May 2, 2018

Satellite Image Finder Parking Lot & Spots
Final Report

Department of Computer Science
Virginia Tech
Blacksburg, VA 24061

Class: CS4624: Multimedia, Hypertext, and Information Access - Spring 2018
Instructor: Dr. Edward A. Fox
Client: Mary Carome - Reinventing Geospatial
Team: Alex Lambrides, Thomas Wolfe, Khoa Le, Patrick Jahnig

Table of Contents

Table of Figures

Table of Tables

I. Executive Summary

Satellite imagery in recent years has drastically increased in both quality and quantity. Today, the problem is too much data. Map features such as roads, buildings, and other points of interest are mainly extracted manually, and we just don't have enough humans to carry out this mundane task.

The goal of this project is to develop a tool that automates this process. Specifically, the focus of this project is to extract parking lots using Object Based Imagery Analysis. The final deliverable is a Python tool that uses Machine Learning algorithms to identify and extract parking lots from high resolution satellite imagery.

This project was divided into two main steps: labeling data and training an algorithm. For the first step, the project team gathered a large dataset of satellite imagery in the form of GeoTIFFs, used GDAL to convert these files into JPEG image files, and used labelImg to label the images. The labelling consisted of creating an XML layer corresponding to each GeoTIFF image, where the XML layer contained bounding boxes outlining each parking lot. With all of the training data labeled, the next step was training the algorithm. The project lead tried several different models for the learning algorithm, with the final model being based on Faster RCNN.

After training, the project team tested the model and determined the accuracy was too low, so the team decided to obtain and label more images to improve it. Once the accuracy met the determined standards, a script was built that would take an input of a GeoTIFF image, convert this to a JPEG image, run the image on the model to detect any parking lots and output bounding boxes depicting those parking lots, and finally, convert these bounding boxes into a single GeoJSON file. The main use case of the application is quickly finding parking lots with relative accuracy in satellite imagery. The model can also be built upon to be improved or used in related tasks, for example detecting individual parking spots.

The project has managed to achieve the expected goals using labelImg and a Faster RCNN model. However, due to a limitation of labelImg, the model cannot detect parking lots that are not horizontal or vertical. The project team researched several methods to solve this problem but were not able to fully implement a suitable solution due to time and infrastructure constraints. The team has described all of its research in this final report so that those who want to improve on this project will have a good starting point.

II. Introduction

1. Objective

The main goal of the project is to take high resolution satellite images and detect parking lots in them. The output is a GeoJSON[1] file consisting of polygons that outline the parking lots within the GeoTIFF [2]image. The stretch goal of the project was to detect the individual parking spots, and whether or not that parking spot is currently taken. A final tool was created that allows users to input a GeoTIFF image, and they will receive a corresponding GeoJSON outlining all parking lots, if any were found. Due to the complex nature of this task, the model has a total loss of around 0.4. Users are free to do what they want with the information provided. They could use it for real time analysis of parking spaces, the neural network could be built on top of for more imagery analysis tasks, or the tool could be combined with another to detect more than just parking lots.

2. Report Outline

First, the requirements that the team came up with for building this project will be outlined, including basic requirements as well as eventual stretch goals (Section III). Next is the actual design of the project (Section IV) and how the team went about designing the two basic steps of the project (labelling and training). The implementation section (Section V) details the steps of the actual implementation, specifically how the project team trained the model. After implementing the model, the next task was testing (Section VI) the model to ensure accuracy, and making adjustments as needed.

The user's manual (Section VII) outlines how to use the final deliverable. It details downloading and setting up the Python tool, and how to run it from a Jupyter notebook. This tool is meant for anyone to be able to use their own GeoTIFF images, and output a GeoJSON of the parking lots found. The developer's manual (Section VIII) describes in detail how one can pick up from where the project team left off to improve upon the results. Finally, the lessons learned through the course of the project will be discussed (Section IX), followed by the appendices (Section X).

3. Client

The client who supported this semester-long project was Reinventing Geospatial (RGI), specifically Mary Carome, Steven Lander, and Michael Szaszy. RGI is a consulting company specializing in geospatial technologies. One of the key areas RGI works in is data science, so training a neural network to detect parking lots could benefit them in this area. Building a machine learning algorithm, with a tool to use it, not only aids in automatic detection of parking lots, but the team's model can be used to help build models for other similar tasks.

[1] GeoJSON: http://geojson.org
[2] OSGeo: trac.osgeo.org/geotiff/

III. Requirements

1. Basic requirements

For basic requirements, the team wanted to be able to distinguish between satellite images that have parking lots and those that do not. The final product would be a Python script / tool that is able to take in a satellite image (GeoTIFF) and output a GeoJSON containing the position of the parking lots. The script should be able to distinguish if the input files are in correct format, and convert the GeoTIFF into a JPEG for use with the model.

For the machine learning algorithm itself, it should be able to distinguish parking lots in different levels of visibility, e.g., at night when the parking lines are harder to see, or in the snow when the parking lines blend in. As long as the human eye can discern a parking lot, the algorithm should be able to recognize it as well.

The project team decided to exclude cases where parking lots are obscured to the point of being unrecognizable. This meant that while a human might look at the image and from subtle visual cues decide that a parking lot does exist in that image, it is a hard task for an algorithm to correctly determine that without also recognizing other non-parking lots as parking lots. An example of this is when a parking lot is not visible due to the shading caused by trees or buildings, or in the case of covered parking lots.

2. Stretch goals

The stretch goal for this project is, after determining whether or not the satellite image contains parking lots, the algorithm will also detect whether or not these parking lots have empty spaces. The algorithm must look at individual parking spaces inside a parking lot to determine whether or not there is a car parked there. It should be able to differentiate between a parking lot with cars, which will cause the parking lot to consist of potentially brighter colors, to a parking lot without one, which would be more uniform and consisting of darker colors.

The algorithm would then ideally output whether or not any parking spaces are available. If there are any available parking spaces, the algorithm should produce a GeoJSON file that contains the coordinates of the available parking spaces.

Unfortunately due to time constraints and other unforeseen problems, the project team were unable to meet the stretch goals.

IV. Design

1. Labelling the Data

The satellite images that the model was trained on were retrieved from SpaceNet[3]. A GeoTIFF file is an image that contains geographical coordinates as well as the geographical area that those coordinates represent. The model required that the training use JPEG image files; therefore the first step was converting the GeoTIFF files to JPEG files using GDAL[4]. The project team used the open source software labelImg[5] to process these images into corresponding XML files in the PASCAL VOC[6] format by labeling any parking lot within the images by hand.

The corresponding XML files simply contained bounding boxes, by noting the four corners of the box in terms of pixels. The major downside of using labelImg (also a downside of the model), is that rotated parking lots cannot be accurately labelled. The boxes can only be drawn along the horizontal and vertical axes with no rotation, so rotated parking lots required large bounding boxes that contained other parts of the image.

2. Applying the Algorithm

The project team decided to use the TensorFlow[7] deep learning framework as a base for the learning implementation. The satellite images obtained in the first phase were converted from GeoTIFF images to JPEG images and labelled according to pixel coordinates. This is to match the format of the training data so that it is consistent with the general import of training data. In order to solve the problem of object recognition in imagery the team decided to use Faster R-CNN[8]. This is a faster region-based convolutional neural network. The way this works is by running a convolutional neural network over images, followed by a region proposal network. This allows the network to identify regions of interest within the image that may correlate with the specified object.

[3] SpaceNet AWS: https://registry.opendata.aws/spacenet/
[4] GDAL: http://www.gdal.org
[5] labelImg: https://github.com/tzutalin/labelImg
[6] PASCAL Visual Object Classes: http://host.robots.ox.ac.uk/pascal/VOC/
[7] TensorFlow: https://www.tensorflow.org
[8] Faster-RCNN: https://arxiv.org/abs/1506.01497

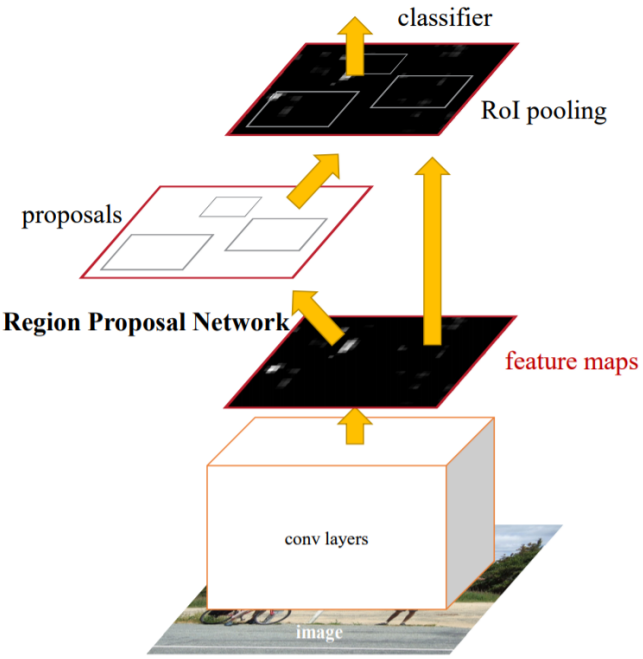

Figure 1. Faster R-CNN Overview[9]

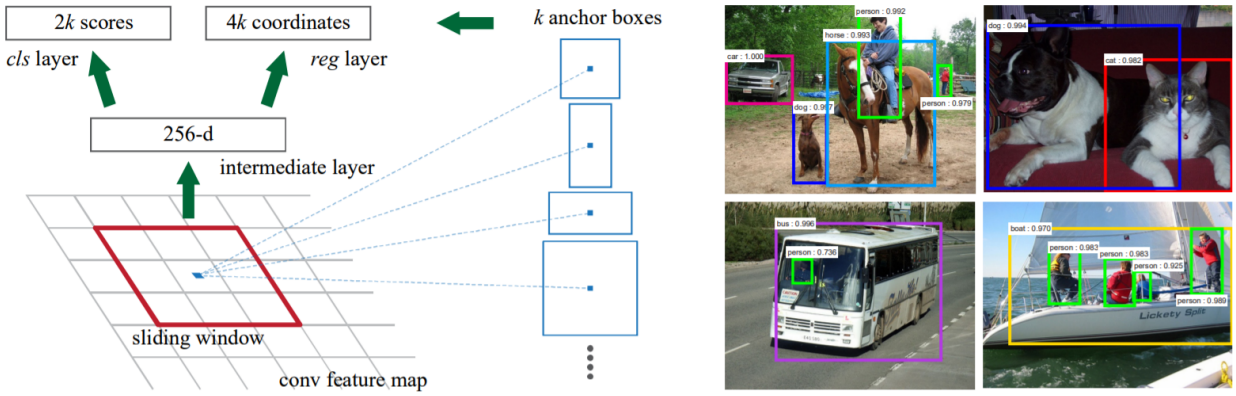

Figure 2. Faster R-CNN RPN and Example Detections[9]

As depicted in Figures 1 and 2, this algorithm classifies images within different region proposals in order to determine whether they contain the feature being searched for. Once this is conducted it is possible to create bounding boxes that pool different regions that have the object in order to generate an area that identifies the object of interest. This learning technique has been chosen as it is the fastest and most accurate iteration of a region based network. Once trained it generates bounding boxes that will contain parking lots. The pixel coordinates of these bounding boxes can then be converted back to geographic coordinates in order to provide a final output of geographical locations of parking lots within satellite imagery.

[9] Faster R-CNN Overview: https://arxiv.org/abs/1506.01497

V. Implementation

1. Acquiring Satellite Images

SpaceNet is a collection of thousands of satellite images as well as labelled data, hosted on AWS with the intention of fostering innovation in computer vision algorithms. The data consists of five different areas of interest: Rio de Janeiro, Las Vegas, Paris, Shanghai, and Khartoum. Rio de Janeiro consists of 50 cm imagery, while the rest consist of 30 cm imagery. The team decided on using Las Vegas as it was assumed there would be a much higher chance of parking lots. After evaluating the model, the team determined that more images were required, and as such downloaded similar .tar.gz files from Paris and Shanghai. The .tar.gz file the team needed from Las Vegas contained the following directories:

```
├── AOI_2_Vegas_Test_Public.tar.gz
│   ├── MUL           # Contains Tiles of 8-Band Multi-Spectral raster data from WorldView-3
│   ├── MUL-PanSharpen # Contains Tiles of 8-Band Multi-Spectral raster data pansharpened to 0.3m
│   ├── PAN           # Contains Tiles of Panchromatic raster data from Worldview-3
│   └── RGB-PanSharpen # Contains Tiles of RGB raster data from Worldview-3
```

For the purpose of this project, the team used 3-band imagery (RGB) and so did not need the 8-band images. After downloading the file, the team extracted the RGB-PanSharpen directory to obtain 1,282 GeoTIFF image files. Each of these files is 650 x 650 pixels. An example of what a file might look like can be seen in Figure 3:

Figure 3. Sample Las Vegas GeoTIFF

The SpaceNet data is hosted on a Requester Pays S3 bucket on AWS[10]. In order to download this data, the team had to use aws-cli commands. Due to timeouts on the commands, it was required to download the files in multiple parts. First, the team had to use this command three times with corresponding byte ranges: `aws s3api get-object --bucket spacenet-dataset --key AOI_2_Vegas/AOI_2_Vegas_Test_Public.tar.gz --request-payer requester AOI_2_Vegas_Test_Public.tar.gz --range bytes=3000000000-6000000000`. Next, after downloading three separate parts of the file, the team was able to combine them using the command: `cat AOI_2_Vegas_Test_Public_2.tar.gz >> AOI_2_Vegas_Test_Public_all_2.tar.gz`. The team used this command on the three parts we downloaded in the correct order to get the entire .tar.gz file. From there the team was able to correctly extract the directories and proceed to labelling.

2. Converting GeoTIFF to JPEG

In order to label the satellite images in labelImg, the GeoTIFFs had to be converted to JPEG format. To do this, the project team used an open source tool called GDAL to convert the GeoTIFF files into JPEG files. Two members wrote a script (convert.py) to do this conversion easily. The main command that was used was `gdal_translate`, using the geographic information that was extracted. In the convert.py script, the following command was used: `gdal_translate -scale_1 20 1463 -scale_2 114 1808 -scale_3 139 1256 -ot Byte -of`.

The convert.py script was run using `python convert.py -t jpeg` to convert all the GeoTIFF files in the current directory and subdirectories into JPEG files. With all of the files converted to the correct format, the team was able to start labelling.

3. Satellite Image Labelling

Labelling the JPEG satellite images for this model was done through an open source software tool called labelImg. The output is an XML file in the PASCAL VOC format (common for projects such as this) containing all of the bounding boxes for a single image. The format of that XML file is quite simple as seen in Figure 4 with the important parts highlighted:

[10] Requester Pays Buckets: https://docs.aws.amazon.com/AmazonS3/latest/dev/RequesterPaysBuckets.html

```xml
<annotation>
    <folder>Lots JPEGS</folder>
    <filename>54.jpg</filename>
    <size>
        <width>650</width>
        <height>650</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>parking lot</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>381</xmin>
            <ymin>127</ymin>
            <xmax>407</xmax>
            <ymax>336</ymax>
        </bndbox>
    </object>
```

Figure 4. Example Labelled XML

For the purpose of this project, the team labelled each bounding box as "parking lot." Figure 5 shows how one would label an image using labelImg:

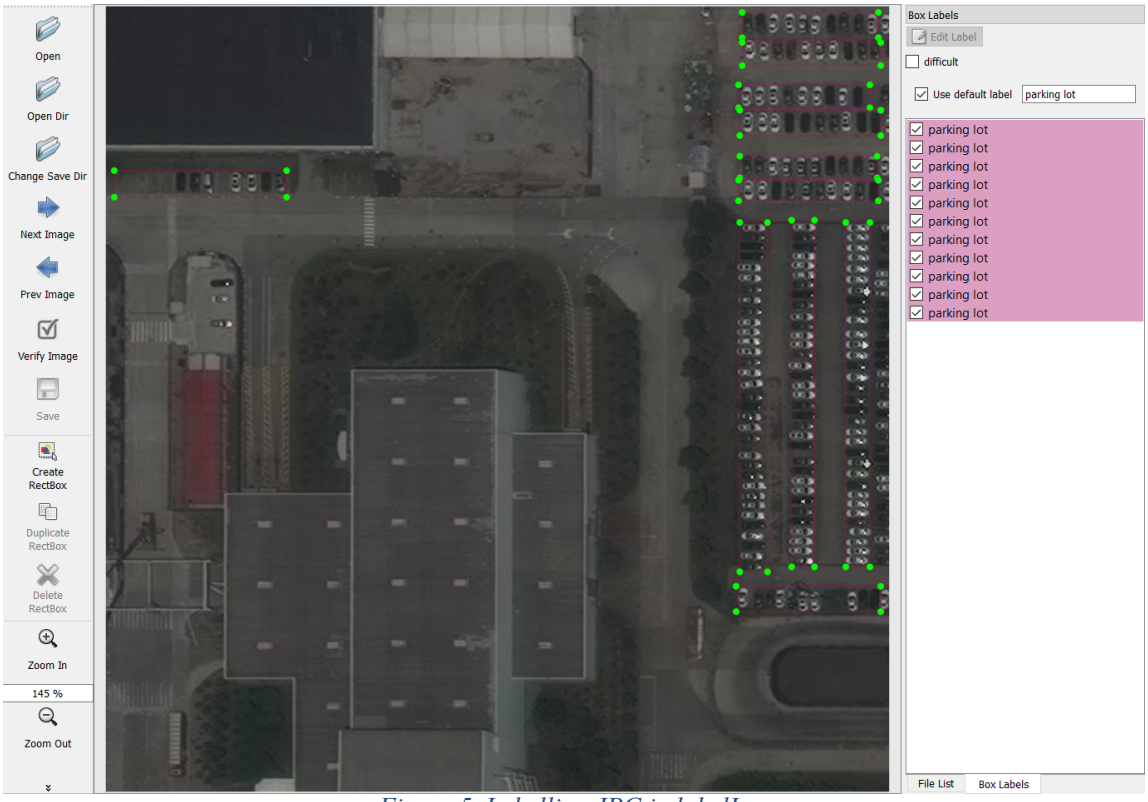

Figure 5. Labelling JPG in labelImg

One of the key issues with the model is the inability to rotate bounding boxes. In satellite images, often parking lots are arbitrarily rotated, and as such presented difficulties when labelling. This required the team to have large bounding boxes that contained parts of the image that were not the parking lot. Figure 6 shows how this was accomplished:

Figure 6. Difficulty with Rotated Parking Lots

4. Parking Lot Identification Algorithm

The implementation of the deep learning algorithm works through the use of the Tensorflow framework. In order to facilitate efficient learning the team loaded CUDA[11] and cuDNN[12] onto a machine with a NVIDIA GeForce GTX 960M. This allows for the use of the graphics card for processing, which greatly increases the speed of training the network. The Tensorflow framework is then used to communicate and apply the layers of the learning network. In order to conduct training the team is currently using the Faster-RCNN-RESNET101-COCO[13] implementation. This is used for initial training, and to gauge functionality of the method, which the team can then tweak to further meet the project's needs.

In order to train the neural network, parking lots are identified within JPEG images and their bounding boxes represented by an XML file similar to the PASCAL VOC dataset training format, which is a common way to represent training data for region-based networks. This information is then converted into the TFRecords, which is the format for training data of Tensorflow. The team uses this training data to train the faster region-based CNN, which will then create predictors of the locations of parking lots displayed by bounding boxes. This can then be mapped back to geographical coordinates, which accomplishes the goal of identifying parking lots in satellite imagery.

[11] NVIDIA CUDA: https://developer.nvidia.com/cuda-zone
[12] NVIDIA cuDNN: https://developer.nvidia.com/cudnn
[13] Faster-RCNN-RESNET101-COCO: https://github.com/endernewton/tf-faster-rcnn

Figure 7 shows the principle of this method on a small training set. In this figure the green boxes depict the predicted location of a parking lot. This percentage represents the confidence that the network has in the generated bounding box being a parking lot. This figure represents one example of the general problem with the initial training run. With the use of a small training set the network had a huge localization loss, which is demonstrated in this image by parking lots with confidence over 50% producing very inaccurate bounding boxes.

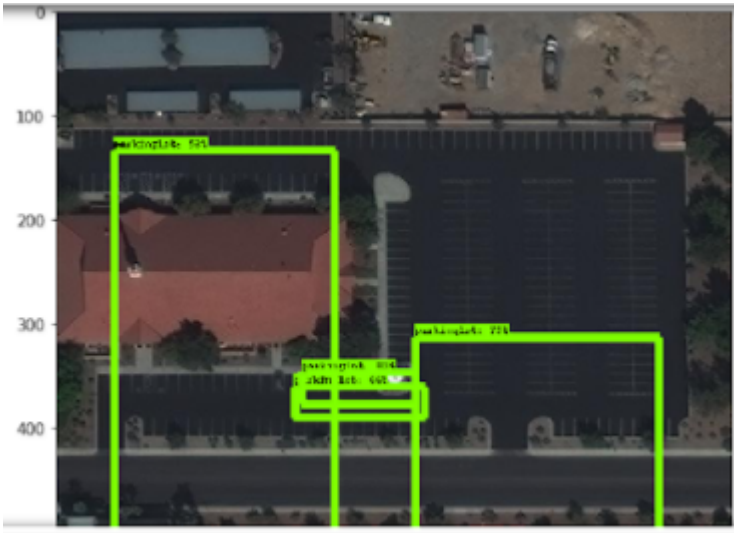

Figure 7. Initial Test Run of the Algorithm

VI. Testing and Evaluation

The first test run of the algorithm provided poor results. The classification was decent, however, the localization was very off. This was likely due to the small training set the algorithm was initially provided (~50 labelled images).

As can be seen in Figure 8, the algorithm was not able to accurately identify where the parking lots were during the first phase of testing. In this figure the green boxes depict the estimated locations of parking lots, and the red boxes have been manually inserted to highlight parking lots that were not recognized. It can be seen that many were missed, and the localization is quite inaccurate. Additionally, the model's bounding box regression appears to heavily favor two distinct shapes for the bounding boxes, which is likely due to the small size of the dataset.

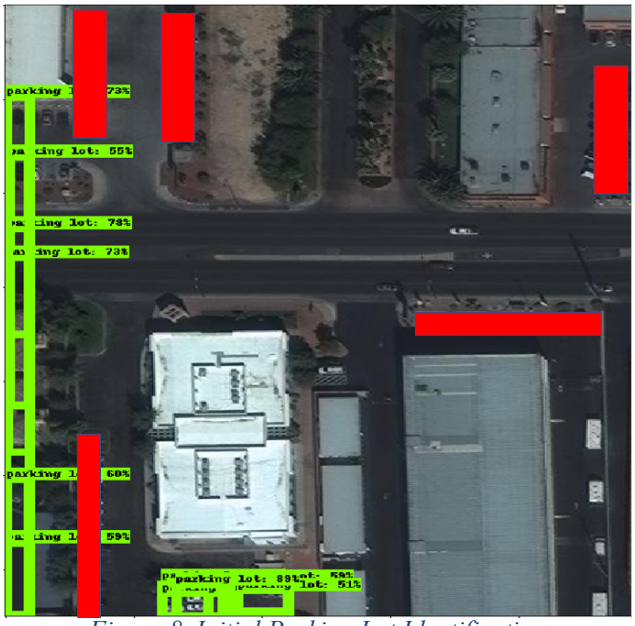

Figure 8. Initial Parking Lot Identification

The network's difficult to accurately identify the location of parking lots can be confirmed by the information presented in Figure 9. This figure represents the localization loss over iterations of training the network weights. Ideally a consistent downward trend would be visible, however, the dominant orange line, which represents the smoothed localization loss struggles to decrease consistently. This is exacerbated when viewing the faint line, which represents the loss without smoothing, however, due to the small testing set it is expected that for some iterations the loss will be high. Since localization loss represents the accuracy of scaling the outputted bounding boxes, the results of this graph provide a good explanation for the inaccuracy in the initial training run.

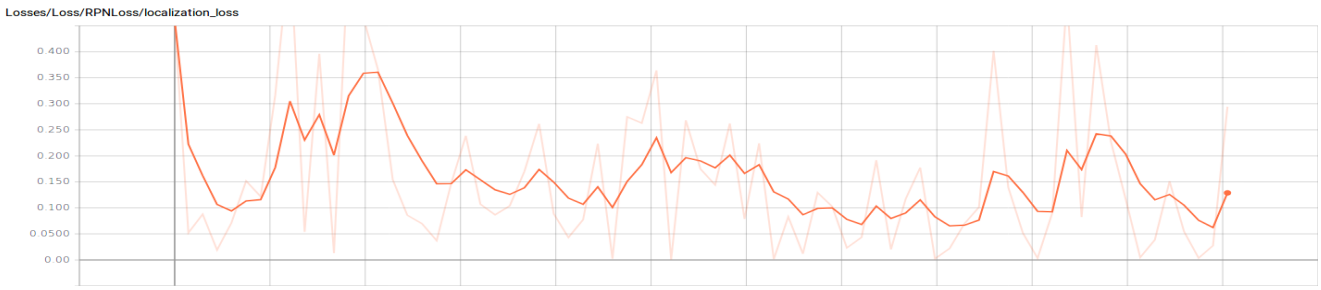

Figure 9. Initial Localization Loss

However, the team was optimistic and hoped adding many more labelled images (~500), as well as using a checkpoint from the Kitti[14] model, would help to improve the model's results. A checkpoint would assist primarily due to not having to train the model from scratch. This is because a model with checkpoints already has weights[15] that can be modified, a weight representing the strength of connections between units, essentially meaning the amount of influence one unit has over another.

[14] Faster-RCNN Kitti: https://github.com/czhu95/kitti
[15] Meaning of Weights: https://stats.stackexchange.com/questions/213325/neural-network-meaning-of-weights

As can be seen in Figure 10, the team was successful in detecting horizontal and vertical parking lots, meaning that if the parking lot sits in line with the horizontal or vertical axis relative to the image. In the figure below all of the parking lots outputted for horizontal parking lots were displayed with a high confidence and provided the correct output. If the bottom right corner of the image is examined the issue with detecting off axis parking lots can be seen, however, a bounding box is still generated to capture its location. This figure demonstrates the success of training our model as false positives are generated, and a bounding box is generated for all of the parking lots present in the figure.

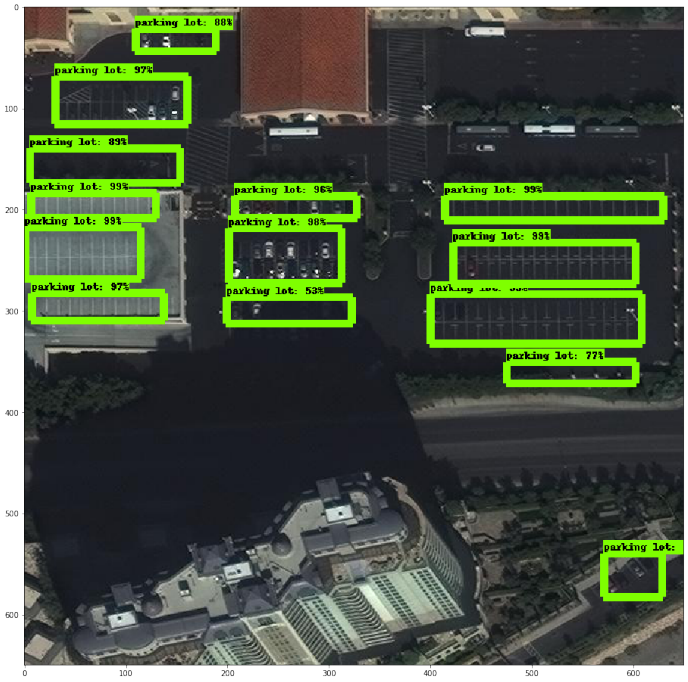

Figure 10. Final Detection Parking Lots

For rounded / angled parking lots, meaning that the parking lots sit at an angle, the team found some lots were detected fairly well, as can be seen in Figure 11. In this figure a red circle has been added, to show the location of a semicircular parking lot. Here it can be seen that although there is angulation in the structure that the model is able to capture most of the parking lot. This demonstrates that while angular parking lots provide a challenge to the model a general localization of these parking lots is still possible. Additionally, the other parking lots in the image are located with a very high certainty and in the correct location.

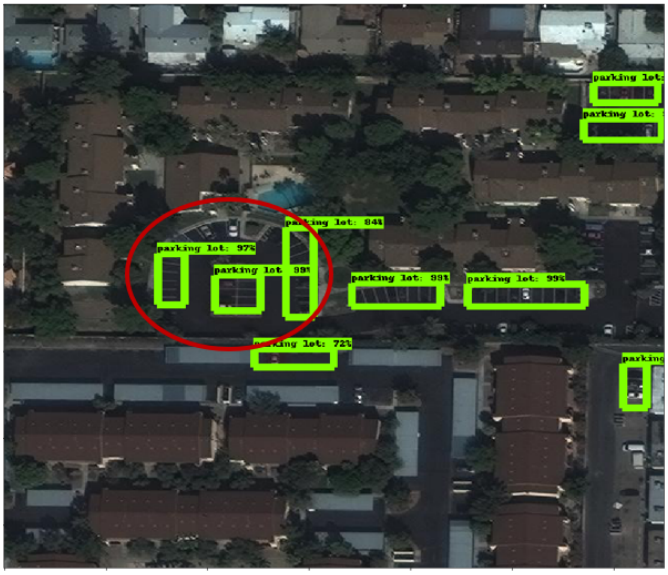

Figure 11. Final Detection Rotated Parking Lots

Figures 12 – 14 show the loss over iterations through the neural network. On the last training run the team saw a decrease in the localization loss which can be seen in Figure 13. This is the deviation from the location of parking lots in the testing set in comparison to those generated by the model. This was a large improvement from the inconsistent change over time that was seen during previous iterations. For the purpose of parking lot detection, the accuracy at which the outputted boxes are able to resemble the true bounding boxes is extremely important, and therefore this loss presents the most important aspect. Additionally, the faded lines in Figure 13, which represent the unsmoothed loss, begin to peak at .1 which suggests this is the worst-case scenario. This means that a loss of .1 is a high bound which is a satisfactory loss.

In Figure 13 it can be seen that the classification loss follows an unusual path, before beginning to converge to around .1. The reason for this behavior is due to the use of a starting model with pretrained weights. This meant that at the early iterations the loss of classifying parking lots was already a lot lower than a model with no preset weights, and as the weights adjusted to classify parking lots. Although this is the case it ultimately began to converge.

The total loss, depicted in Figure 12, which is a combination of localization and classification loss also fell to an acceptable number. Generally, a total loss below 1 suggests that the model is finding some accurate results, and the convergence to .5 produced effective results. Although an improvement would still be beneficial the team predicts the main cause of not having a lower final loss was the angulation of parking lots. By manually observing the outputs of our testing set we predict that if only horizontal and vertical parking lots were present that the loss would reach a very low bound.

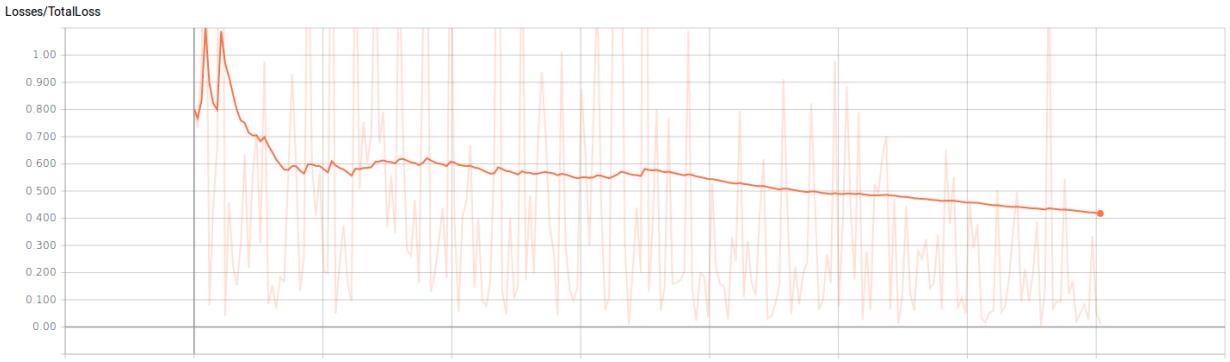

Figure 12. Final Total Loss

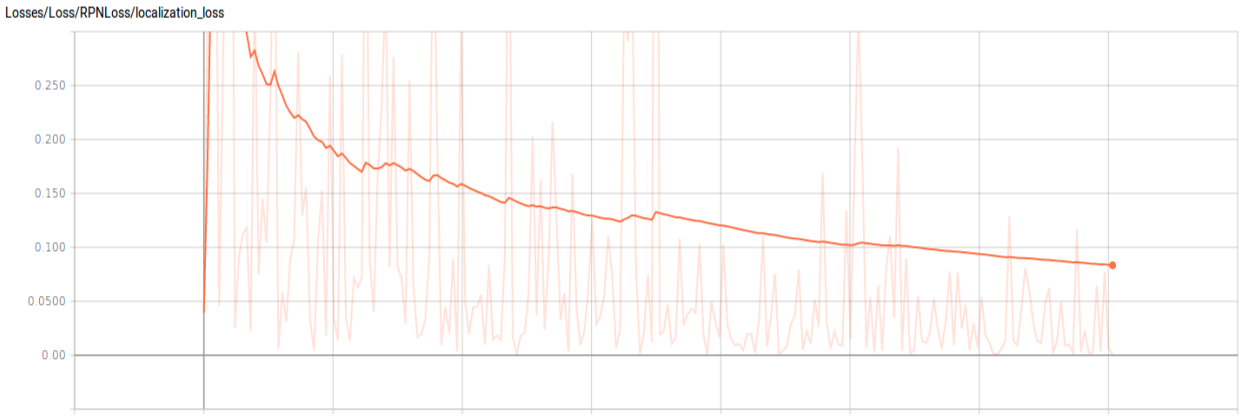

Figure 13. Final Localization Loss

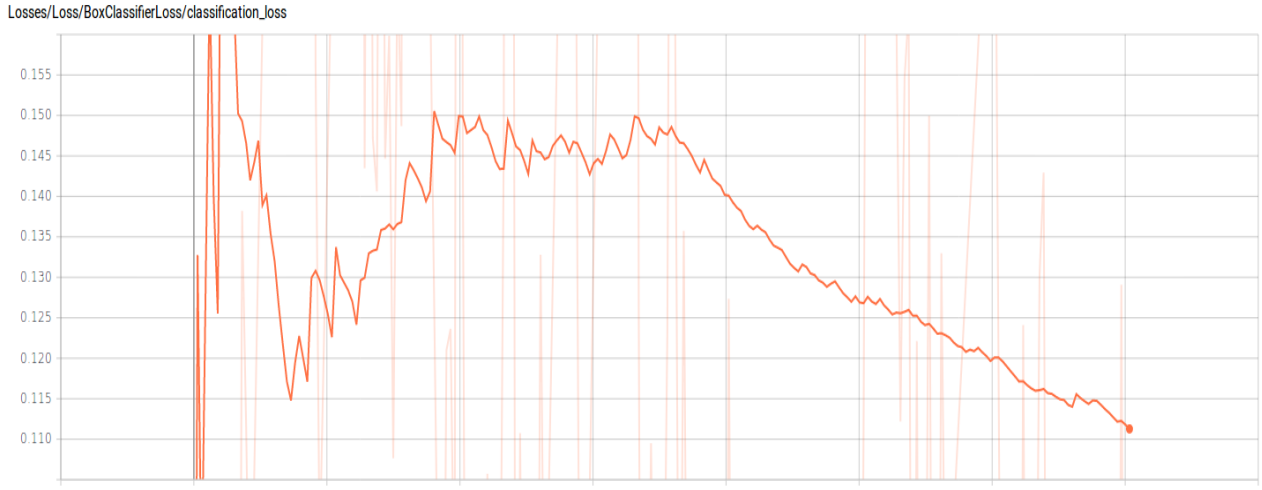

Figure 14. Final Classification Loss

VII. User's Manual

1. Use Cases

A user of the system produced by this project would need some technical skill with interacting with command-line applications as well as installing dependencies for software tools. The system takes a single image or series of images in the GeoTIFF file format, and will process these images and identify parking lots within those images by outputting a GeoJSON file for each image in which a parking lot was identified. A user should note that the system defines a parking lot as a sequence of adjacent parking spaces, as opposed to the entire paved lot.

The system will produce GeoJSON files that can then be parsed be any application that is capable of accepting GeoJSON files. The system is only focused on producing data, and thus will have minimal user interaction. This is due to the way internal details are handled by the system. Little configuration can be done by the user and so the user can (or should) only interact with a single script, referred to as the master script. This script handles accepting input and passing that input to the other parts of the system, as well as returning the output of those parts of the system.

As such, the main use case for this final script is detecting parking lots automatically in satellite images. The user may wish to use these detected parking lots for whatever they desire, such as analyzing which certain geographic areas contain the most parking lots. Another use case for this tool could be for mapping purposes and keeping them updated, as a user could run this script over areas that were known to not contain parking lots beforehand and now do.

Users can also use this project as a starting point to further develop it into a more refined system. The team has included some ideas as to what could be done to improve the accuracy of the model. The team was not able to make these ideas work due to time and infrastructure constraints. However, there are files and documentation included on how to set up the improved Faster-RCNN Kitti model. The user can use all of these resources to make a model that is more likely to accurately detect parking lots.

Finally, the model could be used to build separate models that are used for similar tasks. The team has provided documentation on how the model was built, and as such could be used in the same way for detecting objects other than parking lots. The checkpoints, which are the model weights generated from training the neural network to identify parking lots are located in the training directory of the supporting files. This collection of model weights from the trained model can be used as a starting point in order to train another network with a similar purpose. Starting a new training run with these weighs will allow for a much faster convergence to a low loss on a new dataset, and therefore improve accuracy and reduce the amount of training data needed.

2. Tutorial

A. Install Tensorflow - while the Tensorflow website provides an installation guide, this is what worked for this project.

For Linux / macOS:

1. Install TensorFlow by invoking **one** of the following commands:
   $ **pip install tensorflow** # Python 2.7; CPU support (no GPU support)
   $ **pip3 install tensorflow** # Python 3.n; CPU support (no GPU support)
   $ **pip install tensorflow-gpu** # Python 2.7; GPU support
   $ **pip3 install tensorflow-gpu** # Python 3.n; GPU support
2. After installing tensorflow, validate your installation by running a short code snippet.

For Windows (The team used Anaconda for installation):

1. Create an Anaconda environment named ‘tensorflow’ by invoking the following command:
   C:> **conda create -n tensorflow pip python=3.5**
2. Activate the Anaconda environment by issuing the following command:
   C:> **activate tensorflow**
   (tensorflow)C:> # The prompt should change
3. Issue the appropriate command to install TensorFlow inside the Anaconda environment.
   CPU-only version of Tensorflow:
   (tensorflow)C:> **pip install --ignore-installed --upgrade tensorflow**
   GPU version of Tensorflow:
   (tensorflow)C:> **pip install --ignore-installed --upgrade tensorflow-gpu**
4. After installing tensorflow, validate your installation by running a short code snippet.

Validating installation:

1. Navigate to the Tensorflow install directory.
2. Invoke Python from the shell by typing in: **python**
3. Type the following inside the shell:

```
# Python
import tensorflow as tf
hello = tf.constant('Hello, TensorFlow!')
sess = tf.Session()
print(sess.run(hello))
```

   The output should be: Hello, TensorFlow!

B. Install the necessary dependencies.

C. Run the model.

1. Move parkinglot_finder.ipynb to models/research/object detection.
2. Move parking_lot_inference_graph.tar.gz to models/research/object_detection and extract.
3. Adjust variables within the parkinglot_finder.ipynb notebook to represent the correct paths to necessary files on your local machine.

Figure 15 describes the variables necessary to edit (note discussion in Section IX, Future Work), where the red boxes indict the import areas of the Notebook to update when identifying parking lots:

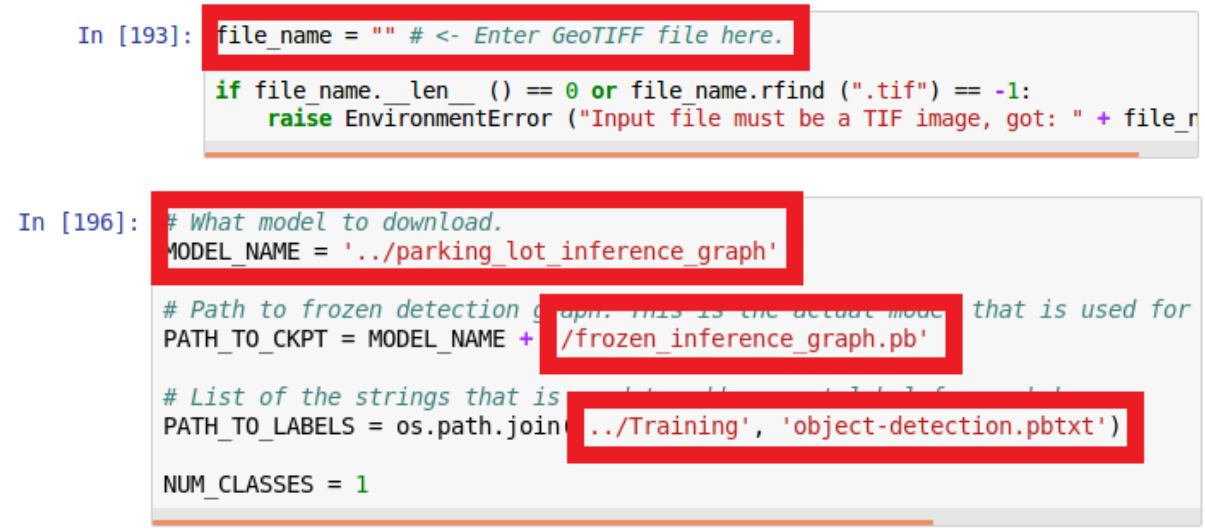

Figure 15. Running the Script

After running the notebook, a GeoJSON file should be produced. There are several ways to open this GeoJSON. The user can open QGIS with the GeoTIFF file that was entered, and then open the GeoJSON file as well, which shows the parking lots outlined on top of the image. Or, the user could go to a GeoJSON file viewer online[16] and load the files in there. Figure 16 is an example of what inputting a GeoTIFF might lead to when loaded in QGIS:

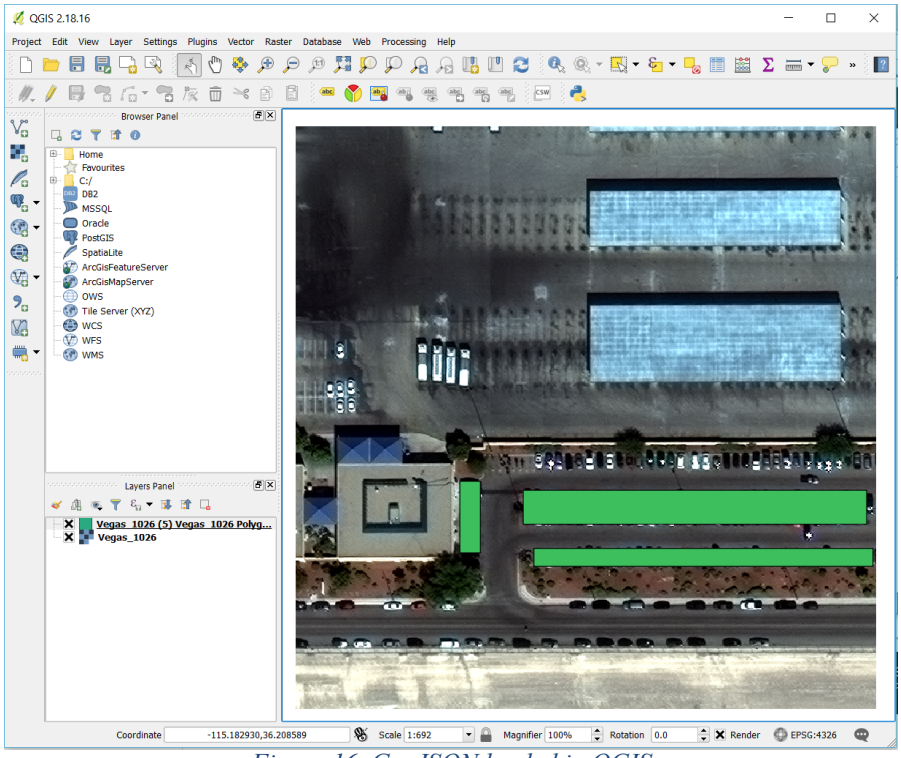

Figure 16. GeoJSON loaded in QGIS

[16] GeoJSON Map Viewer: http://geojsonviewer.nsspot.net

The same GeoJSON loaded with a map viewer[17] verifies that the output is correct in terms of its geographic coordinates, as can be seen in Figure 17. This figure provides a proof of concept that the outputted GeoJSON file of the network can be correctly mapped to the positions identified in Figure 16. In Figure 17, the faint black boxes on the bottom section of the image, are in the same area as the green boxes represented in Figure 16, which provides the desired output.

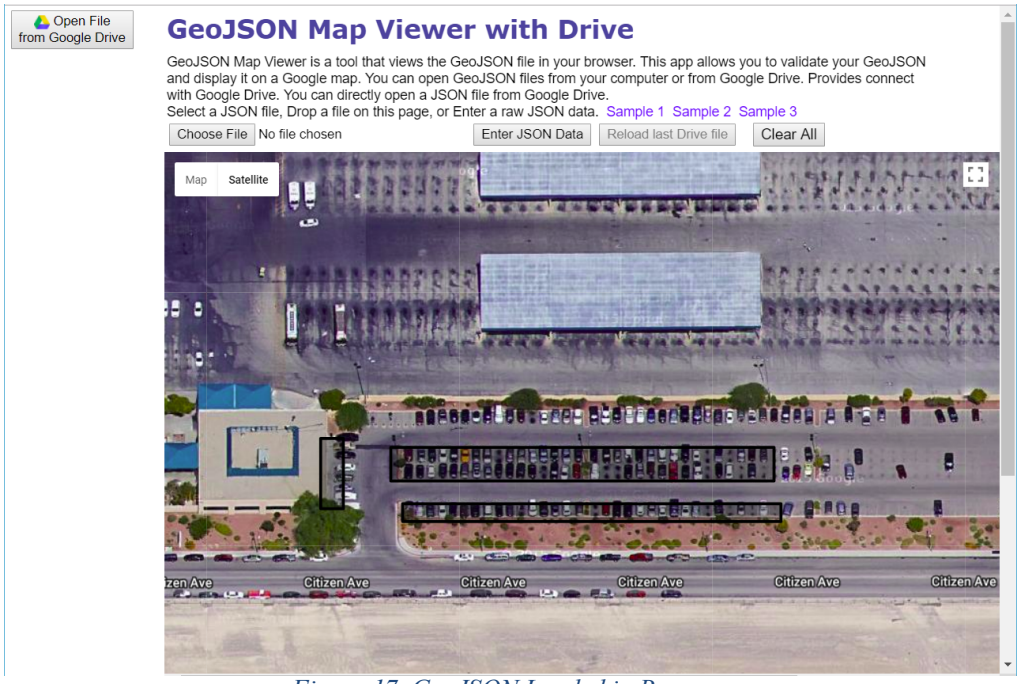

Figure 17. GeoJSON Loaded in Browser

[17] GeoJSON Map Viewer: http://geojsonviewer.nsspot.net

VIII. Developer's Manual

This section of the report serves to provide information that would be helpful to any developer that wishes to extend or modify the system to suit their needs. Additionally, the project structure that was used as well as the flow of project development will be given.

1. Project Structure and Flow

Due to the nature of geographic identification, the system was comprised of two phases: data labeling and classification, and training and assessment of the identification algorithm. In order to accomplish the project's ultimate goal, the entire project team focused on identifying parking lots within images, and labeling those parking lots. This was a team-wide effort so that a sizable introductory training set would exist for the identification algorithm to accept and learn from.

After the creation of the initial training set, the identification algorithm was created using Tensorflow to accept a set of one or more images. Currently, a master script is used to execute the system. This master script executes a couple of conversion scripts, primarily due to the effect that limitations of time had on the algorithm requiring an input of JPEG image files. The first conversion script takes an optional input GeoTIFF image file and a required file output flag. If no image file was given as input, then the script will execute on all GeoTIFF image files found in the current resident directory and any subdirectories of that resident directory. In the case of an image file as input, the script will only execute on that specific file. The required file output flag is used to specify the file type that the script should convert GeoTIFF files to (using GDAL). In the instance of the master script, an output of JPEG is assumed. The JPEG files are then labeled using labelImg. The team used the bounding boxes provided by labelImg to identify parking lots. The labeled images and the negatives (images that contains no parking lot) are then fed into the model for training. After the training phase is complete - a satisfiable accuracy is found with the current model - the project team ran images through the trained algorithm, grabbed the output array, and converted it into GeoJSON file using a Python script.

Moving forward, this project may have to be modified based on the needs of the entities that plan on using and / or expanding the system. If it is used simply for parking lot identification and the user is happy with the final level of accuracy, then no changes may be necessary. However, if the user is not happy with the final level of accuracy, then several options are possible. The user could attempt to retrain the algorithm such that it will have a larger dataset to learn from, which would entail labeling more images than the algorithm was originally trained with. In addition, or as a substitute, the user could attempt to modify the algorithm itself to ensure that it is as accurate as it possibly can be with the current dataset that it was trained on. It should be noted that as the project currently stands, no maintenance is required for the system to report the parking lots found in the images it is supplied with. Modifications, and maintenance for those modifications, are only required if the user wants / needs higher accuracies in reporting the locations of parking lots.

2. Inventory of Project Files

The project files used in the creation of the system as it currently stands are detailed below:
Parking Lot Identification Algorithm Files:

- convert.py: Python script to convert the GeoTIFF files into JPEG for the faster R-CNN model. This script will scour the current directory for files with a TIF extension and convert them to JPEG using GDAL. The converted files will keep the same name but

with .jpg extension instead of .tif. This conversion is necessary because GeoTIFF contains geographical coordinates and which wouldn't work with our labeling program. The script can also convert a single image if desired.

- to_geojson.py: Python script to convert the output XML into GeoJSON containing the geographical coordinates of each parking lot.
- xml_to_rbox.py (future use): This script converts the XML from roLabelImg to Rbox format for the DRBox Caffe model. This model only uses Rbox so this conversion is necessary.
- parkinglot_finder.ipynb: The main script that runs the project. It's a Jupyter notebook that runs the model and outputs a GeoJSON file that contains the geographical coordinates of the parking lots.
- xml_to_csv.py: Script to convert the XML file to a CSV file. The CSV is used to train the model.
- splittraintest.py: Script to split the CSV file of all labels into two CSV files, one for training and one for testing.
- generate_tfrecord.py: Python script to generate the training and testing data for the model.
- Training: Contains all the necessary files for the training process. This includes different checkpoints that the team used to speed up the training process as well as data for the model.
- Data: Contains all information in the format that Tensorflow requires.

Labeled Satellite Image Files:

- Training_Data/GeoTIFF: ~1400 GeoTIFF files that were initially pulled from SpaceNet. These images are from the cities of Las Vegas and Paris.
- Training_Data/Labeled_JPEG: ~500 labeled images using labelImg for the Faster R-CNN model training.
- Training_Data/Negative_JPEG: ~600 negatives for Faster R-CNN model training. These images do not contain any parking lots in them.
- Training_Data/Labeled_roLabel: 76 labeled images using roLabelImg for the Caffe model training. These images are from the Las Vegas image pool.

3. Tutorial on Installing Software

A. Training model

Install CUDA and cuDNN

- CUDA Installation Guide: https://docs.nvidia.com/cuda/cuda-installation-guide-linux/
- CuDNN Installation Guide: https://docs.nvidia.com/deeplearning/sdk/cudnn-install/

Install Tensorflow

- Tensorflow Installation Instructions: https://www.tensorflow.org/install/

Install additional Python dependencies:

- Install pillow, lxml, jupyter, matplotlib

Retrieve and make Tensorflow Models:

- Git clone the following repository: https://github.com/tensorflow/models
- If using Ubuntu navigate to models/research and execute command: protoc object_detection/protos/*.proto --python_out=. And export PYTHONPATH=$PYTHONPATH:`pwd`:`pwd`/slim

Retrieve the project code:

- Clone the following repository: https://github.com/khoale95/SatImageParkingLotFinder

B. Labelling

Follow the steps that were detailed in the implementation section (Section V) for labelling data. The future work section details labelling rotated bounding boxes for use in the DRBox model.

- labelImg: Download and use LabelImg at https://github.com/tzutalin/labelImg
- roLabelImg: Download and use roLabelImg at https://github.com/cgvict/roLabelImg . For roLabelImg, the user will want to always use the rotating bounding box as the normal bounding boxes does not store an angle in the output XML file.

4. Training on New Data

In order to train the object detection once information has been labeled using labelImg, a few steps must be taken.

A. Data Manipulation

- Put labelImg XML files into the Bounding_Boxes folder.
- Put the JPEG files into the Images folder
- Run xml_to_csv.py
- Run splittraintest.py with the arguments of the CSV file name and ratio of training to testing. For example 'python splittraintest.py data/parkinglot_labels.csv 0.8' will create a test and train split of the overall CSV file with 80% of the training images being part of the training set.
- Create TFRecords for use by the neural network:
    - Run: python generate_tfrecord.py --csv_input=data/train_labels.csv --output_path=train.record
    - Run: python generate_tfrecord.py --csv_input=data/test_labels.csv --output_path=test.record
- The information is now formatted in the correct way to be passed to the neural network.

B. Model Setup

- Locate a model: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md
- Add the .config file to the training directory. The project team used faster_rcnn_resnet101_kitti.config. The model selected can differ based on training needs.
- Once a config file has been downloaded, the user will need to adjust the necessary paths and parameters to the data. Reference faster_rcnn_resnet101_kitti.config in the repository as an example.
- If the user is so inclined, the user can also acquire model checkpoints. To use these, download the TAR file and extract them in Tensorflow's model/research/object_detection directory.

- In the training directory create an object-detection.pbtxt file. It should look something like:

```
item {
        id: 1
        name: 'parking lot'
}
```

C. Running Model

- All setup has now been completed and training can begin.
- To train, navigate to models/research/object_detection and run: python train.py --logtostderr --train_dir=training/ --pipeline_config_path=training/faster_rcnn_resnet101_kitti.config
- To monitor the progress, run tensorboard --logdir='training' from models/object_detection
- **NOTE:** The directory training is the one provided in the project team's repository; the user path to get there may have to have precursors. For example: /home/user/Documents/SatImageParkingLotFinder/training

IX. Lessons Learned

The timeline below details the required goals for this project to be considered complete, and the stretch goals if the required goals are able to be completed, as well as the expected date for those goals to be completed by.

1. Timeline

Required Goals:

- Feb. 8 - Documenting the project plan and goals
- Feb. 22 - Label data, develop the training set for parking lot identification
- Mar. 8 - Create / adapt a learning algorithm for accepting the data
- Mar. 15 - Training the algorithm for parking lot identification
- Mar. 22 - Testing the algorithm and accessing accuracy
- Mar. 29 - Finalize the algorithm for parking lot identification

Stretch Goals:

- April 12 - Label the training set for parking space identification
- April 19 - Create / adapt a learning algorithm for accepting the data
- April 26 - Training the algorithm for parking space identification, adjusting the algorithm
- May 1 - Finalize presentation and deliver final product

Unfortunately, due to several complications along the way, the project team was unable to meet the project's stretch goals. However, the team felt very satisfied with the final model for the required goals, and feel identifying parking spaces (stretch goals) can be easily accomplished with the work that was completed.

2. Problems

1. Downloading the SpaceNet data proved to be surprisingly difficult. The SpaceNet dataset was hosted on a S3 bucket that could only be accessed through command line through the aws-cli commands. Therefore, the team was unable to pick and choose which files were wanted, and were forced to either download images one at a time or in large 8GB+ .tar.gz files. Unfortunately, these files provided a lot of data that was unnecessary in the scope of the project. In addition, using the aws-cli would timeout after several minutes, thus not allowing the team to download the entire .tar.gz files.
2. The visibility of the parking lots during the labeling phase caused a small debate in the team. It had to be decided which parking lots were visible enough for the algorithm to reliably identify. The baseline was if it looks like a parking lot - large open area with rectangular spaces marked with white lines - then it would be a labeled. However in the satellite images that were downloaded, some parking lots were obscured to the point that while a human can tell that it is a parking lot, a machine might mix it up with something else. Examples of this are parking lots with canopies over them, parking lots obscured by building shadow or at night, parking lots with trees over them, etc.
3. The model proved difficult to get set up. The project team first attempted to use Caffe and Faster-RCNN which proved to be more pain than it was worth. The team then moved to the Tensorflow framework using Faster-RCNN-RESNET-101-COCO, starting from scratch. This ran, but had bad overall loss because it wasn't given any negatives. The

second attempt with Tensorflow was one where negatives were given, which decreased the overall loss but gave many false positives.

4. In addition to the trouble that the model set up gave the team, the team had trouble getting the model to accept and train from GeoJSON files. This meant that attempting to produce trainable data with QGIS was not worthwhile.
5. Realizing that bounding boxes could not be rotated with the model, the team wanted to build a model that had this ability. The team found a model called DRBox from a recent research paper that appeared to be exactly what was needed, but had several difficulties running this model.
6. Setting up roLabelImg on PC using Anaconda and Python 2.7. The Python libraries to set up roLabelImg on PC don't set up in the proper directory. This causes pathing issues when trying to run the program. The library that causes the issue is PyQT4.

3. Solutions

1. In order to download the appropriate dataset, after much trial and error, the team had to download the .tar.gz file in parts. It was found that the team could download about 3GB of data before getting timed out, therefore the team downloaded three parts of the .tar.gz file, and combined them into one. First, the team used a byte range: `aws s3api get-object --bucket spacenet-dataset --key AOI_2_Vegas/AOI_2_Vegas_Test_Public.tar.gz --request-payer requester AOI_2_Vegas_Test_Public.tar.gz --range bytes=3000000000-6000000000`. Next, after downloading three separate parts of the file, the team combined them using: `cat AOI_2_Vegas_Test_Public_2.tar.gz >> AOI_2_Vegas_Test_Public_all_2.tar.gz`. After combining the three parts, the team was able to finally successfully extract the file and retrieve the data.
2. After debating amongst the team and consulting with RGI, it was decided to label parking lots that the team can still make out an outline. So even if it is in the dark, or has some debris obscuring small part of it, the team would still include it. The team would, however, exclude parking lots that are completely obscure like those with canopies or if it is in complete darkness such that the white lines are not visible.
3. The final model uses Faster-RCNN-RESNET-101-Kitti which gave the system better execution speed, a low number of false positives, and overall significantly much better results.
4. This allowed the team to pursue easier techniques of labeling trainable data, such as using labelImg. However, labelImg cannot open GeoTIFF image files, so to get around this the team needed a script that could accept a GeoTIFF image and convert it to a JPEG image that the team can then label.
5. First, the ran into several errors regarding dependencies in order to run this model. After much troubleshooting, all dependency errors were removed, yet the team still had errors when training. It was discovered that the model DRBox[18] was built upon (VGGNet) and required over 11 GB of GPU memory. Because the team was trying to run this on a personal laptop, the team did not have the resources to run it, so the team looked into running this model on Virginia Tech's high powered computers. After a consultation with the Advanced Research Computing division, the team was told running this model would be quite challenging given the current setup of the machines. Due to this and the limited

[18] DRBox: https://github.com/liulei01/DRBox

amount of time that the team had, the team was unable to train DRBox, however, the team documented how to run this model for users wishing to have improved results. Another potential fix to this is that the team can rotate the images with a Python script using GDAL. The images will be rotated by set degrees (like 30, 45, 60, ...) and run through the model. The script will then merge the output of these multiple rotations to hopefully provide a more accurate model for parking lots that are not horizontal or vertical. The team was able to merge this because the team knows the angle of rotation so the team can translate the point of the rotated images into its actual point in the real image. This approach is a bit ad hoc, but it is also faster than switching the model. What the team ended up doing was just trying to exclude irrelevant objects from the bounding boxes. This meant that for large / long parking lots that are tilted, the team used smaller bounding boxes that encompass only a small section of the parking lot. There are limitations to this approach as the shape of parking lots are very varied and sometimes the team had to include more irrelevant data than what was wanted. This causes the model to miss many of the tilted parking lots but it does catch some, from time to time.

6. Instead of just running 'conda install pyqt=4' on PC, run 'conda install -c anaconda pyqt=4'. This will cause PyQT4 to install into the correct directory of Anaconda.

4. Future Work

One of the most obvious limitations of the model is that it has difficulty identifying parking lots that are at an angle (i.e., a parking lot that is not horizontal or vertical with respect to the source image dimensions). This is primarily due to the difficulty that was encountered when attempting to train the model using bounding boxes that stored the angle of rotation. The project team has potentially solved this by using roLabelImg and the new Caffe model. However, the team could not get the model to work as the team lacked the necessary hardware. The model requires a minimum of 11GB of video RAM which the team did not have. So for anyone who wanted to improve the model and has the necessary hardware, this would be a good place to start.

Another feature that would be useful is modifying the way the tool accepts input. Currently the tool is a Jupyter notebook in which the user must modify a couple of lines of code to adjust the input to the model. This is not particularly user-friendly, and so it would be beneficial to adjust this input to either be stored in a config file or interactive input (i.e. command line or user interface).

The model that can accept rotated bounding boxes is called DRBox and is based on a research paper titled "Learning a Rotation Invariant Detector with Rotatable Bounding Box."[19]. Follow the instructions for installation and preparation on the repository. It is important to note this model requires 11GB of GPU memory, so it must be set up on a powerful computer. Next, the user must label satellite images with rotated bounding boxes. Follow the instructions in the developer manual (Section VII) for installing roLabelImg. This tool can be used to label parking lots in the same manner as labelImg, with the addition of rotation.

The DRBox uses its own format for labelled images (.rbox) unlike the project model which uses XML files. As such, the labelled images the team has provided must be re-labelled using roLabelImg. To label the rotated parking lots, use roLabelImg on a JPEG file and create a RotatedRbox with "parking lot" as the label. It is important that even for parking lots that are

[19] Learning a Rotation Invariant Detector with Rotatable Bounding Box: https://arxiv.org/abs/1711.09405

vertical or horizontal (no rotation), one still labels using a RotatedRbox rather than a RectBox so that the conversion script works.

Figure 18 gives an example of how a user would label parking lots in roLabelImg, so that angled parking lots can be handled:

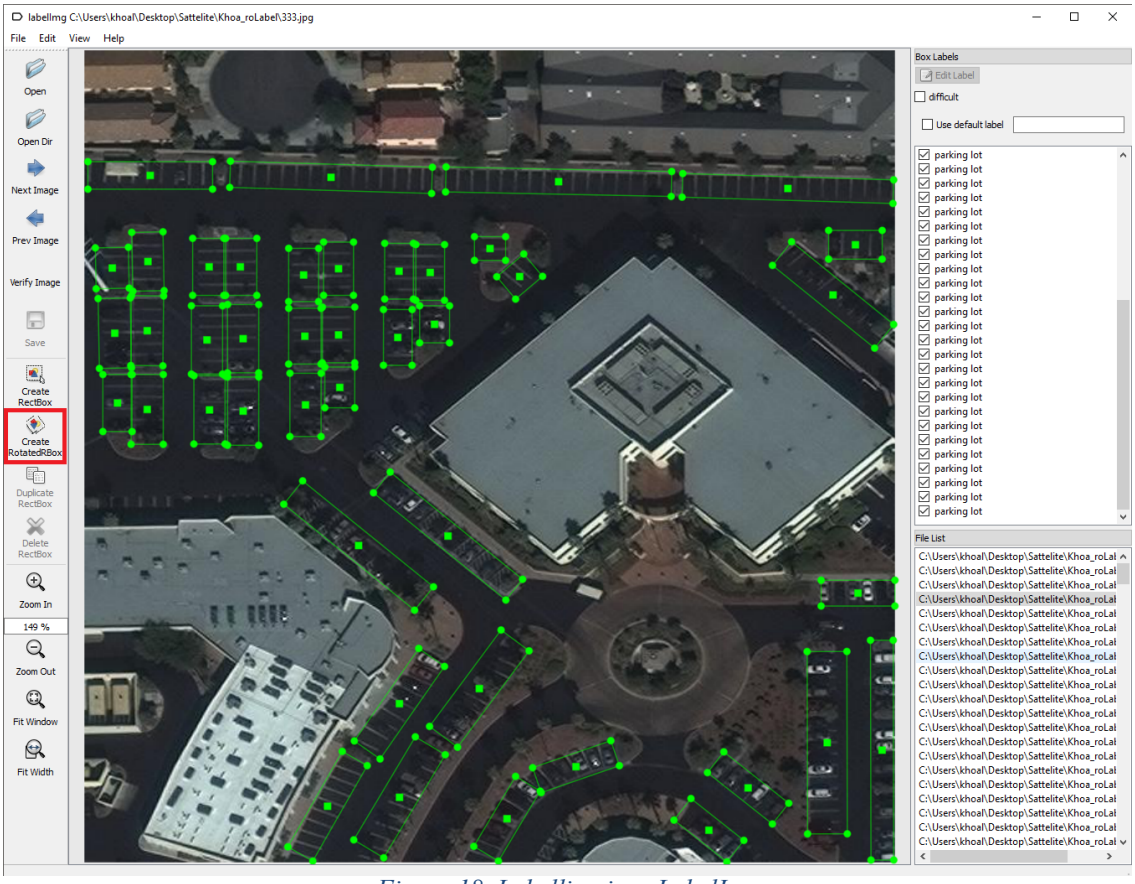

Figure 18. Labelling in roLabelImg

After labelling and saving the XML for the rotated bounding boxes, use the xml_to_rbox.py script that has been provided to convert the files into a usable format for the model. This Python script requires Python 3.X in order to run. This script can be run in a directory by simply calling the following command in the directory of your choice: `python3 xml_to_rbox.py`
If you wish to run this script on a single XML file, run: `python3 xml_to_rbox.py foo.xml`.

With all of the labelled images in the correct directory, follow the directions in the DRBox repository for training. The training should take some time, but after it has completed, follow the View Results section to see the output of the newly trained model. The team believes this improved model should provide significantly better results compared to the current trained model.

X. Acknowledgements

This project was supported by Mary Carome, Steven Lander, and Michael Szaszy. We would like to thank them as well as everyone at Reinventing Geospatial Inc. for all of their help and guidance. We would also like to thank Dr. Fox for his assistance as well as directing the course that allowed us to work on this project.

Client: Reinventing Geospatial Inc.

- Mary Carome: mary.carome@rgi-corp.com
- Steven Lander: steven.lander@rgi-corp.com
- Michael Szaszy: michael.szaszy@rgi-corp.com

Instructor: Dr. Edward A. Fox - fox@vt.edu
Graduate Teaching Assistant: Yilong Jin - e1337@vt.edu

XI. References

1. P. Hagerty. Establishing a Machine Learning Workflow. Sept. 2016. Web. https://medium.com/the-downling/establishing-a-machine-learning-workflow-530628cfe67. Accessed 26 Mar. 2018.
2. A. Van Etten. Getting Started With SpaceNet Data. Jan. 2017. Web. https://medium.com/the-downling/getting-started-with-spacenet-data-827fd2ec9f53. Accessed 26 Mar. 2018.
3. Machinalis. Establishing a Machine Learning Workflow. Mar. 2016. Web. http://www.machinalis.com/blog/obia/. Accessed 26 Mar. 2018.
4. Ujjwalkarn. Python Data Science Tutorials. Jul. 2017. Web. https://github.com/ujjwalkarn/DataSciencePython. Accessed 26 Mar. 2018.
5. T. Bai, D. Li, K. Sun, Y. Chen, and W. Li, "Cloud Detection for High-Resolution Satellite Imagery Using Machine Learning and Multi-Feature Fusion," *Remote Sensing*, vol. 8, no. 9, p. 715, Aug. 2016 [Online]. Available: http://dx.doi.org/10.3390/rs8090715
6. J. Xu. Deep Learning for Object Detection: A Comprehensive Review. Sept. 2017. Web. https://towardsdatascience.com/deep-learning-for-object-detection-a-comprehensive-review-73930816d8d9. Accessed 26 Mar. 2018.
7. S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks" *CoRR*, Jan. 2016. Accessed on: May. 1, 2018. [Online]. Available: https://arxiv.org/abs/1506.01497
8. L. Liu, Z. Pan, B. Lei, "Learning a Rotation Invariant Detector with Rotatable Bounding Box" *CoRR*, Nov. 2017. Accessed on: May. 1, 2018. [Online]. Available: https://arxiv.org/abs/1711.09405
9. TensorFlow. *TensorFlow*. 2018. Web. https://www.tensorflow.org/ Accessed 1 May 2018.
10. Tzutalin. *LabelImg*. 2018. Web. https://github.com/tzutalin/labelImg Accessed 1 May 2018.
11. cgvict. *roLabelImg*. 2018. Web. https://github.com/cgvict/roLabelImg Accessed 1 May 2018.
12. TensorFlow. Tensorflow detection model zoo. 2018. Web. https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md. Accessed 1 May 2018.

XII. Appendices

1. Project Team Members and Responsibilities

Team Member	Role	Responsibilities
Thomas Wolfe	Project Lead Machine Learning Lead	● Maintain contact between client, group, and professor ● Develop and train ML algorithm
Alex Lambrides	GIS Lead	● Research GIS labelling techniques ● Acquire data ● Label GeoTIFFs ● Assist in developing model
Khoa Le	Presentation Lead	● Lead presentation ● Label GeoTIFFs ● Assist in developing model
Patrick Jahnig	Report Lead Programming Lead	● Lead interim and final report ● Label GeoTIFFs ● Wrote conversion scripts

Table 1. Team Member Responsibilities