4-29-2019

Deep Learning Methods for 3D Aerial and Satellite Data

Mohammed A. Yousefhussien
mxy7332@rit.edu

Deep Learning Methods for 3D Aerial and Satellite Data

by

Mohammed A. Yousefhussien

A dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy
in the Chester F. Carlson Center for Imaging Science
College of Science
Rochester Institute of Technology

April 29, 2019

Signature of the Author _____

Accepted by _____
Coordinator, Ph.D. Degree Program Date

CHESTER F. CARLSON CENTER FOR IMAGING SCIENCE

COLLEGE OF SCIENCE

ROCHESTER INSTITUTE OF TECHNOLOGY

ROCHESTER, NEW YORK

CERTIFICATE OF APPROVAL

Ph.D. DEGREE DISSERTATION

The Ph.D. Degree Dissertation of Mohammed A. Yousefhussien
has been examined and approved by the
dissertation committee as satisfactory for the
dissertation required for the
Ph.D. degree in Imaging Science

_____

Dr. Carl Salvaggio, Dissertation Advisor

_____

Dr. Hossein ShahMohamad, External Chair

_____

Dr. Dave Messinger

_____

Dr. Eli Saber

_____

Dr. David Kelbe

_____

Date

Deep Learning Methods for 3D Aerial and Satellite Data

by

Mohammed A. Yousefhussien

Submitted to the
Chester F. Carlson Center for Imaging Science
in partial fulfillment of the requirements
for the Doctor of Philosophy Degree
at the Rochester Institute of Technology

Abstract

Recent advances in digital electronics have led to an overabundance of observations from electro-optical (EO) imaging sensors spanning high spatial, spectral and temporal resolution. This unprecedented volume, variety, and velocity is overwhelming our capacity to manage and translate that data into actionable information. Although decades of image processing research have taken the human out of the loop for many important tasks, the human analyst is still an irreplaceable link in the image exploitation chain, especially for more complex tasks requiring contextual understanding, memory, discernment, and learning. If knowledge discovery is to keep pace with the growing availability of data, new processing paradigms are needed in order to automate the analysis of earth observation imagery and ease the burden of manual interpretation.

To address this gap, this dissertation advances fundamental and applied research in deep learning for aerial and satellite imagery. We show how deep learning—a computational model inspired by the human brain—can be used for (1) tracking, (2) classifying, and (3) modeling from a variety of data sources including full-motion video (FMV), Light Detection and Ranging (LiDAR), and stereo photogrammetry. First we assess the ability of a bio-inspired tracking method to track small targets using aerial videos. The tracker uses three kinds of saliency maps: appearance, location, and motion. Our approach achieves the best overall performance, including being the only method capable of handling long-term occlusions. Second, we evaluate the classification accuracy of a multi-scale fully convolutional network to label individual points in LiDAR data. Our method uses only the 3D-coordinates and corresponding low-dimensional spectral features for each point. Evaluated using the ISPRS 3D Semantic Labeling Contest, our method scored second place with an overall accuracy of 81.6%. Finally, we validate the prediction capability of our neighborhood-aware network to model the bare-earth surface of LiDAR and stereo photogrammetry point clouds. The network bypasses traditionally-used ground classifications and seamlessly integrate neighborhood features with point-wise and global features to predict a per point Digital Terrain Model (DTM). We compare our results with two widely used softwares for

DTM extraction, ENVI and LAStools. Together, these efforts have the potential to alleviate the manual burden associated with some of the most challenging and time-consuming geospatial processing tasks, with implications for improving our response to issues of global security, emergency management, and disaster response.

Contents

1 Introduction **8**
- 1.1 Motivation . . . 8
- 1.2 Problem Statements and Research Questions . . . 9
  - 1.2.1 Task 1: Aerial target tracking . . . 10
  - 1.2.2 Task 2: 3D point cloud labeling . . . 11
  - 1.2.3 Task 3: Digital Terrain Modeling . . . 11
- 1.3 Organization . . . 12

2 Background **13**
- 2.1 Review of deep learning . . . 13
- 2.2 Review of target tracking . . . 19
  - 2.2.1 Brain regions for smooth pursuit . . . 20
  - 2.2.2 Algorithms for visual object tracking . . . 21
  - 2.2.3 Related deep CNN and saliency trackers . . . 21
- 2.3 Review of 3D point cloud labeling . . . 22
  - 2.3.1 Direct methods . . . 23
  - 2.3.2 Indirect methods . . . 24
- 2.4 Review of digital terrain modeling . . . 26
- 2.5 Summary . . . 29

3 Task 1: Aerial target tracking **30**
- 3.1 The smooth pursuit tracking algorithm . . . 31
  - 3.1.1 Appearance saliency . . . 32
  - 3.1.2 Location saliency . . . 34
  - 3.1.3 Motion saliency . . . 35
  - 3.1.4 Getting Boxes from the Smooth Pursuit Map . . . 35
  - 3.1.5 Handling occlusion . . . 35
- 3.2 Results . . . 36

3.2.1 Tracking individual objects . . . 37
3.2.2 Simultaneously tracking multiple objects . . . 39
3.2.3 Appearance vs. location vs. motion . . . 40
3.2.4 Timing benchmarks . . . 41
3.3 Conclusions . . . 42

4 Task 2: Point cloud labeling 43
4.1 Methodology . . . 45
4.1.1 Adaptation of CNN to Point Clouds . . . 45
4.1.2 Batch normalization . . . 47
4.1.3 Contextual feature learning . . . 48
4.1.4 Inference . . . 49
4.2 Evaluation . . . 49
4.2.1 Dataset . . . 50
4.2.2 Preprocessing . . . 51
4.2.3 Training Parameters . . . 52
4.2.4 Classification Results . . . 53
4.2.5 Effect of Individual Features . . . 58
4.2.6 Extension to 2D Semantic Labeling . . . 60
4.3 Conclusions . . . 63

5 Task 3: Digital Terrain Modeling 64
5.1 Methodology . . . 66
5.1.1 Network Architecture . . . 68
5.1.2 Learning Neighborhood Features . . . 69
5.1.3 Inference . . . 70
5.2 Evaluation . . . 71
5.2.1 Dataset . . . 71
5.2.2 Preprocessing . . . 73
5.2.3 Training Parameters . . . 75
5.2.4 DTM Results . . . 76
5.3 Discussion . . . 78
5.3.1 Effect of Neighborhood Size . . . 84
5.4 Conclusions . . . 85

6 Summary and Research plan 86
6.1 Conclusion . . . 86
6.2 Main Contributions . . . 87
6.3 Future work . . . 88

Appendices **102**

A List of publications **103**

Chapter 1

Introduction

1.1 Motivation

Recent advances in digital electronics, computing power, and low-cost imaging sensor platforms have led to an exponential growth in the availability of aerial and satellite-based imagery, video, and other data [1]. For example, Planet, an American small-satellite company, has launched a total of 149 satellites in the past five years and is now imaging the Earth's entire landmass (nearly 150 million square kilometers) at 3m resolution every single day [2]. The National Geospatial Intelligence Agency (NGA) collects the equivalent of three NFL seasons recorded in high-definition video, from a single sensor, every single day [3]. And in the time it took to read this sentence, the National Aeuronautics and Space Administration (NASA) has downloaded 1.73 GB of data from active missions [4]. In total, the global archive of satellite imagery is estimated to exceed one Exabyte [1], a figure which is growing by at least 20% every year [5].

This unprecedented volume, variety, and velocity of digital data now presents significant [3] challenges to the effective management, processing, and interpretation of these data towards actionable information products [1]. For example, an overabundance of remote sensing data following a disaster can paralyze traditional humanitarian response efforts that are more adept in dealing with information scarcity than overflow [6]. And if the NGA were to manually exploit all of the imagery they collect over the next 20 years, they would require 8 million imagery analysts [3]. In a variety of domains ranging from security and intelligence to humanitarian assistance and disaster response, the data from remote sensing platforms is straining our analytic capacity. In this era of geospatial "big data"—that is, spatial data sets exceeding the capacity of current computing systems [5]— new processing paradigms are needed in order analyze, interpret, and extract actionable information from the raw data volume.

As we think ahead to developing new machine-assisted methods for quantitative analysis of overhead imagery, it is important to align this research with the long history and trajectory of

remote sensing image exploitation. From the inception of aerial imaging through the development of quantitative computing, the field of *photo interpretation* has remained a key component to extracting information from overhead imagery [7]. Photo interpretation—the act of examining photographic images for the purpose of identifying objects and judging their significance [8]–relies on human visual cues such as shape, size, tone, texture, pattern, shadow, and site [9] to infer information about an object. An important note is that this requires some sort of prior learning process—both for the human and machine [7]. Recognition of an overhead image of a baseball diamond would be very difficult for someone who had never seen these structures before. In short, the ability of the human analyst to combine these visual cues with a lifetime of experience make them irreplaceable in the image exploitation chain [7].

On the other hand, human analysts require training and experience; they are expensive and in scarce supply; they are subject to fatigue, have difficulty combining multiple inputs simultaneously, and are poorly suited to large volumes of quantitative operations. If knowledge discovery is to keep pace with the growing availability of data, new processing paradigms are needed in order to automate the analysis of Earth observation imagery and ease the burden of manual interpretation. The research proposed herein therefore aims to address this mounting technological gap between data and its dependence on complex image interpretation—a task that cannot be met by human or machine alone. We leverage both the quantitative power of high-performance, hardware-accelerated computing, and the learning capacity of the human-inspired cognitive system to advance fundamental and applied research in deep learning for aerial and satellite imagery.

In this dissertation, we will discuss challenges and propose deep-learning solutions for three complex tasks: tracking, classification, and modeling from overhead data sources. First we assess the ability of a bio-inspired tracking method to track small targets using aerial videos. Second, we evaluate the classification accuracy of a multi-scale fully convolutional network to label individual points in Light Detection and Ranging (LiDAR) data. Finally, we validate the prediction capability of our neighborhood-aware network to model the bare-Earth surface of LiDAR and stereo photogrammetry point clouds. Together, these represent some of the most challenging and time-consuming geospatial processing tasks; automation through the recent advances in deep learning and computer vision offer the potential to alleviate the manual burden and more effectively take advantage of the geospatial data explosion.

1.2 Problem Statements and Research Questions

Target tracking, 3D point cloud labeling, and bare-Earth surface modeling are some of the most labor-intensive tasks for the image analyst. As opposed to quantitative photogrammetry or radiometry, these tasks require complex, high-level cognitive and contextual learning. These and other inherent challenges have so for precluded machine-assisted automation, which is a critical step to reducing the data burden on existing operational infrastructures. The following sections

will identify the specific challenges for each of our three tasks and propose three corresponding solutions to address these needs using deep learning.

1.2.1 Task 1: Aerial target tracking

Target tracking is the process of predicting the location of an object defined by a bounding box in the first frame. Challenges in target tracking include: change of appearance due to illumination and orientation, occlusion, and the existence of similar targets. Methods in the literature (see Section 2.2) mostly focus on improving features used to represent the target (the appearance model). However, the majority of videos used to evaluate the performance of such features include targets of high resolution (*i.e*. large number of pixels) [10], which are often not available due to size, weight, and power (SWaP) limitations. This is especially true for aerial target tracking applications from Unmanned Aerial Vehicles (UAV) video streams where collection constraints result in a relatively small number of pixels on the target. Without sufficient resolution, changes in lighting conditions, shadows, aircraft altitude, and perspective confound the tracking task.

To address these challenges, we answer the following questions: Is it possible to learn robust appearance features from relatively small number of pixel to handle changes in appearance from aerial video streams? Instead of relying solely on the target's appearance, are there other sources of information that can improve upon the state-of-the-art? How can we mitigate the negative impact of occlusion given the target's low resolution? Given these three questions, we propose following solutions to improve upon the state-of-the-art in aerial target tracking:

- **Task 1.1: Learn robust appearance features given the target's low resolution**. Unlike methods in the literature using SIFT, color histograms, or template patches as appearance models, we propose to use a convolutional neural network pre-trained using a large number of images as a feature extractor.

- **Task 1.2: Employ a selective search to constrain the search space and handle occlusion** Existing methods rely on a full image search, using either a computationally-expensive sliding window or the prediction of a Bayesian filter to search for the target when occluded. This is expensive and error-prone; selective search has the potential to minimize false alarms while increasing processing speed.

- **Task 1.3: Combine multiple sources of information to improve the tracking result**. Prior methods rely only on the target appearance to perform tracking. We propose the use of three types of information about that target and scene itself, namely the target appearance, current motion in the scene, and the possible future location of the target.

- **Task 1.4: Extend single-target tracking to multiple-target tracking**. Existing online trackers handle multi-traget tracking by running the same algorithm multiple times, which

is inefficient, redundant, and not scalable. We extend our single target tracker to handle the online tracking of multiple targets with little computational overhead.

1.2.2 Task 2: 3D point cloud labeling

A similar labor-intensive task to the human analyst is 3D semantic labeling of point clouds. This task seeks to attribute a semantic classification label to each 3D point. Classification labels, *e.g.* vegetation, building, road, etc., can subsequently be used to inform derivative processing efforts such as 3D modeling [11] and object detection [12]. Although great progress has been made in recent years for deep learning-based labeling of 2D imagery, LiDAR-derived point clouds are irregular, unstructured (*i.e.* unsegmented), and unordered. This makes semantic labeling of point cloud a challenging task, especially in urban scenes with different object types and spatial scales, ranging from small to large spatial neighborhoods (e.g., power lines to buildings).

In this dissertation, we answer the following research questions: Can we design a semantic labeling method that operates respects the inherent 3D nature of input point clouds? Can we design a model that is invariant to point clouds with varying density, irregular sampling, and unordered/unstructured data? Given the previous challenges in semantic classification, we propose the design of a multi-scale fully convolutional deep learning network that can:

- **Task 2.1: Consume unordered and unstructured point clouds directly**. Existing deep learning methods unnecessarily transform 3D points into 2D images or 3D volumes, we design a multi-scale deep network that respects the nature of 3D point clouds and mitigates the issue of non-uniform point density.
- **Task 2.2: Overcome the need to explicitly calculate contextual relationships**. Unlike methods that use computationally expensive algorithms, we use a simple layer that can learn a per-block global features during training.
- **Task 2.3: Perform multimodal fusion in 2D semanatic segmentation**. Instead of designing multiple methods for each modality, our method combines multiple inputs in a simple way.

1.2.3 Task 3: Digital Terrain Modeling

The third task aims to extend our previous work in Task 2 to learn the underlying Digital Terrain Model (DTM) from LiDAR and streo-photogrammetry derived point clouds in an end-to-end fashion. Extracting an accurate DTM is critical for many applications such as: city planning [13], agricultural surveying [14], post-disaster recovery [15], and flood mapping and assessment [16] to name a few. The existing large body of research in this field suggests solutions that requires a previous knowledge about the nature of the scene in order to tune the proposed solutions.

With an eye towards supporting these downstream we answer the following questions. Given the task of extracting the DTM from a 3D-point cloud: Is it possible to estimate the DTM without classifying points into ground and non-ground cover types? Can such a model operate directly on the point cloud instead of relying on derivative representations created from the data, such as images or TIN models? Can we eliminate the dependency on multiple parameters, while still incorporating spatial context, and thus reduce the time and effort spent on manual tuning?

Inspired by the success of our contributions in Task 2, we propose to design a neighborhood-aware deep network that can:

- **Task 3.1: Bypass the need for classifying ground points.** Existing methods have presented a variety of methods to generate the DTM by first detecting ground points using classical image processing methods or recently machine learning methods. Here we design a deep network that learns the underlying DTM in an end-to-end fashion bypassing the classification stage.
- **Task 3.2: Predict a point-wise DTM value directly.** Prior methods rely on intermediate steps, such as triangulated irregular network (TIN), to "sample" DTM values. Unlike prior methods, we estimate the point-wise DTM directly for every point in the scene.
- **Task 3.3: Integrate neighborhood information with point-wise and global features.** The features learned in Task 2 are limited to point-wise and global features. Here we build upon and extend our prior work to learn and seamlessly integrate neighborhood information with point-wise and global features.

1.3 Organization

This dissertation proposal is organized as follows. Chapter 2 reviews machine learning background necessary to understand deep learning concepts used throughout this dissertation. Also, it provides the literature background needed for each of our three subsequent tasks, which are discussed in detail in the following three chapters: In Chapter 3, we discuss a state-of-the-art solution for Task 1: tracking targets using aerial footage. In Chapter 4, we present a novel deep learning method for Task 2: 3D point cloud labeling using a multi-scale fully convolutional network. In Chapter 5, we demonstrate a novel neighborhood-aware network for Task 3: an end-to-end learning-based method for digital terrain model estimation. Finally, Chapter 6, will summarize our current findings and present plans for future/remaining work.

Chapter 2

Background

In this section we will introduce the background needed to present deep learning as the machine learning tool used throughout this dissertation proposal. Then, we will provide an in-depth literature review to contextualize each of our three tasks: (1) target tracking, (2) semantic classification, and (3) digital terrain modeling. Each of these tasks represent a complex, time-consuming process that has so far been challenging to automated due to the reliance on higher-level cognitive and contextual learning. However if successful, machine-assisted automation would alleviate the burden on human analysts and enable existing computational infrastructures to keep pace with, and fully exploit, the growing volume of airborne and satellite-based imagery.

2.1 Review of deep learning

Deep learning is an approach to machine learning that was initially inspired by our knowledge of the human brain, statistics and applied math, and has evolved over the past several decades. In recent years, deep learning has seen tremendous growth in its popularity and utility, largely due to more powerful computers, larger datasets and improved techniques to train deeper networks. Deep learning builds upon well-known, successful, and simple machine learning algorithms. Prior to deep learning, logistic regression and support vector classifier were the primary methods for supervised classification, while support vector regressor and least square methods were used for regression tasks. For example, Let $x^{(i)}$ be a the i-th input sample, then the task of binary classification is to predict the label $y^{(i)}$ such that $y^{(i)} \in \{0, 1\}$. On the other hand, the goal for a regression task is to predict the target $y^{(i)}$ where $y^{(i)} \in \mathbb{R}$. In regression tasks, a hypothesis $h_\theta(x)$ can be used to predict the underlying model that generated the data. Such hypothesis can simply be a polynomial in the form of $h_\theta(x) = \theta^\top x^{(i)}$ where θ is a parameter to be estimated. Intuitively, it doesn't make sense to use this hypothesis which outputs values larger than 1 and smaller than 0 for a classification task. To fix this, we can modify the hypothesis $h_\theta(x)$ non-linearly to squash it

into the range [0,1] using the sigmoid function as follows

$$h_\theta(x^{(i)}) = \sigma(\theta^\top x^{(i)}) = \frac{1}{1 + \exp(-\theta^\top x^{(i)})} \tag{2.1}$$

where $\theta^\top x^{(i)}$ is called the scoring function. Since the output values of this hypothesis are between 0 and 1, we can think of them as probabilities. Therefore, we can rewrite the previous equation as follows

$$p(y^{(i)} = 1|x^{(i)}) = h_\theta(x^{(i)}) \tag{2.2}$$

$$p(y^{(i)} = 0|x^{(i)}) = 1 - P(y^{(i)} = 1|x^{(i)}) = 1 - h_\theta(x^{(i)}) \tag{2.3}$$

which can be compactly written as

$$p(y^{(i)}|x^{(i)}; \theta) = (h_\theta(x^{(i)}))^{y^{(i)}}(1 - h_\theta(x^{(i)}))^{1-y^{(i)}} \tag{2.4}$$

Given the logistic regression hypothesis, we need to find θ that produces the correct output for a given input. Following the assumption we made in 2.2 and the compact form in 2.4, we can think of the logistic output $y^{(i)}$ as a Bernoulli random variable. Assuming that m training examples were generated independently, we can then write the likelihood of the parameters as

$$L(\theta) = p(y|X; \theta) \tag{2.5}$$

$$= \prod_{i=1}^{m} p(y^{(i)}|x^{(i)}; \theta) \tag{2.6}$$

$$= \prod_{i=1}^{m} (h_\theta(x^{(i)}))^{y^{(i)}}(1 - h_\theta(x^{(i)}))^{1-y^{(i)}} \tag{2.7}$$

Since it is easier to maximizes the log likelihood, the previous product can be simplified into a summation by applying the monotonically increasing natural log (written as ln) operation as follows

$$\ell(\theta) = \ln L(\theta) \tag{2.8}$$

$$= \sum_{i=1}^{m} y^{(i)} \ln(h_\theta(x^{(i)})) + (1 - y^{(i)}) \ln(1 - h_\theta(x^{(i)})) \tag{2.9}$$

where equation 2.9 now represents the cost function that measures how well a particular set of θ fit the hypothesis and maximizes the likelihood. Given the logistic regression cost function, we can

use a searching method (known as optimization algorithm) to find the unknown θ. One popular optimization algorithm is the gradient ascent update rule, which is posed as follows

$$\theta := \theta + \alpha \nabla_\theta \ell(\theta) \tag{2.10}$$

where $\nabla_\theta \ell(\theta)$ is the gradient of $\ell(\theta)$ with respect to the unknown θ. For a single example (x, y), the derivative of the cost function with respect to a particular element of θ is calculated as

$$\begin{aligned}
\frac{\partial}{\partial \theta_j}\ell(\theta) &= \left(y\frac{1}{\sigma(\theta^\top x)} - (1-y)\frac{1}{1-\sigma(\theta^\top x)} \right)\frac{\partial}{\partial \theta_j}\sigma(\theta^\top x) && (2.11)\\
&= \left(y\frac{1}{\sigma(\theta^\top x)} - (1-y)\frac{1}{1-\sigma(\theta^\top x)} \right)\sigma(\theta^\top x)(1-\sigma(\theta^\top x))\frac{\partial}{\partial \theta_j}\theta^\top x && (2.12)\\
&= \Big(y(1-\sigma(\theta^\top x)) - (1-y)\sigma(\theta^\top x) \Big) x_j && (2.13)\\
&= (y - h_\theta(x)) x_j && (2.14)
\end{aligned}$$

where we used the fact that $\sigma'(x) = \sigma(x)(1-\sigma(x))$. This therefore provides us with the stochastic gradient ascent update rule

$$\theta_j := \theta_j + \alpha(y^{(i)} - h_\theta(x^{(i)}))x_j \tag{2.15}$$

To generalize this framework to handle multiclass classification such that $y^{(i)} \in \{1, ..., K\}$ where K is the number of classes, softmax regression (known as multinomial logistic regression) is used. Given an input x, it is desired to have a hypothesis that estimates the probability $p(y = k|x)$ for each value of k. The output of such a hypothesis will be a K-dimensional vector (whose elements sum to 1) giving us our K estimated probabilities. To do so, we define a different set of θ for each class, which results in the following hypothesis:

$$h_\theta(x) = \begin{bmatrix} P(y=1|x;\theta) \\ P(y=2|x;\theta) \\ \vdots \\ P(y=K|x;\theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^{K} \exp(\theta^{(j)\top} x)} \begin{bmatrix} \exp(\theta^{(1)\top} x) \\ \exp(\theta^{(2)\top} x) \\ \vdots \\ \exp(\theta^{(K)\top} x) \end{bmatrix} \tag{2.16}$$

Here $\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(K)} \in \mathbb{R}^n$ are the parameters of our model. Notice that the term $\frac{1}{\sum_{j=1}^{K} \exp(\theta^{(j)\top} x)}$ normalizes the distribution, so that it sums to one. This setup shows that we constructed K classifiers each with a separate θ. Using the indicator function $1\{\cdot\}$, *i.e.* $1\{\text{a true statement}\} = 1$ and $1\{\text{a false statement}\} = 0$, we can rewrite the negative log likelihood (also known as the logistic regression cost function) in 2.9 as follows

$$\ell(\theta) = -\left[\sum_{i=1}^{m} \sum_{k=0}^{1} 1\left\{ y^{(i)} = k \right\} \ln p(y^{(i)} = k|x^{(i)};\theta) \right] \tag{2.17}$$

which can be easily extend to handle K classes by increasing the upper limit of the summation over classes as follows

$$\ell(\theta) = -\left[\sum_{i=1}^{m}\sum_{k=1}^{K} 1\left\{y^{(i)} = k\right\} \ln \frac{\exp(\theta^{(k)\top} x^{(i)})}{\sum_{j=1}^{K} \exp(\theta^{(j)\top} x^{(i)})}\right] \tag{2.18}$$

This is also known as a Categorical Cross Entropy loss function. While logistic regression and softmax regression provide elegant frameworks for classification, their decision boundaries are still linear (*i.e*. a line, a plane, or a hyperplane). Performance of these algorithms depends heavily on the representation of the data they are trained on. For example, when logistic regression is used to classify whether a patient suffers from a certain disease or not, it does not examine the patient directly. Instead, several pieces of relevant information (features) are used to represent the patient. logistic regression then tries to learn how to weight each of these features to provide the desired outcome. However, it cannot inuence how features are dened in any way.

In order to allow the decision surface to separate non-linearly distributed data, one can introduce non-linearity directly into the input space by transforming the raw data representations into higher dimensional features that are linearly separable in the high dimensional space. Such methods are known as $Kernels$. While increasing the dimensionality might transform the data into a space where they become linearly separable, it undesirably increases the complexity of the classification algorithm. Kernels are therefore not suitable for a large number of data samples.

To overcome the shortcomings of hand engineering features and work with large amount of data, a new method is desired. Neural Networks (NN) offer an elegant way of transforming the raw data into another space while maintaining the same complexity regardless of the number of data samples. This is done by leveraging computational precepts that mimic the human brain and neural system. While much about the human neural system is still unknown, in broad terms, a neural network is composed of a large number of highly interconnected processing elements (neurons) working simultaneously to transmit a signal from one to another. Similarly, a computational neural network consists of multiple connected layers, namely input, hidden, and output layers. The input layer is where the data of interest are presented to the network. The hidden layer is the heart of a neural network where a task-specific transformation of the input data is learned and passed to the output layer. Finally, the output layer processes the received representations and produces a prediction. A trained neural network can be thought of as an "expert" in the category of information it has been given to analyze. This expert can then be used to provide projections given new situations of interest.

Logistic regression and softmax regression can both be thought of as simple neural networks were the input layer is directly connected to the output layer. From another prospective, logistic regression output can be considered a single neuron in a bigger neural network. Figure 2.1 shows that by starting with logistic regression representation as a neuron, we can expand this representation to build softmax regression by stacking multiple logistic regression vertically, and build a

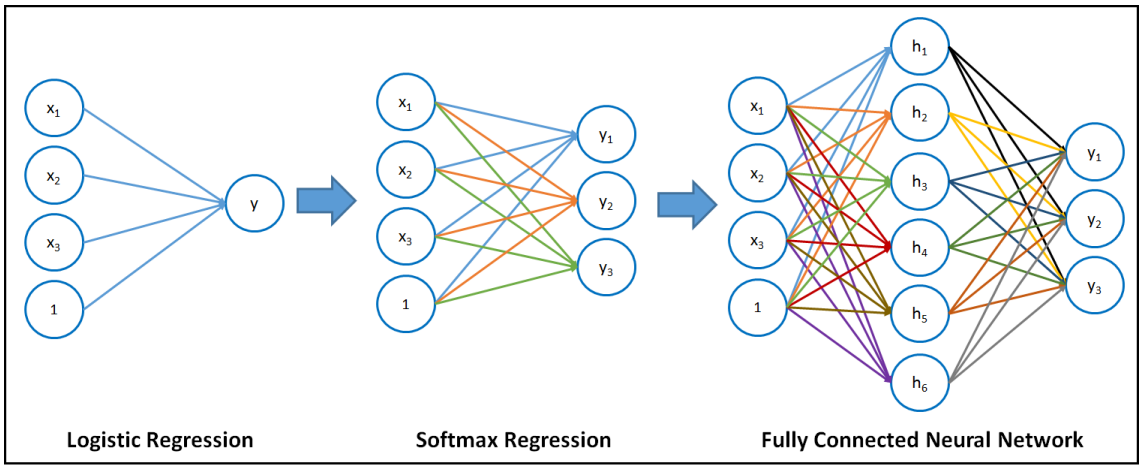

Figure 2.1: The figure shows logistic regression (right) modeled as a single output neuron. By stacking multiple logistic regression neurons vertically, softmax regression is created (middle). When the neurons are expanded both vertically and horizontaly, a fully connected neural network (left) is created.

neural network by stacking them horizontally afterwords. Similar to logistic regression, the connections between the neurons represent learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity transformation (e.g. sigmoid function). The whole network still expresses a single differentiable cost function as in 2.18 on the last softmax layer. When presenting an image to a fully-connected neural network for classification, typically this image is lexicographically ordered (*i.e.* unrolled) into a long vector. Each element in this vector is fully connected to each neuron within the hidden layer. If the input image is 32×32 pixels, and we have 1000 neurons in the hidden layer, the weight matrix will be 1024×1000 elements. While this might work well for small images, larger resolution images such as 512×512 pixels will result in a weight matrix of shape 262144×1000. This increase in required memory limits the use of neural networks to only handle smaller sized images. An additional challenge of this design is due to the fact that in an image, pixels closer to each other are generally correlated while those farther away are generally less correlated. Therefore, fully connecting all the pixels to a single neuron in the hidden layer doesn't take advantage of such locality.

Recently, deep learning [17, 18] methods apply a hierarchy of non-linear transforms to the data with the goal of generating an abstract, useful representation. A popular deep learning architecture for vision tasks is the convolutional neural network [17] (CNN). Inspired by the mammalian visual system, these neural networks contain convolution layers, non-linear functions, and pooling layers

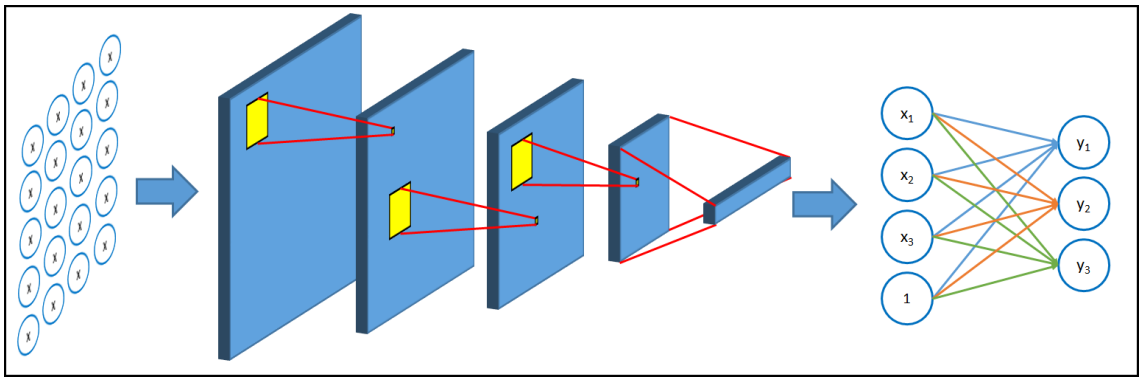

Figure 2.2: This figure shows that stacking input nodes from Figure 2.1(right) in the depth direction extends a neural network to handle two-dimensional input data such as images. Instead of working with individual nodes (left), inputs and outputs are arranged as two dimensional layers (middle). The yellow boxes represent the convolution kernel. The network's final representation is attached to a classifier (e.g. softmax) allowing a joint learning of both the kernels and the classifier.

for learning low-level to mid-level features in the beginning layers of a deep network. While fully connected neural networks operate on one-dimensional vectors as input, limiting them to handle only low resolution images, a CNN has the advantage of operating on raw 2D images without altering their original representation. Also, rather than being fully connected, A CNN allows each neuron in any hidden layer (known as *receptive fields* in this context) to be locally connected via shared weights to its input. Since the connections are local and the same across the input, the weights can be modeled as a convolution kernel; thus, allowing the network to learn shift invariant features, reducing the number of weights to be learned, and allowing the network to operate on large images. In order to learn hierarchical scale invariant features, pooling layers downsample incoming layers by computing simple statistics (e.g. mean, min, or max operations) within a small neighborhood. Figure 2.2 shows that by stacking the input nodes in the depth direction, one-dimensional neural networks can be extended to accept two-dimensional inputs such as images. Since pixels within a local neighborhood are correlated, local connectivity is used. The last representation of the network is a one-dimensional vector that can be used with a softmax classifier.

In summary, deep learning is a sub-field of machine learning that has proved its success in a variety of diverse domains. The main advantage of deep learning approaches is the ability to learn both the features and classifier/regressor jointly for a specific task. The following sections present the literature review for three specific tasks where deep learning can be utilized to alleviate the burden of manual image interpretation on human analysts. Chapters 3, 4 and 5 will present how we utilize deep learning to address challenges in (1) target tracking, (2) semantic classification,

and (3) digital terrain modeling.

2.2 Review of target tracking

In computer vision, target tracking is concerned with the analysis of video sequences for the goal of predicting the location of a single or multiple targets over a sequence of frames. In single target tracking (known as visual object tracking), the task is to predict a specific target's location during the sequence of a video starting from a bounding box given in the first frame. In this case, the tracking is formulated as a binary classification task predicting whether a region in the current frame contains a target or not. Single target trackers are also considered general purpose target trackers since the object of interest is manually selected in the first frame (*i.e.*any object can be selected as a target). On the other hand, multiple object trackers (also known as multiple hypotheses trackers) are "target-specific" offline trackers. Multiple Hypotheses Tracking (MHT) is one of the earliest successful algorithms for visual tracking [19]. It builds a tree of potential track hypotheses for each candidate target, thereby providing a systematic solution to the data association problem. The likelihood of each track is calculated and the most likely combination of tracks is selected. Importantly, MHT is ideally suited to exploiting higher-order information such as long-term motion and appearance models, since the entire track hypothesis can be considered when computing the likelihood. MHT has been popular in the radar target tracking community [20]. However, in visual tracking problems, it is generally considered to be slow and memory intensive, requiring many pruning tricks to be practical. In this case, the targets of interest are known before the video is observed. These algorithms start by using specialized object detection algorithms (e.g. detecting humans or cars) to detect all possible targets in the full sequence in an offline manner. Given a detection hypothesis in each frame, the tracker's task is to associate detections in the current frame with the correct detection from the previous and the following frames, thereby creating a tracking path that can be optimized globally as a post processing step. Our contribution in this thesis is mainly focused on the visual object tracking.

When tracking small moving objects, primates use two types of eye movements to keep the object foveated: smooth pursuit and saccades. Saccades are ballistic eye movements in which objects in the retinal periphery are brought to the center of the field of view. Smooth pursuit eye movements, in contrast, are continuous eye movements that attempt to counteract the object's movement to keep it continually in the center of the field of view. While saccades can be elicited in a wide variety of situations, smooth pursuit eye movements are only possible when an object is in motion [21]. Since most of the videos are recorded at 30 frames/second, it is assumed that most targets exhibits a slow motion between 2 frames. Thus, smooth pursuit presents a plausible biological inspiration for developing a tracking algorithm.

Motivated by the brain regions that underlie smooth pursuit, we created the Smooth Pursuit Tracking (SPT) algorithm that combines appearance, motion, and location information of a target.

We refer to these three types of information as saliency maps. Saliency maps are topologically organized maps that represent the salient information in a visual scene [22]. Algorithms for saliency maps have been widely used to model human attention and to predict eye movements.

There are two main kinds of saliency maps: bottom-up (or stimulus driven) and top-down (or task-driven). For static images, bottom-up maps assign higher saliency to rare features in the image, where the definition of rarity differs per algorithm. For videos, either feature rarity or motion are assigned high saliency. Bottom-up maps have been used to predict eye movements during free-viewing in static scenes and videos, when human subjects are not given a specific task [23, 24, 25]. Top-down maps assign high saliency to image regions that are likely relevant to the current task. When trying to look for particular targets, the process is often modeled by looking for locations where the object is likely to be and regions that resemble the target's appearance [26, 27, 28, 29]. In Chapter3, we use one kind of bottom-up saliency map (object motion), and two kinds of top-down saliency, (target appearance and future target location).

2.2.1 Brain regions for smooth pursuit

Many regions of the brain are involved in smooth pursuit. Among the most important are the middle temporal visual area (MT or V5) [30, 31], medial superior temporal area (MST) [30], lateral intraparietal sulcus (LIP) [32], and frontal eye fields (FEF) [32]. The MT receives information from earlier visual areas (V1 and V2), and nearly every cell responds to visual motion in a direction-sensitive manner [30]. Pursuit cells in the MT stop firing when blinking [33]. The MST receives input directly from the MT. During smooth pursuit, MST neurons respond to retinal image motion as well as head and eye movements [30], and many neurons continue to be active during blinking [33]. We interpret these brain regions as computing two kinds of saliency maps: MT is a map for target motion, and MST is a map that predicts the tracked object's future location.

LIP is a topologically-organized brain region that has been found to provide one of the most important saliency maps in the brain for visual attention [34]. LIP neurons indicate which regions of the visual field are relevant to the current task. LIP is involved in both smooth pursuit and saccades, and receives information from many visual areas involved in feature extraction and object recognition. We hypothesize that LIP is the source of the object appearance saliency map during tracking.

FEF is a topologically-organized brain region that produces the motor commands responsible for eye movements, including smooth pursuit. It is often considered the location of the brain's final saliency map for eye movements [35, 36]. It receives information from many brain regions, including the MT, MST, and LIP [37, 38]. V2, V4, and IT are all involved extracting visual features and understanding object appearance in an invariant manner, with IT being considered the location where high-level invariant object recognition occurs in the brain [39].We consider these regions as generating a map for the target object's appearance.

2.2.2 Algorithms for visual object tracking

There have been a large number of tracking algorithms developed in recent years (for a review, see [10] and [40]). The L1 tracker [41] poses the visual tracking problem in a particle filter framework. It tracks objects by finding a sparse approximation of the target in a template subspace, where the sparsity is achieved by solving an L1-regularized least squares problem. The L1-tracker incorporates a mechanism to handle occlusions. ASLA [42] uses a set of target templates to perform tracking. The ground-truth target is divided into a set of overlapping local image patches, which are then sampled to build a sparse dictionary representing the target's appearance. The target is then detected in subsequent frames using a dynamical Bayesian framework with an alignment pooling operator. The MIL tracker [43] uses an online variant of boosting that incorporates Multiple Instance Learning. Instead of using a single positive patch to update its classifier, it uses a collection of positive patches. The KCF Tracker [44] uses a dense sampling strategy around the given bounding box to produce a circulant matrix structure when kernels are applied. The algorithm uses a regularized least square classifier by labeling the training samples with continuous Gaussian values. Using the circulant structure of the matrix, the tracker uses the filter theorem to perform tracking in the frequency domain with high frame-rate. The CVT [45] integrates the color attribute model of [46] into the KCF [44]. CVT constructs a multi-dimensional color attribute vector by assigning observations in the RGB color space to linguistic color labels. The OAB tracker [47] uses an online version of AdaBoost to update a feature ensemble used for tracking the target. OAB uses the initial bounding box and the surrounding background as positive and negative examples. The weak classifiers use hand-crafted features. While most multiple-target tracking algorithms are used offline, SPOT [48] is one of the few online multi-target tracking algorithms. Using a structured SVM trained on features from the first frame, it jointly learns object appearance and structural constraints among the targets. The published implementation of SPOT requires multiple targets to be tracked, so it is only evaluated on our multi-target tracking video.

2.2.3 Related deep CNN and saliency trackers

In computer vision, there is currently a great interest in deep convolutional neural networks (CNNs) due to their efficacy at object recognition, object detection, semantic segmentation, and other areas. It is also possible to simply use CNNs as a kind of feature representation, treating the output of each convolutional layer as a collection of topologically organized and highly discriminative descriptors. A few recent papers have used CNNs or other deep learning algorithms to track objects in videos. In [49], the authors used a stacked denoising auto-encoder for tracking. During tracking, the algorithm draws particles as candidate locations from the new frame. The last layer of the auto-encoder was used to represent the features of the target and background, and a logistic regression classifier was trained to detect the target. However, they did not use motion information and their features were not trained to be discriminative for recognizing objects. In closely

related work, [50] performed target tracking by creating a target specific saliency map found by back-propagating SVM weights. The map was then used to generate an appearance model distribution in a Bayesian filtering framework. In [51], the authors proposed the discriminant saliency model. They first filter an image with DCT filters, and then each filter response is used to generate a saliency map. The target is detected at the location of the maximum value in the sum of the saliency maps. However, the authors did not use motion or location saliency. In [52], the authors developed a bottom-up saliency tracker that tracked any salient target in the scene, which was done in an entirely unsupervised manner. This is in contrast to the typical online tracking problem, in which a particular target is to be followed throughout a video. The algorithm uses color features and sparse optical flow to generate a bottom-up saliency map. The salient region in the frame is tracked using a particle filter. In [25], the authors used a ConvNet that is trained online to discriminate and track a target from the surrounding background. The algorithm proposes a truncated structural loss function to avoid the constraint of requiring large training data. In subsequent frames, the target is detected by classifying candidates using sliding window approach. The method did not use motion or location saliency. In [53], the authors used a ConvNet to track humans in videos. They trained the ConvNet to learn both spatial and temporal features together.

2.3 Review of 3D point cloud labeling

The past decade of computer/machine vision research and remote sensing hardware development has broadened the availability of 3D point cloud data through innovations in Light Detection and Ranging (LiDAR), Synthetic Aperture Radar (SAR), dense stereo- or multiview-photogrammetry and structure from motion (SfM). Despite the prevalence of 3D-point cloud data, automated interpretation and knowledge discovery from 3D-data remains challenging due to the irregular structure of raw point clouds. As such, exploitation has typically been limited to simple visualization and basic mensuration [54]. To overocome this, some authors rasterized the point cloud onto a more tractable 2.5D- Digital Surface Model (DSM) from which conventional image processing techniques are applied, *e.g.* [55, 56]. In order to generate exploitation-ready data products directly from the point cloud, semantic classification is desired. Similar to per-pixel image labeling, 3D-semantic labeling seeks to attribute a semantic classification label to each 3D-point. Classification labels, *e.g.* vegetation, building, road, etc., can subsequently be used to inform derivative processing efforts such as surface fitting [57], 3D modeling [11], object detection [12], and bare-Earth extraction [58]. However, the task of labeling every data point in the irregularly distributed point cloud captured by aerial platforms is challenging, especially in urban scenes with different object types and various scales ranging from very small spatial neighborhoods (power lines) to very large spatial neighborhoods (buildings). Moreover, point clouds are unstructured and unordered data with variable spatial densities. In order to scale the semantic classification task to meet the demands of emerging data volumes potentially at sub-meter resolution and global in coverage an

efficient, streamlined, and robust model that directly operates on 3D point clouds is needed.

Point cloud labeling algorithms can generally be grouped into two main categories. Section 2.3.1 describes "Direct Methods", which operate immediately on the point clouds themselves and do not change the 3D-nature of the data. Section 2.3.2 describes "Indirect Methods", which transform the input point cloud into an image or a volume as a preconditioning step to more traditional (raster-based) segmentation approaches.

2.3.1 Direct methods

Direct methods assign semantic labels to each element in the point cloud based on a simple point-wise discriminative model operating on point features. Such features, known as "eigen-features", are derived from the covariance matrix of a local neighborhood and provide information on the local geometry of the sampled surface, *e.g.* planarity, sphericity, linearity [59]. To improve classification, contextual information can explicitly be incorporated into the model. For example, [60] used covariance features at multiple scales found using the eigenentropy-based scale selection method [61] and evaluated four different classifiers using the ISPRS 3D Semantic Labeling Contest[1]. Their best-performing model used a Linear Discriminant Analysis (LDA) classifier in conjunction with various local geometric features. However, scalability of this model was limited due to the dependence upon various handcrafted features and the need to experiment with various models that don't incorporate contextual features and require effort to tune.

Motivated by the frequent availability of coincident 3D data and optical imagery, [62] proposed the use of point coordinates and spectral data directly, forming a per-point vector of (X,Y,Z,R,G,B) components. Labeling was achieved by filtering the scene into ground and non-ground points according to [63], then applying a 3D-region-growing segmentation to both sets to generate object proposals. Like [60], several geometric features were also derived, although specific details were not published. Without incorporating contextual features, each proposed segment was then classified according to the five classes from the ISPRS 3D Semantic Labeling Contest.

Alternatively, [64] classified full-waveform LiDAR data using a point-wise multiclass support vector machine (SVM). And [65] used random forests (RF) for feature detection and classification of urban scenes collected by airborne LiDAR. The reader is referred to [66] for a more complete review of discriminative classification models. While simple discriminative models are well-established, they are unable to consider interactions between 3D points.

To allow for spatial dependencies between object classes by considering labels of the local neighborhood, [67] proposed a contextual classification method based on Conditional Random Field (CRF). A inear and a random forest model were compared when used for both the unary and the pairwise potentials. By considering complex interactions between points, promising results were achieved, despite the added cost of computation speed: 3.4 minutes for testing using an RF

[1] `https://goo.gl/fSK6Fy`

model, and 81 minutes using the linear model. This computation time excludes the additional time needed to estimate the per-point, 131-dimensional feature vector prior to testing.

This contextual classification model was later extended to use a two-layer, hierarchical, high-order CRF, which incorporates spatial and semantic context [68]. The first layer operates on the point level, utilizing higher-order cliques and geometric features [61] to generate segments. The second layer operates on the generated segments, and therefore incorporates a larger spatial scale. Features included geometric- and intensity-based descriptors, in addition to distance and orientation to road features [69]. By iteratively propagating context between layers, incorrect classifications can be revised at later stages; this resulted in good performance on a 2.25 million point dataset of Hannover, Germany. However, this method employed multiple algorithms, each designed separately, which would make simultaneously optimization challenging. Also, the use of computationally-intensive inference methods limits the run-time performance. In contrast to relying on multiple individually-trained components, an end-to-end learning mechanism is desired.

2.3.2 Indirect methods

Indirect methods – which mostly rely on deep learning – offer the potential to learn local and global features in a streamlined, end-to-end fashion [70]. Driven by the reintroduction and improvement of Convolutional Neural Networks (CNNs) [71, 72], the availability of large-scale datasets [73], and the affordability of high-performance computing resources such as graphics processing units (GPUs), deep learning has enjoyed unprecedented popularity in recent years. This success in computer vision domains such as image labeling [74], object detection [75], semantic segmentation [76, 77], and target tracking [49, 78], has generated an interest in applying these frameworks for 3D classification.

However, the nonuniform and irregular nature of 3D-point clouds prevents a straightforward extension of 2D-CNNs, which were originally designed for raster imagery. Hence, initial deep learning approaches have have relied on *transforming* the 3D data into more tractable 2D images. For example, [79] rendered multiple synthetic "views" by placing a virtual camera around the 3D object. Rendered views were passed though replicas of the trained CNN, aggregated using a view-pooling layer, and then passed to another CNN to learn classification labels. Several other methods use the multiview approach with various modifications to the rendered views. For example, [80] generated depth images as the 2D views, while other methods accumulated a unique signature from multiple view features. Still other methods projected the 3D information into 36 channels, modifying AlexNet [74] to handle such input. For further details, the reader is referred to [81].

Similar multiview approaches have also been applied to ground-based LiDAR point clouds. For example, [82] generated a mesh from the Semantic3D Large-scale Point Cloud Classification Benchmark [83]; this allowed for the generation of synthetic 2D views based on both RGB information and a 3-channel depth composite. A two-stream SegNet [76] network was then fused

with residual correction [84] to label corresponding pixels. 2D labels were then back-projected to the point cloud to generate 3D semantic classification labels. Likewise, [85] generated multiple overhead views, embedded with elevation and density features, to assist with road detection from LiDAR data. A Fully-Convolutional Network (FCN) [77] was used for a single-scale binary semantic segmentation $\{$road, not-road$\}$ based on training from the KITTI dataset [86].

Despite their initial adoption, such multiview transformation approaches applied to point clouds lose information on the third spatial dimension through a projective rendering. Simultaneously, they introduce interpolation artifacts and void locations. Together, this complicates the process by unnecessarily rendering the data in 2D and forcing the network to ignore artificial regions caused by the voids. While this is less consequential for binary classification problems, multi-class problems require that each point be assigned to a separate class; this increases the complexity and may reduce the network's performance.

In light of these limitations of multiview transformation methods, other authors have taken a volumetric approach to handle points clouds using deep learning. For example, [87] presented a method for vehicle detection in ground-based LiDAR point clouds. The input point cloud was voxelized and then appended with a fourth binary channel representing the *binary occupancy*, *i.e*. the presence or the absence of a point within each voxel. Using the KITTI dataset, a 3D-FCN was then trained and evaluated to produce two maps representing the objectness and bounding box scores. Similarly, [88] generated occupancy voxel grids based on LiDAR point cloud data, labeling each voxel according to the annotation of its center point. A 3D-CNN was then trained to label each voxel into one of seven classes; individual points were then labeled according to their parent voxel. Other authors have explored variations of voxelization methods including a binary occupancy grid, a density grid, and a hit grid. In VoxNet, [89] tested each voxelization model individually, to train 3D-CNNs with 32x32x32 grid inputs. To handle multi-resolution inputs, they trained two separate networks, each receiving an occupancy grid with a different resolution.

Parallel development of both multiview and volumetric CNNs has resulted in an empirical performance gap. [90] suggested that results could collectively be improved by merging these two paradigms. To address this, a hybrid volumentric CNN was proposed, which used long anisotropic kernels to project the 3D-volume into a 2D-representation. Outputs were processed using an image-based CNN adapted from the Network In Network (NIN) architecture [91]. To combine the multiview approach with proposed volumetric methods, the 3D-object was rotated to generate different 3D-orientations. Each individual orientation was processed individually by the same network to generate 2D-representations, which were then pooled together and passed to the image-based CNN.

Finally, [92] took a different approach by combining image-like representations with a CRF. Instead of directly operating on the LiDAR data, they interpolated the DSM map as a separate channel. Using the imagery and the LiDAR data, two separate probability maps were generated. A pre-trained FCN was used to estimate the first probability map using optical imagery. Then, by

handcrafting another set of features from both the spectral and the DSM map, a logistic regression was applied to generate a second set of probability maps. At the end of this two-stream process, the two probability maps were combined using high-order CRF to label every pixel into one of six categories.

2.4 Review of digital terrain modeling

When 3D-point clouds from overhead sensors are used as input to remote sensing data exploitation pipelines, a large amount of effort is devoted to data preparation. Among the multiple stages of the preprocessing chain, estimating the Digital Terrain Model (DTM) model is considered to be of a high importance; however, this remains a challenge, especially for raw point clouds derived from optical imagery. An accurate DTM is critical for many applications such as: city planning [13], agricultural surveying [14], post-disaster recovery [15], and flood mapping and assessment [16], to name a few. In the past decades, researchers have presented a variety of methods to generate the DTM, generally taking one of the following approaches: slope-based, triangulated irregular network (TIN)-based analysis, morphological filtering of a raster DSM, or more recently, machine learning. Given the large body of research in this field, here we summarize a selection of previous contributions. We refer the reader to [93, 94, 95, 96] for comprehensive reviews.

In [97], the author presented an algorithm to classify ground points by analyzing the slope and distance relationships between a pair of nearby points. Given a point pair, the algorithm initially hypothesizes that a large height difference between them is unlikely to be caused by a steep slope in the terrain, and asserts that the probability that the higher point is ground should increase as the distance between the points increases. Using this hypothesis, a set of geometric rules and thresholds were designed to classify all points into ground and non-ground classes.

In the same year, [63] introduced a method based on an adaptive TIN model. Statistics such as minimum z-values within a cell are collected to seed an initial coarse TIN. The TIN model is then iteratively densified by including more points that satisfy a set of thresholds until no further points meet these criteria. The remaining points are considered non-ground points, while the points included within the TIN model are classified as ground.This method has become the base model for many commercial software packages such as TerraScan[2].

Rather than operating directly on the point cloud, other authors have opted to use a raster DSM. In [98] the authors presented a hierarchical multi-scale approach for the detection of non-terrain "pixels" in a DSM. First a raster image with a large cell element is created. In this coarse representation, a Laplacian-of-Gaussian (LoG) filter is used as a blob detector to remove large non-ground objects. Next, a finer raster image is created with previously-detected non-ground objects removed. The process is repeated, iteratively, until a desired resolution; the resulting raster

[2] `http://www.terrasolid.com/home.php`

is the DTM.

A similar raster-based analysis was presented by [99]. Based on the hypothesis that smooth ground (e.g., pavement) is present in urban and suburban areas, the author approximated local regions within a window by a planar surface. This eliminated all irregular points (e.g., trees, shrubs) not belonging to ground or rooftops. These planar surfaces were then analyzed with a connected component analysis, after transforming the point cloud into a 1m grid. Using an area threshold, smaller connected regions were considered non-ground points while larger components were classified as ground points and used to interpolate a DTM. While this method may work in flat areas, the planar approximation doesn't scale well for irregular terrain.

This idea was extended by [100], who used planar surface features and connectivity, along with local "low points" to improve bare-Earth modeling. Point cloud data is first rasterized, and then planar surfaces such as ground and rooftops are detected. Local surface normals are analyzed to remove pitched rooftops and other features, based on the assumption that ground surfaces fall within a certain slope threshold. Several heuristics are then applied, which add ground points to the DTM model if the local distance and slope relationships meet predefined thresholds. This slope-based paradigm was extended for photogrammetric point clouds by [101], although they required a low-resolution DTM to seed the ground point detection.

Although these methods have shown promise in the literature, a major limitation is the reliance on heuristics. These algorithms require the tuning of global, predefined parameters, such as slope and various distances. As a result, they often require human intervention to achieve optimal results. This is exemplified by the numerous options and parameters presented to the user in software packages such as LASTools [102] or ENVI [103].

In order to avoid heuristics based on slope or point-wise distances, another class of algorithms has proposed the use of morphological operations. This approach casts the problem as a image processing task, operating on a grayscale DSM "image" representing the height information.

For example, [104] proposed an algorithm to identify above-ground objects based on their size, height, and edge characteristics. First, a DSM is created from the point cloud, and then morphological gradients are calculated to analyze local elevation changes. Small objects near the ground surface are removed using a Modified White Top-Hat (MWTH) transform with directional edge constraints and a height difference threshold. This process is iterated at hierarchical levels with increasing window size to eliminate larger and larger objects.

[105] proposed a modification to the Top-Hat filter by introducing a sloped brim. The authors showed how this modification makes the method more robust for complex objects and terrain. Similar to the previous approach, internal and external gradients where found using a fixed structural element, and then a height-thresholded Top-Hat is used. For every (row, column) in the DSM, a pixel is determined to be ground if the difference between the original and the morphologically "opened" heights are not larger than a predefined threshold.

[106] used morphological reconstruction to estimate the DTM using geodesic dilation. This

process uses two input images termed a "marker" and "mask". The reconstruction procedure suppresses unwanted regions of high intensity (*i.e.*non-ground) while preserving the intensity of the regions of interest, which are seeded with "markers".The marker is dilated by an isotropic structuring element, while the mask acts as a limit for the dilated output. Dilation of the marker and point-wise minimum between the dilated image and the mask is applied iteratively until stability. The marker image, generated by subtracting an offset from the DSM image, is dilated by an isotropic structuring element, while the mask acts as a limit for the dilated output. After grayscale reconstruction, heights less than the offset values are removed, with the remaining heights representing the final DTM.

Alternatively, [107] proposed a robust surface interpolation method. A rough approximation of the surface is computed, and then the residuals *i.e*. the oriented distances from the surface to the measured points, are computed. Each point is weighted according to how far it is from the surface, and then the surface is recomputed based on these weights. A point with a high weight will attract the surface, resulting in a small residual, whereas a point that has been assigned a low weight will have little influence on the surface. This process is repeated iteratively, while simultaneously classifying points into ground/non-ground based on their distance from the surface.

Recently, a new class of methods have tried to move away from the manual fine-tuning of heuristic parameters. These learning-based methods propose to estimate the DTM by constructing a binary classifier that can identify ground and non-ground data, based on previously annotated scenes.

For example, [108] used a Convolutional Neural Network (CNN) to classify pixels as ground/not ground. Since CNN architectures are generally designed to work on RGB raster imagery, the authors created training images as follows: For each point, a (128×128) image patch was generated, i.e., a DSM patch with each pixel representing z-values. From each image patch, three channels were created by subtracting from the DSM patch the minimum, maximum, and mean z-values for that patch. The results are treated as surrogate red, green, and blue channels for a binary deep CNN framework designed to determine whether the point belongs to the ground or non-ground class. While the network performed well, it required a very large amount of training data (more than 17 million labeled samples). Furthermore, there is a signficiant amount of overhead, since every point was represented by a (128×128) window, and the label for the whole window must be transferred back to the corresponding 3D point.

Instead of classifying the input image into a ground and non-ground cover types, [109] formulated the problem of estimating the DTM as a semantic labeling task where each pixel in the input RGB image is labeled (road, car, tree, etc.). To create labeled training data, the corresponding DSM image is used with a Top-Hat filter to generate labeled data. Next, a Fully Convolutional Network (FCN) [77] framework is used for semantic labeling of the input. In order to avoid downsampling the input image, the authors suggested using dilation filters to capture the features at multiple scales. Once an image is segmented into semantic labels, the ground points are isolated

and used to interpolate a DTM based on traditional methods, e.g., kriging.

Alternatively, [110] used a deep Multi-Layer Perceptron (MLP) to separate ground from non-ground objects. The network's parameters where found using a random search. The authors claim an improvement when artificial modifications are introduced to the data.

Finally, [111] formulated the problem in an Encoder-Decoder framework. The authors developed a sparse autoencoder where the input is the original DSM and the output is the DTM. From this perspective, the input is treated as "noisy" data, whereas the DTM is considered to be the clean data. This approach bypasses the classification stage to predict the DTM directly from the input, similar to the approach we propose. However, the input data is limited to a scanline from the DSM, which doesn't capture the surrounding two-dimensional neighborhood information. Also, the autoencoder was constructed using a fully-connected architecture, which by design, does not capture local features.

2.5 Summary

In this chapter, we provided the machine learning background needed to introduce deep learning as a successful tool that can be applied across many domains. Next, we reviewed the literature for three specific tasks (target tracking, semantic classification, and digital terrain modeling) that can successfully leverage deep learning to advance the state-of-the-art in automated remote sensing data exploitation. In the following chapters, we will present our research for each of three state-of-the-art solutions that address existing challenges and leverage novel deep learning techniques to improve upon existing methods.

Chapter 3

Task 1: Aerial target tracking

The first task we will address through biologically-inspired algorithm is the task of target tracking from aerial full motion video. Tracking in aerial footage is becoming increasingly important. Unmanned aerial vehicles (UAVs) continue to become more popular every year, and they are now being used in film production, mining, construction, real estate, news media, and agriculture. Moreover, the world's security agencies are gathering enormous amounts of video data in which they are looking for events of interest, such as suspicious vehicles. When these vehicles of interest are found, in many cases they would like a UAV to follow the vehicle over time; thus requiring online visual tracking. However, tracking ground vehicles in UAV video streams is especially difficult because of the relatively small number of pixels on the target. Furthermore, targets can change drastically in appearance due to variation in lighting conditions, UAV altitude, and perspective differences. As a result, target tracking, though straightforward for the human observer, has proved very difficult to reliably automate. We address this gap by developing a deep learning algorithm inspired by the human visual system. This inspiration is based on the observation that when tracking small moving objects, e.g. distant target such as cars seen from airborne videos, primates use two types of eye movements to keep the object foveated: smooth pursuit and saccades.

Motivated by the brain regions that underlie smooth pursuit (see Section 2.2.1), and the success of deep learning as feature extractors, we developed the Smooth Pursuit Tracking (SPT) algorithm. We also show that SPT can do online tracking of multiple targets, which requires little additional overhead compared to single-target tracking. The following sections describe this state-of-the-art algorithm that utilizes deep features to track moving vehicles in videos acquired by an aerial platform.

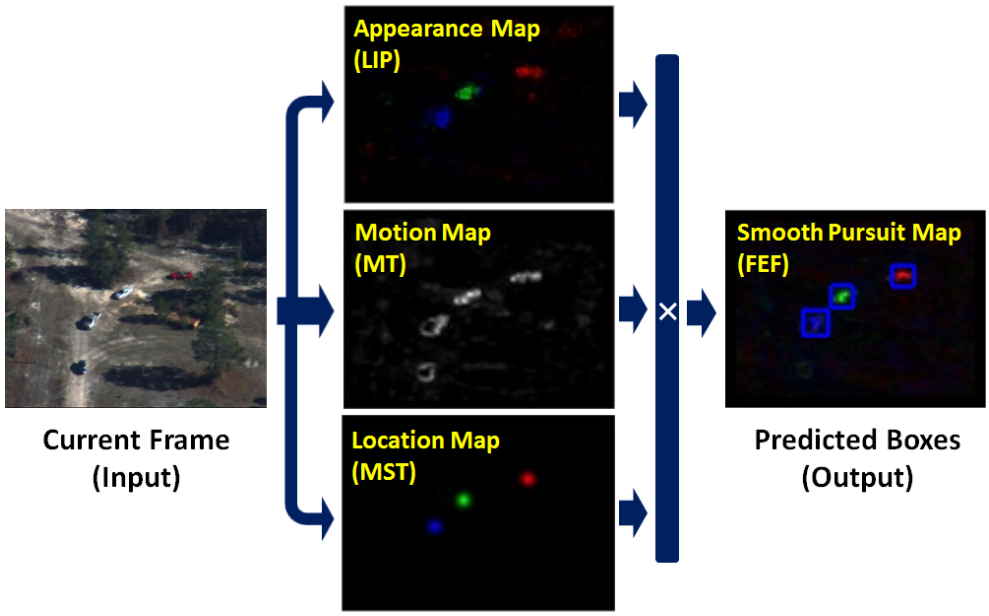

Figure 3.1: In this Chapter we present a new algorithm for online tracking, which combines three kinds of saliency maps: appearance, motion, and location. The algorithm can track multiple objects simultaneously with little additional overhead. We evaluate the algorithm on its ability to track ground vehicles in aerial video footage.

3.1 The smooth pursuit tracking algorithm

Our SPT algorithm multiplicatively combines appearance, location, and motion saliency maps to generate a smooth pursuit map. We assume that each of the sub-saliency maps contains only a few salient regions, so the multiplicative combination of them should produce a sparse map with the most salient region containing the target. Formally, to track a target k, SPT defines the smooth pursuit saliency map s as

$$s\left(k,t,z\right) = a\left(k,t,z\right)\ell\left(k,t,z\right)m\left(t,z\right), \tag{3.1}$$

where z is the (x,y) Cartesian pixel location, t represents the frame number, a is a function that computes the appearance saliency map, ℓ is a function that computes the location saliency map, and m is a function that computes the motion saliency map. This equation is depicted in Fig. 3.1.

We explain how each of these saliency maps are computed in the following subsections, and example saliency maps are shown in Fig. 3.2. As explained in Section 3.1.5, our algorithm uses an alternative technique when its confidence is low, enabling it to handle occlusions.

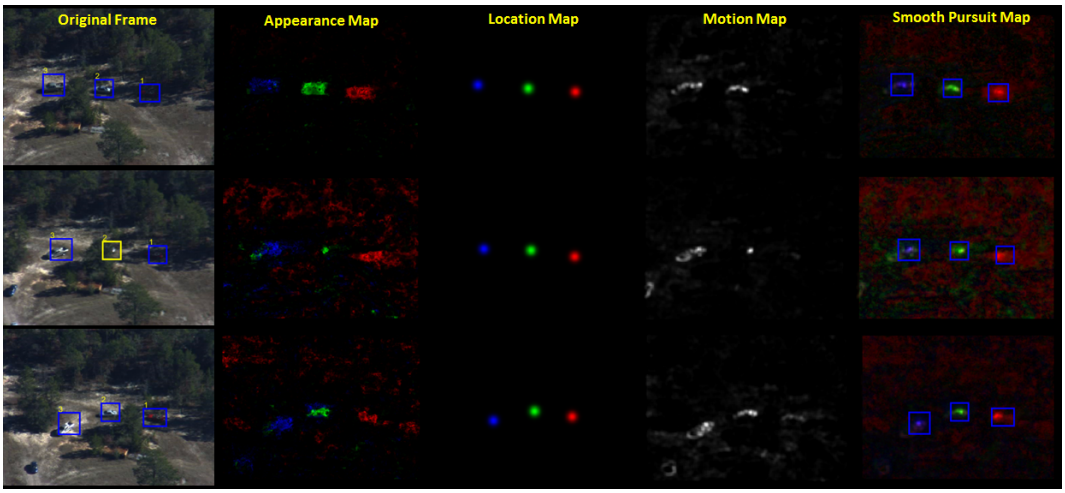

Figure 3.2: The appearance, location, motion, and smooth pursuit saliency maps for three frames of a video, where we are overlaying saliency maps for each of the three vehicle targets using different colors (blue, green, and red) along with the current bounding box prediction.

3.1.1 Appearance saliency

Top-down saliency maps use information about the algorithm's goal to generate the saliency map. When a person is trying to look for a particular kind of object, top-down saliency maps driven by object appearance are far more predictive than bottom-up models [27]. These saliency maps have high values in regions that correspond to areas of the image that resemble the target. Here, we created a new appearance-based saliency algorithm using an online variant of gnostic fields and features extracted from deep convolutional networks. A gnostic field consists of competing gnostic sets, with each set containing a population of gnostic neurons that act as template matchers for a particular class. More recently, gnostic fields achieved good results on various object recognition benchmarks [112, 113]. Gnostic fields generally operate on dense topologically organized image descriptors, and each gnostic set compares each descriptor to the learned templates for the class. We have adapted this object recognition model to serve as our appearance saliency map for each tracked target. We use one gnostic field for each feature channel c (*i.e.*, feature type), which can serve as a classifier for all K targets of interest. A gnostic field first applies a whitening matrix $\mathbf{W}_c$ to the features $\mathbf{g}_{c,t,z}$, which decorrelates and reduces the dimensionality of the features, followed by making these features unit length, *i.e.*,

$$\mathbf{f}_{c,t,z} = \frac{\mathbf{W}_c \mathbf{g}_{c,t,z}}{\| \mathbf{W}_c \mathbf{g}_{c,t,z} \|}. \tag{3.2}$$

For each channel, $\mathbf{W}_c$ is computed using whitened principal components analysis applied to the descriptors extracted in the first video frame, and only the first 100 principal components are retained. To track K objects, we create a gnostic field with $K+1$ gnostic sets, with one set per object plus one set representing the background. The initial activity of each gnostic set is given by the most active unit in the set, *i.e.*,

$$a_{c,k,t,z} = \max_j (\hat{\mathbf{v}}_{c,k,j} \cdot \mathbf{f}_{c,t,z}), \tag{3.3}$$

where each $\hat{\mathbf{v}}_{c,k,j}$ represents gnostic unit j for set k that has been normalized to unit length, *i.e.*, $\hat{\mathbf{v}}_{c,k,j} = \frac{\mathbf{v}_{c,k,j}}{\|\mathbf{v}_{c,k,j}\|}$. Subsequently, we competitively normalize the responses of the gnostic sets,

$$q_{c,k,t,z} = \max\left(a_{c,k,t,z} - \theta_{c,t,z}, 0\right), \tag{3.4}$$

where $\theta_{c,t,z} = \frac{1}{K+1} \sum_{k'} a_{c,k',t,z}$. The non-zero responses are then normalized using

$$\beta_{c,k,t,z} = q_{c,k,t,z} \frac{\sum_{k'} q_{c,k',t,z}}{(r + \sum_{k'} q^2_{c,k',t,z})^{3/2}}, \tag{3.5}$$

where $r = .01$ regularizes the strength of the normalization. This competitive normalization step was found to be crucial for ensuring good object recognition accuracy in [112]. We then bilinearly interpolate $\beta_{c,k,t,z}$ to align it with and make it the same size as the input frame at time t, which yields $\beta'_{c,k,t,z}$. To create the final appearance map for the target, these are summed across all channels and then the map is normalized across spatial locations to sum to one, *i.e.*,

$$a\left(k,t,z\right) = \frac{\sum_c \beta'_{c,k,t,z}}{\sum_z \sum_c \beta'_{c,k,t,z}}. \tag{3.6}$$

Earlier papers on gnostic fields were trained offline and used spherical k-means to learn the gnostic units. However, that approach is not suitable for online tracking, so we created an online version of spherical k-means, in which the number of units is allowed to increase. To incorporate a descriptor $\mathbf{f}_{c,h}$ into the gnostic set corresponding to channel c and tracked object k, we do the following. If the gnostic set contains one or more units, then the descriptor is compared to all of the set's units to find the most active unit j', *i.e.*,

$$j' = \arg\max_j \hat{\mathbf{v}}_{c,k,j} \cdot \mathbf{f}_{c,h}. \tag{3.7}$$

If $\hat{\mathbf{v}}_{c,k,j'} \cdot \mathbf{f}_{c,h} > \rho$, where ρ acts as a similarity threshold, or if $\psi_{c,k} = \lambda$, where $\psi_{c,k}$ represents the current number of units in the gnostic set and λ represents the maximum number of units allowed, then we update unit j' using

$$\mathbf{v}_{c,k,j'} = \alpha \mathbf{f}_{c,h} + (1-\alpha)\, \mathbf{v}_{c,k,j'}, \tag{3.8}$$

where α controls the balance between the old and new representations. This update rule is based on the exponential moving average, so it could potentially forget object appearances in the distant past. Otherwise, the descriptor is simply copied into the set as an additional unit. In our experiments, we set $\lambda = 1000$, $\alpha = 0.5$, $\rho = 0.98$ for the tracked targets, and $\rho = 0.9$ for the background set. No attempt was made to tune these parameter values.

In the first frame, the gnostic sets for target k are initialized using all of the descriptors within the ground truth bounding box. All descriptors not assigned to targets are used to initialize the background gnostic set. In subsequent frames, we run a Harris corner detector and we only update the target gnostic sets with descriptors at the corners found within the predicted target bounding boxes, and we use descriptors that are not located within any predicted target boxes to update the background gnostic set.

For image descriptors, we used multiple layers of a ConvNet that was pre-trained on ImageNet. Specifically, we used the 16-layer ConvNet known as VGG16 [114], which has 13 convolutional layers and 3 fully-connected layers. The fully-connected layers were not used. We convolved the network with each video frame, and used the output of convolutional layers 4, 7, 10, and 13 as distinct channels that were input to four gnostic fields. Respectively, these descriptors were 128–, 256–, 512–, and 512–dimensional, and for 640×480 images (as in our dataset), each respective layer produced 16384, 4096, 1024, and 256 topologically organized descriptors. For the descriptors, all negative values were set to zero prior to feeding them to the gnostic fields (*i.e.*, applying the rectified linear unit non-linearity). We used the MatConvNet toolbox [115] to run the network.

3.1.2 Location saliency

For each tracked object k in a video, we estimate the location saliency map $\ell\,(k, t, z)$ using the Kalman filter prediction stage. The Kalman filter is applied to generate the location map, and it is not used as the main tracking framework. The filter predicts the future location of the target k using a constant velocity motion model, based on online updates of its past locations. In the first frame, the center location of the target's bounding box is used to initialize the Kalman filter. In subsequent frames, if the confidence in the target's position is sufficiently high (see Sec. 3.1.5), the Kalman filter is updated using the new location, and then the prediction stage is used again to estimate the location in the next frame. However, if the confidence is low, *e.g.*, due to an occlusion, Kalman filter will not be updated and the prediction stage of the last updated model will be used. It should be noted that We only used the Kalman filter to predict the target's central location and not the size of the target. Instead, we used a spherical 2-dimensional Gaussian centered at that location with a fixed variance of $\sigma^2 = 300$, which was chosen in preliminary experiments.

3.1.3 Motion saliency

One of the key characteristics of smooth pursuit is that it can only be used to track moving objects. We capture object movement in frame t using a motion (or spatiotemporal) saliency map $m(t, z)$, which uses a simple background subtraction algorithm. More sophisticated spatiotemporal saliency map algorithms are possible (e.g., [116]), but we have chosen to adopt this simple approach because of its speed and ability to compensate for camera motion. The motion saliency map is created by performing background subtraction using an average model of the previous n frames as the background, *i.e.*,

$$m'(t, z) = I_{t,z} - \frac{1}{n}\sum_{j=1}^{n} I'_{t-j,z}, \tag{3.9}$$

where $I_{t,z}$ is the current frame and $I'_{j,z}$ is a version of the frame at time j that has been aligned to the frame at time t. This representation is normalized by subtracting the mean across locations and doing half-wave rectification, *i.e.*,

$$m(t, z) = \max\left(m'(t, z) - \theta(t, z), 0\right) + \epsilon, \tag{3.10}$$

where $\theta = \frac{1}{n}\sum_{z'} m'(t, z')$ and $\epsilon = .01$ to ensure that if the tracked objects have stopped moving that the smooth pursuit map will not be null. Subsequently, $m(t, z)$ is normalized to sum to one. Aligning the previous frames to the current frame is necessary to counteract camera motion. We did this alignment using the MATLAB Image Alignment Toolbox [117]. We configured the toolbox to find corresponding points between images using SURF descriptors and a RANSAC fitting was used to estimate the transformation between the correspondences. In our experiments, we set $n = 3$ in general, but if less than n frames were available then all of the frames were used. The versions of the images used in the background subtraction are in an opponent color-space [118].

3.1.4 Getting Boxes from the Smooth Pursuit Map

To generate bounding boxes from the smooth pursuit map for a target k in frame t, we find the location of the largest value in the saliency map, *i.e.*, $z' = \arg\max_z s(k, t, z)$. Then we threshold the map to set all values less than $\frac{1}{3}s(k, z', t)$ to zero and setting all other points to one. Finally, we find the connected components about point z' and use that to generate the predicted bounding box $B_{k,t}$.

3.1.5 Handling occlusion

Smooth pursuit eye movement only works when the tracked object is moving and visible, meaning that other mechanisms is needed to recover a track that has been lost due to occlusion. Primates

use saccades to recover the object's location in this situation. Here, we use an object detection algorithm to try to do the same.

To determine if a track k has been lost in frame t we use the appearance saliency map prior to its final normalization by taking the mean around $B_{k,t}$, which is the bounding box predicted by the smooth pursuit map, *i.e.*,

$$\phi_{k,t}\left(B_{k,t}\right) = \frac{\sum_{z \in B_{k,t}} a\left(k,t,z\right)}{|B_{k,t}| \max_{z'} a\left(k,t,z'\right)}, \tag{3.11}$$

where $|B_{k,t}|$ contains the number of points in the bounding box. The greater $\phi_{k,t}$ is for $B_{k,t}$, the more confident the system that the box contains the target. If $\phi_{k,t}\left(B_{k,t}\right) < 0.4$ then the system may not use $B_{k,t}$. Instead, we run the Selective Search algorithm [119] on the appearance and temporal saliency maps to generate bounding box hypotheses. Selective Search hierarchically segments the maps and generates a bounding box hypotheses. We prune the box hypotheses to constrain their centers to be within $10p$ pixels, where p is the number of frames in which confidence was low, *i.e.*, $\phi_{k,t} < 0.4$ has been true. This allows the algorithm to search for boxes farther away as the length of time the track has been lost increases. For each box hypothesis we evaluate equation 3.11. Let the box hypothesis with the greatest confidence be $B'_{k,t}$. If the confidence in $B'_{k,t}$ is greater than 0.4, then we use it instead of $B_{k,t}$.

3.2 Results

Here we present the results of our algorithm on the VIVID dataset.We compare our method to: L1 tracker [41], CVT [45], ASLA [42], MIL [43], KCF [44], OAB [47], and SPOT [48]. There were no publicly available results using our evaluation metrics on the dataset we evaluated on, so we ran code for each of these algorithms ourselves. We briefly summarize how each of these online trackers works. With the exception of SPOT [48], each of these trackers is only designed to follow one object at a time. VIVID is a large video dataset captured by an aerial platform [120]. Each video contains multiple vehicles moving on dirt or paved roads. Many videos have occlusions, such as tree canopy, and others have identical vehicles moving in formation, making it especially challenging for methods that rely on appearance alone to succeed because the tracks can easily be confused. Only five RGB VIVID videos have been annotated by others, and these annotations only have ground-truth for a single vehicle in each video. We first show results on these single-vehicle tracking results. To study our approach's ability to track multiple vehicles simultaneously, we also hand-annotated three vehicles in a single VIVID video. We quantitatively compare our algorithm to the trackers mentioned in Section 2.2.2. All algorithms, including our own, had all of their meta-parameters fixed across all videos. All of our qualitative results are presented in videos[1].

[1] A list of links can be found in the supplementary materials, and a YouTube playlist with these videos can be found here: `https://goo.gl/7XVR9v`

Figure 3.3: Frames from three single-target tracking videos with predicted tracks for each algorithm. Each row is for Egtest01, Egtest03, and Egtest05 respectively, while the columns represent the selected frames. The bounding boxes are colored as follows: ground-truth, CVT, ASLA, L1, MIL, KCF, OAB, and Ours.

We measure the performance of the algorithms using recent standard metrics for online tracking [10]: precision plots, success plots, and center location error (CLE) plots. For success plots, the Area Under the Curve (AUC) is reported in the figure, and we also report the precision at a threshold of 20 in precision plots. The mean CLE for the whole video per tracker is also reported.

3.2.1 Tracking individual objects

Fig. 3.3 shows four example frames for three of the five videos in each row, Egtest01, Egtest03, and Egtest05, as well as predicted bounding boxes for each method. See supplementary materials for additional figures. For video Egtest01, all of the trackers were able to follow the target for the first 400 frames. By frame 420, other trackers start to drift, and only our method, ASLA and OAB continued tracking the target throughout the video. In contrast, for video Egtest03 (second row), ASLA performs poorly, while ours and OAB track well. The third row, video Egtest05, contains strong changes in illumination and has long occlusions, but our method tracked well throughout the video compared to all trackers. Table 3.1 shows the mean CLE for each video, as well as the average CLE across videos. Overall, our method performed significantly better than the other algorithms, and our overall error was the lowest by a margin of 117 pixels. While our method did

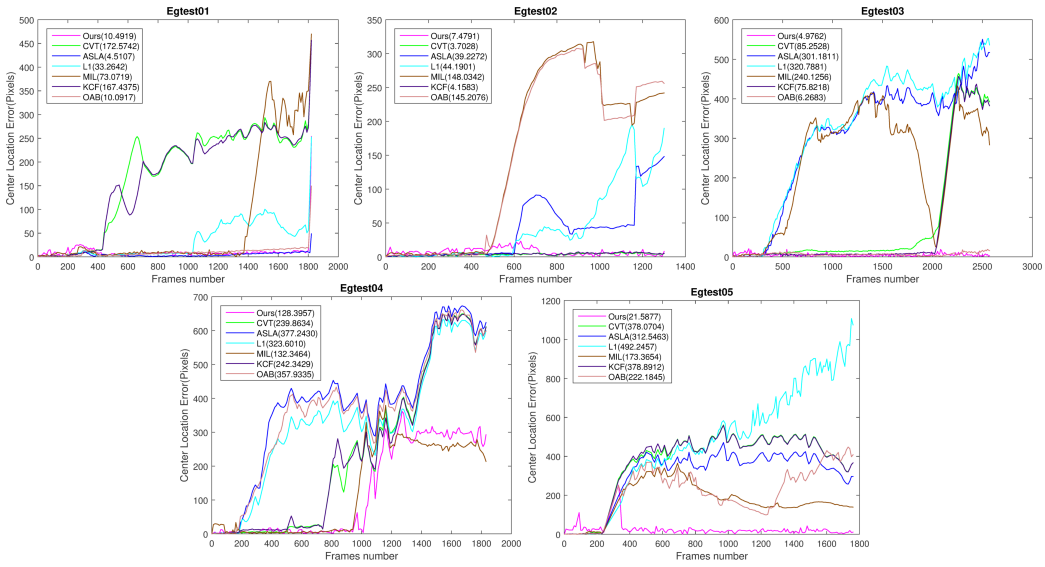

Figure 3.4: Center location error on the five VIVID videos in which only a single-target was tracked.

not score the best on two of the videos, the differences from the best and second best methods were minor (see Table 3.1).

Fig. 3.4 shows CLE for each frame in the individual videos. In video Egtest03, our method outperformed most of the trackers by a large margin, scoring the mean CLE of only 4.9 pixels. In comparison, all other trackers, except for OAB, had a mean CLE of over 60 pixels. Our method outperformed most of the others by a margin of over 100 pixels for Egtest04, except for MIL. In Egtest05, our algorithm outperforms all the trackers by at least 150 pixels. Interestingly, some of the algorithms produced similar error patterns. For example, in Egtest04 video, ASLA, OAB and L1 have similar error curves, which hints that neither algorithm can handle the series of occlusions presented in the video. Also, in Egtest05, which contains changes in illumination and long occlusions, ASLA, KCF and CVT have similar error curves with only a height offset. Fig. 3.5 shows the precision and success plots averaged across all five videos. Our method performs better than the comparison algorithms by a large margin. For precision, our method scored 81% at a threshold of 20 pixels, and the second closest is the OAB tracker with 52%. This indicates that our method's predicted bounding boxes were consistently close to the ground truth bounding box centers. Similarly, in the success plot, our method achieves very good results, with an AUC of 0.39. OAB again was the second best method, achieving an AUC of 0.29.

Table 3.1: Mean center location error for each of the VIVID videos in the single-object tracking results and the mean error across all of the videos. For each video, the algorithm with the lowest error is denoted in red and the second lowest error is denoted in blue.

	Ours	CVT	ASLA	L1	MIL	KCF	OAB
Egtest01	10.49	172.57	4.51	33.26	73.07	167.43	10.09
Egtest02	7.47	3.70	39.22	44.19	148.03	4.15	145.20
Egtest03	4.97	85.25	301.18	320.78	240.12	75.82	6.26
Egtest04	128.39	239.86	377.24	323.60	132.34	242.34	357.93
Egtest05	21.58	378.07	312.54	492.24	173.36	378.89	222.18
Mean	34.57	170.37	206.93	242.81	153.38	173.72	151.93

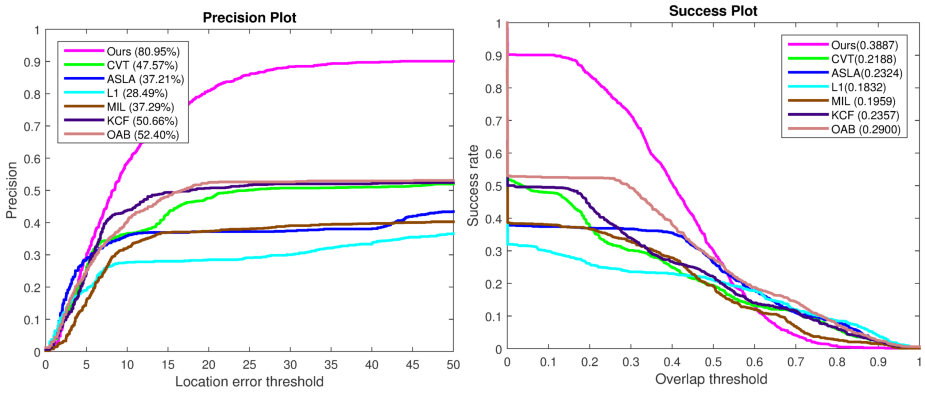

Figure 3.5: Precision and success plots for the single-target tracking results, averaged across all five videos.

3.2.2 Simultaneously tracking multiple objects

We used VATIC [121] to carefully annotate three vehicles in a challenging VIVID video that contains numerous long duration occlusions. While our method and SPOT [48] can track multiple objects simultaneously, for the other methods we simply re-run their algorithm three times, *i.e.*, once per vehicle. Our algorithm did especially well on the multi-target tracking video compared to the other methods[2]. As shown in Fig. 3.6, the mean CLE across the three vehicles was 13 pixels for our method, while the second best was 179 pixels for SPOT. Fig. 3.7 shows the precision and success plots for each of the trackers averaged across the three vehicles, and our method performed

[2]A YouTube multi-target video showing the results for all trackers can be found here: `https://goo.gl/ptIoHA`

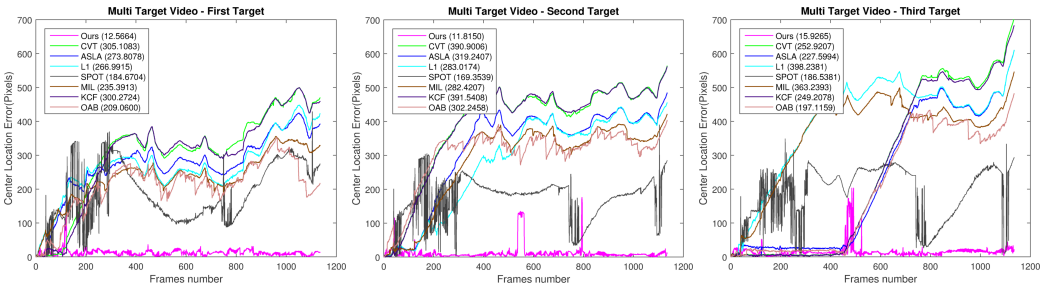

Figure 3.6: Center location error for each target in the multi-target video.

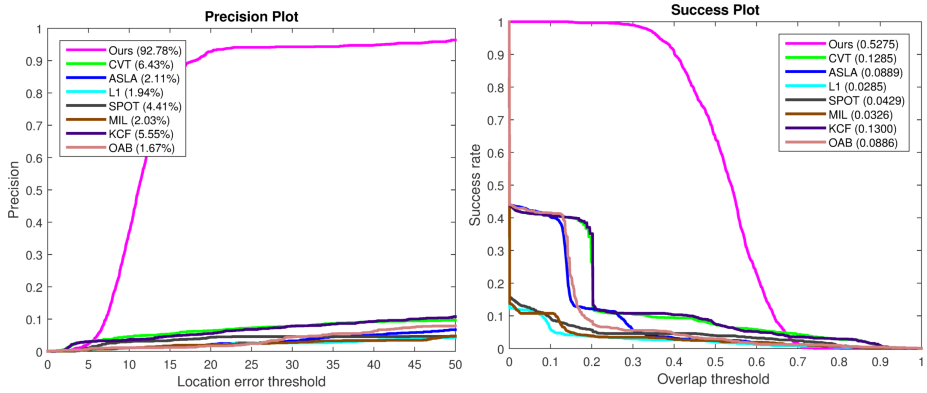

Figure 3.7: Precision and success plots for multi-target tracking, computed by averaging across the three targets in the video.

very well by these metrics.

3.2.3 Appearance vs. location vs. motion

How important is each component of SPT? To study this, we created all sensible sub-models of SPT: 1) $a\left(k,t,z\right)\ell\left(k,t,z\right)m\left(t,z\right)$, 2) $a\left(k,t,z\right)\ell\left(k,t,z\right)$, 3) $a\left(k,t,z\right)m\left(t,z\right)$, 4) $\ell\left(k,t,z\right)m\left(t,z\right)$, and 5) $a\left(k,t,z\right)$. We evaluated each of these models on all of the single-target videos. Results with center location error are shown in Table 3.2. Overall, the entire model works better than any of the sub-models. Location and motion alone is the second-best model, and the three top models all use motion, which is consistent with its central role in our model of smooth pursuit and in the primate visual system. Additional results are shown in Supplementary Materials.

Table 3.2: Mean CLE values for SPT sub-models

	$a\,\ell\,m$	$\ell\,m$	$a\,m$	$a\,\ell$	a
Egtest01	10.49	105.37	99.17	229.49	108.59
Egtest02	7.47	92.50	110.64	134.01	306.89
Egtest03	4.97	117.69	100.75	174.49	203.43
Egtest04	128.39	81.97	179.54	312.97	167.25
Egtest05	21.58	59.55	170.16	87.39	166.47
Mean	34.57	91.41	132.05	187.67	190.52

Table 3.3: The speed, in seconds per frame, and implementation language for each tracker.

Method	**Overall Speed (s)**	**Language**
Ours	0.65	MATLAB
CVT	0.02	MATLAB
ASLA	0.52	MATLAB
L1	0.08	MATLAB
MIL	0.28	C++
KCF	0.07	C++
OAB	0.71	C++
SPOT	0.27	MATLAB

3.2.4 Timing benchmarks

A comparison of the speed for all of the methods is shown in Table 3.3. We did our timing benchmarks on a desktop with an Intel Core i7-5820K CPU and an NVIDIA Titan X, which was used to speed up ConvNet feature computation. SPT is written in MATLAB, and it is not optimized for speed. While our implementation is slower than most of the methods, it was faster than OAB. We further analyzed how much time each component of SPT requires. Feature computation took 0.07s, computing the motion map took 0.16s, computing the appearance map took 0.08s, and computing the location map took 0.01s. One of the most expensive operations was Selective Search, which required 0.93s each time it was invoked. The frame alignment required to compensate for camera motion in the computation of the motion map is expensive, and most of that time could be eliminated if a stationary camera was used.

3.3 Conclusions

In this chapter, we proposed an algorithm for online visual tracking of small moving objects from aerial full-motion video data. The algorithm elegantly combines saliency maps for appearance, future location, and motion. SPT extends easily to tracking multiple objects simultaneously with little computational overhead. We showed that the algorithm surpassed comparison methods from the literature on tracking moving vehicles in videos acquired by an aerial platform. In the next chapter, we continue the research and development of deep-learning to solve a similar, complex data exploitation task from an aerial imaging platform: semantic point cloud classification.

Chapter 4

Task 2: Point cloud labeling

Similar to target tracking from aerial full-motion video, the task of semantic classification is relatively easy for the human visual and cognitive system, but exceedingly difficult for machines, due to the higher level of contextual understanding that is required. Nevertheless, automated, machine-assisted solutions are becoming increasingly important as the volume, variety, and velocity of remote sensing data increases. This chapter presents a novel deep learning method that leverages a flexible and simple multi-scale deep learning framework for direct semantic labeling of 3D aerial point clouds, thus eliminating the need for calculating costly, handcrafted features. The algorithm respects the permutation-invariance of input points and therefore avoids the need to transform the points to images or volumes.

This research is driven by the following observations: In Section 2.3, we introduced two groups of methods for labeling point clouds, direct and indirect. Although indirect methods introduced the application of deep learning for the semantic labeling task, they typically require a transformation of the input data, *i.e.* to views or volumes, in order to meet the ingest requirements of conventional image-based networks. Unfortunately, these transformations introduce computational overhead, add model complexity, and discard potentially relevant information. Likewise, direct methods have relied on the proliferation of various handcrafted features, in addition to contextual relationships, in order to meet increasing accuracy requirements with simple discriminative models. This added complexity has come at the cost of computational efficiency.

Meanwhile, the generation of 3D point cloud data has increased rapidly in recent years due to the availability of high-resolution optical satellite/airborne imagery and the explosion of modern stereo photogrammetry algorithms leveraging new computational resources. Such algorithms triangulate 3D-point coordinates directly from the optical imagery, and thus retain inherent spectral information; these attributes should be considered in the development of a successful model. In order to scale the semantic classification task to meet the demands of emerging data volumes – potentially at sub-meter resolution and global in coverage – an efficient, streamlined, and robust

model that directly operates on 3D point clouds is needed.

Our approach utilizes a modified version of PointNet [122], a deep network which operates directly on point clouds and so provides a flexible framework with large capacity and minimal overhead for efficient operation at scale. Moreover, it respects the permutation-invariance of input points and therefore avoids the need to transform the points to images or volumes. The network allows for summarizing the entire input point set with a feature vector that can be used for object classification, retrieval, and 2D space visualization as shown in [122]. We will refer to this vector as a global feature for the input set. The use of a global descriptor of an entire set of input samples is widely used in the video understanding domain. For example, [123] temporally pooled the per-frame features extracted by a CNN to generate a global descriptor for the entire video to recognize the activity within the video. Inspired by the success of PointNet in applications such as object classification, part segmentation, and semantic labeling, we make the following contributions:

1. We extend PointNet to handle complex 3D data obtained from overhead remote sensing platforms using a multi-scale approach. Unlike CAD models, precisely-scanned 3D objects, or even indoor scenes, airborne point clouds exhibit unique characteristics, such as noise, occlusions, scene clutter, and terrain variation, which challenge the semantic classification task.

2. We present a deep learning algorithm with convolutional layers that consume unordered and unstructured point clouds directly, and therefore respects the pedigree of the input 3D data without modifying its representation and discarding information.

3. We eliminate the need for calculating costly handcrafted features and achieve near state-of-the-art results with just the three spatial coordinates and three corresponding spectral values for each point. At the same time, the overhead of adding additional features to our model is minimal compared to adding new channels or dimensions in 2D and volumetric cases.

4. We avoid the need to explicitly calculate contextual relationships (*e.g.* by using CRF) and instead use a simple layer that can learn a per-block global features during training.

5. Being fully convolutional, our network mitigates the issue of non-uniform point density, a common pitfall for stationary LiDAR platforms.

6. We show that test time is on the order of seconds, compared to minutes for existing techniques relying on handcrafted features or contextual methods, and operating on the same dataset.

7. Finally, we show how the network can easily be applied to handle the issue of multimodal fusion in 2D-semantic segmentation, with a simple modification to the data preparation step.

This Chapter is organized as follows: Section 4.1 will describe the network used and the adaptation of convolutions to 3D point clouds. Sections 4.2 will present the methodology used to evaluate our approach. Finally, Section 4.3 draws the conclusions and presents the future work.

4.1 Methodology

In this section we present a CNN-based deep learning method that is able to learn point-level and global features directly in an end-to-end fashion, rather than relying upon costly features or contextual processing layers. In Section 4.1.1, we describe how convolutional networks can be adapted to irregular point cloud data. Section 4.1.2 describes how batch normalization is used to precondition the outputs of the activation functions. Section 4.1.3 describes a pooling layer that is used to learn contextual features. Finally, Section 4.1.4 details the inference of semantic classification labels from the learned local and global features.

4.1.1 Adaptation of CNN to Point Clouds

CNN architectures consist of multiple, layered convolution operations, wherein at each layer, the set of filter weights is learned based on training data for a specific task. Recall that for a single 2D-convolution (Equation 5.1) a filter (h) in layer (ℓ) and channel (d) "slides" across the domain of the input signal, $x[u, v]$, accumulating and redistributing this signal into the output, $f^{(\ell,d)}[m, n]$.

$$f^{(\ell,d)}[m,n] = \sum_{u=-\infty}^{\infty} \sum_{v=-\infty}^{\infty} x[u,v] h^{(\ell,d)}[m-u, n-v] \tag{4.1}$$

This sliding process can be thought of as replicating the filter at every spatial location. Replicating the filter in this way allows for the extraction of features regardless of their position, and enables a linear system to be shift-invariant. Additionally, sharing the same set of weights at multiple locations increases the learning efficiency by reducing the number of parameters in the model. Based on the nature of the convolution, CNNs typically require highly regular data, *e.g.* images, which are organized based on a 2D-grid. Also, note that such a convolution (applied to gridded data, *e.g.* images), is not invariant to permutations of the input members *i.e.* pixels. In other words, the spatial distribution of pixels within a filter window is important to capture local features such as edges. Therefore, reordering the input points will result in meaningless output.

This introduces challenges for the application of CNNs to classify irregular, unstructured 3D point cloud data. Given an input set of N 3D-points $X = \{x_1, x_2, x_3, ..., x_N\}$, where every point represents a row in a 2D-array, the goal of point cloud classification is to assign every point x_i an object-level label from a set of predefined labels $Y = \{y_1, y_2, y_3, ..., y_C\}$, where C is the total number of classes. Since point clouds are not defined on regular grids, and convolutional layers

require regular inputs, a modification to either the input or the network architecture is needed. In order to directly operate on point clouds and avoid transforming the data to a different representation (see Section 2.3.2), we follow [122] by adapting convolutional operations to point clouds. The complete architecture of our network is shown in Figure 5.1. The input to the network is an

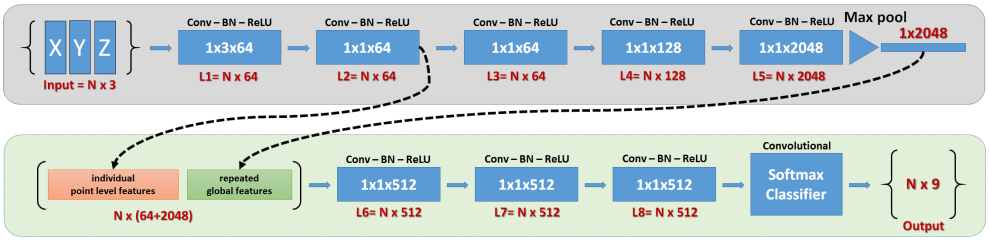

Figure 4.1: The basic semantic labeling network takes as input an Nx3 set of points (a point "cloud"), and, in the first stage, passes it through a series of convolutional layers to learn local and global features. In the second stage, concatenated features are passed though (1×1) convolutional layers and then to a softmax classifier to perform semantic classification. Text in white indicates the filter size, while text in red indicates the layer's output shape.

$N \times M$ array of unordered data points, where N is the number of points, and M is the number of features for each point, *i.e.* spatial coordinates and/or spectral information. Shown in Figure 5.1 is the simple case where the input data is the raw point cloud X, defined by its spatial coordinates (x,y,z) as columns of the array. The input could optionally be expanded to include any other features, such as spectral information. The first layer of the network applies a 1D-convolution – with a filter width equal to the width of the input vector – across the columns to capture the interactions between coordinates for each row (data point). The output of this layer is a single column where each value corresponds to an individual 3D-point within the set, *i.e.* an $(N \times M)$ input array is transformed to an $(N \times 1)$ output array. This layer operates on each point independently, an advantage which allows for the incorporation of point-wise operations such as scaling, rotation, translation, etc. Since such operations are differentiable, they can be included within the network as a layer and trained in an end-to-end fashion. This concept, first introduced in [124], allows the network to automatically align the data into a canonical space and therefore makes it invariant to geometric transformations.

The subsequent convolutional layers perform feature transformation – such as dimensionality reduction or feature expansion – using (1×1) convolutions.We avoid fully-connected layers due to their expensive computational cost. Since the convolution process is a linear operation that performs a weighted sum of its inputs, a non-linear operation – known as the activation function – is needed in order to derive and learn more complex features. Three activation functions are

frequently used in the literature: $sigmoid$, hyperbolic tangent ($tanh$), and rectified linear unit ($ReLU$). Sigmoid activation, $\sigma(x) = 1/(1 + e^{-x})$, reduces the input values to the range of [0,1]; this can be thought of as assigning a likelihood value to the input. Similarly, $tanh$ activation, $\tanh(x) = 2\sigma(2x) - 1$, maps the input values to the range of [-1,1]; this has the added advantage that the output is zero centered. Finally, $ReLU$ activation, $f(x) = \max(0, x)$, applies a simple ramp function. It reduces the likelihood of vanishing gradient, greatly accelerates the convergence rate [74], and involves simpler mathematical operations; therefore it is the most common activation function used in deep convolutional networks. We implement the $ReLU$ function in our network as follows:

$$f^{(1)} = \max(0, \langle h, x_i \rangle + b) \tag{4.2}$$

where the convolution in Equation 5.2 is represented now by the dot product $\langle \cdot \rangle$, b is the bias, and $f^{(1)}$ is the first layer's output.

4.1.2 Batch normalization

Although the $ReLU$ activation function has many advantages, it does not enforce a zero-centered distribution of activation values, which is a key factor to improve the gradient flow. One way to adjust this distribution is to change the weight initialization mechanism. [125] and [126] showed that good initialization is the key factor for a successful, convergent network, however, control over the distribution of activations was handled indirectly. For improved control, [127] introduced Batch Normalization (BN) that directly operates on the activation values. An empirical mean and variance of the output (after the convolutional layer and before the non-linearity) are computed during training; these are then used to standardize the output values, *i.e.*

$$\hat{s} = \frac{s - E[s]}{\sqrt{Var[s]}} \tag{4.3}$$

$$z = \gamma \cdot \hat{s} + \beta \tag{4.4}$$

where $s = \langle h, x_i \rangle$ is the output after the convolutional layer, and γ and β are the scale and shift parameters learned during training. Setting $\gamma = \sqrt{Var[s]}$, and $\beta = E[s]$ allows the network to recover the identity mapping. BN improves flow through the network, since all values are standardized, and simultaneously reduces the strong dependence on initialization. Also, since it allows for homogeneous distributions throughout the network, it enables higher learning rates, and acts as a form of regularization.

Incorporating these advantages, we initialize weights using the method of [125] and insert BN layers after every convolutional layer, as shown in Figure 5.1. Note that the BN layer functions differently during testing and training. During testing, the mean and the variance are not computed. Instead, a single fixed value for the mean and the variance – found empirically during training

using a running average – is used during testing. After integrating BN, the activation function (Equation 5.2) can now be written as follows:

$$f = \max(0, BN(h^T x + b)) \tag{4.5}$$

Evaluating Equation 5.3 multiple times for different values of h allows each layer to capture various aspects of its input. This results in an output array of $(N \times K)$ dimensions, where K is the total number filters used. The size of K per-layer is shown in Figure 5.1 (white text). Following the output with a series of convolutional layers, as shown in the upper part of Figure 5.1, can be defined mathematically as a sequence of nested operations as follows:

$$f^{(\ell)} = f^{(\ell-1)}(f^{(\ell-2)}(...f^{(1)}(x))) \tag{4.6}$$

where f^{ℓ} is defined as in Equation 5.3, and ℓ is the layer index.

4.1.3 Contextual feature learning

During training, we desire a network that can learn both local features (at the point-level) and global features (at the block-level), which provide additional contextual information to support the classification stage. Here, we describe how global features can be extracted using a pooling layer, which simultaneously provides permutation-invariance, *i.e.* the order of the input points does not affect classification results. Contrary to 2D-images, point clouds are unstructured and unordered, and so an appropriate network should respect this original pedigree.

Three options are available: sorting the data as a preprocessing step, using a Recurrent Neural Network (RNN), or incorporating a permutation-agnostic function that aggregates the information from all points. In the first case, the optimal sorting rules are not obvious. In the second case, the point cloud must be considered as a sequential signal, thus requiring costly augmentation of the input with all possible permutations. While some architectures such as long short-term memory (LSTM) and gated recurrent unit (GRU) neural networks can deal with relatively long sequences, it becomes difficult to scale to millions of steps, which is a common size in point clouds.

Given these considerations, we incorporate a permutation-agnostic function as an additional layer. Specifically, pooling layers, commonly used to downsample 2D-images, work perfectly for this purpose. Instead of downsampling the point set, we pool values across all points to form a single global signature that represents the input point cloud. Such signature could be used directly to label the whole set. However, recall that for the task of semantic labeling, a label for each 3D-point is desired. Therefore, both local *and* global features are needed to describe the point and capture contextual information within the input set.

In this network, local features are obtained from the second convolutional layer $f^{(2)}$ with an output shape of $(N \times 64)$, *i.e.* this represents each 3D-point with a 64D-vector. On the other hand, a global signature for the point set is derived from the output of the fifth convolutional layer

with dimensions of $(N \times 2048)$. This serves as input to the global feature extraction function, specifically, a *max-pooling* layer, which aggregates the features across all points and produce a signature with a shape of (1×2048) as follows:

$$g = \max_{row}(f^{(5)}) \tag{4.7}$$

where g is the global feature vector, $f^{(5)}$ is the output at the 5^{th} layer, and row indicates that the aggregation is applied vertically across the rows, *i.e.* points.

4.1.4 Inference

The global feature vector is concatenated with the point level features, yielding a per-point vector that contains both local and contextual information necessary for point-wise labeling. This concatenated feature vector is then passed to a series of feature transformation layers (1×1 convolutions) and finally to a softmax classifier as shows in the lower part of Figure5.1. We use the cross-entropy cost function to train the network. Cross-entropy is a special case of the general Kullback-Leibler (KL)-divergence D_{KL}, which measures how the ground-truth probability distribution p diverges from the output probability distribution q, *i.e.* :

$$\begin{aligned} D_{\text{KL}}(p||q) &= \textstyle\sum_i p(x_i) \cdot (\log p(x_i) - \log q(x_i)) \\ &= -\textstyle\sum_i p(x_i) \log q(x_i) - \sum_i p(x_i) \log \frac{1}{p(x_i)} \\ &= H(p,q) - H(p) \end{aligned} \tag{4.8}$$

If the discrete distribution p is zero everywhere except a single location with maximum probability, the expression is reduced to the cross-entropy $H(p,q)$. In our case, p is the ground truth distribution represented by one-hot vectors encoding the label per-point, while q is the output of the softmax layer, representing the normalized class probabilities. Here, each individual class probability is calculated as follows:

$$P(y_i \mid f^{(\ell)}(x_i); W) = \frac{e^{f_{y_i}}}{\sum_c e^{f_c}} \tag{4.9}$$

where $f^{(\ell)}$ is the last convolutional layer and the input to the softmax layer, y_i is the current correct label for the input x_i, W is the weight matrix for the softmax classifier, and f_{y_i} is the unnormalized log probability of the output node indexed by y_i, and c is the class (output) index. Figure 4.2 illustrates the methodological concepts presented in Section 4.1.3 ans Section 4.1.4.

4.2 Evaluation

This section describes the methodology used to evaluate the performance of our approach. In Section 4.2.1 we describe the datasets used. In Section 4.2.2 we describe preprocessing steps.

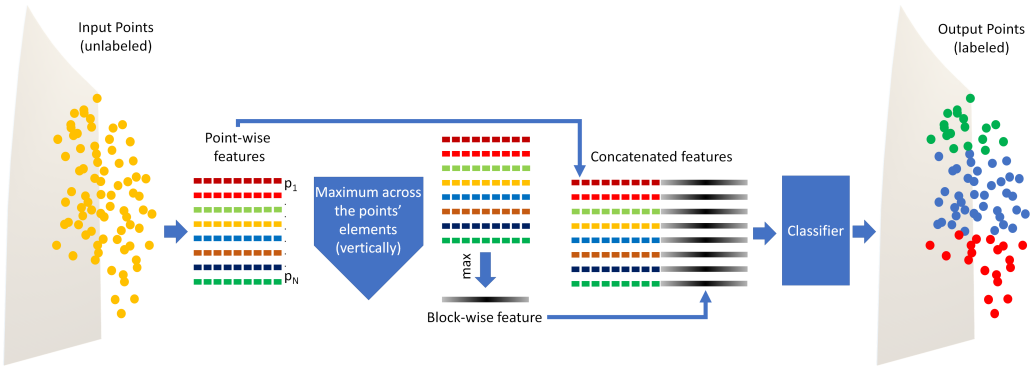

Figure 4.2: A graphical illustration of the methodological concepts presented in Section 4.1.3 ans Section 4.1.4. The Point-wise features are the output of L2 in Figure 5.1. For more details about the classifier, please see the bottom part of Figure 5.1.

In Section 4.2.3 we outline training parameters. In Section 4.2.4 we present our results on 3D-point clouds along with various experiments showing qualitative performance of our method when applied to different testing cases. In Section 4.2.5 we analyze the effects of the input feature selection. Finally, in Section 4.2.6 we show the extension of our network to handle the fusion of LiDAR and spectral data in 2D-semantic segmentation tasks.

4.2.1 Dataset

For this Chapter, we use data provided by both the ISPRS 2D[1] and 3D-Semantic Labeling Contest, as part of the urban classification and 3D-reconstruction benchmark. Both airborne LiDAR data and corresponding georeferenced IR-R-G imagery are provided from Vaihingen, Germany (Figure 4.3). The georeferenced image of the whole area has a ground sampling distance of 8cm and a size of 20250x21300 pixels. For the 3D contest, 9 classes have been defined, including *Powerline*, *Low vegetation*, *Impervious surfaces*, *Cars*, *Fence/Hedge*, *Roof*, *Facade*, *Shrub*, and *Tree*. The area is subdivided into two regions, for training and testing. Each region includes a text file that contains the LiDAR-derived (x,y,z) coordinates, backscattered intensity, and return count information, acquired using a Leica ALS50 system at a mean height of 500m above ground. The point density is approximately 4 points/m^2 with a total of 753,859 training points, and 411,721 testing points. The test area is within the center of Vaihingen city, and is characterized by dense, complex buildings.It covers an area of 389m x 419m. The training area, on the other hand, is mostly residential, with detached houses and high rise buildings. It covers an area of 399m x 421m.

[1] `https://goo.gl/mdvbwM`

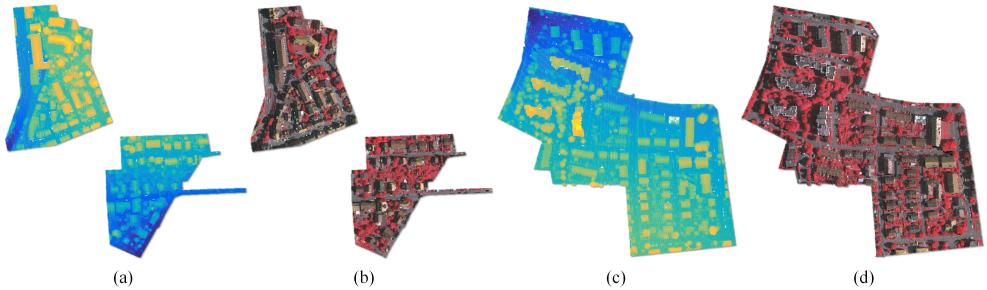

Figure 4.3: From left to right: point cloud (a) color-coded by height, and (b) by spectral information (IR,R,G) for the test area; and point cloud (c) color-coded by height, and (d) by spectral information (IR,R,G) for the training data.

4.2.2 Preprocessing

Two preprocessing methods were employed to obtain our desired input. First, spectral information was attributed to each (x,y,z) triplet in the point cloud by applying a bilinear interpolation using the georeferenced IR-R-G imagery as shown in Figure 4.3. Note that in the case of stereo-derived point clouds from optical imagery, this spectral information is inherently available, and would not need to be obtained separately. Next, we normalize the z values in the point cloud by subtracting a Digital Terrain Model (DTM), generated using LAStools[2], in order to obtain height-above-ground. Then, to train our deep learning method from scratch, a sufficiently large amount of labeled data are required; however, a single and small training scene is provided. We solve this issue by subdividing our training and testing regions into smaller 3D-blocks. Such blocks are allowed to overlap (unlike PointNet), thus increasing the quantity of data available, and robustness by allowing overlapped points to be part of different blocks. Each point within the block is represented by a 9D-vector, containing the per-block centered coordinates (X,Y,Z), spectral data (IR,R,G), and normalized coordinates (x,y,z) to the full extent of the scene. Note that since our method is fully convolutional, the number of points per-block can vary during training and testing. This contribution resolves the typical challenge of working with point clouds of varying density. While we test using different densities, we sample fixed number of points per-block during training for debugging and batch training purposes.

To sample points from each block, we randomly choose 4096 points during training without replacement. If the number of points per-block is lower than the desired number of samples, random points within the block are repeated. However, if the number of points per-block is lower than 10 points, the block is ignored. To learn objects with different scales, *e.g.* building vs.

[2] `http://www.lastools.org/`

car, one can train separate networks, with each network trained using a down-sampled version of the point cloud, as in [89]. However, this is not practical, as it introduces an additional and unnecessary computational overhead. Instead, current deep learning approaches – *e.g.* a single network with high capacity – should be able handle multi-scale objects in an end-to-end fashion given appropriate inputs. To handle different resolutions, we generate blocks with different sizes and train our network using all scales simultaneously. Blocks of size 2m×2m, 5m×5m, and 10m×10m work well in our case given the scale of the features of interest *e.g.* cars and roofs. Our final result during testing is the average of all three scales. While splitting the data into blocks with different sizes increases the number of training samples, robustness to noise and orientation could be further improved by augmenting the training data with modified versions of the original training data. We augment the training data by randomly rotating points around the z-axis (*i.e.* to adjust their geographic orientation), and jittering the coordinates. Jitter is added by applying an additive noise, sampled from a zero-mean normal distribution with $\sigma = 0.08$m for the x, y coordinates, and $\sigma = 0.04$m for the z coordinates. Next we clip the values to a maximum horizontal and vertical jitter of 30cm and 15cm respectively. The values were chosen empirically to add sufficient noise while preserving the relative differences between various objects. Rotation and jitter are applied before splitting the data into blocks. However, we also apply jitter during the per-block sampling process, in order to avoid duplicating points.

4.2.3 Training Parameters

To asses and monitor the performance of our model during training, a validation set is needed. Instead of taking samples from the training set directly, we desire a validation set that is as different as possible to the training data. We address this by splitting the original training data before augmentation into a new training and validation subsets using stratified splits to preserve the distribution of classes in both sets. Class-distributions within each set are then balanced by repeating under-represented classes until they match the class with the highest number of members, resulting in a uniform distribution of classes. We then augment the new training data by applying the jitter and rotations as described in Section 4.2.2.

After splitting both the training and validation sets into blocks, we train using the augmented data only as input to the network shown in Figure 5.1. We use the Adam optimizer [128] with an initial learning rate 0.01, a momentum of 0.9 and a batch size 32. The learning rate (lr) is iteratively reduced based on the current number of epochs, according to:

$$lr_{new} = lr_{initial} \times (1.0 - \frac{epoch_{current}}{epoch_{total}}) \tag{4.10}$$

This proceeds for a total of 30 epochs, *i.e.* $epoch_{total} = 30$. We monitor the progress of the validation loss and save the weights if the loss improves. If the loss does not improve after 3 epochs, training is terminated and the weights with the best validation loss are used for testing. Training

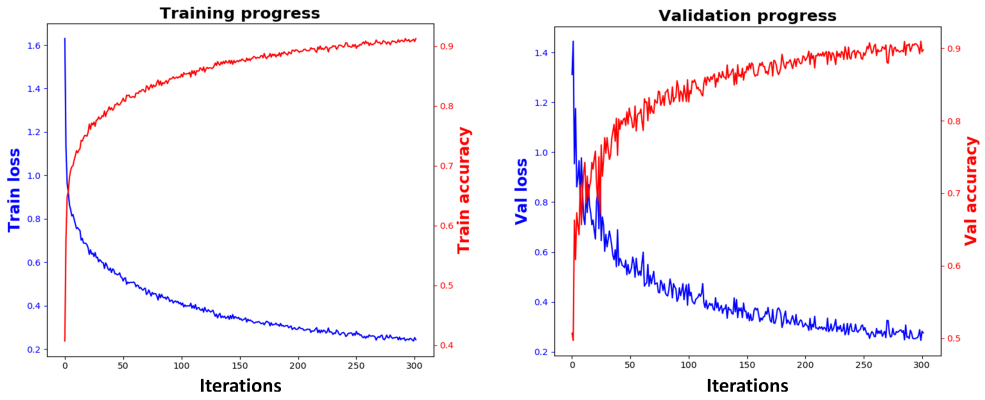

Figure 4.4: The loss and overall accuracy progress for training (left) and validation (right). Every 10 iterations correspond to a single epoch.

our network takes around 12 to 18 hours to converge using a Tesla p40 GPU and Keras [129] with the Tensorflow backend. The feed forward time during testing is 3.7s for the full scene ($\sim$ 412k points). Figure 4.4 shows the loss and overall accuracy progress during training and validation.

4.2.4 Classification Results

Classification results are based on the data provided by the ISPRS 3D Semantic Labeling Contest (see Section 4.2.1). During testing, we split the data into blocks similar to the training stage, in order to recover objects at different scales. The block sizes, overlap, and number of points per-block are reported in Table 4.2. We forward pass the data to output a 9D-vector per-point indicating the probability of each point belonging to each of the nine categories. Since we use a fixed number of points for each block size, some points from the original data may remain unclassified due to sampling, while others may be duplicated due to repetition. Therefore, we interpolate the output results to classify the original set of points. We use nearest neighbor interpolation on each class probability separately, to generate a total of nine class probability maps. A label corresponding to the index of the highest probability from the nine maps is assigned to each point, indicating the classification label. Quantitative performance metrics are based on the ISPRS contest: Per-class accuracy, precision/correctness, recall/completeness, and F1-score, in addition to the overall accuracy. Although it is possible to submit results while excluding some classes, we chose to evaluate our method on all available classes. The confusion matrix in Table 4.1 shows resulting per-class accuracies. Figure 4.5 shows the corresponding classification map, along with the error map provided by the contest organizers. As shown in Table4.1, the proposed method performs

Table 4.1: Confusion matrix showing the per-class accuracy using our deep learning framework. The overall accuracy (not shown) is **81.6**%

Classes	power	low_veg	imp_surf	car	fence_hedge	roof	fac	shrub	tree
power	**29.8**	00.0	00.2	00.0	00.0	54.2	00.7	00.2	15.0
low_veg	00.0	**69.8**	10.5	00.5	00.2	00.5	00.6	16.1	01.9
imp_surf	00.0	05.2	**93.6**	00.2	00.0	00.2	00.1	00.7	00.0
car	00.0	05.0	01.2	**77.0**	00.2	02.6	00.8	12.9	00.3
fence_hedge	00.0	05.9	01.4	01.7	**10.4**	01.5	00.6	68.5	10.0
roof	00.1	00.5	00.4	00.0	00.0	**92.9**	02.8	02.3	00.9
fac	00.2	04.3	00.8	00.9	00.1	23.3	**47.4**	19.9	03.1
shrub	00.0	07.9	00.5	01.0	00.5	02.6	02.0	**73.4**	12.0
tree	00.0	00.8	00.0	00.2	00.1	01.2	01.3	17.1	**79.3**
Precision/Correctness	50.4	88.0	89.6	70.1	66.5	95.2	51.4	33.4	86.0
Recall/Completeness	29.8	69.8	93.6	77.0	10.4	92.9	47.4	73.4	79.3
F1 Score	37.5	77.9	91.5	73.4	18.0	94.0	49.3	45.9	82.5

Table 4.2: The block sizes and the corresponding overlap and number of points during testing.

Size	Overlap	# Points
2m×2m	1m	1024
5m×5m	2m	3072
10m×10m	2m	4096

well on *impervious surfaces* and *roof* classes. The worst performance is for the *fence/hedge* and *powerline* classes, which according to Table 4.1 is due to the confusion between closely-related classes. For example, *powerline* is mainly confused with *roof*, and *fence/hedge* is confused with *shrub*, both of which have similar topological and spectral characteristics. Likewise, the accuracy of *low vegetation* is affected by the presence of *impervious surfaces* and *shrubs*. *Shrub* appears to be causing most of the confusion. This is likely due to the fact that the spectral information is similar among the vegetation classes *low vegetation*, *shrub*, and *trees*. While height information may improve the results, the presence of classes with similar heights such as *car* and *fence/hedge* make this differentiation challenging.

To evaluate our performance against others, we compare our method to all submitted ISPRS contest results (both published and unpublished) at the time of paper submission. Since some submissions are unpublished we will review them briefly using the available information on the contest website[3]; We refer to the submitted methods according to names posted on the contest

[3] https://goo.gl/6iTj6W

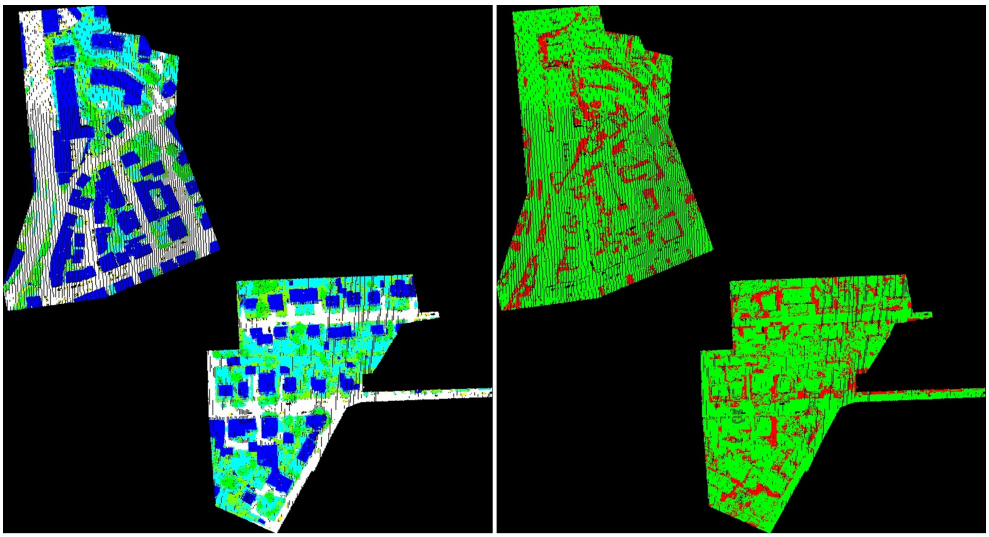

Figure 4.5: The left image shows the classification map, while the right image shows the corresponing error map. The results were provided by the contest organizers.

website. Interested readers are encouraged to review the website for further details.

The **IIS_7**[4] method used spectral and geometrical features, combined with a segmentation based on supervoxels and color-based region growing. In contrast to **IIS_7**, we don't handcraft geometrical features or utilize the spectral information to segment the point cloud into similar coherent regions before classification. The **UM**[5] method used a One-vs-One classifier based on handcrafted features, including point attributes such number of returns, textural properties, and geometrical attributes. The **HM_1**[6] method utilized a CRF with RF classifier and contrast-sensitive Potts models, along with 2D-and 3D-geometrical features. The **WhuY3** deep learning method [130] used a window around each 3D point and divided such window into 128x128 cells. Five features (planarity, sphericity, intensity, variance of angles with respect to the normal, and height above ground) were estimated for each point in a cell. The whole cell was then transformed into a single pixel to for 128x128 image. A convolutional network was then used to classify the images. The **K_LDA** [60] method used covariance features at multiple scales. Finally, the

[4] https://goo.gl/zepC6d
[5] https://goo.gl/uzZFcs
[6] https://goo.gl/d7AmXY

Table 4.3: The per-class accuracy for each method and the corresponding Overall Accuracy (OA). Values in **red** correspond to the highest score, **blue** to second highest score, and **green** to third highest score.

Methods	power	low_veg	imp_surf	car	fence_hedge	roof	fac	shrub	tree	OA
Ours	29.8	69.8	93.6	**77.0**	10.4	92.9	47.4	**73.4**	79.3	**81.6**
IIS_7	40.8	49.9	**96.5**	46.7	**39.5**	**96.2**	—	52.0	68.8	76.2
UM	33.3	**79.5**	90.3	32.5	02.9	90.5	43.7	43.3	**85.2**	**80.8**
HM_1	**82.8**	65.9	**94.2**	67.1	25.2	91.5	49.0	**62.7**	**82.6**	80.5
WhuY3	0.247	**81.8**	91.9	**69.3**	14.7	**95.4**	40.9	38.2	78.5	**82.3**
K_LDA	**89.3**	12.4	47.6	28.9	20.4	80.7	**51.3**	38.4	72.8	50.2
LUH	53.2	72.7	90.4	63.3	**25.9**	91.3	**60.9**	**73.4**	79.1	**81.6**

Table 4.4: The F1-scores per-class for each method and the average value. Values in **red** correspond to the highest score, and **blue** to second highest score. Our score is marked with **green** color.

Methods	power	low_veg	imp_surf	car	fence_hedge	roof	fac	shrub	tree	Avg. F1
Ours	37.5	77.9	**91.5**	**73.4**	18.0	**94.0**	49.3	45.9	**82.5**	**63.33**
IIS_7	54.4	65.2	85.0	57.9	28.9	90.9	—	39.5	75.6	55.27
UM	46.1	**79.0**	89.1	47.7	05.2	92.0	52.7	40.9	77.9	58.96
HM_1	**69.8**	73.8	**91.5**	58.2	**29.9**	91.6	**54.7**	**47.8**	80.2	**66.39**
WhuY3	37.1	**81.4**	90.1	63.4	23.9	93.4	47.5	39.9	78.0	61.63
K_LDA	05.9	20.1	61.0	30.1	16.0	60.7	42.8	32.5	64.2	37.03
LUH	**59.6**	77.5	**91.1**	**73.1**	**34.0**	**94.2**	**56.3**	**46.6**	**83.1**	**68.39**

LUH[7] method used a two-layer hierarchical CRF that explicitly defines contextual relationships and utilizes voxel cloud connectivity segmentation, along with handcrafted features such as Fast Point Feature Histograms (FPFH).

Per-class accuracy, and overall accuracy (OA) for each submission, including ours, are shown in Table 4.3. As seen in Table 4.3, our method ranks second overall, sharing an overall accuracy of 81.6% with **LUH**. The method with the highest overall accuracy is **WhuY3**, achieving 82.3%. However, **WhuY3** achieved only one per-class highest accuracy score, as opposed to two for our method: the *car* and *shrub* classes. Likewise, the **IIS_7** method, which achieved the most (3) highest scores on a per-class basis, and use spectral data as well, only ranked eighth overall with an overall accuracy of 76.2%.

We also evaluated our results using the F1-score, which is generally considered to be a better

[7] `https://goo.gl/fGzrf3`

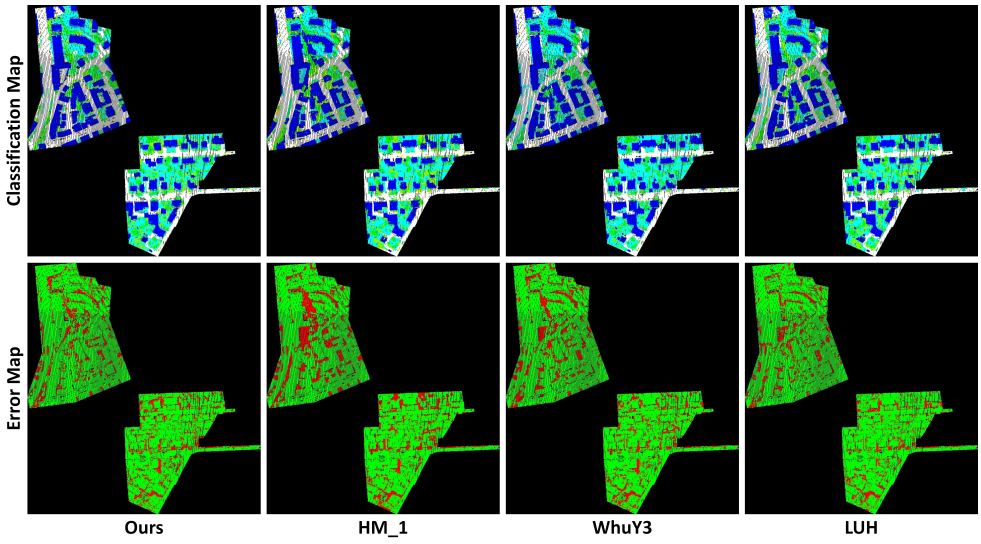

Figure 4.6: A qualitative comparison showing the output classification maps (top row) and the error maps (bottom row) for the top 4 methods (columns) in Table 4.4.

comparative metric when an uneven class distribution exists and/or when the costs of false positives and false negatives are very different. Table 4.4 compares our method to others using the F1-score. Our method performed well across all classes except for the *fence/hedge* class. Other methods demonstrated similarly poor results on this same class. While **LUH** did score the highest on the *roof* and *tree* classes, their scores were only marginally better than ours, 0.2% and 0.6%, respectively. Their higher performance for the *fence/hedge* and *powerline* classes did, however, allow them to achieve the highest F1-score of 68.39%, with **HM_1** ranking second with a score of 66.39%. Our presented technique did ranked third with a score of 63.33%, surpassing **WhuY3** which scored the highest on overall accuracy (Table 4.3). Figure 4.6 shows a qualitative visual comparison between the top 4 methods in Table 4.4.

The scores in both Tables should be thought of in context of the algorithm complexity. In other words, minimal accuracy gains may not be worth the added computational overhead required by other methods. The **LUH** method, for example, fares well in both overall, and per-class, accuracy and F1-scores. However, this method uses two independent CRFs with handcrafted features, and segmentation methods that force points within a segment to share the same label. Likewise, **HM_1** uses a variety of contrast-sensitive Potts models, which may help preserve edges during segmentation, but adds NP-hard components to the problem [131]. As a result, while smoother results may be achieved for small data sets, this comes at the cost of slow run-time performance

(see Section2.3.1) and scalability limitations for massive data sets. In contrast, we utilize a series of simple 1D-convolutions that operate directly on the point cloud, without engineering additional features, or requiring structured representations such as segments. Instead, point-wise features (*e.g.* spatial coordinates and/or spectral information), and per-block contextual descriptors are learned in a straightforward, end-to-end fashion. Our average inference time is 3.7s for a point cloud with N approximately equal to 412k.

4.2.5 Effect of Individual Features

Our 1D convolutional network has flexible input feature requirements, and can consume directly (i) spatial coordinates only, *i.e.* a 3D point cloud, (ii) spatial coordinates and spectral information, or (iii) spectral information only. Moreover, 3D spatial coordinates (x,y,z) may optionally be normalized to remove the effects of terrain, providing (x, y, height-above-ground). Finally, models can be trained at different scales by adjusting the block size. In this section we provide two experiments to analyze the impact of feature selection and digital terrain model on model performance for various use-cases.

In the first experiment, we investigate the effect of the input feature selection, *i.e.* spatial and/or spectral information, by training our model based on three sets of input data. The first model is trained using 3D coordinates only. The second model is trained using spectral information only. The third model is trained using both 3D-coordinates and spectral information (IR-R-G) for each point; this is the best-performing network, which was evaluated in more detail in Section 4.2.4. Results are shown in Figure 4.7 for multiple scales, *i.e.* block sizes of 2m$\times$2m, 5m$\times$5m, and 10m$\times$10m, and, in column 4, the average result across all scales.

When using the 3D-coordinates only (first row), the larger scale performs better than the smaller scale. On the other hand, when using the spectral information only (second row), the smaller scale performs better than the larger scale. This result is interesting since it shows the effect of global features at multiple scales. For example, when using the spectral data, smaller scales will generally include similar points as opposed to larger scales. Therefore, the global features will accurately describe the group (block) of points at a smaller scale. In contrast, the global features will not sufficiently describe structures when using only the 3D-coordinates at a smaller scale. In this case, a larger scale is needed to capture structural information that can help distinguish between 3D objects.

This suggests that combining both features could improve the results. The result submitted to the ISPRS 3D Semantic Labeling Contest (bottom right corner) used the average of 3 block sizes, trained on both (x,y,z) point coordinates and spectral information. The red circle highlights how using all features helped to correctly classify a troublesome *low vegetation* region. Likewise, the red box highlights how the spectral data, in the absence of 3D coordinates, tends to classify all vegetation as *tree*, while the 3D-coordinates, in the absence of spectral data tends to confuse *im-*

pervious surfaces with *low vegetation*. See Figure 4.5 for reference regrading correctly classified regions in our submission.

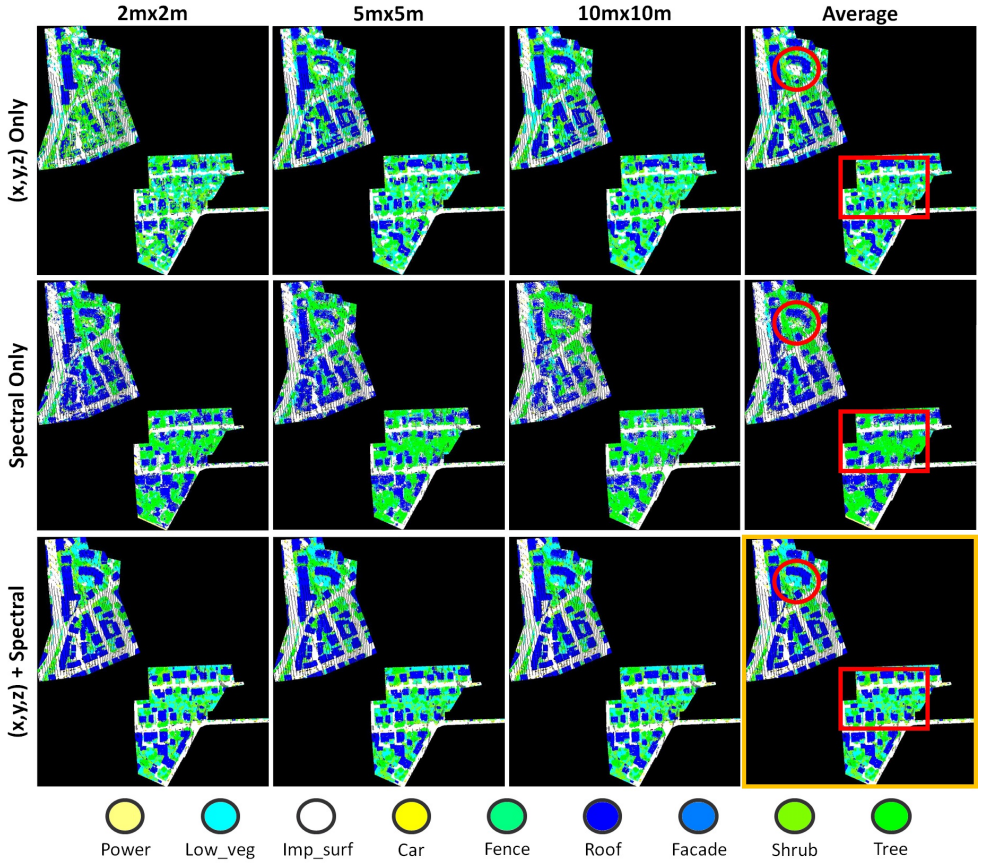

Figure 4.7: Results matrix showing the effect of training the network using different input features (rows), and at different block sizes (columns). The submitted result to the ISPRS 3D Semantic Labeling Contest is shown in the bottom right. Red markers indicate regions of comparison. Class color keys are shown at the bottom.

In the second experiment, we investigated the effect of normalizing the z-coordinates to height-above-ground, based on the DTM model. This was done both in the absence (Figure 4.8) and presence (Figure 4.9) of spectral information, which may not always be available. Figure 4.8 shows that in the absence of spectral information, normalizing the 3D coordinates to height-above-ground provides a cleaner and less-fragmented classification at all scales, including the average.

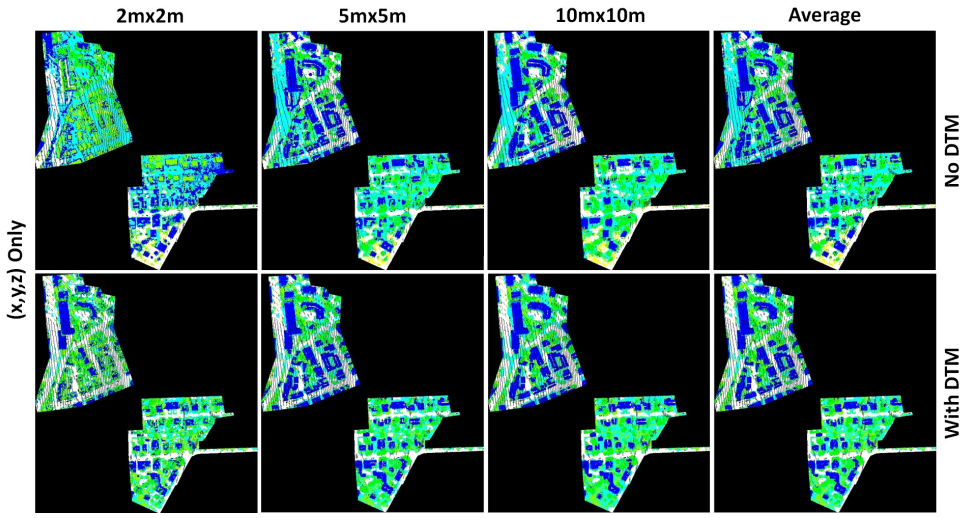

Figure 4.8: A comparison between training with (bottom) and without (top) a digital terrain model using only 3D-coordinates for block sizes.

As shown in Figure 4.9, the best classification results are achieved when spectral information is available, and after obtaining height-above-ground using a DTM. Close scrutiny of the regions marked in red shows that terrain-normalized input points improves the classification, especially for parts of roofs. However, in general, when spectral information is available, the results without using a DTM are still reasonable. This is exciting, as it opens the door to more streamlined point cloud exploitation workflows that can directly ingest 3D data in its original form. Furthermore, this may enable new techniques for generating, at scale, more precise DTMs based on the spectral content inherent in stereo-derived point clouds [132].

4.2.6 Extension to 2D Semantic Labeling

A primary contributions of our method is the compact, 1D CNN architecture. This complements the input data characteristics of point clouds in ways that traditional 2D CNNs do not. In other words, deep learning methods that rely on mapping features to 2D image-like maps, *i.e.* DSM or density maps [79, 85, 92] don't take into account the increased complexity when adding new features. In such a case, adding a new feature involves concatenating the data with a new (full) 2D-channel, subsequently requiring an additional set of 2D-filters and greatly increasing the amount memory required. On the other hand, our method operates directly on the point cloud, representing each point with a 1D-vector. Adding a new feature only requires appending each per-point vector

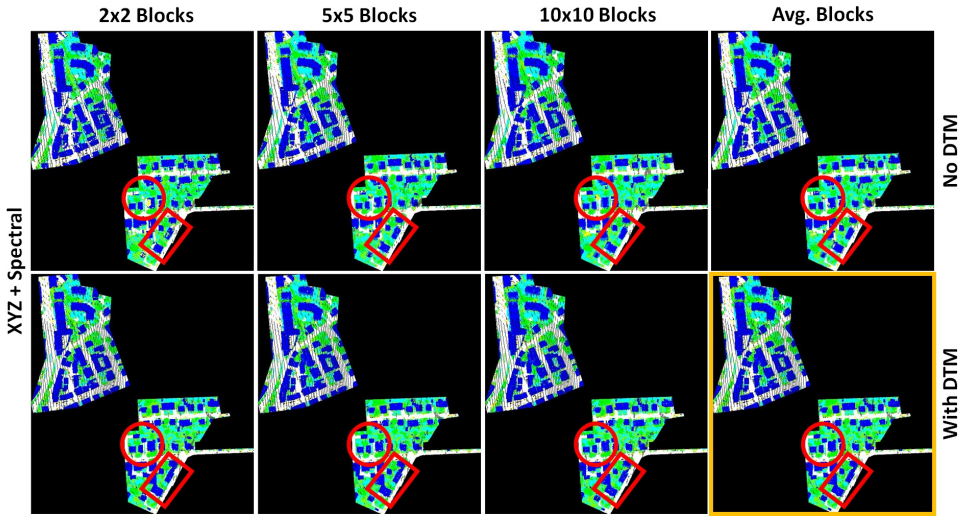

Figure 4.9: A comparison between training with and without a digital terrain model using 3D-coordinates and spectral data. Regions marked with red highlight differences.

with a single value; this increases the width of the 1D-convolutions by only one element in the first layer and does not modify the rest of the network. This advantage provides an elegant solution to the 3D multi-sensor fusion problem, allowing us to readily combine complementary information about the scene from different modalities for semantic segmentation.

Additionally, we show that this 1D architecture can be easily extended to the traditional task of 2D-image labeling. To handle 2D images, a preprocessing step simply restructures the data from a raster representation to a 2D-array where each row represents a 3D-point vector.The spatial position of each point corresponds to the 2D-pixel coordinates, while the height value corresponds to the digital counts in the DSM image. The spectral information of each point is derived from the corresponding image's digital numbers. We then train our model as described previously, excluding the data augmentation or multi-scale stages, due to the high resolution nature of the images which provided sufficient training samples.

We qualitatively evaluated our method against two relevant submissions from the Potsdam 2D-Semantic Labeling Contest[8]. The first method, **RIT_L7** [92] used a CRF model with an FCN and a logistic regression classifier to fuse the normalized DSM and the spectral information, scoring an overall accuracy of 88.4%. The **RIT_L7** method was chosen for comparison since it uses CRF to explicitly utilize contextual information. The second method (unpublished), **CASIA2**, fine-

[8] https://goo.gl/rN3Cge

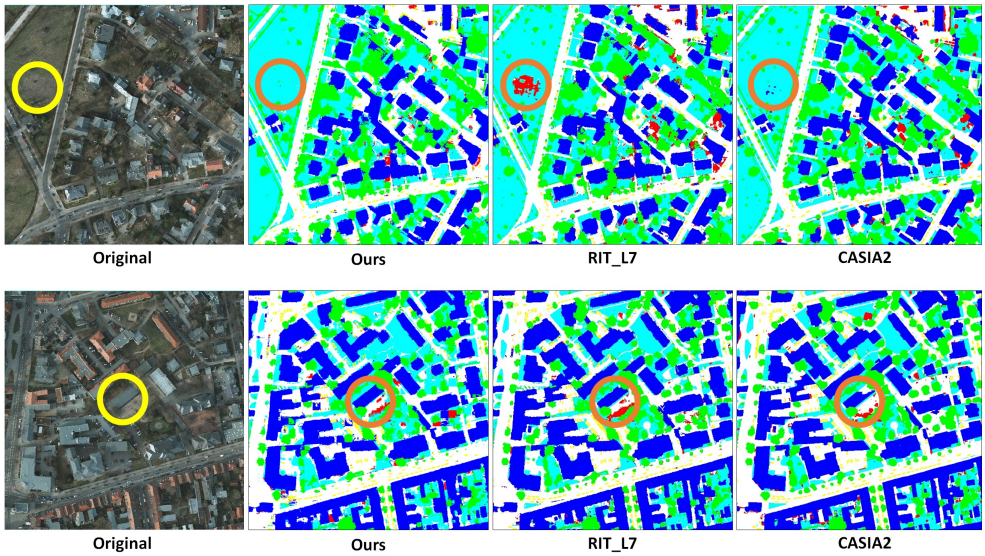

Figure 4.10: A qualitative comparison between our method to two submissions that uses 2D-deep networks. The yellow circles shows the original regions of interest, while the brown circles mark the corresponding regions in the classification maps.

tuned Resnet101 [72] and used only the spectral data, scoring an overall accuracy of 91.1%. The **CASIA2** method was chosen for comparison since it relies on a very large and computationally intensive Network, in contrast to our very compact network. For example, our network has about 1.9M parameters, while Resnet101 has about 44.5M parameters.

Figure 4.10 compares our output semantic labeling results against the **RIT_L7** and **CASIA2** methods, for two example 2D images. The circled areas in the upper image show that our method was able to correctly (see the corresponding spectral image) classify challenging low-vegetation regions, which were incorrectly classified as clutter by **RIT_L7**, and had residuals of the building class in **CASIA2**. Likewise, the circled regions in the lower image shows that both our method and the **RIT_L7** method produced good results when classifying the center building. The **CASIA2** method missed the top right corner of the building; this is likely due to a lack of consideration for height information. Including height information as another channel in a very large model such as Resnet101 is not a trivial task due to three-channel design. Also, even if height information were included as a fourth channel, finetuning would not be possible and training would be infeasible given the limited number of images provided in the contest. This highlights the advantages of our compact model: adding another feature is as simple as adding a single value per-point. And, given the relatively few number of parameters, a simple data augmentation is sufficient to train our

network.

4.3 Conclusions

In this chapter we presented a deep learning framework to semantically label 3D point clouds with spectral information. Our compact, 1D fully convolutional architecture directly consumes unstructured and unordered point clouds without relying on costly engineered features or 2D image transformations. We achieved near state-of-the-art results using only the 3D-coordinates and their corresponding spectral information. By design, our network can consume regions with varying densities, and is able to learn point-wise and block-wise features in an end-to-end fashion. Furthermore, our model is flexible, and can be readily extended to 2D semantic segmentation. Also, our experiments showed promising results when classifying unnormalized points. Given the compact, end-to-end framework, and fast testing time, our model has the potential to scale to much larger datasets, including those derived from optical satellite imagery. Future work will reduce the time and the memory needed for preparing the input data blocks, as well as improve the methods used to learn neighborhood features along with the point-wise and block-wise features. Additionally, we will extend the model to operate on optically derived point clouds, while learning the underlying digital elevation model from unnormalized points. These efforts are encapsulated in the final task (Chapter 5), which develops a framework to learn digital terrain models for point clouds derived from LiDAR and stereo photogrammetry.

Chapter 5

Task 3: Digital Terrain Modeling

While the task of semantic classification is relatively easy for the human visual and cognitive system, estimating the digital terrain model presents a more challenging task. Given a scene collected from an overhead platform, which often can only observe the uppermost surface, it is easier to visually recognize objects compared to estimating the Earth's elevation due to nadir view of the scene. With the dramatic improvements in the capacity to collect and interpret 3D data from Earth observing (EO) platforms, such complexity presents the human analyst with additional challenges. It was shown in the Chapter 4 that an accurate digital terrain model plays a critical role in improving the performance of the semantic classification task. Furthermore, the success of many applications (see Section2.4) hinges upon complex processing in order to translate raw 3D point data into actionable information products. This Chapter extends and build upon our work presented in Chapter 4 to estimate a point-wise DTM value. The algorithm seamlessly learns and combines neighborhood information with the corresponding point-wise and global features.

This research is driven by the following observations: In Section 2.4, we introduced prior research for DTM estimation, generally taking one of the following approaches: slope-based, triangulated irregular network (TIN)-based analysis, morphological filtering of a raster DSM, or more recently, machine learning. Based on the observations in Section 2.4, we conclude that there are still several limitations with respect to extracting the DTM from point cloud data. First, traditional methods based on slope-, TIN-, or morphological filtering rely on manually-tuned heuristics and do not scale well to variable data sets. As a result, they often require human intervention to achieve optimal results. This is exemplified by the numerous options and parameters presented to the user in software packages such as LASTools [102] or ENVI [103]. Second, most of the available methods rely on first classifying the points into ground and non-ground cover types; a potentially unnecessary step which adds complexity and computational cost. Third, learning-based methods (while having the potential to address the challenges above) are designed for 2D raster imagery, and require a costly data transformation process in order to operate on 3D point

cloud data directly. Together, these limitations hinder the ability to efficiently, accurately, and objectively extract DTMs from large-scale EO point cloud data for anywhere in the world.

To overcome previous issues in order to meet the demands of rapidly increasing data volumes and to address the needs of large scale exploitation while alleviating the burden on the human analyst, an automated and streamlined DTM extraction that operates directly on 3D point clouds is desired. We introduce a flexible and simple deep learning framework that directly estimates the DTM from 3D-point clouds obtained using either LiDAR or optical imagery. We extend PointNet [122] to learn neighborhood information while preserving the simple and elegant network architecture. For each point, the algorithm estimates the bare-Earth z-value directly, without the need to pre-classify the data into ground and non-ground cover types. This preserves the form of the point cloud data and avoids the additional error arising from misclassification.

This Chapter makes the following contributions, as related to the science questions presented for Task 3 in Section 1.2.3.

(1) All of the existing methods agree on the importance of using context within a neighborhood to decide whether a point is ground or not. For example, slope-based methods examine neighboring points to determine a height and a slope threshold, while morphological methods rely on the coverage area of the structural element to remove objects within a window. However, they require manual tuning of several parameters to achieve the desired results. Similarly, the learning-based methods define a window around the point of interest to capture contextual information before passing it to the network of choice. In this paper, we extend our previous work [133] to learn neighborhood information while preserving the simple and elegant architecture of the PointNet network [122].

(2) Although learning-based methods introduced the application of deep learning to estimate the DTM, they typically cast the problem as a binary classification task to distinguish between ground and non-ground points. In fact, a majority of existing methods rely on first classifying the points into ground and non-ground cover types. Such methods rely heavily on labeled points (e.g., tree, car, pavement) to train the algorithms. Here we design a learning-based method that bypasses the need for classifying ground points, thus eliminating the need for labeled training data. Instead, our model is trained on truth DTMs directly, and can subsequently be easily deployed on any point cloud for inference.

(3) Learning-based methods operate on a transformed version of the point cloud *i.e.* DSM images, in order to meet the requirements of deep learning architectures that were designed for 2D imagery. This transformation introduces unnecessary overhead and may introduce a loss of information. For example, if a DSM image is created from LiDAR points, one has to decide which return to use at a given pixel, as well as finding a way to fill in the void regions where data are missing. Similarly, when using optical photogrammetry, multiple points may exist at a given pixel; this requires a decision on which points are used in constructing the DSM. Here, we operate on the input point clouds directly, therefore respecting their unstructured and unordered nature.

In summary, inspired by the success of methods that operate directly on point clouds and allow the seamless integration of geometric and spectral information [133, 122], we make the following contributions:

1. We design a learning-based method that bypasses the need for classifying ground points.
2. Our algorithm predicts the DTM value for each input point within the 3D point cloud without the need for intermediate steps *e.g*. creating a TIN model.
3. We operate on the input point clouds directly, therefore respecting their unstructured and unordered nature.
4. We extend PointNet to learn neighborhood information while keeping the simple and elegant architecture of the network.
5. We eliminate the need for calculating handcrafted features derived from the DSM or fine tuning parameters that are scene specific, *i.e*. Top-Hat's structural element.

This Chapter is organized as follows: Section 5.1 will describe the network used and the extension of PointNet to handle neighborhood information. Sections 5.2 will present the evaluation of our approach. Finally, Section 5.4 draws the conclusions and presents our vision for future work.

5.1 Methodology

In this section we present a deep learning method that can be used to predict the DTM value for each point in an input point cloud. Recognizing the importance of spatial context, our method is able to learn point-level, neighborhood-level, and global features directly in an end-to-end fashion, rather than relying upon handcrafted features and hand-tuned parameters. The term 'global' in our case, represents the block-wise (*i.e*. the input set) features, unless stated otherwise in the text.

This end-to-end learning operates directly on the point cloud, without relying on derivative image-based representations (contribution 3). We do this by extending the PointNet architecture. Although one could design a simple, fully connected network that accepts a single point at a time, such a design would be limited in that it would only learn a single point-level feature for each point. Instead, PointNet proposed an effective, yet very simple, deep learning architecture that consumes 'blocks' of point cloud data and aggregates point-wise features into a global feature vector that represents the entire input point set (*i.e*. block-wise) features. This work showed promising results when applied to tasks such as semantic labeling of aerial LiDAR data my$_i sprs_p aper$.

However, the aggregation of point-wise features to create a global representation for the input block set suggests that the network doesn't capture the neighborhood contextual information.

Several attempts have been made to address this issue. For example, [134] proposed PointNet++ where they group a point set into smaller clusters, then apply PointNet recursively on a nested partitioning of the input set. Unfortunately, this modification to the architecture fails to preserve the point-wise features in their object classification architecture and the global features in their segmentation network.

In an attempt to retain the point-wise and global features of PointNet, [135] proposed the use of a kernel correlation layer to exploit local geometric structures using a graph-based approach. They construct K-nearest neighbor graphs (KNNG) by considering each point as a vertex, with edges connecting only nearby vertices. Using the graph information, kernels were learned and applied to the vertices to capture local information and concatenated with the point-wise and the global vectors from the original PointNet network. However, this method introduced unnecessary complexity and computational overhead.

Instead, we extend the original PointNet to allow the use of the point-wise, neighborhood, and global features, while retaining its simple, yet effective architecture (contribution 1). The complete architecture of our network is shown in Figure 5.1.

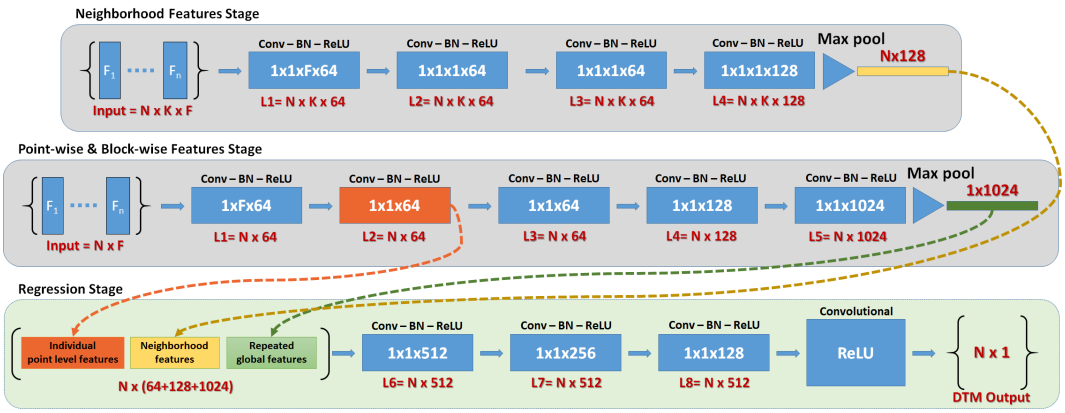

Figure 5.1: The network takes as input two representations of the points within a block. The first representation (center panel) is a set of size $N \times F$ used to learn point-wise features and later pooled to generate a global features for that block. The second set (upper panel) is of size $N \times K \times F$ and is used to learn neighborhood features within that block, where K is the number of neighborhood points (see 5.1.2). Next, features learned from both sets are concatenated and passed though (1×1) convolutional layers and then to a regressor to perform DTM prediction per point (lower panel), *i.e.* the $N \times 1$ output. This can be repeated for each block within the point cloud. Text in white indicates the filter size, while text in red indicates the layer's output shape.

5.1.1 Network Architecture

Our network takes as input *a pair* of blocks. The first is a block X of N 3D-points, where X = $\{x_1, x_2, x_3, ..., x_N\}$. The second set, however, is a $N \times K \times F$ array representing the neighborhood data within that block. Note that this is merely an alternative structuring of X, just grouped and replicated in such a way as to provide spatial context. Further details on the neighborhood block will be explained in Section 5.1.2.

In addition to spatial coordinates, the points in block X potentially have point-level feature attributes such as return number, intensity and/or spectral information, such that x_i is of length F where F is the number of feature attributes per point, e.g., $x_i = \mathrm{X}_i, \mathrm{Y}_i, \mathrm{Z}_i, \mathrm{R}_i, \mathrm{G}_i, \mathrm{B}_i$. Given the truth terrain height Y = $\{y_1, y_2, y_3, ..., y_N\}$ for the training set, the goal of DTM estimation is to predict the z-value for the ground-level at every point in the test set, *i.e.* $\hat{Y} = \{\hat{y}_1, \hat{y}_2, \hat{y}_3, ..., \hat{y}_N\}$. For each input set, the network ingests this $N \times F$ array of unordered data points X, (along with an $N \times K \times F$ array representing the neighborhood data) and outputs a $N \times 1$ array $\hat{Y}_i$ where each entry is the DTM prediction per input point. This can be repeated for each input set (*i.e.* a block of data), with the results aggregated so as to return the DTM estimates for each point in the entire point cloud. For a given scene, we subdivide our training data into overlapping blocks, thus increasing the number of training samples.

Although the network is designed to flexibly accept inputs with different numbers F of features, we demonstrate the network using the following 10 features given the available data: Each point x_i is represented as a 10D-vector, containing the block-wise centered coordinates (X, Y, Z) representing the normalized coordinates with respect to the input block, spectral data (IR,R,G), the Normalized Difference Vegetation Index [136] (NDVI), and the normalized coordinates ($\bar{\mathrm{X}}, \bar{\mathrm{Y}}, \bar{\mathrm{Z}}$) with respect the full extent of the scene.

To learn point-wise features, the $N \times F$ array of points from each block of data is convolved with a set of 1D-filters of size $1 \times F$, *i.e.* the filters are applied across the the columns to capture the interactions between the point attributes as shown in the central portion of Figure 5.1. It should be noted that this operation can be carried out using a 2D-filter while making sure that the filter is applied only in one dimension. This allows us to define 1D-filters in the form on ND-filters, thus allowing a simpler implementation, by ensuring that the convolution operation is applied on a specific axis. The output of this operation is an $N \times F'$ array with F' representing the new point-wise features. To capture an embedding that describes the whole input set, *i.e.* the block, the point-wise features are aggregated using a pooling operation (*e.g.* maxpooling, average pooling, or weighted average pooling) to a single vector. In this stage (center panel in Figure 5.1), the network can only learn point-wise features and a global feature vector per input block. Note the similarity to PointNet.

5.1.2 Learning Neighborhood Features

It was pointed out in Section 2.4 that neighborhood relationships are critical in estimating the DTM value at a given point [108, 104]. To allow our network to capture neighborhood information, we need to first determine which points are within a 'neighborhood'. Several options exist. Using a Kd-tree search, one can either find the closest K points to the query, or define a search area and collect all points within that area. In the first case, while the number of neighborhood points is fixed, the points themselves may be found at varying distances from the query point. Alternatively, while a predefined search area allows for neighborhoods at different scales, the number of points found within a fixed search area may be different from one point to another.

Recognizing these trade-offs, we take the following approach: For every point x_i in the input set where $i = \{1, 2, 3, ..., N\}$, we define the neighborhood as the set of K sampled points found within a sphere centered at x_i with a radius R. This ensures that the total number of points within a fixed radius will be the same for all points; we replicate points if the available number of samples is less than K.

In this stage (upper panel in Figure 5.1), each individual input point within the block can be represented by its neighborhood points in the form of an $N \times K \times F$ array. Recall that N is the number of points within the input block, K is the number of neighborhood points, and F is the number of attributes per point within the neighborhood. Note that in PointNet++, the algorithm first subsets the original input data before finding the neighborhood; this results in an array of $N' \times K \times F$ where $N' < N$. However, as opposed to classical computer vision tasks such as object classification, where fewer points can describe an object, estimating the DTM is done at every point in the scene. Therefore, we choose not to exclude points from the input data set.

Next, the $N \times K \times F$ input array is convolved with a set of F' 1D-filters along the feature dimension, *i.e*. a single filter shape is $1 \times 1 \times F$ as shown in Equation 5.1,

$$G_{(N,K,F')} = X_{(N,K,F)} * h_{(1x1xF)} \tag{5.1}$$

where G is the output of the convolution process with a shape of $N \times K \times F'$, and F' is the number of new features learned (*i.e*. embeddings) for every point within the neighborhood. At this stage, every point in the input array is still represented by the corresponding features embedded in the collection of K neighboring points.

In order to summarize this neighborhood information for every input point, we aggregate/pool the information from all neighboring points to form a single vector per-point that encodes the neighborhood information as follows:

$$G' = max\ (0, G, axis = 1) \tag{5.2}$$

where G' is an array of size $N \times F'$ and each input point is now represented by the encoding of all the features within the neighborhood using the max operation along the neighborhood axis.

The result from Equation 5.2 is then concatenated with both the point-wise features and the global feature vector from the original PointNet architecture as show in Equation 5.3

$$T = concat\ (P, B, G') \tag{5.3}$$

where P is the point-wise feature vector, B is the global feature vector, and G' is the neighborhood vector. These are represented in Figure 5.1 as the orange, yellow, and green boxes, respectively.

It should be noted that PointNet++ aggregates the information from the neighborhood data only to produce the final representation without including the point-wise and the global feature vectors found using the original PointNet architecture. Figure 5.2 shows the difference between our architecture and PointNet++ presented by [134]. As shown, we learn neighborhood information for every point within the block, instead of for just a sampled set of the points from the block. In addition, we learn point-wise and global features.

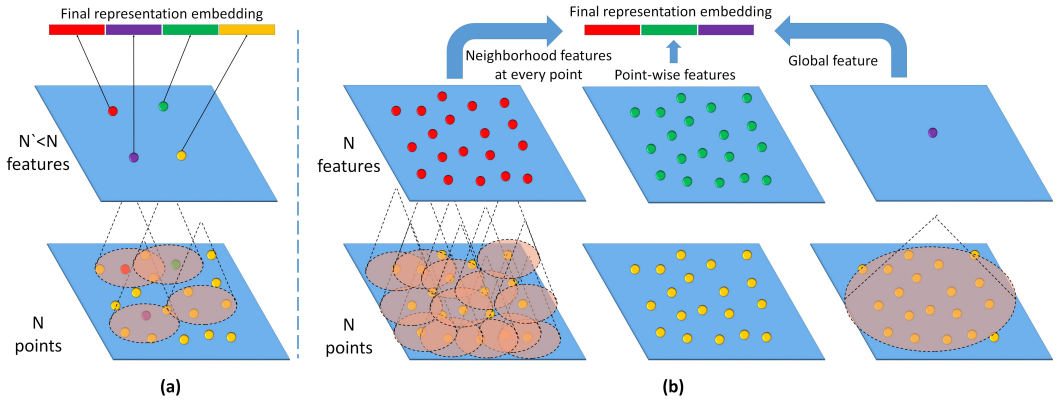

Figure 5.2: The figure shows the difference between PointNet++ (a), and our network (b), in terms of the final feature representation. Our network combines all point-wise, neighborhood, and global features from all input points, while PointNet++ only aggregates the information learned from a sampled set neighborhood to create a final representation. The blue tile represents a block

5.1.3 Inference

The output from Equation 5.3 is passed through a series of a 1×1 convolution layers and finally to a linear layer with a ReLU activation as shown in Figure 5.1 (bottom panel). The ReLU activation forces the outputs to be greater than or equal to zero. This guarantees that all the DTM estimations

are positive. This design can be thought of as solving the following constrained optimization task:

$$\begin{aligned} \underset{w}{\text{minimize}} \quad & cost\ (Y_i, f_i(X; w)) \\ \text{subject to} \quad & f_i(X; w) \geq 0 \end{aligned} \tag{5.4}$$

where Y_i is the ground truth values, $f_i(X; w)$ is the network prediction, X is the input array, and w are the network parameters to be learned.

To train the network, two popular loss functions can be considered; Mean Squared Error (MSE) and Mean Absolute Error (MAE). An advantage of using a MSE loss is the convex nature of the function, which makes is easier to optimize with linear derivatives. However, since the errors are squared, it is very sensitive to outliers, which are common in LiDAR data due to multiple scattering, or in optical point clouds due to imperfections in dense matching (*e.g.* spikes) from multi-view imagery. To avoid biasing the prediction based on these outliers, while ignoring other good samples, one can use the MAE loss function. However, since this loss function uses the absolute value operator, it is not differentiable at zero, and the derivatives are constant on both sides of the origin due to the linear nature of the function. Seeking a trade-off between these loss functions, we choose the logarithm of the hyperbolic cosine function as shown in Equation 5.5, which offers a balance between the MSE and MAE behaviors.

$$\ell = \sum_{i=1}^{N} \log(\cosh\left(Y_i - \hat{Y}_i\right)) \tag{5.5}$$

It is a convex function with a quadratic behavior at small values, allowing for non-constant derivatives in this range, while providing a linear behavior at large values. This makes it insensitive to outliers while remaining fully differentiable.

5.2 Evaluation

5.2.1 Dataset

For this paper, we use two different sources of point cloud data, comprising 4 individual test sets. The first set is LiDAR data provided by the ISPRS 3D-Semantic Labeling Contest, as part of the urban classification and 3D-reconstruction benchmark[1]. The remaining three sets are sample point clouds derived from dense stereo matching using multiple-view imagery available from DigitalGlobe and processed by Vricon [137].

The LiDAR data were acquired using a Leica ALS50 laser scanner flown at a mean height of 500m above ground level over Vaihingen, Germany. This resulted in a point density of approximately 4 points/m^2. Additionally, corresponding georeferenced IR-R-G imagery was provided,

[1] `https://goo.gl/6iTj6W`

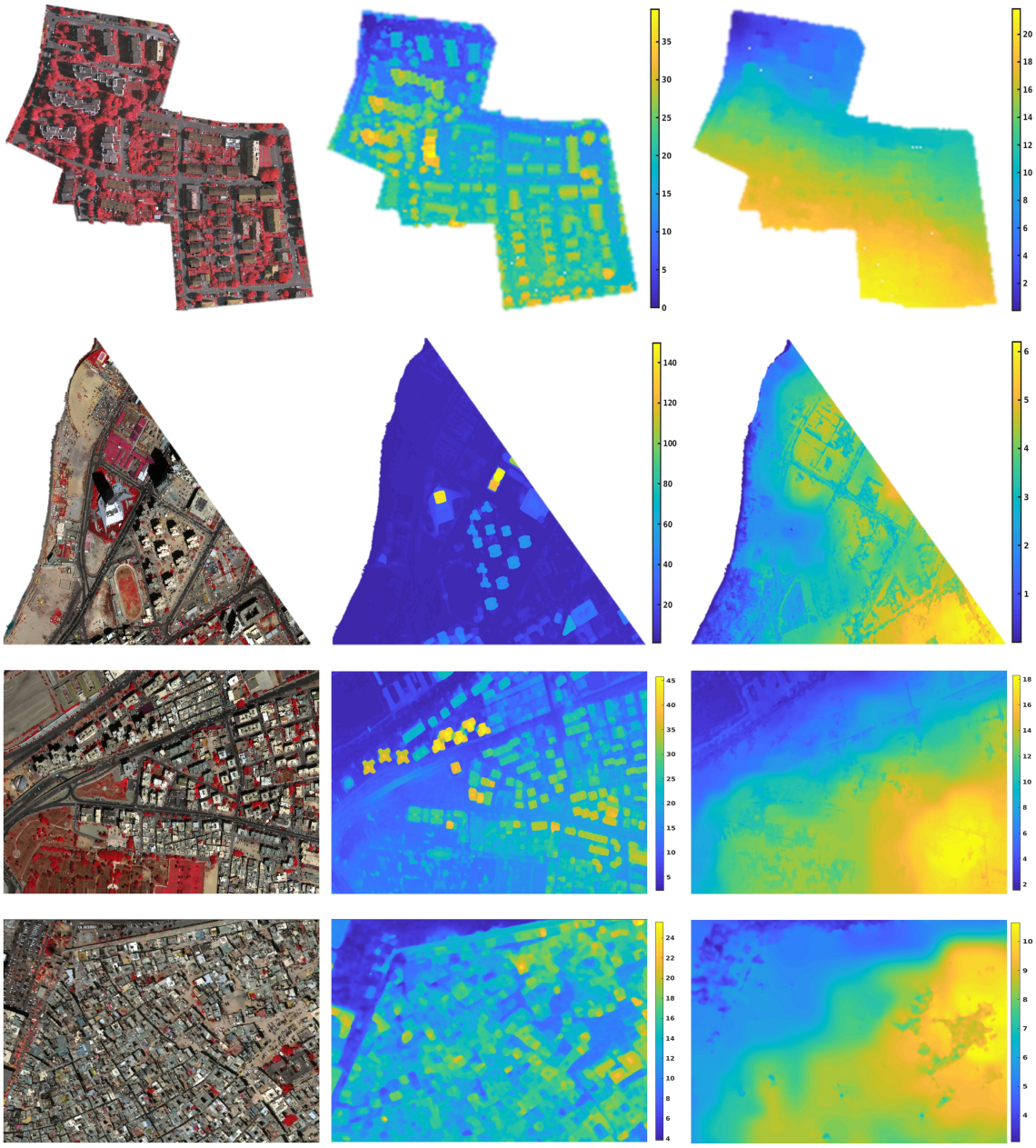

Figure 5.3: Training data sets, from top to bottom: ISPRS LiDAR data, Vricon high-rise area, lower extra-urban data, and historic old-city area. For each, we render, from left to right: point cloud colored using the spectral attributes, the DSM (showing height information), and the ground-truth DTM.

with a ground sampling distance of 8cm and dimensions of 20250×21300 pixels. We used the image data to extract the spectral attributes for each point in the LiDAR data.

The LiDAR coverage area was subdivided into two regions; one for training and the other for testing, with a total of 753,859 and 411,721 points, respectively. The training area is comprised mostly of residential detached houses. It covers an area of 399m $\times$ 421m. The test area, on the other hand, is located at the center of the Vaihingen city proper, and is representative of complex urban structures. It covers an areal extent of $389m \times 419m$.

The dense optical point clouds were generated by Vricon, which employs a dense reconstruction from multiple-view satellite imagery [138, 139]. High-resolution, multispectral imagery from DigitalGlobe's constellation of WorldView satellites [140] provides a nominal post spacing of 0.5m with spectral features in the R,G,B, and IR bands.

Three scenes were used, all based in in Tripoli, Libya. The first scene, of size 1000m$\times 1300$m, contains 3.7 million points within an extra-urban area, with many high-rise buildings distributed about a large open area. The second scene, of size 1500m $\times$ 500m contains 3 million points and is representative of smaller urban structures with wide roadways. The last scene covers the historic old town region of the city, with densely-packed structures and very little ground visible except in sparsely distributed areas. This scene covers an area of 650m $\times$ 450m with 1.1 million points. All of the image-derived point clouds have a point density of 4 points/m^2.

For the purposes of this study, we have subdivided these scenes into training and testing areas, each containing 50% of the data. Figure 5.3 shows the training data set including point clouds and their corresponding DSM and ground-truth DTM for all four scenes. Figure 5.4, likewise, shows the testing data set.

The ground truth data were obtained as follows: For the ISPRS data, we use the DTM generated by LAStools as our ground reference. For the optical image-derived data, we use a DTM of 0.5m resolution, provided by Vricon. We acknowledge that there may be errors in both DTM sources, in particular for the LAStools output which is not subject to production-level quality control. However, both data sources add unique value to this study: The ISPRS data are used due to its recent popularity as a benchmark for the semantic labeling tasks, allowing us to share our findings on a recent publicly available dataset. And the Vricon data are used to evaluate the network on 3D data obtained using optical photogrammetry, thus demonstrating the feasibility for large-scale DTM extraction from EO satellites.

5.2.2 Preprocessing

During training and testing, we represent each point by its 10D-vector (X, Y, Z, IR, R, G, NDVI, $\bar{\text{X}}$, $\bar{\text{Y}}$, $\bar{\text{Z}}$) (see Section 5.1 and Figure 5.1 for details). For the LiDAR data, the spectral information is extracted by interpolating the georeferenced IR-R-G imagery at the point locations for the LiDAR-derived point cloud. For the optical point clouds extracted from multiple-view imagery,

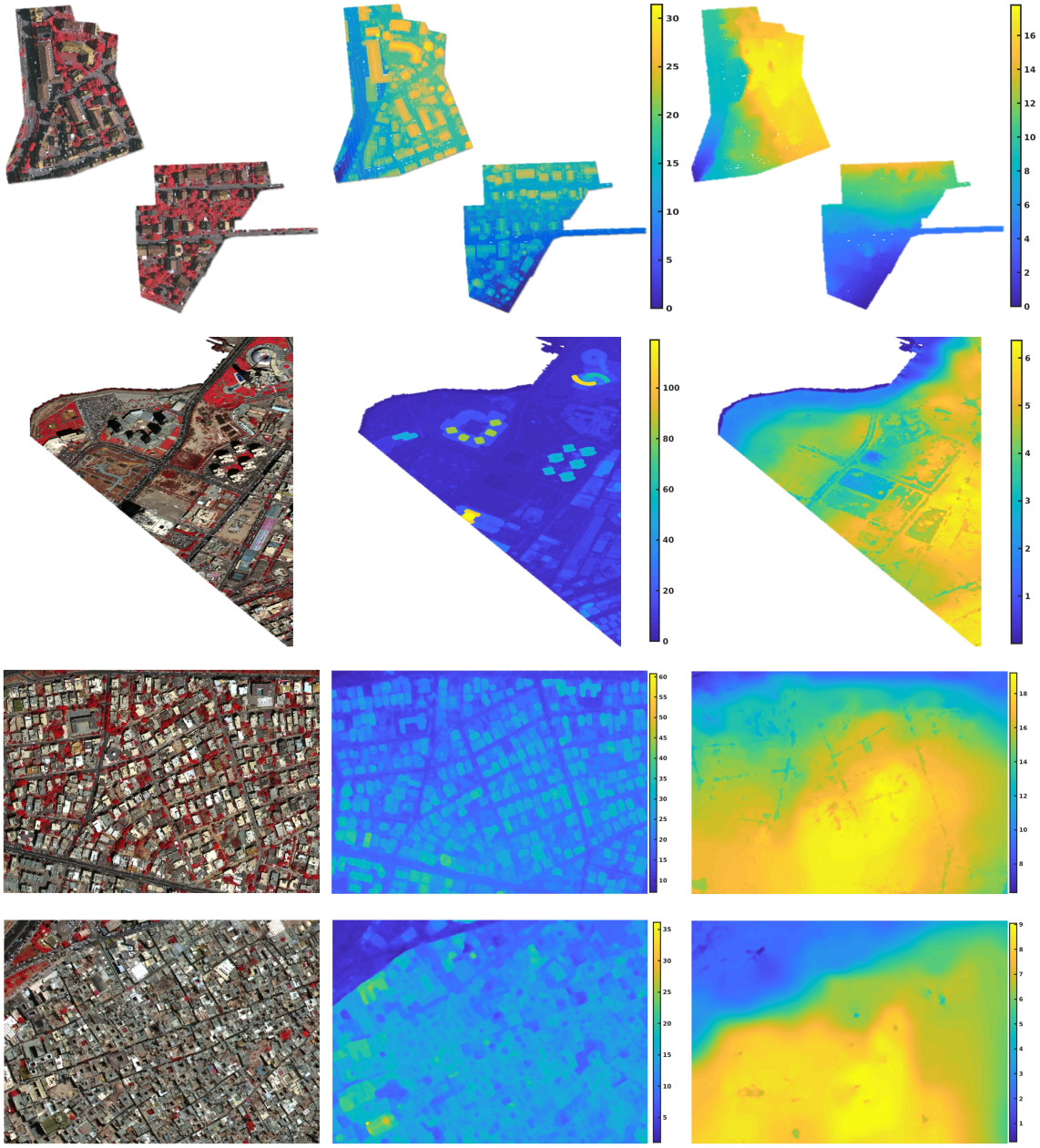

Figure 5.4: Testing data sets, from top to bottom: ISPRS LiDAR data, Vricon high-rise area, lower extra-urban data, and historic old-city area. For each, we render, from left to right: point cloud colored using the spectral attributes, the DSM (showing height information), and the ground-truth DTM.

this spectral information is inherently defined for each point.

The validation data are created by sampling the training data. The training data are augmented by randomly rotating the points to alter their geographic orientation around the z-axis. We also add random noise to the coordinates after each rotation. The additive noise is sampled from a zero-mean Gaussian distribution with $\sigma = 0.08$m for the x, y coordinates and $\sigma = 0.04$m for the z coordinates. These values were chosen empirically such that the overall distribution of the data are similar while adding sufficient noise. The original training data are discarded and only the augmented data are retained to avoid similarity between the training and validation data.

Since deep learning models require large amounts of training data, we subdivide the scenes into smaller blocks with 50% overlap. [133] showed that the size of the blocks affects the classification quality due to the effect of the global feature learned from each block. They used $2m \times 2$m, 5m$\times 5$m, and 10m$\times 10$m blocks to capture objects at small, medium, and large scales. However, in the task of estimating the DTM, we are interested in a more synoptic view of the scene to capture the changes in the terrain. Therefore, the 10m $\times$ 10m block size was chosen for all experiments. During training, we instantiated each block with N=2,048 points sampled from within each block for computational purposes. Note that we operate on all 2,048 points, while PointNet++ selects a smaller subset out of this input to learn the neighborhood information. During testing, we use all the points within a block. By fixing the block size, we are limiting the network to learn only from the geographic area delineated by that fixed-size block. To overcome this issue, our network is designed to accept neighborhood information that complements the point-wise and the global features learned from the blocks. The user has the flexibility to choose the neighborhood radius, depending on the task.

5.2.3 Training Parameters

To generate the $N \times K \times F$ array containing neighborhood relationships, we define a search radius R=25m and sample K=128 points from within each spherical region, thus creating an array of dimensions $2,048 \times 128 \times 10$ where each of the original 2,048 points within a block is represented by 128 points from its 25m neighborhood, each with a 10-dimensional feature vector. While sampling 128 points from the 25m radius area seems very small, applying this for every point within the block sufficiently covers the neighborhood when viewed together.

During the training process, we use the Adam optimizer [128] with an initial learning rate 0.001 and a momentum of 0.9. Since the inputs to the network are a 2048×10 array and a $2048 \times 128 \times 10$ array, we choose a batch size of 12 due to memory limitations in our current computing environment. The learning rate is linearly reduced after each iteration as a function of the current number of epochs. We train the network for a total of 16 hours for 20 epochs and monitor the progress of the validation loss to save the weights as the loss improves. We train using an NVIDIA Tesla P40 GPU and Keras [129] with the Tensorflow backend.

5.2.4 DTM Results

For each of the four testing scenes, we estimate the DTM $\hat{Y}_i$ at every i^{th} point within the scene and compare that to ground-truth data Y, to generate an error map, $|Y - \hat{Y}|$. During testing, we subdivide the data into $10\text{m} \times 10\text{m}$ blocks, similar to training, and report the DTM estimate for all input points. These summary results are shown in Figure 5.5; the bottom panel shows the original DSM and colorized point cloud, for reference. Note that these results are for a neighborhood radius of 25m. More information on the impact of the neighborhood radius is presented in Section 5.3.1. It should be noted that our results were reported without any post-processing, *i.e.*without smoothing the predicted DTM. Although post-processing could improve the results, our intent here is to present our findings in their original form.

From these error maps, we calculate the mean absolute error (MAE) and the standard deviation σ of the error between the ground truth DTM values Y and the predictions obtained from our network $\hat{Y}$. Quantitative results are summarized in the right-most column of Table 5.1.

Furthermore, to benchmark our DTM extraction method against alternative solutions, we also compute the MAE and σ values for the results obtained from two widely-used software packages for DTM extraction from point clouds, namely ENVI [103] and LAStools [102]. These results are presented in the center columns of Table 5.1. Note that this comparison is only done for the Tripoli scenes with Vricon ground truth to avoid a circular comparison (the only available ground truth for the LiDAR dataset was also from LAStools; see Section 5.2.1). So although MAE and σ values are not provided for the LiDAR dataset, in Section 5.3.1 we will show how the LiDAR dataset allows us to examine the effect of the neighborhood size.

A note on the settings we used when evaluating the software packages: Both LAStools and ENVI present the user with multiple tunable parameters. For example, with LAStools, various parameters are required, including 'step' (*i.e.* wilderness, nature, town or flats, city, and metropolis), 'bulge' (how high the initial DTM estimate is allowed to bulge upwards during the refining stage), ground point search (controlling the quality of the initial ground points estimation, and represented by values such 'fine, hyper, and ultra'), and 'spike' (determining how prominent a spike can extend). Similarly, ENVI presents the user with multiple parameters, the three most important being: grid resolution, point filtering, and the variable sensitivity. We want to compare our method against the best possible results from these software packages. Therefore, we ran multiple parameter combinations in order to find the best-performing parameters for each set of training data, and then used these best-performing parameters on the corresponding test data.

Visual results comparing our method to LAStools and ENVI are shown in Figures 5.6, 5.7, and 5.8 for the old-city, urban and high-rise areas of Tripoli, respectively. Note that we have matched the color scale of all predicted DTMs to the ground truth DTM; although this saturates any predicted values that are beyond the maximum of the ground truth, it allows us to visually compare the changes with a common reference.

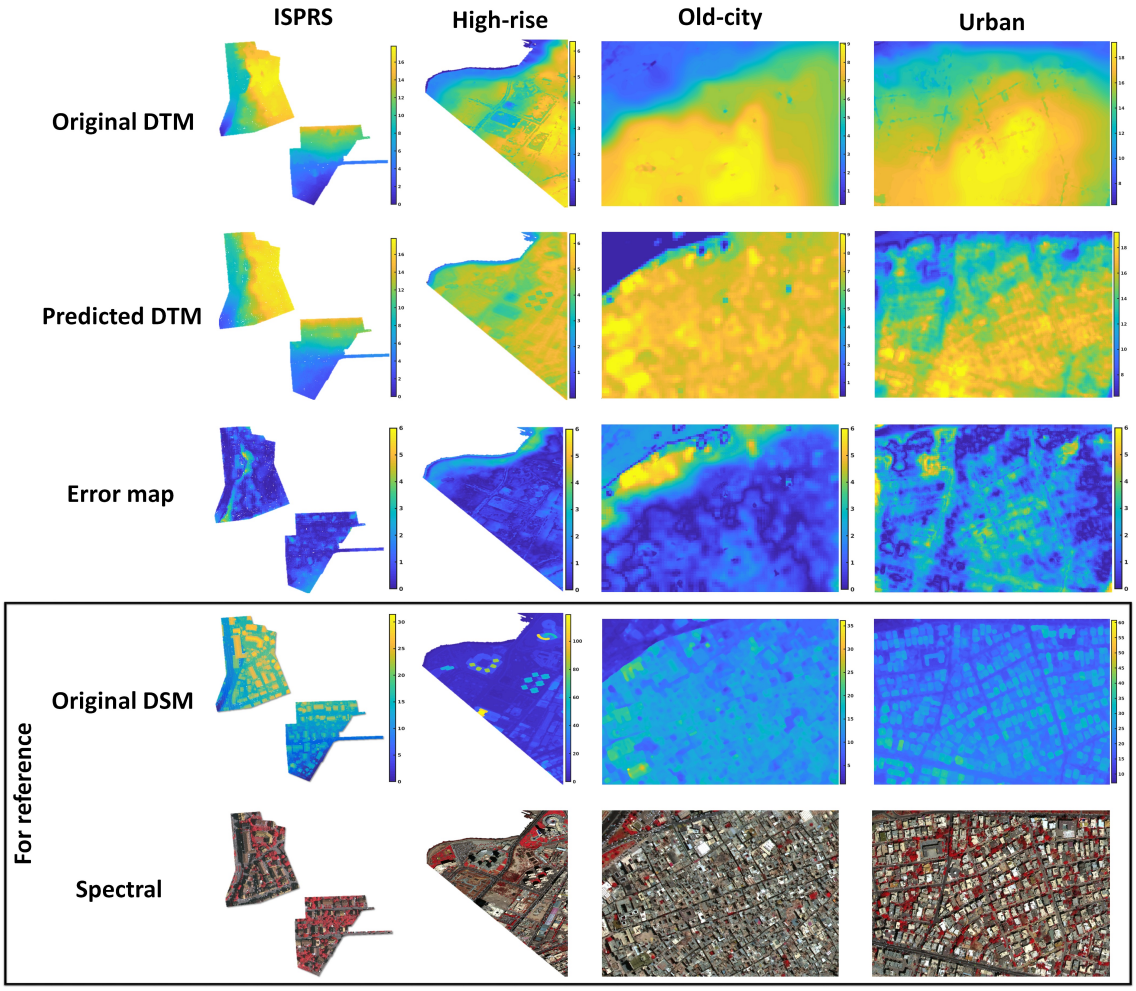

Figure 5.5: For each of the four testing scenes, we show the estimated DTM $\hat{Y}_i$ in comparison to ground-truth DTM Y. The difference gives an error map, $|Y - \hat{Y}|$. The original DSM and colorized point cloud is shown in the bottom panel for reference.

Table 5.1: MAE in meters and the corresponding standard deviation (σ) of the error between the ground truth DTM values (Y) and the predictions obtained from our network ($\hat{Y}$).

Scene	Metrics	LAStools	ENVI	Ours (25m)
Old-city	**MAE**	2.00	3.90	1.17
	σ	1.37	1.8	1.2
Urban	**MAE**	1.80	2.0	1.91
	σ	1.23	1.4	1.1
High-rise	**MAE**	1.22	2.42	0.77
	σ	1.1	3.02	0.72
ISPRS	**MAE**	-	-	0.70
	σ	-	-	0.75

5.3 Discussion

For the old-city scene (Figure 5.6), the best parameters in LAStools were obtained using 'city or warehouses', with 'bulge' is set to False, ground point search set to 'hyper', and a 0.2m 'spike'. For ENVI, the best results were obtained using the default parameters with the exception of setting the variable sensitivity option to 'flat'. As shown in Table 5.1, our method reported a MAE of 1.17m, outperforming LAStools and ENVI, which obtained results of 2.0m and 3.9m, respectively. This is a promising result given the challenging scene content, *i.e*. there are very few regions in the scene where the bare-ground is visible, due to the densely-packed structures. Methods that rely on slope- or TIN-relationships, and on ground vs. non-ground classification typically over-estimate the DTM in this sort of scene, due to the likelihood of misclassification and the lack of abrupt surface discontinuities. These blunders are most apparent in the ENVI prediction.

Also, it is clear that both methods rely heavily on local minima within the scene (see the error maps in Figure 5.6). This results in triangulation artifacts during TIN creation. This is most apparent in the LAStools error map where the final DTM is created by 'growing out' from local minima in the form of triangles. This has the effect of creating ramp-like triangles, which may seem like a smoothing effect, but actually introduce fake slopes that potentially tilt flat surfaces (*i.e*. rooftops).

On the other hand, our method bypasses the classification stage and predicts a per-point DTM value utilizing the neighborhood information. Therefore, our predictions and their corresponding errors are localized (*i.e*. point-wise), instead of being generated due to surface interpolation. This advantage allows us to achieve the lowest standard deviation in this scene of 1.2m compared to 1.37 and 1.8 for LAStools and ENVI respectively.

Because of this point-wise approach, we see different behavior in the top-left region of the scene (marked with a red box). In particular, we see that our approach predicts a sharp height transition, while LAStools and ENVI predict a slowly-varying elevation profile due to their use

of surface interpolation. However, this smooth transition in the ground truth DTM can not be explained by manually investigating the input DSM: The lowest DSM height values in the residential area adjacent to the roadside are over 7m high; we have confirmed that these are true ground surfaces by manually examining the DSM and spectral imagery. However, the ground truth DTM as prepared by Vricon estimates a slowly-varying elevation profile, with height values of 2-3m in this same area. While it is not clear how this ground truth DTM was generated, this difference explains the behavior of our method in this region. This observation also explains the large error for all methods at the transition area.

For the urban scene (Figure 5.7), the scores for all three methods were relatively close to each other. Our method achieved a MAE of 1.91m, only 11cm higher than LAStools with a MAE of 1.8m. ENVI, on the other hand, scored a MAE of 2.0m, as seen in the second row of Table 5.1. For LAStools, we obtained the best results using 'town or flats' without bulge and a 0.2m 'spike'. Similarly, for ENVI we used a 1m grid with urban filtering, and with the variable sensitivity option turned off.

The results from both ENVI and LAStools exhibited piecewise-linear predictions. This effect is more visible as 'patches' in the ENVI DTM result due to TIN interpolation. Note that some areas are saturated due to the consistent color scale with. However, the error maps show that LAStools and ENVI exhibit large error values compared to ours. This is most apparent at the top right corner and at the bottom of the scene. We believe this is due to surface interpolation around misclassified ground points. On the other hand, the errors from our method exhibit lower error values due to the localized predictions. This observation is supported by the standard deviation values shown in Table 5.1. Our method achieved the lowest standard deviation of 1.1m, while LAStools and ENVI achieved 1.23m and 1.4m respectively. It is interesting to note that our network was able to achieve low error values for points representing the roadways (see error map) without having to initial classify them. This indicates that the network has the ability to learn that road regions are good candidates for bare-Earth points.

Finally, for the high-rise scene in Figure 5.8, we scored the lowest MAE of 0.77m, outperforming LAStools and ENVI with MAE values of 1.22m and 2.42m, respectively. The predicted DTM maps for both LAStools and ENVI show large areas of saturated values (*i.e.* values larger than the highest ground truth value) compared to our predictions. The error maps show that ENVI struggled most at the edges compared to LAStools and our predictions, with ours showing the lowest error. The error maps also show that both LAStools and ENVI exhibit high error values around unconventional structures (see arc-like buildings within the red box), while our method was able to achieve a low error for the same region. The middle region (marked by the black box) shows that our DTM predictions at the locations of the high-rise buildings are lower than the ground truth. However, by examining the ground truth DTM, one can notice that our estimated elevation for the building footprints actually matches the surrounding areas within the ground truth DTM.

In summary, the results show that our method can achieve an overall average prediction error

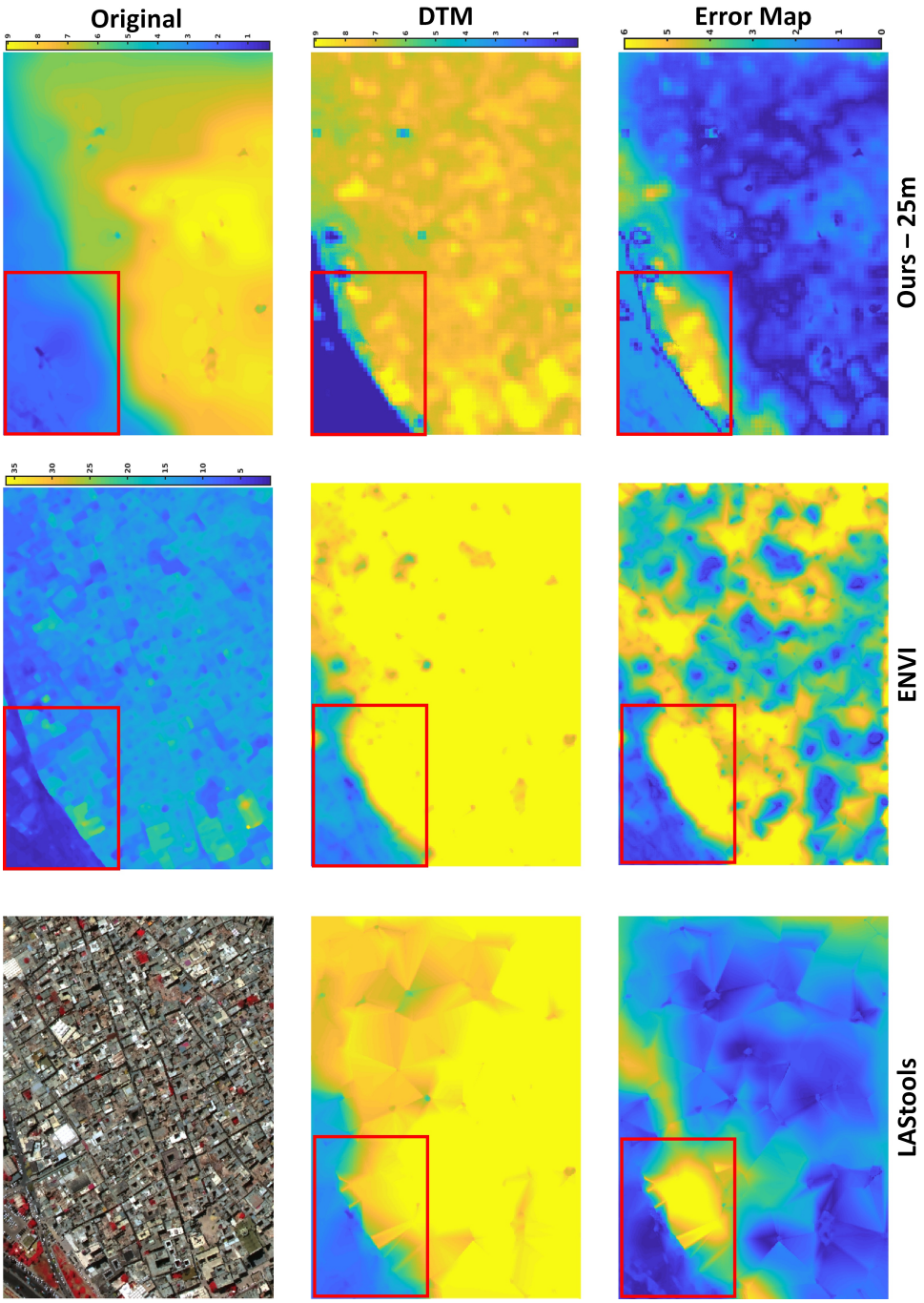

Figure 5.6: Comparison of results for the Tripoli old-city scene. The first row, from left to right, shows the original color point cloud and the corresponding DSM and ground truth DTM. The second row shows the estimated DTM obtained using LAStools, ENVI, and our method. By subtracting these from the ground truth (top-right) we obtain the corresponding error maps for each method (bottom row).

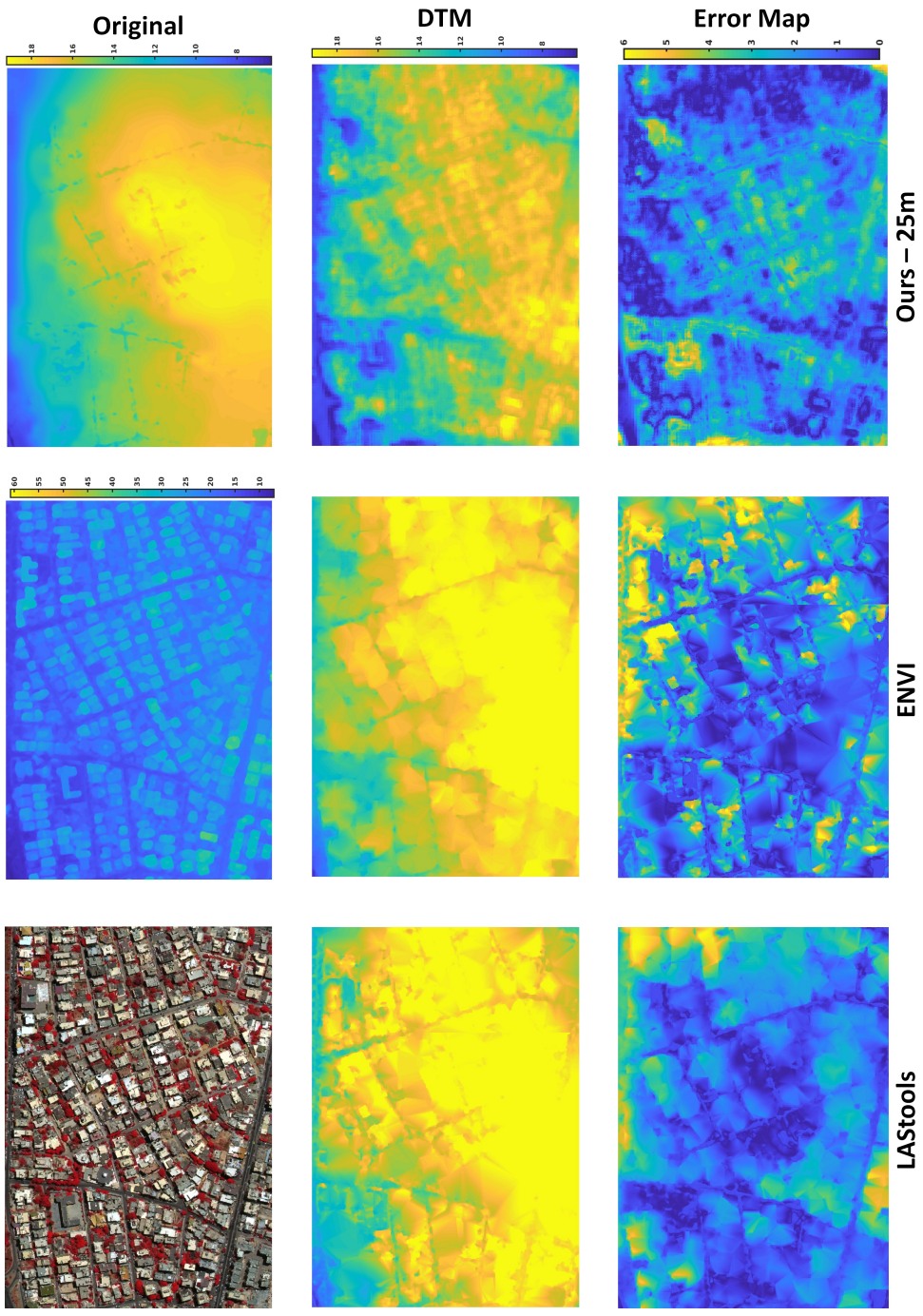

Figure 5.7: Comparison of results for the Tripoli urban scene. The first row, from left to right, shows the original color point cloud and the corresponding DSM and ground truth DTM. The second row shows the estimated DTM obtained using LAStools, ENVI, and our method. By subtracting these from the ground truth (top-right) we obtain the corresponding error maps for each method (bottom row).

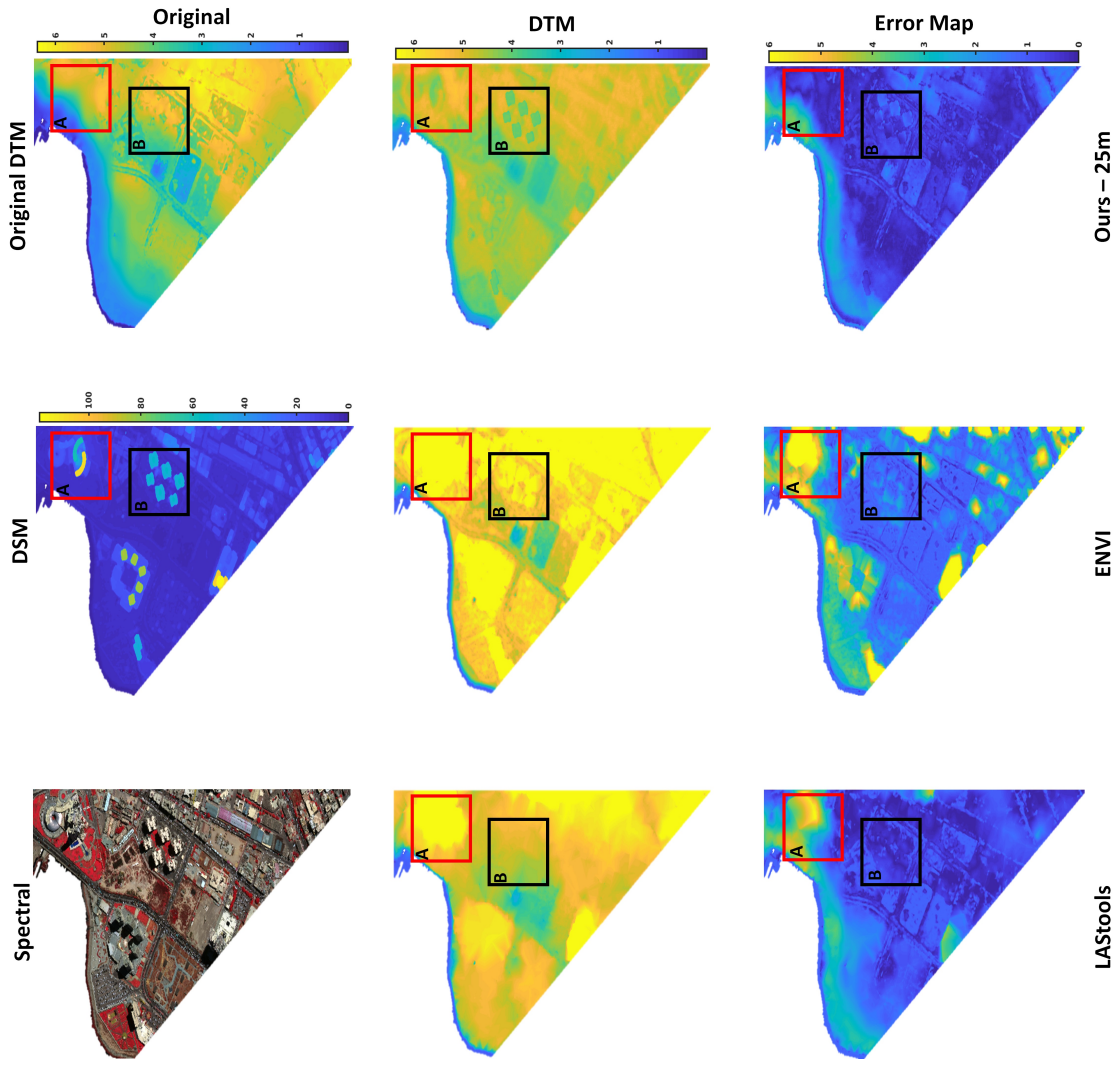

Figure 5.8: Comparison of results for the Tripoli high-rise scene. The first row, from left to right, shows the original color point cloud and the corresponding DSM and ground truth DTM. The second row shows the estimated DTM obtained using LAStools, ENVI, and our method. By subtracting these from the ground truth (top-right) we obtain the corresponding error maps for each method (bottom row). Red and black boxes are marked with ‘A‘ and ‘B‘ respectively for printing purposes.

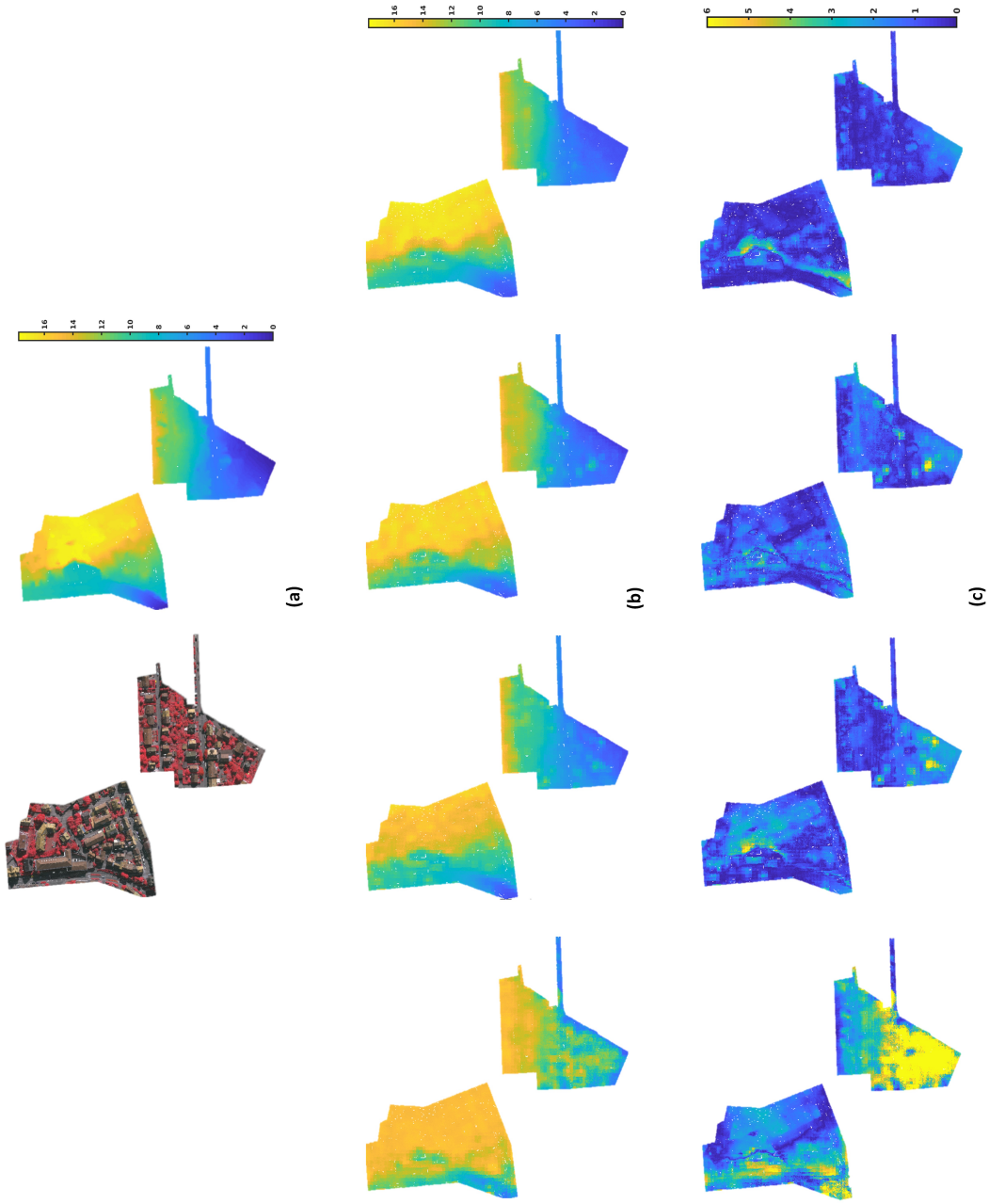

Figure 5.9: In (a), the figure shows the original colored point cloud and the corresponding DTM map. In (b), the predicted DTMs are shown for 5m, 12m, 18m, and 25m radii. In (c), it shows the corresponding error maps.

of 11.5% (*i.e.* dividing the MAE by the maximum value of the ground truth DTM) without post-processing, while LAStools and ENVI achieved an error of 16% and 29%, respectively.

5.3.1 Effect of Neighborhood Size

In Section 5.1.2 we described the importance of choosing a neighborhood radius R, in order to learn contextual information. To evaluate the effect of neighborhood radius on DTM results, we trained four different networks, each with a different neighborhood size: 5m, 12m, 18m, and 25m. Larger radii were not included due to memory limitation. The quantitative results for this experiment were reported without any post-processing as well, to allow for a fair comparison and eliminate factors that might bias the results. The quantitative results are shown in Table 5.2.Figure 5.9, shows the predicted DTM for the different neighborhoods radii, and the corresponding error map. The result from the 5m radius contains the most noise among the different neighborhood sizes. However, increasing the search radius to 12m and 18m shows better predictions with much cleaner results compared to the 5m neighborhood. Finally, the result for the 25m radius shows the best overall prediction among all. While the 5m setting showed the worst results, it very important as it helps us interpret the network's behavior.

Table 5.2 shows the MAE and the σ for each neighborhood size. By examining Figure 5.9, the error map for the 5m radius shows the highest overall error values in many regions with a MAE of 3m. However, such regions are mostly buildings (see Figure 5.5 for reference DSM and colorized point cloud). This is interesting as it indicates that having a small radius is not sufficient to capture the changes in the terrain. Instead, with such a small radius, the neighborhood points will only describe the surface they cover. Increasing the radius allows larger neighborhoods to potentially include terrain points, as shown for the 12m and 18m radii, with MAE of 1.2m, and 1m respectively. Using a 25m radius allow the neighborhood region to contain a larger area than the 12m and 18m radii, therefore, resulting in lower error values with a MAE of 0.7m.

Table 5.2: The effect of varying the neighborhood search radii on the corresponding MAE and σ values using the ISPRS data.

Radius	**MAE**	σ
5m	3.0m	2.10m
12m	1.2m	0.92m
18m	1.0m	0.71m
25m	0.7m	0.75m

5.4 Conclusions

In the is Chapter, we introduced a method to estimate the DTM, bypassing the classification stage and operating directly on the point cloud. Our method extends previous work by learning neighborhood information and seamlessly integrating this with point-wise and block-wise global features. We achieved a lower overall error compared two widely used software packages for DTM extraction, namely LAStools and ENVI. Furthermore, we showed that using a larger neighborhood area results in a lower error as compared to a smaller neighborhood. Future work will address the following remaining gaps: First, while our results show promising performance, it is important to collect more data in order to thoroughly test this approach. Furthermore, incorporating more data into the training process, will enable our network to handle more diverse scenes. Our vision is to develop a unified framework that can use a single network to infer the DTM for solutions at scale. This will benefit both the average user and geoprocessing pipelines by eliminating the need to manually adjust scene-specific parameters. Second, in this paper, our network was trained using a direct mapping to a ground truth DTM. We are currently investigating another loss function that may improve our DTM estimation by introducing a correction factor using other point cloud products (*e.g.* nDSM). Finally, future work should study the effect of combining other tasks (such as segmentation) with our DTM estimation framework, as well as investigating the use of unsupervised learning for terrain modeling.

Chapter 6

Summary and Research plan

6.1 Conclusion

In this dissertation proposal, we investigated methods to apply deep learning to automate some of the most challenging and complex image interpretation tasks for remote sensing.

In Chapter 3, we proposed a a biologically inspired state-of-the-art aerial target tracking algorithm using deep features from a pretrained Convolutional Network. SPT was motivated by how the primate smooth pursuit system is thought to work, and the algorithm elegantly combines saliency maps for appearance, future location, and motion. We also showed how SPT can be extended easily to tracking multiple objects simultaneously with little computational overhead. We showed that the algorithm surpassed comparison methods from the literature on tracking moving vehicles in videos acquired by an aerial platform.

In Chapter 4 we presented a multi-scale deep learning framework to semantically label aerial 3D point clouds with spectral information. Our compact, 1D fully convolutional architecture directly consumes unstructured and unordered point clouds without relying on costly engineered features or 2D image transformations. We achieved near state-of-the-art results using only the 3D-coordinates and their corresponding spectral information. By design, our network can consume regions with varying densities, and is able to learn point-wise and block-wise features in an end-to-end fashion. Furthermore, our model is flexible, and can be readily extended to 2D semantic segmentation.

In Chapter 5 we evaluated a neighborhood-aware deep network for predicting a digital terrain model from 3D LiDAR and stereo photogrammetry point clouds. Our network bypasses the need for classifying ground points, and predicts the DTM value for each input point within the 3D point cloud without the need for intermediate steps. Building upon our previous work, our network operates on the input point clouds directly, therefore respecting their unstructured and unordered nature. Furthermore, it eliminates the need for calculating handcrafted features derived from the

DSM or fine tuning parameters that are scene specific.

Together, these efforts aim to advance fundamental and applied research in computer vision and deep learning for aerial and satellite imagery. Discussed methods help reducing burden on the human image analyst, broadening the scope and capacity of quantitative image processing to solve increasingly challenging and complex problems, and providing an opportunity for computing capability to meet the needs of the growing volume of digital data.

6.2 Main Contributions

Task 1: Aerial target tracking

- First to use deep networks to extract feature for aerial target tracking.
- Developed a tracking algorithm inspired by the smooth pursuit eye movement.
- Proposed the use of region proposals to handle occlusions.
- Extended the algorithm to handle online tracking of multiple targets.

Task 2: 3D point cloud semantic labeling

- Proposed a near state-of-the-art multi-scale 1D convolutional network to semantically label aerial point clouds.
- Proposed a fully convolutional deep learning framework that consume unordered and unstructured point clouds directly, mitigates the issue of non-uniform point density
- Extended the algorithm to handle multimodal fusion in 2D semanatic segmentation.

Task 3: Digital terrain modeling

- Proposed a novel framework to estimate the digital terrain model while bypassing the need for classifying ground points
- Proposed a deep network that combines point-wise, neighborhood, and global features to predict a point-wise DTM value.
- Developed a framework that eliminates the need for calculating handcrafted features or fine tuning parameters that are scene specific

6.3 Future work

In this Section, we present the limitations and future research paths for the tasks presented in Chapters 3, 4, and 5.

Task 1: Aerial target tracking In this task we presented a single object tracker that can be easily extended to track multiple targets with minimal overhead. However, we noticed the following limitations:

- Our Gnostic Fields object recognition doesn't perform a correlation check between different Gnostic sets. This limitation causes the classifier to include features from the background in the object set, thus reducing the robustness and increasing the chance of drift. Therefore, we recommend performing a similarity check between different sets to remove correlated features as well as projecting the remaining features into a space that maximizes the distance between sets.
- Our final bounding box detection is performed using a simple connected component analysis. This explains the rapid changes in bounding box size that is seen to occur during tracking. Additional work needs to occur to generate a more robust methodology that produces a more constant sized object bounding box.
- In general, a single object tracker, in contrast to multiple object tracking, is a short-term tracker that doesn't handle the interactions between objects, ID switching, and objects appearing/disappearing from the frame (to some extent). A possible research path forward borrows ideas from multiple object tracking to extend our method to handle the interactions between objects, thus increasing the robustness in "crowded" areas of the scene.

Task 2: 3D point cloud semantic labeling In this task we presented a multiscale network to classify individual points into different categories. However, we noticed the following limitations:

- Our results were sensitive the accuracy of the digital terrain model used to normalize the height value. A possible future research path would investigate the use and robustness of our DTM learning-based method, presented in Task 3, for semnatic labeling.
- The multiscale network does not include neighborhood information during the learning process. A suggested improvement would be to extend the network to handle neighborhood information by using the neighborhood branch from the network presented in Task 3.
- The training data used, the ISPRS data set, was very limited in the structures and terrain represented. To apply this method at scale, much more data is required. Here we suggest

a research path where simulated data, such as that generated using the Digital Imaging and Remote Sensing Image Generation (DIRSIG) simulation environment available at RIT, be used to generated labeled data with a wide variety of structure and terrain variations.

Task 3: Digital terrain modeling In this task we presented a neighborhood-aware network to estimate the underlying terrain model for a given aerial- or satellite-generated point cloud. The following limitations should be noted:

- This task investigated the estimation of the DTM only. The estimated DTM was not tested in other processing pipelines such as semantic segmentation. A possible future research path would investigate the effect of our solution presented in Task 3 when used with other tasks.
- In order to provide a DTM solution at scale, multiple questions need to be answered. For example, given *a priori* knowledge about the scene, is it possible to use that information with the network to increase its performance for that particular scene? This question can lead to 2 possible research paths. The first is adding such *a priori* knowledge as an auxiliary input, while the second is to use such information to guide the design of the network (*i.e.* modifying the hyperparameters).
- One of the major limitations in this task was the limited training data. The potential of using simulated data is unbounded and opens the door for many future research directions. For example, one can investigate the difference in performance when LiDAR or dense multi-view stereo data are used. Another path is to simulate unconventional/anomalous regions to increase the robustness of the algorithm when used at scale. A possible research path is to investigate the use of unsupervised DTM estimation combined with a large amount of simulated, high-quality, unlabeled data.

Deep learning for 3D point cloud data is still in its early stages, and we encourage the reader to explore other ideas as there are many more interesting future directions for research.

Bibliography

[1] Y. Ma, H. Wu, L. Wang, B. Huang, R. Ranjan, A. Zomaya, and W. Jie, "Remote sensing big data computing: Challenges and opportunities," *Future Generation Computer Systems*, vol. 51, pp. 47–60, 2015.

[2] R. Schingler, "Planet launches satellite constellation to image the whole planet daily." `https://goo.gl/5K7vDQ`, 2017.

[3] M. Alderton, "Riding the data wave." `http://trajectorymagazine.com/riding-data-wave/`, 2017.

[4] N. Skytland, "What is nasa doing with big data today?." `https://goo.gl/3qkTJS`, 2012.

[5] J.-G. Lee and M. Kang, "Geospatial big data: Challenges and opportunities," *Big Data Research*, vol. 2, no. 2, pp. 74 – 81, 2015. Visions on Big Data.

[6] P. Meier, *Digital Humanitarians: How Big Data Is Changing the Face of Humanitarian Response*. Boca Raton, FL, USA: CRC Press, Inc., 2015.

[7] J. Schott, *Remote Sensing: The Image Chain Approach*. Oxford Series on Optical and Imaging Sciences Series, Oxford University Press, 1997.

[8] R. N. Colwell, "History and place of photographic interpretation," *Manual of photographic interpretation*, vol. 2, pp. 33–48, 1997.

[9] A. society for photogrammetry and remote sensing, *Manual of remote sensing*. No. v. 2, Estes J., 1983.

[10] Y. Wu, J. Lim, and M.-H. Yang, "Online object tracking: A benchmark," in *Proc IEEE Computer Vision and Pattern Recognition*, 2013.

[11] A. Moussa and N. El-Sheimy, "Automatic classification and 3D modeling of lidar data," in *Proceedings of the ISPRS Commission III symposium*, vol. 38, (Saint-Mand, France), pp. 155–159, ISPRS, September 2010.

[12] A. Jochem, B. Höfle, M. Hollaus, and M. Rutzinger, "Object detection in airborne LIDAR data for improved solar radiation modeling in urban areas," in *International Archives of Photogrammetry, Remote Sensing, and Spatial Information Sciences*, vol. 38, Part 3/W8, (Paris, France), pp. 1–6, September 2009.

[13] F. Deng, Z. Zhang, and J. Zhang, "Construction 3d urban model from lidar and image sequence," 03 2019.

[14] M. P. Christiansen, M. S. Laursen, R. N. Jrgensen, S. Skovsen, and R. Gislum, "Designing and testing a uav mapping system for agricultural field surveying," *Sensors*, vol. 17, no. 12, 2017.

[15] B. Yang and I. Jahan, "Comprehensive assessment for post-disaster recovery process in a tourist town," *Sustainability*, vol. 10, no. 6, p. 1842, 2018.

[16] G. Kabite, "Lidar dem data for flood mapping and assessment; opportunities and challenges: A review," *Journal of Remote Sensing GIS*, vol. 06, 01 2017.

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[18] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[19] D. Reid, "An algorithm for tracking multiple targets," *IEEE transactions on Automatic Control*, vol. 24, no. 6, pp. 843–854, 1979.

[20] S. Blackman and R. Popoli, "Design and analysis of modern tracking systems (artech house radar library)," *Artech house*, 1999.

[21] C. Rashbass, "The relationship between saccadic and smooth tracking eye movements," *Journal of Physiology*, vol. 159, pp. 326–338, 1961.

[22] L. Itti and C. Koch, "Computational Modeling of Visual Attention," *Nature Reviews Neuroscience*, vol. 2, no. 3, pp. 194–203, 2001.

[23] D. Gao, V. Mahadevan, and N. Vasconcelos, "On the plausibility of the discriminant center-surround hypothesis for visual saliency," *Journal of Vision*, vol. 8, no. 7, pp. 1–18, 2008.

[24] L. Zhang, M. Tong, T. Marks, H. Shan, and G. Cottrell, "SUN: A Bayesian framework for saliency using natural statistics," *Journal of Vision*, vol. 8, pp. 1–20, 2008.

[25] H. Li, Y. Li, and F. Porikli, "Deeptrack: Learning discriminative feature representations by convolutional neural networks for visual tracking," in *Proceedings of the British Machine Vision Conference*, 2014.

[26] K. Ehinger, B. Hidalgo-Sotelo, A. Torralba, and A. Oliva, "Modeling search for people in 900 scenes," *Visual Cognition*, vol. 17, pp. 945–978, 2009.

[27] C. Kanan, M. Tong, L. Zhang, and G. Cottrell, "SUN: Top-down saliency using natural statistics," *Visual Cognition*, vol. 17, pp. 979–1003, 2009.

[28] A. Torralba, A. Oliva, M. Castelhano, and J. Henderson, "Contextual guidance of eye movements and attention in real-world scenes: The role of global features on object search," *Psychology Review*, vol. 113, no. 4, pp. 766–786, 2006.

[29] J. Yang and M. Yang, "Top-down visual saliency via joint CRF and dictionary learning," in *Proc IEEE Computer Vision and Pattern Recognition*, 2012.

[30] U. Ilg, "The role of areas mt and mst in coding of visual motion underlying the execution of smooth pursuit," *Vision Research*, vol. 48, pp. 2062–2069, 2008.

[31] M. Spering and A. Montagnini, "Do we track what we see? Common versus independent processing for motion perception and smooth pursuit eye movements: A review," *Vision Research*, vol. 51, pp. 836–852, 2011.

[32] R. Krauzlis, "Recasting the smooth pursuit eye movement system," *Journal of Neurophysiology*, vol. 91, no. 2, pp. 591–603, 2004.

[33] W. Newsome, R. Wurtz, and H. Komatsu, "Relation of cortical areas MT and MST to pursuit eye movements. II. Differentiation of retinal from extraretinal inputs," *Journal of Neurophysiology*, vol. 60, no. 2, pp. 604–620, 1988.

[34] M. Goldberg, J. Bisley, K. Powell, and J. Gottlieb, "Saccades, saliency and attention: the role of the lateral intraparietal area in visual behavior," *Progress in Brain Research*, vol. 155, pp. 157–175, 2006.

[35] L. Itti and C. Koch, "A saliency-based search mechanism for overt and covert shifts of visual attention," *Vision Research*, vol. 40, pp. 1489–1506, 2000.

[36] B. Thompson, "A visual salience map in the primate frontal eye field," *Progress in brain research*, vol. 147, pp. 249–262, 2005.

[37] G. Stanton, C. Bruce, and M. Goldberg, "Topography of projections to posterior cortical areas from the macaque frontal eye fields," *Journal of Comparative Neurology*, vol. 353, pp. 291–305, 1995.

[38] J. Schall, A. Morel, D. King, and J. Bullier, "Topography of visual cortex connections with frontal eye field in macaque: convergence and segregation of processing streams," *Journal of Neuroscience*, vol. 15, pp. 4464–4487, 1995.

[39] J. DiCarlo, D. Zoccolan, and N. Rust, "How does the brain solve visual object recognition?," *Neuron*, vol. 73, pp. 415–434, 2012.

[40] A. Yilmaz, O. Javed, and M. Shah, "Object tracking: A survey," *ACM Computing Surveys*, vol. 38, no. 4, pp. 1–45, 2006.

[41] X. Mei and H. Ling, "Robust visual tracking and vehicle classification via sparse representation," *IEEE Transactions on Pattern Recognition and Machine Intelligence*, vol. 33, pp. 2259–2272, 2011.

[42] X. Jia, H. Lu, and M. Yang, "Visual tracking via adaptive structural local sparse appearance model," in *Proc IEEE Computer Vision and Pattern Recognition*, 2012.

[43] B. Babenko, M.-H. Yang, and S. Belongie, "Visual tracking with online multiple instance learning," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[44] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista, "Exploiting the circulant structure of tracking-by-detection with kernels," in *proceedings of the European Conference on Computer Vision*, 2012.

[45] M. Danelljan, F. Khan, M. Felsberg, and J. van de Weijer, "Adaptive color attributes for real-time visual tracking," in *Proc IEEE Computer Vision and Pattern Recognition*, 2014.

[46] J. Van De Weijer, C. Schmid, J. Verbeek, and D. Larlus, "Learning Color Names for Real-World Applications," *IEEE Transactions on Image Processing*, vol. 18, pp. 1512–1523, July 2009.

[47] H. Grabner, M. Grabner, and H. Bischof, "Real-time tracking via on-line boosting," in *Proc. BMVC*, 2006.

[48] L. Zhang and L. van der Maaten, "Structure preserving object tracking," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2013.

[49] N. Wang and D.-Y. Yeung, "Learning a deep compact image representation for visual tracking," in *Advances in Neural Information Processing Systems*, vol. 26, pp. 809–817, 2013.

[50] S. Hong, T. You, S. Kwak, and B. Han, "Online tracking by learning discriminative saliency map with convolutional neural network," in *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, 2015.

[51] V. Mahadevan and N. Vasconcelos, "Saliency-based discriminant tracking," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

[52] G. Zhang, Z. Yuan, N. Zheng, X. Sheng, and T. Liu, "Visual saliency based object tracking," in *Computer Vision ACCV 2009*, 2010.

[53] J. Fan, W. Xu, Y. Wu, and Y. Gong, "Human tracking using convolutional neural networks," *IEEE Transactions on Neural Networks*, vol. 21, pp. 1610–1623, 2010.

[54] T. Hackel, J. D. Wegner, and K. Schindler, "Fast semantic segmentation of 3D point clouds with strongly varying density.," in *ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences*, vol. III-3, (Prague, Czech Republic), pp. 177–184, 2016.

[55] C. Hug and A. Wehr, "Detecting and identifying topographic objects in imaging laser altimeter data," in *ISPRS International Archives of Photogrammetry and Remote Sensing*, vol. 32, Part 3–4W2, pp. 19–26, 1997.

[56] N. Haala, C. Brenner, and K.-H. Anders, "3D urban GIS from laser altimeter and 2D map data," in *ISPRS International Archives of Photogrammetry, Remote Sensing & Spatial Information Sciences*, vol. 32, pp. 339–346, 1998.

[57] S. Xing, P. Li, Q. Xu, D. Wang, and P. Li, "Surface Fitting Filtering of LIDAR Point Cloud with Waveform Information," in *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. IV-2, pp. 179–184, Sept. 2017.

[58] B. Yunfei, L. Guoping, C. Chunxiang, Z. Hao, H. Qisheng, B. Linyan, and C. Chaoyi, "Classification of lidar point cloud and generation of DTM from lidar height and intensity data in forested area," in *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XXVII-7, pp. 313–318, 2008.

[59] C.-H. Lin, J.-Y. Chen, P.-L. Su, and C.-H. Chen, "Eigen-feature analysis of weighted covariance matrices for LiDAR point cloud classification," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 94, pp. 70–79, 2014.

[60] R. Blomley, B. Jutzi, and M. Weinmann, "3D semantic labeling of ALS point clouds by exploiting multi-scale, multi-type neighborhoods for feature extraction," in *Proceedings of the International Conference on Geographic Object-Based Image Analysis (GEOBIA)*, (Enschede, The Netherlands), pp. 1–8, September 2016.

[61] M. Weinmann, B. Jutzi, and C. Mallet, "Semantic 3D scene interpretation: A framework combining optimal neighborhood size selection with relevant features," in *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. II-3, pp. 181–188, Aug. 2014.

[62] A. Ramiya, R. R. Nidamanuri, and R. Krishnan, "Semantic labelling of urban point cloud data," in *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XL-8, (Hyderbad, India), pp. 907–911, December 2014.

[63] P. Axelsson, "DEM generation from laser scanner data using adaptive TIN models," in *ISPRS International Archives of Photogrammetry and Remote Sensing*, vol. XXXIII-Part B4/1, pp. 111–118, 2000.

[64] C. Mallet, *Analysis of full-waveform lidar data for urban area mapping*. PhD thesis, Télécom ParisTech, 2010.

[65] N. Chehata, L. Guo, and C. Mallet, "Airborne lidar feature selection for urban classification using random forests," in *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XXXVIII-Part 3, 2009.

[66] E. Grilli, F. Menna, and F. Remondino, "A review of point clouds segmentation and classification algorithms," in *ISPRS International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XLII-2/W3, (Nafplio, Greece), pp. 339–344, 2017.

[67] J. Niemeyer, F. Rottensteiner, and U. Soergel, "Contextual classification of lidar data and building object detection in urban areas," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 87, pp. 152 – 165, 2014.

[68] J. Niemeyer, F. Rottensteiner, U. Soergel, and C. Heipke, "Hierarchical Higher Order CRF for the Classification of Airborne LIDAR Point Clouds in Urban Areas," in *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XLI-B3, (Czech Republic), pp. 655–662, July 2016.

[69] A. Golovinskiy, V. G. Kim, and T. Funkhouser, "Shape-based recognition of 3D point clouds in urban environments," in *Proceedings of the 12th International Conference on Computer Vision (ICCV)*, pp. 2154–2161, September 2009.

[70] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson, "Understanding neural networks through deep visualization," *arXiv preprint arXiv:1506.06579*, 2015.

[71] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, pp. 541–551, 1989.

[72] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[73] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009.

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

[75] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 580–587, 2014.

[76] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, pp. 2481–2495, Dec 2017.

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

[78] M. A. Yousefhussien, N. A. Browning, and C. Kanan, "Online tracking using saliency," in *Proceedings of the IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 1–10, IEEE, 2016.

[79] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller, "Multi-view convolutional neural networks for 3D shape recognition," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 945–953, 2015.

[80] S. Bai, X. Bai, Z. Zhou, Z. Zhang, and L. Jan Latecki, "GIFT: A real-time and scalable 3D shape search engine," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5023–5032, 2016.

[81] M. Savva, F. Yu, H. Su, M. Aono, B. Chen, D. Cohen-Or, W. Deng, H. Su, S. Bai, X. Bai, *et al.*, "SHREC16 track: large-scale 3d shape retrieval from ShapeNet core55," in *Proceedings of the Eurographics Workshop on 3D Object Retrieval*, pp. 89–98, 2016.

[82] A. Boulch, B. L. Saux, and N. Audebert, "Unstructured Point Cloud Semantic Labeling Using Deep Segmentation Networks," in *Proceedings of the Eurographics Workshop on 3D Object Retrieval*, vol. 2, pp. 17–24, April 2017.

[83] T. Hackel, N. Savinov, L. Ladicky, J. Wegner, K. Schindler, and M. Pollefeys, "Semantic3d.net: A new large-scale point cloud classification benchmark," *arXiv preprint arXiv:1704.03847*, 2017.

[84] N. Audebert, B. L. Saux, and S. Lefèvre, "Semantic segmentation of earth observation data using multimodal and multi-scale deep networks," in *Proceedings of the Asian Conference on Computer Vision (ACCV)*, (Taipei, Taiwan), 21–23 November 2016.

[85] L. Caltagirone, S. Scheidegger, L. Svensson, and M. Wahde, "Fast LIDAR-based road detection using fully convolutional neural networks," in *Proceedings of the IEEE Intelligent Vehicles Symposium (IV)*, pp. 1019–1024, June 2017.

[86] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The KITTI dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.

[87] B. Li, "3D fully convolutional network for vehicle detection in point cloud," in *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, 2017.

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

[89] D. Maturana and S. Scherer, "VoxDet: A 3D convolutional neural network for real-time object recognition," in *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, pp. 922–928, IEEE, 2015.

[90] C. R. Qi, H. Su, M. Nießner, A. Dai, M. Yan, and L. J. Guibas, "Volumetric and multi-view CNNs for object classification on 3D data," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5648–5656, 2016.

[91] M. Lin, Q. Chen, and S. Yan, "Network in network," in *International Conference on Learning Representations (ICLR)*, 2014.

[92] Y. Liu, S. Piramanayagam, S. T. Monteiro, and E. Saber, "Dense semantic labeling of very-high-resolution aerial imagery and lidar with fully-convolutional neural networks and higher-order crfs," in *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 1561–1570, July 2017.

[93] X. Meng, C. Nate, and K. Zhao, "Ground filtering algorithms for airborne lidar data: A review of critical issues," *Remote Sensing*, vol. 2, 03 2010.

[94] Z.-Y. Chen, B. Gao, and B. Devereux, "State-of-the-art: Dtm generation using airborne lidar data," in *Sensors*, 2017.

[95] Y. Zhang and L. Men, "Study of the airborne lidar data filtering methods," in *2010 18th International Conference on Geoinformatics*, pp. 1–5, June 2010.

[96] G. Sithole and G. Vosselman, "Comparison of filtering algorithms," *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 34, 01 2003.

[97] G. Vosselman, "Slope based filtering of laser altimetry data," *IAPRS*, vol. XXXIII, 01 2000.

[98] R. Wack and A. Wimmer, "Digital terrain models from airborne laser scanner data - a grid based approach," *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 34, 01 2002.

[99] R. Ma, "Dem generation and building detection from lidar data," *Photogrammetric Engineering Remote Sensing*, vol. 71, pp. 847–854, 07 2005.

[100] J. Susaki, "Adaptive slope filtering of airborne lidar data in urban areas for digital terrain model (dtm) generation," *Remote Sensing*, vol. 4, pp. 1804–1819, 2012.

[101] M. Debella-Gilo, "Bare-earth extraction and dtm generation from photogrammetric point clouds including the use of an existing lower-resolution dtm," *International Journal of Remote Sensing*, vol. 37, no. 13, pp. 3104–3124, 2016.

[102] M. Isenburg, "Lastools - efficient lidar processing software, version 190114 (licensed)." `http://rapidlasso.com/LAStools`, 2018.

[103] Exelis, "Exelis visual information solutions: Envi-lidar tool, version 5.3." `https://www.harrisgeospatial.com`, 2018.

[104] Y. Li, B. Yong, H. Wu, R. An, H. Xu, J. Xu, and Q. He, "Filtering airborne lidar data by modified white top-hat transform with directional edge constraints," *Photogrammetric Engineering Remote Sensing*, vol. 80, pp. 133–141, 02 2014.

[105] Y. Li, B. Yong, H. Wu, R. An, and H. Xu, "An improved top-hat filter with sloped brim for extracting ground points from airborne lidar point clouds," *Remote Sensing*, vol. 6, pp. 12885–12908, 12 2014.

[106] H. Arefi, P. dAngelo, H. Mayer, and P. Reinartz, "Automatic generation of digital terrain models from cartosat-1 stereo images," 2009.

[107] K. Kraus and N. Pfeifer, "Advanced dtm generation from lidar data," *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 34, pp. 23–30, 01 2001.

[108] X. Hu and Y. Yuan, "Deep-learning-based classification for dtm extraction from als point cloud," *Remote Sensing*, vol. 8, no. 9, 2016.

[109] C. Gevaert, C. Persello, F. Nex, and G. Vosselman, "A deep learning approach to dtm extraction from imagery using rule-based training labels," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 142, 06 2018.

[110] D. Marmanis, A. Fathalrahman, M. Datcu, T. Esch, and U. Stilla, "Deep neural networks for above-ground detection in very high spatial resolution digital elevation models," 01 2015.

[111] Y. Luo, H. Ma, and L. Zhou, "Dem retrieval from airborne lidar point clouds in mountain areas via deep neural networks," *IEEE Geoscience and Remote Sensing Letters*, vol. 14, pp. 1770–1774, Oct 2017.

[112] C. Kanan, "Recognizing sights, smells, and sounds with gnostic fields," *PLOS ONE*, vol. e54088, 2013.

[113] C. Kanan, "Fine-grained object recognition with gnostic fields," in *Proceedings of the IEEE Winter Applications of Computer Vision Conference*, 2014.

[114] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *ICLR*, 2015.

[115] A. Vedaldi and K. Lenc, "MatConvNet – convolutional neural networks for matlab," *CoRR*, vol. abs/1412.4564, 2014.

[116] V. Mahadevan and N. Vasconcelos, "Spatiotemporal saliency in dynamic scenes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, pp. 171–177, 2010.

[117] G. Evangelidis, "IAT: A matlab toolbox for image alignment," 2013.

[118] J.-M. Geusebroek, R. Van den Boomgaard, A. W. Smeulders, and H. Geerts, "Color invariance," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 23, no. 12, pp. 1338–1350, 2001.

[119] J. Uijlings, K. van de Sande, T. Gevers, and A. Smeulders, "Selective search for object recognition," *International Journal of Computer Vision*, vol. 104, pp. 154–171, 2013.

[120] "VIVID tracking evaluation web site. url: http://vision.cse.psu.edu/data/vivideval/datasets/datasets.html."

[121] C. Vondrick, D. Patterson, and D. Ramanan, "Efficiently scaling up crowdsourced video annotation," *International Journal of Computer Vision*, pp. 1–21. 10.1007/s11263-012-0564-1.

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

[123] M. S. Ryoo, B. Rothrock, and L. H. Matthies, "Pooled motion features for first-person videos.," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 896–904, 2015.

[124] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu, "Spatial transformer networks," in *Advances in Neural Information Processing Systems 28*, pp. 2017–2025, 2015.

[125] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, vol. 9, pp. 249–256, 2010.

[126] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 1026–1034, December 2015.

[127] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on Machine Learning*, vol. 37, pp. 448–456, 2015.

[128] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations*, 2015.

[129] F. Chollet *et al.*, "Keras." `https://github.com/fchollet/keras`, 2015.

[130] Z. Yang, W. Jiang, B. Xu, Q. Zhu, S. Jiang, and W. Huang, "A convolutional neural network-based 3D semantic labeling method for ALS point clouds," *Remote Sensing*, vol. 9, no. 936, 2017.

[131] Y. Boykov, O. Veksler, and R. Zabih, "Fast approximate energy minimization via graph cuts," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 23, pp. 1222–1239, Nov. 2001.

[132] G. Tapper, *Extraction of DTM from Satellite Images Using Neural Networks*. PhD thesis, Linkoping University, 2016.

[133] M. Yousefhussien, D. J. Kelbe, E. J. Ientilucci, and C. Salvaggio, "A multi-scale fully convolutional network for semantic labeling of 3 d point clouds," 2018.

[134] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," in *Advances in Neural Information Processing Systems 30*, pp. 5099–5108, Curran Associates, Inc., 2017.

[135] Y. Shen, C. Feng, Y. Yang, and D. Tian, "Mining point cloud local structures by kernel correlation and graph pooling," pp. 4548–4557, 06 2018.

[136] J. Rouse Jr, R. Haas, J. Schell, and D. Deering, "Monitoring vegetation systems in the great plains with erts," 1974.

[137] DigitalGlobe, "See the world in 3D." `https://explore.digitalglobe.com/see-the-world-in-3D-samples.html?aliId=735396`, 2016. Accessed: 2019-04-04.

[138] F. Isaksson, I. Andersson, J. Bejeryd, J. Borg, P. Carlbom, and L. Haglund, "Method and arrangement for developing a three dimensional model of an environment," 2015.

[139] L. Haglund, O. Nygren, F. Isaksson, and J. Borg, "Method and a system for building a three-dimensional model from satellite images," 2017.

[140] N. T. Anderson and G. B. Marchisio, "Worldview-2 and the evolution of the digitalglobe remote sensing satellite constellation: introductory paper for the special session on worldview-2," in *Algorithms and Technologies for Multispectral, Hyperspectral, and Ultraspectral Imagery XVIII*, vol. 8390, p. 83900L, International Society for Optics and Photonics, 2012.

Appendices

Appendix A

List of publications

We have shared our finding at the following venues.

1. **M. Yousefhussien**, D. Kelbe, and C. Salvaggio. "A Nearest Neighbor Network to Extract Digital Terrain Models from 3D Point Clouds" ISPRS Journal of Photogrammetry and Remote Sensing, 2019. (Submitted)

2. **M. Yousefhussien**, D. Kelbe, E. Ientilucci, and C. Salvaggio. "A multi-scale fully convolutional network for semantic labeling of 3D point clouds." ISPRS Journal of Photogrammetry and Remote Sensing, 2017.

3. **M. Yousefhussien**, N.A. Browning, and C. Kanan. "Online tracking using saliency." Applications of Computer Vision (WACV), IEEE, 2016.

In addition to the original research contributions made towards this dissertation, we have also published the following related work in imaging science and computer vision as part of the PhD program.

1. K. Kafle, **M. Yousefhussien**, and C. Kanan. "Data augmentation for visual question answering." In International Conference on Natural Language Generation (INLG), 2017

2. A. Wong, **M. Yousefhussien**, and R. Ptucha, "Localization using omnivision based manifold particle filters." In Intelligent Robots and Computer Vision XXXII: Algorithms and Techniques, 2015.

3. **M. Yousefhussien**, RL. Easton, R. Ptucha, M. Shaw, B. Bradburn, J. Wagner, D. Larson, E. Saber. "Flatbed scanner simulation to analyze the effect of detector's size on color artifacts." In Computational Imaging XIII, 2015.