AFRL-RY-WP-TR-2020-0101

A STUDY ON APPLYING LEARNING TECHNIQUES TO REMOTE SENSING DATA

Aswathnarayan Radhakrishnan
The Ohio State University

MAY 2020
Final Report

STINFO COPY

AIR FORCE RESEARCH LABORATORY
SENSORS DIRECTORATE
WRIGHT-PATTERSON AIR FORCE BASE, OH 45433-7320
AIR FORCE MATERIEL COMMAND
UNITED STATES AIR FORCE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
May 2020	Thesis	14 February 2020 –14 February 2020

4. TITLE AND SUBTITLE
A STUDY ON APPLYING LEARNING TECHNIQUES TO REMOTE SENSING DATA

5a. CONTRACT NUMBER
N/A

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
N/A

6. AUTHOR(S)
Aswathnarayan Radhakrishnan

5d. PROJECT NUMBER
N/A

5e. TASK NUMBER
N/A

5f. WORK UNIT NUMBER
N/A

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
The Ohio State University
281 W. Lane Ave.
Columbus, Ohio 43210

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Research Laboratory
Sensors Directorate
Wright-Patterson Air Force Base, OH 45433-7320
Air Force Materiel Command
United States Air Force

10. SPONSORING/MONITORING AGENCY ACRONYM(S)
AFRL/RYAP

11. SPONSORING/MONITORING AGENCY REPORT NUMBER(S)
AFRL-RY-WP-TR-2020-0101

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES
PAO case number 88ABW-2020-0545, Clearance Date 14 February 2020. © 2020 The Ohio State University. A thesis presented in partial fulfillment of the requirements for the Degree of Master of Science in Computer Science and Engineering at the Graduate School of The Ohio State University. This work was funded in whole or in part by Department of the Air Force. The U.S. Government has for itself and others acting on its behalf an unlimited, paid-up, nonexclusive, irrevocable worldwide license to use, modify, reproduce, release, perform, display, or disclose the work by or on behalf of the U.S. Government. Report contains color.

14. ABSTRACT

A major issue with data-hungry deep learning algorithms is the lack of annotated ground truth for specific applications. In this thesis, we explore the challenges of applying artificial intelligence techniques for remote sensing data which lacks the availability of large annotated datasets for training in comparison to regular imagery data. We first tackle the problem of improving object tracking in Wide Area Motion Imagery data by using a semantic segmentation model to detect false tracker points on buildings. The combination of image understanding techniques with tracking has the potential to improve track quality and aid automatic situation assessment. We use a manually annotated dataset which confined us to work with a small dataset. We propose a solution for this problem by developing a framework for automated annotation of remote sensing data. We pick satellite imagery due to the high volume of Earth Observation Satellites available today, coupled with crowd-sourced map data that can enable a new means for automated annotation of remote sensing data for applications previously constrained by the lack of available datasets. In the second part of this thesis, we present an automated pipeline for collecting and labeling satellite imagery to facilitate building custom deep learning models. We demonstrate this approach by automatically collecting labeled imagery of solar power plants and building a classifier to detect the presence of such structures. This framework can be used to collect labeled satellite imagery of any object mapped by spatial databases.

15. SUBJECT TERMS
computer vision

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT:	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON (Monitor)
a. REPORT	b. ABSTRACT	c. THIS PAGE			Harris Hall
Unclassified	Unclassified	Unclassified	SAR	96	19b. TELEPHONE NUMBER *(Include Area Code)* N/A

A Study on Applying Learning Techniques to Remote Sensing Data

A Thesis

Presented in Partial Fulfillment of the Requirements for the Degree Master of Science in the Graduate School of The Ohio State University

By

Aswathnarayan Radhakrishnan, B. Tech Information Technology

Graduate Program in Computer Science and Engineering

The Ohio State University

2020

Master's Examination Committee:

Dr. James W. Davis, Advisor

Dr. Roger Crawfis

Dr. Roman Ilin

Abstract

A major issue with data-hungry deep learning algorithms is the lack of annotated ground truth for specific applications. In this thesis, we explore the challenges of applying artificial intelligence techniques for remote sensing data which lacks the availability of large annotated datasets for training in comparison to regular imagery data. We first tackle the problem of improving object tracking in Wide Area Motion Imagery data by using a semantic segmentation model to detect false tracker points on buildings. The combination of image understanding techniques with tracking has the potential to improve track quality and aid automatic situation assessment. We use a manually annotated dataset which confined us to work with a small dataset. We propose a solution for this problem by developing a framework for automated annotation of remote sensing data. We pick satellite imagery due to the high volume of Earth Observation Satellites available today, coupled with crowd-sourced map data that can enable a new means for automated annotation of remote sensing data for applications previously constrained by the lack of available datasets. In the second part of this thesis, we present an automated pipeline for collecting and labeling satellite imagery to facilitate building custom deep learning models. We demonstrate this approach by automatically collecting labeled imagery of solar power plants and building a classifier to detect the presence of such structures. This framework can be used to collect labeled satellite imagery of any object mapped by spatial databases.

This is dedicated to my parents, family, and friends.

Acknowledgments

I would like to start by thanking my advisor, Dr. Jim Davis, for his constant support, guidance, and the countless moments of encouragement throughout my time at OSU. I have been working with Dr. Davis since my first year and could not have hoped for a better advisor. He masterfully guided me through my research providing direction when needed, while consistently giving me the freedom to explore my own ideas. I would also like to thank Dr. Roman Ilin for funding my work and giving me the opportunity to work on projects within my interests.

Furthermore, I want to thank Jamie and Sam for helping with the curation of this thesis. Jamie developed the labeling tool and assisted with handling OSM data. Sam helped train the semantic segmentation model. I would also like to thank my fellow labmates Tong, James, and Logan for creating an exciting workplace environment where I spent the majority of my time in grad school.

I wouldn't be where I am now if not for my parents Radhakrishnan and Mala. They have always wanted the best for me and I couldn't be more grateful for such a wonderful support system. I would also like to thank my uncle Sreedhar and my aunt Suchi for helping me move to the US for grad school and for the countless times making me feel at home in this new country. This wouldn't have been possible without them. And finally, I would like to thank all my family and friends who have supported me through this amazing phase of my life.

Vita

May 2018 ..... B.Tech Information Technology
College of Engineering Guindy
Anna University, Chennai, India

August 2018- Present ..... Graduate Research Associate
Computer Vision Lab
Ohio State University

Publications

Research Publications

R. Ilin, J. Davis, S. Lerner, J. Cunningham, and A. Radhakrishnan "Tracking with Context in Wide Area Motion Imagery". *MSS Passive Sensors Symposium*, Feb. 2019.

A. Radhakrishnan, J. Cunningham, J. Davis, and R. Ilin "A Framework for Collecting and Classifying Objects in Satellite Imagery". *International Symposium on Visual Computing*, Oct. 2019.

Fields of Study

Major Field: Computer Science and Engineering

Contents

List of Tables

Table **Page**

List of Figures

Chapter 1: Introduction

Remote sensing is the process of obtaining information about an object without making physical contact with the object thus bypassing the necessity for local observation when it is difficult or not feasible. Remote sensing is typically carried out from aircraft or satellites using instruments that collect data by detecting the energy that is reflected from Earth's surface. These instruments can be categorized into two types as passive or active sensors as shown in Fig. 1.1. Passive sensors such as cameras read the natural energy that is reflected off the Earth's surface whereas active sensors such as RADAR emit signals and record their interactions with Earth's surface features. Remote sensing has applications in various areas like geography, land surveying, hydrology, ecology, meteorology, oceanography, glaciology, etc. Additionally, remote sensing has a lot of military, intelligence, and humanitarian applications (forest fires, disaster management, etc.).

One of the most popular applications of remote sensing is the ability to obtain aerial and satellite imagery of the Earth's surface. In recent times, there has been an exponential increase in the public availability of aerial and satellite imagery data which in turn creates the need for computerized analysis of such data as manual analysis of a large number of images is a time-consuming task.

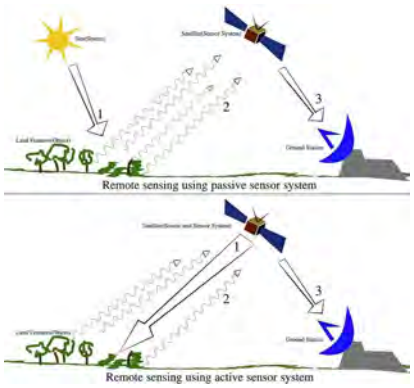


Figure 1.1: Active and Passive Remote Sensing.

The current trend for automated image analysis is the adoption of Artificial Intelligence (AI) learning techniques for Computer Vision such as image classification, object detection, semantic segmentation, etc. to understand the content and context of an image. But the performance of these common AI techniques goes hand in hand with the availability of large amounts of data for training. This is the reason why such AI techniques have not been commonly used for remote sensing imagery data when compared to regular imagery captured by handheld cameras. This is due to the fact that datasets for regular imagery are found in abundance these days with large datasets such as ImageNet containing over 14 million annotated images. Hence advancements in deep learning methods for computer vision tasks are creating profound changes in methods for analysis and inference of regular imagery. But this is not the case when it comes to remote sensing datasets that have only recently been available because their dataset sizes are nowhere close to the size of ImageNet like datasets to promote deep learning methods. Also, most of these remote sensing datasets lack annotated ground truth information required for training AI models. In this thesis,

Figure 1.2: Example WAMI frame from CLIF 2007 dataset.

we explore the challenges and possible solutions for applying certain AI techniques such as image classification and semantic segmentation to manually annotated remote sensing data and then we propose a pipeline for automated collection of annotated remote sensing datasets and demonstrate its applicability to facilitate the training of deep learning models.

In the first part of this thesis, we use a form of aerial imagery known as Wide Area Motion Imagery (WAMI). WAMI typically uses an airborne camera system composed

of multiple sensors to capture and archive imagery with a very large field of view as shown in Fig. 1.2 for extended periods of time and is used for surveillance, reconnaissance, and intelligence applications. Detection, tracking, and classification of dynamic targets in motion imagery is a fundamental computer vision technique essential for numerous military applications such as intelligence analysis, autonomous flight, etc. It involves the method of linking points where specific objects are located temporally in a sequence of images. Tracking objects is a challenging task in WAMI data that is produced by large-format sensors such as Gorgon Stare [1]. WAMI data typically cover large areas however at lower resolution and frame rates. Some WAMI datasets also contain only grayscale imagery which makes it harder to distinguish objects. Additional challenges are introduced by camera noise, frame stitching (mosaicing tiles of images obtained from multiple cameras), and parallax errors introduced by the vatage point and orientations of the camera sensors. This results in tracking software often predicting false positive tracker points on tall buildings as shown in Fig. 1.3

Tracking is often performed with the purpose of detecting movements of interest. In order to move towards the automatic characterization of activities, tracking information has to be supplemented by contextual information of where the particular activity is happening in an image. For example, a vehicle driving fast on the road is not unusual but the same vehicle driving fast through a parking lot may indicate an anomaly. A combination of tracking and image understanding is required to achieve an automatic situation assessment. In this thesis, we examine introducing contextual information by combining tracking outputs and semantic segmentation results. Semantic segmentation is the task of assigning a classification label to every pixel of an image as shown in Fig. 1.4. This by itself is a challenging computer vision technique

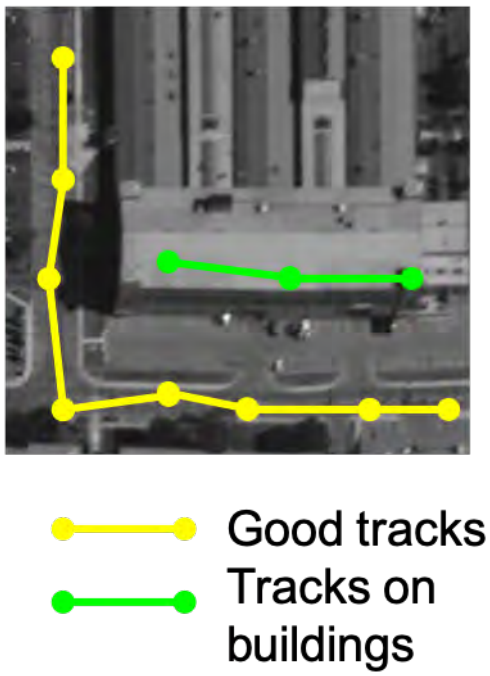


Figure 1.3: Tracking in WAMI data.

that is an ongoing field of research. Recently, there has been significant progress in semantic segmentation for regular imagery in the academic and commercial sectors. We conjecture that utilizing semantic labels in aerial imagery, particularly in WAMI data can contribute to the minimization of false alarm rates in tracker outputs and thereby an increase in target classification accuracy. At the same time, semantic labels can be employed to enhance situation understanding and anomaly detection. In this thesis, we examined the effects of using a segmentation mask of buildings in WAMI data to filter out false tracker points. We used the CLIF 2007 [2] and the UNICORN 2008 [3] WAMI datasets collected by the Air Force Research Laboratory in this thesis. The work involved the creation of training, validation, and testing datasets. Since these datasets lacked ground truth information for semantic segmentation, we manually

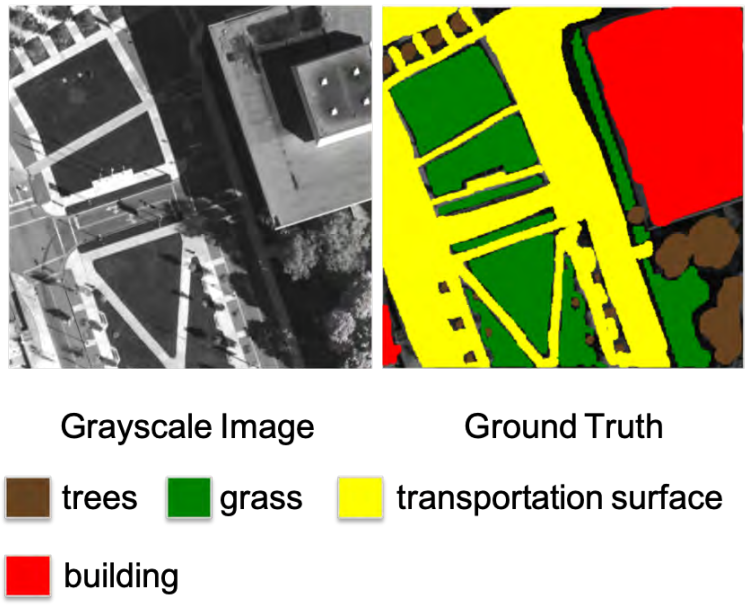


Figure 1.4: Semantic Segmentation in WAMI data.

annotated portions of the images that contain buildings. We utilized a state of the art Deep Neural Network architecture and retrained it to perform semantic segmentation on the WAMI data. We used existing efficient tracking software developed by Kitware [4] to predict tracker points in WAMI data. The results of using the building segmentation mask demonstrate that the semantic information can reduce the false alarm rate by removing tracker points from unlikely locations such as the roofs of buildings.

The performance of our semantic segmentation model in the previous task was constrained by the small size of the training dataset. The burden of manual annotation limited us to working with a small set of annotated images. However supervised machine learning tasks are highly dependent on the availability of large, labeled training datasets. Hence, there is always a need to find a dataset that is sufficiently large

enough to train a model and broad enough to contain diverse data samples to ensure better generalization performance. Satellite imagery is one such source of expansive and rich data. Earth Observation (EO) satellites are launched into space with their primary mission being remote sensing and providing satellite imagery of the Earth's surface from low Earth orbits. The number of EO satellites has increased exponentially over recent years. The data from these satellites have various applications in fields such as environmental monitoring, disaster management, agricultural engineering, cartography, and military intelligence. EO satellites have instruments capable of sensing a wide range of bands in the electromagnetic spectrum as shown in Fig. 1.5. Another important property of satellite imagery is the frequency in which new data is captured. A primary advantage of satellite imagery is that archived imagery of multiple spectral bands at high resolution can be collected and processed.

Although there has been a considerable increase in the volume of publicly available satellite imagery, it remains difficult to find a dataset that contains examples of a specific object for a particular task. Similarly, generating new datasets leads to the additional overhead of annotation. We have seen in the previous task that manual annotation is a time-consuming task that becomes difficult with large amounts of imagery. However, public geographical databases and crowd-sourced map information can be used to automatically locate objects and label the ground truth. In the second part of this thesis, we present a generic framework for extracting satellite imagery of specified object categories using public databases. We demonstrate the approach by automatically extracting satellite imagery of solar power plants from around the world and building a classifier to detect them in novel satellite imagery.

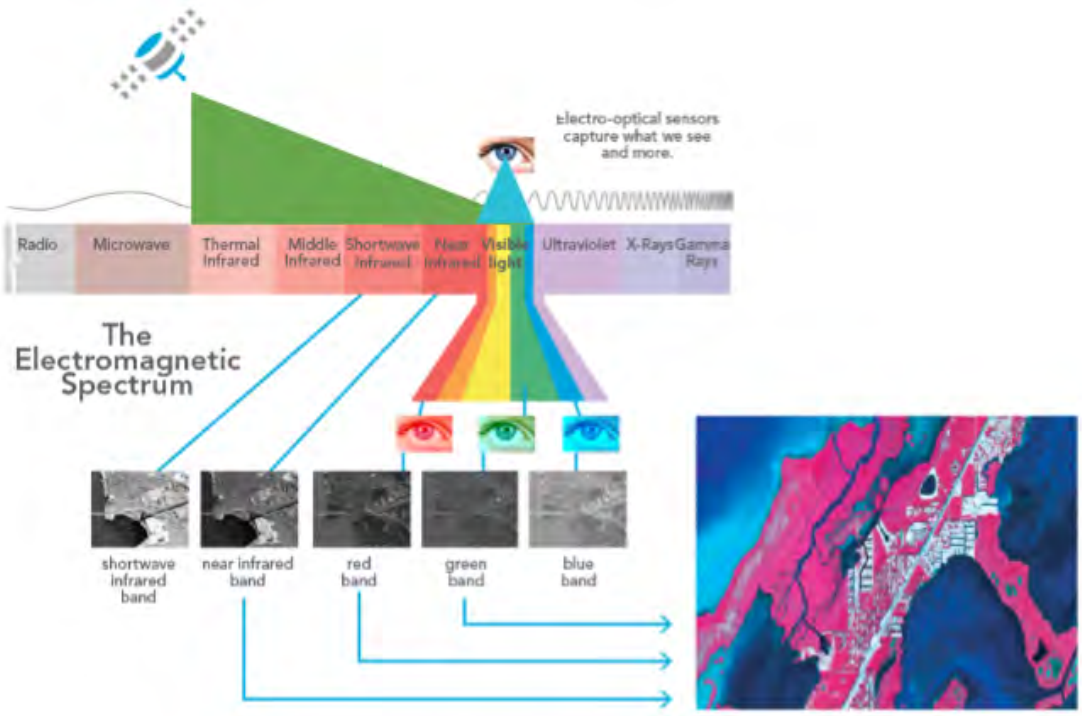


Figure 1.5: Electromagnetic Spectrum measured by EO Satellites.

1.1 Contributions

This thesis presents multiple methods of applying AI learning techniques to remote sensing data such as WAMI and satellite imagery for the purpose of improving tasks such as object tracking in WAMI data and image classification in satellite imagery. Specifically, the novel contributions of this thesis are:

- A method of using contextual information obtained from performing semantic segmentation in WAMI data to improve tracking performance by filtering out false positive tracker points.

- A modular pipeline for automatically collecting satellite imagery of objects mapped by geographical databases.

- A method of accurately and efficiently localizing an object's geo-coordinates by combining information obtained from geographical databases and OpenStreetMap data [5] thus avoiding the overhead of having to search through the whole world to locate an object.

- A method of automatically downloading Sentinel-2 satellite imagery [6] of any geo-location obtained from OpenStreetMap captured in conditions within the specified cloud cover percentage within a specified time period.

- A method of collecting satellite imagery of negative examples (Images not containing a particular object) by using OpenStreetMap to ensure a particular object is not present in an image.

Overall, this thesis explores the challenges of using AI learning techniques for solving problems in remote sensing data and presents a solution of employing automated annotation of remote sensing data to assist in training task-specific deep learning models.

1.2 Outline

We will describe each method presented above in the following chapters. We begin with a brief review of previous related work in applying learning techniques to remote sensing data in Chapter 2. The work in this thesis is divided into one chapter for improving tracking performance using contextual information, Chapter 3, and another for developing a framework for automated collection of annotated

satellite imagery of objects, Chapter 4. In Chapter 3, we discuss the methodology we used to enhance the performance of the Kitware Tracker in WAMI data by training a semantic segmentation model on manually annotated WAMI data to predict a mask of buildings. We demonstrate the success of this approach by comparing tracker performance before and after masking tracker points on buildings. We end with a discussion on the challenges we faced due to the difficulty of manual annotation of training data which led to working with a small annotated training dataset. In Chapter 4, we follow up with our proposed solution of building a data collection pipeline for automatically generating annotated satellite imagery datasets of objects mapped by geo-databases to help train task-specific deep learning models. We then examine the components that make up our pipeline in detail. We go on to demonstrate the applicability of our pipeline by generating a dataset of solar power plants across the world in satellite imagery and training an image classification model on the dataset to detect solar power plants in unseen imagery. We conclude with Chapter 5 wherein we summarize the progress made in this thesis and proposing directions for future work in this area.

Chapter 2: Related Work

In recent years, there has been an exponential rise in the availability of public remote sensing data leading to increased research involving the application of artificial intelligence (AI) techniques to remote sensing data. In particular, there has been an emphasis on tasks such as object tracking and classification in remote sensing data, which help promote automated analysis of remote sensing data. It is widely believed that automated analysis will be an important area for future research in remote sensing data. In this chapter, we provide an overview of previous works related to methods implemented in this thesis.

2.1 Object Tracking in Wide Area Motion Imagery

The first part of this thesis deals with improving object tracking performance in wide area motion imagery (WAMI) which is an active area of research. We make use of the results from the Kitware tracker built by Basharat et. al. [4] in this thesis. They developed a distributed tracking system for efficient handling of large scale WAMI data. The tracker used a multi-frame object detection algorithms that processed each WAMI data tile independently to produce high-quality tracks. Ling et. al. [1] assess the performance of various state-of-the-art visual tracking methods on grayscale WAMI data for tracking vehicles in an urban environment. They used a

low frame rate dataset and analyzed the problems faced by trackers built for regular video faced in WAMI data. Their experiments identified that background registration helped enhance tracker performance significantly. Blasch et. al. [7] performed a comprehensive study on several techniques for improving tracking in WAMI data. They review methods such as geo-registration, compression, and contextual analysis for WAMI tracking enhancements. The small pixel resolution of objects such as vehicles in WAMI data results in the poor performance of context-independent tracking methods. This led to developments in tracking techniques using contextual information in WAMI data. Blasch et. al. [8] also surveyed various methods of applying context in the domain of target tracking. They reviewed literature that uses existing road information to assist in tracking vehicles in WAMI data. Most of the literature stayed away from utilizing supervised learning techniques due to the lack of ground truth information for WAMI datasets. We proposed a method of using contextual information obtained from training a semantic segmentation model on manually annotated WAMI data.

2.2 Semantic Segmentation

Semantic segmentation by itself is a challenging computer vision task that assigns a class label to each pixel in an image. It requires large annotated datasets to train deep neural network models. Long et. al. [9] proposed a method of extending fully convolutional networks used for image classification to perform semantic segmentation. They trained end-to-end (from input pixels to output pixels) models using in-network upsampling layers to interpolate the dense features obtained from the feed-forward convolutions and predict class labels for each pixel in the image. The task

of semantic segmentation becomes even more difficult in the case of remote sensing imagery such as WAMI data. McDonald et. al. [10] explored the technical challenges of applying machine learning techniques to wide area surveillance data such as lack of ground truth data for remote sensing datasets. In the first part of this thesis, we manually labeled ground truth data for an existing WAMI dataset to train a semantic segmentation model. The network architecture we used was adapted from the U-Net model proposed by Ronneberger et. al. [11] for semantic segmentation in biomedical imaging. Medical imagery also suffers from a lack of large training datasets. Hence they developed a fully convolutional network that works with a small number of labeled training images by using a sliding window approach that uses a patch of pixels around the pixel whose class is to be predicted as input thus augmenting the dataset significantly. This model seemed to be a perfect fit for our WAMI dataset which similarly contained few training examples.

Manual annotation of WAMI data proved to be a highly labor-intensive task. The second part of our thesis provides a solution to this problem by developing an automated annotation approach for remote sensing data. The large troves of satellite imagery data available today seemed to be the ideal platform to implement our proposed approach.

2.3 Deep Learning for Analysis of Remote Sensing Data

The large data sources of satellite imagery have already advocated research in the use of deep learning algorithms for analyzing remote sensing data. Albert et. al. [12] focused on predicting land use labels such as urban areas, forests, airports, etc. in satellite imagery of urban areas using state of the art convolutional neural networks

(CNN). However, they were restricted to train with a small, manually-created dataset for 6 European cities. Similarly, in Kussul et. al. [13], an image classifier was trained to predict crop types using LANDSAT-8 and Sentinel-1 satellite imagery. They used manual ground survey data to label ground truth data. They trained an ensemble of CNNs for performing classification in multisource multitemporal remote sensing data. Ishii et. al. [14] developed a CNN for detecting solar power plants in multispectral satellite imagery. They trained their model on a geographically restricted dataset and the ground truth information was manually annotated. Also, there was a large bias between the number of positive and negative examples. All these works were constrained by the overhead of working with manually annotated data. The data is also restricted to small geographical areas.

Our proposed solution enables the collection of automatically annotated data globally promoting the use of deep learning models for new applications in remote sensing data. We make use of OpenStreetMap data to locate and annotate objects in satellite imagery.

2.4 OpenStreetMap Data

Crowd-sourced map information has become widely accessible today and OpenStreetMap (OSM) is one such popular crowd-sourced spatial database primarily used for the purpose of land use mapping. Sui et. al. [15] presented a detailed study about the growth of volunteered geographic information (VGI) projects such as OSM and their impacts on remote sensing applications. This promoted research in the area of using OSM data for annotating remote sensing imagery. Johnson et. al. [16] used existing manually filtered OSM polygonal datasets for labeling footprints of land use

classes such as water, farm, grass, etc. on Landsat-8 imagery. OSM datasets are more prone to errors than land use information obtained from official sources due to its crowdsourced nature. They evaluated the performance of several traditional classification techniques such as Naive Bayes, decision tree, and random forest classifiers on the OSM labeled datasets. Their results proved that even noisy OSM data is a reliable source for labeling training datasets. Schultz et. al. [17] proposed a method of using OSM data for filling gaps in existing land cover maps. They used VGI data from OSM to train machine learning models to predict land use labels on remote sensing data and use the trained model to predict land cover gaps. Audebert et. al. [18] studied fusing OSM raster images to superimpose land use labels on satellite imagery for training fusion-based deep learning architectures. Similarly, Kaiser et. al. [19] developed an approach of using OSM raster data to superimpose labels of roads and buildings on existing satellite imagery to perform semantic image segmentation. Zhao et. al. [20] also integrated OSM data with high spatial resolution imagery to classify types of buildings. We see that there has been considerable research in the area of using OSM data for extracting ground truth knowledge.

However, most of these approaches suffered from the lack of large problem-specific labeled training examples, which has been a bottleneck for applying deep learning techniques. Even those employing OSM data used existing filtered OSM datasets which restrict training data. Our proposed framework can dynamically query the OSM database for the required objects of interest thus bypassing the overhead of having to download and filter the entire OSM database. Additionally, our proposed model extracts OSM data in vector format allowing us to work with the object boundaries (points, lines, or polygons). We can also exploit the broader categories of features

mapped by OSM other than the popular classes such as roads and buildings. Furthermore, we exploit the use of additional geo-location data sources to help localize the objects to a smaller search region on OSM to avoid having to search throughout the whole world to locate an object. The above enhancements from previous works constitute the novel contributions of this thesis.

Chapter 3: Tracking with Context in Manually Annotated Wide Area Motion Imagery Data

In this chapter, we introduce our method of using contextual information to improve tracking performance of the Kitware tracker in wide area motion imagery (WAMI) by using a semantic segmentartion model to predict a mask of buildings in WAMI data and then filtering out the false tracker points on buildings. This method is able to considerably improve the accuracy of the tracker when compared to the unfiltered tracker output. Figure 3.1 shows the overall framework of our proposed method.

We start by discussing the datasets we work with and the preprocessing steps necessary as the datasets have no ground truth data for semantic segmentation. We then examine our approach to manually annotate the training dataset and train a semantic segmentation model that can predict a building mask on WAMI data. We then discuss the post-processing techniques we used to improve the building mask obtained from the semantic segmentation model. We end with a review of the results of applying the building mask to the output of the Kitware tracker.

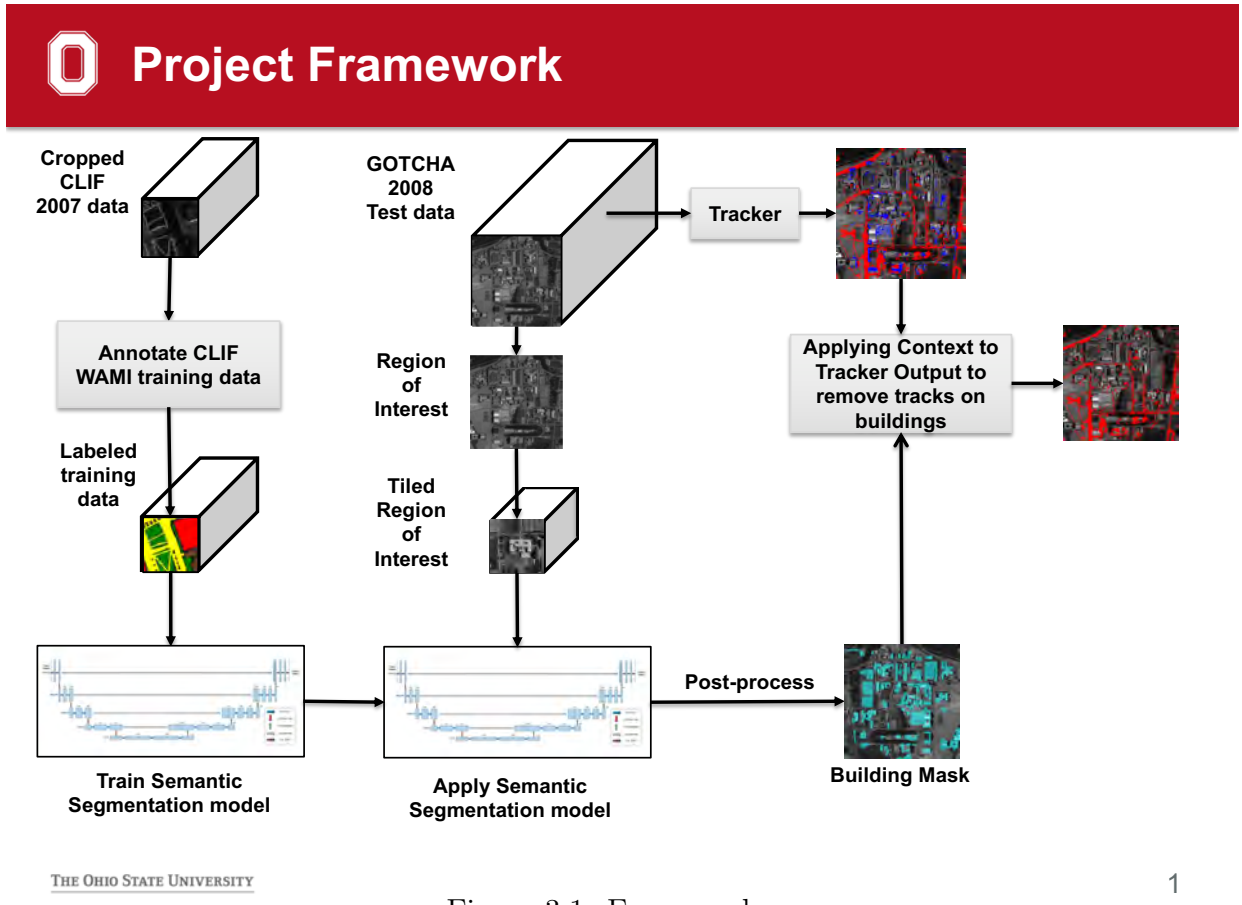


Figure 3.1: Framework.

3.1 Datasets Description

In this work, we examined the ways to improve the performance of the Kitware Tracker on the UNIfied COincident Optical and Radar for recognitioN (UNICORN) 2008 dataset which is a multi-modality, partially truthed, simultaneous and coincident dataset, using both a WAMI large format electro-optical (EO) sensor, and Wide Area Synthetic Aperture Radar (SAR). We just used the WAMI part of the dataset in this work to perform the tracking and analysis. The EO data in UNICORN 2008 was collected over the Wright-Patterson Air Force Base using a similar sensor to what was used for the Columbus Large Image Format (CLIF) 2007 data collection which was collected in October 2007 over the Ohio State University (OSU) campus. In

(a) UNICORN 2008. (b) CLIF 2007.

Figure 3.2: Sample Crops from Datasets.

order to train a semantic segmentation model, we need some training data similar to the data on which the model is to be tested. Hence, the CLIF 2007 data seemed to be the perfect fit for this task as it was collected with the same camera in an urban environment with many large buildings and residential areas with smaller houses, thus providing different types of building data for training. However, the operating conditions, such as the geographical location and the lighting were different. Figure 3.2 compares sample crops from the two datasets used in this work.

The sensor used in both datasets is a large format monochromatic electro-optical sensor comprised of a matrix of six cameras. The size of the matrix is 2 rows by 3 columns. Cameras 3, 1, and 5 make-up the top row of the image, respectively. Cameras 2, 0, and 4 make-up the bottom row of the image. Each camera was oriented in such a way as to maximize coverage, yet allow enough overlap between images to help in mosaicking the image to form a larger image. The overlap between the

Figure 3.3: CLIF Multi-Camera Images Orientation

cameras is approximately 50 pixels. The cameras for this data collection, obtain data at approximately 2 frames per second. To better understand the camera orientation imagine that you are looking through the camera head, you would see what is shown in the Fig. 3.3. In order to optimize the performance of the camera system, the data was stored inverted so each image has to be rotated. The images on the top row are rotated counterclockwise 90 degrees while the images on the bottom row are rotated clockwise 90 degrees. Camera numbers have been embedded in the figure below for illustrative purposes only, the camera numbers are not stored in the imagery.

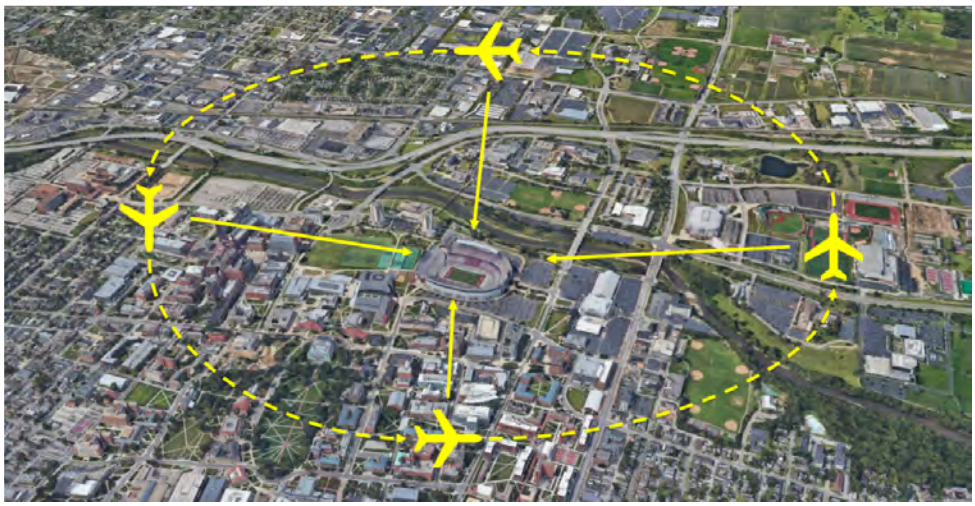


Figure 3.4: Selected Vantage Points of CLIF Imagery.

3.2 Preprocessing Steps

The CLIF 2007 data we used contains unprocessed (raw) data for each of the six cameras. Each raw image is 4,016x2,672 pixels, with an 8-bit per pixel grayscale raw (resulting in an approximate image size of 11 MB for each camera view). The CLIF dataset contains temporal frames taken over several hours. We selected 4 temporally spaced frames (showing a different viewing angle of the scene from vantage points as shown in Fig. 3.4) for each of the six cameras, thus giving a total of 24 images. We then randomly cropped 42 sub-images of size 512x512 from each of the 24 images, giving us a total of 1008, 512x512 sub-images. An example of our cropped subimage is shown in Fig. 3.5

Each image was preprocessed using Adaptive Equalization [21] followed by data whitening. Adaptive equalization was chosen to account for the drastic brightness changes across the images in CLIF, primarily caused by reflective surfaces. The

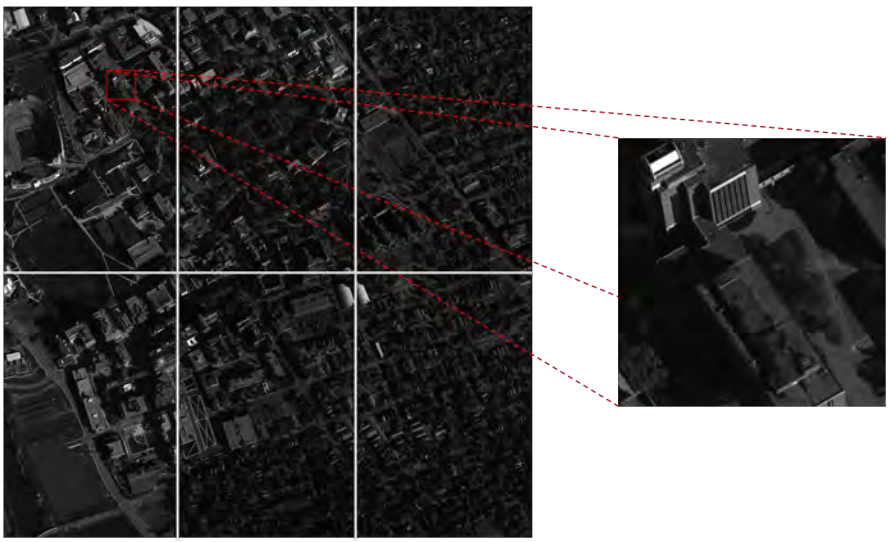


Figure 3.5: Example Cropped CLIF Sub-Image.

mean and variance for the whitening normalization were computed globally on the equalized CLIF2007 dataset.

The UNICORN 2008 data we used, on the other hand, was already processed and provided in the National Image Transmission Format (NITF) format, which is the Department of Defense (DoD) standard for imagery. A NITF file is a large header with the data concatenated onto the rear of the file. We used the Geospatial Data Abstraction Library (GDAL) library for reading and working with the NITF images. The EO data's ground coverage moves around quite a bit but seems centered on the southeast corner of the Wright-Patterson Air Force Base Area B runway triangle. Using GDAL in a Python script, we cropped out a region of interest of the base to evaluate our model. The geo-coordinates of the upper-left corner (origin) is (39.794308, -84.100641). We generated an image which from here on we will refer to

Figure 3.6: Region of Interest in UNICORN 2008 Dataset.

as the "region-of-interest". The size of the image is 6144x6144, with the origin point as the upper left corner, as shown in Fig. 3.6.

3.3 Annotating Training Data

The main hassle of using the CLIF and UNICORN data was the lack of any ground truth labels for the task of semantic segmentation which requires a training dataset that contains imagery with class labels for each pixel in an image. We developed a custom-built labeling tool in Python which can load in the imagery and allows us to use a paintbrush tool to highlight the parts of the image containing buildings. The remaining portions of the image are considered to be non-building pixels. We used

this tool to manually annotate the 1008 sub-images we generated from the CLIF 2007 dataset to create a labeled training dataset. This was a time-consuming process and hence we had to limit our training set to contain only 1008 images. We then came across the presence of dead pixels (black lines) in 12 images which we removed from the set. The final training set consisted of 996 sub-images along with their pixel-wise ground truth labels (building/non-building).

3.4 CNN Architecture

The CNN we used for this task was an adaptation of the U-Net architecture, as specified in [11]. We made this decision primarily because of its relatively small number of parameters when compared to other models, which is a perfect fit considering the small size of our training set. Many other models employ ImageNet-trained backbones. However, given how different the WAMI imagery is from the ImageNet dataset, we decided not to use a pre-trained neural network. The network we used employs four "downsampling blocks" followed by four "upsampling blocks" as shown in Fig. 3.7. Each downsampling block is composed of two convolutional layers followed by a max pooling layer with the stride set to downsample each feature volume by two. Both convolutional layers employ the same number of filters. Each upsampling block is made up of a convolutional layer followed by an upsampling layer (to interpolate each feature volume to twice its size) and two convolutional layers. The last two convolutional layers both employ half the number of filters as the initial convolutional layer.

The first downsampling block uses 64 filters for both of its convolutional layers and each subsequent downsampling block doubles this number so that there are 512

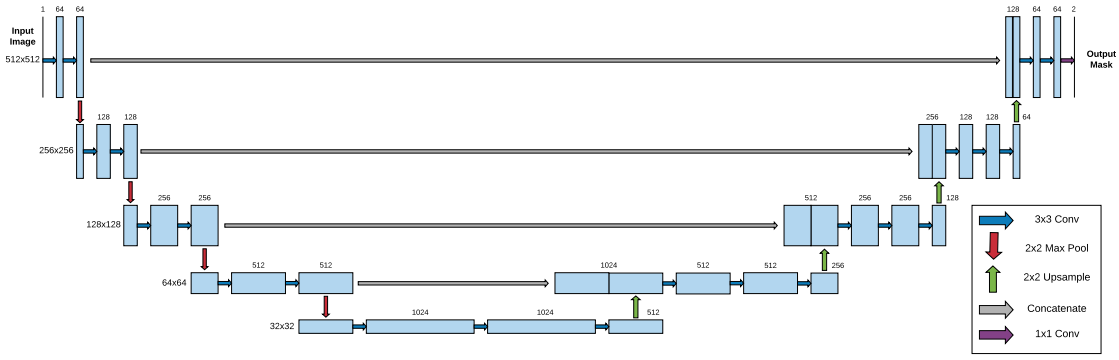


Figure 3.7: U-Net Based Network Architecture.

filters in the last block. The upsampling blocks reverse this process so that the last upsampling block makes use of 64 filters. Between the last downsampling block and the first upsampling block, there are two convolutional layers, each with 1024 filters, to obtain more high-level semantic information before beginning the upsampling and refinement process. After the last upsampling block, we employ a 1x1 convolutional layer followed by a point-wise softmax layer to approximate the per-class softmax scores for each pixel.

3.5 Training

The initial approach we used was to train a model for a binary problem of building vs. not-building. The model was trained with the Adam optimizer using an image batch size of 8. We applied The Adam parameters specified in Kingma et al. [22]. We ran the training script until the validation loss did not improve for 15 epochs. This condition was met at roughly 40 epochs. The standard categorical cross-entropy loss function was used for the model loss. We trained three models with random

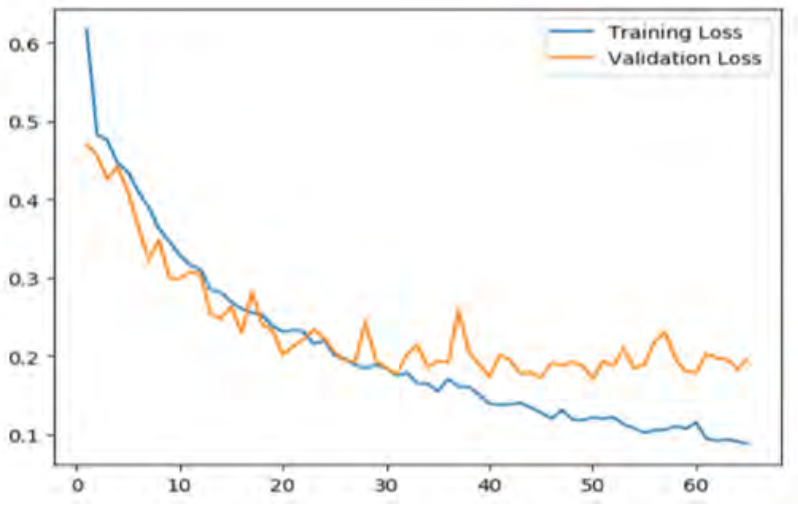


Figure 3.8: Plot of the Model Loss in the Training and Validation Set.

initialization using the exact procedure specified above and combined them later to improve the robustness of the model given then small size of the training data. Figure 3.8 shows the plot of the model loss in the training and validation set over training epochs.

3.6 Segmentation Results on UNICORN 2008

After the training was completed, a mask was generated for the EO WAMI data in the UNICORN 2008 dataset using multiple averaging techniques. Initially, two sets of sub-images (tiles) were generated from the region of interest in the UNICORN dataset: one set contained 144 tiles of size 512x512 starting at the upper-left-hand corner, and another set with 121 tiles of size 512x512 starting from a point shifted by 256 pixels from the top and the left of the original region of interest image. An example of these overlapping tiles can be seen in Fig 6. The white boxes represent the tiles in the first set and the red boxes represent the tiles in the second, shifted

set. The motivation behind this approach was to provide robustness to a building's location within a tile to counter prediction inaccuracy in the case of a building being close to the border of tiles.

For each tile, the original tile, left-right flip, and the up-down flip were passed through each of the three trained U-Net models. The semantic segmentation softmax outputs for each were then averaged (reorienting the flipped predictions before averaging). To clarify, a total of nine softmax predictions are averaged for each original tile. Finally, each pixel in a tile is classified using the standard argmax operation to produce a mask (building or non-building). Once the mask for each tile was generated, each corresponding set of tiles was stitched back together. Given the two different scene tilings (non-shifted, shifted), the two wide-area mask results are then unioned/OR-ed to create the final scene mask. Figure 3.9 shows an overview of the above averaging procedure.

3.7 Post-Processing the Building Mask

Due to the small size of the training data, we used certain morphological post-processing methods to improve the final mask obtained from averaging the models. The unprocessed binary mask of the detected buildings returned by the semantic segmentation model contained many small clusters of pixels that were falsely predicted to be buildings. We used an area-opening morphology function to remove all 8-connected components covering an area of 15,000 pixels or less. There were also many gaps within the mask of predicted buildings. We then dilated the remaining components using a disk-shaped structuring element of radius 18 to produce the final

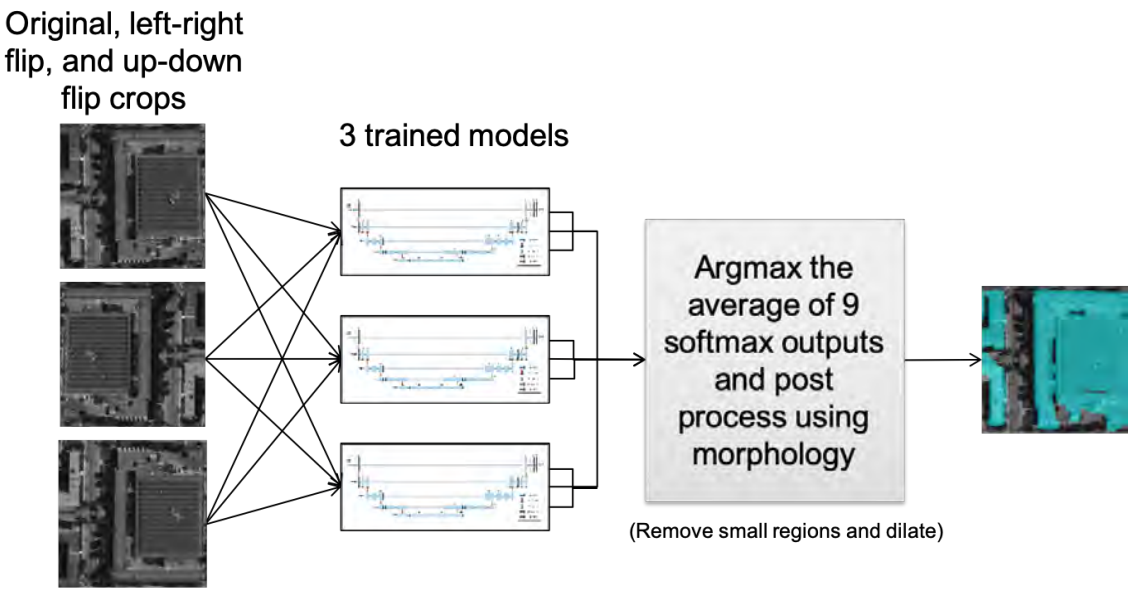
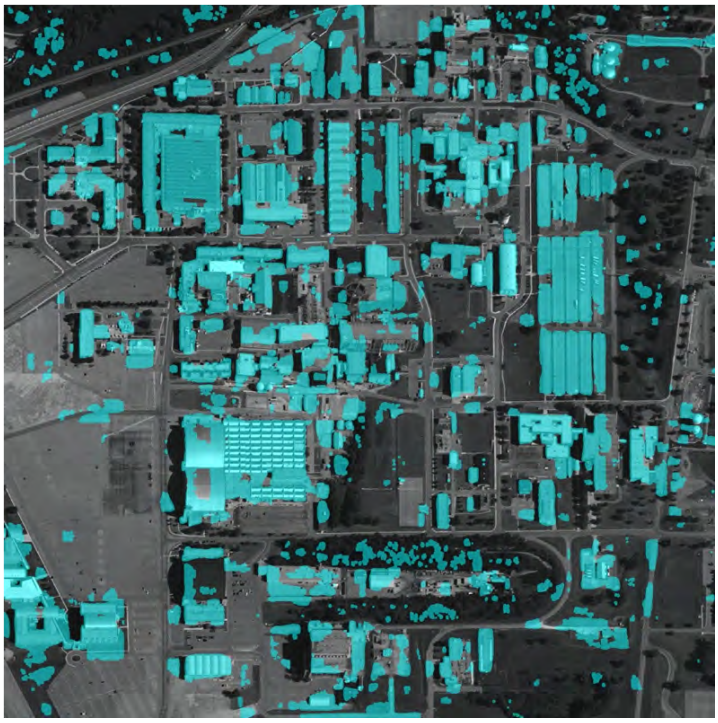
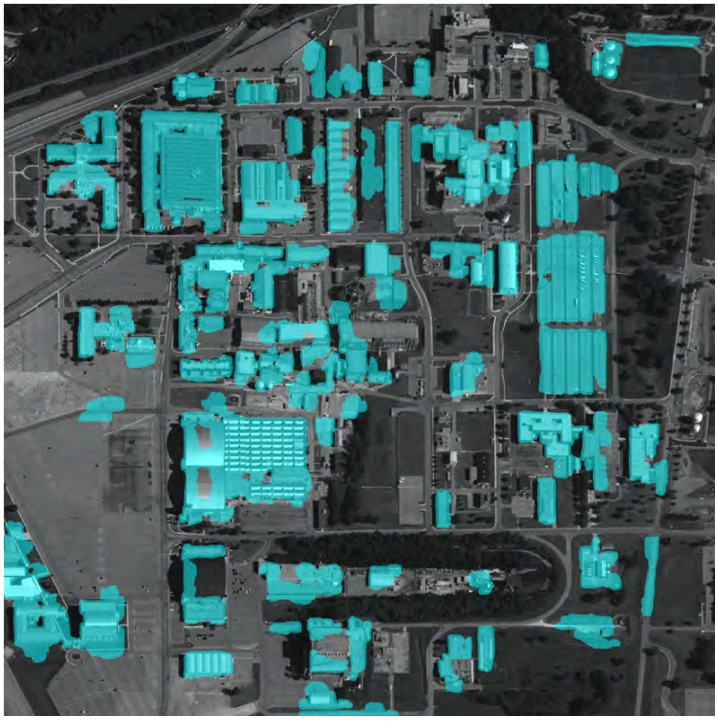


Figure 3.9: Averaging Procedure.

processed mask of the buildings. The pre-processed and the final processed building masks obtained on the UNICORN dataset are shown in Fig. 3.10.



(a) Initial Pre-processed Building Mask.

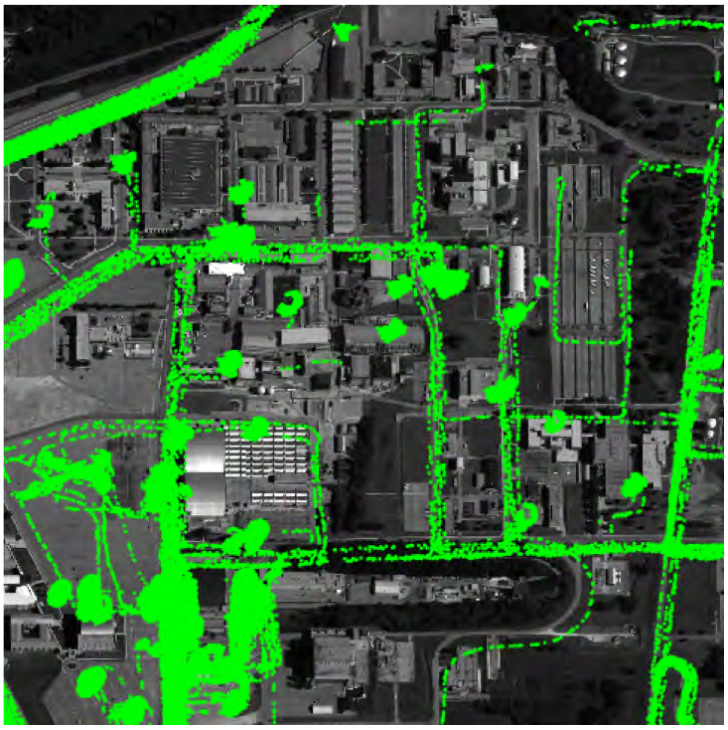


(b) Final Post-processed Building Mask.

Figure 3.10: Building Masks.

3.8 Tracking Objects in WAMI

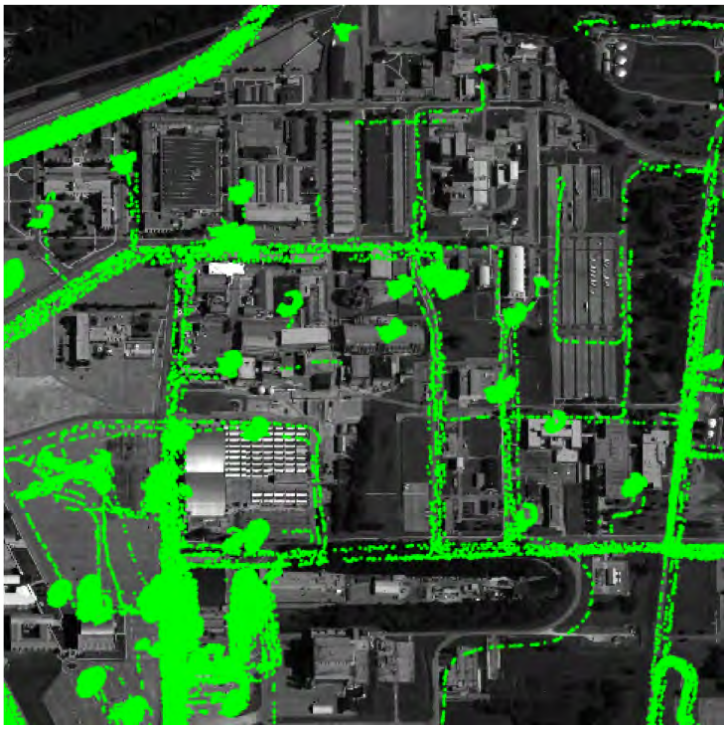
We used the Object detection and Tracking software developed by Kitware [1] in this work. We ran the tracker on the WAMI part of the UNICORN 2008 dataset. The Kitware tracker output contains latitude and longitude geo-coordinates of every predicted tracker point. We converted the latitude and longitude geo-coordinates of all the tracker output points into their corresponding pixel locations in our region of interest. The tracker output can be tuned by adjusting its configuration parameter called z-threshold. Increasing the z-threshold reduces the sensitivity of the tracker and hence with higher values of z-threshold, the tracker predicts a lesser number of points. The UNICORN 2008 dataset contains a variety of objects in the scene that were tracked and truthed (or labeled) by humans. The ground truth points for the region of interest are shown in Fig 8. We ran the Kitware tracker on the region of interest and tuned its parameters to get various tracking results. We generated three different outputs obtained by tuning the tracker output by using Z-threshold values 1.5, 2.25, and 3.0. Figures show these unmasked raw tracker outputs in comparison with the ground truth. We can see that the tracker output with a z-threshold value of 3.0 is the closest to the ground truth.



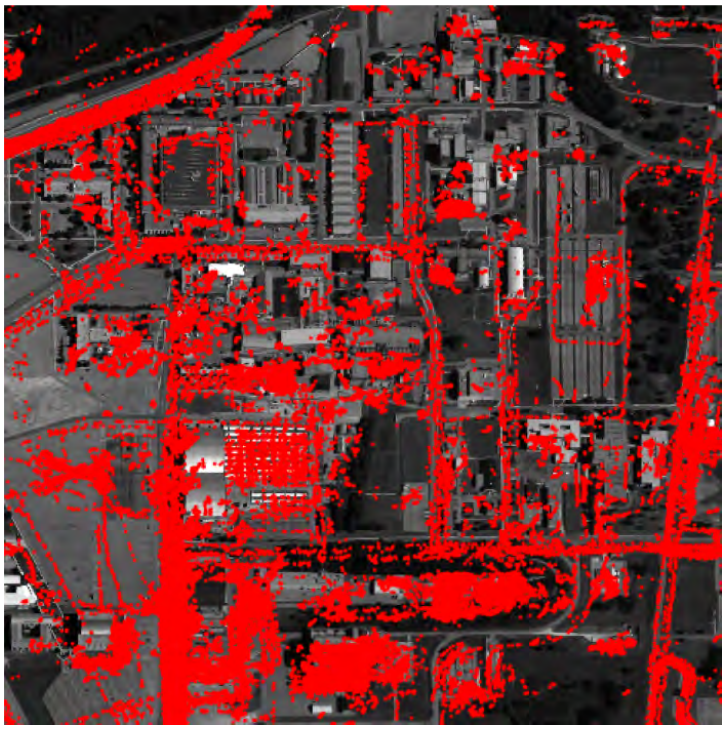
(a) Ground Truth.

(b) Tracker Output.

Figure 3.11: Z-Threshold = 1.5 .

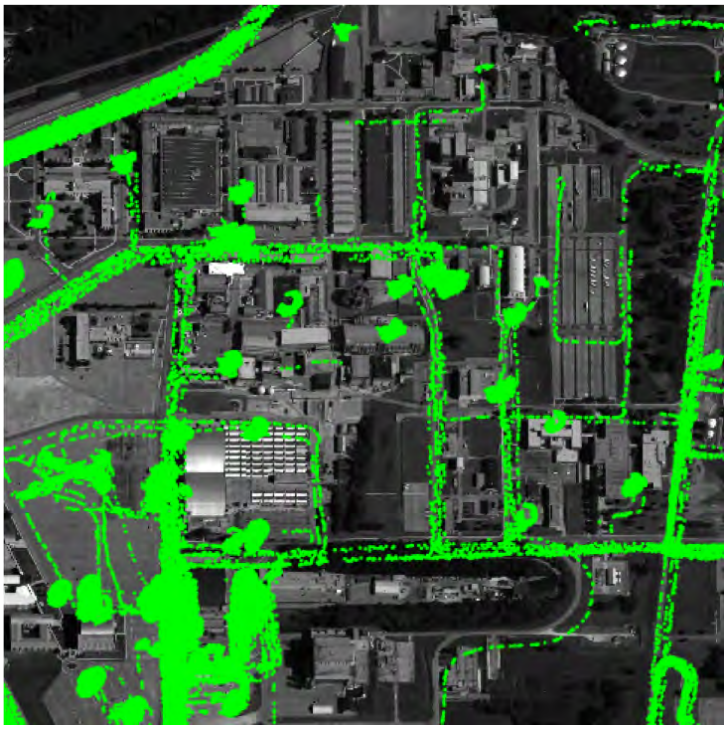


(a) Ground Truth.

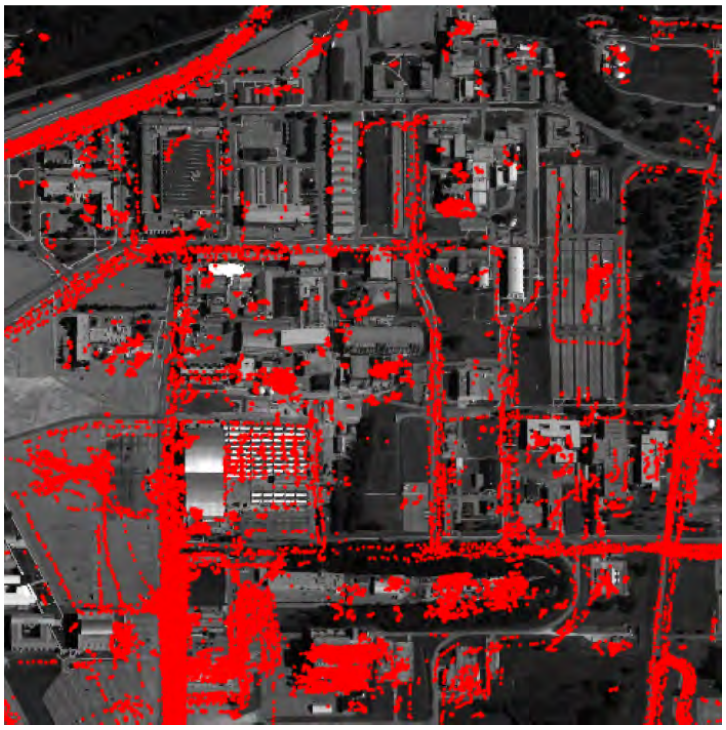


(b) Tracker Output.

Figure 3.12: Z-Threshold = 2.25 .



(a) Ground Truth.



(b) Tracker Output.

Figure 3.13: Z-Threshold = 3.0 .

3.9 Filtering Tracker Output

We combined the tracker output with the segmentation results by overlaying the building mask on the region of interest. For each tracker point in the region of interest, we find its corresponding pixel location and check the classification label of the pixel in the binary building mask. If the underlying pixel is labeled as a building, then the corresponding tracker point is filtered out as it is unlikely for a tracker that detects moving objects on the ground to predict a point on top of a building. We filtered out the tracker outputs for points predicted inside the building mask in the region of interest for the z-threshold values 1.5, 2.25, and 3.0 that we generated. Figure shows the masked tracker points and Fig. shows the final output with the filtered tracker points.

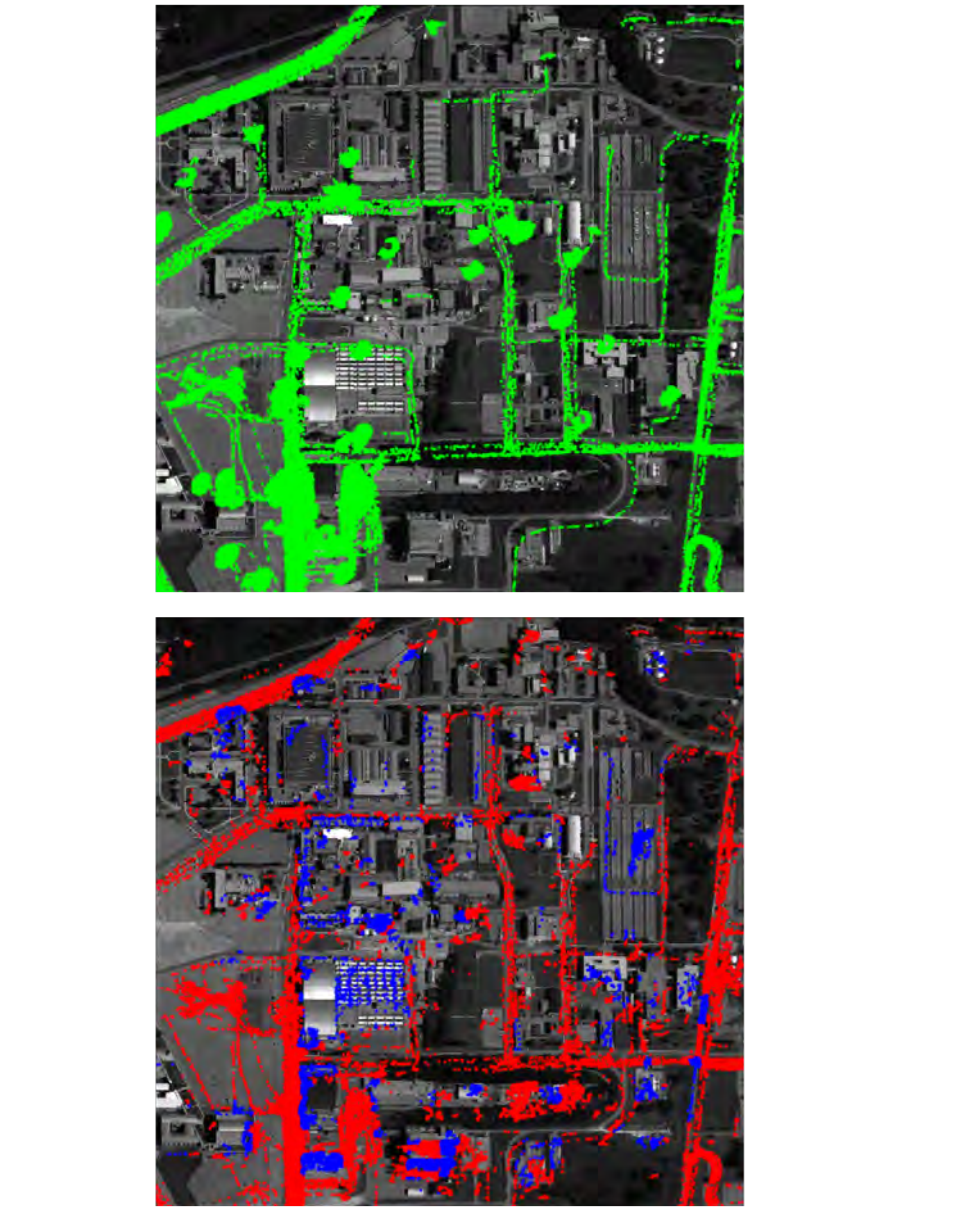


Figure 3.14: Masked Tracker Points for z-threshold = 3.0 .

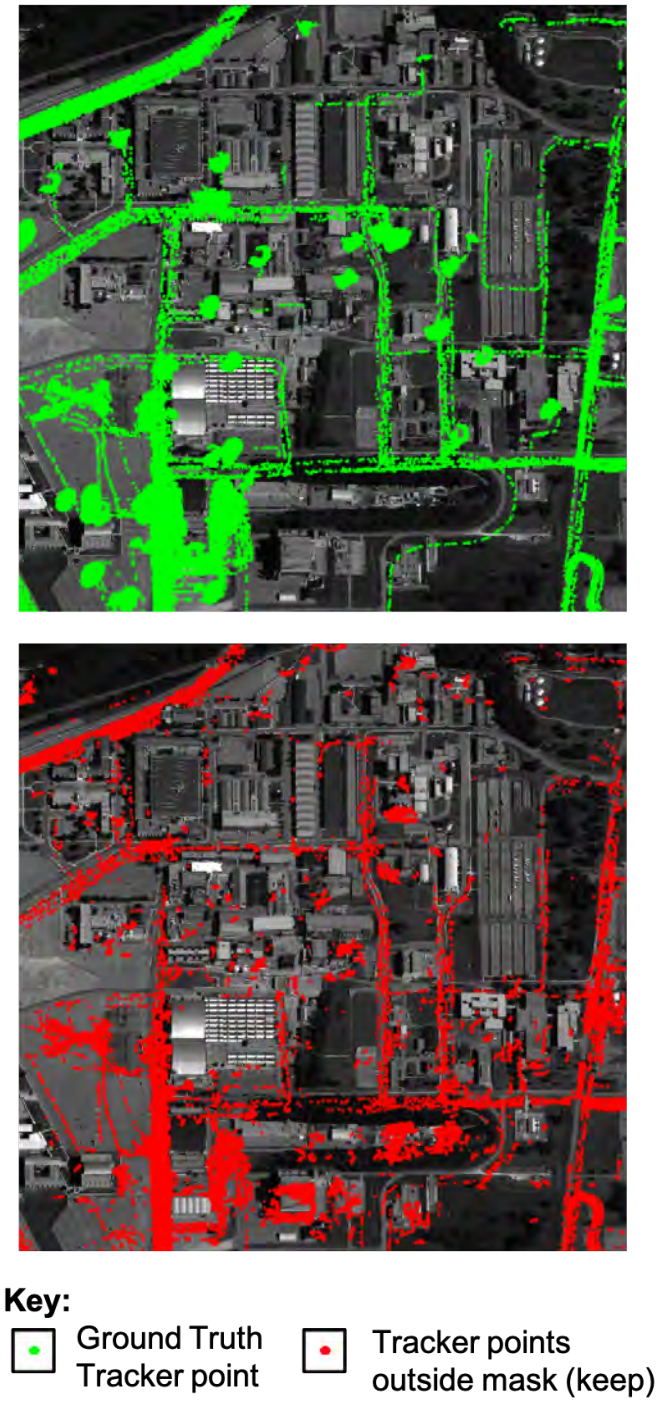


Figure 3.15: Final Filtered Tracker Points for z-threshold = 3.0 .

3.10 Performance Evaluation

In order to quantify the effects of using the predicted building mask to improve the tracker output (pre- and post-masking), we calculated the Precision, Recall, and F1 scores with various tracker outputs (from different z-threshold settings) by comparing the predicted tracker points to the provided ground truth points. We used a circular bounding region to check how often a predicted tracker point has a ground truth point within its bounding region (Precision) and how often a ground truth point has a predicted tracker point within its bounding region (Recall). The final F1 score is the harmonic mean of Precision and Recall. The bounding region for a point is a circle drawn around that point. We evaluated bounding regions of three different radii: 0.00007, 0.00008 and 0.00009 (unit: longitude decimal degree). Figure 3.16 shows the size of a bounding region of radius 0.00007 .

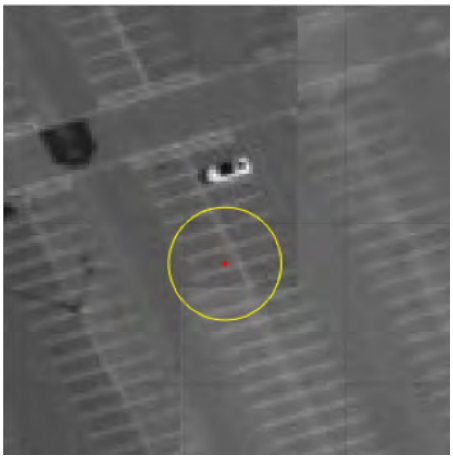


Figure 3.16: Bounding Region of Radius 0.00007 .

We plotted the predicted tracker points and the ground truth points and calculated the scores for all three raw tracker outputs against ground truth. We then used the predicted building mask generated by the semantic segmentation model to filter out potential false alarms from the tracker outputs. The plots and the scores were recalculated for the masked tracker outputs and listed in Table 3.1.

Table 3.1: Umnasked vs Masked Tracker Output Scores.

Z-Threshold	Precision (Unmasked vs Masked)	Recall (Unmasked vs Masked)	F1 (Unmasked vs Masked)
High Quality Tracks (z-threshold = 3.0)	45.8/**56.8** %	83.3/73.9 %	59.1/**64.2** %
Medium Quality Tracks (z-threshold = 2.25)	33.3/42.0 %	89.0/79.3 %	48.5/55.0 %
Low Quality Tracks (z-threshold = 1.5)	25.4/30.3 %	**93.8**/83.7%	40.0/44.5 %

From the results in Table 1 we can see a significant increase in Precision and F1 scores for the masked tracker outputs in comparison to the unmasked tracker outputs. A noticeable drop in Recall is seen in the scores of the masked tracker outputs. This is explained by the fact that masking out noisy track points removed nearby possibilities for a match to ground truth thus lowering Recall. Also, it is observed that the tracker output with Z-score 3.0 has the best F1 score, followed by Z-scores of 2.25 and 1.5.

3.11 Summary

The main outcomes from this task were assembling a manually labeled dataset containing WAMI imagery for the idea of training a U-Net based semantic segmentation model to filter out incorrect tracker points predicted on the roof of buildings. We trained the model on imagery collected by a similar sensor and transferred the trained model to predict a building mask on our region of interest in the UNICORN dataset. We evaluated our approach for improving tracking performance and achieved notable enhancements in tracking scores. But the size of the dataset was limited to a small number due to the overhead of having to manually annotate the imagery. The availability of large labeled datasets would have considerably helped to improve the performance of our semantic segmentation model. We propose a solution to this problem in the following chapter.

Chapter 4: Automated Collection and Classification of Objects in Satellite Imagery

In this chapter, we present our approach of developing an automated data collection pipeline that can collect and label objects in satellite imagery to facilitate building task-specific deep learning models. We examine the different components constituting the modular framework that we developed for collecting satellite imagery of a specified object class using public geographical databases and crowd-sourced labels. Figure 4.1 shows the overall architecture of our pipeline.

4.1 Satellite Imagery

In this work, we employ the European Space Agency's Sentinel-2 (A and B) [6] constellation of twin satellites. It is a wide-swath, high-resolution, multi-spectral imaging mission supporting Copernicus Land Monitoring studies (monitoring vegetation, soil, and water cover, as well as observation of inland waterways and coastal areas). The Sentinel-2 Multispectral Instrument samples thirteen spectral bands in the visible, near-infrared, and short-wave infrared part of the spectrum with a spatial resolution of 10 meters (m) per pixel for the four optical and near-infrared (NIR) bands, 20 m for the six red edge and shortwave infrared (SWIR) bands, and 60 m

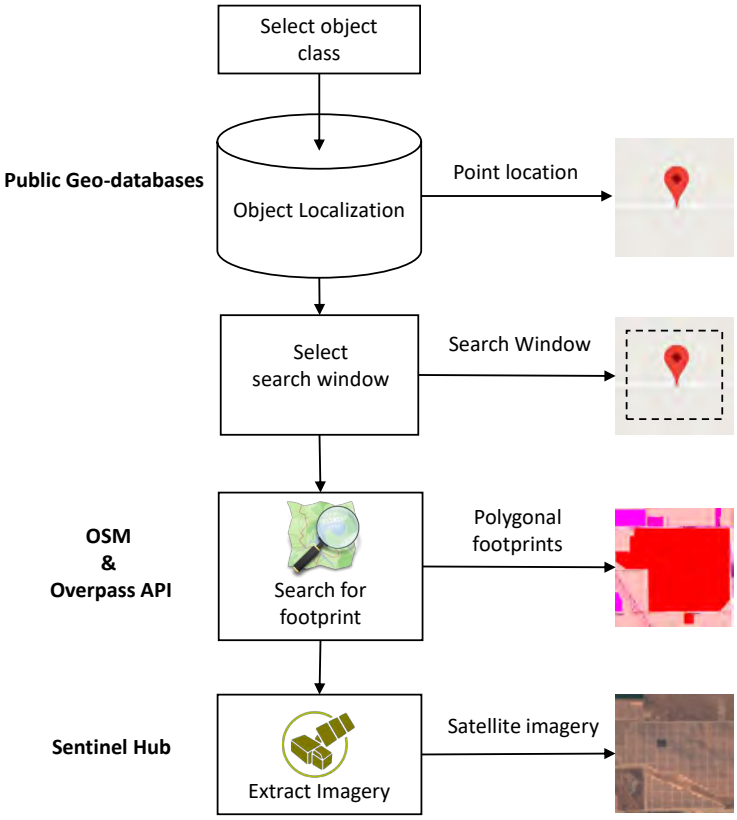


Figure 4.1: Data generation pipeline architecture.

for the three atmospheric correction bands. Sentinel-2 has a high revisit time (frequency of new image collection in a particular location) of 10 days at the equator with one satellite, 5 days with 2 satellites under cloud-free conditions, and 2-3 days at mid-latitudes. Figure 4.3 shows a comparison of Sentinel-2's spatial and temporal resolution with other popular EO satellite systems. This helps support the monitoring of Earth's surface changes on a weekly basis. We picked Sentinel-2 as it provided the highest spatial resolution among the satellite systems that allows free and open access to its data.

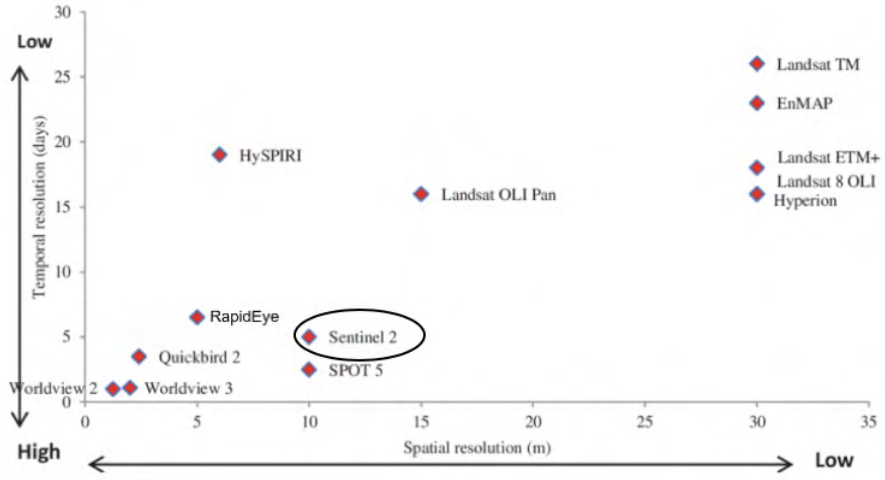


Figure 4.2: Spatial vs Temporal Resolution of Various Satellites.

Sentinel Hub's [23] Open Geospatial Consortium (OGC) API can be used to provide services for data access, display, and processing within hours of image acquisition. The `sentinelhub` Python package facilitates OGC web requests to download and process satellite imagery in batch scripts thus circumventing the need to manually download Sentinel-2 data from the web. Sentinel Hub's Web Map Service (WMS) is used to request downloads of certain Sentinel bands using different settings such as maximum cloud coverage, time range of imagery acquisition, image format, size, etc. The region of interest to be downloaded is represented as a bounding box in geo-coordinates. This enables the extraction of satellite imagery for any particular region of interest at any required time period, thus avoiding time mismatches between imagery and any ground truth data. Figure 4.3 shows some examples of Sentinel-2 RGB imagery acquired at four different times highlighting the variations in the image across different climatic conditions (notice the snow cover in Fig. 4.3(c)).

(a) July 2018 (b) October 2018 (c) January 2019 (d) April 2019

Figure 4.3: Sentinel-2 imagery of the same area acquired at four different times.

4.2 Object Localization

An object in satellite imagery can be a building, forest, industrial area, etc. selected to train a classifier. The geo-coordinates of many types of objects can be found using publicly available resources, such as a catalog of street addresses or a database containing geo-coordinates. Given a list of addresses, open-source Geocoders such as Nominatim [24] can be used to determine their geo-coordinates. Python's `geopy` package supports several popular geocoding web services. Also, some public geo-databases exist that already provide a geo-coordinate for a category of interest (e.g., power plant database [25]). However, such data sources may have only an approximate (inaccurate) location for the object. For example, Fig. 4.4 shows an example entry from the Global Power Plant Database for the Topaz Solar Farm and its specified location is outside the boundary of the solar power plant. We therefore will need to search for the object near the given geo-coordinate. For this, we use a global spatial database which contains object footprints to identify bounding regions.

Country	Name	Capacity MW	Latitude	Longitude	Fuel	Generation GWHD	Ownership
USA	Topaz Solar Farm	585.9	35.4056	120.0686	Solar	1265.76	Topaz Solar Farms LLC

Figure 4.4: Inaccuracy in Object Location obtained from Geo-Databases.

4.3 OpenStreetMap

OpenStreetMap (OSM) [5] is a collaborative project used to create a crowd-sourced spatial database of the world. It was started in 2004 with the simple idea of multiple contributors with local knowledge collaborating to create a detailed labeled map. We employ OSM as our auxiliary data source to extract polygonal footprints for the given geo-location of objects obtained from public databases. Figure 4.5 shows an example OSM footprint of a solar power plant.

The OSM data model consists of three basic data structures: *Nodes*, *Ways*, and *Relations* as shown in Fig. 4.6. A *Node* represents a geographic point expressed in latitude and longitude. A *Way* constitutes at least two *Nodes* (polyline or polygon). A *Relation* is a logical collection of *Ways* and *Nodes* (multipolygon). The physical features on the Earth's surface are described by OSM using tags with key-value pairs attached to its basic data structures (*Nodes*, *Ways*, and *Relations*). The key is used to describe a topic, category, or feature type (e.g., building), and the value details

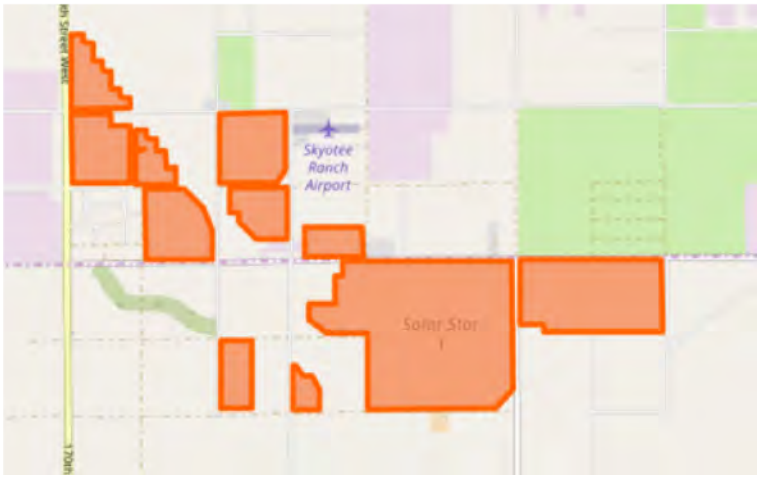


Figure 4.5: OSM Footprint of a Solar Power Plant.

the specific form of the key (e.g., residential). The OSM Map Features pages on the OSM wiki [26] lists the tags agreed upon by the OSM community. Figure 4.7 shows an example OSM urban area with land use labels.

There are multiple methods for downloading data from OSM. It is possible to get the data in the form of XML formatted .osm files. OSM also provides its entire database as a Planet.osm file which is updated weekly to reflect new changes to its database. But often we do not need to work with the entire database when there is a specific region of interest. In such cases, the Overpass API [27] can be used, which is a read-only, web-based service that accepts queries to download custom filtered datasets. Python's `overpy` package provides Python bindings to the Overpass API.

We use the object's initial geo-coordinate to restrict our search space in OSM to avoid the overhead of having to search throughout the entire planet to localize our object footprints. We next define a search space around the object's geo-coordinate. We build a query in the Overpass Query Language which contains the search criteria

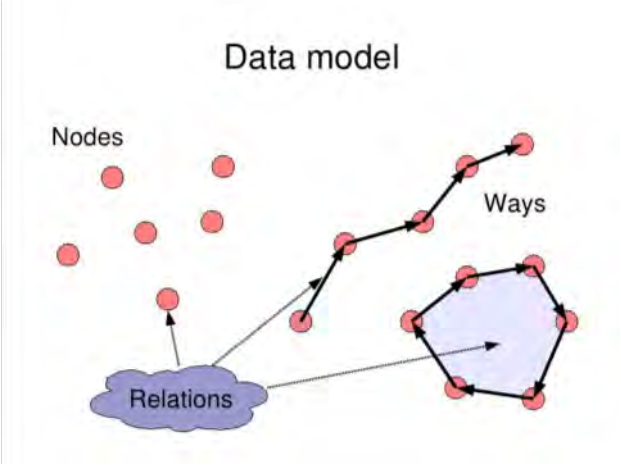


Figure 4.6: OSM Data Model.

and a bounding box defining the search window. We use a pre-determined size for the search window based on the average size of our objects. We then search within the window using the Overpass API for the polygonal footprint of the object class.

We search for both *Relations* and *Ways*, as some objects might be represented as *Relations* (multipolygon) which are made up of multiple *Ways* (polygons). In such cases, we extract all *Ways* which are a member of a *Relation*. The *Ways* are made up of an ordered list of *Nodes* which form the vertices of the object's polygonal footprint. Figure 4.8 shows an example Overpass query searching for Ways and Relations of solar power plants within a specified search window and a visualization of the returned result.

A bounding box is fit to the entire footprint using the min and max latitude and longitude extents. Sentinel Hub is then used to download the satellite imagery corresponding to the footprint bounding box generated for the object.

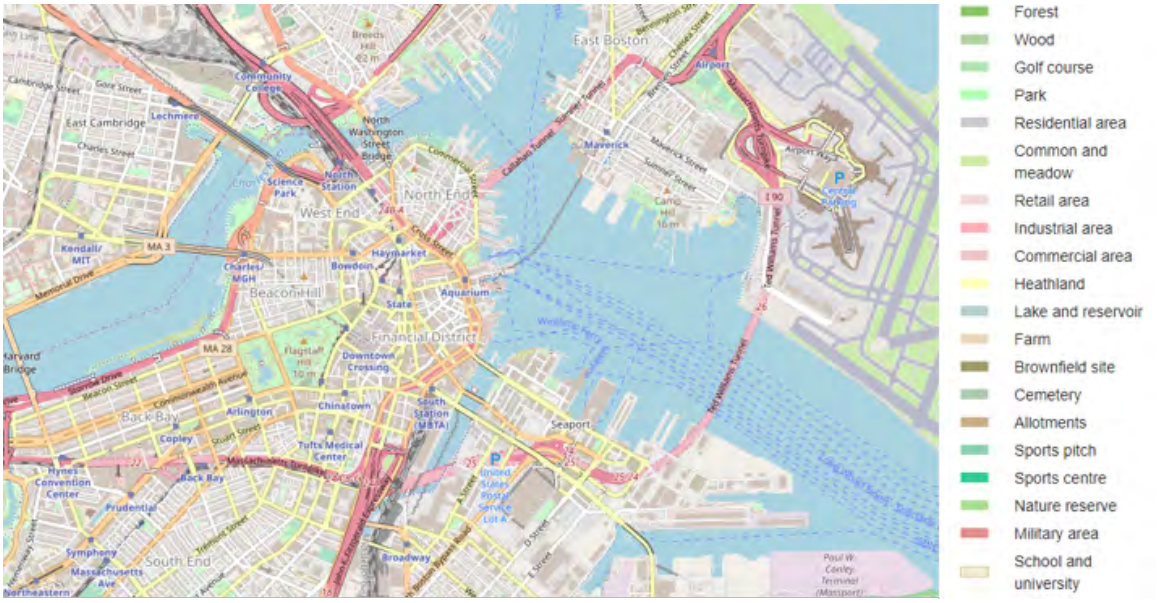


Figure 4.7: OSM region with land use labels.

Using the various packages related to Sentinel and OSM, a batch script can be used to extract imagery of a selected object type from the associated list of geo-coordinate locations. We can use this same approach to generate negative (non-object) examples by querying OSM using the Overpass API to ensure the absence of an object.

4.4 Example and Experiments

We demonstrate the proposed framework by procedurally generating a training and testing dataset with positive and negative examples of solar power plants. To show the applicability of the extracted dataset, we train an image classifier using a Convolutional Neural Network (CNN) to detect the selected object class. The main focus of this work is the automatic generation of annotated imagery (other object classes and classification models could be used).

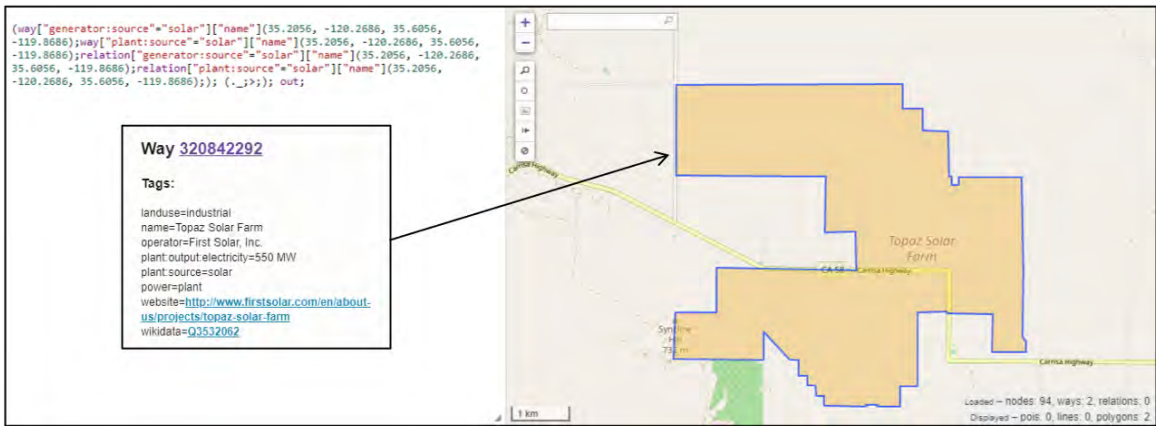


Figure 4.8: Example Overpass API Query for a Solar Power Plant.

4.4.1 Solar Power Plants

We leverage the World Resources Institute's Global Power Plant Database (GPPD) [25], which is a comprehensive, open-source database of power plants around the world. Each power plant listed has information on its geo-location, capacity, generation, ownership, and fuel type. It is continuously updated as new information becomes available. The database version used in this work includes over 28K power plants. However as previously mentioned, some of the geo-locations are only approximate and may not even exist on the footprint of the power plant.

We center our search region in OSM on the provided geo-coordinate of a solar power plant from the database. We define our search window as shown in Fig. 4.9 by ± 0.2 decimal degrees in latitude and longitude around the geo-coordinate (approximately a 16×16 km area centered around that geo-coordinate). We then search through OSM using the Overpass API in the search window for *Relations* and *Ways* with the tags **'generator:source = solar'** or **'plant:source = solar'** (standard

Figure 4.9: Search Window in OSM.

tags for solar power plants in OSM, note that other key-value tags for different objects can be used). We extract all *Ways* which are a member of *Relations*. The *Ways* give us the desired polygonal footprints of the solar power plants. Figure 4.10(a) shows multiple polygons that make up an OSM relation representing a solar power plant.

4.4.2 Image Collection

The next step is to extract the satellite imagery for the located object footprints. We selected to evaluate the RGB and NIR bands as they have the highest spatial resolution (10 m) among the bands in the Sentinel-2 imagery. Also, the solar power plants were visually more distinct in the RGB and NIR bands to the human eye (NIR is closer to the visible range than thermal). Other bands may be more applicable to different object classes. Figure 4.10(b)&(c) show the corresponding Sentinel-2 RGB

(a) OSM. (b) RGB. (c) NIR.

Figure 4.10: OSM polygonal footprint and its corresponding Sentinel-2 RGB and NIR band imagery.

and near-infrared band imagery for the OSM polygonal footprint of a solar power plant (see Fig. 4.10(a)). Table 4.1 lists the central wavelength, bandwidth, and spatial resolution for each of the Sentinel-2 bands used.

Table 4.1: Spectral band properties for Sentinel-2 (A and B).

Sentinel-2 Bands	**Sentinel-2A**		**Sentinel-2B**		**Spatial resolution (m)**
	Central wavelength (nm)	**Bandwidth (nm)**	**Central wavelength (nm)**	**Bandwidth (nm)**	
B02: Blue	492.4	66	492.1	66	10
B03: Green	559.8	36	559.0	36	10
B04: Red	664.6	31	664.9	31	10
B08: NIR	832.8	106	832.9	106	10

4.4.3 Positive Examples

To extract multiple positive examples of each solar power plant, we randomly sampled 5 points within each polygonal footprint and extracted a 256x256 sized image

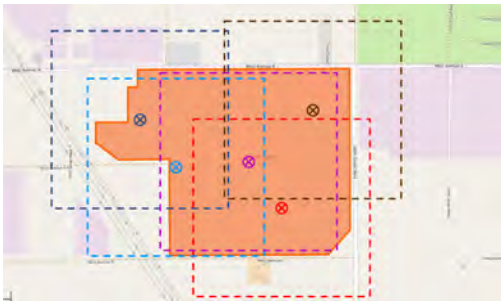


Figure 4.11: Randomly sampled points and their cropping windows.

chip centered around each of these points (see Fig. 4.11). To download the Sentinel-2 imagery, we used Sentinel Hub's WMS request. We downloaded imagery for July 2018, October 2018, January 2019, and April 2019 to account for variations in the appearance across different climatic conditions (as seen in Fig. 4.3). We additionally set the maximum cloud cover percentage to 30% to filter out imagery where the solar power plant may be occluded by clouds. Since the cloud coverage is estimated on larger Sentinel-2 tiles (100 km x 100 km areas), and not just for the region defined by a footprint bounding box, heavily clouded imagery might still be present in the smaller image chips. Also, a part of the image might contain white (empty) regions if that particular data acquisition only partially intersected the specified footprint's bounding box. We manually removed such corrupt images from our collection.

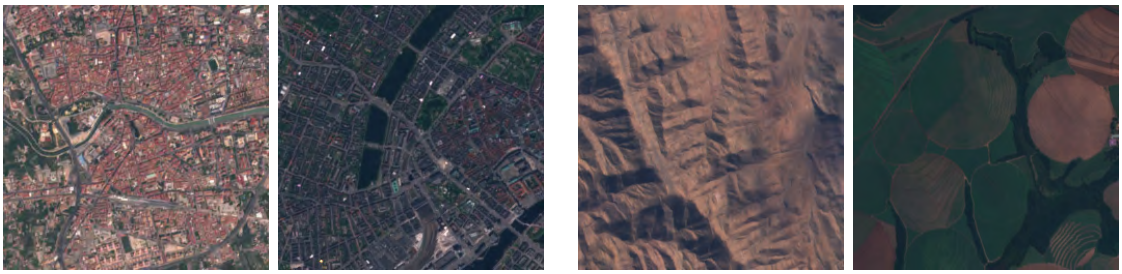
We generated approximately 20K positive samples from 400 solar power plants as our training data and another 500 positive samples from 100 different solar power plants for testing purposes. We created a validation set by randomly sampling 2K images from the training set. Figure 4.12 shows a few positive samples of solar power plants generated by this method.

Figure 4.12: Positive examples of solar power plants.

4.4.4 Negative Examples

The negative samples consist of land regions with no solar power plants, but contain a diverse collection of urban as well as randomly sampled land areas. We sampled from two separate world cities databases of urban areas across the world. We used the freely available World Cities Database [28] (provides an accurate and up-to-date database of the world's cities and towns) for training data and Nordpil's World Database of Large Urban Areas [29] which contains a different set of geo-coordinates of urban areas for generating test data. Both databases provide a single geo-coordinate for geo-referencing cities and towns. The remaining negative samples are located by randomly generating a geo-coordinate on the Earth's surface and verifying with OSM that the randomly generated point is over land (not over water).

We extracted 256x256 sized image chips centered on the geo-coordinates of the selected urban and random land areas. We again use the Overpass API to query the OSM database to ensure that the bounding boxes contain no solar power plants. Several heavily clouded image chips remained in our negative samples. We collected approximately 20K negative samples for training and another 500 negative samples



(a) Urban areas. (b) Random land areas.

Figure 4.13: Negative examples.

(250 urban areas, 250 randomly sampled land areas) for testing. As before, we randomly sampled 2K images from the negative training set with an equal proportion of images from urban areas and randomly sampled land areas for the validation set. Some negative examples from urban and random land areas are shown in Fig. 4.13.

4.4.5 Image Classification Model Architecture

We employed a standard CNN architecture to train our image classification model. The network employs three downsampling blocks, each composed of a convolutional layer (Conv) with 32 filters of size 3x3 and ReLU activation, followed by a 2x2 max pooling layer (MaxPool). The output from the final downsampling block is flattened and fed to 2 fully-connected layers (FC) having 512 nodes each with ReLU activation. The output layer consists of a single node with a sigmoid activation function. Separate models were trained for NIR (1 channel), RGB (3 channels), and RGBNIR (4 channels). The input to the network is the image chip of size 256x256 and the number of input channels is equal to the number of bands being used to train the model. The

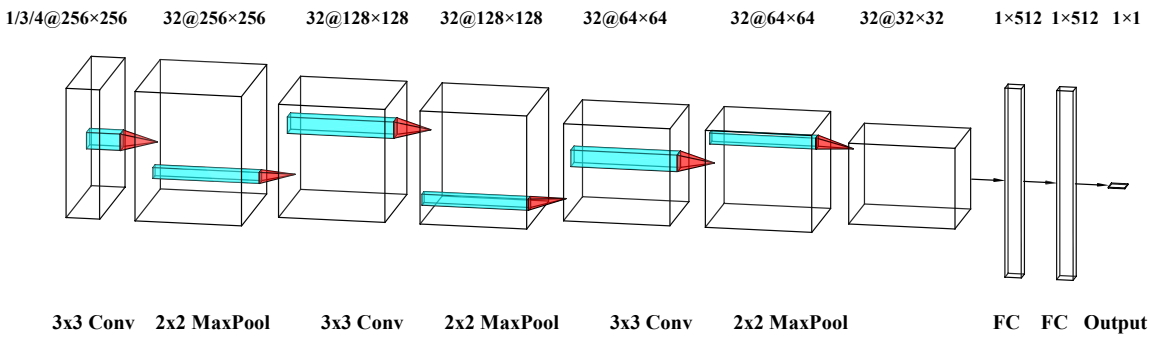


Figure 4.14: CNN Model architecture.

model architecture is shown in Fig. 4.14 and takes an input of size 256x256 with either 1, 3, or 4 channels.

Each image was preprocessed by normalizing the pixel values to the range [-1,1]. The model was trained using stochastic gradient descent with a batch size of 32 images and momentum of 0.9. We initialized the weights in each layer using Xavier initialization and the biases were initialized with values drawn from a normal distribution with zero mean and unit standard deviation. The learning rate was initialized to 0.001 and adjusted using "Poly" learning rate adaptation [30, 31]. We trained each model for 30 epochs and selected the model from the epoch which had the highest validation accuracy. In addition, we also created a model that gives predictions obtained by averaging the sigmoid outputs of the individual RGB and NIR models (RGB+NIR).

4.5 Results

The results of the various models on the solar power plants are summarized in Table 4.2. The best performance was given by the model trained on all the four

Table 4.2: Comparison of results from models trained on different bands.

Bands	Accuracy	Precision	Recall	F1 score	Parameters
NIR	0.92	0.98	0.86	0.91	17,059,905
RGB	0.96	0.98	0.93	0.95	17,060,481
RGB+NIR	0.96	0.98	0.94	0.96	34,120,386
RGBNIR	**0.97**	**0.99**	**0.94**	**0.97**	17,060,769

bands together (RGBNIR), with an accuracy of 0.97 and an F1 score of 0.97. As expected, the joint RGBNIR model with early fusion (at the input) incorporating inter-dependency was better than the individual models. It was also slightly better than the independent late fusion approach (RGB+NIR). The RGBNIR model had an increase of 288 parameters over the RGB model and 864 parameters over the NIR model, but it had nearly half of the parameters of the dual late-fusion approach (RGB+NIR). This shows that the model's performance improves as we increase the number of bands used for training, advocating the use of potentially even more bands provided by Sentinel.

4.6 Scanning Array

We also implemented a scanning array approach of testing our trained models to detect the presence of objects in large areas. This approach downloads satellite imagery of any specified region of interest into 256x256 image tiles. We then feed the tiles sequentially through the trained model and the model detects the presence of solar power plants in every tile. Figure 4.15 shows a visualization of this approach for the Disney World Resort in Orlando, Florida. Disney World uses solar power plants to power their amusement parks. The yellow tiles indicate that the model detected

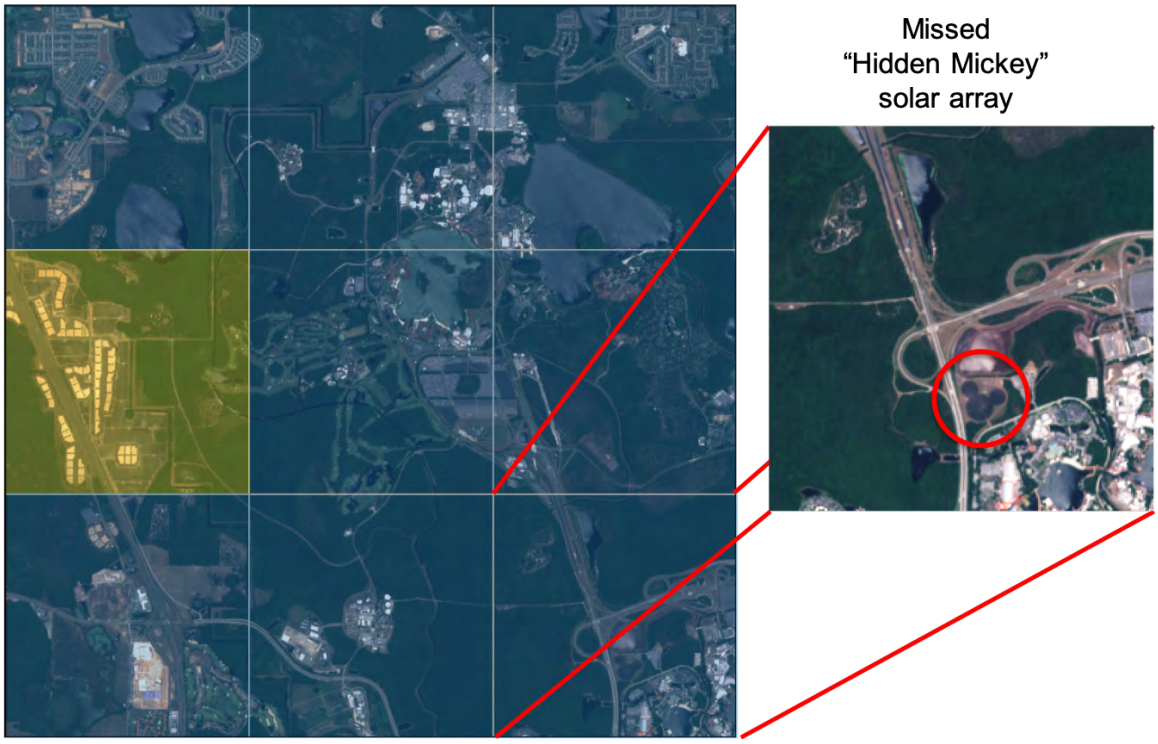
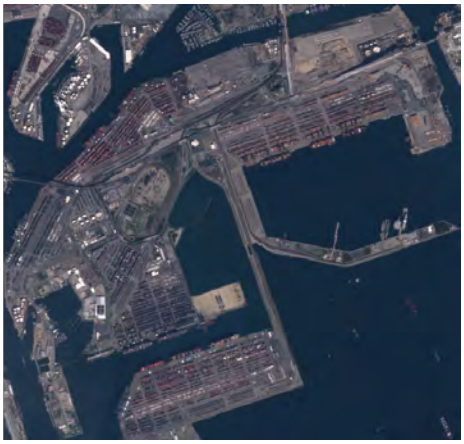


Figure 4.15: Scanning Array Approach on Disney World Resort in Orlando, Florida.

a solar power plant and the blue tiles indicate absence of solar power plants. Our best trained model was able to detect one of these solar power plants but it missed a smaller Mickey mouse shaped power plant in the region. This may be due to the very small size of the panels when compared to the huge power plants on which the model was trained. The proximity to urban regions might also be a reason why the model was not able to detect this power plant as most large solar power plants are located away from urban regions. This scanning approach can be used to scan large areas in satellite imagery for any required object class of interest when provided with a trained model for detecting that particular object class.

(a) Airports.

(b) Reservoirs.



(c) Ports.

Figure 4.16: Example imagery collected for alternate object classes.

4.7 Alternate Object Classes

The above framework can be used to collect satellite imagery of other object classes by swapping the public data source used to extract the point location of the required object class and then using their corresponding OSM tags to search for their polygonal footprints. Hence this framework can be adapted to work with any object class whose geo-locations are mapped by such data sources and they have a corresponding OSM tag. Figure 4.16 shows imagery generated using the framework for airports, reservoirs, and ports using public databases which list out their geo-locations and then using the extents of their OSM footprints to download satellite imagery. We can also work with object classes that don't have public data sources mapping their locations but have an OSM tag associated with it. In this case, we have to work with just OSM to locate objects by filtering through the entire OSM Planet database to extract all instances of the object class of interest.

4.8 Summary

We put forward a solution to counter the limited number of labeled remote sensing datasets available by building an automated data collection pipeline for accumulating task-specific datasets. We make use of public databases and OpenStreetMap information to locate all instances of an object class of interest and download satellite imagery of these locations. We validate our method by creating a dataset of solar power plant imagery and build an image classifier trained on the collected dataset that delivered significant results in classifying solar power plants.

Chapter 5: Contributions and Future Work

In this thesis, we have introduced novel methods of applying AI learning techniques to remote sensing data for increasing the accuracy of tracker outputs in manually annotated WAMI data. We also developed an automated data collection pipeline for gathering labeled satellite imagery of objects using location information obtained from geo-databases such as OpenStreetMap. We began with an introduction to our approaches in Chapter 1. We briefly reviewed the related work in Chapter 2. Next in Chapter 3, we defined our approach of fusing the outputs of tracking and image understanding algorithms for improving tracking efficiency in WAMI data by utilizing contextual information obtained from manually labeled data. We followed this with our solution for automatically gathering labeled satellite imagery of objects to aid deep learning models in Chapter 4. We demonstrated our approach by automatically accumulating a global dataset of solar power plant satellite imagery and training an image classifier on the collected data.

We have shown that the accuracy of tracking software in wide aerial surveillance data can be enhanced by filtering out false positive tracker points predicted on top of buildings by using a mask of buildings obtained from a semantic segmentation model. We discovered that the performance of such deep learning models is highly dependent on the availability of large labeled datasets for training. But in this thesis,

we had to manually label the WAMI data for training our model which restricted us to work with a small dataset. Manually generating and annotating imagery is a time-consuming task and a commonly faced problem. This task becomes more complicated with the use of supervised deep learning models that require large volumes of labeled training data. In this thesis, we presented a general framework employing existing geographical databases and satellite imagery to automatically generate labeled imagery of a selected object class for training image classification models. We demonstrated this method by selecting solar power plants as our object class, though other classes could also be used with the framework. We achieved compelling results with multiple image bands that validate the proposed pipeline. Overall, our work shows that it is possible for creating large datasets of satellite imagery to help train task-specific deep learning models.

We will reemphasize the contributions of this thesis and then present potential future work in this area.

5.1 Contributions

We have explained several methods of using AI techniques for remote sensing applications such as object tracking and satellite image classification. Our contributions in this thesis can be summarized as follows.

- We presented a way of employing semantic segmentation models to extract contextual information in WAMI data for enhancing tracking performance by filtering out inaccurate tracker points detected on buildings. We used a building mask generated by a CNN model trained on manually labeled WAMI data for removing these incorrect tracker points.

- We developed a modular framework for automatically locating objects mapped by spatial databases such as OpenStreetMap and download satellite imagery of the localized objects. We validated our framework by automatically collecting a dataset of satellite imagery of solar power plants around the world.

- We introduced a novel means of combining an object's coordinates obtained from geo-databases and its boundary information extracted from geo-spatial databases like OpenStreetMap for efficiently locating the extents of an object. We demonstrated this approach by using a combination of the Global Power Plant Database and OpenStreetMap to locate solar power plants thus avoiding the brute force approach of having to search through the whole world to locate an object.

- We programmed a script for downloading satellite imagery from the Sentinel - 2 data hub of any object by using geo-coordinates obtained from OpenStreetMap. We used this script to download and gather a dataset containing satellite imagery of solar power plants worldwide.

- We also proposed a way of using OpenStreetMap for downloading satellite imagery of negative examples (Images not containing a particular object). We used this method to collect a negative example dataset of satellite imagery not containing solar power plants to complement the solar power plant imagery for training a binary classifier.

Overall, in this thesis, we developed a solution for improving object tracking performance in WAMI data by building a semantic segmentation model to filter out false positive tracker points on buildings. We showed the limitations of manual annotation

of remote sensing data and presented a novel technique for automated labeling of remote sensing data to solve this problem.

5.2 Future Work

Future work related to this thesis would explore improving the semantic segmentation models to obtain additional contextual information to further improve tracking results and extending the data collection framework for other tasks such as semantic segmentation.

5.2.1 Semantic Segmentation Improvements

While our semantic segmentation model was able to mask false tracker points resulting in a significant enhancement in tracker accuracy, there are still some areas for improvement that could be addressed.

- The most notable weakness of our approach was the small dataset that we used to train the semantic segmentation model. We can train a better semantic segmentation model using a bigger labeled dataset. It would be possible to use our proposed extended data collection framework to automatically label ground truth for semantic segmentation tasks leading to the generation of bigger datasets to assist in training deeper models.

- We also just used a binary mask of buildings vs non-buildings in this thesis. But extending the semantic segmentation model to learn other classes of objects present in the imagery such as transportation surfaces (roads, bridges, parking lots, waterways, etc.), grass, trees, etc. will enable augmenting the contextual information needed for further improvements in tracking accuracy. But this

would again require additional annotation of the training data for new object classes.

- We also worked with grayscale datasets in this thesis which makes the task of segmenting objects very difficult. Availability of RGB WAMI datasets will improve the performance of semantic segmentation models that can learn better information from high-dimensional data.

We presume that implementing the above ideas by integrating WAMI data with our data collection framework will further enhance tracking performance in wide area surveillance data. This will open the doors for automated surveillance techniques in aerial imagery which have potential applications for military and intelligence.

5.2.2 Data Collection Framework Improvements

We view our work in developing a framework for automated data collection as a good stepping stone for future extensions. The modular structure of our framework enables making enhancements to certain modules easier without affecting the whole pipeline. Here we list some possible avenues for exploration in the future.

- Our current framework utilizes the boundary information of objects obtained from footprints in OpenStreetMap just to ensure that an image contains a portion of the object within its extents. We can use these object footprints to create bounding boxes of the objects for the purpose of training object detection tasks.

- We can also use the exact shape of the OpenStreetMap's object footprints which are represented by vector graphic polygons (using geo-coordinates as vertices

of the polygon) to label ground truth information for semantic segmentation tasks provided the imagery data is geo-registered. But given the crowdsourced nature of OpenStreeetMap we must account for the possibilities of errors when overlaying OpenStreetMap's object footprints on image data.

- We can also swap out the satellite imagery source with any geo-registered image database such as aerial imagery, WAMI data, etc. opening avenues for applications previously constrained by the lack of available labeled data.

The framework developed in this thesis can help open avenues for new applications with remote sensing data. We view that extending our pipeline to accommodate the improvements proposed above is an essential next step towards promoting the increased use of AI techniques for the analysis of remote sensing data.

5.3 Conclusion

Manually generating and annotating imagery is a time-consuming task and a commonly faced problem. This task becomes more complicated with the use of supervised deep learning models which require large volumes of labeled training data as we had observed in the first part of this thesis. In the second part of this thesis, we presented a general framework employing existing geographical databases and satellite imagery to automatically generate labeled imagery of a selected object class. We expect the use of deep learning on remote sensing data to increase in the future and our framework for automated annotation perfectly complements this trend by opening avenues for applications previously constrained by lack of training data.

Appendix A: Data Collection Framework Code Components

The data collection framework presented in this thesis was written in Python 3.7. In this appendix, we list the packages used for developing the framework and provide code snippets for each component of the framework.

A.1 Packages Used

We make use of the following Python packages to work with OpenStreetMap (OSM) and Sentinel data.

`sentinelhub`: The `sentinelhub` Python package provides users with API calls to make Open Geospatial Consortium (OGC) (Web Map Service and Web Coverage Service) web requests to download and process satellite imagery within Python scripts. It supports Sentinel-2 L1C and L2A, Sentinel-1, Landsat 8, MODIS and DEM data sources.

`overpy`: The `overpy` Python package provides a Python Wrapper to access the Overpass API to send queries to the OSM database and parse the response data in either JSON or XML formats.

`shapely`: The `shapely` Python package provides tools to manipulate planar geometric objects. This package is used to read the geometries of objects from OSM data and transfer them to satellite imagery data.

Apart from this, we made use of standard Python packages for handling data such as `pandas` and `numpy`.

A.2 Locating Objects

The first component of our framework extracts the location of objects by using a combination of public databases and OSM data. In this appendix, we provide code snippets from our demonstration of gathering a dataset containing satellite imagery of solar power plants using the Global Power Plant Database (GPPD) and OSM data.

A.2.1 Extracting Geo-coordinates from Database

We filter out the GPPD for all entries of solar power plants and then extract their geo-coordinates provided in the database. The class for handling these tasks is provided below.

```
class DBReader():
    def __init__(self):
        self.file = None
        self.df = None

    def apply_file(self, in_file):
        '''Stores in_file in self.file and generates dataframe from
        ↪  in_file. Stores dataframe in self.df'''
        self.file = in_file
        self.df = pd.read_csv(in_file)

    def print_db(self):
        '''Prints self.df'''
        print(self.df)

    def extract_fuel(self, dataframe, fuel_type):
        '''Return subset of dataframe where fuel_type is in one of
        ↪  the fuel columns'''
```

```
        return dataframe[(dataframe['fuel1']==fuel_type) |
        ↪  (dataframe['fuel2']==fuel_type) |
        ↪  (dataframe['fuel3']==fuel_type) |
        ↪  (dataframe['fuel4']==fuel_type)]

    def extract_country(self, dataframe, country):
        '''Return subset of dataframe where country is in country
        ↪  column'''
        return dataframe.groupby('country').get_group(country)

    def get_lat_lon(self, dataframe):
        '''Return new dataframe with attributes latitude and
        ↪  longitude projected from parameter dataframe'''
        return dataframe[['latitude', 'longitude']]

    def sort(self, dataframe, columns, ascending=True):
        '''Sorts dataframe by data in column'''
        return dataframe.sort_values(by=columns, ascending=ascending)
```

A.2.2 Searching OpenStreetMap

We search for the object on OSM by building a search window around the extracted geo-coordinates from the database. We build an Overpass API query to search for an object's OSM tag key-value pair within a particular bounding box defined by the search window. The class provided below contains functions for performing the above actions and some other utility functions to handle OSM data.

```
class OSMHandler():
    def __init__(self):
        self.api = overpy.Overpass()
        self.overpass_url = "http://overpass-api.de/api/interpreter"

    def build_query_from_dataframe(self, keys, values, dataframe,
    ↪  dist, intersection=False):
        '''Build Overpass query from lats/lons in dataframe'''
        query = "[out:json];("
        query_suffix = ");out bb;"
        for i in range(len(dataframe)):
            entry = dataframe.iloc[i]
```

```
            lat, lon = entry['latitude'], entry['longitude']
            bbox_lat = [lat - dist, lat + dist]
            bbox_lon = [lon - dist, lon + dist]
            for coord in bbox_lat:
                if coord < -90:
                    coord = coord + 90
                elif coord > 90:
                    coord = coord - 90
            for coord in bbox_lon:
                if coord < -180:
                    coord = coord + 180
                elif coord > 180:
                    coord = coord - 180
            bbox = [bbox_lat[0], bbox_lon[0], bbox_lat[1],
            ↪  bbox_lon[1]]
            if (bbox[0] < -90 or bbox[0] > 90 or bbox[1] < -180 or
            ↪  bbox[1] > 180):
                print(bbox)
                print(lat)
                print(lon)
            for j in range(len(keys)):
                query_body = "way"
                query_body += """["%s"="%s"]""" % (keys[j],
                ↪  values[j])
                query_body += "(%f, %f, %f, %f);" % (bbox[0],
                ↪  bbox[1], bbox[2], bbox[3])
            for j in range(len(keys)):
                query_body2 = "relation"
                query_body2 += """["%s"="%s"]""" % (keys[j],
                ↪  values[j])
                query_body2 += "(%f, %f, %f, %f);" % (bbox[0],
                ↪  bbox[1], bbox[2], bbox[3])
            query += query_body
            query += query_body2
        query += query_suffix
        return query

    def build_query_from_coords(self, kv_pairs, lat, lon, dist,
    ↪  neg_kv_pairs=None, get_bb=False):
        '''Build Overpass query from lats/lons in dataframe'''
        query = "[out:json];("
        query_suffix = ");"
        if len(neg_kv_pairs) > 0:
```

```
        query += "("
        query_suffix = "););"
    if get_bb:
        query_suffix += "out bb;"
    else:
        query += "("
        query_suffix += ">;); out meta;"
    bbox_lat = [lat - dist, lat + dist]
    bbox_lon = [lon - dist, lon + dist]
    for coord in bbox_lat:
        if coord < -90:
            coord = coord + 90
        elif coord > 90:
            coord = coord - 90
    for coord in bbox_lon:
        if coord < -180:
            coord = coord + 180
        elif coord > 180:
            coord = coord - 180
    bbox = [bbox_lat[0], bbox_lon[0], bbox_lat[1], bbox_lon[1]]
    if (bbox[0] < -90 or bbox[0] > 90 or bbox[1] < -180 or
    ↪  bbox[1] > 180):
        print(bbox)
        print(lat)
        print(lon)
    query_body =''
    for j in range(len(kv_pairs)):
        query_body += "way"
        query_body += """%s""" % kv_pairs[j]
        query_body += "(%f, %f, %f, %f);" % (bbox[0], bbox[1],
        ↪  bbox[2], bbox[3])
    query_body2 = ''
    for j in range(len(kv_pairs)):
        query_body2 += "relation"
        query_body2 += """%s""" % kv_pairs[j]
        query_body2 += "(%f, %f, %f, %f);" % (bbox[0], bbox[1],
        ↪  bbox[2], bbox[3])
    query += query_body
    query += query_body2
    neg_query_body = ''
    for j in range(len(neg_kv_pairs)):
        neg_query_body += "way"
        neg_query_body += """%s""" % neg_kv_pairs[j]
```

```
            neg_query_body += "(%f, %f, %f, %f);" % (bbox[0],
            ↪  bbox[1], bbox[2], bbox[3])
        neg_query_body2 = ''
        for j in range(len(neg_kv_pairs)):
            neg_query_body2 += "relation"
            neg_query_body2 += """%s""" % neg_kv_pairs[j]
            neg_query_body2 += "(%f, %f, %f, %f);" % (bbox[0],
            ↪  bbox[1], bbox[2], bbox[3])
        if len(neg_kv_pairs) > 0:
            query += '); - (%s%s' % (neg_query_body, neg_query_body2)
        query += query_suffix
        return query

    def json_query_osm(self, query):
        '''Query OSM using Overpass'''
        response = requests.get(self.overpass_url, params={'data':
        ↪  query})
        data = ''
        try:
            data = response.json()
        except:
            return ''
        return data

    def query_osm(self, query):
        '''Query OSM using Overpass'''
        while True:
            try:
                return self.api.query(query)
            except:
                s.sleep(30)
                continue
            else:
                break
    def way_lat_lon(self, way):
        '''Calculate and return lat/lon of way from member nodes'''
        lat, lon = 0, 0
        for node in way.nodes:
            lat += node.lat
            lon += node.lon
        return lat/len(way.nodes), lon/len(way.nodes)

    def print_way(self, way):
```

```
        '''Utility function to print way and member nodes'''
        print("Name: %s" % way.tags.get("name", "n/a"))
        print("Lat: %f, Lon: %f" % self.way_lat_lon(way))
        print("  Nodes:")
        for node in way.nodes:
            print("    Lat: %f, Lon: %f" % (node.lat, node.lon))
```

A.2.3 Extracting Geometries of Objects

We use the shapely package to extract the geometry of objects from the OSM Ways (polygonal footprints of objects). In this demonstration, we randomly sampled 5 points within the footprint of a solar power plant and cropped 256x256 image chips centered on these points. The class provided below contains functions to perform the above cropping technique and some other utility functions to handle the OSM geometries.

```
class Cropper():
    '''Creates num_points crop_size*crop_size crops of OSM way
    ↪ polygons'''
    def __init__(self, crop_size=256):
        self.crop_size = crop_size


    def build_polys(self, overpass_result):
        '''Returns polygons built from ways in overpass_result'''
        ways = []
        polys = []
        for relation in overpass_result.relations:
            for rel in relation.members:
                ways.append(overpass_result.get_way(rel.ref))
        for way in overpass_result.ways:
            ways.append(way)
        for way in ways:
            points = [(x, y) for x, y in zip([node.lat for node in
            ↪ way.nodes], [node.lon for node in way.nodes])]
            polys.append(Polygon(points))
        return polys
```

```
    def random_points_within(self, poly, num_points):
        '''Returns self.num_points random points within poly'''
        min_x, min_y, max_x, max_y = poly.bounds
        points = []
        while len(points) < num_points:
            random_point = Point([np.random.uniform(min_x, max_x),
            ↪  np.random.uniform(min_y, max_y)])
            if (random_point.within(poly)):
                points.append(random_point)
        return points

    def random_points_between(self, inside_polys, outside_poly,
    ↪  num_points):
        '''Returns self.num_points random points within poly'''
        min_x, min_y, max_x, max_y = outside_poly.bounds
        points = []
        while len(points) < num_points:
            random_point = Point([np.random.uniform(min_x, max_x),
            ↪  np.random.uniform(min_y, max_y)])
            inside = False
            for poly in inside_polys:
                if random_point.within(poly):
                    inside = True
            if (random_point.within(outside_poly) and not inside):
                points.append(random_point)
        return points

    def point_to_bbox(self, point, dims, resolution):
        lat, lon = point.x, point.y
        utm_point = wgs84_to_utm(lon, lat)
        res_x, res_y = resolution if isinstance(resolution, tuple)
        ↪  else (resolution, resolution)
        dim_x, dim_y = dims if isinstance(dims, tuple) else (dims,
        ↪  dims)
        dist_x, dist_y = (dim_x/2) * res_x, (dim_y/2) * res_y
        utm_crs = get_utm_crs(lon, lat)
        utm_bbox = BBox([utm_point[0] - dist_x, utm_point[1] -
        ↪  dist_y, utm_point[0] + dist_x, utm_point[1] + dist_y],
        ↪  crs=utm_crs)
        east1, north1 = utm_bbox.lower_left
        east2, north2 = utm_bbox.upper_right
        wgs_ll, wgs_ur = to_wgs84(east1, north1, utm_crs),
        ↪  to_wgs84(east2, north2, utm_crs)
```

```
        return BBox([wgs_ll[0], wgs_ll[1], wgs_ur[0], wgs_ur[1]],
        ↪  crs=CRS.WGS84)

    def latlon_to_bbox(self,lat,lon, dims, resolution):

        utm_point = wgs84_to_utm(lon, lat)
        res_x, res_y = resolution if isinstance(resolution, tuple)
        ↪  else (resolution, resolution)
        dim_x, dim_y = dims if isinstance(dims, tuple) else (dims,
        ↪  dims)
        dist_x, dist_y = (dim_x/2) * res_x, (dim_y/2) * res_y
        utm_crs = get_utm_crs(lon, lat)
        utm_bbox = BBox([utm_point[0] - dist_x, utm_point[1] -
        ↪  dist_y, utm_point[0] + dist_x, utm_point[1] + dist_y],
        ↪  crs=utm_crs)
        east1, north1 = utm_bbox.lower_left
        east2, north2 = utm_bbox.upper_right
        wgs_ll, wgs_ur = to_wgs84(east1, north1, utm_crs),
        ↪  to_wgs84(east2, north2, utm_crs)
        return BBox([wgs_ll[0], wgs_ll[1], wgs_ur[0], wgs_ur[1]],
        ↪  crs=CRS.WGS84)

    def get_poly_crops(self, points, res):
        '''Returns num_points random bboxes centered within poly'''
        bboxes = []
        for point in points:
            bboxes.append(self.point_to_bbox(point, self.crop_size,
            ↪  res))
        return bboxes
```

A.2.4 Downloading Satellite Imagery

We use the Sentinel Hub Python package to download satellite imagery from Sentinel-2 database. The class provided below allows downloading images of multiple bands available in Sentinel-2 data. We can request images using either WMS or WCS web requests. Sentinel Hub [23] provides free trials which allow a limited number of imagery requests per day. You can also subscribe to paid plans available on their website for downloading larger amounts of data.

```
class SentinelExtract():
    def __init__(self, instance_id, data_folder=None):
        self.data_folder = data_folder
        self.instance_id = instance_id

    def plot_image(self, image, i=1, factor=1, cmp = 'viridis'):
        """
        Utility function for plotting images.
        """
        plt.subplots(nrows=1, ncols=1, figsize=(15, 7))

        if np.issubdtype(image.dtype, np.floating):
            plt.imshow(np.minimum(image * factor, 1),cmap=cmp)
        else:
            plt.imshow(image,cmap=cmp)

    def build_bbox(self,bbox):
        '''Build bbox from lat/lon coordinates'''
        return BBox(bbox=bbox, crs=CRS.WGS84)

    def RGB_WMS_request(self,bbox,max_cloud,res,time='latest',
    ↪  iid=None,data_folder=None,dim_filter=None,verbose=False):
        '''Send request to Sentinel Hub Web Map Service for RGB
        ↪  image and save and return request result and true color
        ↪  image'''
        iid = self.instance_id if iid is None else iid
        data_folder = self.data_folder if data_folder is None else
        ↪  data_folder
        w, h = bbox_to_dimensions(bbox,(res,res))
        if verbose:
            print("Image dimensions: %i, %i" % (w,h))
        if dim_filter and (w < dim_filter and h < dim_filter):
            return None, False
        if verbose:
            print("Retrieving images...")
        wms_true_color_request =
        ↪  WmsRequest(layer='TRUE-COLOR-S2-L1C',
                                    bbox=bbox,
                                    time=time,
                                    width = w,
                                    height=h,
                                    maxcc=max_cloud,
```

```
                                    instance_id=iid,
                                    data_folder=data_folder)
        wms_true_color_img =
        ↪  wms_true_color_request.get_data(save_data=True)
        '''
        if data_folder is not None:
            wms_true_color_request.save_data(redownload=True)
        '''
        if verbose:
            print('There are %d Sentinel-2 images available with
            ↪  cloud coverage less than %1.0f%%.' %
            ↪  (len(wms_true_color_img),
            ↪  wms_true_color_request.maxcc * 100.0))
        return wms_true_color_img, len(wms_true_color_img)

    def RGB_WCS_request(self,bbox,res,max_cloud,time='latest',
    ↪  iid=None,data_folder=None,verbose=False):
        '''Send request to Sentinel Hub Web Coverage Service for
        ↪  RGB image and save and return request result and true
        ↪  color image'''
        iid = self.instance_id if iid is None else iid
        data_folder = self.data_folder if data_folder is None else
        ↪  data_folder
        if verbose:
            print("Retrieving images...")
        wcs_true_color_request =
        ↪  WcsRequest(layer='TRUE-COLOR-S2-L1C',
                                    bbox=bbox,
                                    time=time,
                                    resx = res,
                                    maxcc=max_cloud,
                                    instance_id=iid,
                                    data_folder=data_folder)
        wcs_true_color_img = wcs_true_color_request.get_data()
        if data_folder is not None:
            wcs_true_color_request.save_data(redownload=True)
        if verbose:
            print('There are %d Sentinel-2 images available with
            ↪  cloud coverage less than %1.0f%%.' %
            ↪  (len(wcs_true_color_img),
            ↪  wcs_true_color_request.maxcc * 100.0))
        return wcs_true_color_img, len(wcs_true_color_img) > 0
```

```
    def all_bands_WMS_request(self,bbox,max_cloud,time='latest',
    ↪  iid=None,data_folder=None,dim_filter=None,verbose=False):
        '''Send request to Sentinel Hub Web Map Service for all
        ↪  bands and save and return request result and true color
        ↪  image'''
        iid = self.instance_id if iid is None else iid
        data_folder = self.data_folder if data_folder is None else
        ↪  data_folder
        w, h = bbox_to_dimensions(bbox,(10,10))
        if verbose:
            print("Image dimensions: %i, %i" % (w,h))
        if dim_filter and (w < dim_filter and h < dim_filter):
            return None, False
        if verbose:
            print("Retrieving images...")
        wms_true_color_request = WmsRequest(layer='BANDS-S2-L1C',
                                    bbox=bbox,
                                    time=time,
                                    width = w,
                                    height=h,
                                    maxcc=max_cloud,
                                    instance_id=iid,
                                    data_folder=data_folder,
                                    image_format=MimeType.TIFF_d32f)
        wms_true_color_img =
        ↪  wms_true_color_request.get_data(save_data=True)
        '''
        if data_folder is not None:
            wms_true_color_request.save_data(redownload=True)
        '''
        if verbose:
            print('There are %d Sentinel-2 images available with
            ↪  cloud coverage less than %1.0f%%.' %
            ↪  (len(wms_true_color_img),
            ↪  wms_true_color_request.maxcc * 100.0))
        return wms_true_color_img, len(wms_true_color_img) > 0

    def Bands_WMS_request(self,band,bbox,max_cloud,res,time='latest',
    ↪  iid=None,data_folder=None,dim_filter=None,verbose=False):
        '''Send request to Sentinel Hub Web Map Service for all
        ↪  bands and save and return request result and true color
        ↪  image'''
        iid = self.instance_id if iid is None else iid
```

```
        data_folder = self.data_folder if data_folder is None else
        ↪  data_folder
        w, h = bbox_to_dimensions(bbox,(res,res))
        if verbose:
            print("Image dimensions: %i, %i" % (w,h))
        if dim_filter and (w < dim_filter and h < dim_filter):
            return None, False
        if verbose:
            print("Retrieving images...")
        wms_true_color_request = WmsRequest(layer=band,
                                    bbox=bbox,
                                    time=time,
                                    width = w,
                                    height=h,
                                    maxcc=max_cloud,
                                    instance_id=iid,
                                    data_folder=data_folder,
                                    image_format=MimeType.TIFF_d32f)
        wms_true_color_img =
        ↪  wms_true_color_request.get_data(save_data=True)
        '''
        if data_folder is not None:
            wms_true_color_request.save_data(redownload=True)
        '''

        if verbose:
            print('There are %d Sentinel-2 images available with
            ↪  cloud coverage less than %1.0f%%.' %
            ↪  (len(wms_true_color_img),
            ↪  wms_true_color_request.maxcc * 100.0))
        return wms_true_color_img, len(wms_true_color_img)

    def all_bands_WCS_request(self,bbox,res,max_cloud,time='latest',
    ↪  iid=None,data_folder=None,verbose=False):
        '''Send request to Sentinel Hub Web Coverage Service for
        ↪  all bands and save and return request result and true
        ↪  color image'''
        iid = self.instance_id if iid is None else iid
        data_folder = self.data_folder if data_folder is None else
        ↪  data_folder
        if verbose:
            print("Retrieving images...")
        wcs_true_color_request = WcsRequest(layer='BANDS-S2-L1C',
```

```
                                    bbox=bbox,
                                    time=time,
                                    resx = res,
                                    maxcc=max_cloud,
                                    instance_id=iid,
                                    data_folder=data_folder,
                                    image_format=MimeType.TIFF_d32f)
        wcs_true_color_img = wcs_true_color_request.get_data()
        if data_folder is not None:
            wcs_true_color_request.save_data(redownload=True)
        if verbose:
            print('There are %d Sentinel-2 images available with
             ↪  cloud coverage less than %1.0f%%.' %
             ↪  (len(wcs_true_color_img),
             ↪  wcs_true_color_request.maxcc * 100.0))
        return wcs_true_color_img, len(wcs_true_color_img) > 0
```

The above code snippets make up the different components of our data collection framework used to collect imagery of solar power plants. They can be modified to collect imagery of any other object mapped by OSM and public geo-databases. The above framework has been built to use Sentinel-2 satellite imagery. The satellite imagery component can be swapped out to use a different geo-registered imagery source. We developed our pipeline to be highly modular to support easy changes or additions in the future.

Bibliography

[1] H. Ling, Y. Wu, E. Blasch, G. Chen, H. Lang, and L. Bai, "Evaluation of visual tracking in extremely low frame rate wide area motion imagery," in *14th International Conference on Information Fusion*, pp. 1–8, IEEE, 2011.

[2] Rovito, Todd and Patrick, James and Walls, Steve and Uppenkamp, Daniel and Mendoza-Schrock, Olga and Velten, Vince and Curtis, Chris and Priddy, Kevin, "Columbus Large Image Format (CLIF) 2007 Dataset.." `https://github.com/AFRL-RY/data-clif-2007`, 2007.

[3] Leong, Colin and Rovito, Todd and Mendoza-Schrock, Olga and Menart, Christopher and Bowser, Jason and Moore, Linda and Scarborough, Steve and Minardi,Michael and Hascher, David , "Unified Coincident Optical and Radar for Recognition (UNICORN) 2008 Dataset." `https://github.com/AFRL-RY/data-unicorn-2008`, 2008.

[4] A. Basharat, M. Turek, Y. Xu, C. Atkins, D. Stoup, K. Fieldhouse, P. Tunison, and A. Hoogs, "Real-time multi-target tracking at 210 megapixels/second in wide area motion imagery," in *IEEE Winter Conference on Applications of Computer Vision*, pp. 839–846, IEEE, 2014.

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

[6] European Space Agency, "Copernicus Sentinel-2 data [2019]." `https://sentinel.esa.int/web/sentinel/missions/sentinel-2`.

[7] E. Blasch, G. Seetharaman, S. Suddarth, K. Palaniappan, G. Chen, H. Ling, and A. Basharat, "Summary of methods in wide-area motion imagery (wami)," in *Geospatial InfoFusion and Video Analytics IV; and Motion Imagery for ISR and Situational Awareness II*, vol. 9089, p. 90890C, International Society for Optics and Photonics, 2014.

[8] E. Blasch, J. G. Herrero, L. Snidaro, J. Llinas, G. Seetharaman, and K. Palaniappan, "Overview of contextual tracking approaches in information fusion," in *Geospatial InfoFusion III*, vol. 8747, p. 87470B, International Society for Optics and Photonics, 2013.

[9] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3431–3440, 2015.

[10] M. McDonald, K. Wei, T. Kirubarajan, Z. Baird, and S. Rajan, "Machine learning for wide area surveillance from aerial platforms," in *2018 International Workshop on Computing, Electromagnetics, and Machine Intelligence (CEMi)*, pp. 3–4, IEEE, 2018.

[11] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*, pp. 234–241, Springer, 2015.

[12] A. Albert, J. Kaur, and M. C. Gonzalez, "Using convolutional networks and satellite imagery to identify patterns in urban environments at a large scale," in *Proceedings ACM SIGKDD*, 2017.

[13] N. Kussul, M. Lavreniuk, S. Skakun, and A. Shelestov, "Deep learning classification of land cover and crop types using remote sensing data," *IEEE Geoscience and Remote Sensing Letters*, vol. 14, no. 5, 2017.

[14] T. Ishii, E. Simo-Serra, S. Iizuka, Y. Mochizuki, A. Sugimoto, H. Ishikawa, and R. Nakamura, "Detection by classification of buildings in multispectral satellite imagery," in *ICPR*, 2016.

[15] D. Sui, M. Goodchild, and S. Elwood, "Volunteered geographic information, the exaflood, and the growing digital divide," in *Crowdsourcing Geographic Knowledge*, Springer, 2013.

[16] B. A. Johnson and K. Iizuka, "Integrating OpenStreetMap crowdsourced data and Landsat time-series imagery for rapid land use/land cover (LULC) mapping: Case study of the Laguna de Bay area of the Philippines," *Applied Geography*, vol. 67, 2016.

[17] M. Schultz, J. Voss, M. Auer, S. Carter, and A. Zipf, "Open land cover from OpenStreetMap and remote sensing," *International Journal of Applied Earth Observation and Geoinformation*, vol. 63, 2017.

[18] N. Audebert, B. Le Saux, and S. Lefèvre, "Joint Learning from Earth Observation and OpenStreetMap Data to Get Faster Better Semantic Maps," in *Proceedings CVPR Workshop: Large Scale Computer Vision for Remote Sensing Imagery*, 2017.

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

[20] W. Zhao, Y. Bo, J. Chen, D. Tiede, B. Thomas, and W. J. Emery, "Exploring semantic elements for urban scene recognition: Deep integration of high-resolution imagery and OpenStreetMap (OSM)," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 151, 2019.

[21] J. Lee, S. R. Pant, and H.-S. Lee, "An adaptive histogram equalization based local technique for contrast preserving image enhancement," *International Journal of Fuzzy Logic and Intelligent Systems*, vol. 15, no. 1, pp. 35–44, 2015.

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

[23] Sinergise Ltd., "Modified Copernicus Sentinel data [2019]/Sentinel Hub." `https://sentinel-hub.com/`.

[24] OpenStreetMap Wiki, "Nominatim." `https://wiki.openstreetmap.org/w/index.php?title=Nominatim&oldid=1848597`, 2019.

[25] L. Byers, J. Friedrich, R. Hennig, A. Kressig, X. Li, C. McCormick, and L. M. Valeri, "A global database of power plants," *World Resources Institute*, vol. 18, 2018.

[26] OpenStreetMap Wiki, "Map Features." `https://wiki.openstreetmap.org/w/index.php?title=Map_Features&oldid=1819914`, 2019.

[27] OpenStreetMap Wiki, "Overpass API." `https://wiki.openstreetmap.org/w/index.php?title=Overpass_API&oldid=1872170`, 2019.

[28] SimpleMaps- Pareto Software, LLC, "World Cities Database." `https://simplemaps.com/data/world-cities`, 2019.

[29] Nordpil, "World database of large urban areas, 1950-2050." `https://nordpil.com/resources/world-database-of-large-cities/`, 2019.

[30] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 40, no. 4, 2017.

[31] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network," in *Proceedings of CVPR*, 2017.