain semantic consistency while style transferring.

The models were trained on two real-world remote sensing imagery sets. The domain discrepancy between the datasets is highly substantial. Not only is there a difference between the satellite sensors, but also the samples of both datasets represent semantically similar but structurally different objects. These obstacles did not allow high adaptational performance, and, therefore, the segmentation task was solved weakly. Compared to the model without any adaptation, however, the models considered in this thesis make a class like roads - that was invisible without adaptation - visible and double the accuracy of the hydro and buildings classes.

As future work, the following steps could be taken. First, adaptation within each domain is also necessary. The samples from the same domain can be substantially different in the visual spectrum because they were taken at different times of the day, illumination, and weather conditions. This potentially creates additional noise for the adaptation model, thus preventing it from accurate domain adaptation. Second, more efficient methods of semantic consistency preservation could be used. For example, the noisy labeller from the CyCADA model can be combined with the adaptive instance normalization layer and the Sobel gradient loss from the DAugNet model. In remote sensing, having semantically consistent objects is crucial when the final task includes style transferring. Third, to lower the complexity of the adaptation task, it might be helpful to build a “per-class" adaptation model , *i.e.* the model which is aimed to adapt and then predict only one class. Combining these models could be used as an alternative to a single model that adapts all classes at once.

Bibliography

[1] Natural Resources Canada, "Land cover & land use," https://www.nrcan.gc.ca/maps-tools-and-publications/satellite-imagery-and-air-photos/tutorial-fundamentals-remote-sensing/educational-resources-applications/land-cover-biomass-mapping/land-cover-land-use/9373, Feb. 2008, accessed: 2022-2-3.

[2] "Sentinel-1 - missions - sentinel online - sentinel online," https://sentinel.esa.int/web/sentinel/missions/sentinel-1, accessed: 2022-2-3.

[3] D. Lu and Q. Weng, "A survey of image classification methods and techniques for improving classification performance," *Int. J. Remote Sens.*, vol. 28, no. 5, pp. 823–870, Mar. 2007.

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

[5] 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 Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 4, pp. 834–848, Apr. 2018.

[6] G. Lin, C. Shen, A. van den Hengel, and I. Reid, "Efficient piecewise training of deep structured models for semantic segmentation," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, Jun. 2016.

[7] Z. Liu, X. Li, P. Luo, C.-C. Loy, and X. Tang, "Semantic image segmentation via deep parsing network," in *2015 IEEE International Conference on Computer Vision (ICCV)*. IEEE, Dec. 2015.

[8] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite"," in *2012 IEEE conference on computer vision and pattern recognition*, 2012, pp. 3354–3361.

[9] Y.-H. Tsai, X. Shen, Z. Lin, K. Sunkavalli, X. Lu, and M.-H. Yang, “Deep image harmonization,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, Jul. 2017.

[10] 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 Comput. Appl.*, vol. 32, no. 12, pp. 8529–8544, Jun. 2020.

[11] C. D. Storie and C. J. Henry, “Deep learning neural networks for land use land cover mapping”, IGARSS 2018 - 2018 IEEE international geoscience and remote sensing symposium,” pp. 3445–3448, 2018.

[12] I. Goodfellow, “Generative adversarial nets,” *Advances in neural information processing systems*, vol. 27, 2014.

[13] Y.-H. Tsai, W.-C. Hung, S. Schulter, K. Sohn, M.-H. Yang, and M. Chandraker, “Learning to adapt structured output space for semantic segmentation,” in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, Jun. 2018.

[14] O. Tasar, S. L. Happy, Y. Tarabalka, and P. Alliez, “ColorMapGAN: Unsupervised domain adaptation for semantic segmentation using color mapping generative adversarial networks,” *IEEE Trans. Geosci. Remote Sens.*, vol. 58, no. 10, pp. 7178–7193, Oct. 2020.

[15] O. Tasar, A. Giros, Y. Tarabalka, P. Alliez, and S. Clerc, “DAugNet: Unsupervised, multisource, multitarget, and life-long domain adaptation for semantic segmentation of satellite images,” *IEEE Trans. Geosci. Remote Sens.*, vol. 59, no. 2, pp. 1067–1081, Feb. 2021.

[16] O. Tasar, S. L. Happy, Y. Tarabalka, and P. Alliez, “SEMI2I: Semantically consistent image-to-image translation for domain adaptation of remote sensing data,” in *IGARSS 2020 - 2020 IEEE International Geoscience and Remote Sensing Symposium*. IEEE, Sep. 2020.

[17] “GeoEye-1 satellite sensor,” https://www.satimagingcorp.com/satellite-sensors/geoeye-1/, accessed: 2022-2-3.

[18] “WorldView-2 satellite sensor,” https://www.satimagingcorp.com/satellite-sensors/worldview-2/, accessed: 2022-2-3.

[19] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, “Unpaired image-to-image translation using cycle-consistent adversarial networks,” in *2017 IEEE International Conference on Computer Vision (ICCV)*. IEEE, Oct. 2017.

[20] H. Chen, M. Guan, and H. Li, “ArCycleGAN: Improved CycleGAN for style transferring of fruit images,” *IEEE Access*, vol. 9, pp. 46 776–46 787, 2021.

[21] X. Jin, Y. Qi, and S. Wu, *Cyclegan face-off”*, 2017.

[22] A. Almahairi, S. Rajeswar, A. Sordoni, P. Bachman, and A. Courville, “Augmented CycleGAN: Learning many-to-many mappings from unpaired data,” Feb. 2018.

[23] C.-W. Wu, J.-Y. Liu, Y.-H. Yang, and J.-S. R. Jang, “Singing style transfer using cycle-consistent boundary equilibrium generative adversarial networks,” Jul. 2018.

[24] J. Hoffman, E. Tzeng, T. Park, J.-Y. Zhu, P. Isola, K. Saenko, A. Efros, and T. Darrell, “CyCADA: Cycle-consistent adversarial domain adaptation,” in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. PMLR, 10–15 Jul 2018, pp. 1989–1998. [Online]. Available: https://proceedings.mlr.press/v80/hoffman18a.html

[25] X. Huang and S. Belongie, “Arbitrary style transfer in real-time with adaptive instance normalization,” in *2017 IEEE International Conference on Computer Vision (ICCV)*. IEEE, Oct. 2017.

[26] E. R. Kandel, J. H. Schwartz, T. M. Jessell, S. A. Siegelbaum, and A. J. Hudspeth, *Principles of neural science, fifth edition*, 5th ed. New York, NY: McGraw-Hill Medical, Dec. 2000.

[27] F. Rosenblatt, “The perceptron: a probabilistic model for information storage and organization in the brain,” *Psychol. Rev.*, vol. 65, no. 6, pp. 386–408, Nov. 1958.

[28] T. Szandała, “Review and comparison of commonly used activation functions for deep neural networks,” in *Bio-inspired Neurocomputing*, ser. Studies in computational intelligence. Singapore: Springer Singapore, 2021, pp. 203–224.

[29] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” Feb. 2015.

[30] 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.

[31] S. Basodi, C. Ji, H. Zhang, and Y. Pan, "Gradient amplification: An efficient way to train deep neural networks"," *Big Data Mining and Analytics*, vol. 3, no. 3, pp. 196–207, 2020.

[32] H. Daumé, *A course in machine learning.* Hal Daumé III, 2017.

[33] M. Schuld, I. Sinayskiy, and F. Petruccione, ""simulating a perceptron on a quantum computer"," *Physics Letters A*, vol. 379, no. 7, pp. 660–663, 2015.

[34] V. Kŭrková, "Kolmogorov's theorem and multilayer neural networks"," *Neural networks*, vol. 5, no. 3, pp. 501–506, 1992.

[35] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Netw.*, vol. 2, no. 5, pp. 359–366, Jan. 1989.

[36] A. Pinkus, "Approximation theory of the MLP model in neural networks"," *Acta numerica*, vol. 8, pp. 143–195, 1999.

[37] H. Zhao, O. Gallo, I. Frosio, and J. Kautz, "Loss functions for neural networks for image processing," Nov. 2015.

[38] K. Ikeuchi, *Computer vision: A reference guide.* Springer Publishing Company, 2014.

[39] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," Dec. 2014.

[40] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, Oct. 1986.

[41] M. A. Nielsen, *Neural networks and deep learning.* San Francisco, CA: Determination press, 2015, vol. 25.

[42] E. Million, "The hadamard product," *Course Notes*, vol. 3, no. 6, 2007.

[43] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," Mar. 2016.

[44] H. Wu and X. Gu, "Max-pooling dropout for regularization of convolutional neural networks," in *Neural Information Processing*, ser. Lecture notes in computer science. Cham: Springer International Publishing, 2015, pp. 46–54.

[45] D. Ulyanov, A. Vedaldi, and V. Lempitsky, "Instance normalization: The missing ingredient for fast stylization," Jul. 2016.

[46] E. Fix and J. L. Hodges, join(' ', "Discriminatory analysis. nonparametric discrimination: Consistency properties," *Int. Stat. Rev.*, vol. 57, no. 3, p. 238, Dec. 1989.

[47] A. C. Bovik, "Basic gray level image processing," in *The Essential Guide to Image Processing.* Elsevier, 2009, pp. 43–68.

[48] M. Drozdzal, E. Vorontsov, G. Chartrand, S. Kadoury, and C. Pal, "The importance of skip connections in biomedical image segmentation," in *Deep Learning and Data Labeling for Medical Applications*, ser. Lecture notes in computer science. Cham: Springer International Publishing, 2016, pp. 179–187.

[49] R. J. Lipton and N. E. Young, "Simple strategies for large zero-sum games with applications to complexity theory," in *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing - STOC '94.* New York, New York, USA: ACM Press, 1994.

[50] S. Barua, S. M. Erfani, J. Bailey, and . Fcc-Gan, *"FCC-GAN: A fully connected and convolutional net architecture for GANs"*, 2019.

[51] D. B. Kirk and W.-M. W. Hwu, *Programming Massively Parallel Processors, Third Edition: A Hands-on Approach.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc, 2016.

[52] S. Saadatnejad, S. Li, T. Mordan, and A. Alahi, "A shared representation for photorealistic driving simulators," *IEEE Trans. Intell. Transp. Syst.*, pp. 1–11, 2021.

[53] D. Bahdanau, J. Chorowski, D. Serdyuk, P. Brakel, and Y. Bengio, "End-to-end attention-based large vocabulary speech recognition," in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* IEEE, Mar. 2016.

[54] D. Ding, Z. Ma, D. Chen, Q. Chen, Z. Liu, and F. Zhu, "Advances in video compression system using deep neural network: A review and case studies," *Proc. IEEE Inst. Electr. Electron. Eng.*, vol. 109, no. 9, pp. 1494–1520, Sep. 2021.

[55] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP).* Stroudsburg, PA, USA: Association for Computational Linguistics, 2014.

[56] M. Mohammadi, A. Al-Fuqaha, S. Sorour, and M. Guizani, "Deep learning for IoT big data and streaming analytics: A survey," *IEEE Commun. Surv. Tutor.*, vol. 20, no. 4, pp. 2923–2960, 2018.

[57] M. M. Yapıcı, A. Tekerek, and N. Topaloğlu, "Derin öğrenme araştırma alanlarının literatür taraması," *Gazi Muhendis. Bilim. Derg.*, vol. 5, no. 3, pp. 188–215, Dec. 2019.

[58] H. Li, *Remote Sensing Images Semantic Segmentation with General Remote Sensing Vision Model via a Self-Supervised Contrastive Learning Method*, 2021.

[59] X. Yang, S. Li, Z. Chen, J. Chanussot, X. Jia, B. Zhang, B. Li, and P. Chen, "An attention-fused network for semantic segmentation of very-high-resolution remote sensing imagery," *ISPRS J. Photogramm. Remote Sens.*, vol. 177, pp. 238–262, Jul. 2021.

[60] Y. Ganin and V. Lempitsky, "Unsupervised domain adaptation by backpropagation," in *International conference on machine learning.* PMLR, 2015, pp. 1180–1189.

[61] H. Chen, C. Wu, Y. Xu, and B. Du, "Unsupervised domain adaptation for semantic segmentation via low-level edge information transfer," Sep. 2021.

[62] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* IEEE, Jun. 2015.

[63] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," Sep. 2014.

[64] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* IEEE, Jun. 2016.

[65] "The PASCAL visual object classes homepage," http://host.robots.ox.ac.uk/pascal/VOC/, accessed: 2022-2-3.

[66] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," Nov. 2015.

[67] 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.

[68] D. L. Torres, "Applying fully convolutional architectures for semantic segmentation of a single tree species in urban environment on high resolution UAV optical imagery," *Sensors*, vol. 20, no. 2.

[69] F. Yu, V. Koltun, and T. Funkhouser, "Dilated residual networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* IEEE, Jul. 2017.

[70] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, Dec. 2015.

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

[72] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *Lecture Notes in Computer Science*, ser. Lecture notes in computer science. Cham: Springer International Publishing, 2015, pp. 234–241.

[73] Z. Ren, B. Hou, Z. Wen, and L. Jiao, "Patch-sorted deep feature learning for high resolution SAR image classification," *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.*, vol. 11, no. 9, pp. 3113–3126, Sep. 2018.

[74] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition.* IEEE, Jun. 2009.

[75] Y. Xu, L. Wu, Z. Xie, and Z. Chen, "Building extraction in very high resolution remote sensing imagery using deep learning and guided filters," *Remote Sens. (Basel)*, vol. 10, no. 1, p. 144, Jan. 2018.

[76] K. He, J. Sun, and X. Tang, "Guided image filtering," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 6, pp. 1397–1409, Jun. 2013.

[77] D. Cheng, G. Meng, S. Xiang, and C. Pan, "FusionNet: Edge aware deep convolutional networks for semantic segmentation of remote sensing harbor images," *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.*, vol. 10, no. 12, pp. 5769–5783, Dec. 2017.

[78] Z. Lu and Y. Chen, "Single image super-resolution based on a modified u-net with mixed gradient loss", signal, image and video processing," pp. 1–9, 2021.

[79] I. Sobel, *An Isotropic 3x3 Image Gradient Operator," Presentation at Stanford A.I. Project*, 1968.

[80] D. Marmanis, J. D. Wegner, S. Galliani, K. Schindler, M. Datcu, and U. Stilla, "Semantic segmentation of aerial images with an ensemble of CNSS", ISPRS annals of the photogrammetry," *Remote Sensing and Spatial Information Sciences*, vol. 3, pp. 473–480, 2016.

[81] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk, "SLIC superpixels compared to state-of-the-art superpixel methods," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 11, pp. 2274–2282, Nov. 2012.

[82] Y. Sun, X. Zhang, Q. Xin, and J. Huang, "Developing a multi-filter convolutional neural network for semantic segmentation using high-resolution aerial imagery and LiDAR data", ISPRS journal of photogrammetry and remote sensing," vol. 143, pp. 3–14, 2018.

[83] P. Ghamisi, Y. Chen, and X. X. Zhu, "A self-improving convolution neural network for the classification of hyperspectral data," *IEEE Geosci. Remote Sens. Lett.*, vol. 13, no. 10, pp. 1537–1541, Oct. 2016.

[84] W. Zhao and S. Du, "Spectral–spatial feature extraction for hyperspectral image classification: A dimension reduction and deep learning approach," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 54, no. 8, pp. 4544–4554, 2016.

[85] S. Yu, S. Jia, and C. Xu, "Convolutional neural networks for hyperspectral image classification," *Neurocomputing*, vol. 219, pp. 88–98, Jan. 2017.

[86] Y. Li, H. Zhang, and Q. Shen, "Spectral–spatial classification of hyperspectral imagery with 3D convolutional neural network"," *Remote Sensing*, vol. 9, no. 1, 2017.

[87] B. Fang, Y. Li, H. Zhang, and J. Chan, "Hyperspectral images classification based on dense convolutional networks with spectral-wise attention mechanism," *Remote Sens. (Basel)*, vol. 11, no. 2, p. 159, Jan. 2019.

[88] Y. Ganin, "Domain-adversarial training of neural networks", the journal of machine learning research," vol. 17, pp. 2096–2030, 2016.

[89] E. Tzeng, J. Hoffman, N. Zhang, K. Saenko, and T. Darrell, "Deep domain confusion: Maximizing for domain invariance," Dec. 2014.

[90] W. Hong, Z. Wang, M. Yang, and J. Yuan, "Conditional generative adversarial network for structured domain adaptation," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition.* IEEE, Jun. 2018.

[91] E. Tzeng, J. Hoffman, K. Saenko, and T. Darrell, "Adversarial discriminative domain adaptation," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* IEEE, Jul. 2017.

[92] Y. Li, L. Yuan, and N. Vasconcelos, "Bidirectional learning for domain adaptation of semantic segmentation," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).* IEEE, Jun. 2019.

[93] X. Mao, Q. Li, H. Xie, R. Y. K. Lau, Z. Wang, and S. P. Smolley, "On the effectiveness of least squares generative adversarial networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 41, no. 12, pp. 2947–2960, Dec. 2019.

[94] Y. Taigman, A. Polyak, and L. Wolf, "Unsupervised Cross-Domain image generation," Nov. 2016.

[95] Natural Resources Canada, "Natural resources canada," https://www.nrcan.gc.ca/home, Jul. 2016, accessed: 2022-2-3.

[96] https://www.loc.gov/preservation/digital/formats/fdd/fdd000279.shtml., accessed: 2022-2-3.

[97] "Dstl satellite imagery feature detection," https://kaggle.com/c/dstl-satellite-imagery-feature-detection, accessed: 2022-2-3.

[98] "Defence science and technology laboratory," https://www.gov.uk/government/organisations/defence-science-and-technology-laboratory, accessed: 2022-2-3.

[99] X. Meng, H. Shen, H. Li, L. Zhang, and R. Fu, "Review of the pansharpening methods for remote sensing images based on the idea of meta-analysis: Practical discussion and challenges," *Inf. Fusion*, vol. 46, pp. 102–113, Mar. 2019.

[100] J. Herring, *Opengis® implementation standard for geographic information-simple feature access-part 1: Common architecture*, 2011.