SEMANTIC LABELING OF LARGE GEOGRAPHIC AREAS USING MULTI-DATE AND MULTI-VIEW SATELLITE IMAGES AND NOISY OPENSTREETMAP LABELS

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Bharath Kumar Comandur Jagannathan Raghunathan

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

August 2020

Purdue University

West Lafayette, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF DISSERTATION APPROVAL

Dr. Avinash C. Kak, Chair

School of Electrical and Computer Engineering

Dr. Charles A. Bouman

School of Electrical and Computer Engineering

Dr. Sanjay G. Rao

School of Electrical and Computer Engineering

Dr. Tanmay Prakash

Google Inc., Boston

Approved by:

Dr. Dimitri Peroulis

Head of the School of Electrical and Computer Engineering

To the women in my life

ACKNOWLEDGMENTS

"While it is always best to believe in oneself, a little help from others can be a great blessing."

- Iroh

Firstly, I wish to express my sincere gratitude to Dr. Avinash C. Kak for his immense understanding, guidance and encouragement, without which this research would not exist. He has time and again, gone above and beyond the typical role of an academic advisor and has helped me tremendously at both a professional and personal level. I am really fortunate to have had such an incredible mentor and friend throughout my years at Purdue.

Thanks must also be expressed to Dr. Tanmay Prakash and Dr. Tommy Chang for their invaluable help and patience. Whatever meager knowledge that I now possess about Linux, cloud computing and open-source software has been gleaned by shadowing Dr. Tommy during his time at RVL. I am incredibly glad that I got the opportunity to work with such a wonderful researcher as Dr. Tanmay, and I will forever cherish both our research and non-research conversations.

My confusion about the order in which I should thank my friends is surpassed only by my perplexity at how I lucked out with every single one of them. So, in no particular order, I would like to thank Saranya and Vasuki for looking after me when I needed it the most, Purnima for her incredible patience and unwavering support, Prasad for giving me some of my best memories at Purdue, and Swetha for always keeping her home and kitchen open for me. I do not believe that I could have earned my PhD without all of your kindness and support. I should also express my heartfelt gratitude to Kaushik, Aditya and Adharsh for adding a lot of fun to my life outside research. It really helped keep things in perspective. I wish to thank Vignesh for his virtual support and for always believing in me more that myself. To Aravind and

Vinod, I would like to save my thanks for tomorrow and then tomorrow, I can tell them what great friends they turned out to be...

To my grandparents, Jayalakshmi and Lakshminarasimhan, I really wish you two were around to see me get a PhD. I hope I have made you proud. I wish to express my sheer admiration of my nephews, Vishnu and Gautham, for constantly reminding me that "artificial intelligence" and "real intelligence" are very different things. I would like to thank my dad Raghunathan for all the sacrifices that he has made to give me such a privileged life, and for always dreaming big for his two children. Which leads me to my sister Madhumathi. She has always been my rock and I would not be here if not for her unflinching support. There are no words that can adequately express my gratitude and affection towards her. And finally, I can't thank my mom, Vaidehi, enough, for everything. She is the toughest and the most courageous person that I have ever known, and her zest to never stop learning has been one of the biggest inspirations in my life.

TABLE OF CONTENTS

Page

Page

Page

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

Figure Page

Figure Page

LIST OF ALGORITHMS

ABSTRACT

Comandur, Bharath Kumar PhD, Purdue University, August 2020. Semantic Labeling of Large Geographic Areas Using Multi-Date and Multi-View Satellite Images and Noisy OpenStreetMap Labels. Major Professor: Dr. Avinash C. Kak.

This dissertation addresses the problem of how to design a convolutional neural network (CNN) for giving semantic labels to the points on the ground given the satellite image coverage over the area and, for the ground truth, given the noisy labels in OpenStreetMap (OSM). This problem is made challenging by the fact that – (1) Most of the images are likely to have been recorded from off-nadir viewpoints for the area of interest on the ground; (2) The user-supplied labels in OSM are frequently inaccurate and, not uncommonly, entirely missing; and (3) The size of the area covered on the ground must be large enough to possess any engineering utility. As this dissertation demonstrates, solving this problem requires that we first construct a DSM (Digital Surface Model) from a stereo fusion of the available images, and subsequently use the DSM to map the individual pixels in the satellite images to points on the ground. That creates an association between the pixels in the images and the noisy labels in OSM. The CNN-based solution we present yields a 4-7% improvement in the per-class segmentation IoU (Intersection over Union) scores compared to the traditional approaches that use the views independently of one another. The system we present is end-to-end automated, which facilitates comparing the classifiers trained directly on true orthophotos vis-à-vis first training them on the off-nadir images and subsequently translating the predicted labels to geographical coordinates. This work also presents, for arguably the first time, an in-depth discussion of large-area image alignment and DSM construction using tens of true multi-date and multi-view WorldView-3 satellite images on a distributed OpenStack cloud computing platform.

1. INTRODUCTION

The next decade is set to witness a deluge of remotely-sensed data collected by numerous satellites with the objective of continuous 24/7 monitoring of the globe. There are many advantages to using satellites over drones – large areas can be imaged quickly, the stability of their orbits is not affected by wind and local weather conditions, there are far fewer restrictions that can be imposed on data collection, etc.

Precise understanding and interpretation of these vast amounts of data is of profound importance to a myriad of applications including agriculture, disaster relief, autonomous driving and many commercial and military ventures. By "understanding and interpretation", we mean extracting semantic and higher-level information from the images such as the location of objects like roads, buildings and transmission towers, changes in forest-cover, estimates of crop yield, man-made and natural changes in the terrain, etc. In this work, we will focus on the specific task of semantic segmentation of roads and buildings. However the principles described here can be applied to the detection of many other objects as well.

We start out with a collection of overlapping satellite images covering a large area ($\sim$100 km^2). These images are captured at different angles (multi-view) and at different dates (multi-date). Some of them are captured even years apart. We also have access to some metadata associated with each image such as the camera model, view angle, time of acquisition, etc. Given this bare-minimum data, we want to semantically label the points on the ground that are covered by these images. Note that labeling points on the ground is more challenging than labeling pixels in images because the former requires that we first map each point on the ground to the correct pixel in each image. This is only possible if – (1) the multi-date and multi-view images are not only aligned with one another but are also aligned well in an absolute

sense to the real world; and (2) if we have accurate knowledge of the heights of the points on the ground.

State-of-the-art approaches for semantic segmentation use deep learning and convolutional neural networks (CNNs), and such approaches have been adopted by the remote-sensing community with much gusto. Semantic-labeling CNNs need a lot of training data in the form of images and precise annotations that provide the label for each pixel of interest. While creating such precise labels is an onerous task for traditional photographs, it is made that much harder for satellite images owing to their lower resolution and complex scene content. Nonetheless, the past few years have seen a push to make large satellite-image datasets publicly available for training deep CNNs. A few examples of such datasets are described by the studies in [1], [2], [3] and [4]. Such datasets contain precise annotations of labels which have been collected by professionals with proprietary software. A lot of expensive manual labor goes into creating these annotations.

One of the key objectives of this research is to minimize this manual labor. While active-learning approaches such as the one proposed by the study described in [5] are good candidates to do this, *we want to go a step further and minimize the effective human supervision of the system to zero.* As a step towards achieving this, we rely upon the publicly available OpenStreetMap (OSM) [6] data which consists of annotations collected by volunteers around the globe. True to the adage that there is no free lunch, the price we pay for using OSM is noise. OSM labels can be sparse in residential and rural areas, the annotations can be incorrect, our satellite images and the images in which the OSM labels were annotated can have significant mismatch due to time differences, crucial information such as the width of the roads is often not present in the OSM – these are just a few examples of the noise in OSM. **Therefore our objective of minimizing the amount of manual labor can be transformed into one of learning from noisy groundtruth.**

The human effort in creating annotations notwithstanding, the aforementioned datasets [1], [2], [3], [4] also consist of preprocessed and derived products such as

orthorectified images [7] and sources of elevation information such as Digital Surface Models (DSMs) [8] or LiDAR [9]. Orthorectification means that we map the pixel values in the images to their corresponding ground-based points in the geographic area of interest, which requires knowledge of the elevation of the points. Such datasets thus impose several constraints on the user. For example, how can one get precise training labels for new regions of the world that can differ greatly in image content vis-à-vis the regions on which the CNNs have been trained? For another example, airborne LiDAR is not readily available across the globe. Even in instances where LiDAR is available, it can be out of date in relation to the satellite images resulting in a mismatch of information between the two sources. For yet another example, what happens if aligned and orthorectified images are not available?

Therefore the second important objective of this research is to **study the feasibility of creating a large-area semantic-labeling framework that relies only on the non-orthorectified multi-view and multi-date satellite images and publicly available OSM data as input.** In the process of designing such a framework, we will have to address a number of related practical issues such as alignment of multi-modal data sources, stereo reconstruction and data fusion. Using our framework, we propose to construct DSMs across large regions (covering a 100 km^2), use these DSMs to derive labels from OSM and subsequently train deep neural networks. It is highly likely that such a framework would be easily overwhelmed by the noise coming from the different components of the framework unless suitable design choices are made.

Therefore, there are two parts to this dissertation. The first part concerns itself with the automated creation of large-scale datasets in a form that can be easily ingested by powerful machine-learning frameworks like CNNs. The second part focuses on developing deep-learning algorithms that can learn from multiple views of the same scene. Both parts are crucial and need to mesh together such that their whole is greater than the sum of their parts.

1.1 Primary Contributions

Towards addressing the aforementioned objectives, we put forth the following contributions:

1. We present a novel multi-view training paradigm, CNN architecture, and loss functions that yield improvements in the range *4-7% in the per-class IoU* (Intersection over Union) metric. *Our evaluation directly demonstrates that updating the weights of the CNN by simultaneously learning from multiple views of the same scene can help alleviate the burden of noisy training labels.* At inference time, information from multiple views is aggregated directly on the GPU resulting in faster and accurate labeling of large areas. Moreover, the proposed framework only adds a small overhead to the memory consumption even when learning from *as many as 32 views for a single scene.* This is in contrast to most of the relevant published literature on CNN-based semantic segmentation of aerial and satellite images which use pre-orthorectified single-view images. Even those studies that use multiple views treat the different views independently during the training phase.

2. We present a direct comparison between training classifiers on 8-band orthorectified images vis-à-vis training them on the original off-nadir images captured by the satellites. The fact that we use OSM training labels poses challenges for the latter approach, as it necessitates the need to transform the labels from geographic coordinates into the off-nadir image-pixel coordinates. Such a transformation requires that we have knowledge of the heights of the points. The comparison presented in this study is unlike most published work in the literature that use pre-orthorectified single-view images.

3. In order to make the above comparison possible, we present a true end-to-end automated framework that aligns large multi-view, multi-date images (each containing about 43008×38000 pixels), constructs a high-resolution accurate

Digital Surface Model (DSM) over a 100 km^2 area (which is needed for establishing correspondences between the pixels in the off-nadir images and the points on the ground), and learns from noisy OSM labels **without any additional human supervision**. We call our approach "true end-to-end automated" to distinguish it from previous state-of-the-art methods which either use pre-orthorectified single-view images that already have high geolocation accuracy, and/or use LiDAR or curated DSMs, and/or use precise training labels that have been supplied or corrected by human annotators.

4. As an added bonus, our framework can be used to gain useful insights into how errors in the sensor model, alignment, stereo matching, and OSM all contribute to errors in the labels. We use our framework to determine what performance can be expected given such a degree of automation. Moreover, our modular framework makes it possible to isolate a specific component and analyze its final impact on the performance of the labeler.

5. We present some insights into designing such a framework so that it can run on an OpenStack-based [10] cloud computing environment. These insights will not only talk about architectures for task-distribution but also provide some useful tips to improve the OpenStack environment itself for massively-parallel remote-sensing applications such as stereo matching.

1.2 Organization of this Dissertation

This dissertation is organized as follows. Chapter 2 discusses important and popular remote-sensing concepts and terminology. Familiarizing oneself with these terms will make it easier to understand the subsequent chapters. Chapter 3 addresses the issue of image-to-image alignment and creation of DSMs for a small area of interest (AOI) covering < 3 km^2. Subsequently, Chapter 4 talks about the issues that need to be resolved while constructing DSMs for a 100 km^2 area. The first half of Chapter 5 then describes how to use these DSMs and images to prepare training data for the

CNNs. Starting from the second half of Chapter 5, the focus shifts to deep learning where we discuss a baseline CNN architecture for semantic segmentation of 8-band satellite images. Chapter 6 focuses on a multi-view training and inference framework for multi-date satellite images and Chapter 7 puts all the various pieces together to describe the overall framework. Chapter 8 is a bonus chapter that provides some useful nuggets of wisdom on designing an OpenStack-based cloud computing framework for semantic labeling of large-areas. This chapter will be useful to university-based labs and small organizations that are interested in conducting large-area remote-sensing research. Extensive quantitative and qualitative evaluation, and ablation studies are presented in Chapter 9. Concluding remarks and possible future directions are discussed in Chapter 10.

2. REMOTE SENSING: CONCEPTS AND TERMINOLOGY

The purpose of this chapter is to act as a primer that the reader can use to become familiar with the different terms and concepts that are domain-specific to remote sensing and satellite imagery. The organization of this chapter is as follows. We first describe the different kinds of satellite imagery that are used in this research and their associated metadata, and then introduce two important concepts called top-of-atmosphere (ToA) correction and pansharpening. Subsequently, we illustrate the differences between three different sources of elevation information, namely Digital Elevation Models (DEMs) [11], Digital Surface Models (DSMs) [8] and point clouds captured by Light Detection and Ranging (LiDAR) [9]. We then explain how to use this elevation information in conjunction with the satellite camera model to map each pixel in each image to a point on the ground, through the process of orthorectification. After this, we present details of another important publicly-available crowdsourced database called OpenStreetMap (OSM) which acts as the source of noisy labels to the proposed deep-learning framework. Finally we conclude this chapter with a small description of important and useful software for viewing and processing remotely-sensed data.

2.1 Satellite Imagery

While there exist different kinds of satellites with different imaging sensors, many high-altitude satellites use pushbroom sensors to capture images. In such sensors, the detectors are aligned in multiple lines that are perpendicular to the direction of motion. The image is recorded one scanline at a time. An illustration of a pushbroom sensor is shown in Figure 2.1. For the rest of the discussion in this manuscript, we

will use satellite images that have been collected by the WorldView-3 (WV3) [12] pushbroom sensor. Since the WV3 satellite orbits the Earth at an altitude of 617 km above the Earth's surface, the pushbroom camera rays are almost parallel to each other. This also means that the camera model can be approximated by an affine camera over small areas of interest (AOIs) with reasonable accuracy.

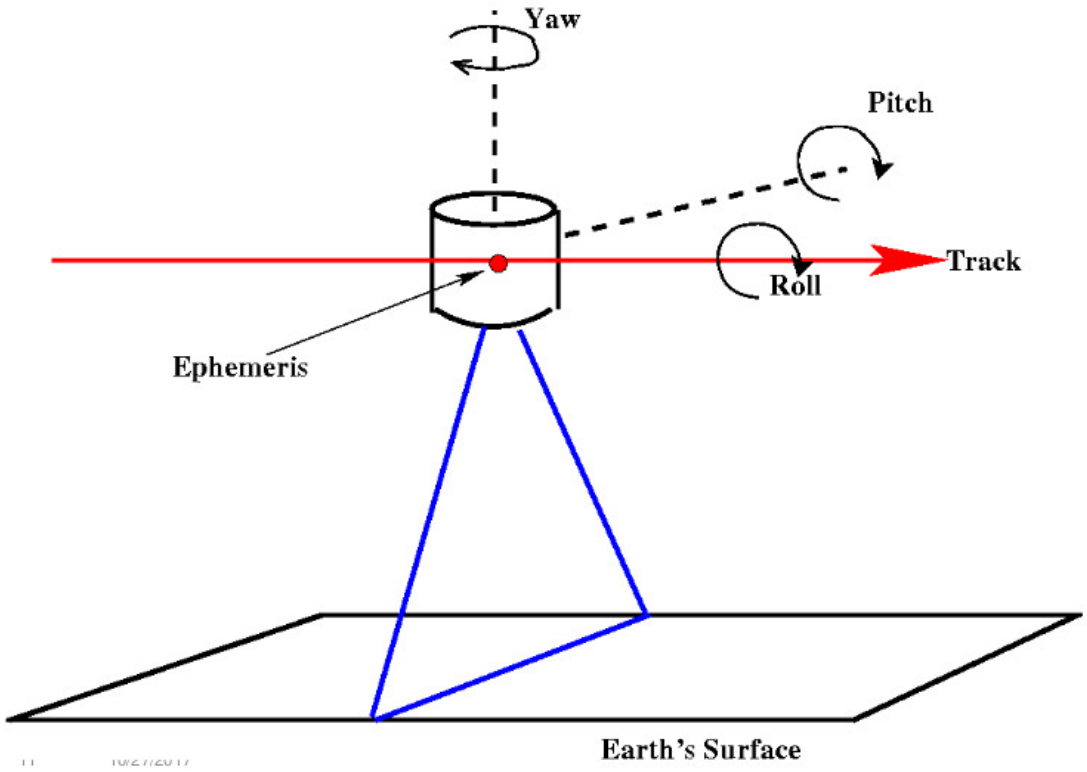


Fig. 2.1.: Illustration of a pushbroom sensor

2.1.1 WorldView-3 Panchromatic and Multispectral Images

The WV3 satellite was launched in 2014 by DigitalGlobe [13] and travels in a heliosynchronous (or sun-synchronous) orbit meaning that it images the same region on the Earth's surface around the same local mean solar time every day. It captures panchromatic (PAN) images with a nadir ground sampling distance (GSD) of 0.31 m per pixel, eight-band multispectral (MS) images with a nadir GSD of 1.24 m per

pixel, and short-wave infrared (SWIR) images with a nadir GSD of 3.7 m per pixel. In addition, it also captures clouds, aerosols, water vapor, ice and snow (CAVIS) data at a nadir GSD of 30 m per pixel. The PAN and MS images are captured simultaneously by different arrays of sensors. Note that we have specified the "nadir GSD" of the images. This means that if the satellite captures a PAN image by looking straight down at the Earth's surface, then the GSD of that image will be 0.31 m per pixel. In practice, the satellite images are often captured at off-nadir angles and therefore the true GSD of each PAN image is larger than 0.31 m per pixel. This also implies that different satellite images could have different GSDs. The corresponding statements hold for the MS and SWIR images as well. In this research, we only use the PAN and MS images and the differences between them are explained below.

Panchromatic Images - In panchromatic (PAN) images, the measurement for each pixel corresponds to the total intensity of radiation recorded at the geolocation or world point corresponding to that pixel. Panchromatic images are usually of higher spatial resolution than MS images.

Multispectral Images - Multispectral (MS) images as the name indicates, contain multispectral information. For each pixel or corresponding geolocation, they contain measurements across different bands (frequencies) of the electromagnetic (EM) spectrum. The WV3 MS images contain 8 spectrum bands per pixel. These 8 bands correspond to the Coastal, Blue, Green, Yellow, Red, Red Edge, Near-Infrared1 and Near-Infrared2 portions of the EM spectrum.

Fig. 2.2 shows a portion of a PAN and a MS image captured by WV3 over a region in Ohio, USA. These are images captured at off-nadir angles as is evident from the fact that the walls of the buildings are visible in the images. Note that we have only displayed the Red, Green and Blue bands of the MS image.

It is important to point out that since these are images of the Earth's surface, each pixel in the 2D image raster corresponds to a real world point (geolocation). Satellite images are usually supplied with metadata that can be used to partially establish this one-to-one correspondence with the assumption that the associated geographic

(a)

(b)

Fig. 2.2.: At top is a portion of a WV3 PAN image from Ohio, USA. At bottom is a portion of a WV3 MS image covering the same area. Only the Red, Green and Blue bands have been displayed for the MS image.

area of the Earth's surface is approximated by a 2D plane. To establish a true and complete correspondence, one needs the information about the height of the world point above the Earth's surface.

The following section talks about some of the important metadata that is supplied with WV3 images.

2.1.2 Metadata Associated with Satellite Images

Satellite imagery vendors such as DigitalGlobe usually supply specific metadata for each satellite image. Such metadata can include, among other things, the satellite elevation and azimuth angles, the solar elevation and solar azimuth angles which indicate the position of the Sun when the image was recorded and an approximation to the camera model in the form of rational polynomial coefficients (RPCs). This metadata can be embedded in the image header or stored separately. In Table 2.1 we list some of the important fields that are usually contained in the metadata of a WV3 satellite image. Additionally, each WV3 image is accompanied by an IMD file that contains information about the spectral response and bandwidth of the sensor. These metadata are necessary for different applications such as orthorectification, ToA correction, etc. We elaborate upon these applications in the subsequent sections. But first, we need to talk about one of the most important metadata - the rational polynomial coefficients (RPCs) that serve as an approximation to the physical camera model (also called the rigorous projection model).

Table 2.1.: Some important fields in the metadata of a WV3 satellite image

Field	Description
GEOGCS	Geographic Coordinate System used for specifying the geodesic coordinates
HEIGHT_OFF	Height offset. This is h_{offset} in Eq. 2.2
HEIGHT_SCALE	Height scale. This is h_{scale} in Eq. 2.2
LAT_OFF	Latitude offset. This is ϕ_{offset} in Eq. 2.2
LAT_SCALE	Latitude scale. This is ϕ_{scale} in Eq. 2.2
LONG_OFF	Longitude offset. This is λ_{offset} in Eq. 2.2
LONG_SCALE	Longitude scale. This is λ_{scale} in Eq. 2.2
LINE_OFF	Scanline or row offset. This is l_{offset} in Eq. 2.2
LINE_SCALE	Scanline or row scale. This is l_{scale} in Eq. 2.2
SAMP_OFF	Pixel number or column offset. This is s_{offset} in Eq. 2.2
SAMP_SCALE	Pixel number or column scale. This is s_{scale} in Eq. 2.2
SAMP_DEN_COEFF	Coefficients of the denominator polynomial for the column, i.e., $\mathbf{f^s_{Den}}$ in Eq. 2.1
SAMP_NUM_COEFF	Coefficients of the numerator polynomial for the column, i.e., $\mathbf{f^s_{Num}}$ in Eq. 2.1
LINE_DEN_COEFF	Coefficients of the denominator polynomial for the row, i.e., $\mathbf{f^l_{Den}}$ in Eq. 2.1
LINE_NUM_COEFF	Coefficients of the numerator polynomial for the row, i.e., $\mathbf{f^l_{Num}}$ in Eq. 2.1
SUN_AZIMUTH	The solar azimuth angle.
SUN_ELEVATION	The solar elevation angle.

2.1.2.1 Rational Polynomial Coefficients

In order to map a point on the ground to its pixel coordinates in an image, we need the camera model for the satellite. The true camera model, also called the rigorous projection model (RPM) is quite complex and involves accounting for the satellite trajectory, its velocity, atmospheric refraction and relativistic effects. For most applications, the rational polynomial coefficients (RPCs) [14] serve as an highly accurate approximation to the RPM and are much easier to use. They are ratios of polynomials that map the 3D coordinates of a world point denoted by a triplet – (latitude (ϕ), longitude (λ), height (h)) to the 2D pixel coordinates denoted by a tuple – (sample (s), line (l)). Note that in the remote-sensing community, the terms "line" and "sample" are popularly used in place of the terms "row" and "column" respectively. Formally, the RPC equations are defined below in Eq. 2.1 and Eq. 2.2:

$$l_n = \frac{\mathbf{f^l_{Num}}(\phi_n, \lambda_n, h_n)}{\mathbf{f^l_{Den}}(\phi_n, \lambda_n, h_n)} \quad , \quad s_n = \frac{\mathbf{f^s_{Num}}(\phi_n, \lambda_n, h_n)}{\mathbf{f^s_{Den}}(\phi_n, \lambda_n, h_n)} \tag{2.1}$$

where each "**f**" is a different cubic polynomial in ϕ_n, λ_n and h_n with 20 coefficients each. ϕ_n, λ_n, h_n, l_n and s_n are the normalized latitude, longitude, height, line and sample coordinates respectively and are defined as:

$$\phi_n = \frac{\phi - \phi_{\text{offset}}}{\phi_{\text{scale}}}, \qquad \lambda_n = \frac{\lambda - \lambda_{\text{offset}}}{\lambda_{\text{scale}}}, \qquad h_n = \frac{h - h_{\text{offset}}}{h_{\text{scale}}}$$

$$l_n = \frac{l - l_{\text{offset}}}{l_{\text{scale}}}, \qquad s_n = \frac{s - s_{\text{offset}}}{s_{\text{scale}}} \tag{2.2}$$

In the above equations ϕ, λ and h denote the latitude, longitude and elevation coordinates for a given geolocation. The variables l and s denote the corresponding line (pixel row) and sample (pixel column) values. The offsets, the scale parameters and the RPC coefficients in these equations are provided as part of the metadata associated with the satellite image. Please refer to Table 2.1 for more details.

Grouping together the operations in Eq. 2.1 and Eq. 2.2, we write the following expression:

$$\text{Proj}_{\text{RPC}}(\phi, \lambda, h) = (s, l) \tag{2.3}$$

to denote the forward projection from a world point (ϕ, λ, h) into its corresponding pixel (s, l).

Errors in the RPCs – Errors in the measurements of the satellite position, velocity and attitude translate into errors in the parameters of the RPC model and these errors are obviously different for each image. For now, we will like to draw the attention of the reader to the l_{offset} and the s_{offset} parameters in Eq. 2.2. In Section 3.3.3, we will see how reducing the alignment error between the images boils down to correcting the errors in these two parameters for each image.

2.2 Top-of-Atmosphere Correction

In the images that are captured by the satellite sensor, the value recorded at each pixel depends on the amount of light that enters the camera aperture, as well as on the filters and the detectors of the sensor that convert this light to a measured signal. As specified in the work in [15], the images provided by DigitalGlobe contain radiometrically corrected image pixels as *recorded* at the sensor. As a first step, these values have to be converted to a measure of the spectral radiance that *enters* the camera aperture. We refer to the latter as the top-of-atmosphere (ToA) radiance. To do this conversion, we use the absolute radiometric calibration factor and the effective

bandwidth which are specified in the IMD file that comes with each image. Details of the corresponding equations are specified in the work in [15].

It is possible that the images have been recorded under vastly different conditions including different solar angles, different distances between the Earth and the Sun, and during different seasons. Hence, the ToA radiance for the same world point can be different for different images. Therefore, radiometric balancing is usually applied before using multiple such images together in any remote-sensing application. Radiometric balancing as defined in [15] amounts to normalizing the solar obliquity angle to 0° and the mean Earth-Sun distance to 1 Astronomical Unit (AU). This can be done using the time of image-capture and the metadata in the IMD files.

In many remote-sensing applications, it is also a common practice to convert the ToA radiance to ToA reflectance values. Reflectance is a measure of the proportion of the amount of light reflected, to the amount of light that is incident at a point. It is a dimensionless quantity with a value between 0 and 1. One can view this step as a data-normalization step. We refer to all these steps together as ToA correction. A detailed process outlining ToA correction for WV3 images can be found in the work in [15]. It should be noted that these steps do not account for atmospheric distortion and topographic effects. To do so, we have to use sophisticated approaches that require a model of the weather conditions at the time of image-capture. These are beyond the scope of our discussion.

For the task of semantic labeling, on the one hand we can expect that widely-varying spectral signatures for the same world point across images can make it difficult to accurately label the world point. On the other hand, CNNs have been shown to be quite adept at handling varying illumination conditions in traditional photographs. Therefore, it is not immediately evident if ToA correction is required since we use a CNN-based semantic labeler. In Section 9.4.2 we provide an answer to this question via quantitative evaluation.

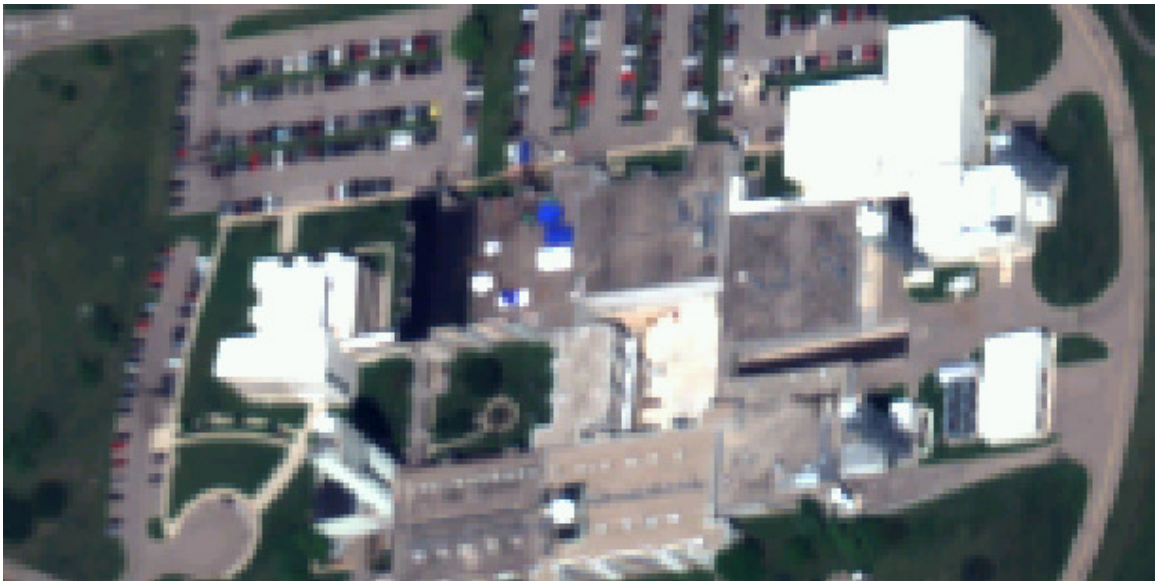
2.3 Pansharpening

A closer look at Fig. 2.2 reveals that the MS image does not clearly capture important features like the building edges due to its lower resolution. However such features are very important for semantic segmentation using a CNN. Since the PAN and MS images are captured simultaneously, it would be very useful to create a higher resolution MS image using the corresponding PAN image. These higher resolution MS images are called as pansharpened (PS) images. They are not directly recorded by the satellite. Instead they are created by fusing the MS and PAN images through a process called pansharpening. A PS image will contain multispectral band measurements at the same spatial resolution as the corresponding PAN image. Some of the steps involved in pansharpening include upsampling the MS image, spatially registering it with the PAN image, applying a weighted interpolation of the MS values, and optional additional steps that can reduce image-distortion. In this work, we use the "weighted Brovey" [16] method to create PS images.

Figure 2.3 shows a portion of a MS and PS image corresponding to the same geographic region. One can clearly observe the difference between the spatial resolution of the MS image and that of the PS image. A casual observer might wonder why one does not always use PS images instead of MS images. The reason is that pansharpening is computationally intensive and time consuming when applied to satellite images covering large geographic regions. PS images are used only when higher spatial resolution is an important requirement for the application.

2.4 Elevation Data – LiDAR Point Clouds, DEM and DSM

We now take a brief diversion from satellite images to talk about sources of remotely-sensed data that provide information about the heights of the world points. The need for elevation information will become clear in the subsequent sections.



(a)

(b)

Fig. 2.3.: At top is a portion of a lower-resolution multispectral image. At bottom is the corresponding portion of the higher-resolution pansharpened image.

There are three popular sources of elevation information in the remote-sensing community. They are Light Detection and Ranging (LiDAR) point clouds, Digital Elevation Models (DEMs) and Digital Surface Models (DSMs).

LiDAR – LiDAR is an airborne-system-based remote-sensing method where light pulses are emitted by a laser and the reflected pulses are captured by a sensor. The return-time differences are used along with a GPS to capture the 3D shape of the Earth's surface. Often, the point clouds are converted to a grid-based elevation map representing a nadir view of the surface. Currently, LiDAR point clouds are the most accurate sources of elevation information. However, there are many cons to using LiDAR data, some of which are listed below.

Why can't we rely on LiDAR?

1. LiDAR data is quite expensive to collect.

2. LiDAR data is not publicly available across the globe. Moreover, there are many administrative challenges to collecting LiDAR data as one might require permission from different countries/governments for operating aircraft within their airspaces.

3. LiDAR data can be out of date with the satellite images under consideration, resulting in possibly large mismatches between the two data sources.

4. The LiDAR point clouds and the images are collected by different sensors and therefore have to be aligned with one another before they can be used together. This is especially difficult if they are out of date with one another.

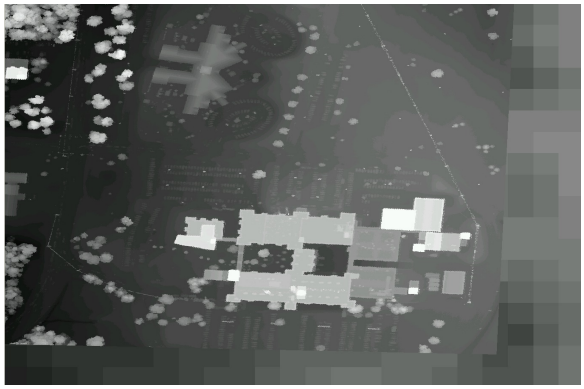
DEM – In remote-sensing parlance, a Digital Elevation Model (DEM) is a nadir-view map which provides the elevation of the bare-earth terrain but does not contain the heights of elevated structures such as buildings, bridges and trees. DEMs can be derived from LiDAR point clouds or from RADAR. An example of the latter is the Shuttle Radar Topography Mission (SRTM) DEM [11]. DEM is usually of much

coarser resolution. This research uses a SRTM DEM that has a GSD of 1 arc-second (roughly 30 m). Despite its coarseness and the fact that it does not contain the heights of the buildings, a DEM is still quite useful to get a rough sense of the elevation in the region of interest.

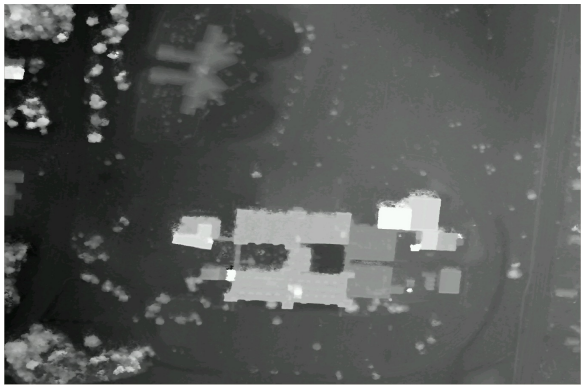
DSM – With respect to the work in this manuscript, Digital Surface Models (DSMs) are the most important sources of elevation information. *Note that while DSMs can be derived from LiDAR point clouds, in this manuscript, the term DSM refers to one that is created solely by stereo reconstruction using satellite images.* Like a DEM, a DSM also provides a nadir or top-down view of the Earth's surface. However, it contains the height of elevated structures such as buildings and trees. DSMs created by stereo reconstruction are usually of much higher resolution than DEMs. For instance, in this work we create DSMs with a GSD of 0.25 m. Because they are created using the satellite images, discrepancies due to time differences (such as those between the LiDAR and the satellite images) are minimized. Automatic construction of accurate DSMs using satellite images is quite complicated and will be described in Chapters 3 and 4. Note that unlike the LiDAR point clouds that are 3D, a DSM by definition is a 2.5D data source because it is created to provide a nadir view.

In Fig. 2.4 we compare portions of a LiDAR-based elevation map, a DSM constructed using WV3 images and the SRTM DEM for a region in Ohio. The LiDAR data was provided as part of the CORE3D program [17]. The LiDAR data, DSM and DEM have GSDs of 0.5 m, 0.25 m and 30 m respectively. The high quality of the LiDAR is reflected in the sharpness of the building edges. However, LiDAR is quite expensive to collect and in fact, this LiDAR data only covers a 1 km^2 region whereas the DSM that we construct using stereo processing spans a 100 km^2 region.

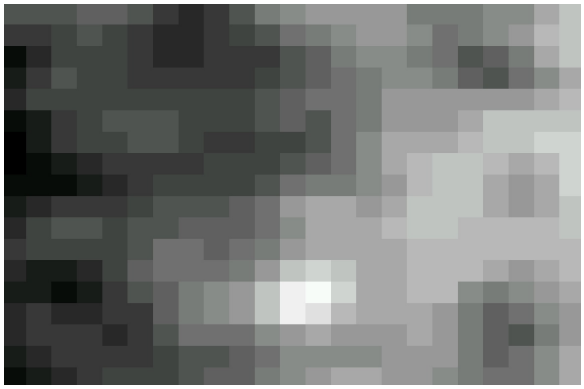
We will now describe how to use the elevation information along with the RPCs to perform "orthorectification" of the satellite images.



(a) LiDAR



(b) DSM



(c) DEM

Fig. 2.4.: At top is a portion of a nadir view of a LiDAR-based elevation map from Ohio. In the center is the corresponding portion of a DSM created by stereo reconstruction using WV3 images. At bottom is the portion of the publicly-available SRTM DEM covering the same region. The LiDAR map was provided as part of the CORE3D program [17].

2.5 Orthorectification

As mentioned in Chapter 1, the objective of this work is to label geographic world points and not just images. To label the world points using a machine-learning framework, one first needs to associate the world points with spectral values derived from the off-nadir images. This will also make it easier to fuse data from different satellite images captured at different off-nadir angles. Additionally, as discussed in Sections 5.1.3 and 5.1.4, this will also facilitate the extraction of training labels from OpenStreetMap (OSM).

Orthorectification is the process by which the pixels in an image are mapped to their corresponding world points in order to create an ortho or nadir view of the ground. In common practice, the following steps are used to create an orthorectified image that corresponds to a particular off-nadir image –

1. Using the RPCs and an optional source of elevation information such as a DEM, we first find the longitude and latitude coordinates of the corners of the geographic area of interest (AOI) that is covered by the image. Note that the RPC equations in Eq. 2.1 provide a mapping from a world point to its pixel coordinates. In order to go from the pixel coordinates of the corners to their corresponding world points, we use an iterative solution that minimizes an error function. This error function is the distance between the true pixel coordinates of a corner, and the RPC-based projection (into the image) of the current estimate of the world-point that corresponds to that corner.

2. Once the corners have been estimated in latitude and longitude units, we divide the AOI into a rectangular grid with a specified GSD. If a DEM is available, then the height for each grid cell is obtained from the DEM. Otherwise, we use a constant value equal to the HEIGHT_OFF parameter (Table 2.1).

3. The world point corresponding to the center of each grid cell is then projected into the image using the RPC equations in Eq. 2.1. The spectral values of that

pixel are assigned to that grid cell. The spectral values can also be interpolated over a small neighborhood of the pixel. The grid thus created can be saved as an image where each pixel corresponds to a grid cell. Such an image is called an orthorectified image or ortho image.

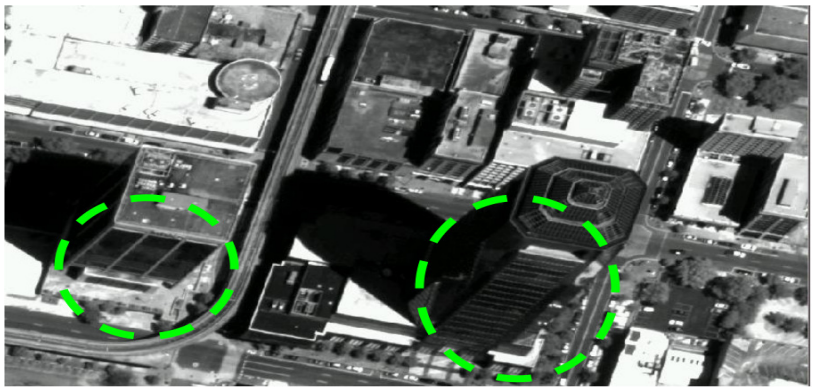
Orthorectification as described above is quite straightforward and the Geospatial Data Abstraction Library (GDAL) [18] provides a very popular tool called "gdalwarp" [19] for orthorectifying satellite images using RPCs.

2.5.1 True Orthorectification with a DSM

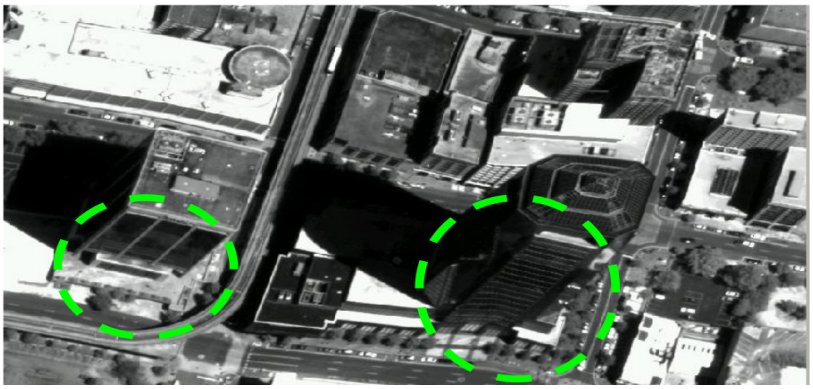
The steps for creating orthorectified images as described in the previous section suffer from some important drawbacks that can be nicely illustrated with some visual aids. In Fig. 2.5, we compare a portion of the orthorectified images that have been created by running the "gdalwarp" utility on the same image with three different height sources. Fig. 2.5a represents the case when we use a constant height of 0 meters for all points. Fig. 2.5b shows the orthorectified image created using a SRTM DEM as the height source and Fig. 2.5c depicts the orthorectified image created using a DSM as the height source.

Two artifacts leap to the eye when comparing the images in Fig. 2.5. These two artifacts have been highlighted using green and red circles. The source of these artifacts is described below.

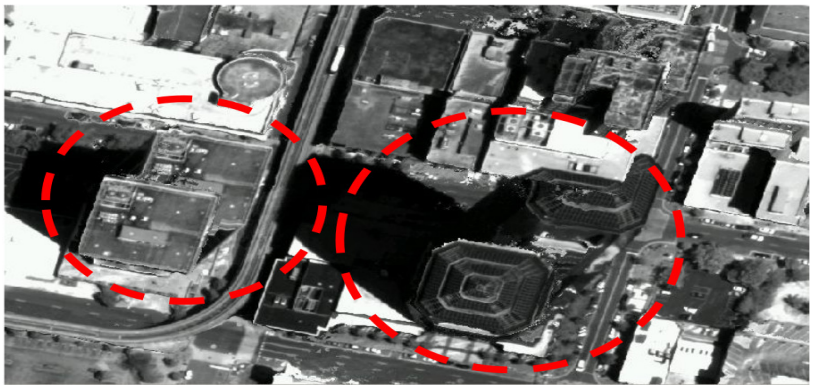
- The green circles in Figs. 2.5a and 2.5b indicate that the vertical walls of buildings are visible in these two orthorectified images. Strictly speaking, an ortho view is a view in which the camera is looking straight down at the ground. Therefore vertical walls should not be visible in a true ortho view. This artifact arises because the heights of the buildings are not taken into account when using a constant height or a DEM. **Therefore, a DSM is necessary to create a true ortho image.**



(a)



(b)



(c)

Fig. 2.5.: Shown in the subfigures are portions of orthorectified images created using the gdalwarp [19] utility. The image in Fig. 2.5a was created using a constant height of 0 meters. The image in Fig. 2.5b was created using a SRTM DEM as the height source, while the image in Fig. 2.5c was created using a DSM as the height source. Important artifacts are highlighted using red and green circles. A discussion about these artifacts can be found in the text of the manuscript.

- Fig. 2.5c shows a portion of an orthorectified image created using a DSM and the "gdalwarp" utility. We can see that it is a true ortho view and the vertical walls of the buildings are not visible any more. However, we now see some bizarre-looking artifacts where the same building appears to have magically duplicated itself. These are marked by the red circles in Fig. 2.5c. **We call these artifacts as "ghosts"**.

  These "ghosts" are created because the "gdalwarp" utility does not detect portions of the AOI that are occluded by tall structures. Consider the illustration in Fig. 2.6. Both point A on the ground and point B on the roof of a building project to the same pixel P in the off-nadir image via the RPC model. In reality, point A is not visible in the off-nadir image since it is occluded by the tall building in front of it. However, the "gdalwarp" utility blindly assigns the spectral value of P to both A and B in the orthorectified grid, resulting in duplicated buildings or "ghosts".

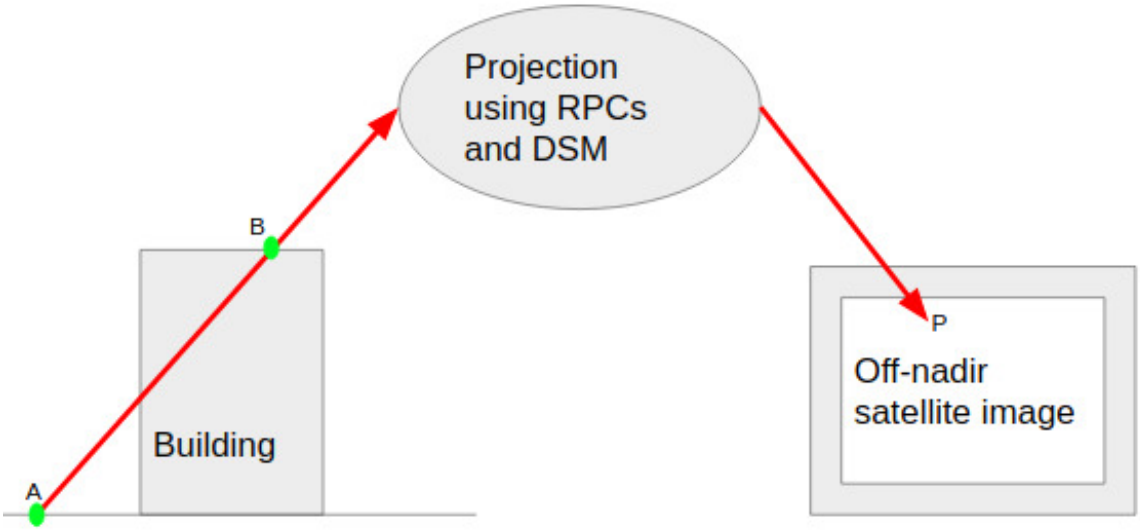


Fig. 2.6.: An illustration to show why duplicate buildings ("ghosts") are created when orthorectifying an image using a DSM as the height source with the "gdalwarp" utility. The spectral value of pixel P is assigned to both world points A and B, resulting in duplicated buildings.

In summary, to create an orthorectified image that contains a true nadir view of the ground, we need a DSM. Additionally, we need to detect portions of the ground that are occluded by elevated structures. We refer to this process as "true orthorectification".

As of the time of writing this manuscript, there are no open-source utilities to create true ortho images using RPCs and DSMs. Moreover, we need to orthorectify multiple images covering large areas, in an efficient manner. **To do this we have developed a utility that we have named "gwarp++". Using this tool, we can create true ortho images covering large areas, quickly and efficiently.** More details about this utility can be found in Section 5.1.2.

As mentioned in the previous paragraphs, an orthorectified grid can be saved as an image where each grid cell is mapped to a pixel. Within this ortho image, the one-to-one map between its pixels and the grid-cell points is given by two linear equations or equivalently a matrix transformation as shown below in Eq. 2.4. This matrix is commonly referred to as the georeferencing matrix.

$$\begin{bmatrix} longitude \\ latitude \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} \begin{bmatrix} \text{pixel column} \\ \text{pixel row} \\ 1 \end{bmatrix} \tag{2.4}$$

2.5.2 Iteratively Finding the World Point that Projects to a Pixel

Given a world point (ϕ, λ, h), it is quite straightforward to calculate $\text{Proj}_{\text{RPC}}(\phi, \lambda, h)$ (Eq. 2.3) by plugging the ϕ, λ and h values into the RPC equations. In contrast, finding the 3D world point that corresponds to a given pixel (s, l) is more challenging. Obviously this can be done only if we have access to LiDAR data or a DSM. To find the world point, we use an iterative "inverse-RPC-projection" procedure. In this procedure, we apply a binary search along the backprojected ray from (s, l) to find

a world point (ϕ, λ, h) such that $\text{Proj}_{\text{RPC}}(\phi, \lambda, h)$ is as close as possible to (s, l) and h is as close as possible to the DSM/LiDAR elevation value at $(\phi,\ \lambda)$. Following the convention in Eq. 2.3, we denote this procedure as $\text{Proj}^{-1}_{\text{RPC}}$. Therefore,

$$\text{Proj}^{-1}_{\text{RPC}}(s, l) = (\phi, \lambda, h) \tag{2.5}$$

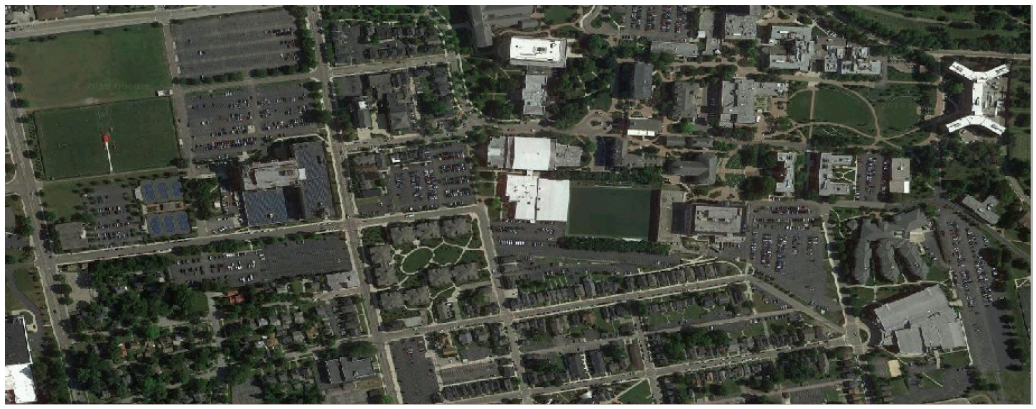
We now switch the discussion to the third important source of georeferenced data called OpenStreetMap (OSM) which will act as the source of training labels for our deep-learning framework.

2.6 OpenStreetMap

OpenStreetMap (OSM) [6] is a publicly available mapping database that is created and maintained by volunteers around the globe who manually annotate orthorectified images. It belongs to the category of Volunteered Geographic Information (VGI) systems. OSM data contains among other things, the outline of buildings, centerlines of roads, railway tracks, power transmission lines, etc. The images that are used for creating such annotations are typically of higher resolution than WV3 images.

Unlike satellite images which are rasters, OSM data is available in vector format. For example, the centerline of a road is represented as a polyline, i.e., a line made up of short straight line segments. Each line segment is represented by its two endpoints in longitudinal and latitudinal coordinates. For another example, buildings are represented by polygons using the coordinates of the corners of the polygon. Sometimes, the buildings are stored in 3D where each corner has the height information as well. However, predominantly, OSM buildings are represented by 2D polygons. Figure 2.7 shows the OSM buildings and roads overlaid on top of a Google Earth [20] image of a 0.7 km^2region in Ohio, USA.

Another useful feature of OSM is that the data is often stored/organized in an hierarchical order and can be filtered using additional tags. We can use these tags to



(a) Google Earth image

(b) OSM buildings overlaid on top of the Google Earth image as red polygons

(c) OSM roads overlaid on top of the Google Earth image as green line segments

Fig. 2.7.: OSM buildings and roads for a 0.7 km^2 region in Ohio, USA

extract data relevant to the application. For example, we can choose to select bicycle lanes or major highways depending upon our application.

2.6.1 Noise in OSM

OSM data is published under the Open Database License [21] and is free from restrictions imposed on proprietary map data such as Google Maps, etc. However, the free and open nature of the data means that it does come at a cost. OSM data is noisy with many annotation errors. It is hard to precisely quantify the amount and variation of this error across the globe. Some of the common types of annotation errors are as follows:

1. The object was not annotated. This is very common in rural and semi-urban areas.

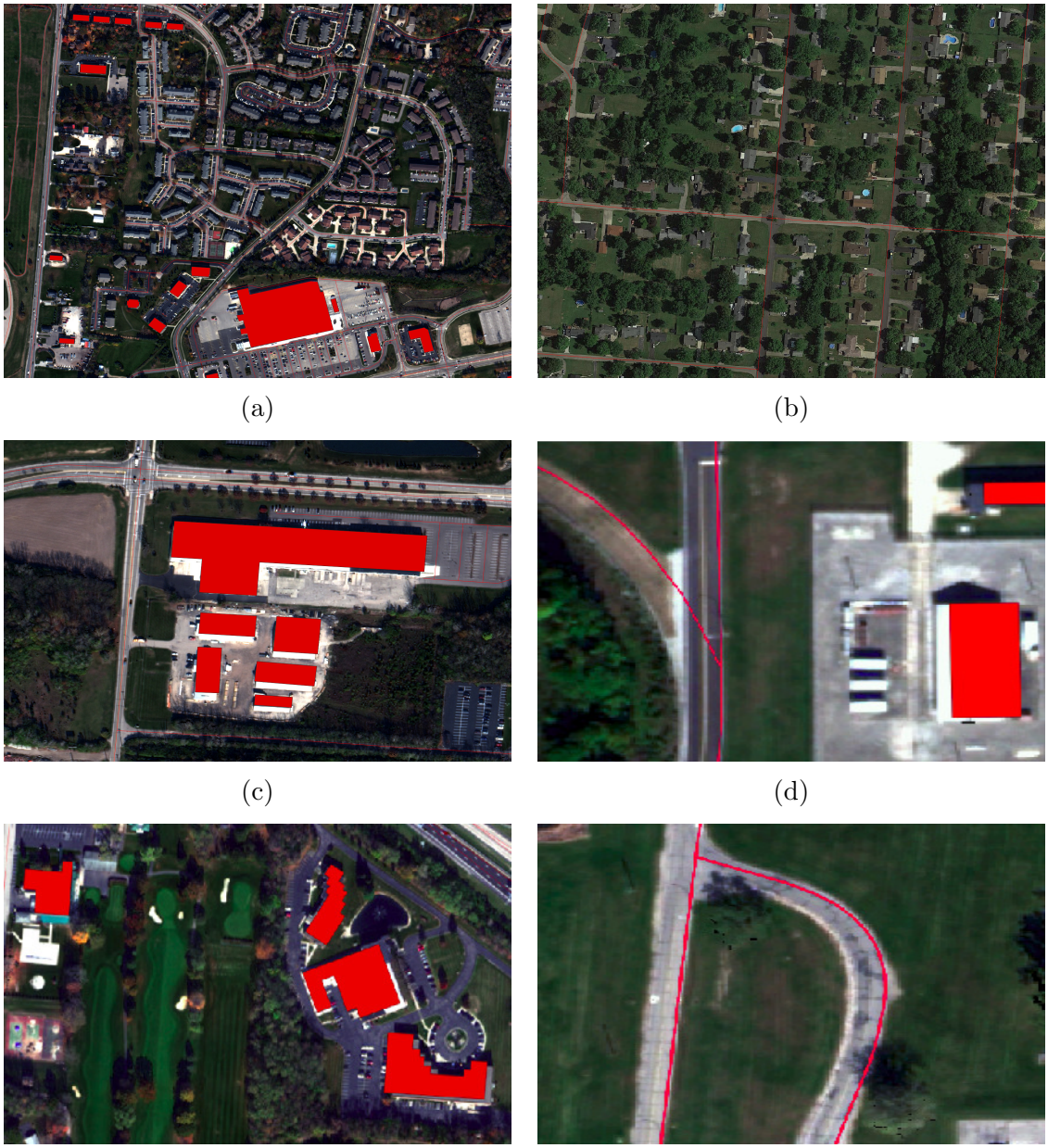
2. The annotator made a mistake. For example, the edges of buildings can be annotated incorrectly.

3. The object that was annotated using an older image does not exist any more in our WV3 images and vice versa.

4. Roads are marked only by their centerlines with no information about their width. For large highways, these centerlines might not necessarily be along their center but rather, along one of the lanes.

In addition to these annotation errors, in order to derive training labels from OSM for our deep-learning framework, we need to account for misalignment between the OSM and the true orthorectified satellite images. Aligning the OSM data with the satellite images is a challenging problem unto itself and we present a robust solution for the same in this manuscript.

In Fig. 2.8, we show some examples of noisy OSM buildings and roads. Figs. 2.8a and 2.8b show examples of missing OSM buildings that have not been annotated.



(a) (b)

(c) (d)

(e) (f)

Fig. 2.8.: Assorted examples of the noise in OSM. The OSM buildings and roads have been overlaid in red on top of a true ortho WV3 image. Figs. 2.8a and 2.8b show missing OSM buildings. Figs. 2.8c, 2.8d, 2.8e and 2.8f are zoomed-in sections highlighting the misalignment between the OSM data and the image.

Figs. 2.8c, 2.8d, 2.8e and 2.8f are zoomed-in sections highlighting the misalignment between the OSM data and the image.

We conclude this chapter with a small primer on some important software that have been useful to this research.

2.7 Some Popular Software for Processing and Viewing Remotely-Sensed Data

There are a lot of software, both open-source and commercial, for viewing and processing remotely-sensed data. Commercial software like ENVI [22] are quite expensive, especially for cloud-based distributed processing of large AOIs. We will restrict the discussion to important open-source and free software that have been used in this research.

For viewing and simple processing of satellite images and OSM data, the Quantum Geographic Information System (QGIS) software [23] is extremely useful. It also provides an interface to write custom Python plugins. It supports a number of installable open-source Python plugins for simple vector and raster data processing such as filtering OSM data by attributes, thickening OSM roads, subtracting or adding rasters, etc. One can also create new layers and annotations using QGIS.

By default, QGIS will use the RPCs embedded in a satellite image's metadata to provide an orthorectified view of the image. If one wants to look at the original off-nadir image, the Monteverdi viewer [24] which is a part of the Orfeo Toolbox [25] is very useful.

For more complex processing of satellite images (and raster data), the Geospatial Data Abstraction Library [18] is a very powerful candidate and supports C, C++, Java and Python APIs in addition to supporting shell-based commands. In fact, many of the processing tools in QGIS call GDAL under the hood. For simple orthorectification we use the "gdalwarp" tool [19], and for pansharpening we use the

"gdal_pansharpen.py" utility [26], both of which are part of the GDAL software [18]. GDAL can be installed within or outside the popular Anaconda [27] environment.

For processing vector data such as OSM, the OGR toolkit that is part of GDAL is a great software that supports many operations such as filtering attributes by tags, converting vectors to raster data, thickening of OSM vector data, merging multiple OSM data files, etc.

For viewing 2.5D data such as DSMs, QGIS is sufficient. However to get a 3D view of DSMs and LiDAR point clouds, we recommend the free Quick Terrain Reader that is provided by Applied Imagery [28]. They also offer a paid version of the software called Quick Terrain Modeler which is expensive but arguably one of the best software today for visualizing and processing 3D remotely-sensed data. For quick visualization of small point clouds, the open-source CloudCompare [29] is also a good option.

The code used in this research has been developed using a mix of C++, Python and Bash scripts. Many scripts have been developed to run in a distributed fashion across multiple virtual machines (VMs) running on an OpenStack [10] based in-house cloud (RVL Cloud [30]). A number of design choices, network considerations, hardware and software optimizations are required for efficient cloud-based processing of large AOIs as used in this work. We provide a detailed discussion of our cloud-based setup in Chapter 8. *This manuscript attempts to not only present state-of-the-art computer-vision and deep-learning algorithms, but also to discuss the often-overlooked yet extremely important computational and software-development aspects of creating a large-area system.* We hope this will be useful for researchers who wish to develop algorithms for large-scale processing of satellite images in a relatively inexpensive manner. By inexpensive, we mean in comparison to the cost of running virtual machines (VMs) on paid cloud-based services such as Amazon Web Services (AWS) [31], Microsoft Azure [32], etc.

3. TILE-BASED CONSTRUCTION OF DSMS

3.1 Motivation for Constructing DSMs

As explained in Chapter 1, we are interested in accurately labeling world points rather than merely labeling the pixels of satellite images. In order to do so, we need to first associate the world points with spectral values derived from the corresponding pixels in the satellite images. True orthorectification can help create this association. The discussion in Section 2.5.1 has made it crystal clear that it is necessary to possess knowledge of the heights of elevated structures such as buildings and trees in order to create true ortho views of the ground. This knowledge of the heights can come from either the LiDAR point clouds or from the Digital Surface Models (DSMs) created via stereo processing of the images. The disadvantages of using the LiDAR point clouds have already been listed in Section 2.4. **We are therefore left with no choice but to construct the DSMs using the satellite images.**

Creating DSMs for areas as large as 100 km^2 using multi-date and multi-view satellite images is an onerous task unto itself due to the following reasons:

- *Variability of the data* - Important steps in the DSM construction pipeline include image alignment, stereo rectification and stereo matching, all of which rely on extracting reliable correspondences between images. Extracting a "correspondence" means correctly identifying the pixels belonging to the same world point in different images. This is made challenging by the following factors:

    1. The images are captured at varying view angles, and during different times of the year. In fact, some images used in this study have been captured as far apart as 2 years.

2. Small changes in the solar elevation angle can cause significant changes in illumination.

3. The image content can vary significantly across a 100 km^2 region.

- *Size of the data* - The WV3 images that are used in this study are typically of size 43008 $\times$ 38000 in pixels and cover an area of the ground of size 147 km^2. This translates to roughly 1.6 billion pixels. *Many traditional computer-vision algorithms are designed for much smaller images with sizes of the order of 1000 $\times$ 1000 = 1 million pixels, roughly three orders of magnitude smaller than WV3 images.* Such algorithms do not directly scale and generalize to large WV3 images.

Most contributions in the public literature on DSM construction using multi-date satellite images only consider small areas covering a few tens of km^2. Even the large-area DSM contribution published in [33] is based on a very small number of in-track images that are typically captured just seconds or minutes apart by the Pléiades satellite. We include a review of these contributions in Section 3.2. It should be mentioned that there are a few commercial entities such as Vricon [34] which claim that they can construct large-area DSMs using multi-date and multi-view images. However, they use proprietary algorithms and the extent of human effort that goes into creating their DSM products is not public knowledge. Moreover, purchasing DSMs from such commercial operators is quite expensive and not feasible for research. Additionally, we will run into the issue of mismatch between our images and the DSMs purchased from such a third party, due to misalignment and time differences between the two data sources.

Therefore, before we describe the process of generating training labels from OSM, or our machine learning framework, it is necessary to first discuss large-area DSM construction. *One of the novel contributions of this research is to demonstrate a distributed and scalable approach for constructing large-area DSMs using a cloud-based framework.*

We propose to construct the DSMs in a tilewise manner. In this tiling strategy, images are broken into *image patches*, with each image patch covering a *tile*[1] on the ground. The motivations for constructing the DSMs in such a tilewise fashion are discussed in Sections 4.1 and 4.1.1. Note that the notion of a tile is used only for aligning the images and for constructing the DSMs. *For the CNN-based machine-learning portion of the system, we work directly with the whole images and with the DSM and the OSM for the entire geographic area of interest.*

In the rest of this chapter, we focus exclusively on creating the DSM for a single such tile. Issues pertaining to merging the tile-level DSMs into a large-area DSM are discussed in Chapter 4. The rest of this chapter is organized as follows. We first present a brief literature review of prior state-of-the-art approaches to DSM construction. In the subsequent sections, we elaborate on three important steps in constructing a tile-level DSM – alignment of image patches, stereo matching, and fusion of point clouds into a 2.5D DSM. It is assumed that the reader is familiar with standard computer-vision concepts such as rectification, disparity, etc.

3.2 Relevant Literature

As mentioned in the discussion presented in [35], there are broadly two categories of approaches for creating DSMs. The approaches that belong to the first category divide the 3D scene into voxels, project each voxel into all the images and subsequently reason about the probability of occupancy of each voxel using the corresponding pixel features from all the images. The methods presented in [36], [37] and [38] belong to this category.

In the second category of approaches (henceforth referred to as *multi-view pairwise stereo*), multiple pairs of images are selected for stereo processing. Each such pair of images is rectified and then a dense disparity map is computed for each such pair.

[1] A more accurate way to refer to a tile would be that it exists on a flat plane that is tangential to the WGS ellipsoid model of the earth. This definition does not depend on whether the underlying terrain is flat or hilly.

The pair of images, the disparity map and the RPCs are then used in a triangulation step to produce a pairwise point cloud. Subsequently, multiple such pairwise point clouds are fused to produce a dense DSM. The algorithms described by the studies in [39], [40] and [41] are a few good examples of such approaches.

In the work presented in [42], the authors compare these two categories of approaches and conclude that multi-view pairwise stereo produces more accurate DSMs. It should also be noted that the top-performing approaches in the IARPA's Multi-View Stereo 3D Mapping Challenge [43] were all multi-view-pairwise-stereo approaches. This challenge was conducted using the dataset described in [35], which is one of the first public benchmark datasets for multi-view stereo from satellite images.

State-of-the-art approaches for constructing DSMs from satellite images are described by the studies in [44], [39], [42], [41], [45], [46] and [47]. *In all of these contributions, the DSMs that are constructed cover small areas, typically less than a few tens of* km^2. The large-area DSM contribution in [33] is based on a very small number of in-track images that are typically captured just seconds or minutes apart by the Pléiades satellite.

3.3 Image-to-Image Alignment

The first step in DSM construction is image-to-image alignment. Before describing the relevant algorithms, we first present an example to justify why image-to-image alignment is necessary.

3.3.1 An Example to Illustrate the Need for Image-to-Image Alignment

Errors in the GPS measurements of the satellite location and errors in the measurement of the satellite attitude translate into errors in the RPC model. As a consequence of these errors, if we were to use the RPCs to project a hypothetical world point into each of the WV3 images without first aligning the images, the projected pixels in the images could be off by as much as 7-10 pixels from their true recorded

positions. In Fig. 3.1 we illustrate this by projecting a single world point into four WV3 images without aligning the images.

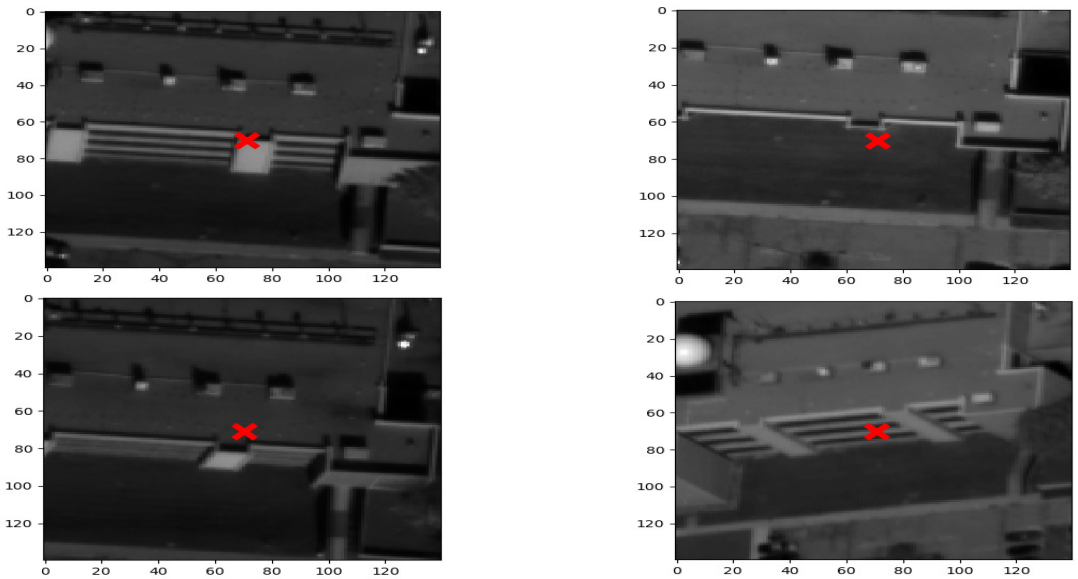


Fig. 3.1.: Due to misalignment, the same world point is projected into different erroneous pixel locations in four images. The projected pixels are marked in red.

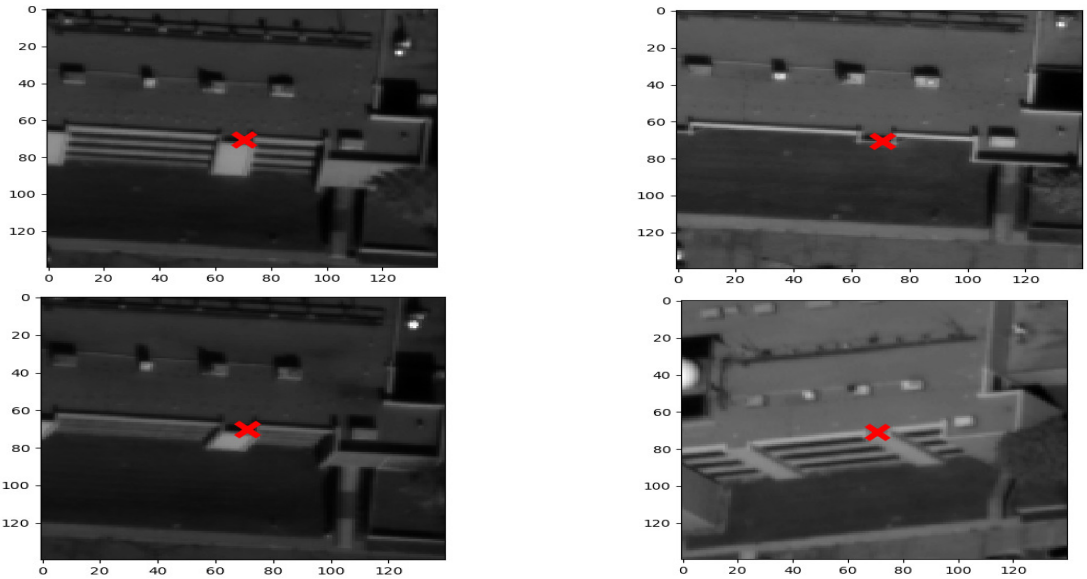


Fig. 3.2.: After alignment, the same world point is projected into the pixel locations in four images with sub-pixel precision. The projected pixels are marked in red.

Aligning the images means finding corrections to the RPCs such that if we were to project a hypothetical world point into each of the images using the corrected RPCs, the pixels thus obtained would correspond to their true recorded positions with sub-pixel precision. Fig. 3.2 shows the results obtained by first aligning the images and then projecting the same world point that was used in the example shown in Fig. 3.1. The improvement is obvious when we compare Figs. 3.1 and 3.2.

3.3.2 Relative vs Absolute Alignment Error

It is important to realize that there are two distinct components to image-to-image alignment. Firstly, the projected pixels in the different images should correspond to one another with sub-pixel precision. For instance, if a world point "WP" projects to the corner of a building in one image, then the projected pixels in the other images must also be as close as possible to the same corner of the building, unless said corner is not recorded or is occluded in an image. To achieve this, the alignment process should minimize the *local or relative alignment error* between the images. Note that for the relative alignment of the images, it does not matter whether or not the 3D coordinates of the world point "WP" actually correspond to the GPS coordinates of the corner of the building. As long as "WP" projects in a consistent fashion across the images, the images are well aligned relatively with one another. For many applications, minimizing the relative alignment error is sufficient. For example, we can construct high quality point clouds with images that have been relatively well aligned with one another with sub-pixel precision. However, *merely minimizing the relative error does not guarantee that the 3D geographical coordinates of the points in these point clouds will correspond to their true locations on the Earth's surface.*

Our application additionally requires that each world point should project as close as possible to its true location in each image. For instance, if we were to use a GPS sensor or surveying equipment to measure the 3D coordinates of the corner of a building, and subsequently project this 3D point into the images, the projections

should line up with the pixels corresponding to the building corner as recorded in each image. To achieve this, the alignment process must also minimize the *global or absolute error of alignment.* Therefore, we need to align the images with one another while simultaneously minimizing both the relative and the absolute errors.

3.3.3 Standard Approach to Bundle-Adjustment-Based Alignment of Image Patches

It was shown by Grodecki and Dial [48] that the residual errors in the rational polynomial coefficients (RPCs) on account of small uncertainties in the measurements related to the position and the attitude of a satellite can be corrected by adding a *constant bias* to the projected pixel coordinates of the world points, provided the area of interest on the ground is not too large. As will be explained in Chapter 4, the size of each tile used in our study is empirically chosen so as to satisfy this condition. Therefore, aligning the image patches that correspond to a tile means correcting the l_{offset} and the s_{offset} parameters in Eq. 2.2 or equivalently, the SAMP_OFF and the LINE_OFF parameters that are defined in Table. 2.1, for each image patch. *The corrections to these two offset parameters are referred to as the bias corrections.*

Camera calibration and alignment of images is a well-studied topic in computer vision. However, due to the relatively more complex content of multi-view and multi-date satellite images and in order to operate on a large-area basis, we had to extend the standard approach of bundle adjustment that is used to align small images. The standard approach consists of the following steps:

1. Extracting the interest points or keypoints using an operator like SIFT [49] or SURF [50]

2. Establishing correspondences between the keypoints in pairs of images on the basis of the similarity of their descriptor vectors

3. Using RANSAC [51] to reject the outliers in the set of correspondences (we refer to the surviving correspondences as the *pairwise tie points*)

4. Estimating the optimum bias corrections for each of the images by the minimization of a reprojection-error-based cost function in an iterative manner. A standard practice is to use an L2 function to calculate L_{relative}, i.e., the relative or local error of image-to-image alignment (Section 3.3.2). Formally, for N image patches,

$$L_{\text{relative}} = \sum_{i} \Bigg(\Big((R_k(X^i_{k,j}) - r^i_k)^2 + (C_k(X^i_{k,j}) - c^i_k)^2 \Big) + \Big((R_j(X^i_{k,j}) - r^i_j)^2 + (C_j(X^i_{k,j}) - c^i_j)^2 \Big) \Bigg)$$

In the above equation, i is used to index the world points. The indices j and k are used to index the image patches, where $j, k \, \epsilon \, [1, N]$. For instance, $X^i_{k,j}$ is used to denote the i^{th} world point and its subscript "k, j" is used to indicate that it is triangulated by using the tie points x^i_k and x^i_j that are detected in the images with indices k and j respectively. The row and the column coordinates of the tie point x^i_k are denoted by r^i_k and c^i_k respectively. $R_k(X^i_{k,j})$ and $C_k(X^i_{k,j})$ are respectively the row and the column coordinates of the pixel obtained by projecting $X^i_{k,j}$ into the k^{th} image. Since $R_k(X^i_{k,j})$, $C_k(X^i_{k,j})$, $R_j(X^i_{k,j})$ and $C_j(X^i_{k,j})$ are calculated using the RPCs, L_{relative} is a non-linear function of the bias corrections that need to be estimated for each image. Viewing this as a non-linear-least-squares-minimization problem, one can use the Gradient Descent (GD) or the Levenberg-Marquardt (LM) algorithm [52] to iteratively estimate the bias corrections that minimize L_{relative}.

We have extended the standard approach by: (1) augmenting the *pairwise tie points* with *multi-image tie points*; and (2) adding an L_2 regularizer to the reprojection-error-based cost function. In what follows, we start with the need for *multi-image tie points*.

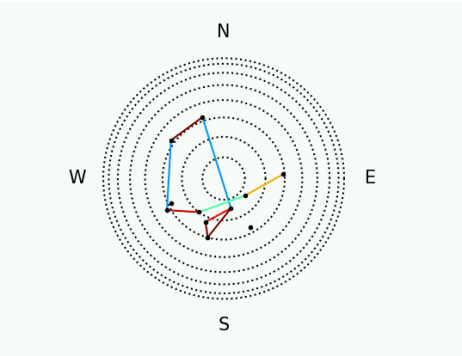


Fig. 3.3.: An attributed graph representing the pairwise tie points for 11 image patches that cover a 2 km^2 AOI in Kabul. Each vertex represents an image and is positioned at $(r{\cdot}\cos(\phi), r{\cdot}\sin(\phi))$ where ϕ is the azimuth angle and $r = \sin(\theta)$, with θ representing the obliquity angle at the time of image capture. The edges are colored based on the number of tie points detected between the corresponding vertices.

3.3.4 The Need for Multi-Image Tie Points

Our experience has shown that doing bundle adjustment with the usual pairwise tie points does not yield satisfactory results when there are significant illumination differences across the images. This could happen under a number of conditions such as when the Sun is just above the horizon or when there is significant snow-cover on the ground, etc. Under these conditions, the decision thresholds that one normally uses for extracting the keypoints from the images often yield an inadequate number of keypoints. And if one were to lower the decision thresholds, while that does increase the number of keypoints, it also significantly increases the number of false correspondences between them.

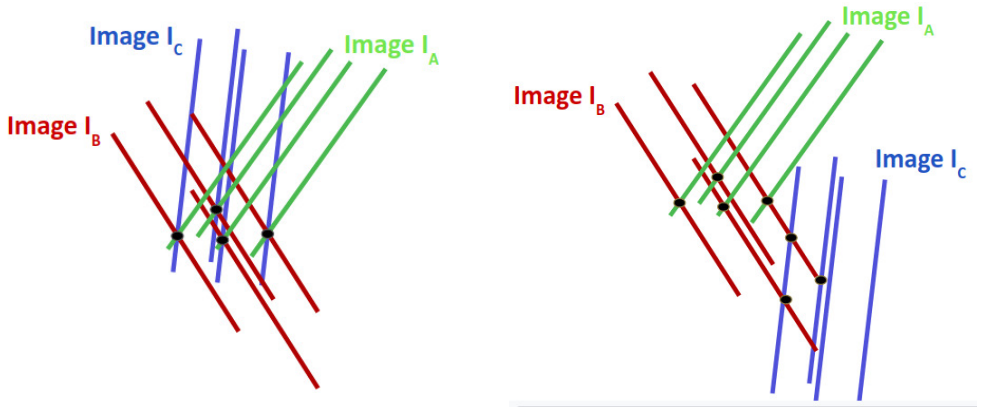
To illustrate the inadequacies of using pairwise tie points extracted with tight decision thresholds, we construct an attributed graph in which each vertex stands for an image and each edge for the number of keypoint correspondences between the corresponding pair of images. Fig. 3.3 shows the attributed graph thus obtained for eleven image patches that cover a 2 km^2 AOI in Kabul, Afghanistan. These images were captured mostly around the time of sunrise and therefore, small differences in the view angles can cause drastic illumination differences between the images. Although

the attributed graph displayed in Fig. 3.3 is sparse, we have still detected pairwise tie points in 9 out of the 11 images. Therefore, we might be tempted to conclude that these 9 images would be well aligned with one another which is a pretty good solution for the task at hand. However, as it turns out, due to the near-parallel nature of the rays of the pushbroom sensor, such a conclusion is far from the truth.

3.3.4.1 Ambiguity Due to the Near-Parallel Nature of the Pushbroom Sensor Rays

For convenience, we consider a relatively simpler task of aligning three images denoted by $\mathrm{I_A}$, $\mathrm{I_B}$ and $\mathrm{I_C}$. Let us assume that our algorithm has found correct pairwise tie points between the images $\mathrm{I_B}$ and $\mathrm{I_A}$ and between the images $\mathrm{I_B}$ and $\mathrm{I_C}$. Let us further assume that no tie points have been detected between the images $\mathrm{I_A}$ and $\mathrm{I_C}$ due to the choice of tight thresholds. Fig. 3.4a depicts a true solution that minimizes the L_{relative} error (Eq. 4). For simplicity, we assume that the tie points detected between the images $\mathrm{I_A}$ and $\mathrm{I_B}$ and the tie points detected between the images $\mathrm{I_B}$ and $\mathrm{I_C}$ correspond to the same world points. In Fig. 3.4a, we draw the rays backprojected from the tie points in each image. The points of intersection of these rays are the world points corresponding to the detected tie points. Note that the backprojected rays are near-parallel, as is the case for WV3 images that are captured from an altitude of ∼617 km above the ground. The backprojected rays from all the three images intersect at the correct world points which is to be expected for the true solution.

Now consider the camera configuration depicted in Fig. 3.4b where we have merely shifted the backprojected rays from image $\mathrm{I_C}$ along the principal axis of the camera that captured image $\mathrm{I_B}$. This corresponds to modifying the l_{offset} and the s_{offset} parameters (Eq. 2.2) of image $\mathrm{I_C}$. **Note that this configuration of cameras does not change the reprojection error between the images $\mathrm{I_B}$ and $\mathrm{I_C}$ due to the near-parallel nature of the backprojected rays. Therefore, it yields the**



(a) (b)

Fig. 3.4.: An illustration depicting two different camera configurations that equally minimize L_{relative} (Eq. 4)

same L_{relative} error as the true solution because L_{relative} is computed as the sum of such pairwise reprojection errors. However, we see that the backprojected rays no longer intersect at the same set of world points. What this means is that the images $\mathrm{I_A}$ and $\mathrm{I_B}$ are still well aligned with each other and the images $\mathrm{I_B}$ and $\mathrm{I_C}$ are still well aligned with each other, but the images $\mathrm{I_A}$ and $\mathrm{I_C}$ are no longer well aligned with each other. This also implies that a point cloud $\mathrm{P_{A,B}}$ reconstructed using the images $\mathrm{I_A}$ and $\mathrm{I_B}$, and a point cloud $\mathrm{P_{B,C}}$ reconstructed using the images $\mathrm{I_B}$ and $\mathrm{I_C}$ will not be aligned with each other.

To summarize, the calculation of the L_{relative} error on a pairwise basis combined with the near-parallel nature of the backprojected rays results in multiple camera configurations that equivalently minimize L_{relative}. However, not all of these configurations ensure that all the images are well aligned with one another. We cannot guarantee that our optimization algorithm will converge to a useful configuration.

In such cases, one gets better results overall by extracting what we refer to as *multi-image tie points*. The main idea in multi-image tie-point extraction is to construct a graph using a large number of keypoints detected with relaxed decision thresholds and then to identify the keypoints that correspond to the same putative world point across multiple images as opposed to just two images. Unfortunately, using multi-image tie points is computationally more expensive than using pairwise tie points — roughly three times more expensive. This is not acceptable since we need to process more than a hundred tiles. Therefore, multi-image tie points must be used only when needed.[2]

3.3.5 A Novel Graph-Connectivity-Based Algorithm to Detect Poor Alignment

We have developed a "detector" that automatically identifies the tiles that need the extra robustness provided by the multi-image tie points. The detector is based

[2]The bundle-adjustment framework and the tie-point extraction modules were developed by Dr. Tanmay Prakash.

on the rationale that the larger the extent to which each image shares keypoint correspondences with *all* other images, the more accurate the alignment is likely to be. This rationale is implemented by constructing an attributed graph in which each vertex stands for an image and each edge for the number of keypoint correspondences between a pair of images. If we denote the largest component in this graph by C, the extent to which each vertex in C is connected with all the other vertices in the same component can then be measured by the following "density":

$$D(C) = \frac{2|E_c|}{|C|(|C|-1)} \tag{3.1}$$

where $|E_c|$ is the total number of edges in C and $|C|$ is the total number of vertices in C. The detection of the need for multi-image tie points is carried out by first applying a threshold on $|C|$ and then on $D(C)$. This detection algorithm is described in detail in Algorithm 3.1. The algorithm is motivated by the observation that a dense tie-point graph based on pairwise tie points is indicative of good alignment.

Algorithm 3.1: To detect the need for multi-image tie points

N – Total number of image patches to be aligned

Step 1: Run alignment using pairwise tie points

T_p – Tie-point graph returned by alignment using pairwise tie points

V – Set of all image patches $\{v_i\}$. Each image patch is a vertex of T_p

E – Set of all edges $\{e_{ij}\}$. e_{ij} is an edge between the vertices v_i and v_j with a weight equal to the number of tie points between v_i and v_j

k, D_{min} – User-specified thresholds

AQ – Flag set to True if alignment is of satisfactory quality. Otherwise set to False.

Evaluate alignment quality

1. Find the largest connected component C of T_p.
   $|C|$ is the number of image patches in C. $|E_c|$ is the number of edges in C.

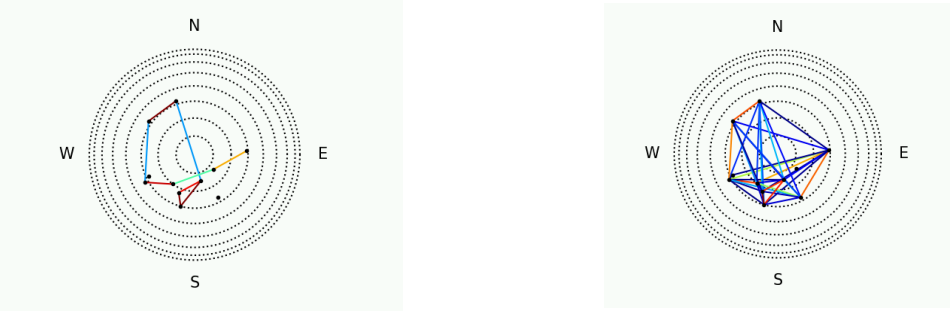
2. Check how many image patches have been aligned.
   If $|C| < k \cdot N$, where $0 < k < 1$, $AQ \leftarrow$ *False*. Return AQ

3. Check if C is a tree, i.e., if $|E_c| == |C| - 1$, $AQ \leftarrow$ *False*. Return AQ
   Explanation – The pushbroom camera model can be closely approximated by an affine camera model, i.e., the camera rays are almost parallel. Therefore, if C is a tree,
   then for each pair of image patches, the two image patches might be well aligned with
   each other. However, distinct pairs might not be aligned with one another.

4. Check the sparsity of C. $D(C)$ is the density of C. $D(C) = \frac{2|E_c|}{|C|(|C|-1)}$
   If $D(C) < D_{min}$, $AQ \leftarrow$ *False*. Return AQ

Step 2: If $AQ ==$ *False*, rerun alignment using multi-image tie points

In Fig. 3.5, we compare the attributed graphs obtained using the pairwise and the multi-image tie points from eleven image patches that cover a 2 km^2 AOI in Kabul. Fig. 3.5a is a reproduction of Fig. 3.3 and shows the sparse graph obtained using the pairwise tie points. In comparison, Fig. 3.5b shows the much denser graph obtained using the multi-image tie points.



(a) (b)

Fig. 3.5.: Comparison of the attributed graphs representing the pairwise and the multi-image tie points for a 2 km^2 AOI in Kabul. Each vertex represents an image and is positioned at $(r \cdot \cos(\phi), r \cdot \sin(\phi))$ where ϕ is the azimuth angle and $r = \sin(\theta)$, with θ representing the obliquity angle at the time of image capture. The edges are colored based on the number of tie points detected between the corresponding vertices.

3.3.6 Using Regularization to Reduce the Absolute Error of Alignment

As mentioned in Section 3.3.2, we need to minimize the absolute error of alignment as well. To do this, we add an L2 regularizer term to the $L_{relative}$ error (Eq. 4). The L2 regularizer is defined as:

$$L_{\text{regularize}} = \sum_{k=1}^{N} (\zeta_k^2 \cdot (l_{\text{offset}_k} - l^o_{\text{offset}_k})^2 + \nu_k^2 \cdot (s_{\text{offset}_k} - s^o_{\text{offset}_k})^2) \tag{3.2}$$

where the index k is used to indicate a specific image I_k, with $k \, \epsilon \, [1, N]$. l_{offset_k} and s_{offset_k} are respectively the optimal line and sample offset parameters that need to be calculated for I_k. $l^o_{\text{offset}_k}$ and $s^o_{\text{offset}_k}$ are respectively the uncorrected line and

sample offset parameters for $\mathrm{I_k}$, that are provided by the satellite vendors as part of the RPCs. Therefore, $l_{\mathrm{offset}_k} - l^o_{\mathrm{offset}_k}$ is the line bias correction and $s_{\mathrm{offset}_k} - s^o_{\mathrm{offset}_k}$ is the sample bias correction for $\mathrm{I_k}$. ζ_k^2 and ν_k^2 are the weights that control the extent of the regularization imposed on the bias corrections for $\mathrm{I_k}$.

Adding $L_{\mathrm{regularize}}$ to L_{relative}, we get the following expression for the total error to be minimized:

$$L_{\mathrm{absolute}} = L_{\mathrm{relative}} + L_{\mathrm{regularize}} \tag{3.3}$$

It can be shown that the L2 form of the $L_{\mathrm{regularize}}$ error as defined in Eq. 3.2 is a natural consequence of assuming that the line bias correction $l_{\mathrm{offset}_k} - l^o_{\mathrm{offset}_k}$ is drawn from a Gaussian distribution $\mathcal{N}(0, \zeta_k^2)$, and that the sample bias correction $s_{\mathrm{offset}_k} - s^o_{\mathrm{offset}_k}$ is independently drawn from an iid Gaussian distribution $\mathcal{N}(0, \nu_k^2)$. In fact, one can show that minimizing L_{absolute} is equivalent to obtaining a probabilistic estimate of the bias corrections while taking into account the prior probabilities of the bias corrections. In our study, we simply assume that $\zeta_k^2 = \nu_k^2 = 0.5 \ \forall \ k$. One could incorporate additional information into the algorithm by assuming more complex probability distributions for the priors.

In practice, L_{absolute} is converted into a matrix-vector form and the bias corrections that minimize L_{absolute} are calculated using matrix and vector operations. It turns out that there is a lot of sparsity in the matrices involved in these operations. This sparsity can be exploited by applying sparse-bundle-adjustment techniques (SBA [53], [54]) to speed up the alignment process. After estimating the bias corrections via SBA, the only remaining issue with regard to the alignment of the images is the alignment between tiles, which we will discuss in Section 4.2. For now, we now turn our attention to the issue of creating a tile-based DSM using the aligned image patches.

3.4 Creating a Tile-Based DSM

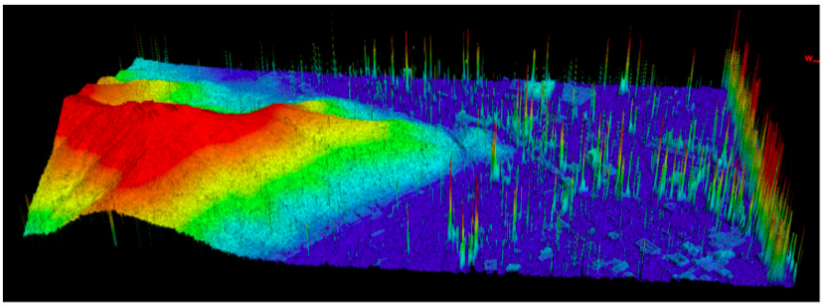
3.4.1 Pair Selection and Stereo Rectification

Stereo processing is carried out in a pairwise manner. Since using an exhaustive combination of pairs of image patches is computationally infeasible, as a first step, pairs of image patches are selected based on heuristics similar to those used by the work in [41]. These heuristics include the difference between the view angles, the difference between the solar elevation angles, and the difference between the times of acquisition of the two image patches that make up a pair. In addition, image patches are selected to cover as wide an azimuth-angle distribution as possible. We err on the side of caution and select a minimum of 40 and up to a maximum of 80 pairs per tile. Note that this number could be further reduced, for instance, by training a classifier on the pairwise sun-angle differences as suggested by the approach presented in [55].

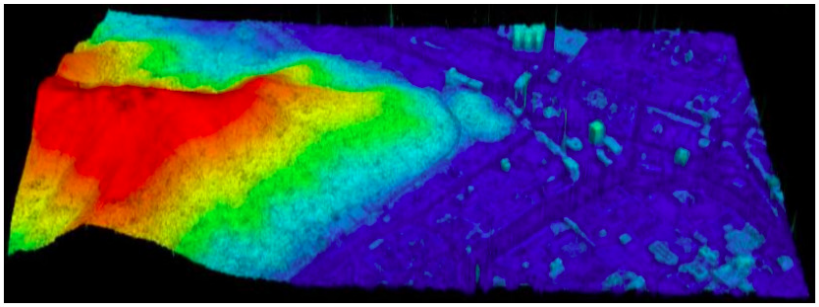
The next step is stereo rectification. In stereo rectification, planar homographies are applied to the two image patches such that the pixel $x_s = [c_s, r_s]^T$ in the second image patch that corresponds to the pixel $x_f = [c_f, r_f]^T$ in the first image patch has the same row coordinate as x_f. In other words, $r_s = r_f$. This makes it much easier to search for correspondences between the pixels of the two image patches. Note that a pushbroom sensor is better approximated by an affine camera model rather than by a pinhole camera model, and thus, the stereo-rectification algorithm has to be designed accordingly. In our study, the image patches are rectified using the approach described in [56].

3.4.2 Stereo Matching

For stereo matching, we use the hierarchical tSGM algorithm [57] with some enhancements to improve both the matching accuracy and speed. We modify the penalty parameters in the matching cost function as described in [58]. We noticed that this improves the accuracy near the edges of elevated structures. The second



(a)



(b)

Fig. 3.6.: At top is a fused DSM for a 2 km^2 AOI in Kabul, Afghanistan, reconstructed without using "DEM-Sculpting". At bottom is the fused DSM for the same area reconstructed with the help of "DEM-Sculpting". The DSMs have been colored based on the elevation values. The improvements in the DSM quality clearly show the power of "DEM-Sculpting".

enhancement is to use the Shuttle Radar Topography Mission (SRTM) Digital Elevation Model (DEM) [11] that provides terrain-elevation information at a coarse GSD of 30 m. Note that this DEM does not contain the heights of buildings and trees. We use the DEM to better initialize the disparity search range for every point in the disparity volume through a novel procedure that we refer to as "DEM-Sculpting" [59]. This improves accuracy and speeds up stereo matching. Additionally we use a guided bilateral filter for post-processing. With these additions, the matching algorithm is able to handle varying landscapes across a large area. The tSGM algorithm and these improvements are explained in greater detail in our contribution in [59]. In Fig. 3.6, we provide qualitative evidence for the power of "DEM-Sculpting".

3.4.3 Pairwise Point Cloud Creation and Fusion

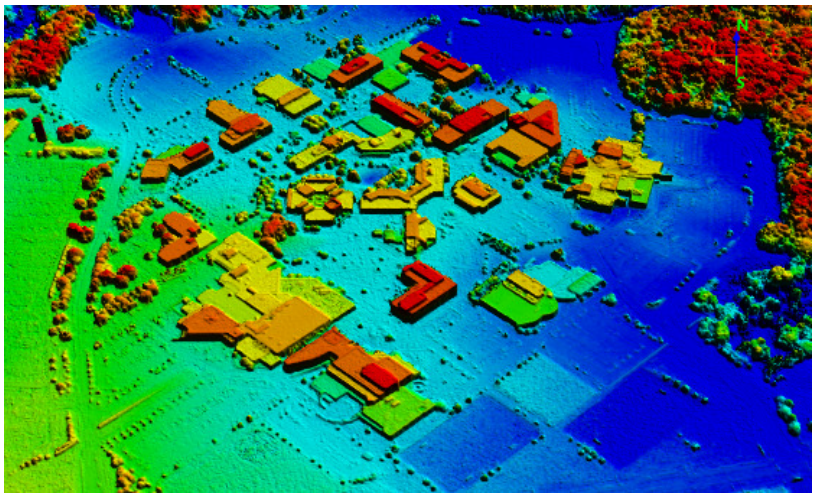


Fig. 3.7.: A fused DSM for a portion of a tile from Ohio, USA, as reconstructed by the framework described in this chapter. The DSM has been colored based on the elevation values.

The image patches, disparity maps and corrected RPCs are then used to construct pairwise point clouds. Since the image patches have already been aligned, the corresponding point clouds are also aligned and can be fused without any further 3D alignment. At each grid point in a tile, the median of the top Y values is retained as

the height at that point, where Y is an empirically chosen parameter. Subsequently, median filtering and morphological-and-boundary-based hole-filling techniques are applied.[3] An example of a fused DSM for a portion of a tile in Ohio, USA, as constructed by the framework described above is displayed in Fig. 3.7.

[3]The particular version of the pair selection and rectification module as used in our framework was developed by ETH Zurich [60] in conjunction with Applied Research Associates (ARA) [61]. The point cloud generation and the DSM fusion modules as used in our framework were developed by John Papadakis from ARA for the IARPA CORE3D program [17]. The stereo-matching algorithm was developed by Sonali Patil and Bharath Comandur.

4. CONSTRUCTING LARGE AREA DSMS

In the previous chapter, we have proposed that DSMs be constructed in a tilewise manner. In this chapter, we first justify why this is a valid and necessary assumption. We then discuss issues pertaining to merging the tile-level DSMs into a single DSM for the entire large area (100 km^2). Such a discussion is a unique contribution of this study since most of the literature on DSM construction from satellite images are carried out over small areas and therefore do not encounter the issues discussed in this chapter.

4.1 Motivation for Tile-Based Processing

Tile-based processing is made necessary by the following three considerations:

- The corrections to the camera models (calculated via high-precision alignment of the images with one another) cannot be assumed to be constant (or to vary linearly) across an entire large-area satellite image. We will support this claim with examples in Section 4.1.1.

- As part of the image-alignment process, the correspondences for a given pair of images are established initially by comparing the descriptor vectors associated with the keypoints and subsequently refined through outlier rejection with, say, the RANSAC algorithm [51]. The computational effort required for extracting the tie points goes up quadratically as the size of the images increases since the descriptors of the keypoints must be compared across larger images.

- The run-time memory requirements of modern stereo-matching algorithms, such as those based on semi-global matching (SGM) [62], can become much too onerous for full-sized satellite images. SGM requires the creation of disparity

volumes which can easily exceed the memory capabilities of the computing system.

4.1.1 An Example to Illustrate the Need for Tile-Based Bias Corrections

Our discussion in Chapter 3 regarding the alignment of the image patches that cover a single tile was based on the seminal work by Grodecki and Dial [48] that showed that the residual errors in the RPCs on account of small uncertainties in the measurements related to the position and the attitude of a satellite can be corrected by adding a *constant bias* to the projected pixel coordinates of the ground points, provided the area of interest on the ground is not too large. We refer to this as the *constant bias assumption* for satellite image alignment. Since keypoint extraction and descriptor matching are computationally expensive for large images (with a billion pixels), an astute reader would conclude that if the constant bias assumption was true, one could simply align the image patches that cover a single tile and then use the same bias corrections for the full-sized images.

While the study in [48] showed that the constant bias assumption assumption holds true for Ikonos [63] images whose length is shorter than 50 kms, it also claimed that a simple linear model using six parameters is sufficient to correct the RPCs for Ikonos images with lengths > 50 kms. Most studies in the literature that use WV3 images also stick to the constant bias assumption since they focus on relatively small areas covering a few km^2. Even the study on large-area DSM construction in [33] showed that both the constant bias assumption and the linear model approach yield roughly equivalent absolute geolocalization accuracies for images captured by the Pléiades satellite. However, note that the study in [33] uses pairs or triplets of images typically captured just seconds or minutes apart by the Pléiades satellite.

The only public contribution that we are aware of that talks about non-linear variations in the corrections for the RPC offset parameters is the study described in [64]. In that study, the authors claim that non-linear distortions in the RPC

model cannot be accounted for by using constant bias corrections and propose to first apply plane rectification to the full-sized images before carrying out alignment. However, they evaluate their algorithm using only a few images and do not include any evaluation of the constant bias assumption. Moreover, they only focus on image alignment and do not discuss the impact of their alignment algorithm on the quality of the point clouds/DSMs constructed from the images.

To properly study the validity of the constant bias assumption for full-sized WV3 images, we conduct the following two experiments:

1. In the first experiment, we first align all the image patches that cover a tile (denoted as tile A). We then apply these bias corrections to the corresponding image patches that cover a different tile (denoted as tile B). Tile B is chosen such that it is a fair distance away from tile A. We then select two image patches from tile B and construct a pairwise point cloud. Thus, this point cloud is constructed using the constant bias assumption, i.e., the assumption that the bias corrections are identical for the corresponding image patches from tile A and tile B.

2. For the second experiment, we apply the alignment algorithm directly to all the image patches that cover tile B. We then construct a pairwise point cloud using the same pair of image patches that was chosen in the first experiment. Thus, this point cloud is constructed using tile-based bias corrections.

In Fig. 4.1, we compare the ortho views of a section of the two point clouds constructed using the experiments described above. **We notice significant degradation in the quality of the point cloud that is constructed using the constant bias assumption. In the following section, we provide examples to show that the constant bias assumption holds true for some but not all full-sized WV3 images.**. This becomes patently obvious when constructing DSMs with a GSD of 0.3 m, where even a 1-2 pixel difference in the bias parameters can cause noticeable degradation in the point-cloud quality. Since there is no direct way to determine if

Fig. 4.1.: An example to show why one cannot use a constant bias correction for a full-sized image. At top is the ortho view of a portion of a pairwise point cloud for the constant bias assumption. At bottom is the same for tile-based bias corrections. The points have been colored using the color from the images.

the constant bias assumption holds for a specific image, it becomes necessary to allow for varying bias corrections for all the images.

4.1.2 Non-Linear Variation of the Bias Corrections Across the Full-Sized Images

Now that we have established that the corrections to the RPC offset parameters can vary across the full-sized images, it is natural to wonder about the pattern of this variation. For instance, if the bias corrections were to vary linearly across the image, then we could model this linear variation using six parameters as demonstrated by the studies in [48] and [33]. To understand the nature of this variation, we divided a large region in California, USA into 140 tiles, and then independently applied the alignment procedure (outlined in the previous chapter) to the image patches covering each tile.

Closely following the notation established in Section 3.3.6, let $\mathrm{I}_{\mathrm{k_p}}$ denote the p^{th} image patch of image $\mathrm{I_k}$. For this patch $\mathrm{I}_{\mathrm{k_p}}$, $\hat{l}_{\mathrm{offset}_{k_p}}$ and $\hat{s}_{\mathrm{offset}_{k_p}}$ are respectively the line and sample offset parameters that are calculated using the alignment procedure. $l^o_{\mathrm{offset}_{k_p}}$ and $s^o_{\mathrm{offset}_{k_p}}$ are respectively the uncorrected line and sample offset parameters for $\mathrm{I}_{\mathrm{k_p}}$, that are provided by the satellite vendors as part of the RPCs. We can write the calculated bias corrections as a vector in 2D space as – $[\hat{s}_{\mathrm{offset}_{k_p}} - s^o_{\mathrm{offset}_{k_p}}, \hat{l}_{\mathrm{offset}_{k_p}} - l^o_{\mathrm{offset}_{k_p}}]^T$.

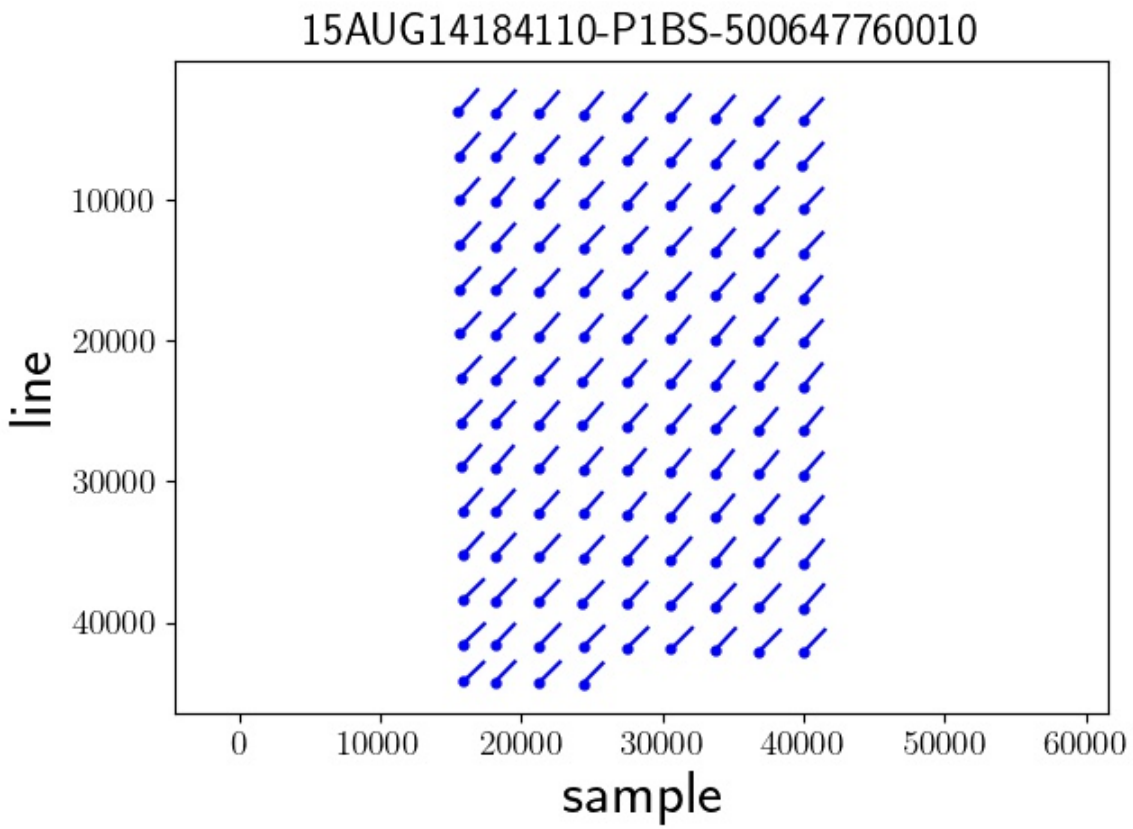


Fig. 4.2.: Tile-level bias corrections for image I_1 from California. The image name is displayed at the top. Each blue circle indicates the center of a tile. The corresponding bias corrections are drawn as 2D vectors with their origin being the corresponding tile centers. The bias corrections have been scaled for display purposes.

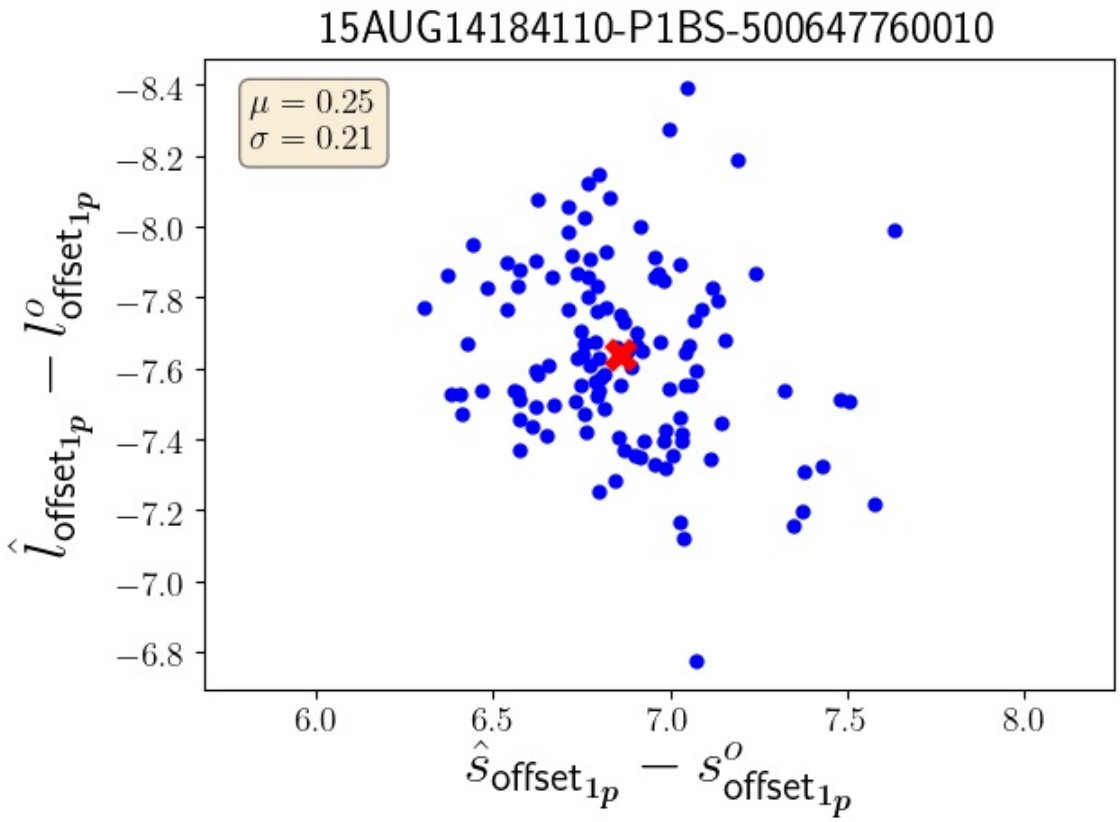


Fig. 4.3.: Scatter plot of the tile-level bias corrections for image I_1 from California. The image name is displayed at the top. The red X indicates the center of the cluster. μ and σ are respectively the average and the standard deviation of the Euclidean distances of the tile-level bias corrections from the cluster center.

Considering a specific image I_k, we create two different graphics to study the variation in the calculated bias corrections across the full-sized image. For the first graphic, we scale the bias corrections for each image patch of I_k by a factor of 200, and then plot the scaled vectors in 2D space. The origin of each vector is the center of its corresponding image patch. The second graphic is a scatter plot of the bias corrections for all the image patches of I_k. Additionally, we calculate the mean vector of the 2D bias corrections across all the image patches of I_k and subsequently calculate the mean and the standard deviation of the Euclidean distances of all the patch-level (tile-level) bias corrections from this mean vector. These are respectively denoted as

μ and σ in the scatter plots in Figs. 4.3 and 4.5. μ and σ give us a measure of the variation in the bias corrections across the full-sized image.

Figs. 4.2 and 4.3 show the two graphics described above for image I_1. From Figs. 4.2 and 4.3, we see that for I_1, all the tile-level bias corrections are almost equal to one another. As shown in Fig. 4.3, the tile-level bias corrections form a tight cluster in 2D space. Additionally, $\mu = 0.25$ and $\sigma = 0.21$ for I_1 indicating that the constant bias assumption holds well for I_1.

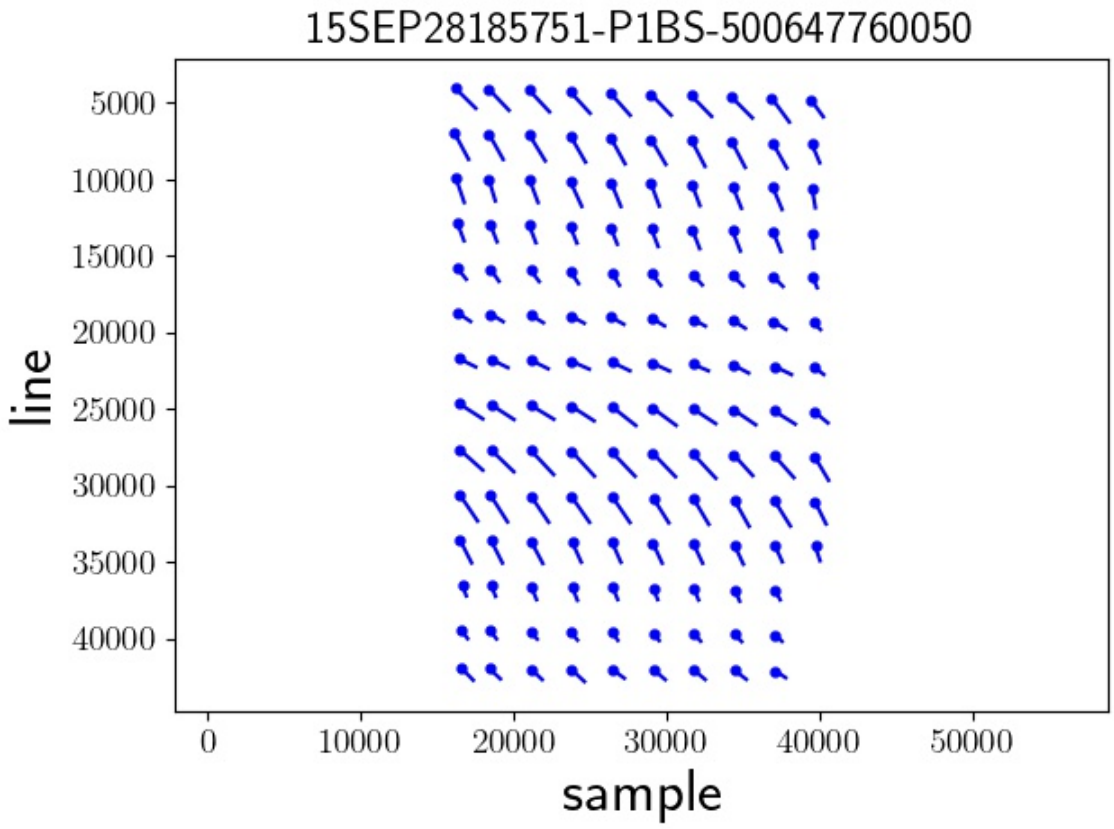


Fig. 4.4.: Tile-level bias corrections for image I_2 from California. The image name is displayed at the top. Each blue circle indicates the center of a tile. The corresponding bias corrections are drawn as 2D vectors with their origin being the corresponding tile centers. The bias corrections have been scaled for display purposes.

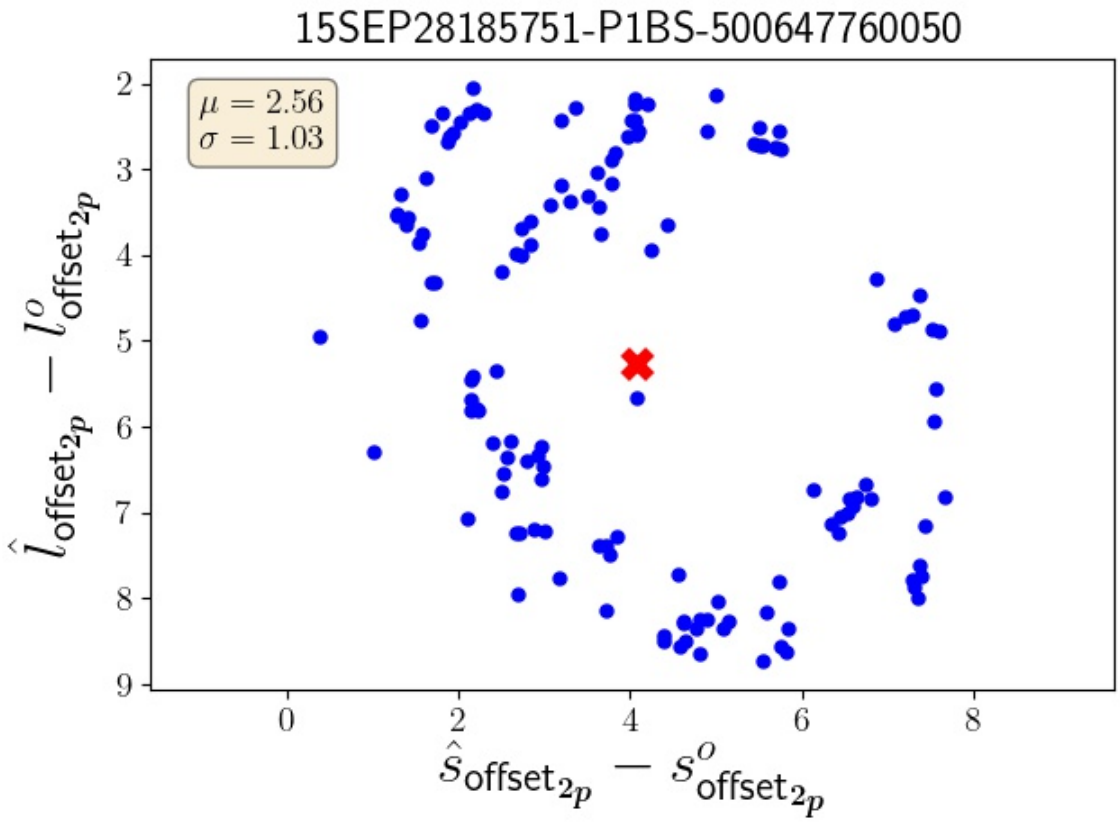


Fig. 4.5.: Scatter plot of the tile-level bias corrections for image I_2 from California. The image name is displayed at the top. The red X indicates the center of the cluster. μ and σ are respectively the average and the standard deviation of the Euclidean distances of the tile-level bias corrections from the cluster center.

Figs. 4.4 and 4.5 show the corresponding graphics for another image I_2. As shown in Fig. 4.5, the tile-level bias corrections are relatively spread out, and $\mu = 2.56$ and $\sigma = 1.03$ indicating that the constant bias assumption fails for I_2. **From Fig. 4.4, we observe that for I_2, the bias corrections not only vary across tiles but also change in a non-linear fashion. Therefore, a simple linear function cannot be used to model the variations in the RPC bias corrections.**

4.1.3 Tiling Strategy

Based on the preceding observations, we adopt a strategy different to the approach in [64] for large-area image alignment. We independently align the image patches for each tile. Now we are faced with the following question – *"if the satellite images must be broken into patches, how small or large should the individual patches be?"* The answer to that depends on two competing constraints. In practice, one would want the tiles to be as large as possible keeping in mind the constraints mentioned in Section 4.1. Making the tiles large reduces the duplicate calculations in the overlap regions and also reduces the need for reconciling the differences between the adjoining tiles in the overlap regions. At the same time, if a tile is too large, the constant bias assumption might not hold across the tile. Via experimental analysis, we have verified that the constant bias assumption is valid for WV3 image patches of size 5300×5300 pixels. This translates to a tile of area between 2-3 km^2. Based on our experience with WV3 images, we divide the geographic area into overlapping tiles where each tile consists of a central 1 km^2 main area and a padding of 300 m on each side. This makes for a total area of 2.56 km^2 for each tile. It also means that each tile (excluding the border tiles) has an overlap of area 1.6 km $\times$ 0.6 km = 0.96 km^2 with each of its four adjoining tiles. These numbers were chosen keeping in mind the memory requirements of the dense stereo-matching algorithm as well. Note that in choosing these numbers, we have erred on the side of caution. They can be adjusted appropriately to satisfy specific computational-resource constraints. Additionally, one can take advantage of a cloud computing framework to carry out alignment for different tiles in parallel. More details can be found in Chapter 8.

4.2 Between-Tile Alignment

Since alignment is carried out independently for each tile, it is possible that the neighboring patches belonging to the same image are no longer well aligned with one another. Put differently, the DSMs of neighboring tiles might not be aligned well

with one another. In this section, we briefly discuss two possible strategies to resolve this issue. *For both of these strategies, we assume that the tiles can be aligned using a translation.* Our experience with WV3 images indicates that this is a reasonable assumption to make.

4.2.1 Aligning Tiles via DSM Alignment

Consider the simple case of aligning two tiles. Since the DSMs of adjacent tiles overlap, in this strategy, we can fix one of the DSMs and find a three-dimensional (ϕ_t, λ_t, h_t) shift that can be applied to the second DSM in order to bring it into alignment with the first DSM. ϕ_t, λ_t and h_t respectively represent the translations in the latitude, longitude and height coordinates. To find this shift, we could try to maximize the NCC (normalized cross correlation) between the two DSMs [41], or minimize the median of the height differences between the two DSMs [35].

After finding the (ϕ_t, λ_t, h_t) shift, we can translate it into bias corrections for the image patches of the second tile. To understand this procedure, consider a single world point W that lies within the geographic area spanned by the second tile. For a single image patch of the second tile, we can project W into it using the RPCs. Next, we can translate W by the calculated (ϕ_t, λ_t, h_t) shift to get a world point W$'$ and then project W$'$ into the same image patch. The 2D shift between the two projected pixels gives us an estimate of the bias correction for this image patch.

In practice, we can first randomly sample a 3D grid of points over the AOI spanned by the second tile, and then apply the above procedure to all these world points. The bias corrections for each image patch can be calculated as the average of the 2D shifts across the projected pixels. This procedure can be naturally extended to T tiles by incrementally aligning one new tile at a time.

4.2.2 Multi-Stage Bundle Adjustment

In this section, we present an alternative strategy for between-tile alignment. From Fig. 4.4, we can see that the bias corrections for the adjacent patches of the same image are similar, even if they change noticeably across the full-sized image. Our second strategy for between-tile alignment is based on this observation.

Let us first consider a simple case where we only have two adjacent tiles. Assume that in the first stage of alignment, the image patches of each tile have been aligned with one another independently. To align the image patches of these two tiles with one another, we try to find a 3D translation for each tile so as to bring them into alignment with each other. We use the tie points that were extracted for each tile during the first stage of alignment to define a loss function, that minimizes the differences between the bias corrections for the same image patch from the two tiles. The motivation behind this is that we expect the bias corrections for the same image patch from adjacent tiles to be close to each other. Note that in this case, we apply the same (ϕ_t, λ_t, h_t) shift to all the world points that are triangulated from the tie points that belong to a single tile. ϕ_t, λ_t and h_t respectively represent the translations in the latitude, longitude and height coordinates. Additionally, *we can use an L2 regularizer to ensure that the new bias corrections do not deviate significantly from the corrections that have already been computed in the first stage of alignment.* The reasoning behind this is that we trust the bias corrections produced in the first stage. We can then apply SBA to find (ϕ_t, λ_t, h_t) shifts for all the tiles, that minimize the new loss function. Subsequently, the calculated (ϕ_t, λ_t, h_t) shift for each tile are used to compute the bias corrections for all the image patches of that tile, using the procedure described in the previous strategy.

In order to extend this logic to T tiles, we have two choices of approaches. In the first approach, we can define a loss function using the tie points from all the T tiles simultaneously. However, this can increase the computational complexity significantly. An alternative approach would be to carry out multiple stages of alignment

where new tiles are incrementally added in each successive stage. For instance, we can first align the image patches of two adjacent tiles T_A and T_B with one another. In the next stage, we can select two tiles T_C and T_D that are respectively adjacent to T_A and T_B and align them to T_A and T_B, and so on.

4.3 Merging Tile-Level DSMs

It turns out that the strategies for between-tile alignment that were proposed in the previous section are not really required for our application. In the experiments that we have run in Ohio and California, our experience so far has been that the addition of the regularization term (Eq. 3.2) to the bundle-adjustment logic for each tile does such a good job in each tile so as to render unnecessary any need for additional corrections for between-tile alignment, given the noise in the OSM and the residual errors in the DSM. We discard the overlapping portions between the neighboring DSMs and create one large-area DSM. We believe that the residual noise from errors in stereo-matching and triangulation drown out the noise due to tiny differences in the bias corrections between neighboring tiles. We will confirm this in Section 9.2.2 where we have included results on the evaluation of the alignment algorithms.

5. SINGLE-VIEW DEEP LEARNING FOR SEMANTIC SEGMENTATION

This chapter is divided into two halves. The first half describes two different approaches for using the off-nadir satellite images, the DSM and the OSM data to prepare training, validation and inference data samples for a convolutional neural network (CNN). The second half focuses on how to train, validate and test these data samples with a single-view CNN. By a "single-view CNN" we mean that even if images taken from multiple viewpoints of the same scene are available, they are treated as independent images while training the CNN. Recall from Chapter 1 that one of our primary goals is to combine information from multiple views of the same scene while training the CNN. Before we can explain our proposed multi-view training paradigm and CNN architecture, it is necessary to understand the basics of semantic segmentation using single-view CNNs, and we will cover this in the second half of this chapter where we opt to use the popular U-Net [65] for establishing the baseline for single-view semantic segmentation.

During training, a semantic-segmentation CNN usually requires a pair of inputs – an image and its corresponding groundtruth label map which contains a label for each pixel in the image. To create such an input pair, we are faced with two different choices of approaches. In the first approach, we can orthorectify the off-nadir images and then convert the OSM vector labels into per-point raster labels in the orthorectified space. Alternatively, in the second approach, we could project the OSM vectors into the off-nadir image pixel coordinates, thereby creating the input pair in the off-nadir space. *Most of the studies in the literature that use OSM labels do not have the ability to implement the second approach because they directly use pre-orthorectified single-view images and/or do not use DSMs/LiDAR. However, our approach possesses all the*

required elements to carry out such a study. We describe both these approaches in the following sections.

5.1 Creating Groundtruth Data in the Orthorectified Space

5.1.1 Pansharpening

As a first step, we pansharpen the full-sized images. Recall from Section 2.3 that pansharpening is a procedure that enables us to assign the multispectral values from the lower resolution multispectral images to the higher resolution panchromatic images (meaning grayscale images). We do this using the open-source "gdal_pansharpen.py" utility [26]. Another possible choice of software is the Orfeo ToolBox [25]. Note that the RPC bias corrections that are calculated for the panchromatic images can be used for the pansharpened images.

5.1.2 True Orthorectification Using gwarp++

We can orthorectify the pansharpened images using the fused DSM as the elevation map. Recall from Section 2.5 that orthorectification is the process of mapping the pixel values in the images to their corresponding points in the geographic area of interest. We had remarked in Section 2.5.1 that there are no open-source utilities to create true ortho images using RPCs and DSMs at this time. Therefore, we have developed a utility, which we have named "gwarp++", to create full-sized true-ortho images quickly and efficiently. We now provide a brief overview of "gwarp++".

Let us first consider the case of orthorectifying an image patch (that belongs to a single tile) with the help of a DSM. We have already explained the procedure for carrying out orthorectification using a DEM in Section 2.5. If this procedure is used as is with a DSM, it creates "ghosts" as shown in Fig. 2.5c. "gwarp++" builds upon this procedure to detect these ghosts.

Algorithm 5.1: gwarp++

ϕ – Latitude, λ – Longitude, h – height

ϕ_{step} – Latitude step size, λ_{step} – Longitude step size, h_{step} – Height step size

$\text{Image}_{\text{Patch}}$ – Off-Nadir image patch

L – Length of $\text{Image}_{\text{Patch}}$ in pixels, W – Width of $\text{Image}_{\text{Patch}}$ in pixels

$LT \leftarrow Zeros(W, L)$ `// Initialize lookup table to zeros`

OutArray – Output array for the orthorectified grid

Step 1: Find extents of the AOI spanned by the image patch

$(\phi_{\min}, \lambda_{\max})$ – Top-left corner of the AOI

$(\phi_{\max}, \lambda_{\min})$ – Bottom-right corner of the AOI

Step 2: Project points into $\text{Image}_{\text{Patch}}$ and update LT

for $\phi = \phi_{min}$; $\phi \leq \phi_{max}$; $\phi = \phi + \phi_{step}$ **do**
  for $\lambda = \lambda_{max}$; $\lambda \geq \lambda_{min}$; $\lambda = \lambda - \lambda_{step}$ **do**
    $h \leftarrow \text{DSM}(\phi, \lambda)$
    $h_{\text{ground}} \leftarrow \text{DEM}(\phi, \lambda)$
    for $h' = h$; $h' \geq h_{ground}$; $h' = h' - h_{step}$ **do**
      $(s, l) \leftarrow \text{Proj}_{\text{RPC}}(\phi, \lambda, h')$
      if $LT(s, l) < h'$ **then**
        $LT(s, l) \leftarrow h'$
      end
    end
  end
end

Step 3: Create OutArray with a second pass over the grid

for $\phi = \phi_{min}$; $\phi \leq \phi_{max}$; $\phi = \phi + \phi_{step}$ **do**
  for $\lambda = \lambda_{max}$; $\lambda \geq \lambda_{min}$; $\lambda = \lambda - \lambda_{step}$ **do**
    $h \leftarrow \text{DSM}(\phi, \lambda)$
    $(s, l) \leftarrow \text{Proj}_{\text{RPC}}(\phi, \lambda, h)$
    if $LT(s, l) > h + \gamma$ **then**
      $\text{OutArray}(\phi, \lambda) \leftarrow \text{NODATA}$
    else
      $\text{OutArray}(\phi, \lambda) \leftarrow \text{Image}_{\text{Patch}}(s, l)$ `// Can also interpolate values`
    end
  end
end

Consider two points $W_1 = (\phi_1, \lambda_1, h_1)$ and $W_2 = (\phi_2, \lambda_2, h_2)$ that both project to the same pixel coordinates in the image patch. ϕ, λ and h denote the latitude, longitude and height coordinates respectively. If $h_2 > h_1$, it means that W_1 is occluded by W_2. This is the core idea that "gwarp++" uses to detect "ghosts".

Now, consider a single world point $W = (\phi, \lambda, h)$, where h is the height value from the DSM. Let h_{ground} be the corresponding height value in the DEM. The DEM gives us a rough estimate of the height of the ground. It is possible to use more sophisticated techniques, such as the one described by the study in [66], to directly estimate the elevation of the ground from the DSM. The DEM is sufficient for our application. Instead of just projecting W into the image patch, "gwarp++" projects a set of points $\mathcal{W}' = \left\{(\phi, \lambda, h')\right\} \; \forall \; h' \; \epsilon \; [\; h,\; h - h_{\text{step}},\; h - 2 \cdot h_{\text{step}}, \ldots \;,\; h_{\text{ground}} \;]$ where h_{step} is a user-defined step size. $\mathcal{W}'$ is therefore a set of points sampled along the vertical line from W to the ground. To understand the motivation for doing this, it might help to consider the case when W is the corner of a building. In that case, $\mathcal{W}'$ is the set of points along the corresponding vertical building edge from W to the ground. If we apply this procedure to all the points on the roof of a building, we will end up projecting the entire building into the image patch.

We now describe the implementation of "gwarp++" below. The algorithm is summarized in Algorithm 5.1.

1. "gwarp++" starts out by dividing the AOI into a 2D grid of world points. The grid is 2D in the sense that only the longitude and the latitude coordinates are considered. Refer to Section 2.5 to understand how the extents of this grid are determined. The distance between the points of this grid is a user-defined parameter.

2. Using the height values from the DSM, for each point in the grid, "gwarp++" projects a set of points into the image patch as explained above. For each pixel in the image patch, a lookup table "*LT*" stores the maximum height, with the

maximum being computed across all the points that project into this pixel. This procedure is repeated for all the points in the 2D grid.

3. At this stage, for each point J in the 2D grid, we know three things:

   - h_J – The DSM height value at J
   - (s, l) – The pixel into which J projects after assigning J an elevation value of h_J
   - $LT(s, l)$ – The maximum height of a world point that projects into (s, l)

   If $LT(s, l) > h_J$, we can conclude that J is occluded by some other world point that has a height value of $LT(s, l)$.

4. Therefore, using a second pass over all the points of the 2D grid, "gwarp++" marks the "ghost" points with a "NODATA" value. In practice, to account for quantization errors and the noise in the DSM, "gwarp++" checks if $LT(s, l) > h_J + \gamma$ where γ is chosen appropriately.

To orthorectify the full-sized image, we orthorectify each image patch using its corrected RPCs and the large-area DSM. The orthorectified image patches are then mosaiced into a full-sized orthorectified image during which the overlapping portions between the image patches are discarded.

"gwarp++" is written in C++. It has the nice property of being massively parallel since the projection for each point can be carried out independently and since each tile can be processed independently. This parallelism is exploited at both stages. For each image patch, OpenMP [67] is used to process the points in parallel. And the different image patches are themselves orthorectified in parallel by different virtual machines running on a cloud-based framework.

For our application, each full-sized orthorectified image is resampled at a GSD of 0.5 m. Furthermore, the occluded points are delineated with a mask that is subsequently used during training of the CNN to prevent gradients at those points from being backpropagated.

5.1.2.1 Accuracy of "gwarp++"

We conclude our discussion on true orthorectification with a few remarks on the accuracy of the orthorectified images produced by "gwarp++".

3D vs 2.5D: For each point W, "gwarp++" considers points along the vertical line from W to the ground. This is not a good strategy for buildings that possess more exotic shapes such as spherical water towers or buildings with walls that slope inwards. In these cases "gwarp++" can incorrectly mark some points as occluded points. The only way to handle such cases is by using a 3D point cloud instead of a 2.5D DSM, which is beyond the scope of our discussion.

Error Propagation: Errors in the RPCs and errors in the DSM will translate into errors in the orthorectified images. However, in our application, these errors are largely drowned out by the errors in the OSM labels. Nevertheless, it might be useful to study how these errors propagate, which we leave for future work.

5.1.3 Aligning OSM with the Orthorectified Images

Fig. 5.1.: This figure shows typical results obtained by aligning the orthorectified images with OSM. What is shown in red at left are the unaligned OSM vectors, and what is in blue at right are the aligned versions of the same.

In Section 2.6.1, we have provided examples of the errors in the OSM data. Errors such as missing and incorrect annotations are relatively harder to resolve. However,

using some simple techniques, it is possible to reduce the extent of misalignment between the OSM data and the orthorectified images. Our framework incorporates the following two strategies to align the OSM data with the orthorectified images:

1. Using Buildings: First, the system subtracts the DEM from the constructed DSM to extract building footprints. Subsequently, these building footprints are used to align the orthorectified images with the OSM using Normalized Cross Correlation (NCC). This strategy has proved useful in areas with inadequate OSM road labels.

2. Using Roads: First, the system uses the "Red Edge" and "Coastal" bands to calculate the Non-Homogeneous Feature Difference (NHFD) [68], [5] for each point in the orthorectified image and subsequently applies thresholds to the NHFD values to detect the roads. The NHFD is calculated using the formula:

$$\text{NHFD} = \frac{(\text{Red Edge - Coastal})}{(\text{Red Edge + Coastal})} \tag{5.1}$$

   Subsequently, the roads (noisy obviously) are aligned with the OSM roads using NCC. The system uses this strategy in rural areas that may not contain the buildings needed for the previous approach to work.

After alignment, the OSM vectors are converted to raster format with the same resolution as in the orthorectified images. Thus there is a label for each geographic point in the orthorectified images. The OSM roads are thickened to have a constant width of 8 m.

Fig. 5.1 shows an example of misaligned and aligned OSM vectors. It should be noted that some residual alignment error does persist. We plan to improve this module by aligning each building/road separately.

5.1.4 Generating Orthorectified Data Samples

Minimal human supervision and the sparse and noisy nature of OSM labels necessitate important design choices for creating training data samples which is the focus of this section.

The training data is generated by using an $F \times F$ window to randomly sample each orthorectified image. The parameter F is a design parameter and is currently set to 572 for the orthorectified images. We refer to each $F \times F$ block of pixels in the image as an image-window and the corresponding block of labeled points in the OSM as a label-window. Fig. 5.2 shows an example of an image-window and its associated label-window in the orthorectified space.

5.1.4.1 Efficient Selection of Single-View Training Samples in the Presence of Noise

Adopting a sliding-window approach is not efficient when extracting image-windows and label-windows from a 100 km^2 region as it will produce a lot of redundant win-

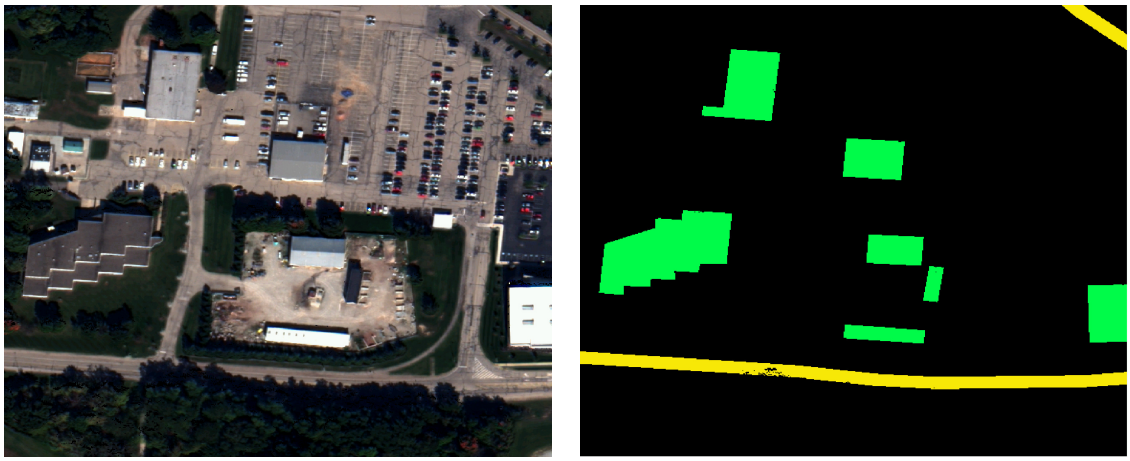


Fig. 5.2.: Example of an orthorectified image-window and label-window pair. The 572×572 array at left was extracted from an orthorectified image. At right are the OSM labels for the points. Roads are marked in yellow and buildings are marked in green.

dows thereby slowing down the training process. Additionally, OSM labels are often sparsely annotated in rural and residential areas. We include some additional logic into our data sampler to handle these problems. In order to eliminate redundancies in the training data, as the system randomly samples the orthorectified images with $F \times F$ windows, it keeps track of such windows already visited in an image through the expedient of setting the bits in a mask array. Should a randomly generated window have more than 30% overlap with the portions of an image already visited, that random selection is discarded. Another reason for discarding a random window selection would be if it contained an insufficient percentage of pixels with the desired OSM labels. These ploys considerably reduce the training time. Without the strategy mentioned here, the training time may be as long as 50 epochs [69]. With the help of these strategies our network converges in 12 epochs.

5.2 Creating Groundtruth Data in the Off-Nadir Space

5.2.1 Projecting OSM into the Off-Nadir Space

Until now, our discussion has focused on using true ortho images for semantic segmentation. This approach does have one disadvantage. Assume for a minute that we are tasked with segmenting a new WV3 image using a CNN that has been trained on orthorectified images. We would need to align this new image to the images that we already possess and then orthorectify it before we can use the trained CNN to segment it.

More generally, it would be useful to directly train CNNs on the off-nadir images for many applications. Even for the task of labeling world points, it would be interesting to compare the strategy of training CNNs on the orthorectified images vis-à-vis first training CNNs on the off-nadir images and subsequently orthorectifying the predicted labels. However, this would require a way to project the OSM training labels from geographic coordinates into the off-nadir images. Most prior OSM-based studies in the literature are ill-equipped to carry out such a comparison because they

use pre-orthorectified images. On the other hand, our end-to-end automated framework which includes the ability to create large-area DSMs allows us to easily achieve this. DSMs are crucial to this exercise because we need the elevation information to use the RPCs for translating the OSM labels from their geographic coordinates into their pixel coordinates in the off-nadir images.

At first glance, it looks like creating labels in the off-nadir space is computationally easier than creating orthorectifed images because the RPC equations give us a direct mapping from each world point to its pixel coordinates (Eq. 2.3). A naive approach to create the off-nadir label maps would be to project the rasterized OSM labels (Section 5.1.3) into the off-nadir images in a point-by-point fashion. However, such an approach will create a lot of holes in the projected label maps.

To create smooth and filled label maps in the off-nadir space, we have two choices –

1. For each pixel in the off-nadir image, we can iteratively find the world point that projects to it. We have already described a procedure to do this in Section 2.5.2.

(or)

2. We can project the corners of the OSM polygons into the images and subsequently mark all the pixels inside the projected polygons as labels.

We choose to use the second approach because it is significantly faster than the first approach. However this speedup does come at a cost. Consider an example of projecting a polygon that represents a building roof into an off-nadir image. If the DSM height for a corner of this polygon is incorrect, then since we first project the vector data into the image and subsequently rasterize it, the projected shape of the label for the entire building roof could become distorted. In contrast, in orthorectification, if the height of a single point is incorrect then the spectral value assigned to that point in the orthorectified image would be incorrect. *Thus, the noise*

in the DSM has greater impact on the noise in the training labels when using off-nadir images vis-à-vis using true ortho images.

5.2.2 Generating Off-Nadir Data Samples

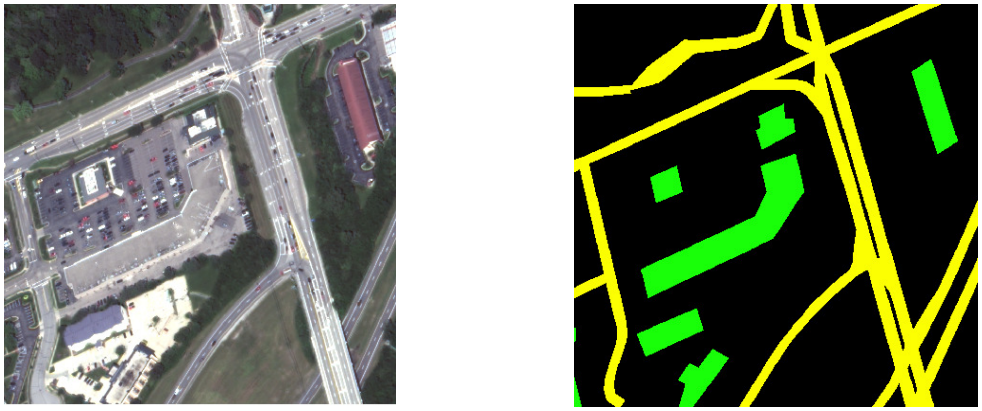


Fig. 5.3.: Example of an off-nadir image-window and its corresponding label-window. The 828 × 828 array at left was extracted from an off-nadir image. At right are the OSM labels for the pixels. Roads are marked in yellow and buildings are marked in green.

Once the OSM data has been projected into the off-nadir images, we employ the same strategies that we described in Section 5.1.4.1 to create non-redundant and useful off-nadir training windows, with one modification. Instead of extracting windows of size 572 × 572, we extract windows of size 828 × 828 from the off-nadir images. The reason for this modification is as follows.

For a fair comparison, we want to train the CNN(s) with input windows that capture a similar amount of spatial context around the roads and buildings, in the orthorectified and in the off-nadir space. The orthorectified images are created at a GSD of 0.5 m. The off-nadir images have an average GSD of 0.31 m. Therefore a

window of size 572×572 in the orthorectified space will correspond to a window of size

$$\frac{(572) \cdot (0.5)}{0.31} \times \frac{(572) \cdot (0.5)}{0.31} = 922 \times 922$$

in the off-nadir space. However, the U-Net architecture requires that the images that are input to it have sizes that satisfy some specific constraints [65]. Hence we choose a window size of 828×828 that satisfies these constraints and is comparable to a window size of 922×922. Fig. 5.3 shows an example of one such off-nadir image-window and its corresponding label-window.

We conclude this section with the following additional remarks. True orthorectification introduces "NODATA" or occluded portions into the images, assume that the reader is cognizant of the basics of deep learning. Despite the existence of a large volume of literature on semantic segmentation, a vast majority of these studies use single-view images and/or single-view CNNs. By a "single-view image", we mean that for each scene we only have a single image that has been captured from a single viewpoint. By a "single-view CNN", we mean that even if images taken from multiple viewpoints of the same scene are available, they are treated as independent images while training the CNN.

While there exist a multitude of single-view CNNs for semantic segmentation, all these networks share some common fundamental elements which we briefly discuss below. The rest of this chapter is organized as follows. We first review relevant literature on semantic segmentation. Subsequently, we describe the architecture of a CNN called the "U-Net" [65] which has proven quite popular for semantic segmentation. Then we briefly touch upon the loss function and the single-view training strategy that we use to train the U-Net. This will serve as a baseline for measuring the performance of single-view training.

5.3 Relevant Literature

A detailed review of modern methods for semantic segmentation can be found in the following studies – [70], [71], [72], [73], [74], [75] and [76]. Broadly speaking, the number of deep-learning based studies on semantic segmentation of traditional images, such as those used for self-driving applications or those captured with handheld cameras, outstrips the number of such studies on segmentation of aerial and satellite images. The primary reason for this is the discrepancy in the amount of precisely labeled groundtruth data that is available for these applications. However the recent years have witnessed good efforts to bridge this gap.

Using Precise Groundtruth Training Labels

Popular datasets for semantic labeling of overhead imagery with manually-generated and/or manually-corrected training labels have been created by the contributions described in [77], [1], [2], [3], [4], [78], [79], [80], [81] and [82]. The DeepGlobe dataset [77] provides satellite images and precise labels annotated by experts for road and building detection and land cover classification. The contribution presented in [2] provides satellite images, airborne LiDAR, and building labels (derived from LiDAR) that are manually corrected. The study described in [4] combines multi-view satellite imagery and large-area DSMs (obtained from commercial vendors) [3] with building labels that are initialized using LiDAR from the HSIP 133 cities data set [83]. The IEEE GRSS Data Fusion Contest dataset [78], [79] provides true ortho images, LiDAR and hyperspectral data along with precise groundtruth labels for 17 LCZs (local climate zones). A summary of the top-performing algorithms on this dataset can be found in [84]. The algorithms proposed by the recent studies in [85], [86], [87], [88], [89], [90], [91], [92], [93] and [94] are some examples of approaches that use datasets with precise training labels for semantic labeling of overhead imagery.

For aerial imagery, the popular ISPRS 2D labeling dataset [80] contains true ortho aerial images, DSMs and carefully annotated groundtruth for six class labels. Some examples of CNN-based approaches that use this dataset include the contributions presented by the studies in [95], [96], [97], [98], [99], and [100]. While the approaches described in [97], [98] and [99] discuss fusion strategies for features extracted from DSMs and images, the study in [100] supplements image features with features from OSM data. Other recent studies that focus on semantic labeling of aerial images include those described in [101], [102], [103], [104], [105], [106], [107], [108] and [109].

Using Noisy Groundtruth Training Labels from OSM

For aerial imagery, one of the first studies on large-area segmentation of roads and buildings using OSM labels was reported by the study in [110]. The building dataset and the road dataset used in this study span areas of 340 km^2 and 2600 km^2 respectively. However, with a GSD of 1 meter, this road dataset is effectively covered by a single full-sized orthorectified image of size 50000×50000 pixels. In addition, that work restricts the training data to regions with an omission-noise level of roughly 5% or less. In comparison, our study uses multiple full-sized and off-nadir images and we place no restrictions on the noise level in our training data. Some relevant contributions that use the dataset created by the work in [110] include the studies described in [69], [111], [112] and [113]. The work reported in [113] uses an ensemble of CNNs whereas the study reported in [69] uses residual units with a U-Net [65] to outperform the approaches described in [114] and [111] for road segmentation. The authors of the study described in [112] recommend a two-step training strategy that comprises of first training a classifier on noisy OSM labels and then fine-tuning it with an accurate hand-labeled dataset.

The work reported in [115] employs a variant of Fully Convolutional Networks [116] to evaluate the effectiveness of using OSM data for the segmentation of aerial imagery. It uses orthorectified Google Maps images that already align relatively well with the OSM data. It shows that the errors in the OSM data do affect the

segmentation accuracy and concludes that both large amounts of data and large-scale data-augmentation are required to compensate for the errors in OSM. Similar to the study described in [112], it also recommends pre-training the CNN with noisy OSM labels and subsequently using domain adaptation with accurate manual annotations as an optimal strategy. The contributions described in [117], [118] and [119] also train CNNs on orthorectified images and OSM labels. The work reported in [117] attempts to address the noise in OSM by using other sources of data whereas the study described in [118] manually corrects the incorrect labels. Some non-deep-learning contributions that use OSM labels for training include the approaches proposed in [120] and [121]. The study described in [120] focuses on enhancing the roads in OSM by using a Markov random field (MRF) and heuristic knowledge. The work reported in [121] takes a different approach and tries to filter out noisy training samples using contextual cues.

More recent state-of-the art approaches that demonstrate the use of labels derived from OSM for finding roads and/or buildings in overhead images include the studies described in [122], [123], [124], [125], [126], [127], [128], [129] and [130].

All of the approaches listed above use single-view images that are usually pre-orthorectified and/or single-view CNNs. A review of the existing multi-view approaches in the literature can be found in Section 6.1.

5.4 Encoder-Decoder CNN Architecture for Semantic Segmentation

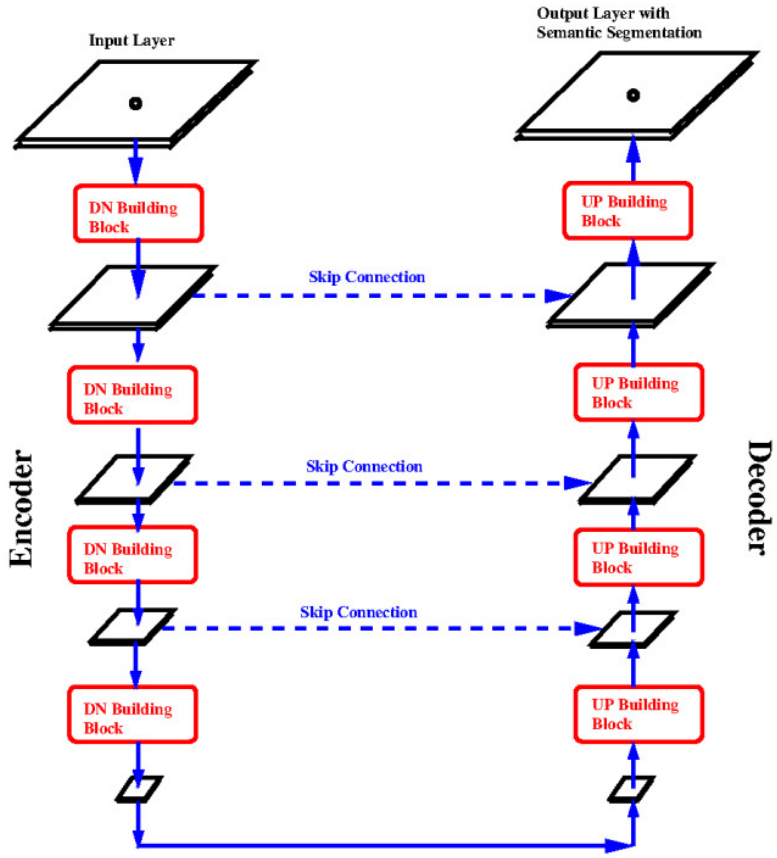


Fig. 5.4.: An example of an encoder-decoder CNN architecture for semantic segmentation. This figure has been reproduced from the lecture notes in [131].

Most of the popular CNNs for semantic segmentation possess what is popularly referred to as an encoder-decoder architecture. While the exact structure and the layers of the encoder and the decoder components can vary drastically between networks, the basic idea remains the same. The encoder typically creates a lower-dimensional representation of the image that captures important features and semantic information, and the decoder uses this encoded representation to label the pixels in the original image. Fig. 5.4 shows a simple example of an encoder-decoder style CNN. This figure has been reproduced from the lecture notes in [131].

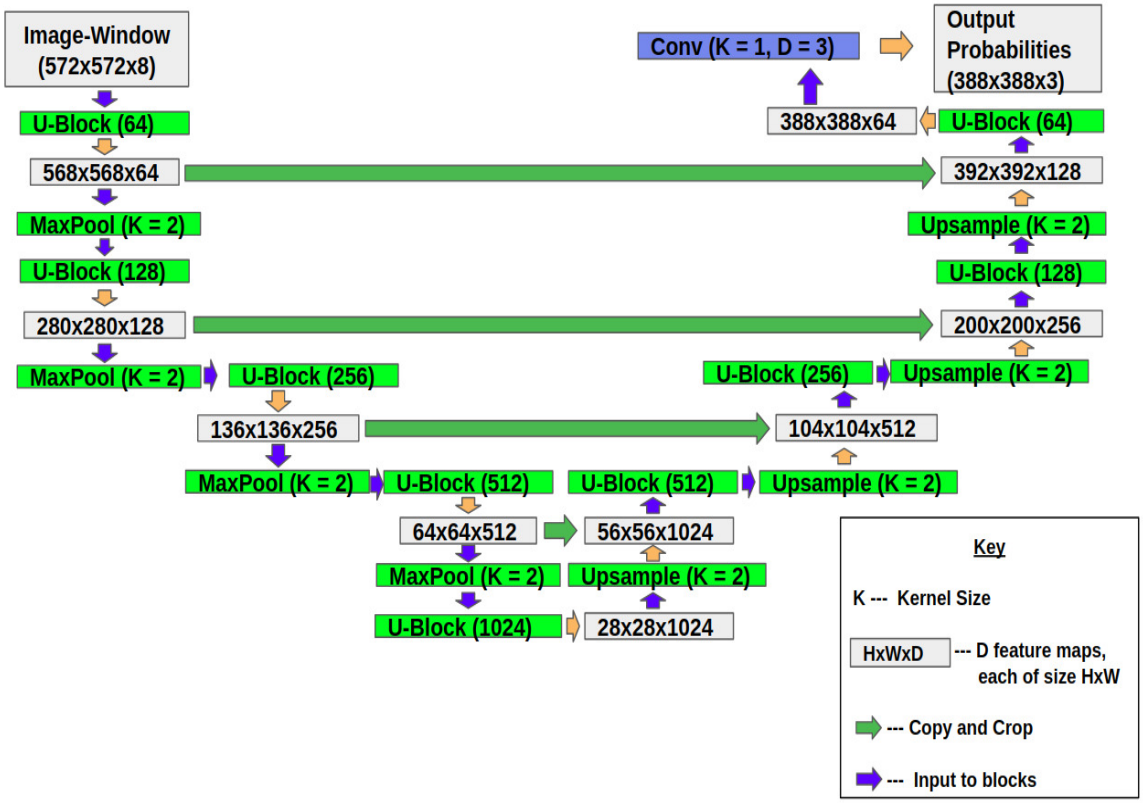


Fig. 5.6.: An inference view of the U-Net architecture as used in our framework for 8-band orthorectified satellite image-windows. In the training view of the same architecture, the output probabilities are compared with the one-hot representations of the label-windows. A detailed view of the U-Block is shown in Fig. 5.5.

5.5 U-Net for Multi-Spectral Images

Fig. 5.5.: Expanded view of a single U-Block. A block has kernel size K = 3, padding P = 0 and stride S = 1. It has D output channels and includes batch normalization.

We want to train a CNN completely from scratch on noisy data. For that purpose, we have chosen to use the popular U-Net [65] which can learn from relatively small amounts of training data. However, the original configuration of the U-Net has to be

changed slightly to suit the needs of the satellite data. Note that the original U-Net architecture was meant for grayscale biomedical images in which each pixel contains only a single scalar value. On the other hand, our satellite data consists of 8-band pansharpened images. Therefore, our U-Net must use 8 channels for the input. As for the output, since at this time, we are only interested in labeling the buildings and the roads, we have three channels, one channel each for the background, road and building classes.

This adaptation of the original U-Net to our needs is shown in Figs. 5.5 and 5.6, where we have shown 8 for the number of input channels that would correspond to the eight bands of the pansharpened image-windows. Our adaptation of the U-Net also includes a batch-normalization layer [132] added to each convolution block. Fig. 5.6 also depicts the three output layers for the probabilities associated with the three classes. As shown, all convolutional operators are of size 3×3, all downsampling is achieved with 2×2 maxpooling, and all upsampling is achieved with 2×2 convolutions. Please note that the values of the length and the width dimensions as indicated in Fig. 5.6 are for the case of using orthorectified image-windows. In Section 5.2.2 we have already explained that the length and the width dimensions will be different when we use the off-nadir image-windows and label-windows. All the other parameters of the CNN will remain the same.

5.6 Loss Function, Optimizer and Hyperparameters

The image-windows and label-windows are input to the CNN in batches. The size of the batch depends upon the available GPU memory. Let us denote the batch size as B. To create each batch we randomly select B windows out of all the available training windows. Random rotations and left-right flips are then applied as data-augmentation techniques to this batch which is then input to the CNN. The corresponding label-windows are used for calculating the loss. The loss for a single batch of windows is calculated as follows:

$$L_{\text{CNN}} = \frac{1}{B}\sum_{i=1}^{B} \text{CE}_i(G_i, O_i) \tag{5.2}$$

where CE_i is the weighted pointwise cross-entropy loss for the i^{th} window with the class weights set to 0.2, 0.4 and 0.4 for the background, road and building classes respectively. These class weights are chosen so as to help alleviate the class-imbalance in the noisy and sparse training labels. Otherwise the background class might dominate the gradient calculation and subsequent weight updates. O_i is the output tensor of the CNN for the i^{th} image-window. To calculate CE_i, we mask the i^{th} label-window with the occlusion (NODATA) mask of the i^{th} image-window. This masked ground-truth is denoted by G_i in the equation above. Note that this mask is implicitly computed during the process of true orthorectification. The gradients of L_{CNN} are not backpropagated at these masked points.

For training, we initialize the weights of the CNN randomly. Additionally, instead of using bilinear interpolation, we learn the weights of the upsampling filter in the decoder part of the U-Net. The learning rate is reduced gradually as the network converges. We use stochastic gradient descent with momentum, as the optimization routine.

In each epoch we use all the available windows. At the end of each epoch, we validate the U-Net on a set of windows obtained from a separate region that is specially marked for validation. We use the mean IoU (Intersection over Union) as the validation metric.

5.6.1 Inference

We save the learned weights of the U-Net for the epoch with the best mean IoU on the validation windows. We use this trained CNN to run inference on an "unseen" region that does not overlap with the geographic areas used for training and validation. We divide each image patch covering this inference region into an

overlapping grid of windows. The predictions for the windows are then merged with majority voting. This gives us one large set of predictions per image patch. If these predictions are in the off-nadir space, they are subsequently orthorectified using "gwarp++" (Section 5.1.2).

The remaining question is how to merge the predictions from the individual image patches. **Note that while the training process treats each image-window independently, the inference mechanism should take into account that there could be multiple predictions for the same world point from patches belonging to different overlapping images.** We propose the following two strategies to combine the predictions from all the overlapping patches to get a final label for a world point: (1) majority voting (Maj-Vote) and (2) averaging the per-point predicted probabilities (soft scores) from overlapping image patches and subsequently converting the averaged probabilty into a hard label (Avg-Soft). We compare both these strategies in Section 9.4.1.

6. MULTI-VIEW DEEP LEARNING FOR SEMANTIC SEGMENTATION

The previous chapters have hopefully equipped the reader with the vocabulary and the tools that are needed to understand the multi-view (and multi-date) training and inference framework that we will expound upon in this chapter. Our multi-view deep-learning framework depends upon the availability of multiple overlapping orthorectified images that are aligned to one another with sub-pixel accuracy.

In Section 5.1.4 we had described our procedure for sampling $F \times F$ image-windows and label-windows from the orthorectified images. *In a more general sense, we can refer to an $F \times F$ window on the ground as a ground-window.* If the images have been aligned with sub-pixel accuracy before orthorectification, then the image-windows and the label-windows drawn from different images that view the same ground-window will correspond to one another on a point-by-point basis, thereby giving us multi-view training data. Each ground-window can be captured by different views at different times.

The rest of this chapter is organized as follows. In Section 6.1 we briefly review relevant literature on multi-view machine learning with a focus on deep-learning approaches. Section 6.2 discusses important considerations to be kept in mind when designing a multi-view learning framework for multi-date satellite images. Section 6.3 presents our proposed CNN architectures, loss functions and strategies for multi-view training and inference.

6.1 Relevant Literature

We have already reviewed the relevant literature on single-view semantic segmentation in Section 5.3. In this section we will restrict our discussion regarding prior

studies that use information from multiple views, to CNN-based approaches. Variants of multi-view CNNs have been proposed primarily for segmentation of image-sequences and video frames, and for applications such as 3D shape recognition/segmentation and 3D pose estimation. State-of-the-art examples include the approaches described by the studies in [133], [134], [135], [136], [137], [138], [139], [140], [141], [142], [143], [144] and [145]. These contributions share one or more of the following attributes – (1) They synthetically generate multiple views by either projecting 3D data into different planes, or by viewing the same image at multiple scales; (2) They extract features from multiple views, subsequently concatenate/pool such features and/or enforce consistency checks between these features; (3) They use only a few views (of the order of 5 or less).

There are not many studies that use CNNs for labeling of multi-view and multi-date satellite images. A relevant contribution is the one described in [47] which won the 2019 IEEE GRSS Data Fusion Contest for Multi-View Semantic Stereo [146]. The work in [46] also uses off-nadir WV3 images for semantic labeling. *Both these approaches treat the different views of the same scene as independent data samples during training. To the best of our knowledge, there does not exist a true multi-view and deep-learning based approach for semantic segmentation using satellite images.*

6.2 Motivation for Our Proposed Approach

From the literature survey in Section 6.1, we see that a common theme in traditional multi-view CNN-based semantic segmentation is to use the encoder to extract features from each view, concatenate them and subsequently pass the concatenated features to the decoder. Many of these approaches also assume the availability of a fixed number of views for both training and inference.

Our multi-view training framework is motivated by the following factors:

Convenience: With newer and better single-view CNNs being designed so frequently, it would be convenient if the multi-view fusion module could be designed as

an add-on to an existing pretrained architecture. This would make it easy to absorb the latest improvements in the encoder and in the decoder architectures directly into the multi-view fusion framework. We won't have to rethink the feature concatenation for each new single-view CNN architecture. Additionally, semantic segmentation networks are memory hungry due to the per-pixel labeling requirement. Therefore, it would be efficient to train the single-view weights in parallel across multiple GPUs and carry out the multi-view data fusion on a single GPU.

Multi-Date Images: The satellite images could have been collected years apart under different illumination and atmospheric conditions. Thus, our task is very different from traditional multi-view approaches that work with 3D shapes or images captured by moving a (handheld) camera around the same scene.

Varying Number of Views: The number of views covering a ground-window can vary between 1 to the total number of available images. For example, Fig. 6.1 shows the view distribution for 858 ground-windows that were sampled from a 100 km^2 region in Ohio. We see that at one extreme, some ground-windows are covered by exactly 1 image and at the other extreme, some windows are covered by all the available 32 images. Therefore, one cannot assume that a minimum number of views will always be available for a ground-window when updating the weights of a CNN. This causes practical challenges in backpropagating gradients when using multi-view CNN architectures that assume the availability of a fixed number of views for concatenating features. More importantly, if we consider a single batch for training to be made up of views of the same ground-window, then the batch size can vary greatly from batch to batch which adversely impacts the convergence rates of the CNNs.

A naive solution of only training with ground-windows that are covered by at least S views (where S is a user-specified parameter) suffers from the drawback that a lot of useful training data is likely to be excluded. This is magnified in the presence of sparse and noisy OSM labels as corroborated by our own experimental results. We do not want to exclude windows that are covered by less than a specified number of

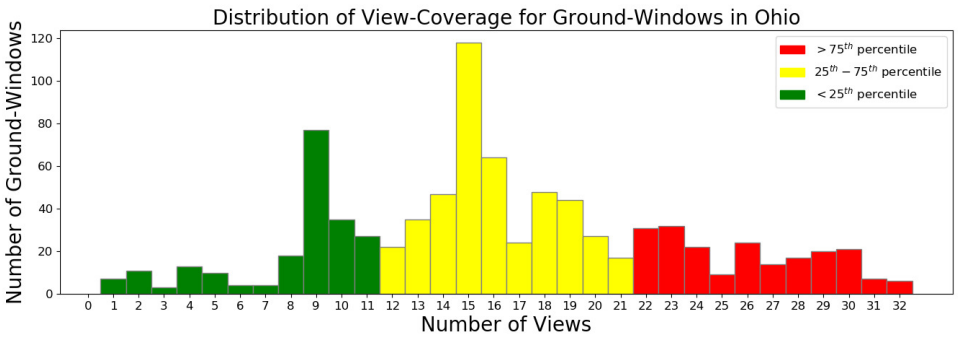


Fig. 6.1.: Distribution of the number of views for 858 ground-windows sampled across a 100 km^2 region in Ohio. This can also be interpreted as the distribution of the batch sizes if each batch consists of all the views for a single ground-window.

views. Our goal is to use all the available training data and all the available views for every ground-window.

The variability in the number of views also necessitates that the CNN should learn to predict labels from just a single-view as well as from multiple views. Therefore, the decoder portion of the CNN might also benefit from looking at a single view. What we mean by this is that it might be beneficial to fuse the multi-view information after the decoder stage rather than at the encoder stage.

Impact of View-Angle Changes on the Scene Content: WV3 images are captured from an altitude of ∼617 km with a nadir GSD of 0.31 m and off-nadir GSDs that can be as large as 0.4-0.45 m. At this resolution, due to occlusion by tall structures and illumination artifacts from reflecting surfaces, even a modest change in the view angle can produce a dramatic change in the scene content when compared to traditional multi-view applications. This also means that a true ortho image can have a lot of missing data due to occlusion and noise in the DSM, which can affect the fusion of features from multiple views.

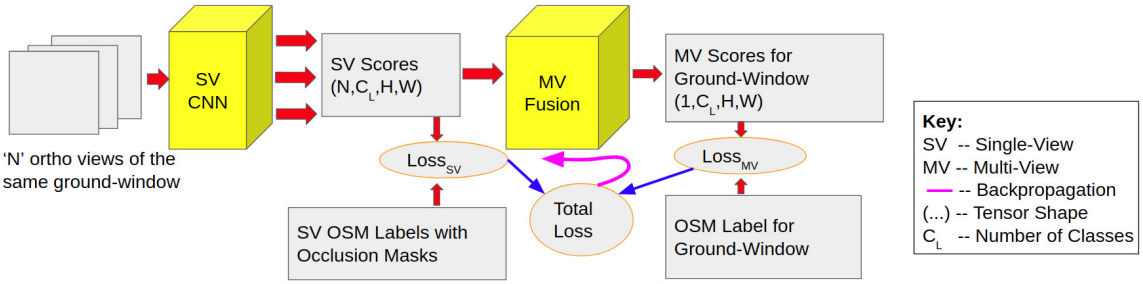


Fig. 6.2.: Overview of Multi-View Training

6.3 Multi-View Fusion CNN

Fig. 6.2 shows an overview of our multi-view training framework where we propose that the multi-view information be aggregated using the predictions of the single-view CNN (SV CNN). In this sense, our approach is related to the strategies employed by the studies described in [140] and [145]. While the contribution described in [140] considers the "RGB" and the depth channel of the same RGB-D image as two "views" (which is a much simpler case), the 3D shape segmentation approach proposed in [145] synthetically generates multiple-views of the same 3D object. In contrast, the significantly more complex nature of our data makes our problem very different from these tasks. Nevertheless, the strong performances of these approaches encourage us to intelligently combine the multi-view and multi-date information.

The multi-view fusion (MV Fusion) module shown in Fig. 6.2 can be added to any existing/pretrained single-view CNN (SV CNN). We experimented with different choices for this module and present two that give good performance yields. These are shown in Fig. 6.3 and we denote them as MV-A (Multi-View-A) and MV-B(Multi-View-B) respectively. Both MV-A and MV-B consist of a single block of weights with kernel size, stride and padding set to 1.

In the following discussion, V denotes a subset of views for a single ground-window. N is the number of views in V. H and W are the height and width of a single view respectively. M is the maximum number of possible views for a ground-window. C_L is the number of target classes.

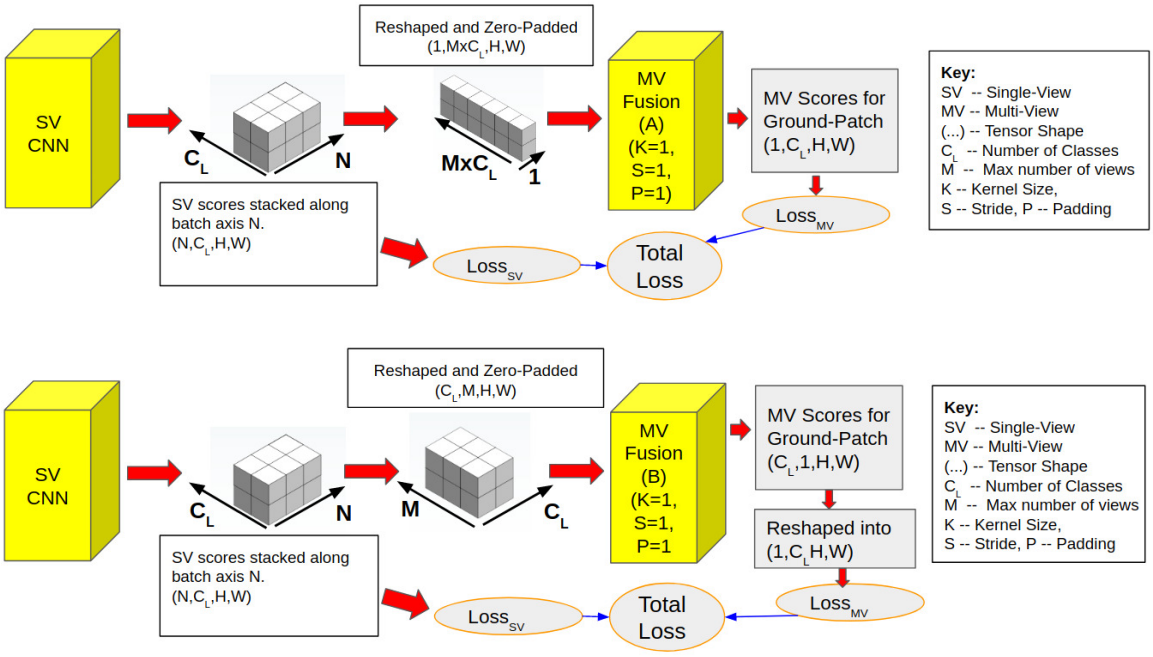


Fig. 6.3.: Two choices for Multi-View Fusion. At top is MV-A in which the weights of the MV Fusion layer are different for each channel of each view. At bottom is MV-B where the weights of the MV Fusion layer are shared by all the channels of a view.

As shown in Fig. 6.2, the Single-View CNN (SV CNN) outputs a tensor of shape (C_L, H, W) for each of N views which are concatenated along the batch axis to yield a tensor of shape (N, C_L, H, W), which we denote as T_i. This tensor is then inserted into a larger tensor which we denote as T_{MV}. Each view has a fixed index in T_{MV}. Missing views are filled with zeros. The difference between MV-A and MV-B can now be explained as follows.

MV-A: In this case T_i is reshaped into a tensor of shape $(1, N \times C_L, H, W)$. It is then inserted into T_{MV} which is of shape $(1, M \times C_L, H, W)$. T_{MV} is then input to the MV-A module which subsequently outputs a tensor of shape $(1, C_L, H, W)$. MV-A thus contains a total of $M \times C_L$ trainable weights, one for each channel of each view.

MV-B: In this case, T_i is first reshaped into a tensor of shape (C_L, N, H, W). It is then inserted into T_{MV} which is of shape (C_L, M, H, W). T_{MV} is then input to the MV-B module which subsequently outputs a tensor of shape $(C_L, 1, H, W)$. MV-B

thus contains a total of M trainable weights, one for each view. The first and second axis of this tensor are swapped to yield a tensor of shape $(1, C_L, H, W)$ which is then used to calculate the loss.

We now describe how the loss is calculated while training our network.

6.4 Loss Function

The total loss is defined as

$$L = \alpha \cdot L_{SV} + \beta \cdot L_{MV} \tag{6.1}$$

where L_{SV} represents the single-view loss, L_{MV} represents the multi-view loss and α and β are scalars used to weight the two loss functions. The single-view loss is calculated as follows.

$$L_{SV} = \frac{1}{N} \sum_{i=1}^{N} \mathrm{CE}_i(G_i, T_i) \tag{6.2}$$

where CE_i is the pointwise cross-entropy loss for the i^{th} view. As defined in the previous subsection, N is the number of views in a subset V of views that cover a single ground-window and T_i is the output tensor of the SV CNN for the i^{th} view. To calculate CE_i, we mask the OSM labels for the ground-window with the occlusion mask of the i^{th} view. This masked ground-truth is denoted by G_i in the equation above. Note that this mask is implicitly computed during the process of true orthorectification. The gradients of L_{SV} are not backpropagated at these masked points. What this means is that for each individual view, L_{SV} only focuses on portions of the ground-window that are visible in that view.

The pointwise cross-entropy loss between two probability distributions A and B, each defined over C_L classes, is calculated as follows.

$$\text{CE}(A, B) = -\sum_{p}\sum_{j=1}^{C_L} A(p, j) \cdot log(B(p, j)) \tag{6.3}$$

where p refers to a single point. $A(p, j)$ is the probability that point p belongs to class j as defined by A. $B(p, j)$ is the probability that point p belongs to class j as defined by B.

The multi-view loss is calculated as follows.

$$L_{MV} = \text{CE}(G, P_{MV}) \tag{6.4}$$

where $\text{CE}(G, P_{MV})$ is the pointwise cross-entropy loss for the ground-window. This is calculated using the unmasked OSM label G and the output P_{MV} of the MV Fusion module. P_{MV} can be viewed as a final probability distribution that is estimated by fusing the individual probability distributions that are output by the SV CNN for each of the N views. We can denote P_{MV} as a function $f(T_1, T_2, ..., T_N)$ where f depends upon the architecture of the MV Fusion module. Note that f is differentiable. Thus, Eq. 6.4 can be rewritten as

$$L_{MV} = \text{CE}(G, f(T_1, T_2, ..., T_N)) \tag{6.5}$$

Substituting the expression for the CE loss from Eq. 6.3 into Eq. 6.5, we get the following expression for L_{MV}.

$$L_{MV} = -\sum_{p}\sum_{j=1}^{C_L} G(p, j) \cdot log(f(T_1, T_2, ..., T_N)(p, j)) \tag{6.6}$$

Note that L_{MV} **is not linearly separable over the views in V.** In other words, unlike L_{SV}, we cannot separate it into a sum of losses for each view. Thus,

L_{MV} captures the predictions of the network in an ensemble sense over multiple views covering a ground-window. When backpropagating the gradients of L, the gradients from L_{MV} are influenced by the relative differences between the predictions for each view, and this in turn translates into better weight-updates. Moreover, by using L_{MV}, the network is shown labels for all portions of the ground including those that are missing in some views of V. This enables the network to make better decisions about occluded regions using multiple views.

We are now ready to explain the different ways in which we group the training data in order to employ different strategies to train the network. In Section 9.5, we provide a quantitative comparison of these strategies. We have carried out these comparisons for both the MV-A and MV-B modules.

6.5 Strategies for Data-Loading

The term "data-loading" refers to how the data samples are grouped into batches and input to the CNN. We use two different data-loading approaches.

Single-View Data-Loading (SV DATALOAD): This is a conventional data-loading strategy where a single training batch can contain views of different ground-windows. The batch size is constant and only depends on the available GPU memory. We use the efficient selection strategy described in Section 5.1.4.1 to get training windows. We denote these set of windows as SV WINDOW. To create SV WINDOW, the images are sampled independently in parallel.

Multi-View Data-Loading (MV DATALOAD): In this case, a single training batch consists solely of views that cover the same ground-window. The number of such views can vary from window to window. Therefore, while creating training windows, we have to sample the same ground-window from multiple images. We denote these set of windows as MV WINDOW. Note that the same training region is used to collect both SV WINDOW and MV WINDOW. One can therefore expect a high degree of correlation and overlap between MV WINDOW and SV WINDOW. We will revisit

this point at the end of this chapter. Focusing for now on MV DATALOAD, due to memory constraints, we might not be able to simultaneously load all the available views for one ground-window onto the GPU(s) in one batch. As a work around, we use the following approach.

Let $|Q|$ denote a pre-specified number of views that can fit into the GPU memory, R denote the set of available views for a ground-window and $|R|$ denote the total number of views in R. If $|R| < |Q|$, we skip loading this ground-window. If $|R| > |Q|$, we randomly split R into a collection of overlapping subsets $\{Q_j\}$, such that each Q_j has $|Q|$ views and $\cup\, Q_j = R$ where $\cup$ denotes the union operator. The tensor T_{MV} (defined in Section 6.3) that is input to the MV Fusion module is reset to zero before inputting each Q_j to the CNN. Note that *this random split has the added advantage that the CNN sees different groupings of views in different epochs for the same ground-window, which should help it to learn better.*

The design of MV DATALOAD was motivated by our observation that both small batches and widely-varying batch sizes produced unstable gradient updates that adversely impacted the rate of convergence during training. These are well-known issues when using stochastic gradient descent. Therefore, we exclude ground-windows with less than $|Q|$ views, and for the remaining windows we make sure that every subset Q_j has $|Q|$ views. *This enforces a constant batch size of* $|Q|$, resulting in faster convergence.

In the following section, we will explain how we use SV DATALOAD and MV DATALOAD as part of different strategies for training the network.

6.6 Training Strategies

We propose to use the following different training strategies.

Single-View Training (SV TRAIN) - In this strategy, only the SV CNN is trained. We apply the SV DATALOAD approach to use all the windows in SV WINDOW. One can also interpret this as setting $\beta = 0$ in Eq. 6.1.

MV TRAIN-I - We first train the SV CNN using SV TRAIN. Subsequently, we use MV DATALOAD to only train the MV Fusion module by setting $\alpha = 0$ in Eq. 6.1, and by freezing the weights of the SV CNN. Hence, L_{MV} only affects the weights of the MV Fusion module and does not affect the SV CNN.

MV TRAIN-II - We first train the SV CNN using SV TRAIN. Subsequently, both the pretrained SV CNN and the MV Fusion module are trained together using MV DATALOAD and the total loss (Eq. 6.1). Thus, L_{MV} influences the weight updates of the SV CNN as well. In practice, we lower the initial learning rate for the SV CNN as it has already been trained and we only want to fine-tune its weights. The weights of the MV Fusion module are initialized to $1/|W|$ where $|W|$ denotes the total number of weights in the MV Fusion module.

MV TRAIN-III - In this strategy, *we do not pretrain the SV CNN*, but rather train both the SV CNN and the MV Fusion module together from scratch using the total loss L (Eq. 6.1) and MV DATALOAD. This has the disadvantage that the network never sees ground-windows with less than $|Q|$ views, where $|Q|$ is a user-specified parameter. One might expect this reduction in the size of the training data to negatively impact performance, given the sparse nature of the OSM labels. Our experimental evaluation confirms this.

To make a decision on when to stop training, a common practice in machine-learning is to use a validation dataset. However, in our case the validation data is also drawn from OSM (to avoid any human intervention), and is therefore noisy. To handle this, we make the following proposal. We train a network until the training loss stops decreasing. At the end of every epoch, we measure the IoU using the validation data. For inference, we save the network weights from two epochs – one with the smallest validation loss and the maximum validation IoU, and the other with a very small training loss and a validation IoU that is close to the maximum validation IoU. We denote the former as **EPOCH-MIN-VAL** and the latter as **EPOCH-MIN-TRAIN** respectively. Please note that the validation labels are also automatically

derived from OSM. The validation samples do not overlap with the training samples or with the "unseen" test region used for inference.

6.7 Inference

To establish a baseline, we use a SV CNN trained with the SV TRAIN strategy defined above, and merge the predictions from the overlapping views via majority voting (or alternatively by averaging the soft scores from each view). We will denote this approach as SV CNN + VOTE. Inference using the SV CNN + MV Fusion module is noticeably faster than SV CNN + VOTE because the former combines multi-view information directly on the GPU.

For inference using the SV CNN + MV Fusion network, it might be possible to simultaneously load all the views for each ground-window onto multiple GPUs if they are available. If we wish to use a single GPU for inference, the MV DATALOAD approach can be used with the following minor modification. As in training, we split a set of views R for a ground-window into a collection of subsets Q_j, forward each Q_j through the SV CNN and store the outputs in the T_{MV} tensor. *Unlike training, we do not reset the T_{MV} tensor to zeros before inputting each Q_j.* After all the Q_j's corresponding to one ground-window have been forwarded through the SV CNN, T_{MV} is input to the MV Fusion module. This means that the final prediction is still made using all the views. T_{MV} is reset to zero for the next ground-window.

6.8 Data Augmentation vs MV Training

We conclude this chapter with a discussion on the difference between multi-view training and data augmentation. Assume for a minute that it turns out that one of the MV TRAIN strategies performs better than the SV TRAIN strategy. One might wonder if the MV TRAIN strategy accomplishes anything or whether the improvement is merely because of the extra training samples that MV DATALOAD shows to the network. More precisely, is the improvement merely a result of training

the network using the samples from both SV WINDOW and MV WINDOW? Or is the improvement actually coming from L_{MV} and the MV TRAIN and the MV DATALOAD strategies?

One could argue that the efficient window-selection principles that we employ (Section 5.1.4.1) combined with the fact that both SV WINDOW and MV WINDOW are collected from the same training region make it highly likely that the extra data samples in MV WINDOW by themselves do not provide any new information to the network. Nonetheless, to dispel any doubts, we also train the SV CNN using SV TRAIN on **all** the available training data samples, i.e., SV WINDOW and MV WINDOW. This CNN is denoted as SV CNN + DATA-AUG. *The SV CNN + MV-A (or MV-B) network never sees any training sample that is not seen by SV CNN + DATA-AUG.* We run inference using SV CNN + DATA-AUG and merge the predictions using majority vote. This strategy is denoted as SV CNN + DATA-AUG + VOTE.

7. PUTTING THE PIECES TOGETHER TO CREATE AN END-TO-END AUTOMATED FRAMEWORK

As the title of this chapter suggests, we will now talk about the overall architecture of our framework and how the different pieces described thus far come together to create an end-to-end automated framework. We would like to highlight that *apart from the crowd-sourced effort in creating OSM annotations, there is no human supervision to the system.*

7.1 Semantic Labeling of World Points

A bird's-eye view of the proposed framework for labeling world points is shown in Fig. 7.1. The framework has three inputs that are shown by the orange colored boxes: (1) the panchromatic and 8-band multispectral satellite images; (2) the camera model (RPCs) and the metadata associated with the images; and (3) the OSM vectors. The framework directly outputs semantic labels for the world points. We now provide a brief rundown of each of the modules shown in Fig. 7.1.

- **ToA Correction: (Section 2.2)**
  - **Input:** Off-nadir images and their metadata
  - **Output:** Top-of-Atmosphere corrected off-nadir images containing reflectance values
  - **Comments:** We apply Top-of-Atmosphere (ToA) correction to the images to compensate for the variations in the solar zenith and the Earth-Sun distance. Images are processed independently and in parallel. We use the publicly-available geoio software [147] with a few extra modifications of our own, to apply ToA correction to the images.

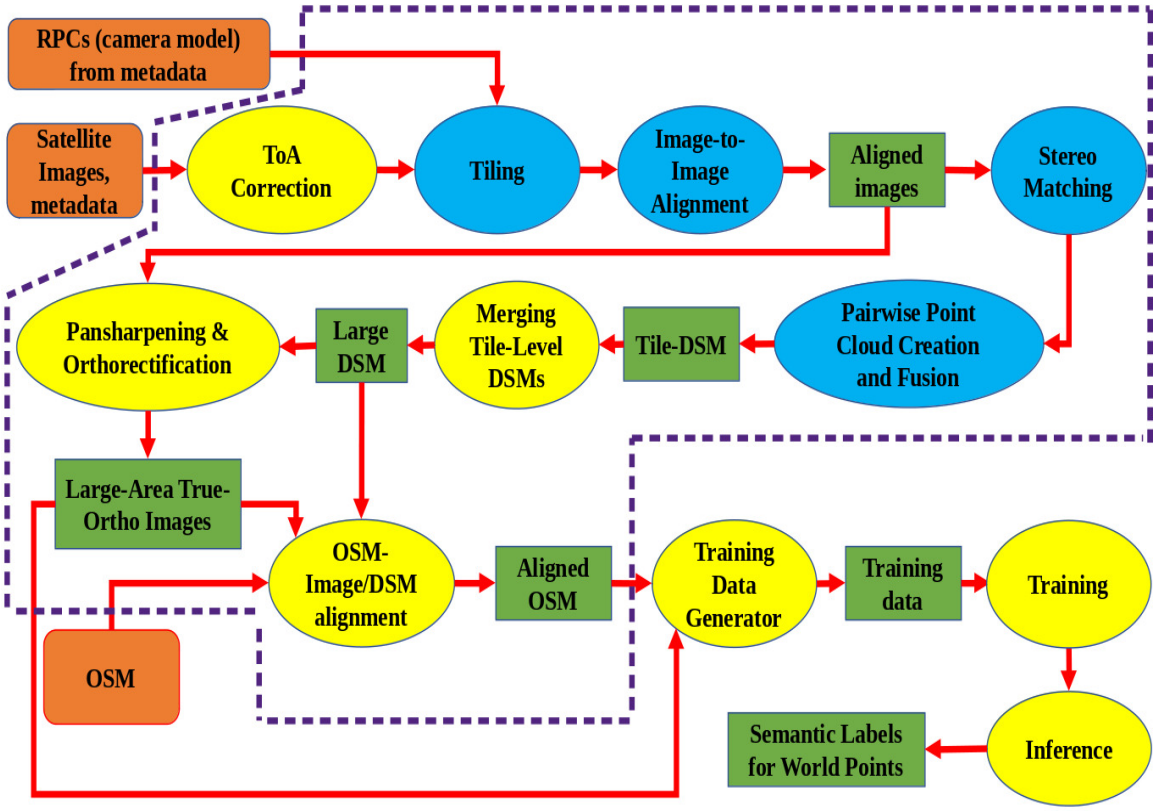


Fig. 7.1.: Overview of our framework. The modules enclosed in the dashed purple lines are collectively referred to as the geo-processing module.

- **Tiling: (Section 4.1.3)**
  - **Input:** Off-nadir images and the uncorrected RPCs. Additionally, we specify the dimensions of a tile
  - **Output:** For each tile, the off-nadir image patches and the corresponding uncorrected RPCs, and a vector polygon representing the tile in geodesic coordinates
  - **Comments:** The system first calculates the extents of the area of interest such that any point in this area is covered by atleast A images where A is a user specified value. Subsequently, the images are divided into tiles, in parallel.
- **Image-to-Image Alignment: (Section 3.3)**
  - **Input:** For each tile, the off-nadir image patches and the corresponding uncorrected RPCs
  - **Output:** Corrected RPCs for each image patch of each tile
  - **Comments:** Tiles are processed independently and in parallel.
- **Stereo Matching: (Sections 3.4.1 and 3.4.2)**
  - **Input:** For each tile, the off-nadir image patches and the corresponding corrected RPCs. Additionally a coarse SRTM DEM is provided
  - **Output:** For each tile, the stereo-rectified image patches and the corresponding disparity maps for each selected pair of image patches
  - **Comments:** For a single tile, multiple pairs of image patches are processed in parallel to create multiple pairwise disparity maps. Tiles are processed sequentially.
- **Pairwise Point Cloud Creation and Fusion: (Section 3.4.3)**

  - **Input:** For each tile, the off-nadir image patches, the stereo-rectified image patches, the corrected RPCs and the disparity maps
  - **Output:** DSM for each tile
  - **Comments:** For a single tile, multiple pairs of image patches are processed in parallel to create multiple pairwise point clouds. Fusion and conversion to a 2.5D DSM is then carried out on a single machine. Tiles are processed sequentially.

- **Merging Tile-Level DSMs: (Section 4.3)**
  - **Input:** DSM for each tile
  - **Output:** Large-area DSM

- **Pansharpening and Orthorectification: (Sections 2.3 and 5.1.2)**
  - **Input:** Panchromatic and MS image patches and the corresponding corrected RPCs from all the tiles, Large-area DSM
  - **Output:** Full-sized pansharpened true ortho images and occlusion masks
  - **Comments:** Images are pansharpened in parallel. For true orthorectification, tiles are processed in parallel.

- **OSM-Image/DSM Alignment: (Section 5.1.3)**
  - **Input:** Full-sized orthorectified images, OSM vectors
  - **Output:** Aligned and rasterized OSM labels
  - **Comments:** Since the images are aligned with one another, it is sufficient to align the OSM data to any one image. The OSM rasters have the same GSD as the images.

- **Training Data Generator: (Section 5.1.4)**
  - **Input:** Full-sized true ortho pansharpened images and their corresponding rasterized OSM labels

  - **Output:** Training data samples
  - **Comments:** As described in Section 6.5, it is necessary to keep track of which image-windows and label-windows correspond to the same ground-window for MV DATALOAD

- **Training: (Sections 5.6 and 6.6)**
  - **Input:** Training data samples
  - **Output:** Trained CNN weights
  - **Comments:** The framework can use any CNN in single-view or multi-view mode.

- **Inference: (Sections 5.6.1 and 6.7)**
  - **Input:** Trained CNN weights, inference data samples
  - **Output:** Semantic labels for world points

7.2 Semantic Labeling of Off-Nadir Images

As explained in Section 5.2, our framework can also be used to train a CNN directly on the off-nadir image-windows. Fig. 7.3 depicts an over-view of this approach.

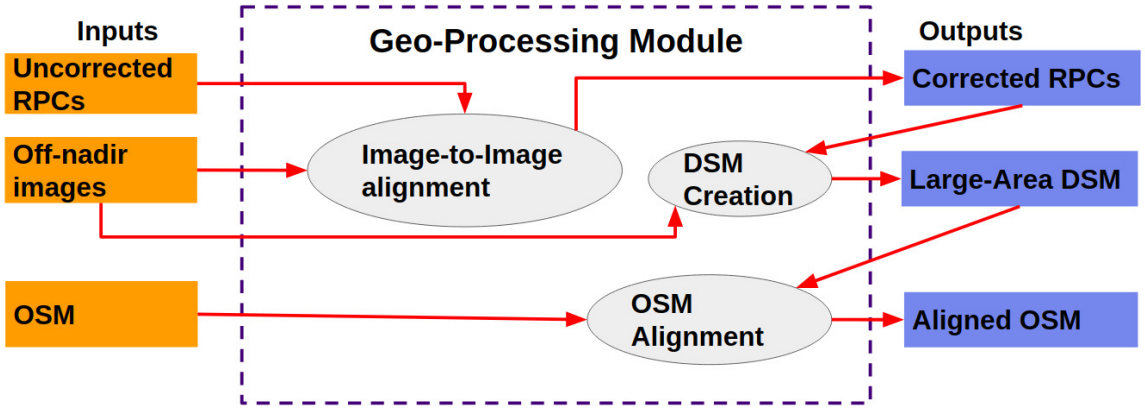


Fig. 7.2.: Summary of the geo-processing module. The different components of this module are enclosed within purple dashed lines in Fig. 7.1.

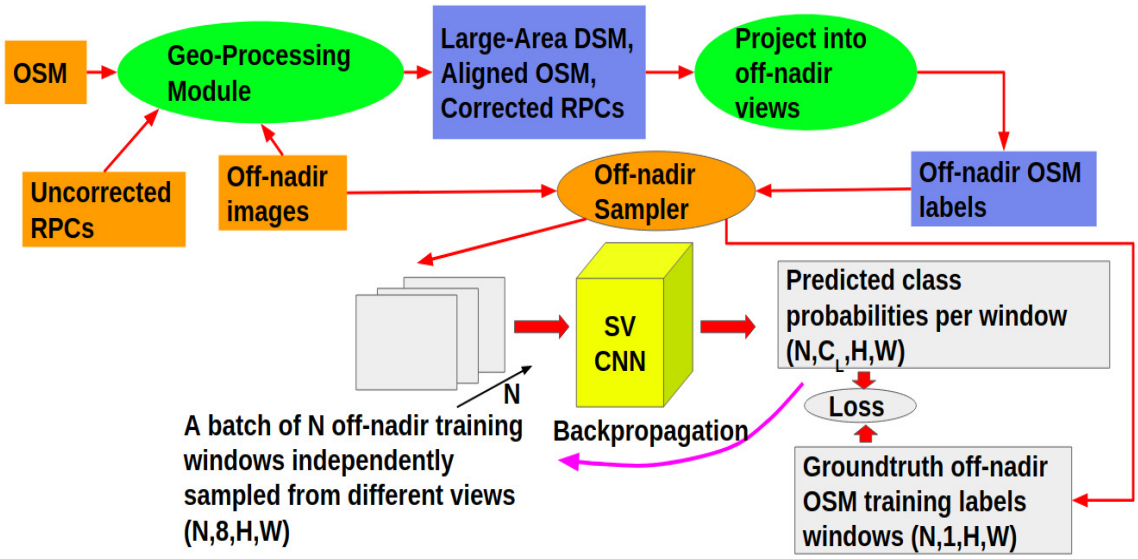


Fig. 7.3.: Overview of our approach to train CNNs directly on the off-nadir image-windows. The geo-processing module is shown in Figs. 7.2 and 7.1.

With respect to the framework described in the previous section, the key difference is the "Off-nadir Sampler" module in Fig. 7.3. We have already explained the procedure to create OSM labels in the off-nadir space in Section 5.2. As shown in Fig. 7.3, we only use the SV CNN (Single-View CNN) in this case. Technically it is possible to construct a multi-view training framework in the off-nadir space as well and we leave that for future work.

8. AN OPENSTACK CLOUD-BASED IMPLEMENTATION OF THE FRAMEWORK

This research would not have been possible without access to RVL Cloud [30] – the OpenStack cloud computing framework at Purdue RVL. To aid the research community, in this chapter we would like to provide some useful insights that we have gained while developing an end-to-end automated and distributed cloud-based framework for large-area DSM creation and semantic labeling.

Among other things, this chapter describes some important tips and tricks for efficient distribution of tasks across a cloud. We highlight why CPU pinning [148], [149], [150] and non-uniform memory access (NUMA) nodes [151], [148] are important for getting code-execution times on virtual machines (VMs), that are comparable to the code-execution times on physical/host machines. We also explain our distributed process for creating pairwise point clouds and show how we save days of processing with efficient communication between nodes via a captain-worker protocol. As part of this description, we touch upon some built-in safeguards to handle any runtime failures, and explain a few additional design choices. Finally we talk about some important features of our PyTorch [152] based multi-GPU and multi-view training framework. *Please note that the contents of this chapter might appear a tad conversational for someone with a hardware/systems background. The objective is to provide an accessible explanation for remote-sensing and computer-vision researchers who are interested in developing algorithms for large-areas.*

8.1 Improving the Performance of a Cloud: Matching Runtimes on a Virtual Machine and a Host Machine

This section will be useful for researchers who have their own in-house cloud computing framework. Anybody who uses other popular cloud computing frameworks such as Amazon Web Services (AWS) or Microsoft Azure can feel free to skip this section because those frameworks abstract away such technical details. In fact, although RVL Cloud has been operational since 2015, we became cognizant of the issues described in this section only after we were tasked with designing algorithms for stereo matching. Our subsequent investigation and conclusions resulted in a overall speedup of the cloud across the board.

The cost-aggregation step in the disparity-map calculation portion of any SGM-based stereo-matching algorithm (Section 3.4.2) is embarrassingly parallel when setup as a dynamic programming problem. OpenMP [67] gives us an easy and straightforward way to exploit this parallel nature. However, we ran into the following issues when we tried to parallelize our algorithm.

8.1.1 Less is More: Reserving CPUs for the Host Machine

For a simple test case, we wanted to run the tSGM algorithm on 8000 $\times$ 8000 image patches. On a host machine with 28 cores (56 with hyperthreading) and 125 GB of RAM, it ran to completion in 4-5 minutes on average.

We then launched an OpenStack VM with 60 GB of RAM and 40 VCPUs[1] (virtual CPUs) on the same host machine and ran the tSGM algorithm on the same input images. The run time was around 10-12 minutes. Our initial guess was to attribute this slower runtime to the overhead caused by virtualization and the fact that the VM had a fewer number of cores when compared to the host machine. We therefore

[1]If you have ever launched a virtual machine (VM) on any cloud computing framework, you might remember selecting the number of processors for your VM. It is important to note that what you are actually selecting is the number of "virtual" processors. Specifically in OpenStack, these are called as "VCPUs" or virtual CPUs. Please note that the term CPU is abused here in the sense that it refers to the virtual thread or core rather than a CPU.

decided to repeat the experiment on a larger VM with 125 GB of RAM and all available 56 VCPUs. **The run time crawled to a whopping 40 minutes on this VM running on the same host machine.**

Our subsequent investigation suggested that since the OpenStack framework runs some native processes on the host machine, the larger VM and these native processes were competing for the same 56 VCPUs. Quoting from [150], "The exact configuration depends on the NUMA topology of your host system; however, you must reserve some CPU cores across all the NUMA nodes for host processes and let the rest of the CPU cores handle your guest virtual machine instances". With the smaller VM (with 40 VCPUs), there were 16 free VCPUs which OpenStack could use to execute its native processes. **The important takeaway from these experiments is to reserve some CPUs for the host machine.**

8.1.2 Discrepancy in the Code-Execution Times between the Virtual and the Host Machines

Even after reserving some CPUs for the host machine, *the same tSGM stereo-matching code ran 2-3 times slower on a VM than on a host machine (with identical RAM and number of CPUs) for the same input data. Even stranger was the observation that adding more VCPUs to the VM further slowed it down.*

This 2-3x slowdown might not be an issue when dealing with small AOIs. However it makes a noticeable difference for large AOIs. Consider an example where we have a single VM and a host machine with the same number of CPUs and a similar amount of RAM. Assume that creating a single pairwise point cloud for one tile takes 10 minutes and that we want to create 80 pairwise point clouds for each tile. On the host machine this will take 800 minutes which equals roughly 13 hours. On the VM this will take between 26-39 hours. Now consider the case where we have to process 100 tiles covering a 100 km^2 area. The host machine would take roughly 55 days whereas the VM would take between 110-165 days. Even if we had access to 10 VMs

and processed the tiles in parallel, there would still be a run time of 11-16 days. If we could make a VM match the runtime of a host machine, we could cut this time down to 5-8 days thereby saving a week's worth of processing time.

Our investigation into this issue ultimately led us to the two concepts of CPU pinning and NUMA nodes. In the following subsections, we explain these two concepts and show how to make a VM nearly match the runtime of a host machine.

8.1.3 CPU Pinning

By default, OpenStack uses floating VCPUs meaning that the VCPU processes are not fixed to a specific host CPU. Rather they float across the host CPUs. This default behavior allows for overcommitting the resources of the underlying host machine. Fig. 8.1 shows the output of running the "virsh vcpuinfo" command on the same default OpenStack VM at two different times. The "virsh" tool [153] is part of the libvirt toolkit [154] that lets us manage and inspect virtual machines. The "virsh vcpuinfo" command can be run with an instance name or id number to obtain details about the VM's VCPUs.

The VM in Fig. 8.1 is named "instance-00000264" and has 4 VCPUs. From this figure, we see that each VCPU has an associated "CPU" field which refers to the id of the underlying physical core to which that VCPU is attached. Comparing Figs. 8.1a and 8.1b, we see that the "CPUs" associated with VCPUs 0, 1 and 3 have changed in the time interval between the two calls to "virsh vcpuinfo". Also notice the "--yyyy...yyyy" entry for the "CPU Affinity" field. The underlying host machine on which this VM is running has 72 cores (36 physical with hyperthreading = 72). 4 of these are reserved for the host machine itself and these are indicated by the "-" entries in the "CPU Affinity" field. The "y" entries for the remaining 68 places indicate that the VCPUs can float between any of these 68 CPUs.

```
[redacted]~# date; virsh vcpuinfo instance-00000264
Thu Jun  4 00:13:41 EST 2020
VCPU:           0
CPU:            71
State:          running
CPU time:       26934.9s
CPU Affinity:   --yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy--yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy

VCPU:           1
CPU:            3
State:          running
CPU time:       28143.3s
CPU Affinity:   --yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy--yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy

VCPU:           2
CPU:            65
State:          running
CPU time:       26847.1s
CPU Affinity:   --yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy--yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy

VCPU:           3
CPU:            5
State:          running
CPU time:       29222.1s
CPU Affinity:   --yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy--yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy
```

(a)

```
[redacted]~# date; virsh vcpuinfo instance-00000264
Thu Jun  4 00:14:42 EST 2020
VCPU:           0
CPU:            5
State:          running
CPU time:       26935.2s
CPU Affinity:   --yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy--yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy

VCPU:           1
CPU:            7
State:          running
CPU time:       28143.5s
CPU Affinity:   --yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy--yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy

VCPU:           2
CPU:            65
State:          running
CPU time:       26847.4s
CPU Affinity:   --yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy--yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy

VCPU:           3
CPU:            9
State:          running
CPU time:       29222.3s
CPU Affinity:   --yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy--yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy
```

(b)

Fig. 8.1.: Output of running the "virsh vcpuinfo" command for the same OpenStack VM with 4 floating VCPUs, at two different times. Note how the "CPU" id has changed for the VCPUs with ids 0, 1 and 3 between the top and bottom figures. Also note the entries for the "CPU Affinity" fields for each VCPU.

Under normal circumstances one would not notice the latency caused by these floating VCPUs. However, **due to the massive parallelization in tSGM, a floating VCPU causes repeated processor cache-coherency checks which introduces significant overhead and latency. This is also why increasing the number of VCPUs further aggravates the problem.**

Luckily OpenStack allows for "CPU pinning". Loosely speaking, we can configure the underlying host machine and a VM such that its VCPUs are no longer floating, but rather pinned to specific physical CPUs. This eliminates the need for repeated cache-coherency checks, thereby producing significant speedup for tSGM. Fig. 8.2 shows the output of running the "virsh vcpuinfo" command on a VM that has been configured with CPU pinning. The "CPU" id for each VCPU remain static. Also note how the "CPU Affinity" field for each VCPU has a "y" at only one fixed place indicating that that VCPU is pinned to that particular CPU. More details on how to configure CPU pinning in OpenStack can be found in [148].

With CPU pinning enabled, we observed that the tSGM algorithm ran significantly faster on the VM. For our specific example of 8000×8000 image patches, there was still a residual latency of a minute between the VM and the host machine. More importantly, we noticed that increasing the number of VCPUs reduced the runtime only up to a point. Beyond this, increasing the number of VCPUs increased the latency slightly. We elaborate upon the cause of this issue in the next subsection.

8.1.4 NUMA

As mentioned in [151], NUMA stands for Non-Uniform Memory Access. The total RAM is divided between the CPUs such that each processor can access its local memory with much lower latency than its non-local memory. There exist different possible NUMA configurations depending upon the number of CPUs, physical cores and the manufacturer. In RVL Cloud there are usually two configurations – the cores are either split in an odd-even manner or in a sequential manner. For a specific

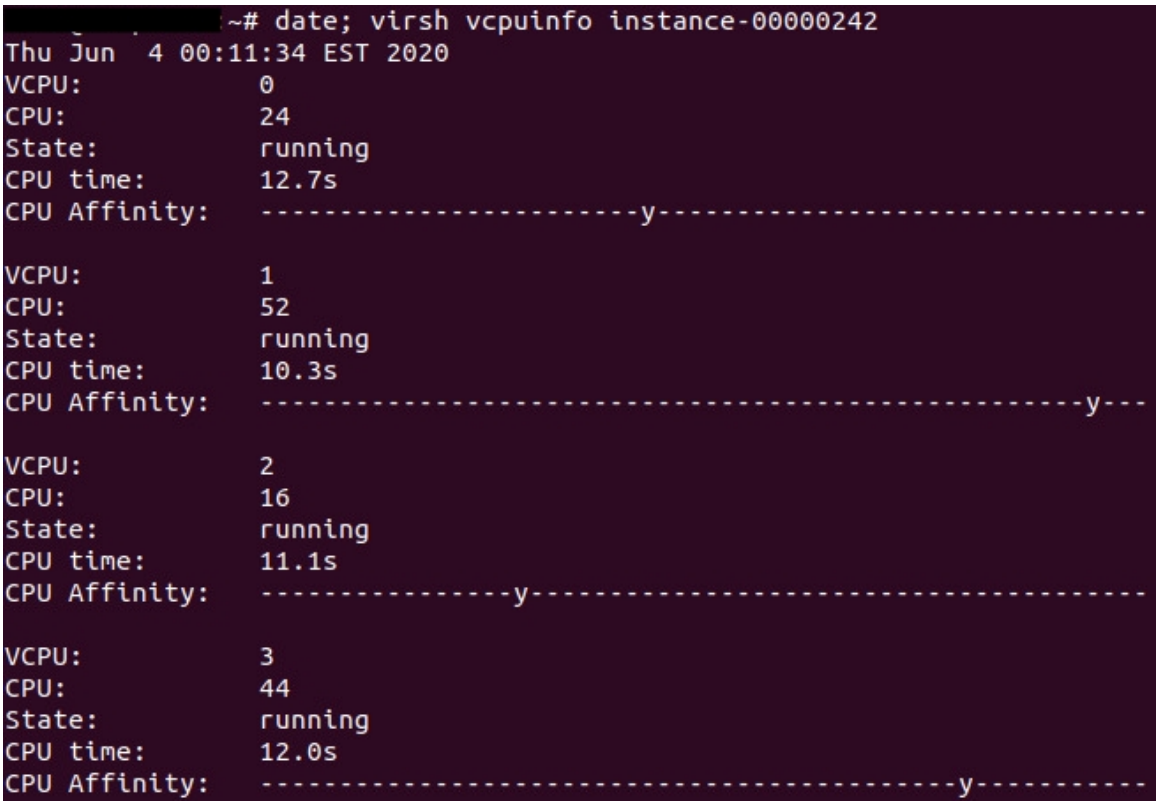


Fig. 8.2.: Output of running the "virsh vcpuinfo" command for an OpenStack VM with 4 pinned VCPUs. Note the entries for the "CPU Affinity" field for each VCPU.

example, consider a host machine with 20 cores and 2 NUMA nodes denoted as NUMA 0 and NUMA 1. In an odd-even split, cores with even ids 0, 2, 4,.., 18 will be assigned to NUMA 0 and cores with odd ids 1, 3, 5,.., 19 will be assigned to NUMA 1. In a sequential split, cores with ids 0, 1,.., 9 will be assigned to NUMA 0 and cores with ids 10, 11,.., 19 will be assigned to NUMA 1.

One can immediately see a possible explanation for why one cannot keep on increasing the number of VCPUs and expect to see faster runtimes. For our example host machine with 20 cores, consider a VM with 8 VCPUs and another VM with 12 VCPUs running on the host machine. It is possible to configure the 8-VCPU VM such that all its VCPUs share the same NUMA node. However, for the latter VM with 12 VCPUs, the VCPUs are definitely split across NUMA nodes. In this case, when multiple threads are operating on the same data in parallel, **the data is split across NUMA nodes resulting in slightly increased latency.**

Even for the case of the VM with 8 VCPUs, OpenStack by default splits the VCPUs across NUMA nodes. For our specific example, we noticed that *ensuring that all the requested VCPUs share the same NUMA node almost closed the 1-minute latency gap between the VM and the host machine for tSGM.* And as mentioned before, increasing the number of VCPUs beyond a threshold results in diminishing gains. This threshold depends on the host machine and the number of available cores.

To sum up, by pinning the VCPUs to the host CPUs and by ensuring that the VCPUs are not split across NUMA nodes, one can make a virtual machine match the runtimes of the underlying host machine. In our experiments, we sometimes relax the NUMA constraint so that we can get additional VMs. Although each such VM has slightly higher latency, the increased number of VMs more than makes up for this small overhead. More details on how to configure NUMA nodes in OpenStack can be found in [148].

8.2 Distributing Workload Across a Cloud

8.2.1 Some Useful Minutiae

During the course of our experiments we encountered a few issues (some intuitive and some not so intuitive) that need to be kept in mind when designing a distributed framework for large-area processing. We briefly discuss some of them below.

(i) **Network Bandwidth:** - Satellite images are huge. If we factor into account the sizes of the intermediate and the final outputs produced by tiling, pansharpening, ToA correction, stereo matching, point cloud creation and DSM fusion, etc., for a 100 km^2 region, we are talking about tens of Terabytes (TB) of data. Therefore, it is practical to store all the data on Network Attached Storage (NAS) devices. The RVL Cloud has a 10 Gbps backbone LAN (Local Area Network) for this purpose. However, if say 10 VMs are working in parallel with multiple read/write operations, this can cause a lot of network traffic. It turns out that the following simple strategy can alleviate this problem. For any module in the framework, we create local copies of the input for that module and then run the module on these local copies. We write the intermediate and final outputs locally and then move them to their appropriate destinations on the NAS. This simple strategy turns out to be much faster than directly carrying out read/write operations across the network.

(ii) **Unequal VMs:** For some operations it might be prudent to use as many VMs as possible. However, when using a shared cloud environment one cannot arbitrarily command all available resources. Therefore, different VMs can have different number of VCPUs and different amounts of RAM, and consequently can complete a specific task at different times. This necessitates two important considerations. Firstly, synchronization between VMs needs to be managed with care. Secondly, tasks have to be distributed and messages passed to VMs

in such a way as to minimize the time interval that a VM spends waiting for its next task. We will provide an example of this in Section 8.3.

(iii) **Fault Tolerance:** Unexpected errors can occur when processing data over a large region. Sometimes alignment might completely fail for a tile. Alternatively, since the terrain can vary across a 100 km^2 region, the disparity volume can become very large for some tiles or even for some specific pairs of image patches. This can lead to "out-of-memory" errors on VMs with a small amount of RAM. It is important to introduce safeguards into the framework to handle and cleanly recover from such failures. We will provide an example of this in Section 8.3.

8.2.2 Multi-Level Parallelism

Parallelism is present in our framework at multiple levels. For example, the tSGM algorithm can be parallelized at the pixel level. For a given tile, the orthorectification process can be parallelized at an image-patch level. Tiles themselves can be processed in parallel as well. Last but not the least, there is parallelism at the level of the full-sized images. An example of the latter would be ToA correction or pansharpening of the images. For each module in our framework (Fig. 7.1) we have to choose a suitable level to exploit the parallelism.

Parallel processing can either be implemented within a single VM by utilizing multiple threads or by distributing work across multiple VMs on a cloud. In our framework, parallelism within a VM is implemented using OpenMP in C++ and/or the multiprocessing module in Python and/or Bash scripts. Parallel processing across the cloud is more challenging and we will dwell upon this in greater detail in the following sections.

8.2.3 Distributed Architectures

In this section we use the term VMs and workers interchangeably. Also, applying the same computation/algorithm on different data samples (image patches, pairs of image patches, etc.) will be considered as different tasks. To distribute tasks across a cloud, we use a mix of three architectures to communicate and coordinate between the workers. The names and the designs of these architectures are inspired from the discussion in [155].

1. **Asymmetric Captain-Worker Push-Based (APSB):** There is a designated captain VM that "pushes" or distributes the tasks to the workers. Additionally, the captain also operates as a worker. Depending upon the task at hand, each worker can either report back to the captain or not, upon the successful completion of its task. This architecture can be used to distribute those modules whose runtime stays roughly the same across VMs. Otherwise powerful VMs might end up idle while waiting for other VMs to complete their tasks. Examples of such modules includes ToA correction and pansharpening. A graphic illustration of this architecture is shown in Fig. 8.3. In this illustration, we assume that there are only 3 VMs, a captain, a small VM and a large VM.

2. **Asymmetric Captain Worker Pull-Based (APLB):** In this architecture there is a designated captain VM that is in charge of preparing the list of tasks, monitoring the progress of each worker and handling failures. Additionally, the captain also operates as a worker. Each worker requests or "pulls" a task from the list. Hence, it becomes important to prevent race conditions where two workers end up pulling the same task from the list. We use the atomic nature of the UNIX "mkdir" operation to handle race conditions. What this means is that a worker can request a task if and only if it can create a directory with that task name, in a specific shared location on the NAS. There is an important gotcha to keep in mind while using this atomic "mkdir" trick to communicate between the workers in a cloud. The internal clocks of the NAS and all the

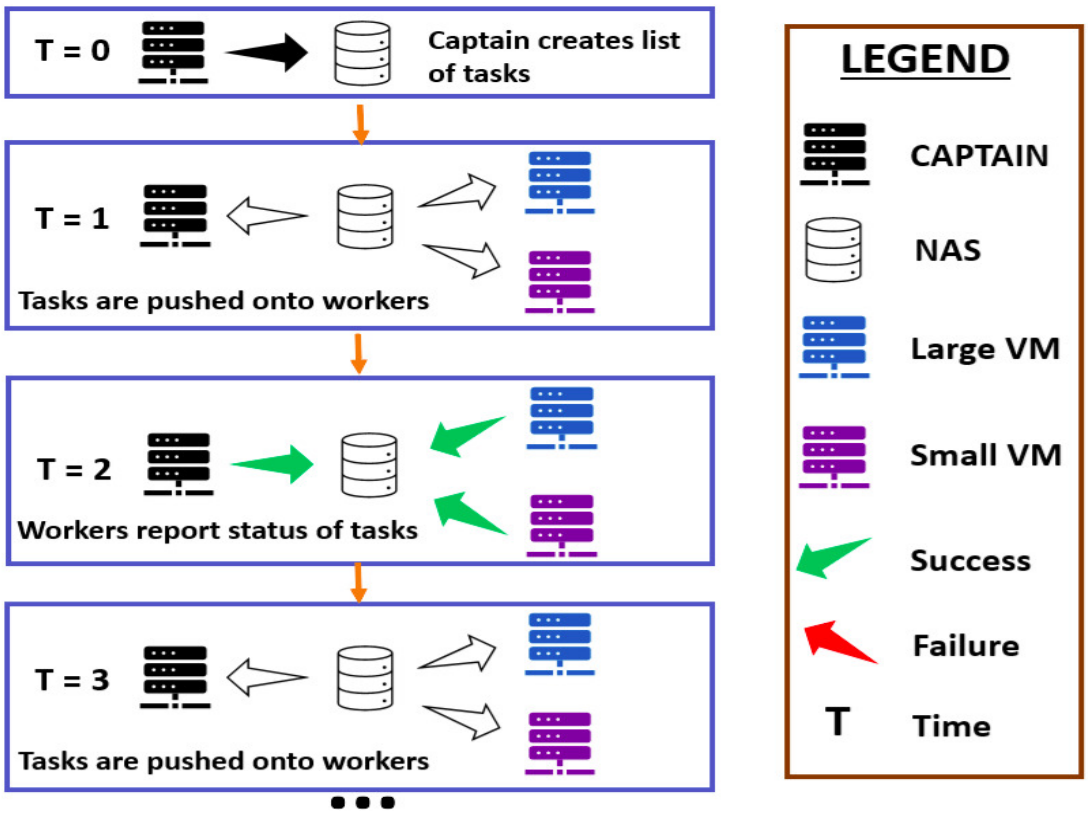


Fig. 8.3.: An example to illustrate the APSB distributed architecture. In this example, there are only 3 VMs, a captain, a small VM and a large VM. T indicates the time stamp.

VMs have to be synchronized. Otherwise one can run into unpredictable errors. Each worker also reports the completion of its task to the captain. The captain is equipped with protocols to handle failures. Depending upon the module, the captain can appropriately combine and post-process the outputs from each worker. Stereo matching and DSM creation are carried out using this architecture as explained in Section 8.3. While this architecture is more complicated than APSB, it minimizes the idle times of VMs and handles failures cleanly. A graphic illustration of this architecture is shown in Fig. 8.4. In this illustration, we assume that there are only 3 VMs, a captain, a small VM and a large VM.

3. **Symmetric Peer-to-Peer (SP2P):** As the name suggests, all the workers are peers and there is no captain. Each worker requests its task in a pull-based manner from a shared list, in a manner similar to APLB. Workers thus communicate via the atomic "mkdir" operation. Tiling, image-to-image alignment, training data generator, etc., are examples of modules that are distributed using this architecture. The runtimes for these modules can vary across tiles (or images), making them suitable candidates for this architecture.

The reader might wonder about the need to have both the APSB and the SP2P architectures. Note that APSB can be replaced with SP2P. However, for SP2P, additional care has to be taken to synchronize the internal clocks of the different components of the cloud. More importantly, in SP2P, there is no control over which task is allotted to which worker. It might happen that a smaller VM requests a very computationally-intensive or memory-intensive task, resulting in increased latency or failures.

8.3 A Distributed Architecture for Stereo Matching and DSM Creation

Creating DSMs for a 100 km^2 region is the most computationally-intensive and the slowest module in the frameworks shown in Figs. 7.1 and 7.3. It is also the module that is most likely to cause "out-of-memory" errors as explained in point (iii)

in Section 8.2.1. Therefore, we need to carefully choose some specific design attributes for this module, which we will highlight in this section.

As mentioned in Section 8.2.3, we use the APLB architecture for distributed stereo matching and DSM creation. The steps are enumerated below:

1. The captain prepares a list of the selected stereo pairs of image patches for each tile. This is done for all the tiles at the beginning. All the tiles are added to a queue. All the lists are stored on the NAS.

2. The captain sends a message to all the workers to start. The captain also assumes the role of a worker at this step.

3. For the first tile in the queue, the workers pull/request a pair of image patches to process. Each worker gets a unique pair by using the atomic "mkdir" trick explained in Section 8.2.3.

4. Each worker attempts to create a pairwise point cloud and subsequently reports the status of its task. Each worker then pull/requests the next unprocessed stereo pair for the current tile. Successfully processed stereo pairs are marked as done.

5. If there are no more unprocessed stereo pairs for this tile then:

   (i) The current tile is removed from the queue. All the idle workers, except for the captain and the large VMs, move on to the next tile in the queue, i.e., to step 3.

   (ii) All the stereo pairs for which point-cloud creation failed are processed for a second time by the remaining workers. Even if processing fails again, they are still marked as done.

6. At this stage all the selected stereo pairs for the current tile are marked as done. The large VMs join their smaller counterparts on the next tile, i.e., at step 3. The captain alone starts the process of fusing the multiple pairwise point clouds

into a single fused DSM for the current tile. After this the captain also proceeds to join the other VMs in step 3.

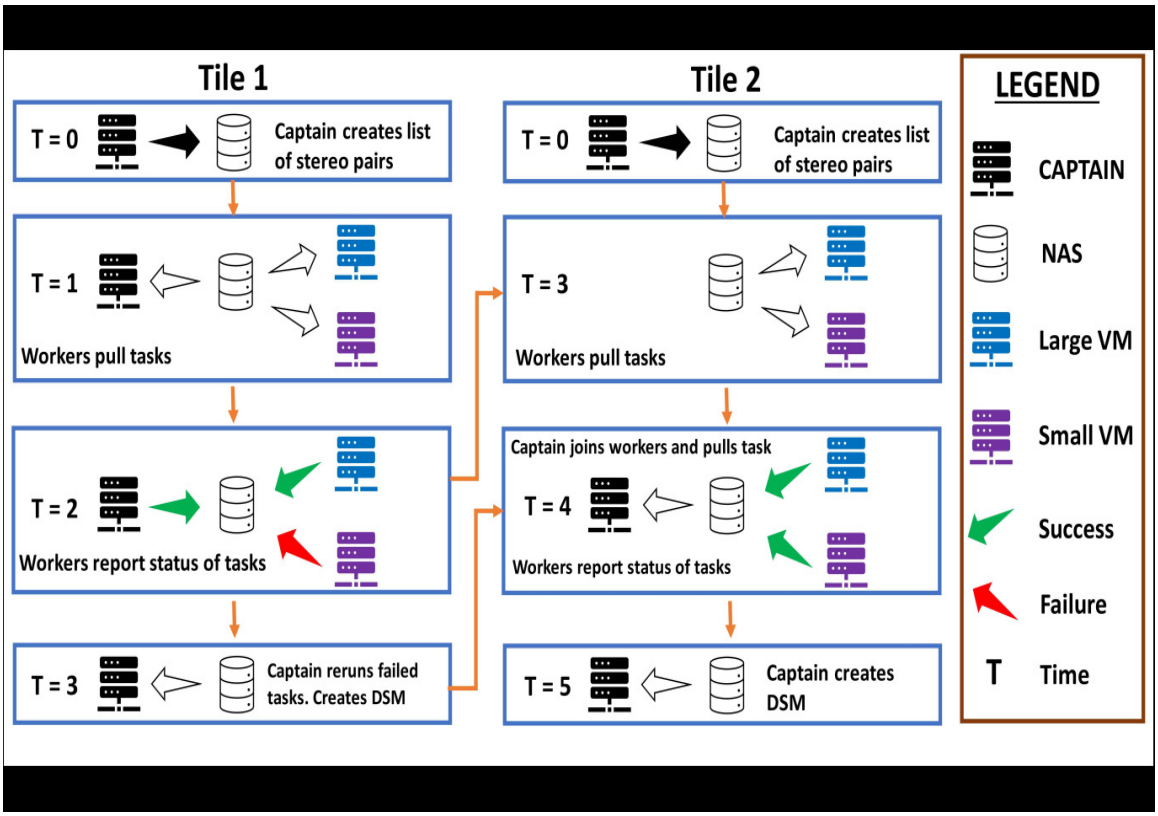


Fig. 8.4.: An example to illustrate our distributed stereo-matching and DSM-creation workflow. In this example, there are only 2 tiles and 3 selected stereo pairs for each tile. There are only 3 VMs, a captain, a small VM and a large VM. T indicates the time stamp. Notice how at T = 3, two of the VMs have moved onto Tile 2 whereas the captain stays back to finish processing Tile 1.

A graphic illustration of the above workflow is shown in Fig. 8.4. For the sake of clarity, in this illustration, we assume that there are only 2 tiles and that there are only 3 selected stereo pairs for each tile. We also assume that there are only 3 VMs, a captain, a small VM and a large VM.

8.3.1 Advantages of This Distributed Workflow

- No VM remains idle except for the last processing stage of the very last tile.

- Failed pairs are processed twice to handle "out-of-memory" errors.
- The intensive process of creating a fused DSM is carried out on the most powerful captain VM.
- Note that we could have opted to use a simple SP2P workflow where all the VMs wait for a fused DSM to be created before proceeding to the next tile. By opting for an APLB workflow over such an SP2P workflow, we can reduce the processing time by a number of days. For an example, assume that there are 10 VMs and 100 tiles. Also assume that each stereo pair takes 20 minutes to process, that we process 80 pairs per tile and that the point-cloud fusion takes 60 minutes. An SP2P architecture would take $\frac{(\frac{80\times20}{10}+60)}{60} \approx 3$ hours and 40 minutes to finish processing a single tile. For 100 tiles it would take ≈ 15 days and 6 hours. An APLB architecture takes $\frac{(\frac{80\times20}{10})}{60} \approx 2$ hours and 40 minutes for a single tile. This is because while the captain is fusing the point clouds for a tile, the other VMs will be processing the next tile. For 100 tiles it would take ≈ 11 days and 2 hours, roughly saving us 4 days of processing time.

8.4 Distributed Multi-GPU Training in PyTorch

In this section we will talk about how we train our single-view (SV) and multi-view (MV) CNNs using PyTorch [152] in a distributed fashion across multiple GPUs. We train our networks using 4 NVIDIA GeForce GTX 1080 Ti GPUs. PyTorch provides the following 3 ways to distribute an SV CNN across GPUs on the same machine:

1. **DataParallel [156]:** As the name suggests, in DataParallel the network is trained in parallel across multiple input batches. The same network weights are copied to all the GPUs. The forward pass on each GPU is done with a different batch of training samples. The total loss is summed over all the batches. The gradient updates are averaged across the GPUs. Using DataParallel is the easiest way to implement distributed training. It only requires us to change one

or two lines of code. The disadvantage is the need for repeated copying of the network weights across the GPUs. Also, it works only on a single machine.

2. **ModelParallel [157]:** ModelParallel is useful if a CNN is too large to fit onto a single GPU. We can split the network layers across GPUs and still apply backpropagation. The disadvantage is that some GPUs can remain idle while the forward pass is being executed on a different GPU.

3. **DistributedDataParallel [158]:** DistributedDataParallel is the most efficient way to do multi-GPU training on a single machine and across multiple machines. It provides us with the ability to have different processes for controlling the dataloading, the forward pass and the backpropagation operations. For a simple example, while the GPUs are running on the current batch, we could load the next batch into memory and make it ready for the GPUs. DistributedDataParallel can be used in conjunction with ModelParallel as well, to alleviate some of the shortcomings of ModelParallel.

We use DataParallel for our networks. As mentioned above, DataParallel is straightforward to use for the SV CNN. Using DataParallel for the MV CNN is more tricky and we have already elaborated in some detail about the MV DATALOAD and the MV TRAIN strategies in Chapter 6. Here we will focus on how the MV CNN is distributed across the GPUs. We apply DataParallel on the SV CNN portion of the MV CNN. This will make identical copies of the SV CNN on all the GPUs. We load the MV Fusion module (MV-A or MV-B) only onto the first GPU henceforth denoted as GPU:0. In the forward pass, the different batches are first passed through the SV CNNs on the different GPUs. The output predictions are aggregated and then passed to the MV Fusion module on GPU:0. The total loss (Eq. 6.1) is then computed and the gradients are backpropagated. **Using this architecture ensures that we can use the same batch sizes for SV TRAIN and MV TRAIN.**

We conclude this section with a small point about batch-normalization when using multiple GPUs for semantic labeling. It is a widely accepted rule of thumb that batch-

normalization works well with larger batch sizes (≥ 16). Since semantic segmentation is memory-intensive, the dimensions of our image-windows restrict us to loading only 4 windows on a single NVIDIA GeForce GTX 1080 Ti GPU when training a U-Net. With 4 GPUs, this gives us a respectable total batch size of 16. However, currently PyTorch does not allow us to synchronize the batch-normalization statistics across GPUs when using DataParallel. We did experiment with replacing PyTorch's BatchNorm2d layers [159] with third-party synchronized batch-norm layers from [160] but did not observe any noticeable improvement in the IoUs.

9. EXPERIMENTAL EVALUATION

In this chapter we describe the quantitative and qualitative results of several experiments that we have conducted using our framework. This chapter is split into two parts. In the first part, we describe the datasets that we use, evaluate the alignment algorithm and the DSMs, and establish a baseline for the semantic-labeling accuracy using a U-Net that has been trained in single-view mode. We also conduct ablation studies to test the importance of the different components of the geo-processing module (Figs. 7.1 and 7.2) on the final segmentation IoUs. The second part of this chapter focuses on the multi-view training and inference framework. We carry out different ablation studies and comparisons to understand how the multi-view training helps alleviate the noise in the training labels.

9.1 Description of the Datasets Used

We use two datasets to evaluate the different components of our framework. The first dataset consists of 36 WV3 PAN images and the corresponding 36 MS images, covering a 120 km^2 region in Ohio. All the images come with RPCs and the metadata needed for ToA correction. However, due to an error by the satellite vendor, 4 of the MS images contain only 4 bands instead of 8. Nonetheless, since the DSMs are constructed using PAN images, we use all 36 PAN images for the same. For semantic labeling, we use the 32 eight-band pansharpened images. The building and road label data are downloaded from the OSM website. We also download the SRTM DEM [11] that has a GSD of 30 m. *No other preprocessing is done before feeding the data to our framework.*

The second dataset consists of 43 WV3 PAN and 43 MS images covering a 150 km^2 region in California. 35 of these 43 images are part of the publicly-available Spacenet

[1] repository. We evaluate our alignment algorithms and our DSM-construction algorithms using the California dataset as well.

Figs. 9.1 and 9.2 respectively summarize some important statistics of the satellite images that belong to the Ohio and California regions. Figs. 9.1a and 9.2a respectively show the image-acquisition times for the Ohio and California regions. Figs. 9.1b and 9.2b respectively display the histograms of the obliquity angles at which the images were captured, for the Ohio and California regions. Figs. 9.1c and 9.2c respectively depict the histograms of the differences between the acquisition times of the images, computed on a pairwise basis, for the Ohio and California regions. Figs. 9.1d and 9.2d respectively show the histograms of the differences between the view angles of the images, computed on a pairwise basis, for the Ohio and California regions. Fig. 9.1e portrays the 2D histogram of the pairwise view-angle differences and the pairwise acquisition-time differences of the images that belong to the Ohio region and Fig. 9.2e portrays the same for the California region.

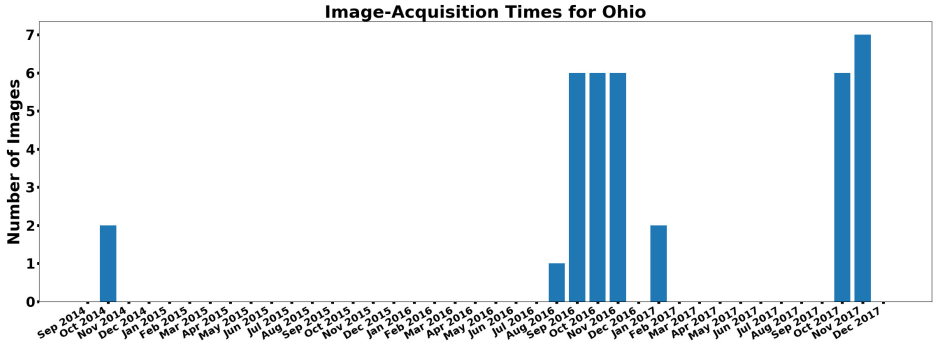
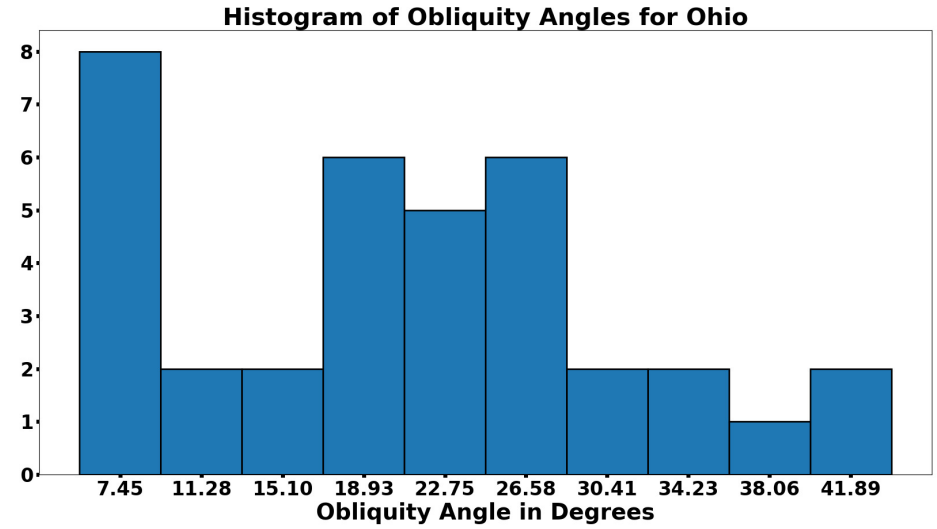
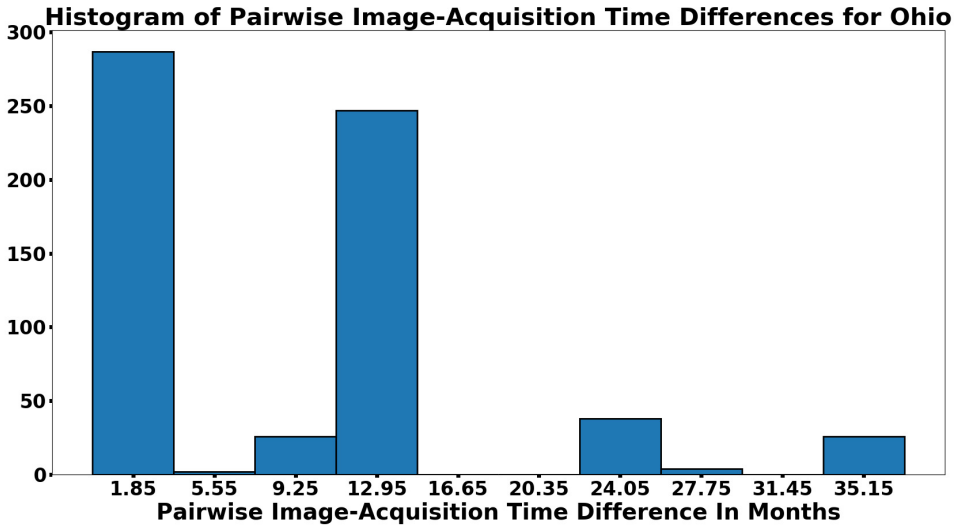


Image-Acquisition Times for Ohio
Number of Images
0
1
2
3
4
5
6
7
Sep 2014
Oct 2014
Nov 2014
Dec 2014
Jan 2015
Feb 2015
Mar 2015
Apr 2015
May 2015
Jun 2015
Jul 2015
Aug 2015
Sep 2015
Oct 2015
Nov 2015
Dec 2015
Jan 2016
Feb 2016
Mar 2016
Apr 2016
May 2016
Jun 2016
Jul 2016
Aug 2016
Sep 2016
Oct 2016
Nov 2016
Dec 2016
Jan 2017
Feb 2017
Mar 2017
Apr 2017
May 2017
Jun 2017
Jul 2017
Aug 2017
Sep 2017
Oct 2017
Nov 2017
Dec 2017

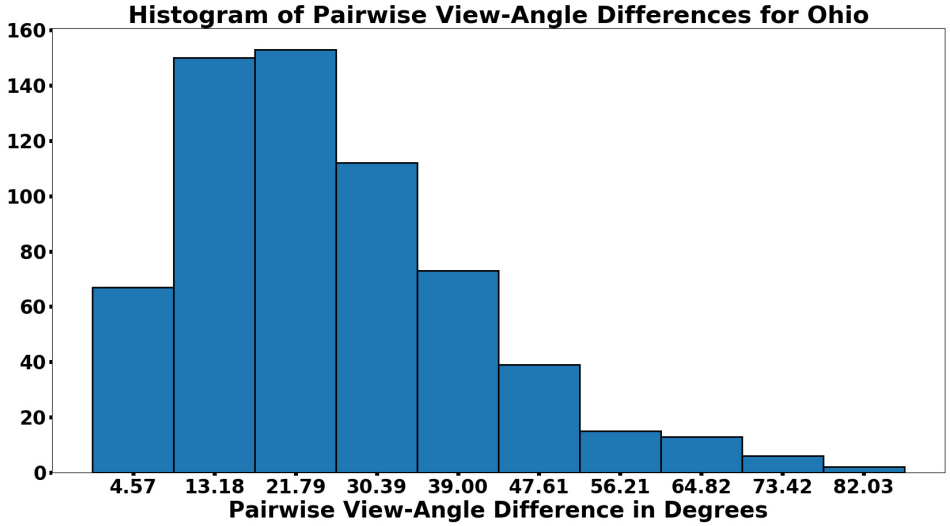
(a)


Histogram of Obliquity Angles for Ohio
0
1
2
3
4
5
6
7
8
7.45
11.28
15.10
18.93
22.75
26.58
30.41
34.23
38.06
41.89
Obliquity Angle in Degrees

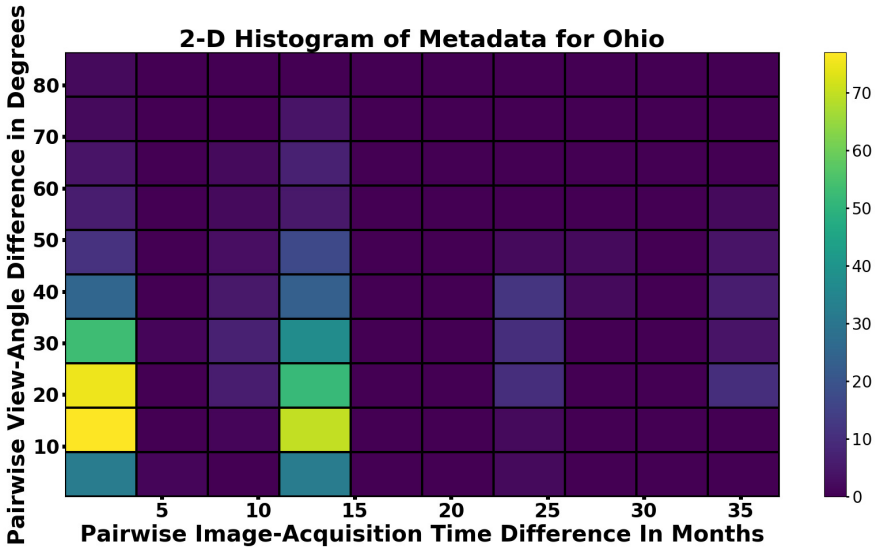
(b)


Histogram of Pairwise Image-Acquisition Time Differences for Ohio
300
250
200
150
100
50
0
1.85
5.55
9.25
12.95
16.65
20.35
24.05
27.75
31.45
35.15
Pairwise Image-Acquisition Time Difference In Months

(c)


Histogram of Pairwise View-Angle Differences for Ohio
160
140
120
100
80
60
40
20
0
4.57
13.18
21.79
30.39
39.00
47.61
56.21
64.82
73.42
82.03
Pairwise View-Angle Difference in Degrees

(d)



(e)

Fig. 9.1.: Summary of important statistics of the satellite images for Ohio

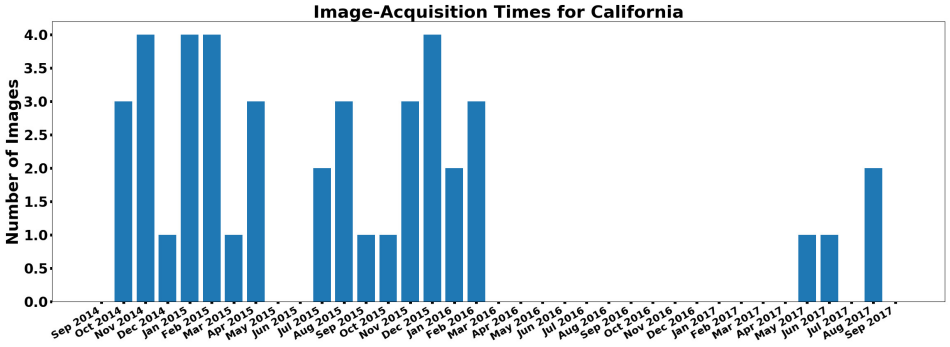
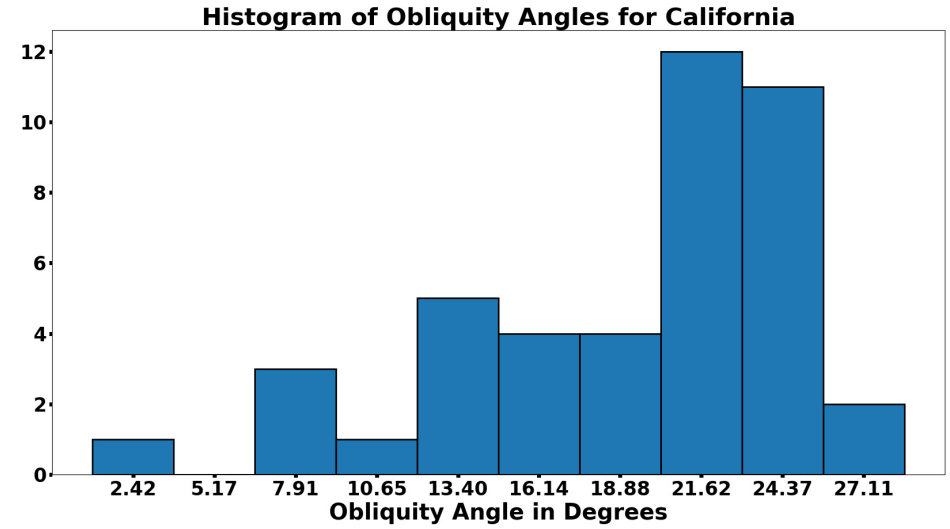
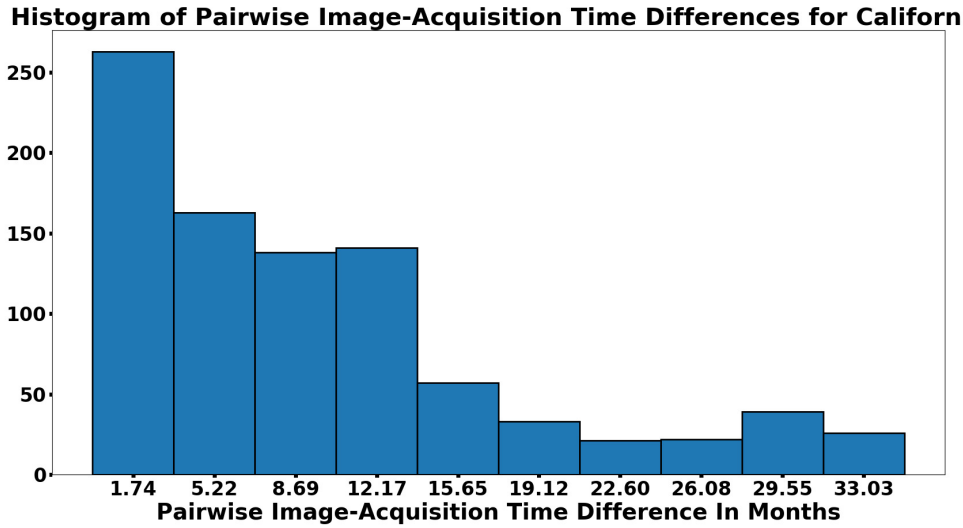


Image-Acquisition Times for California
Number of Images
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Sep 2014
Oct 2014
Nov 2014
Dec 2014
Jan 2015
Feb 2015
Mar 2015
Apr 2015
May 2015
Jun 2015
Jul 2015
Aug 2015
Sep 2015
Oct 2015
Nov 2015
Dec 2015
Jan 2016
Feb 2016
Mar 2016
Apr 2016
May 2016
Jun 2016
Jul 2016
Aug 2016
Sep 2016
Oct 2016
Nov 2016
Dec 2016
Jan 2017
Feb 2017
Mar 2017
Apr 2017
May 2017
Jun 2017
Jul 2017
Aug 2017
Sep 2017

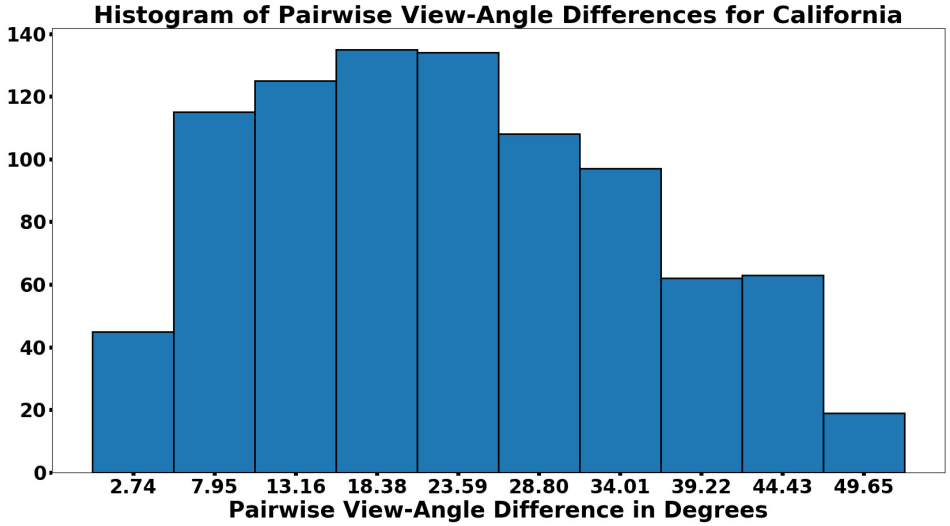
(a)


Histogram of Obliquity Angles for California
0
2
4
6
8
10
12
2.42
5.17
7.91
10.65
13.40
16.14
18.88
21.62
24.37
27.11
Obliquity Angle in Degrees

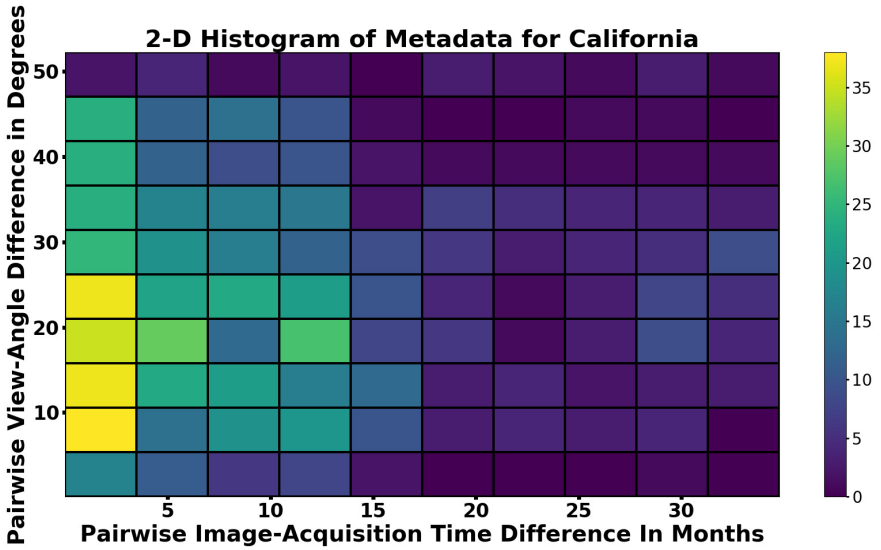
(b)


Histogram of Pairwise Image-Acquisition Time Differences for California
250
200
150
100
50
0
1.74
5.22
8.69
12.17
15.65
19.12
22.60
26.08
29.55
33.03
Pairwise Image-Acquisition Time Difference In Months

(c)


Histogram of Pairwise View-Angle Differences for California
140
120
100
80
60
40
20
0
2.74
7.95
13.16
18.38
23.59
28.80
34.01
39.22
44.43
49.65
Pairwise View-Angle Difference in Degrees

(d)



(e)

Fig. 9.2.: Summary of important statistics of the satellite images for California

ToA correction, pansharpening, image-to-image alignment, DSM construction, orthorectification and training-data generation are carried out using multiple virtual machines running on the RVL Cloud (Chapter 8).

9.2 Quantitative Evaluation of Alignment

9.2.1 Within-Tile Alignment

We use multiple metrics to evaluate the quality of our image-to-image alignment algorithms. Table 9.1 shows the average reprojection error across tiles (and images) for both regions, before and after alignment. The average reprojection error goes down from 5-7 pixels to about 0.3 pixels for both the regions.

Table 9.1.: Average reprojection error in pixels across tiles and images in Ohio and California

Region		Mean	Variance
Ohio	Unaligned	6.70	0.18
	Aligned	**0.30**	0.003
California	Unaligned	5.71	0.28
	Aligned	**0.32**	0.001

Table 9.2.: Pairwise alignment-error statistics using manually-annotated groundtruth for Ohio and California

Region	No. of images	No. of pairs with error < 1 pixel	No. of pairs with error < 2 pixels	Total No. of pairs
Ohio	36 (35 annotated)	532	582	595
California	43 (42 annotated)	846	861	861

Table 9.3.: Pairwise alignment-error statistics using manually-annotated groundtruth for subsets of the images from Ohio and California

Region	No. of images	No. of pairs with error < 1 pixel	No. of pairs with error < 2 pixels	Total No. of pairs
Ohio	32	417	455	465
California	35	571	595	595

The task of stereo-matching requires subpixel-level alignment accuracy. However, since the pushbroom sensors can be closely approximated by affine cameras with parallel rays (Section 3.3.4.1), the reprojection error alone does not give the complete picture. Therefore, to evaluate the alignment using a second metric, we manually annotate tie points in 35 out of 36 images over a 1 km^2 region in Ohio and in 42 out of 43 images over a 1-2 km^2 region in California. Within these regions, we measure

the pairwise alignment errors[1] for all possible pairs of images and report them in Table 9.2. One can observe that most of the pairs are aligned with subpixel error. This is a much harder metric than the average reprojection error. We also show the pairwise alignment error for the 32 PAN images from Ohio whose corresponding MS images have 8 bands each, and for the 35 images from California that are part of the publicly-available Spacenet [1] repository, in Table 9.3. The good quality of alignment across the large region is also reflected in the high quality of the DSM and the semantic-labeling metrics.

9.2.2 Between-Tile Alignment

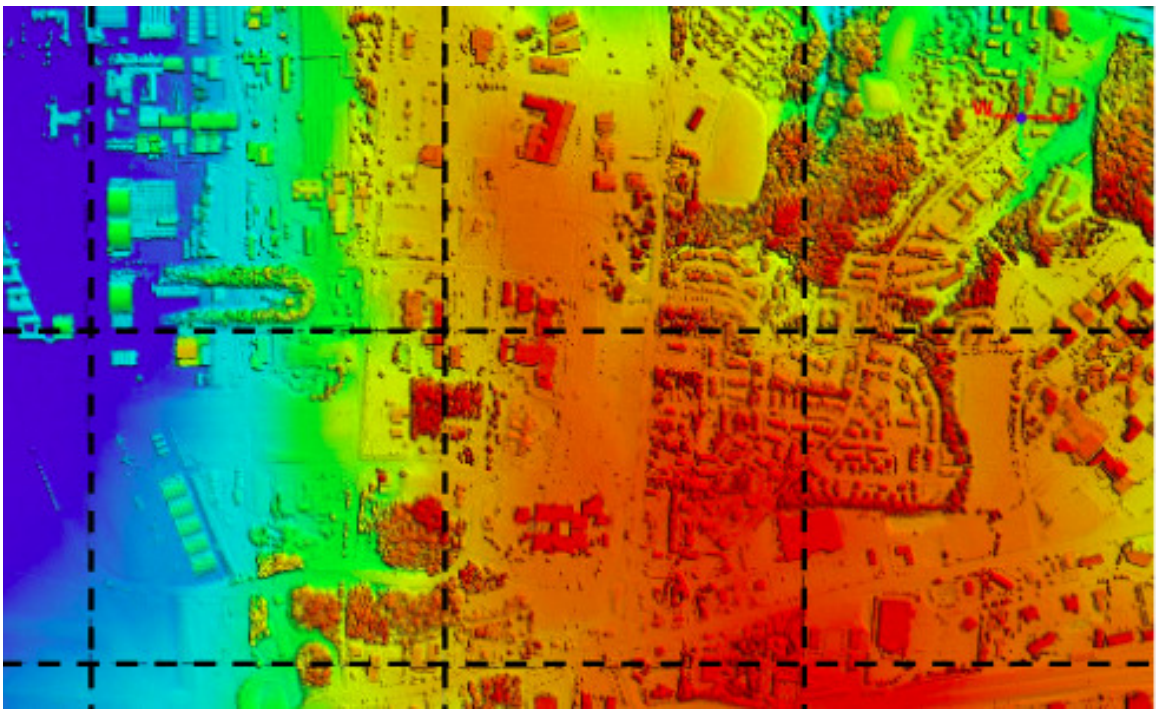


Fig. 9.3.: The tile-level DSMs are colored using the elevation values and the tile boundaries are marked with dashed lines. The continuity of the color across the tile boundaries is a good indication of how well the tile-level DSMs align at their boundaries.

[1]The code for evaluating the pairwise alignment errors was developed by Dr. Tanmay Prakash

Since the image patches of each tile are aligned independently, we initially thought that there would be a need to align the tiles with one another and we proposed two different strategies for the same in Section 4.2. However, as mentioned in Section 4.3, on account of the high absolute alignment precision achieved by using the L_2 regularization term (Eq. 3.2) in the bundle adjustment logic, it turns out that nothing further needs to be done for merging the tile-level DSMs into a larger DSM for our datasets. For a visual proof, Fig. 9.3 shows six tile-level DSMs from Ohio. The tile-level DSMs are colored using the elevation values and the tile boundaries are marked with dashed lines. The continuity of the color across the tile boundaries is a good indication of how well the tile-level DSMs align at their boundaries. For a quantitative evidence, the statistics of the differences between the elevations in the overlapping portions of adjacent tiles are shown in Table 9.4. We see that the median absolute elevation difference in the overlapping portions is less than 0.5 m – an error that is much too small to introduce noticeable errors in orthorectification. Therefore, we crop out the center 1 km^2 region from each tile-level DSM and place it in the coordinate frame of the larger DSM. This sidesteps the need to resolve any noise-induced variations in the overlapping regions.

Table 9.4.: Median of the absolute differences in elevation, and median of the RMS value of the differences in elevation, at the overlapping portions of adjacent tiles

Region	Median absolute Z diff	Median RMS of Z diff
Ohio	0.42 m	0.72 m
California	0.47 m	0.79 m

9.3 Qualitative Evaluation of DSMs

We constructed a DSM covering an area of about 120 km^2 in Ohio and a DSM covering an area of about 62 km^2 in California. In this section we provide qualitative evidence for the high quality of these DSMs produced by our framework. In addition

to these static images, the reader is encouraged to look at the flyby videos "Ohio-DSM-flyby.mp4" and "California-DSM-flyby.mp4" [161] for a better visualization of the quality of the DSMs.

9.3.1 Image of a Large-Area DSM (120 km^2) in Ohio

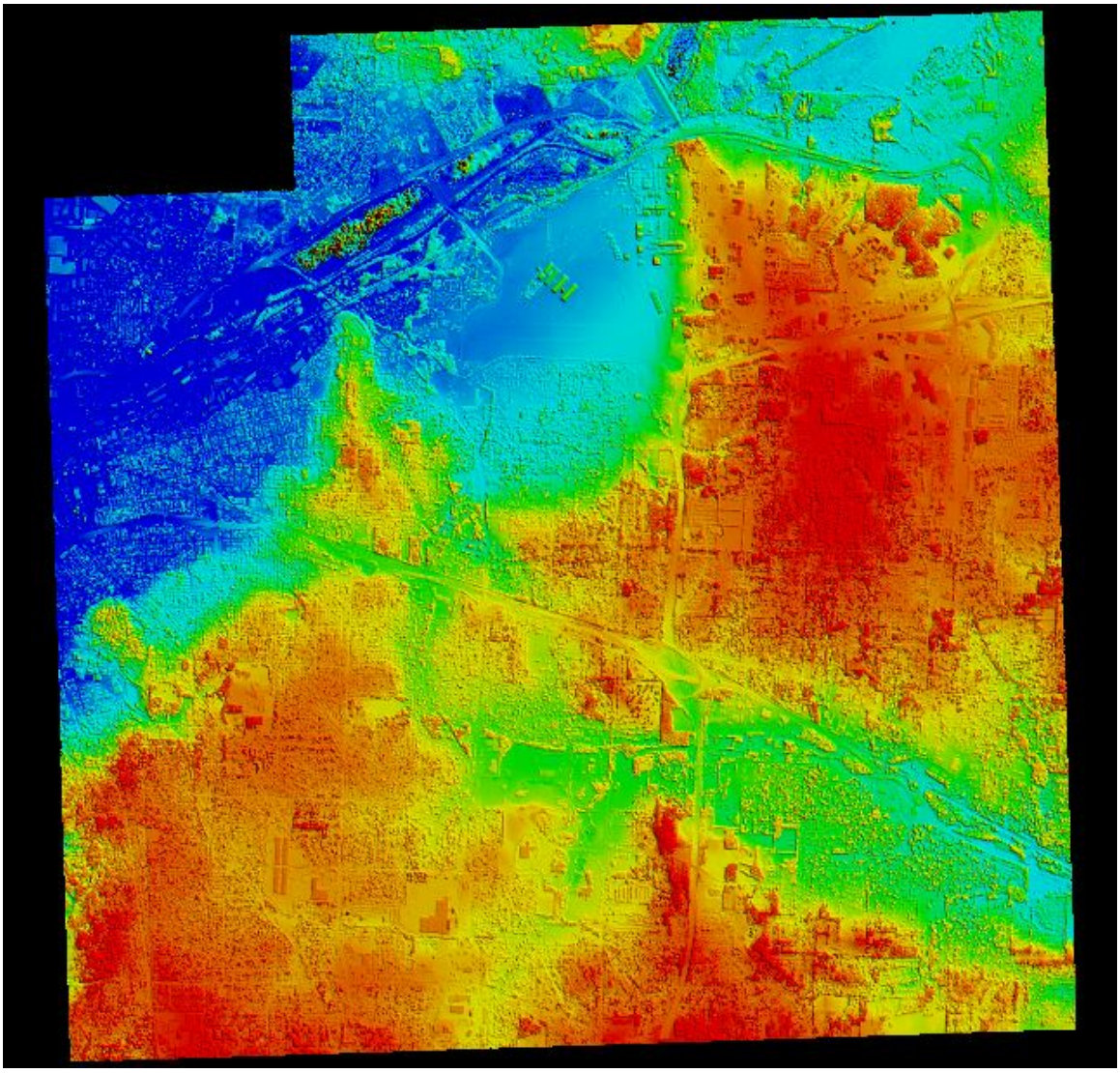


Fig. 9.4.: Ortho view of the full-sized DSM for the area from Ohio covering 120 km^2. The DSM depiction has been colored according to the elevation values. For viewing the colors in this figure, the reader is referred to the web version of this article.

9.3.2 Two Zoomed-In Sections from the Ohio DSM

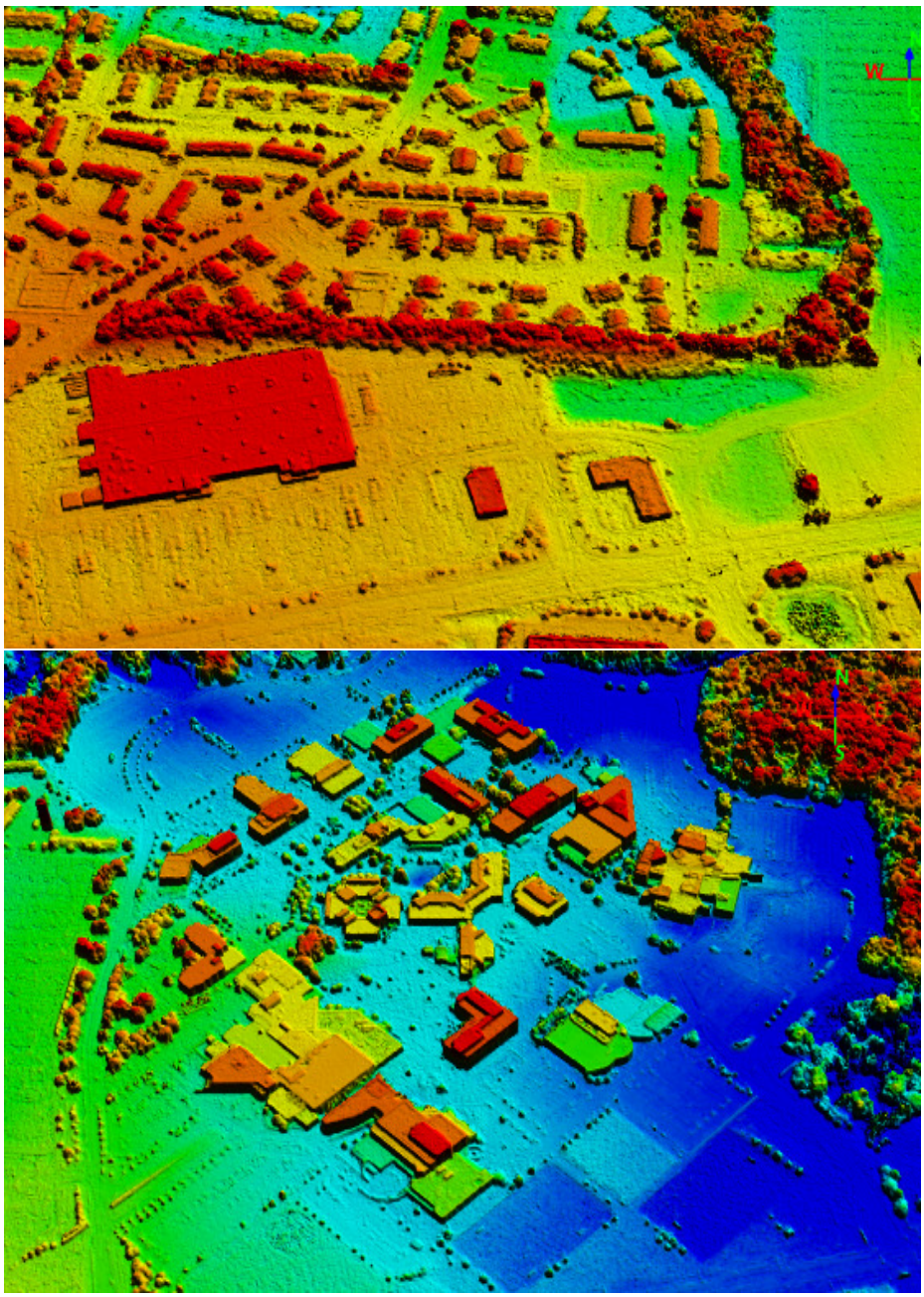


Fig. 9.5.: Two zoomed-in sections of the full-sized DSM for Ohio that is shown in Fig. 9.4. *The DSM depictions have been colored according to the elevation values within the boundaries of each section.* For viewing the colors in this figure, the reader is referred to the web version of this article.

9.3.3 Image of a Large-Area DSM (62 km^2) in California

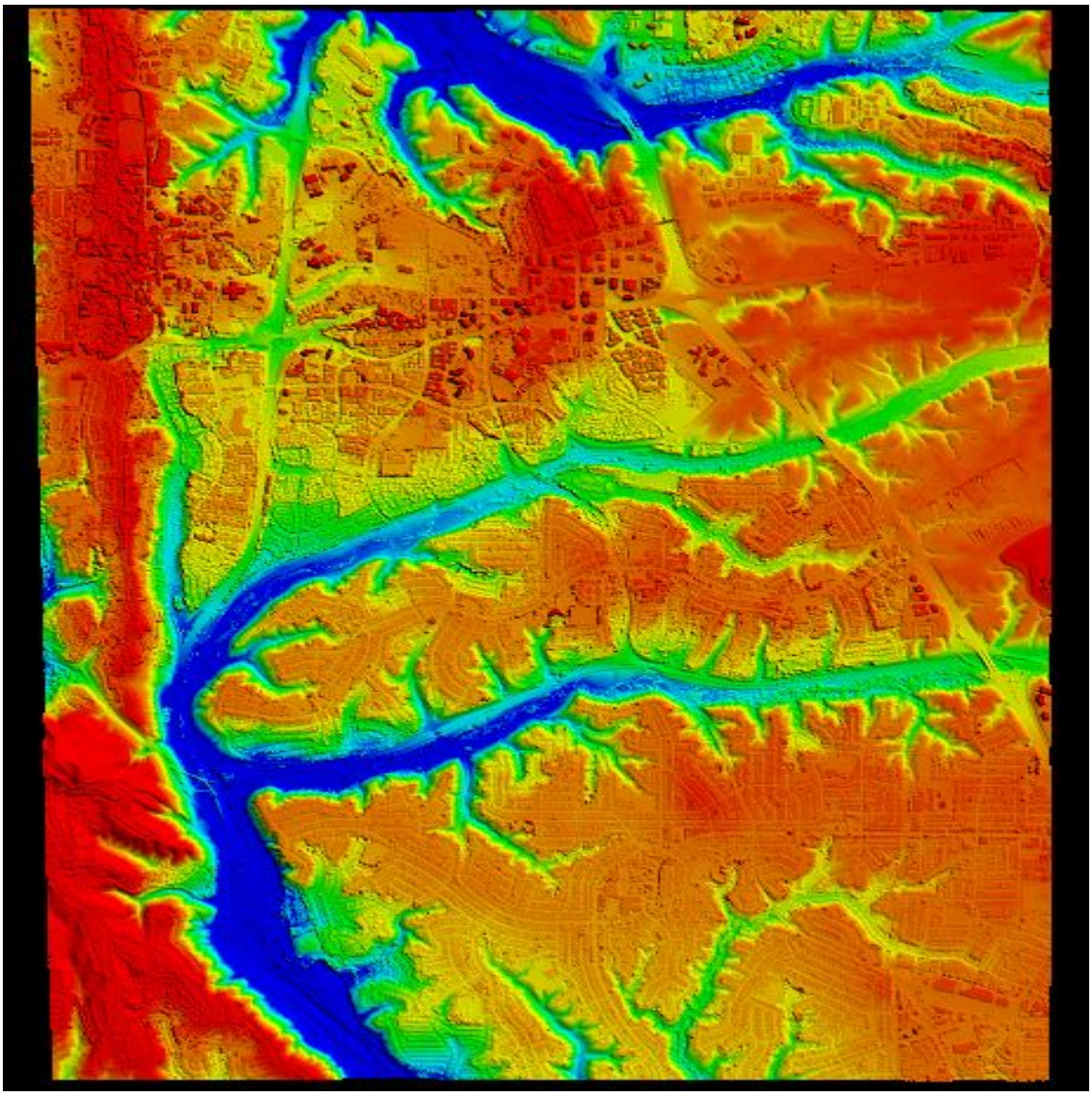


Fig. 9.6.: Ortho view of the full-sized DSM for the area from California covering 62 km^2. The DSM depiction has been colored according to the elevation values. For viewing the colors in this figure, the reader is referred to the web version of this article.

9.3.4 Two Zoomed-In Sections from the California DSM

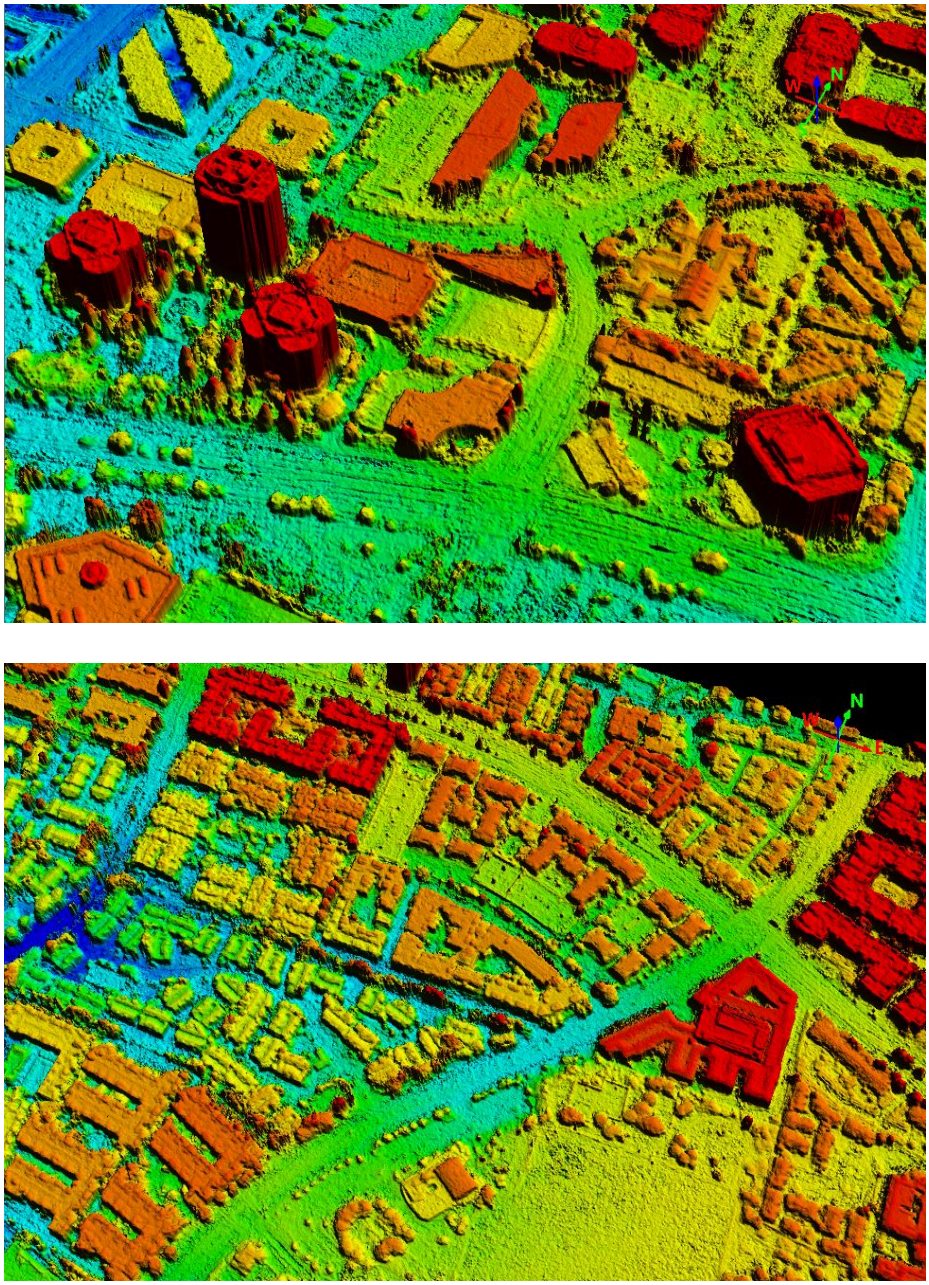


Fig. 9.7.: Two zoomed-in sections of the full-sized DSM for California that is shown in Fig. 9.6. *The DSM depictions have been colored according to the elevation values within the boundaries of each section.* For viewing the colors in this figure, the reader is referred to the web version of this article.

9.4 Quantitative Evaluation of the Single-View Semantic Labeler

In this section we focus on the following issues:

1. We want to get a sense of how well a single-view CNN can learn to predict labels in the presence of different kinds of noise such as the errors in the DSMs, alignment errors and incorrect OSM annotations. These numbers will serve as a baseline to which we can compare our multi-view training strategy.

2. For many traditional computer-vision applications, it has been shown that CNNs can perform well even in the presence of different illumination conditions. So is ToA correction even necessary? Maybe the CNN can learn to apply suitable filters to the 8-band images, thereby rendering ToA correction moot.

3. How important is it to align the images? In other words, how robust are CNNs to the image-alignment errors? Note that without aligning the images we cannot create a DSM. However, we could merely orthorectify the images using a DEM, without aligning them. While this process will not create true ortho images, such a study can still give serve as a useful baseline.

To address the first point in the aforementioned list, we carry out an extensive quantitative assessment of the trained CNN. To do this, we divide the 120 km^2 region in Ohio into a 109 km^2 region for training, a 1 km^2 region for validation, and an unseen 10 km^2 region for inference. The last region is "unseen" because no samples in the training and the validation regions fall inside that region. The unseen region is further divided into sub-regions – a) with precise manual annotations for a quantitative evaluation and b) without annotations for a qualitative assessment of the results. The buildings were carefully annotated over sub-regions that together cover a 2 km^2 area while the roads were carefully annotated over sub-regions that together cover a 1 km^2 area. We also show qualitative results from the unseen region.

To address the second and third points in the aforementioned list, we carry out ablation studies by turning off the appropriate modules of our framework. *This is a*

rather unique contribution of this study. The term "ablation studies" is usually used in the context of deep learning to refer to studies that tease out the importance of different layers in a neural network by selectively turning those layers off/on. We extend this term to refer to more general studies where we turn off entire modules of our framework (Fig. 7.1) and directly evaluate the impact on the final segmentation metrics. Since our framework is modular and end-to-end automated, it is possible for us to directly evaluate the net impact of a module on the semantic-labeling accuracy.

As mentioned in Chapter 5, we select the popular U-Net architecture [65] as the CNN because it is lightweight and has been used in many prior studies with overhead imagery [69], [126], [130]. To assist the reader, we reproduce here some of the details from Chapter 5. The U-Net is modified to accept 8-band data, and we add batch-normalization layers. Since the OSM labels are sparse, we use weighted cross-entropy losses with the weights set to 0.2, 0.4 and 0.4 for the background, building and road classes respectively. For training, we initialize the weights of the CNN randomly. Additionally, instead of using bilinear interpolation, we learn the weights of the upsampling filter in the decoder part of the CNN. The learning rate is reduced gradually as the network converges. We use stochastic gradient descent (SGD) with momentum, as the optimization routine. Training is distributed across 4 NVIDIA GTX 1080 Ti GPUs as described in Section 8.4. For inference, we save the learned weights in the U-Net for the epoch with the best mean IoU on the validation samples.

9.4.1 Semantic-Labeling Metrics for the Single-View CNN: A Closer Look

We use the precision, recall, IoU and F1-score to evaluate the CNN. The F1-score is defined as the geometric mean of the precision and recall, i.e.,

$$\text{F1-score} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{9.1}$$

The precision, recall, IoU and F1-score for buildings and roads in Ohio are presented in Table 9.5. We report the numbers using two different strategies for merging the predictions from the overlapping image patches, as explained in Section 5.6.1. To remind the reader, for Maj-Vote we decide the final label for a point by taking a majority vote of the predictions (for that point) from the overlapping image patches, and for Avg-Soft we average the per-point predicted probabilities (soft scores) and convert the averaged probability to a hard label. For both buildings and roads, Maj-Vote and Avg-Soft perform comparably.

Table 9.5.: Semantic-labeling metrics for buildings and roads in Ohio using a single-view U-Net. Maj-Vote stands for majority voting and Avg-Soft stands for averaging the soft scores (predicted probabilities).

Class	Metric	Maj-Vote	Avg-Soft
Buildings	Precision	0.92	0.92
	Recall	0.80	0.80
	IoU	0.75	0.75
	F1-score	0.86	0.86
Roads	Precision	0.65	0.66
	Recall	0.81	0.81
	IoU	0.57	0.57
	F1-score	0.72	0.73

We report the relaxed precision, relaxed recall [110], relaxed F1-score and relaxed IoU in Table 9.6. These metrics are commonly used when using noisy labels for training. Similar to the studies described in [110] and [69], the relaxation factor is set to a distance of 3 meters. Thus for example, when calculating the relaxed precision, any predicted foreground point that lies within 3 meters of a groundtruth foreground point will count as a true positive. The other metrics are relaxed in a similar fashion. Both Maj-Vote and Avg-Soft perform comparably.

To compare our results with some previous studies that use aerial images and OSM data, we compute the break-even point [110] which is defined as the point on the relaxed precision-recall curve where its precision value equals its recall value. Our

Table 9.6.: Relaxed semantic-labeling metrics for buildings and roads in Ohio using a single-view U-Net. Maj-Vote stands for majority voting and Avg-Soft stands for averaging the soft scores (predicted probabilities).

Class	Relaxed Metric	Maj-Vote	Avg-Soft
Buildings	Relaxed-Precision	0.97	0.97
	Relaxed-Recall	0.90	0.90
	Relaxed-IoU	0.80	0.80
	Relaxed-F1-score	0.94	0.93
Roads	Relaxed-Precision	0.72	0.73
	Relaxed-Recall	0.97	0.97
	Relaxed-IoU	0.62	0.63
	Relaxed-F1-score	0.83	0.83

break-even points for buildings and roads are 0.96 and 0.87 respectively, which are in the ballpark of the corresponding numbers reported in [110] and [69]. Our F1-score for the buildings is better than the F1-score reported in [115]. Our IoU for the buildings is comparable to what is reported in [46] for satellite images where the focus is solely on buildings. Even for the case of just buildings, that study only consider 1-2 km^2 of urban landscapes. It uses the OSM data to eliminate the confounding effects of bridges and elevated roads from the final predictions. In contrast, we train a U-Net on noisy data across large regions and do no further post-processing. Although these other studies use their own datasets, note that they do not use end-to-end automation. It is still heartening to see that our baseline numbers are in the same ballpark as in these other studies. The system is not completely overwhelmed by the different sources of noise.

Note: Since Maj-Vote and Avg-Soft perform comparably in the aforementioned cases, in all the experiments described in the rest of this chapter, we apply Maj-Vote at inference time when using a CNN that has been trained in single-view mode, to combine the predictions from overlapping image patches.

9.4.2 Ablation Studies

In the beginning of Section 9.4, we had mentioned that we want to directly measure the net impact that ToA correction and image-to-image alignment have on the semantic-labeling metrics. To do this we carry out two different ablation studies resulting in the following three methods for us to compare:

1. **Method 1 (M1):** There is no ToA correction and no image-to-image alignment. Consequently there is no DSM construction and no OSM alignment. The images are orthorectified using their uncorrected RPCs and the SRTM DEM.

2. **Method 2 (M2):** We apply ToA correction but no image-to-image alignment. Consequently there is no DSM construction and no OSM alignment. The images are orthorectified using their uncorrected RPCs and the SRTM DEM.

3. **Method 3 (M3):** We activate all the modules in our framework.

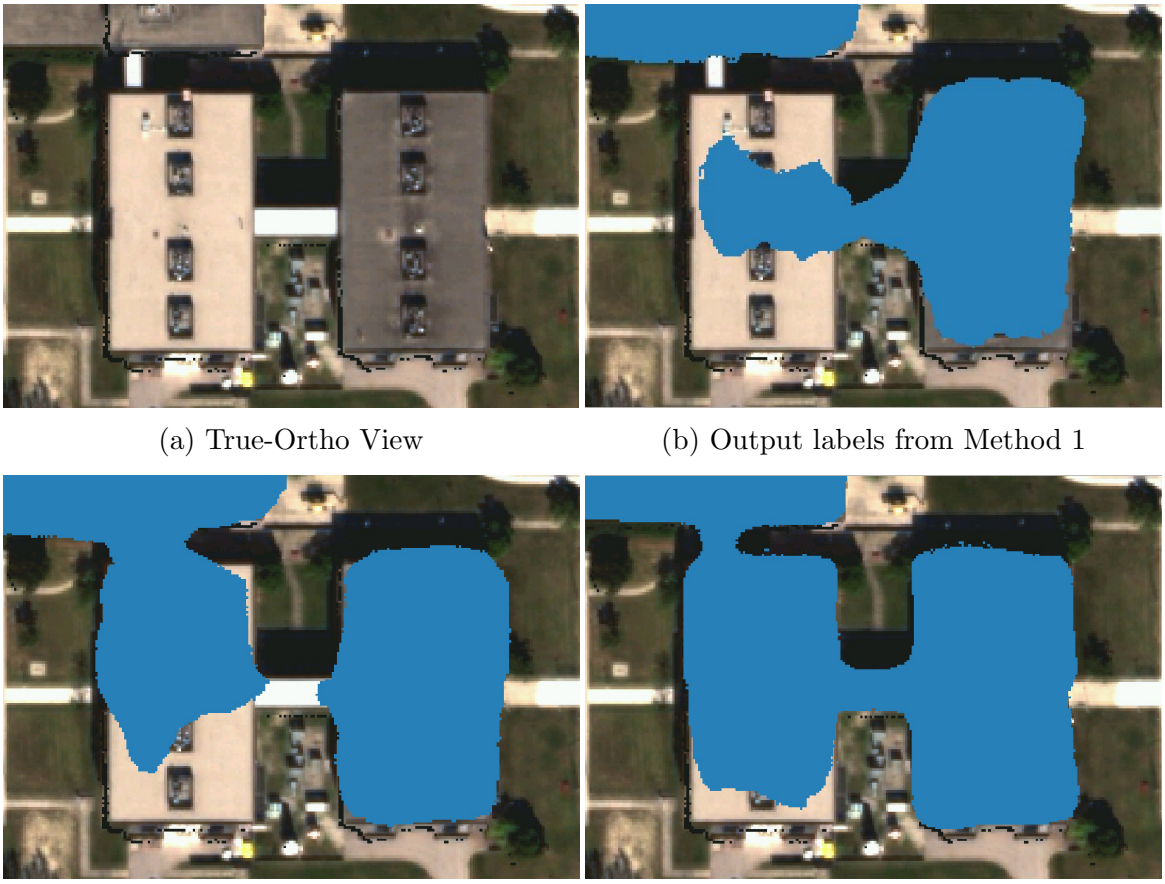
Table 9.7.: Results of the ablation studies to investigate the impact of different modules on the semantic-labeling accuracy in Ohio. M1, M2 and M3 are the three methods described above.

Class	Method	F1	IoU
Buildings	M1	0.81	0.68
	M2	0.83	0.71
	M3	0.86	0.75
Roads	M1	0.68	0.55
	M2	0.73	0.58
	M3	0.72	0.57

In Table 9.7, we show the IoU and F1-score for all the three methods described above. Comparing the entries for M1 and M2 in this table, we see that applying ToA correction to the images improves the F1-score and IoU by 2% and 3% respectively for the building class and by 5% and 3% respectively for the road class. This is quite

illuminating as it reveals that the CNN is unable to learn filter weights that can compensate for the variations in the solar zenith and the Earth-Sun distance, at the time of image-capture.

Comparing the entries for M2 and M3 in Table 9.7, we see that the F1-score goes up by an additional 3% while the IoU goes up by an additional 4% for the building class when we activate the image-to-image alignment and DSM-creation components of the geo-processing module (Figs. 7.1 and 7.2). However, for the road class, we do not see a similar improvement and in fact, the IoU and F1-score go down by 1%. Here is a possible explanation for this decrease. Firstly, the noise in the OSM road labels is much greater than the noise in the building labels because we assume a constant-width for all the roads and because the OSM annotations are often not along the true centers of the roads. Secondly, if the image-to-image alignment process was doing a poor job, then it would have negatively affected the metrics for the building class as well, which is clearly not the case. This seems to point to the DSMs (and consequently orthorectification) as the source of the problem. This is not a bad hypothesis because, in general, the calculated disparities and the DSMs are more accurate for elevated objects such as buildings when compared to the terrain. The terrain of the DSM (which has a GSD of 0.25 m) does not change smoothly. Hence, orthorectification with a DSM introduces greater noise for the road and ground points than for the building points. In the case of M2, orthorectification is done with a smoother and coarser DEM (with a GSD of 30 m) in which the elevation of the terrain changes more smoothly.



(a) True-Ortho View (b) Output labels from Method 1

(c) Output labels from Method 2 (d) Output labels from Method 3

Fig. 9.8.: An example highlighting the improvements produced by the different modules of the framework. Methods 1-3 refer to the ablation studies as defined in the beginning of Section 9.4.2. The predicted building labels are marked in blue.

In Figure 9.8, we show some qualitative differences between the predicted labels from all the three ablation studies. Comparing figures 9.8b and 9.8c, we can see that due to ToA correction, more building points are labeled correctly. Using all the modules of the framework significantly improves the quality of the building labels as seen in Figure 9.8d.

9.5 Evaluation of the Multi-View Training and Inference Framework

Now that we have established a baseline for the performance of a single-view CNN trained in the presence of the different sources of noise, we would like to study the performance of the multi-view training and inference framework described in Chapter 6. Please note that due to GPU memory constraints, the parameter $|Q|$ for MV DATALOAD is set to 16 when using a U-Net as the SV CNN.

9.5.1 Single-View vs Multi-View CNNs

We carry out multiple experiments with different combinations of training strategies and inference models. The relevant notations and definitions have already been defined in Chapter 6. Nevertheless, to assist the reader, we will explain the notation used in the tables below with a couple of examples. Please note that while counting the rows in any of the tables below, we do not include the rows that contain the names of the columns. Thus, for instance, the first row in Table 9.8 refers to the row that contains the entry for "SV CNN + VOTE (Baseline)".

Consider the first row in Table 9.8. In this entry, "SV CNN + VOTE" indicates that we are using the Single-View CNN and that the predicted labels are merged using majority voting. "SV TRAIN" indicates that we train the network using the "SV TRAIN" strategy. "EPOCH-MIN-VAL" indicates that we run inference with the network weights that correspond to the epoch with the minimum validation loss and maximum validation IoU. In a similar vein, the eleventh row of Table 9.8 reports the performance obtained by training the SV CNN + MV-A net using the MV TRAIN-II strategy, and subsequently running inference using the network weights that correspond to an epoch with a very small training loss and a validation IoU that is close to the maximum validation IoU, i.e., EPOCH-MIN-TRAIN.

Since the reader might be overwhelmed by the number of entries in Table 9.8, in the rest of this discussion we will use additional tables whose entries are subsets of the entries in Table 9.8.

Table 9.8.: Comparison of the segmentation IoUs for different combinations of training strategies and inference models

CNN	Training Strategy	Inference Model	IoU	
			Buildings	Roads
SV CNN + VOTE (Baseline)	SV TRAIN	EPOCH-MIN-VAL	0.75	0.57
SV CNN + DATA-AUG + VOTE	SV TRAIN	EPOCH-MIN-VAL	0.76	0.55
SV CNN + MV-B	MV TRAIN-I	EPOCH-MIN-VAL	0.75	0.57
SV CNN + MV-A	MV TRAIN-II	EPOCH-MIN-VAL	**0.79**	0.55
SV CNN + MV-B	MV TRAIN-II	EPOCH-MIN-VAL	**0.80**	0.57
SV CNN + MV-A	MV TRAIN-III	EPOCH-MIN-VAL	0.63	0.59
SV CNN + MV-B	MV TRAIN-III	EPOCH-MIN-VAL	0.62	0.56
SV CNN + VOTE (Baseline)	SV TRAIN	EPOCH-MIN-TRAIN	0.75	0.56
SV CNN + DATA-AUG + VOTE	SV TRAIN	EPOCH-MIN-TRAIN	0.75	0.56
SV CNN + MV-B	MV TRAIN-I	EPOCH-MIN-TRAIN	0.75	0.57
SV CNN + MV-A	MV TRAIN-II	EPOCH-MIN-TRAIN	0.73	**0.60**
SV CNN + MV-B	MV TRAIN-II	EPOCH-MIN-TRAIN	0.73	**0.64**
SV CNN + MV-A	MV TRAIN-III	EPOCH-MIN-TRAIN	0.55	0.60
SV CNN + MV-B	MV TRAIN-III	EPOCH-MIN-TRAIN	0.62	0.56

Table 9.9.: Comparison of SV TRAIN vs MV TRAIN-II

CNN	Training Strategy	Inference Model	IoU	
			Buildings	Roads
SV CNN + VOTE (Baseline)	SV TRAIN	EPOCH-MIN-VAL	0.75	0.57
SV CNN + MV-A	MV TRAIN-II	EPOCH-MIN-VAL	**0.79**	0.55
SV CNN + MV-B	MV TRAIN-II	EPOCH-MIN-VAL	**0.80**	0.57
SV CNN + VOTE (Baseline)	SV TRAIN	EPOCH-MIN-TRAIN	0.75	0.56
SV CNN + MV-A	MV TRAIN-II	EPOCH-MIN-TRAIN	0.73	**0.60**
SV CNN + MV-B	MV TRAIN-II	EPOCH-MIN-TRAIN	0.73	**0.64**

Table 9.9 compares the IoUs of SV TRAIN and MV TRAIN-II. Recall that MV TRAIN-II allows both the L_{SV} and L_{MV} losses (Eq. 6.1) to influence the weights of the SV CNN. Our hypothesis is that even the earlier layers of the CNN can benefit from the multi-view information. Let us focus on the first three rows of Table 9.9 which correspond to running inference using the EPOCH-MIN-VAL weights. Using MV TRAIN-II to train the SV CNN + MV-B network, we outperform the baseline with a 5% increase in the IoU for the building class, while performing comparably with the baseline for the road class. Although the MV-A module also improves the

IoU for the building class by 4%, it does come at a cost as the corresponding IoU for the road class decreases by 2% when compared to the baseline.

Let us now focus on the last three rows of Table 9.9. The noise in the training and validation labels for the roads is much more than that for the buildings because we assume a constant width of 8 m for all the roads, and because the OSM centerlines of the roads are often not along their true centers. Since the validation labels are also noisy, in Section 6.6 we proposed that we would also save the network weights for the epoch with a very small training loss and a validation IoU that is close to the maximum validation IoU. By looking at the validation IoU as well, we reduce the chances of these network weights being overfitted to the data. Our intuition is borne out by the last three rows of Table 9.9. When compared to the baseline, using MV TRAIN-II with the SV CNN + MV-A network increases the IoU for the road class by 4% while slightly lowering the building IoU by 2%. When compared to the baseline, using MV TRAIN-II with the SV CNN + MV-B network increases the IoU for the road class by 8% while slightly lowering the building IoU by 2%. It is interesting to note that in contrast, EPOCH-MIN-VAL and EPOCH-MIN-TRAIN perform comparably for the SV TRAIN strategy

Table 9.10.: Comparison of SV TRAIN vs MV TRAIN-III

CNN	Training Strategy	Inference Model	IoU	
			Buildings	Roads
SV CNN + VOTE (Baseline)	SV TRAIN	EPOCH-MIN-VAL	0.75	0.57
SV CNN + MV-A	MV TRAIN-III	EPOCH-MIN-VAL	0.63	0.59
SV CNN + MV-B	MV TRAIN-III	EPOCH-MIN-VAL	0.62	0.56
SV CNN + VOTE (Baseline)	SV TRAIN	EPOCH-MIN-TRAIN	0.75	0.56
SV CNN + MV-A	MV TRAIN-III	EPOCH-MIN-TRAIN	0.55	0.60
SV CNN + MV-B	MV TRAIN-III	EPOCH-MIN-TRAIN	0.62	0.56

Next, we conduct another study to establish the importance of using a combination of SV DATALOAD and MV DATALOAD for data-loading. In Table 9.10 we compare SV TRAIN with MV TRAIN-III. Recall that in MV TRAIN-III, the SV CNN + MV-A (or MV-B) network is trained from scratch on just the MV WINDOW samples.

From Table 9.10 we see that the IoU for the building class drops down significantly by $> 11\%$ when compared to the SV CNN baseline. This is as expected because in this case, the network is trained with fewer training samples. It never sees ground-windows that are covered by less than $|Q|$ views. Therefore, it is important that the network be trained with as much non-redundant data as possible, and with multi-view constraints as is done by MV TRAIN-II.

Based on the above results, *the MV TRAIN-II strategy is the best approach for multi-view training and the MV-B Fusion module yields the maximum gains.* We recommend using EPOCH-MIN-VAL for segmenting buildings, and EPOCH-MIN-TRAIN for segmenting roads.

9.5.2 Data Augmentation vs MV Training

Table 9.11.: Impact of simple data augmentation on the SV CNN

CNN	Training Strategy	Inference Model	IoU	
			Buildings	Roads
SV CNN + VOTE (Baseline)	SV TRAIN	EPOCH-MIN-VAL	0.75	0.57
SV CNN + DATA-AUG + VOTE	SV TRAIN	EPOCH-MIN-VAL	0.76	0.55
SV CNN + VOTE (Baseline)	SV TRAIN	EPOCH-MIN-TRAIN	0.75	0.56
SV CNN + DATA-AUG + VOTE	SV TRAIN	EPOCH-MIN-TRAIN	0.75	0.56

In Section 6.8, we had stressed on the difference between data augmentation and multi-view training. In this section, we compare SV CNN with SV CNN + DATA-AUG. SV CNN is trained using SV TRAIN with SV WINDOW, i.e., the training windows used by the SV DATALOAD strategy. SV CNN + DATA-AUG is trained using SV TRAIN with **all** of the available training windows, i.e., SV WINDOW and MV WINDOW where MV WINDOW are the windows used by the MV DATALOAD strategy. As a reminder, the reason for the existence of both SV WINDOW and MV WINDOW is merely due to a choice of implementation. For SV DATALOAD, it is faster to sample image-windows from the individual images separately and in

parallel, to create SV WINDOW. In MV DATALOAD, for each ground-window, multiple image-windows that cover it are extracted from the overlapping images to create MV WINDOW.

Table 9.11 compares the IoUs of SV CNN + VOTE with SV CNN + DATA-AUG + VOTE. We can see that there is no noticeable change in the performance. This is indeed what we expected because of the high correlation and overlap between MV WINDOW and SV WINDOW. Nevertheless, it is crucial to carry out this comparison to prove that the extra samples in MV WINDOW *by themselves do not contain any new information for the CNN to learn from.* The multi-view training strategy, the MV Fusion modules and L_{MV} are what are responsible for the improvements.

9.5.3 Ablation Studies: Does Multi-View Training Improve the Single-View CNN?

To obtain additional insights into how multi-view training improves accuracy, we carry out two ablation studies using the SV CNN + MV-B network which yielded the maximum gains with the MV TRAIN-II strategy as shown in the preceding discussion.

Table 9.12.: Ablation Study I – Comparison of SV TRAIN vs MV TRAIN-I

CNN	Training Strategy	Inference Model	IoU	
			Buildings	Roads
SV CNN + VOTE (Baseline)	SV TRAIN	EPOCH-MIN-VAL	0.75	0.57
SV CNN + MV-B	MV TRAIN-I	EPOCH-MIN-VAL	0.75	0.57
SV CNN + VOTE (Baseline)	SV TRAIN	EPOCH-MIN-TRAIN	0.75	0.56
SV CNN + MV-B	MV TRAIN-I	EPOCH-MIN-TRAIN	0.75	0.57

For the first ablation study, we first train the SV CNN using SV TRAIN and freeze its weights. The MV-B module is then appended to it and only the weights of the MV-B module are trained using the MV TRAIN-I strategy. The IoUs are reported in Table 9.12 and we see that there are no noticeable differences between using the MV TRAIN-I and SV TRAIN strategies. Remember that in MV TRAIN-I, the multi-view

loss L_{MV} (Eq. 6.4) only modifies the weights of the MV Fusion module. This points to the need for allowing L_{MV} to influence the weights of the SV CNN as well, as is done by MV TRAIN-II.

Table 9.13.: Ablation Study II - Impact of multi-view training on the SV CNN

CNN	Training Strategy	Inference Model	IoU	
			Buildings	Roads
SV CNN + VOTE (Baseline)	SV TRAIN	EPOCH-MIN-VAL	0.75	0.57
$SV_{(MV)}$ + VOTE	MV TRAIN-II	EPOCH-MIN-VAL	**0.80**	0.55
SV CNN + MV-B	MV TRAIN-II	EPOCH-MIN-VAL	**0.80**	0.57
SV CNN + VOTE (Baseline)	SV TRAIN	EPOCH-MIN-TRAIN	0.75	0.56
$SV_{(MV)}$ + VOTE	MV TRAIN-II	EPOCH-MIN-TRAIN	0.74	**0.62**
SV CNN + MV-B	MV TRAIN-II	EPOCH-MIN-TRAIN	0.73	**0.64**

For the second study, we take the SV CNN + MV-B network that was trained using the MV TRAIN-II strategy and remove the MV-B module from it. We denote this SV CNN as $SV_{(MV)}$. We run inference using this $SV_{(MV)}$ network and merge the predictions from the overlapping views using majority voting. We do this for both the EPOCH-MIN-VAL and EPOCH-MIN-TRAIN network weights and the corresponding IoUs are shown in the second and fifth rows of Table 9.13.

Looking at the first and second rows of Table 9.13, we see that the building IoU has gone up by 5% while the road IoU has gone down by a relatively smaller 2% for $SV_{(MV)}$, when compared to the SV CNN + VOTE baseline using the EPOCH-MIN-VAL weights. Looking at the fourth and fifth rows of the same table, we see that the road IoU has gone up by 6% while the building IoU has gone down by a relatively smaller 1% for $SV_{(MV)}$, when compared to the SV CNN + VOTE baseline using the EPOCH-MIN-TRAIN weights. **Thus, multi-view training has improved the performance of the $SV_{(MV)}$ network itself, without any increase in the number of parameters.** Therefore, intelligently training a SV CNN using all the available views for a scene can alleviate the effect of noise in the training labels, without changing the original architecture of the SV CNN.

Finally, for easy comparison, we reproduce the IoUs of the complete SV CNN + MV-B network trained with MV TRAIN-II, in the third and sixth rows of Table 9.13. Comparing the 2nd and 3rd rows with each other, and the 5th and 6th rows with each other, we see that the MV Fusion module does provide an additional 2% improvement in the IoU for the road class, over the $SV_{(MV)}$ network.

9.6 Comparison to Prior State-of-the-Art

For a fair comparison, we consider the most relevant prior state-of-the-art studies that use multi-view off-nadir images for semantic segmentation. The work presented in [47] discusses the entry that won the 2019 IEEE GRSS Data Fusion Contest for Multi-view Semantic Stereo. That approach trains single-view networks using both WV3 images and DSMs over a small 10-20 km^2 region with *precisely-annotated labels* and reports an IoU of about 0.8-0.81 for the building class on that dataset. The performance gains for that approach come from training the network on DSMs which help to segment buildings more accurately. Our best IoU for the building class on our Ohio inference dataset is also 0.8, but we use only noisy training labels that are automatically derived from a much larger 100 km^2 region. It is possible that by adding the DSMs as inputs to our network, we could further improve the IoU.

Our IoU for the building class is noticeably better than that reported by the work in [46], which trains single-view CNNs on WV3 images and OSM labels covering 1-2 km^2 to segment buildings. Additionally, that work uses OSM to eliminate the confounding effects of bridges and elevated roads from the final predictions. Most of the other studies in the literature use single-view pre-orthorectified images. It should be pointed out that our multi-view training strategy could be applied to any of those network architectures.

DeepLabv3+: For another comparison, we take a pretrained DeepLabv3+ (DLabv3) CNN with a WideResNet38 trunk [162] that is one of the top performers on the CityScapes benchmark dataset [163]. We modify the first layer to accept 8 bands.

With this modification the network has ∼137 million trainable parameters whereas in comparison the U-Net only has ∼31 million parameters.

We first train the network using SV TRAIN. Since this network is much bigger than the U-Net, each epoch for this network takes significantly longer than for the U-Net. Due to the size of the network, we set the batch size to 12 for SV TRAIN. At inference time we use majority voting. This strategy is denoted as $\text{SV}_{(\text{DLabv3})}$ CNN + VOTE. For the multi-view training, we append the MV-B Fusion module to the network and then employ the MV TRAIN-II strategy. Recall that this is the best performing strategy for the U-Net. $|Q|$ is set to 12 for the MV TRAIN-II strategy.

In Table 9.14 we show the IoUs for the DeepLabv3+ experiments. Firstly, we note that $\text{SV}_{(\text{DLabv3})}$ CNN + VOTE achieves an IoU of 0.82 and 0.55 for the building and road classes respectively. This network has already been trained on a large amount of precise labels from the CityScapes dataset [163] and is therefore more robust to noisy data. What is interesting is that these numbers are comparable to the corresponding IoUs of 0.80 and 0.57 for the U-Net + MV-B network trained with MV TRAIN-II, despite the fact that *the U-Net has significantly fewer trainable parameters than the DeepLabv3+ network and that it has been trained only on noisy labels.*

The second row of Table 9.14 has the entry for running inference using the EPOCH-MIN-VAL weights of the $\text{SV}_{(\text{DLabv3})}$ CNN + MV-B network after being trained with MV TRAIN-II. Compared to the $\text{SV}_{(\text{DLabv3})}$ CNN + VOTE, the building IoU goes down by 3% whereas the road IoU goes up by 5%. The mean IoU goes up by 1% when we use multi-view training. It is possible that we are already at the limits of how much a CNN can learn, given the extent of noise in the system. Another possibility is that since the DeepLabv3+ network is much bigger than the U-Net, and since the multi-view features are fused at the end, the influence of L_{MV} on the earlier layers of the DeepLabv3+ network might be reduced. We might get better results by fusing the multi-view data at an earlier stage in the network. This needs further investigation. It should be pointed out that in the work described in [164], the authors claim that the DeepLabv3+ net achieves much lower accuracies than a U-Net

for the task of segmenting buildings in WV3 images belonging to the DeepGlobe 2018 Building Extraction Challenge [77].

Table 9.14.: Comparison of SV TRAIN with MV TRAIN-II for DeepLabv3+

CNN	Training Strategy	Inference Model	IoU	
			Buildings	Roads
$SV_{(DLabv3)}$ CNN + VOTE	SV TRAIN	EPOCH-MIN-VAL	0.82	0.55
$SV_{(DLabv3)}$ CNN + MV-B	MV TRAIN-II	EPOCH-MIN-VAL	0.79	0.60

9.7 Training on True Ortho Images vs on Off-Nadir Images

We have described our framework for training a SV CNN directly on the off-nadir images in Sections 7.2 and 5.2. For a quantitative evaluation, we run inference using this trained CNN on the portions of the off-nadir images that correspond to the "unseen" inference region. For a fair comparison, the predicted labels are then orthorectified so that the evaluation can be done in the same orthorectified space. Predictions from overlapping images are merged via majority voting.

When the off-nadir images and the projected OSM labels are used for training, the EPOCH-MIN-VAL weights yield IoU scores of 0.73 and 0.55 for the building and road classes respectively. These scores are $\sim 2\%$ lower than the corresponding numbers for the SV CNN + VOTE that is trained directly on the true ortho images (Table 9.8). As mentioned in Section 5.2, one possible reason for these reduced IoU scores might be the increased error in the OSM labels after they have been projected into the off-nadir images. Another reason could be that the CNN finds it difficult to separate the building walls from the roofs in the off-nadir images. In contrast, perpendicular building walls are not present in the true ortho images. Nevertheless, we have demonstrated that it is possible to train a CNN on off-nadir images using noisy OSM labels and obtain decent IoU scores. Such a CNN can be directly used with new off-nadir images without having to align them or orthorectify them. It should be

pointed out that multi-view training using the off-nadir images is also possible, albeit more challenging, which we leave for future work.

9.8 Qualitative Evaluation of Semantic Segmentation

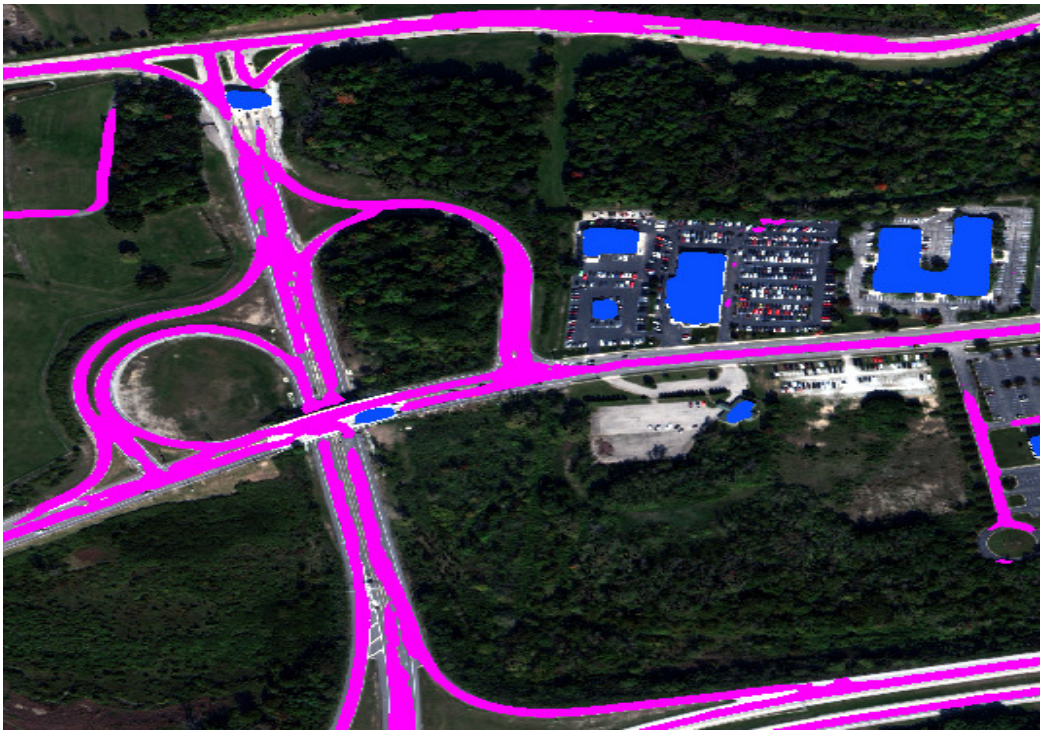
In this section we present some qualitative examples of the semantic labels output by our framework. In Figs. 9.9, 9.10, 9.11, 9.12, 9.13 and 9.14, we show some typical examples of semantic labels output by our CNN.

(a) Ortho View

(b) Predicted Labels

Fig. 9.9.: Examples of orthorectified images and semantic labels output by our framework. Buildings are marked in blue and roads are marked in magenta.

(a) Ortho View



(b) Predicted Labels

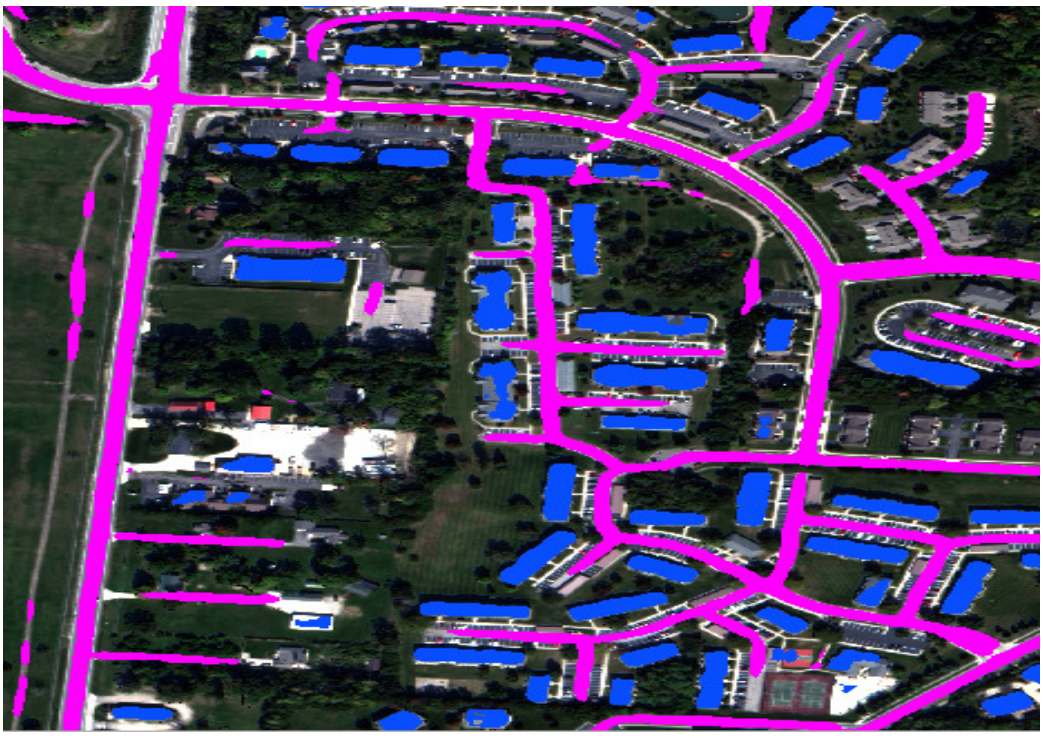
Fig. 9.10.: Examples of orthorectified images and semantic labels output by our framework. Buildings are marked in blue and roads are marked in magenta.

(a) Ortho View

(b) Predicted Labels

Fig. 9.11.: Examples of orthorectified images and semantic labels output by our framework. Buildings are marked in blue and roads are marked in magenta.

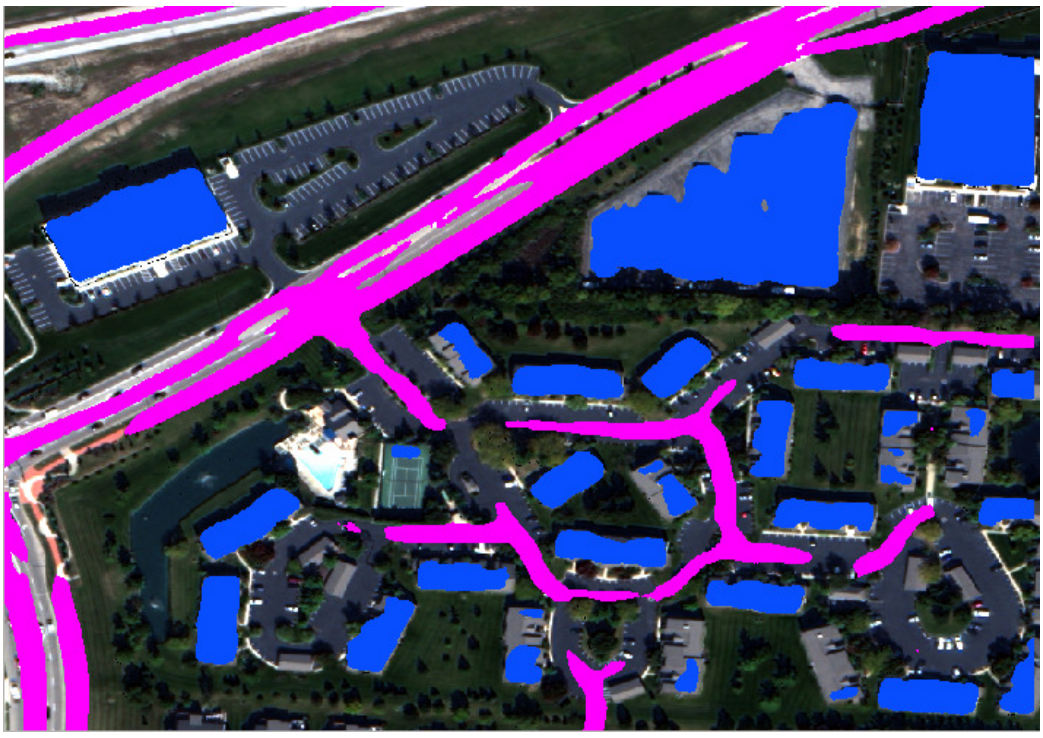
(a) Ortho View



(b) Predicted Labels

Fig. 9.12.: Examples of orthorectified images and semantic labels output by our framework. Buildings are marked in blue and roads are marked in magenta.

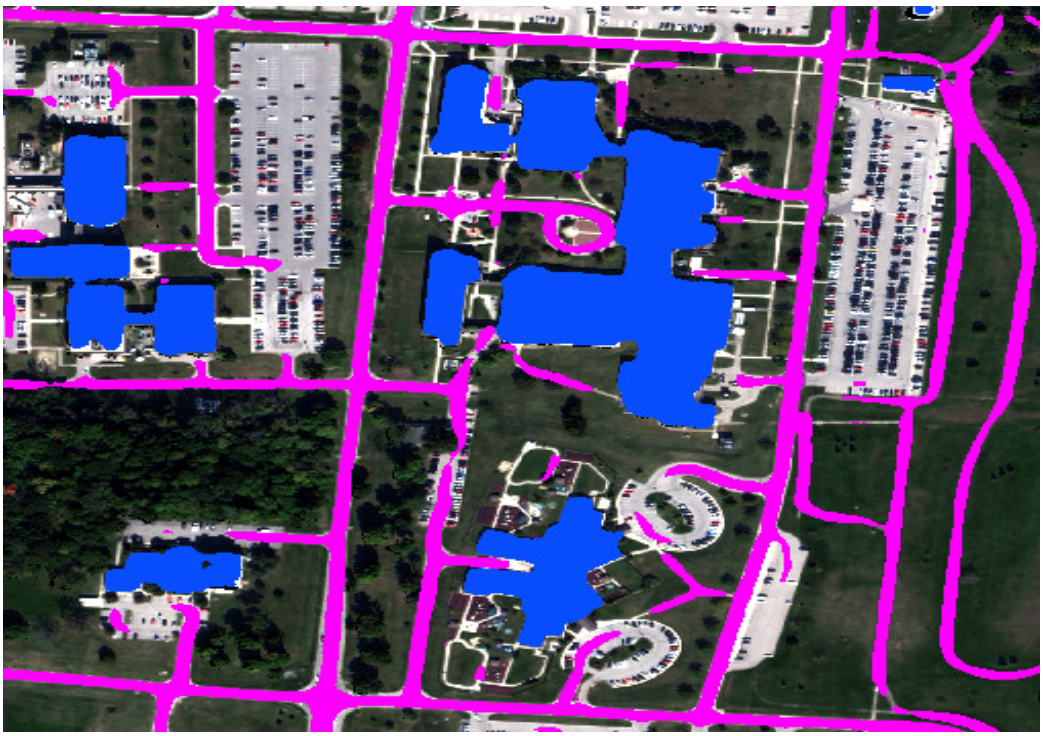
(a) Ortho View



(b) Predicted Labels

Fig. 9.13.: Examples of orthorectified images and semantic labels output by our framework. Buildings are marked in blue and roads are marked in magenta.

(a) Ortho View



(b) Predicted Labels

Fig. 9.14.: Examples of orthorectified images and semantic labels output by our framework. Buildings are marked in blue and roads are marked in magenta.

Figs. 9.15 and 9.16 highlight how multi-view training can help the CNN to label challenging examples of buildings. Fig. 9.15 shows examples of residential buildings. While we have displayed an orthorectified view that was constructed using an image captured at a near-nadir view-angle, these roofs can be significantly occluded by the trees in the other off-nadir views. Multi-view training is able to segment out such roofs well. In Fig. 9.16 we show other challenging examples such as roofs made of highly reflective surfaces, and small buildings. Multi-view training helps segment out such buildings as well.

With respect to segmentation of roads, parking lots pose a particularly difficult challenge because their shape and spectral signatures are very similar to true roads. However, multi-view training is able to learn from the differences caused by the absence and presence of vehicles in images taken on different dates, and this is illustrated in Figs. 9.17, Figs. 9.18 and Figs. 9.19.

Fig. 9.15.: To illustrate the power of our approach, the challenging buildings in the right column were extracted by our approach based on multi-view training for semantic labeling. Compare with the left column where the training is based on single-views. Buildings are marked in blue in a semi-transparent manner.

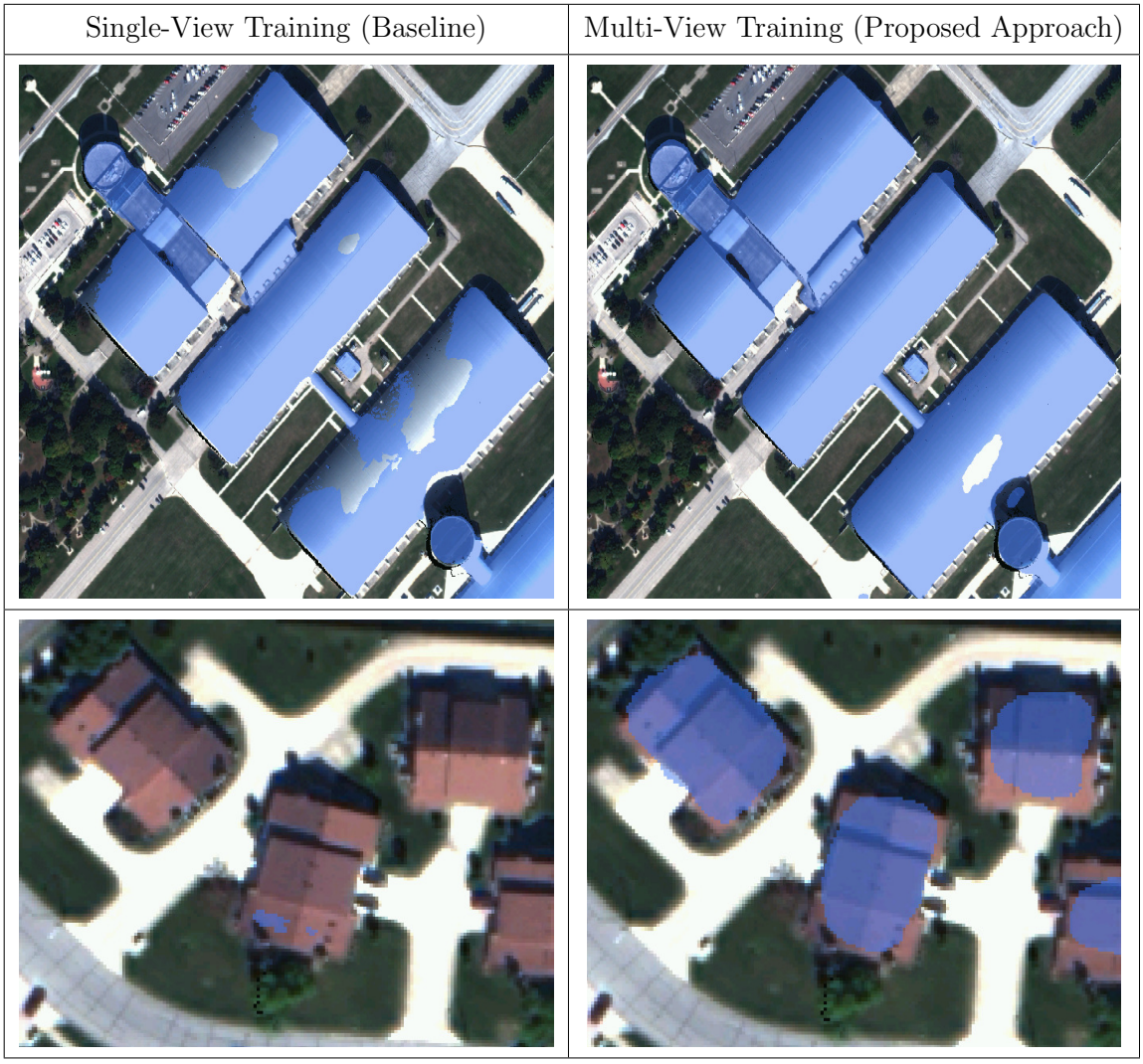


Fig. 9.16.: To illustrate the power of our approach, the challenging buildings in the right column were extracted by our approach based on multi-view training for semantic labeling. Compare with the left column where the training is based on single-views. Buildings are marked in blue in a semi-transparent manner.

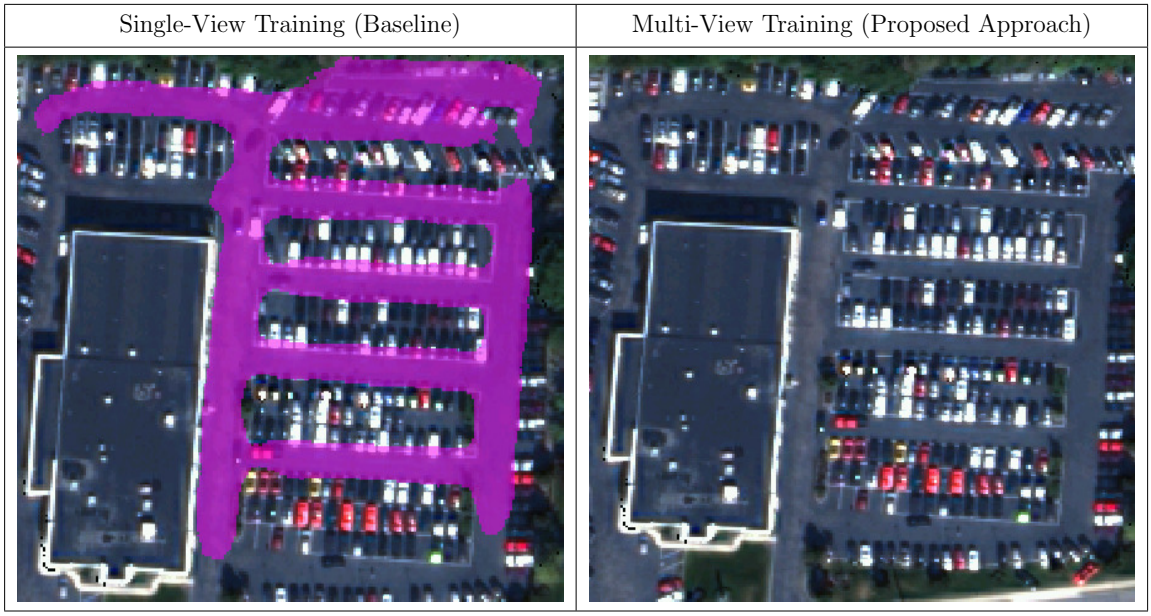


Fig. 9.17.: An example illustrating how multi-view training helps to distinguish parking lots from true roads. Predicted road labels are marked in magenta.

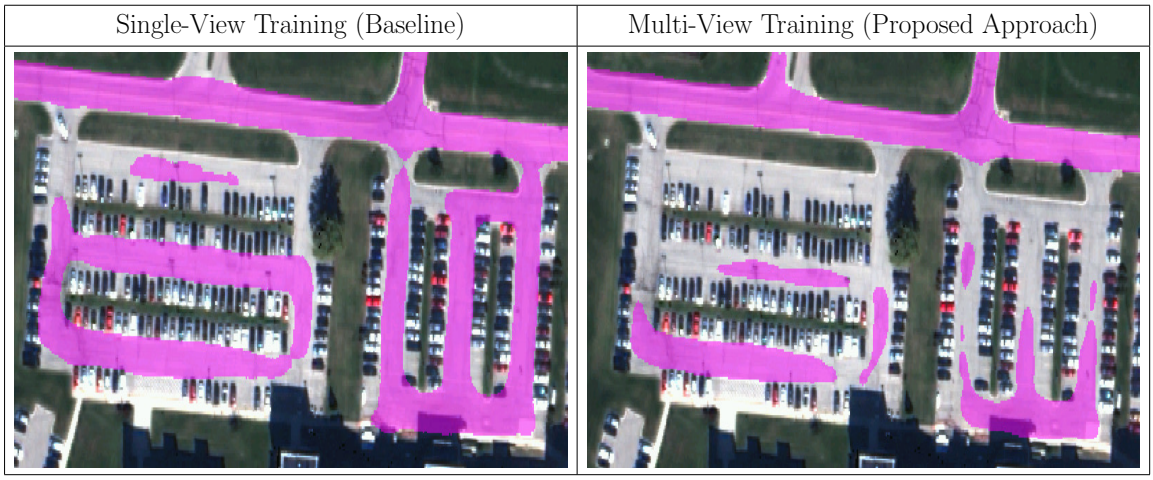


Fig. 9.18.: An example illustrating how multi-view training helps to distinguish parking lots from true roads. Predicted road labels are marked in magenta.

Fig. 9.19.: An example illustrating how multi-view training helps to distinguish parking lots from true roads. Predicted road labels are marked in magenta.

10. CONCLUSIONS AND FUTURE WORK

This chapter summarizes the conclusions of our work and presents possible directions for new research.

10.1 Conclusions

By trying to answer the questions that we laid out in Chapter 1, this work has resulted in the following important conclusions and contributions.

10.1.1 Semantic Segmentation

1. We have demonstrated that a CNN can be trained using only noisy training labels and minimal human supervision to achieve state-of-the-art semantic segmentation performance. Our proposed system can not only handle the noise in the groundtruth labels, but also the noise coming from the different components of an end-to-end automated system.

2. One of the key contributions of this work is to show that there is good value in fusing information from multiple views of the same scene at the time of training and not just at the time of inference. Via multi-view training, the network learns not just from the image content but also from the variation of this content across time and view angles.

3. We have proposed and experimented with novel CNN architectures, loss functions, data loading and training paradigms for multi-view and multi-date training. Via quantitative evaluation we have validated the benefits of the proposed design and shown an increase in the segmentation IoUs in the range of 4-7%.

4. Ablation studies reveal that multi-view training has a positive influence on the weights in the earlier layers of a network as well. It can improve the performance of the Single-View CNN itself without any increase in the number of parameters of the trained model.

5. It is customary to use the validation loss as a metric for model selection in machine learning applications. The training loss is often ignored due to fear of overfitting. However, when using noisy validation data for hyper-parameter tuning and model selection, there is good value in considering models with a low training loss and sufficiently high validation IoU.

6. It is indeed possible to use noisy OSM data to directly train CNNs for semantic labeling of off-nadir images albeit with a small decrease in labeling accuracy, due to propagation of errors from different sources.

7. We have evaluated the direct impact of ToA correction on the semantic labeling accuracies. A surprising conclusion is that CNNs might not be able to learn to apply the same radiometric correction as ToA correction achieves to account for different Earth-Sun distances and solar elevation angles at the time of image capture. ToA correction by itself provides a 3% improvement to the segmentation IoUs.

8. Sub-pixel accurate image-to-image alignment, and accurate DSM construction can noticeably improve the segmentation IoUs as well (by up to 4% for our dataset).

10.1.2 Large-Area Image-to-Image Alignment and DSM Construction

1. Contrary to popular literature, we have found credible evidence to show that the RPC bias corrections can vary nonlinearly for full-sized WV3 images. This has a direct impact on the quality of the 3D point clouds constructed via stereo matching of such images.

2. We have proposed an algorithm that analyzes the graph-connectivity properties to automatically detect poor image alignment. This is especially important for images taken by satellite sensors owing to the ambiguity arising from the near-parallel nature of the camera rays. This algorithm is very useful for fast alignment of full-sized images as it selectively runs slower and more robust alignment algorithms only when required.

3. A surprising observation is that a tile-based image-to-image alignment scheme is sufficient to create high quality large-area DSMs. It is remarkable that a simple L2 regularization of the estimated bias corrections can lower the median Z difference between the overlapping portions of adjacent tile-level DSMs to within 0.5 m. While this difference is small enough to be ignored for our application, we also propose two hierarchical algorithms to possibly resolve the residual between-tile alignment errors.

4. We have designed a fast algorithm called "gwarp++" for creating true ortho views of full-sized satellite images using the RPCs and a 2.5D DSM.

10.1.3 Distributing Remote-Sensing Applications Across an OpenStack Cloud

1. Repeated cache-coherency checks due to floating VCPUs is an important bottleneck in designing massively-parallel algorithms for a cloud computing framework. The direct impact of CPU pinning and NUMA nodes on the execution time of an algorithm such as tSGM is a revelation.

2. Simple rules can yield significant benefits in processing time. For instance, copying data locally and reading/writing data in big chunks instead of directly carrying out per-pixel operations across the network can noticeably reduce network congestion.

3. It is important to choose the right level at which to parallelize each component of our framework. To this end we have proposed the use of different task-distribution schemes for different components of the framework. For instance, one can save days of processing for large-area stereo-matching by using an Asymmetric Captain Worker Pull-Based (APLB) distributed architecture.

10.2 Future Work and Possible Extensions

10.2.1 Large-Area Image-to-Image Alignment and DSM Construction

- We have proposed two hierarchical extensions to the image-alignment algorithm in order to resolve residual alignment errors between neighboring tiles. These can be evaluated against the approach outlined by the study in [64] which proposes to first apply plane rectification to the full-sized images before carrying out alignment.

- It is possible that the absolute error of alignment can be further reduced using different regularizers on the bias corrections. This needs further investigation.

- The tSGM algorithm could be replaced by state-of-the-art deep-learning approaches for stereo matching. This of course requires groundtruth disparity maps which could be created using LiDAR [165], [2].

- It is possible to model the noise in the image-alignment and the stereo-matching processes and subsequently propagate these models into a final noise model for the DSM-creation process. This could potentially improve the accuracy of the DSMs.

- One could estimate the terrain elevation from the DSM using an approach as proposed by the work described in [66]. This can then be used to improve the accuracy of "gwarp++" to create more precise true ortho images.

10.2.2 Semantic Segmentation

- The proposed multi-view fusion module can be interpreted as a late-stage data fusion. It might be likely that a multi-stage fusion of information might yield more improvements. More precisely, one could pool together the features from the different views of the same scene at different depths in the CNN. This can open up a whole new category of multi-view and multi-date CNN architectures for satellite-image labeling.

- One could estimate the digital terrain model (DTM) from the DSM using an approach as proposed by the work described in [66]. Subtracting the DTM from the DSM will give us a normalized DSM (nDSM). Subsequently it is possible that by adding the nDSM as an input to our network, we could further improve the IoU as demonstrated by the work described in [47].

- One could try to improve the semantic labels by using information about the boundaries as proposed by the authors of the studies described in [94], [166], etc.

- Another possible area of research is to model the noise in the OSM. This might require some human effort which itself could be guided by an initial classifier. One can identify regions of maximum discrepancy between the predictions of a classifier and the OSM labels and subsequently estimate a noise model for the OSM. This could serve as an additional input to the CNN.

- Our proposed framework for labeling the off-nadir images is based on using a Single-View CNN. It is possible to extend the multi-view training strategy to the off-nadir images as well. This would need the creation of additional input arrays that store the correspondences between the pixels of the different off-nadir image-windows. This also raises interesting questions on how to concatenate features from the different off-nadir views.

- Evaluating the proposed multi-view training framework using additional CNN architectures might provide additional insights into how best to combine the multi-view and multi-date information.

By removing many of the constraints that beset the popular datasets for semantic-segmentation of remotely-sensed data, our proposed framework can help researchers develop and evaluate their labeling algorithms in new regions of the world, even in the presence of noisy training labels. Labeling the world is an extremely challenging topic and in no way is this work the last word on the subject. That said, we believe that the ideas put forth in this work will serve as a good stepping stone for new avenues of research.

REFERENCES

REFERENCES

[1] "SpaceNet on Amazon Web Services (AWS). Datasets. The SpaceNet Catalog. Last modified April 30, 2018." https://spacenetchallenge.github.io/datasets/datasetHomePage.html, accessed: 2018-08-06.

[2] M. Bosch, K. Foster, G. Christie, S. Wang, G. D. Hager, and M. Brown, "Semantic stereo for incidental satellite images," in *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 2019, pp. 1524–1532.

[3] H. R. Goldberg, S. Wang, G. A. Christie, and M. Z. Brown, "Urban 3D challenge: building footprint detection using orthorectified imagery and digital surface models from commercial satellites," in *Geospatial Informatics, Motion Imagery, and Network Analytics VIII*, K. Palaniappan, P. J. Doucette, and G. Seetharaman, Eds., vol. 10645, International Society for Optics and Photonics. SPIE, 2018, pp. 12 – 31. [Online]. Available: https://doi.org/10.1117/12.2304682

[4] M. Brown, H. Goldberg, K. Foster, A. Leichtman, S. Wang, S. Hagstrom, M. Bosch, and S. Almes, "Large-scale public lidar and satellite image data set for urban semantic labeling," in *Laser Radar Technology and Applications XXIII*, M. D. Turner and G. W. Kamerman, Eds., vol. 10636, International Society for Optics and Photonics. SPIE, 2018, pp. 154 – 167. [Online]. Available: https://doi.org/10.1117/12.2304403

[5] T. Prakash and A. C. Kak, "Active learning for designing detectors for infrequently occurring objects in wide-area satellite imagery," *Computer Vision and Image Understanding*, vol. 170, pp. 92 – 108, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1077314218300390

[6] OpenStreetMap contributors, "Planet dump retrieved from https://planet.osm.org ," https://www.openstreetmap.org, 2017.

[7] "Orthophoto," https://en.wikipedia.org/wiki/Orthophoto, accessed: 2020-06-15.

[8] "DSM," https://gisgeography.com/dem-dsm-dtm-differences/, accessed: 2019-12-15.

[9] "LiDAR," https://en.wikipedia.org/wiki/Lidar, accessed: 2019-12-15.

[10] "OpenStack," https://www.openstack.org/, accessed: 2019-12-15.

[11] "SRTM DEM," https://www2.jpl.nasa.gov/srtm/, accessed: 2019-12-15.

[12] "WorldView-3 Satellite Sensor," http://www.satimagingcorp.com/satellite-sensors/worldview-3/, accessed: 2016-08-05.

[13] "DigitalGlobe," https://www.digitalglobe.com/products/satellite-imagery, accessed: 2019-12-11.

[14] G. Dial and J. Grodecki, "Rpc replacement camera models," *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 34, 2005.

[15] "Radiometric Use of WorldView-3," https://dg-cms-uploads-production.s3.amazonaws.com/uploads/document/file/207/Radiometric_Use_of_WorldView-3_v2.pdf, accessed: 2018-08-06.

[16] "Weighted Brovey for Pansharpening," https://desktop.arcgis.com/en/arcmap/10.3/manage-data/raster-and-images/fundamentals-of-panchromatic-sharpening.htm, accessed: 2019-12-15.

[17] "IARPA CORE3D," https://www.iarpa.gov/index.php/research-programs/core3d, accessed: 2019-12-15.

[18] "GDAL," https://gdal.org/, accessed: 2019-12-15.

[19] "gdalwarp," https://gdal.org/programs/gdalwarp.html, accessed: 2019-12-15.

[20] "Google Earth," https://www.google.com/earth/, accessed: 2020-06-15.

[21] "Open Database License," https://opendatacommons.org/licenses/odbl/index.html, accessed: 2019-04-01.

[22] "ENVI," https://www.harrisgeospatial.com/Software-Technology/ENVI, accessed: 2019-12-15.

[23] "QGIS," https://qgis.org/en/site/, accessed: 2019-12-15.

[24] "Monteverdi," https://www.orfeo-toolbox.org/CookBook/Monteverdi.html, accessed: 2019-12-15.

[25] "Orfeo Toolbox," https://www.orfeo-toolbox.org, accessed: 2019-12-15.

[26] "gdal_pansharpen.py," https://gdal.org/programs/gdal_pansharpen.html, accessed: 2019-12-15.

[27] "Anaconda," https://www.anaconda.com/, accessed: 2019-12-15.

[28] "Applied Imagery," http://appliedimagery.com/download/, accessed: 2019-12-15.

[29] "CloudCompare," https://www.danielgm.net/cc/, accessed: 2019-12-15.

[30] "RVL Cloud," https://engineering.purdue.edu/RVL/rvlCloud, accessed: 2019-12-15.

[31] "AWS," https://aws.amazon.com/, accessed: 2019-12-15.

[32] "Microsoft Azure," https://azure.microsoft.com/en-us/, accessed: 2019-12-15.

[33] R. Perko, H. Raggam, and P. M. Roth, "Mapping with Pléiades–End-to-End Workflow," *Remote Sensing*, vol. 11, no. 17, 2019. [Online]. Available: https://www.mdpi.com/2072-4292/11/17/2052

[34] "Vricon," https://www.vricon.com/, accessed: 2018-08-06.

[35] M. Bosch, Z. Kurtz, S. Hagstrom, and M. Brown, "A multiple view stereo benchmark for satellite imagery," in *2016 IEEE Applied Imagery Pattern Recognition Workshop (AIPR)*, Oct 2016, pp. 1–9.

[36] T. B. Pollard, I. Eden, J. L. Mundy, and D. B. Cooper, "A volumetric approach to change detection in satellite images," *Photogrammetric Engineering & Remote Sensing*, vol. 76, no. 7, pp. 817–831, 2010. [Online]. Available: https://www.ingentaconnect.com/content/asprs/pers/2010/00000076/00000007/art00003

[37] A. O. Ulusoy, M. J. Black, and A. Geiger, "Patches, planes and probabilities: A non-local prior for volumetric 3d reconstruction," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3280–3289, 2016.

[38] D. Crispell, J. Mundy, and G. Taubin, "A variable-resolution probabilistic three-dimensional model for change detection," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 50, no. 2, pp. 489–500, Feb 2012.

[39] G. Kuschk, "Large scale urban reconstruction from remote sensing imagery," *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XL-5/W1, pp. 139–146, 2013. [Online]. Available: https://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XL-5-W1/139/2013/

[40] D. Shean, O. Alexandrov, Z. Moratto, B. Smith, I. Joughin, C. Porter, and P. Morin, "An automated, open-source pipeline for mass production of digital elevation models (dems) from very-high-resolution commercial stereo satellite imagery," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 116, pp. 101–117, 6 2016.

[41] G. Facciolo, C. D. Franchis, and E. Meinhardt-Llopis, "Automatic 3d reconstruction from multi-date satellite images," in *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, July 2017, pp. 1542–1551.

[42] O. C. Ozcanli, Y. Dong, J. L. Mundy, H. Webb, R. Hammoud, and V. Tom, "A comparison of stereo and multiview 3-d reconstruction using cross-sensor satellite imagery," in *2015 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, June 2015, pp. 17–25.

[43] "IARPA MVS Challenge 2016," https://www.iarpa.gov/challenges/3dchallenge.html, accessed: 2018-08-06.

[44] R. Qin, "Automated 3d recovery from very high resolution multi-view satellite images," *CoRR*, vol. abs/1905.07475, 2019. [Online]. Available: http://arxiv.org/abs/1905.07475

[45] K. Gong and D. Fritsch, "Point cloud and digital surface model generation from high resolution multiple view stereo satellite imagery," *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XLII-2, pp. 363–370, 2018. [Online]. Available: https://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XLII-2/363/2018/

[46] M. J. Leotta, C. Long, B. Jacquet, M. Zins, D. Lipsa, J. Shan, B. Xu, Z. Li, X. Zhang, S.-F. Chang, M. Purri, J. Xue, and K. Dana, "Urban semantic 3d reconstruction from multiview satellite imagery," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2019.

[47] P. dAngelo, D. Cerra, S. M. Azimi, N. Merkle, J. Tian, S. Auer, M. Pato, R. de los Reyes, X. Zhuo, K. Bittner, T. Krauss, and P. Reinartz, "3d semantic segmentation from multi-view optical satellite images," in *IGARSS 2019 - 2019 IEEE International Geoscience and Remote Sensing Symposium*, July 2019, pp. 5053–5056.

[48] J. Grodecki and G. Dial, "Block adjustment of high-resolution satellite images described by rational polynomials," *Photogrammetric Engineering & Remote Sensing*, vol. 69, no. 1, pp. 59–68, 2003. [Online]. Available: https://www.ingentaconnect.com/content/asprs/pers/2003/00000069/00000001/art00004

[49] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[50] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," in *European conference on computer vision*. Springer, 2006, pp. 404–417.

[51] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[52] "Refining Homographies with Nonlinear Least-Squares Minimization (Gradient Descent, Levenberg-Marquardt, & DogLeg," https://engineering.purdue.edu/kak//computervision/ECE661Folder/Lecture12.pdf, accessed: 2019-12-15.

[53] M. I. Lourakis and A. A. Argyros, "Sba: A software package for generic sparse bundle adjustment," *ACM Transactions on Mathematical Software (TOMS)*, vol. 36, no. 1, pp. 1–30, 2009.

[54] R. Hartley and A. Zisserman, *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[55] R. Qin, "Analysis of critical parameters of satellite stereo image for 3d reconstruction and mapping," *CoRR*, vol. abs/1905.07476, 2019. [Online]. Available: http://arxiv.org/abs/1905.07476

[56] C. de Franchis, E. Meinhardt-Llopis, J. Michel, J. . Morel, and G. Facciolo, "On stereo-rectification of pushbroom images," in *2014 IEEE International Conference on Image Processing (ICIP)*, 2014, pp. 5447–5451.

[57] M. Rothermel, "Development of a SGM-based multi-view reconstruction framework for aerial imagery," Dissertation, University of Stuttgart, 2016.

[58] J. Zbontar and Y. LeCun, "Stereo matching by training a convolutional neural network to compare image patches," *Journal of Machine Learning Research*, vol. 17, pp. 1–32, 2016.

[59] S. Patil, T. Prakash, B. Comandur, and A. Kak, "A Comparative Evaluation of SGM Variants (including a New Variant, tMGM) for Dense Stereo Matching," 2019.

[60] "ETH Zurich," https://ethz.ch/en.html, accessed: 2020-06-15.

[61] "Applied Research Associates," https://www.ara.com/, accessed: 2020-06-15.

[62] H. Hirschmuller, "Stereo processing by semiglobal matching and mutual information," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 30, no. 2, pp. 328–341, 2007.

[63] "IKONOS Satellite Sensor," https://www.satimagingcorp.com/satellite-sensors/ikonos/, accessed: 2016-08-05.

[64] X. Huang and R. Qin, "Multi-view large-scale bundle adjustment method for high-resolution satellite images," *CoRR*, vol. abs/1905.09152, 2019. [Online]. Available: http://arxiv.org/abs/1905.09152

[65] O. Ronneberger, P.Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, ser. LNCS, vol. 9351. Springer, 2015, pp. 234–241, (available on arXiv:1505.04597 [cs.CV]). [Online]. Available: http://lmb.informatik.uni-freiburg.de/Publications/2015/RFB15a

[66] W. Zhang, J. Qi, P. Wan, H. Wang, D. Xie, X. Wang, and G. Yan, "An easy-to-use airborne lidar data filtering method based on cloth simulation," *Remote Sensing*, vol. 8, no. 6, p. 501, Jun 2016. [Online]. Available: http://dx.doi.org/10.3390/rs8060501

[67] "OpenMP," https://www.openmp.org/, accessed: 2020-05-20.

[68] A. F. Wolf, "Using WorldView-2 Vis-NIR multispectral imagery to support land mapping and feature extraction using normalized difference index ratios," in *Algorithms and Technologies for Multispectral, Hyperspectral, and Ultraspectral Imagery XVIII*, S. S. Shen and P. E. Lewis, Eds., vol. 8390, International Society for Optics and Photonics. SPIE, 2012, pp. 188 – 195. [Online]. Available: https://doi.org/10.1117/12.917717

[69] Z. Zhang, Q. Liu, and Y. Wang, "Road extraction by deep residual u-net," *IEEE Geoscience and Remote Sensing Letters*, vol. 15, no. 5, pp. 749–753, May 2018.

[70] N. Atif, M. Bhuyan, and S. Ahamed, "A review on semantic segmentation from a modern perspective," in *2019 International Conference on Electrical, Electronics and Computer Engineering (UPCON)*, 2019, pp. 1–6.

[71] A. Garcia-Garcia, S. Orts-Escolano, S. Oprea, V. Villena-Martinez, and J. Garcia-Rodriguez, "A Review on Deep Learning Techniques Applied to Semantic Segmentation," 2017.

[72] Y. Guo, Y. Liu, T. Georgiou, and M. S. Lew, "A review of semantic segmentation using deep neural networks," *International journal of multimedia information retrieval*, vol. 7, no. 2, pp. 87–93, 2018.

[73] X. Liu, Z. Deng, and Y. Yang, "Recent progress in semantic image segmentation," *Artificial Intelligence Review*, vol. 52, no. 2, pp. 1089–1106, 2019.

[74] F. Lateef and Y. Ruichek, "Survey on semantic segmentation using deep learning techniques," *Neurocomputing*, vol. 338, pp. 321–348, 2019.

[75] S. A. Taghanaki, K. Abhishek, J. P. Cohen, J. Cohen-Adad, and G. Hamarneh, "Deep semantic segmentation of natural and medical images: A review," *arXiv preprint arXiv:1910.07655*, 2019.

[76] R. Singh and R. Rani, "Semantic segmentation using deep convolutional neural network: A review," *Available at SSRN 3565919*, 2020.

[77] I. Demir, K. Koperski, D. Lindenbaum, G. Pang, J. Huang, S. Basu, F. Hughes, D. Tuia, and R. Raskar, "DeepGlobe 2018: A Challenge to Parse the Earth through Satellite Images," *CoRR*, vol. abs/1805.06561, 2018. [Online]. Available: http://arxiv.org/abs/1805.06561

[78] "IEEE Data Fusion Contest," http://www.grss-ieee.org/community/technical-committees/data-fusion/data-fusion-contest/, accessed: 2018-08-06.

[79] A. Lagrange, B. L. Saux, A. Beaupre, A. Boulch, A. Chan-Hon-Tong, S. Herbin, H. Randrianarivo, and M. Ferecatu, "Benchmarking classification of earth-observation data: From learning explicit features to convolutional networks," in *2015 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, July 2015, pp. 4173–4176.

[80] "ISPRS-2D Dataset," http://www2.isprs.org/commissions/comm3/wg4/semantic-labeling.html, accessed: 2018-08-06.

[81] S. Wang, M. Bai, G. Máttyus, H. Chu, W. Luo, B. Yang, J. Liang, J. Cheverie, S. Fidler, and R. Urtasun, "Torontocity: Seeing the world with a million eyes," *CoRR*, vol. abs/1612.00423, 2016. [Online]. Available: http://arxiv.org/abs/1612.00423

[82] G. Xia, X. Bai, J. Ding, Z. Zhu, S. J. Belongie, J. Luo, M. Datcu, M. Pelillo, and L. Zhang, "DOTA: A large-scale dataset for object detection in aerial images," *CoRR*, vol. abs/1711.10398, 2017. [Online]. Available: http://arxiv.org/abs/1711.10398

[83] "Geospatial Repository and Data Management System," https://griduc.rsgis.erdc.dren.mil/griduc, accessed: 2018-08-06.

[84] N. Yokoya, P. Ghamisi, J. Xia, S. Sukhanov, R. Heremans, I. Tankoyeu, B. Bechtel, B. Le Saux, G. Moser, and D. Tuia, "Open data for global multimodal land use classification: Outcome of the 2017 ieee grss data fusion contest," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 11, no. 5, pp. 1363–1377, May 2018.

[85] G. Mttyus, W. Luo, and R. Urtasun, "Deeproadmapper: Extracting road topology from aerial images," in *2017 IEEE International Conference on Computer Vision (ICCV)*, Oct 2017, pp. 3458–3466.

[86] L. Zhou, C. Zhang, and M. Wu, "D-linknet: Linknet with pretrained encoder and dilated convolution for high resolution satellite imagery road extraction." in *CVPR Workshops*, 2018, pp. 182–186.

[87] A. Batra, S. Singh, G. Pang, S. Basu, C. Jawahar, and M. Paluri, "Improved road connectivity by joint learning of orientation and segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 10 385–10 393.

[88] S. Singh, A. Batra, G. Pang, L. Torresani, S. Basu, M. Paluri, and C. Jawahar, "Self-supervised feature learning for semantic segmentation of overhead imagery." in *BMVC*, vol. 1, no. 2, 2018, p. 4.

[89] G. Rotich, S. Aakur, R. Minetto, M. P. Segundo, and S. Sarkar, "Using semantic relationships among objects for geospatial land use classification," in *2018 IEEE Applied Imagery Pattern Recognition Workshop (AIPR)*. IEEE, 2018, pp. 1–7.

[90] G. Rotich, R. Minetto, and S. Sarkar, "Resource-constrained simultaneous detection and labeling of objects in high-resolution satellite images," *arXiv preprint arXiv:1810.10110*, 2018.

[91] D. Costea, A. Marcu, E. Slusanschi, and M. Leordeanu, "Roadmap generation using a multi-stage ensemble of deep neural networks with smoothing-based optimization." in *CVPR Workshops*, 2018, pp. 220–224.

[92] M. Volpi and V. Ferrari, "Structured prediction for urban scene semantic segmentation with geographic context," in *2015 Joint Urban Remote Sensing Event (JURSE)*, 2015, pp. 1–4.

[93] M. Volpi and V. Ferrari, "Semantic segmentation of urban scenes by learning local class interactions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2015, pp. 1–9.

[94] D. Marmanis, K. Schindler, J. Wegner, S. Galliani, M. Datcu, and U. Stilla, "Classification with an edge: Improving semantic image segmentation with boundary detection," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 135, pp. 158 – 172, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S092427161630572X

[95] S. Paisitkriangkrai, J. Sherrah, P. Janney, and A. V.-D. Hengel, "Effective semantic pixel labelling with convolutional networks and conditional random fields," in *2015 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, June 2015, pp. 36–43.

[96] D. Marmanis, J. D. Wegner, S. Galliani, K. Schindler, M. Datcu, and U. Stilla, "Semantic segmentation of aerial images with an ensemble of cnns," vol. III-3, pp. 473–480, 06 2016.

[97] N. Audebert, B. L. Saux, and S. Lefèvre, "Fusion of heterogeneous data in convolutional networks for urban semantic labeling (invited paper)," *CoRR*, vol. abs/1701.05818, 2017. [Online]. Available: http://arxiv.org/abs/1701.05818

[98] N. Audebert, B. L. Saux, and S. Lefvre, "Beyond rgb: Very high resolution urban remote sensing with multimodal deep networks," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 140, pp. 20 – 32, 2018, geospatial Computer Vision. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0924271617301818

[99] 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)*, July 2017, pp. 1561–1570.

[100] N. Audebert, B. L. Saux, and S. Lefèvre, "Joint learning from earth observation and openstreetmap data to get faster better semantic maps," *CoRR*, vol. abs/1705.06057, 2017. [Online]. Available: http://arxiv.org/abs/1705.06057

[101] D. Marcos, M. Volpi, B. Kellenberger, and D. Tuia, "Land cover mapping at very high resolution with rotation equivariant cnns: Towards small yet accurate models," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 145, pp. 96 – 107, 2018, deep Learning RS Data. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0924271618300261

[102] M. Volpi and D. Tuia, "Deep multi-task learning for a geographically-regularized semantic segmentation of aerial images," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 144, pp. 48 – 60, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0924271618301692

[103] S. Srivastava, M. Volpi, and D. Tuia, "Joint height estimation and semantic labeling of monocular aerial images with cnns," in *2017 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, 2017, pp. 5173–5176.

[104] M. Volpi and D. Tuia, "Dense semantic labeling of subdecimeter resolution images with convolutional neural networks," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 55, no. 2, pp. 881–893, 2017.

[105] D. Marmanis, K. Schindler, J. D. Wegner, M. Datcu, and U. Stilla, "Semantic segmentation of aerial images with explicit class-boundary modeling," in *2017 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, 2017, pp. 5165–5168.

[106] D. Marmanis, M. Datcu, T. Esch, and U. Stilla, "Deep learning earth observation classification using imagenet pretrained networks," *IEEE Geoscience and Remote Sensing Letters*, vol. 13, no. 1, pp. 105–109, 2016.

[107] S. Paisitkriangkrai, J. Sherrah, P. Janney, and A. van den Hengel, "Semantic labeling of aerial and satellite imagery," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 9, no. 7, pp. 2868–2881, 2016.

[108] J. Sherrah, "Fully convolutional networks for dense semantic labelling of high-resolution aerial imagery," *CoRR*, vol. abs/1606.02585, 2016. [Online]. Available: http://arxiv.org/abs/1606.02585

[109] M. Volpi and D. Tuia, "Semantic labeling of aerial images by learning class-specific object proposals," in *2016 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, 2016, pp. 1556–1559.

[110] V. Mnih, "Machine learning for aerial image labeling," Ph.D. dissertation, University of Toronto, 2013.

[111] S. Saito, Y. Yamashita, and Y. Aoki, "Multiple object extraction from aerial imagery with convolutional neural networks," vol. 60, pp. 10 402–1/10 402, 01 2016.

[112] E. Maggiori, Y. Tarabalka, G. Charpiat, and P. Alliez, "Convolutional neural networks for large-scale remote-sensing image classification," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 55, no. 2, pp. 645–657, Feb 2017.

[113] Y. Li, L. Guo, J. Rao, L. Xu, and S. Jin, "Road segmentation based on hybrid convolutional network for high-resolution visible remote sensing image," *IEEE Geoscience and Remote Sensing Letters*, vol. 16, no. 4, pp. 613–617, April 2019.

[114] V. Mnih and G. E. Hinton, "Learning to detect roads in high-resolution aerial images," in *Computer Vision – ECCV 2010*, K. Daniilidis, P. Maragos, and N. Paragios, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 210–223.

[115] P. Kaiser, J. D. Wegner, A. Lucchi, M. Jaggi, T. Hofmann, and K. Schindler, "Learning aerial image segmentation from online maps," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 55, no. 11, pp. 6054–6068, Nov 2017.

[116] E. Shelhamer, J. Long, and T. Darrell, "Fully convolutional networks for semantic segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 4, pp. 640–651, April 2017.

[117] J. Chen and A. Zipf, "Deepvgi: Deep learning with volunteered geographic information," in *WWW*, 2017.

[118] S. Kurath, "Osmdeepod - object detection on orthophotos with and for vgi," vol. Volume 2, pp. 173–188, online available: http://www.austriaca.at/?arp=0x00373589 - Last access:12.8.2018. [Online]. Available: http://www.austriaca.at/?arp=0x00373589

[119] "DeepOSM," https://github.com/trailbehind/DeepOSM, accessed: 2018-08-06.

[120] G. Mttyus, S. Wang, S. Fidler, and R. Urtasun, "Enhancing road maps by parsing aerial images around the world," in *2015 IEEE International Conference on Computer Vision (ICCV)*, Dec 2015, pp. 1689–1697.

[121] C. M. Gevaert, C. Persello, S. O. Elberink, G. Vosselman, and R. Sliuzas, "Context-based filtering of noisy labels for automatic basemap updating from uav data," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, pp. 1–11, 2018.

[122] Z. Li, J. D. Wegner, and A. Lucchi, "Topological map extraction from overhead images," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 1715–1724.

[123] F. Bastani, S. He, S. Abbar, M. Alizadeh, H. Balakrishnan, S. Chawla, S. Madden, and D. DeWitt, "Roadtracer: Automatic extraction of road networks from aerial images," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4720–4728.

[124] F. Bastani, S. He, S. Abbar, M. Alizadeh, H. Balakrishnan, S. Chawla, and S. Madden, "Machine-assisted map editing," in *Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 2018, pp. 23–32.

[125] H. Chu, D. Li, D. Acuna, A. Kar, M. Shugrina, X. Wei, M.-Y. Liu, A. Torralba, and S. Fidler, "Neural turtle graphics for modeling city road layouts," in *The IEEE International Conference on Computer Vision (ICCV)*, October 2019.

[126] X. Yang, X. Li, Y. Ye, R. Y. Lau, X. Zhang, and X. Huang, "Road detection and centerline extraction via deep recurrent convolutional neural network u-net," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 57, no. 9, pp. 7209–7220, 2019.

[127] E. Park *et al.*, "Refining inferred road maps using gans," Ph.D. dissertation, Massachusetts Institute of Technology, 2019.

[128] A. V. Etten, "City-scale road extraction from satellite imagery v2: Road speeds and travel times," in *The IEEE Winter Conference on Applications of Computer Vision (WACV)*, March 2020.

[129] A. Mosinska, M. Kozinski, and P. Fua, "Joint segmentation and path classification of curvilinear structures," *IEEE transactions on pattern analysis and machine intelligence*, 2019.

[130] X. Yang, X. Li, Y. Ye, X. Zhang, H. Zhang, X. Huang, and B. Zhang, "Road detection via deep residual dense u-net," in *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2019, pp. 1–7.

[131] A. C. Kak and C. Bouman, "Semantic segmentation of images with convolutional networks – lecture notes on deep learning, ece 695, purdue university," 2020, accessed: 2020-05-15.

[132] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015. [Online]. Available: http://arxiv.org/abs/1502.03167

[133] L. Ma, J. Stckler, C. Kerl, and D. Cremers, "Multi-view deep learning for consistent semantic mapping with rgb-d cameras," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2017, pp. 598–605.

[134] J. Xiao and L. Quan, "Multiple view semantic segmentation for street view images," in *2009 IEEE 12th International Conference on Computer Vision*, Sep. 2009, pp. 686–693.

[135] L. Ge, H. Liang, J. Yuan, and D. Thalmann, "Robust 3d hand pose estimation in single depth images: From single-view cnn to multi-view cnns," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[136] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller, "Multi-view convolutional neural networks for 3d shape recognition," in *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[137] 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 *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[138] A. Mortazi, R. Karim, K. Rhode, J. Burt, and U. Bagci, "Cardiacnet: Segmentation of left atrium and proximal pulmonary veins from mri using multi-view cnn," in *Medical Image Computing and Computer-Assisted Intervention MICCAI 2017*, M. Descoteaux, L. Maier-Hein, A. Franz, P. Jannin, D. L. Collins, and S. Duchesne, Eds. Cham: Springer International Publishing, 2017, pp. 377–385.

[139] G. Carneiro, J. Nascimento, and A. P. Bradley, "Unregistered multiview mammogram analysis with pre-trained deep learning models," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, Eds. Cham: Springer International Publishing, 2015, pp. 652–660.

[140] J. Han, H. Chen, N. Liu, C. Yan, and X. Li, "Cnns-based rgb-d saliency detection via cross-view transfer and multiview fusion," *IEEE transactions on cybernetics*, vol. 48, no. 11, pp. 3171–3183, 2017.

[141] G. Kang, K. Liu, B. Hou, and N. Zhang, "3d multi-view convolutional neural networks for lung nodule classificationq," *PLOS ONE*, vol. 12, no. 11, pp. 1–21, 11 2017. [Online]. Available: https://doi.org/10.1371/journal.pone.0188290

[142] M. Elhoseiny, T. El-Gaaly, A. Bakry, and A. M. Elgammal, "A comparative analysis and study of multiview cnn models for joint object categorization and pose estimation," in *ICML*, 2016.

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

[144] A. Dai and M. Niessner, "3dmv: Joint 3d-multi-view prediction for 3d semantic scene segmentation," in *The European Conference on Computer Vision (ECCV)*, September 2018.

[145] E. Kalogerakis, M. Averkiou, S. Maji, and S. Chaudhuri, "3d shape segmentation with projective convolutional networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 3779–3788.

[146] B. Le Saux, N. Yokoya, R. Hansch, M. Brown, and G. Hager, "2019 data fusion contest [technical committees]," *IEEE Geoscience and Remote Sensing Magazine*, vol. 7, no. 1, pp. 103–105, 2019.

[147] "Geoio," https://github.com/DigitalGlobe/geoio, accessed: 2020-05-15.

[148] "CPU Pinning in OpenStack Nova," https://docs.openstack.org/nova/pike/admin/cpu-topologies.html, accessed: 2020-06-03.

[149] "CPU Pinning – Wikipedia," https://en.wikipedia.org/wiki/Processor_affinity, accessed: 2020-06-03.

[150] "CPU Pinning in RedHat," https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/9/html/instances_and_images_guide/ch-cpu_pinning, accessed: 2020-06-03.

[151] "NUMA – Wikipedia," https://en.wikipedia.org/wiki/Non-uniform_memory_access, accessed: 2020-06-03.

[152] "PyTorch," https://pytorch.org/, accessed: 2020-06-03.

[153] "virsh," https://libvirt.org/manpages/virsh.html, accessed: 2020-06-03.

[154] "libvirt," https://libvirt.org/, accessed: 2020-06-03.

[155] "Distributed Architectures," https://docs.microsoft.com/en-us/learn/cmu-cloud-computing/cmu-distributed-programming-introduction/7-program-cloud-symmetry, accessed: 2020-06-05.

[156] "Data Parallel in PyTorch," https://pytorch.org/tutorials/beginner/blitz/data_parallel_tutorial.html, accessed: 2020-06-03.

[157] "Model Parallel in PyTorch," https://pytorch.org/tutorials/intermediate/model_parallel_tutorial.html, accessed: 2020-06-03.

[158] "Distributed Data Parallel in PyTorch," https://pytorch.org/tutorials/intermediate/ddp_tutorial.html, accessed: 2020-06-03.

[159] "BatchNorm2d in PyTorch," https://pytorch.org/docs/master/generated/torch.nn.BatchNorm2d.html, accessed: 2020-06-03.

[160] "Synchronized Multi-GPU Batch Normalization in PyTorch," https://github.com/tamakoji/pytorch-syncbn, accessed: 2020-06-03.

[161] "Flyby Videos of Large-Area DSMs," https://engineering.purdue.edu/RVL/CORE3D/webpage/index.php, accessed: 2019-12-15.

[162] Y. Zhu, K. Sapra, F. A. Reda, K. J. Shih, S. Newsam, A. Tao, and B. Catanzaro, "Improving semantic segmentation via video propagation and label relaxation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 8856–8865.

[163] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[164] W. Li, C. He, J. Fang, and H. Fu, "Semantic segmentation based building extraction method using multi-source gis map datasets and satellite imagery," in *CVPR Workshops*, 2018, pp. 238–241.

[165] S. Patil, B. Comandur, T. Prakash, and A. C. Kak, "A New Stereo Benchmarking Dataset for Satellite Images," 2019.

[166] S. Liu, W. Ding, C. Liu, Y. Liu, Y. Wang, and H. Li, "Ern: edge loss reinforced semantic segmentation network for remote sensing images," *Remote Sensing*, vol. 10, no. 9, p. 1339, 2018.

VITA

VITA

Bharath Kumar Comandur received his Bachelor of Technology degree in Electrical Engineering from the Indian Institute of Technology Madras in 2012. Since 2013, he has been pursuing his PhD in the Robot Vision Lab at Purdue University. His research interests include computer vision, machine learning, remote sensing, cloud computing and developing open-source software.