Robust Road Topology Extraction from Aerial Imagery

by

Favyen Bastani

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of
Master of Science
at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2018

© Massachusetts Institute of Technology 2018. All rights reserved.

Signature redacted

Author ............. ........................................

Department of Electrical Engineering and Computer Science

May 22, 2018

Signature redacted

Certified by ............. ........................................

Samuel Madden
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Signature redacted

Accepted by ............. ........................................

Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students
Robust Road Topology Extraction from Aerial Imagery

by

Favyen Bastani

Submitted to the Department of Electrical Engineering and Computer Science
on May 22, 2018, in partial fulfillment of the
requirements for the degree of
Master of Science

Abstract

Creating and updating road maps is currently an expensive and often manual process, and thus maps today are outdated or have poor coverage in large regions of the world. Automatically inferring the road network graph from aerial imagery provides a promising avenue to reducing the cost of maintaining road maps, but existing inference methods have poor precision. This thesis develops a novel iterative graph construction process for extracting graph structures from images, and applies this process to automatic road topology inference to significantly reduce error rates.

Thesis Supervisor: Samuel Madden
Title: Professor of Electrical Engineering and Computer Science
Acknowledgments

I am grateful for the constant guidance and encouragement that my advisor, Samuel Madden, as well as Professors Mohammad Alizadeh and Hari Balakrishnan, provided throughout this project. They were a never-ending source of insightful suggestions, and the incredible effort that they all were willing to put in just before deadlines to get everything to a perfect state was amazing. (If not for this effort, the reader would likely find the text of this thesis indecipherable.)

I also highly appreciate Professor David DeWitt’s feedback over much of this project. He continuously asked interesting questions in directions that I hadn’t thought of exploring before, which were crucial to improving the generalizability of this work and to performing a thorough evaluation.

I would also like to thank Songtao He, who I collaborated with and who was the source of many key insights, and other students in the MIT Database Group, who have always been excited to discuss research ideas.

Feedback provided by Dr. Sanjay Chawla and Sofiane Abbar from the Qatar Computing Research Institute (QCRI), and support from QCRI itself, was likewise invaluable.

Finally, I want to thank my family and friends for always being there when I needed encouragement and support.
6 Evaluation

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1 Dataset</td>
<td>49</td>
</tr>
<tr>
<td>6.2 Baselines</td>
<td>50</td>
</tr>
<tr>
<td>6.3 Metrics</td>
<td>51</td>
</tr>
<tr>
<td>6.4 Quantitative Results</td>
<td>53</td>
</tr>
<tr>
<td>6.5 Qualitative Results</td>
<td>56</td>
</tr>
</tbody>
</table>

7 Conclusion

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>61</td>
</tr>
</tbody>
</table>
List of Figures

1-1 Roads in imagery are frequently occluded. ........................................ 14
2-1 Two-stage segmentation architecture. .................................................. 18
3-1 Stages of segmentation post-processing. .............................................. 24
3-2 Refinement heuristics are unable to correct extensive noise. ................. 25
4-1 Exploring a T intersection in the search process. ................................ 29
4-2 Overview of RoadTracer. ........................................................................ 31
4-3 Problems when training on static training examples. ........................... 33
4-4 A naive matching approach. ................................................................. 35
4-5 Angle targets optimization for wide roads. ........................................... 36
4-6 Single starting location is insufficient in weakly connected cities. .......... 38
4-7 Two-phase map inference. .................................................................... 40
5-1 Imagery in Doha lags behind the physical road network. ..................... 44
5-2 Map datasets have poor quality in northern Alberta. .......................... 45
6-1 The TOPO metric. .................................................................................. 51
6-2 Average $F_{\text{correct}}$ and $F_{\text{error}}$ over the 15 test cities. ............... 54
6-3 Tradeoff between error rate and recall. ................................................ 54
6-4 Average TOPO recall and error rate over the test cities. ..................... 55
6-5 $F_{\text{correct}}$ and $F_{\text{error}}$ in each test city. ........................................... 57
6-6 Qualitative comparison of inferred road networks. ............................ 58
List of Tables

6.1 SP performance........................................... 56
Chapter 1

Introduction

Creating and updating road maps is a tedious, expensive, and often manual process today. As a result, road maps are outdated or have poor coverage in many regions of the world. For example, in Doha, maps from commercial vendors consistently lag behind the physical road network state, even after major changes are made to the layout of arterial roads [22]. At the same time, accurate and up-to-date maps are especially important given the growing popularity of location-based mobile services and the impending arrival of autonomous vehicles.

Aerial imagery provides a promising avenue to automatically infer the topology of the road network. In practice, however, extracting maps from aerial images is difficult even when the images have high resolution. For example, Figure 1-1 shows roads that are occluded by tall buildings, shadows, and trees, making it difficult even for humans to infer road connectivity from imagery; prior approaches do not handle these problems well [2, 15, 16]. Almost universally, they begin by segmenting the image, classifying each pixel in the input as either road or non-road [19, 20, 11, 25, 26, 9, 18]. They then implement a complex post-processing pipeline to interpret the segmentation output and extract topological structure to construct a map. In practice, however, noise frequently appears in the segmentation output, making it hard for the post-processing steps to produce an accurate result. This in turn results in very poor precision except in the most ideal conditions.

The fundamental problem with a segmentation-based approach is that the machine
Roads in imagery are frequently occluded. Learning model is trained only to provide local information about the presence of roads. Key decisions on how road segments are inter-connected to each other are delegated to an error-prone post-processing stage that relies on heuristics instead of data-driven learning or principled algorithms. Rather than rely on an intermediate image representation, we seek an approach that produces the road network directly from the model output. However, it is not obvious how to train a model to learn to produce a graph from aerial images.

In this thesis, we develop RoadTracer, an approach that uses a novel iterative graph construction process for extracting graph structures from images. This approach constructs the road network by adding individual road segments one at a time, using a novel CNN architecture to decide on the next segment to add given as input the portion of the network constructed so far. In this way, we eliminate the intermediate image representation of the road network, and avoid the need for extensive post-processing that limits the accuracy of prior methods.

Training the CNN decision function is challenging because the input to the CNN at each step of the search depends on the partial road network generated using the CNN up to that step. We find that standard approaches that use a static set of labeled
training examples are inadequate. Instead, we develop a dynamic labeling approach to produce training examples on the fly as the CNN evolves during training. This procedure resembles reinforcement learning, but we use it in an efficient supervised training procedure.

We evaluate RoadTracer using aerial images covering 24 square km areas of 15 cities, after training the model on 25 other cities. We implement two baselines, DeepRoadMapper [18] and our own segmentation approach. Across the 15 cities, we find that, at a 5% average error rate on a junction-by-junction matching metric, RoadTracer correctly captures 45% more junctions than our segmentation approach (0.58 vs 0.40). DeepRoadMapper fails to produce maps with better than a 19% average error rate. Because accurately capturing the local topology around junctions is crucial for applications like navigation, these results suggest that this work is an important step forward in fully automating map construction from aerial images.

This thesis is based on work originally presented in CVPR as “RoadTracer: Automatic Extraction of Road Networks from Aerial Images” [3].
Chapter 2

Related Work

This chapter discusses prior work related to automatic map inference. In Section 2.1, we summarize approaches that extract road topology from aerial images, and in Section 2.2, we discuss approaches that use GPS trajectories.

2.1 Extracting Road Topology from Aerial Images

Prior work in extracting road topology from aerial images generally uses a two-stage architecture. First, a machine learning model is trained to perform pixel-wise image segmentation, where pixels are classified as either road or non-road. Then, the model output is post-processed to obtain a road network graph. Figure 2-1 illustrates this architecture.

Image segmentation has been an important problem in computer vision for over a decade. Recent approaches heavily rely on convolutional neural networks (CNNs), which have attracted significant interest following their application with impressive results to the MNIST, CIFAR10, and ImageNet image classification datasets [10, 17]. Ronneberger et al. propose a symmetric arrangement of convolutional layers with skip connections to recover fine-grained image details [21]. DeepLab [8] combines residual networks [14] with upsampling convolutional layers and bilinear interpolation to classify pixels at the input resolution, and with conditional random fields to accurately recover object boundaries.
To post-process the segmentation output for road network graph extraction, Cheng et al. apply binary thresholding and morphological thinning to produce single-pixel-width road centerlines [9]. A graph can then be obtained by tracing these centerlines. Cheng et al. employ a cascaded CNN architecture to ensure output centerlines match closely with the aerial imagery, where the output of a sequence of convolutional layers trained for road detection is fed into a second sequence trained for centerline extraction.

DeepRoadMapper proposes additional post-processing after thinning to further refine the inferred road network graph [18]. First, a set of heuristics are applied to refine the graph by removing topological defects introduced by thinning such as branches and loops. Second, a search-based approach attempts to identify potential missing connections in the inferred graph, and a model is trained to select and incorporate the correct missing connections.

2.2 Road Topology from GPS Trajectories

Several map inference approaches propose inferring road topology from GPS trajectories. These approaches typically build on one of three architectures:

Kernel density estimation approaches first build a spatial histogram over observations in the GPS trajectories, dividing the region into a grid and then counting the
number of observations contained in each cell. The histogram is interpreted as a ma-
trix, and smoothed by convolving with a (typically Gaussian) kernel. Various methods
have been proposed to extract road centerlines from the smoothed histogram, such as
Voronoi tessellation [12], grayscale skeletonization [4], and morphological thinning.

K-means approaches cluster GPS observations into groups that share similar at-
tributes including position and heading. The clusters of observations that appear
consecutively in a trajectory are connected to yield a graph where clusters are ver-
tices. This graph is typically refined with post-processing. For example, Kharita
applies techniques from graph spanners to remove redundant edges [22].

Trajectory merging approaches process the GPS data trajectory by trajectory,
yielding a partial graph on each step [7]. Each iteration involves extending the pre-
vious partial graph by merging the current trajectory into the graph.

However, collecting enough GPS data that can cover the entire map is challenging,
especially for regions of the map that are far from the city core. Nevertheless, GPS
trajectories may be useful to improve accuracy in dense urban areas where roads are
not visible from the imagery, to infer road connectivity at complex intersections where
roads are layered, and to enable more frequent map updates.
Chapter 3

Segmentation Approaches

In this chapter, we detail the segmentation framework that prior approaches for extracting road topology from aerial imagery universally apply. First, a machine learning model is trained to perform pixel-wise image segmentation, where pixels are classified as either road or non-road (Section 3.1). Binary thresholding and morphological thinning are applied to produce an initial road network graph (Section 3.2). Then, refinement heuristics improve the topology of the graph (Section 3.3). Finally, we discuss fundamental issues with segmentation-based approaches in Section 3.4.

3.1 Labeling Road Pixels

Recent work train convolutional neural networks (CNNs) to label road pixels in the input aerial imagery [9, 18]. This follows results in other applications of semantic segmentation, where CNNs have significantly improved segmentation performance.

**CNN Architecture.** CNNs for road pixel labeling generally follow a UNet-like structure, where an encoder inputs the high-resolution imagery and outputs features at successively lower resolutions, and a decoder processes the features to output a probability for each pixel indicating how likely the pixel belongs to a road [21].

**Ground Truth Labels.** Ground truth pixel labels may be obtained in two ways. First, the pixels may be hand-labeled from the aerial imagery. In this case, the
labeling process is typically crowdsourced, and labels from multiple crowdworkers are merged to obtain high precision labels. However, even with crowdsourcing, the requirement of manual labeling limits the amount of data that can be labeled with this method. Thus, most approaches increase the dataset size by computing labels based on a road map dataset like OpenStreetMap [24]. If a pixel is within some distance threshold of a road segment edge in the dataset, then the pixel is labeled road; otherwise, it is labeled non-road. If the map dataset includes information about the width or number of lanes of a road, that information could be used to obtain a different distance threshold for each road segment; otherwise, a fixed threshold is used.

**Loss.** CNNs require a differentiable loss function that can be backpropagated to update the model parameters. Most approaches use cross entropy loss. Suppose that $p_{ic}$ is the probability predicted by the CNN for pixel $i$ and class $c$, and $y_{ic}$ is the label probability (either 0 or 1). Then, cross entropy loss is defined as:

$$\sum_{i,c} y_{ic} \log(p_{ic})$$

Soft intersection-over-union loss may be used instead. This loss is more closely related to the intersection-over-union metric that is often used to score segmentation results, and tends to produce outputs where most pixels are labeled with a very low or very high probability. Soft intersection-over-union loss is defined as:

$$\sum_c \frac{\sum_i y_{ic} p_{ic}}{\sum_i y_{ic} + p_{ic} - y_{ic} p_{ic}}$$

**Joint Training.** The CNN is occasionally trained jointly for building segmentation, as in DeepRoadMapper [18]. Here, instead of two classes, there are three: road, building, and background. Although joint training generally improves performance, most approaches only use road and non-road classes due to the imprecision and poor coverage of building data in map datasets.
3.2 Graph Extraction

Segmentation produces a continuous output in an image representation, where each pixel is labeled with a probability of being road or non-road. However, the objective of map inference is to produce a discrete road network graph structure. To perform this conversion, we need to extract centerlines from the image representation.

First, we apply thresholding to convert the CNN output to a binary mask. In the binary mask, roads may be represented by lines with a multiple-pixel width. Thus, we apply morphological thinning \[27\] to thin these lines down to a single-pixel width. This is achieved by iteratively removing pixels on the border between background and foreground in the binary mask; pixels that are surrounded by background on both sides are retained.

Thinning yields a graph, where every foreground pixel is a vertex, and edges connect adjacent foreground pixels. We apply the Douglas-Peucker method \[13\] to simplify the graph. This step removes unnecessary vertices; for example, if a vertex has two neighbors and is close to the midpoint of its neighbors, then we approximate the vertex by removing it and connecting its neighbors.

Figure 3-1(a) shows an example segmentation output. In Figure 3-1(b), a graph is extracted through thresholding, morphological thinning, and the Douglas-Peucker method.

3.3 Refinement Heuristics

Because the CNN is trained with a loss function evaluated independently on each pixel, it will yield a noisy output in regions where it has a low confidence on the presence and absence of roads. As shown in Figure 3-1(a) and (b), noise in the segmentation output will be reflected in the extracted graph. Refinement heuristics are applied to remove basic types of noise.

First, because thinning operates on a local level, by iteratively examining a pixel and its neighbors, it often produces short dangling segments off of the road centerline.
We can prune these artifacts by removing short dead-end segments [18]. A segment is a dead-end segment if one of its endpoints has only one incident edge.

Second, binary thresholding sometimes leaves gaps in the middle of roads. For example, a road partially occluded by a shadow may be split in two following graph extraction. To correct these gaps, we extend dead-end segments if there is a road close to the dead-end.

Third, intersections and other junctions may be split into multiple vertices. If there are two nearby vertices that each have at least three incident edges, then we merge the vertices.

Figure 3-1(c) shows the graph after applying these three refinement heuristics.

3.4 Segmentation Issues

As we showed in Figure 3-1, refinement heuristics are able to correct for basic types of noise in the segmentation output. But, these heuristics nevertheless frequently fail to correct for more complex errors introduced by thresholding and thinning, especially
Figure 3-2: Refinement heuristics are unable to correct extensive noise.

when the topology is unclear from the aerial image due to occlusion. Moreover, in many cases, thresholding and thinning amplifies noise in the segmentation output. Figure 3-2 highlights an example of this problem, where the segmentation output (left) drops at occluded segments, and post-processing (center) yields a graph with several incorrect disconnections. Refinement heuristics (right) do not improve the graph. In fact, an erroneous diagonal edge at the top is extended, thereby adding an additional error. Here, we overlay the inferred graph (yellow) over ground truth from OSM (blue).

Thus, we find that the error rate of existing methods is too high for the road network graphs that they produce to be useful in practice.
Chapter 4

Iterative Graph Construction

In segmentation-based approaches, the CNN is trained with a loss function evaluated independently on each pixel. As a result, the CNN yields a noisy output in regions where it has a low confidence about the presence and absence of a road. Although refinement is sufficient to remove basic types of noise, we find that many forms of noise are too extensive to compensate for, as in Figure 3-2. Given this segmentation output, even a human would find it difficult to accurately map the road network. Because the CNN is trained only to classify individual pixels in an image as roads, it leaves us with an untenable jigsaw puzzle of deciding which pixels form the road centerlines, and where these centerlines should be connected.

These findings suggest that we need a different approach that can produce a road network directly, without going through the noisy intermediate image representation of the road network. We propose an iterative graph construction architecture to do this. By breaking down the mapping process into a series of steps that build a road network graph iteratively, we will show that we can derive a road network from the CNN, thereby eliminating the requirement of a complex post-processing pipeline and yielding more accurate maps.

In contrast to the segmentation approach, our approach consists of a search algorithm, guided by a decision function implemented via a CNN, to compute the graph iteratively. The search walks along roads starting from a set of locations known to be on the road network. Vertices and edges are added in the path that the search
Algorithm 1 Iterative Graph Construction

**Input:** Starting location $V_0$ and the bounding box $B$

initialize graph $G$ and vertex stack $S$ with $V_0$

while $S$ is not empty do

    action, $\alpha :=$ decision_func($G$, $S_{top}$, Image)

    $u := S_{top} + (D \cos \alpha, D \sin \alpha)$

    if $action = stop$ or $u$ is outside $B$ then

        pop $S_{top}$ from $S$

    else

        add vertex $u$ to $G$

        add an edge ($S_{top}$, $u$) to $G$

        push $u$ onto $S$

    end if

end while

follows. The decision function is invoked at each step to determine the best action to take: either add an edge to the road network, or step back to the previous vertex in the search tree.

In Section 4.1, we introduce the overall search algorithm. Then, in Sections 4.2 and 4.3, we discuss the implementation of the decision function with a CNN. In Section 4.4, we introduce optimizations that improve the performance of the CNN. Lastly, in Section 4.5, we discuss approaches to pick starting locations for the search algorithm.

### 4.1 Search Algorithm

**Overview.** Algorithm 1 shows the pseudocode for the search procedure.

We input a region $(V_0, B)$, where $V_0$ are the known starting locations, and $B$ is a bounding box defining the area in which we want to infer the road network. The search algorithm maintains a graph $G$ and a stack of vertices $S$ that both initially contain only starting vertices $V_0$. $S_{top}$, the vertex at the top of $S$, represents the current location of the search.

At each step, the decision function is presented with $G$, $S_{top}$, and an aerial image centered at $S_{top}$'s location. It can decide either to walk a fixed distance $D$ (we use $D = 12$ meters) forward from $S_{top}$ along a certain direction, or to *stop* and return
Figure 4-1: Exploring a T intersection in the search process.

to the vertex preceding $S_{\text{top}}$ in $S$. When walking, the decision function selects the direction from a set of $\alpha$ angles that are uniformly distributed in $[0, 2\pi)$. Then, the search algorithm adds a vertex $u$ at the new location (i.e., $D$ away from $S_{\text{top}}$ along the selected angle), along with an edge $(S_{\text{top}}, u)$, and pushes $u$ onto $S$ (in effect moving the search to $u$).

If the decision process decides to “stop” at any step, we pop $S_{\text{top}}$ from $S$. Stopping indicates that there are no more unexplored roads (directions) adjacent to $S_{\text{top}}$. Note that because only new vertices are ever pushed onto $S$, a “stop” means that the search will never visit the vertex $S_{\text{top}}$ again.

Figure 4-1 shows an example of how the search proceeds at an intersection, where the decision function makes correct decisions on each step. The blue path represents the position of the road in the satellite imagery. Circles are vertices in $G$, with $S_{\text{top}}$ in purple and $V_0$ in orange. When we reach the intersection, we first follow the upper branch, and once we reach the end of this branch, the decision function selects the “stop” action. Then, the search returns to each vertex previously explored along the left branch. Because there are no other roads adjacent to the upper branch, the decision function continues to select the stop action until we come back to the intersection. At the intersection, the decision function leads the search down the
lower branch. Once we reach the end of this branch, the decision function repeatedly selects the stop action until we come back to the first vertex in $V_0$ and $S$ becomes empty. When $S$ is empty, the construction of the road network is complete.

**Merging.** Since road networks consist of cycles, it is possible that we will turn back on an earlier explored path. The search algorithm includes a merging step to handle this: when processing a walk action, if $u$ is within distance $3D$ of a vertex $v \in G$, then we propose $(u, v)$ as a candidate for merging. We then apply several filtering rules to avoid spurious merges. First, merging should not yield small loops; for example, if a road forks into two at a small angle, we should not merge one fork into the other. So, if the shortest distance in $G$ from $S_{\text{top}}$ to $v$ is less than $6D$, then we don't merge.

Second, when the search process is following one side of a parallel pair of roads, candidates that merge to the other side will be proposed. We include two angle-based filtering rules to reject these proposals. First, if the angle from $u$ to $v$ is similar to all angles between $v$ and its neighbors, then we do not merge, since this implies that the two segments are almost parallel. Additionally, we require that the angle from $S_{\text{top}}$ to $u$ be similar to the angle from $u$ to $v$.

**Bounding Box.** The search process as described above may walk out of our bounding box $B$. To avoid this, when processing a walk action, if $u$ is not contained in $B$, then we treat it as a stop action.

### 4.2 CNN Decision Function

A crucial component of our algorithm is the decision function, which we implement with a CNN. The input layer consists of a $d \times d$ window centered on $S_{\text{top}}$. This window has four channels. The first three channels are the RGB values of the $d \times d$ portion of aerial imagery around $S_{\text{top}}$. The fourth channel is the graph constructed so far, $G$. We render $G$ by drawing anti-aliased lines along the edges of $G$ that fall inside the window. Including $G$ in the input to the CNN is a noteworthy aspect of our method. First, this allows the CNN to understand which roads in the aerial imagery have been explored earlier in the search, in effect moving the problem of excluding these roads
Figure 4-2: Overview of RoadTracer.

from post-processing to the CNN. Second, it provides the CNN with useful context; e.g., when encountering a portion of aerial imagery occluded by a tall building, the CNN can use the presence or absence of edges on either side of the building to help determine whether the building occludes a road.

The output layer consists of two components: an action component that decides between walking and stopping, and an angle component that decides which angle to walk in. The action component is a softmax layer with 2 outputs, $O_{\text{action}} = (O_{\text{walk}}, O_{\text{stop}})$. The angle component is a sigmoid layer with $a$ neurons, $O_{\text{angle}} = (o_1, \ldots, o_a)$. Each $o_i$ corresponds to an angle to walk in. We use a threshold to decide between walking and stopping. If $o_{\text{walk}} \geq T$, then walk in the angle corresponding to $\text{arg max}_i (o_i)$. Otherwise, stop.

Figure 4-2 summarizes the interaction between the search algorithm and the decision function, and the implementation of the decision function with a CNN. The search algorithm maintains the graph constructed so far and a stack of vertices, and calls the CNN on each step to determine the action to take. The CNN input consists of two components, with red squares indicating the input window.

We noted earlier that our solution does not require complex post-processing heuris-
tics, unlike segmentation-based methods where CNN outputs are noisy. The only post-processing required in our decision function is to check a threshold on the CNN outputs and select the maximum-valued index of the output vector. Thus, our method enables the CNN to directly produce a road network graph.

4.3 CNN Training

We now discuss the training procedure for the decision function. We assume we have a ground truth map $G^*$ (e.g., from OpenStreetMap). Training the CNN is non-trivial: the CNN takes as input a partial graph $G$ (generated by the search algorithm) and outputs the desirability of walking at various angles, but we only have this ground truth map. How might we use $G^*$ to generate training examples?

4.3.1 Static Training Dataset

We initially attempted to generate a static set of training examples. For each training example, we sample a region $(V_0, B)$ and a step count $n$, and initialize a search. We run $n$ steps of the search using an “oracle” decision function that uses $G^*$ to always make optimal decisions. The state of the search algorithm immediately preceding the $n$th step is the input for the training example, while the action taken by the oracle on the $n$th step is used to create a target output $O^*_{\text{action}} = (o^*_{\text{walk}}, o^*_{\text{stop}}), O^*_{\text{angle}} = (o^*_1, \ldots, o^*_a)$.

We can then train a CNN using gradient descent by back-propagating a cross entropy loss between $O^*_{\text{action}}$ and $O^*_{\text{action}}$, and, if $o^*_{\text{walk}} = 1$, a mean-squared error loss between $O^*_{\text{angle}}$ and $O^*_{\text{angle}}$.

Determining optimal decisions ($O^*$) involves identifying angles corresponding to the unexplored locations around $S_{\text{top}}$. We maintain a mapping $\star$ from vertices in $G$ to their corresponding positions in $G^*$ during the search. Each time the oracle is queried, we identify all positions in $G^*$ that are $D$ away from $\star(S_{\text{top}})$ by performing a breadth first search from $\star(S_{\text{top}})$ in $G^*$. For every such position $p$, we compute $\alpha_p$, the angle from $S_{\text{top}}$ to $p$, and set $o^*_{\alpha_p} = 1$. To avoid traversing parts of $G^*$ that have already been explored in $G$, we do not traverse edges that appear in the range of $\star$.
Figure 4-3: Problems when training on static training examples.

during the breadth first search. To update $\star$, if the overall search adds a new vertex $u$ at an angle $\alpha_p$ from $S_{top}$, we set $\star(u) = p$.

However, we found that although the CNN can achieve high performance in terms of the loss function on the training examples, it performs poorly during inference. This is because $G$ is essentially perfect in every example that the CNN sees during training, as it is constructed by the oracle based on the ground truth map. During inference, however, the CNN may choose angles that are slightly off from the ones predicted by the oracle, resulting in small errors in $G$. Then, because the CNN has not been trained on imperfect inputs, these small errors lead to larger prediction errors, which in turn result in even larger errors.

Figure 4-3 shows a typical example of this snowball effect for CNNs trained on static training examples. The CNN does not output the ideal angle at the turn; this causes it to quickly veer off the actual road (blue) because it never saw such deviations from the road during training, and hence it cannot correct course. We tried to mitigate this problem by using various methods to introduce noise on $G$ in the training examples. Although this reduces the scale of the problem, the CNN still yields low performance at inference time, because the noise that we introduce does not match the characteristics of the noise introduced inherently by the CNN during inference. Thus, we conclude that a static training dataset is not suitable.
4.3.2 Dynamic Labels

We instead generate training examples dynamically by running the search algorithm with the CNN as the decision function during training. As the CNN model evolves, we generate new training examples as well.

Given a region \((V_0, B)\), training begins by initializing an instance of the search algorithm \((G,S)\), where \(G\) is the partial graph (initially containing only \(V_0\)) and \(S\) is the vertex stack. On each training step, as during inference, we feed-forward the CNN to decide on an action based on the output layer, and update \(G\) and \(S\) based on that action.

In addition to deciding on the action, we also determine the action that an oracle would take, and train the CNN to learn that action. The key difference from the static dataset approach is that, here, \(G\) and \(S\) are updated based on the CNN output and not the oracle output; the oracle is only used to compute a label for back-propagation.

The basic strategy is similar to before. On each training step, based on \(G^*\), we first identify the set of angles \(R\) where there are unexplored roads from \(S_{\text{top}}\). Next, we convert \(R\) into a target output vector \(O^*\). If \(R\) is empty, then \(o^*_{\text{stop}} = 1\). Otherwise, \(o^*_{\text{walk}} = 1\), and for each angle \(\theta \in R\), we set \(o^*_i = 1\), where \(i\) is the closest walkable angle to \(\theta\). Lastly, we compute a loss between \(O\) and \(O^*\), and apply back-propagation to update the CNN parameters.

A key challenge is how to determine \(\prec(S_{\text{top}})\), i.e., how to decide where to start the breadth first search in \(G^*\) to compute \(R\). Recall that in the static dataset approach, the overall search only followed directions output by the oracle decision function, and we could associate each of those directions with a position in \(G^*\). Here, the search algorithm uses the CNN as the decision function.

The naive approach is to start the walk from the closest location in \(G^*\) to \(S_{\text{top}}\). However, we show that this approach does not work well. In Figure 4-4, the training labels would direct the system towards the bottom road instead of returning to the top road. The black circles make up \(G\), while the blue indicates the actual road.

To solve this problem, we apply a map-matching algorithm to find a path in \(G^*\)
that is most similar to a path in $G$ ending at $S_{\text{top}}$. To obtain the path $p$ in $G$, we perform a random walk in $G$ starting from $S_{\text{top}}$. We stop the random walk when we have traversed a configurable number of vertices $w$ (we use $w = 10$), or when there are no vertices adjacent to the current vertex that haven’t already been traversed earlier in the walk. Then, we match this path to the path $p^*$ in $G^*$ to which it is most similar. We use a standard map-matching method based on the Viterbi algorithm [23]. $*(S_{\text{top}})$ is the endpoint of the last edge in $p^*$.

Finally, we maintain a set $E$ containing edges of $G^*$ that have already been explored during the walk. $E$ is initially empty. On each training step, after deriving $p^*$ from map-matching, we add each edge in $p^*$ to $E$. Then, when computing $R$, we exclude angles that correspond to roads already in $E$.

4.4 Optimizations

We have described the essential incremental mapping architecture and basic training process. We now detail several optimizations that we use to improve performance.

**Computing Optimal Angles.** When computing $O^*$, we need to compute an angle from $S_{\text{top}}$ to a position $p$ produced from the breadth first search. The obvious approach is to use the angle of the vector $p - S_{\text{top}}$. However, if $D$ is small relative to the road width, then there is some ambiguity in defining the representative polyline for the road. If the CNN disagrees with the OSM dataset on the position of the polyline,
then this procedure for computing the angle will unjustifiably push the CNN towards the polyline from $G^*$.

To correct this behavior, we average the angle of $p - S_{top}$ with the angle of the edge in $G^*$ that $p$ falls on. This improves $O^*$ because although the ground truth polyline may be offset from the road centerline, the angle of the polyline is typically still accurate.

Figure 4-5 shows an example of both the problem and our approach for computing the target angles for $O^*$. The blue line is the ground truth polyline representing the road; it is slightly offset from the centerline in the imagery. The purple circle is $S_{top}$, and the green circle is the map-matched location. The gold circle is obtained by performing a search for a fixed distance $D$ in $G^*$ along unexplored edges, starting from the green circle. Because $D$ is smaller than the width of the road, using the angle between $S_{top}$ and the position obtained from the search would push the search towards the ground truth polyline. However, this behavior is incorrect since $S_{top}$ is already between the road boundaries. By averaging this angle with the angle of the ground truth polyline, we are able to smooth the target angle and reduce the dependency of the target angles on the offset between $G^*$ and the imagery.

**Improved Explored Edge Set $E$.** We occasionally observe cases where the CNN extends the path past an intersection, but then decides to take a branch rather than
continuing straight. At the time the CNN passes the intersection, \( p^* \) may include the edge extending forwards out of the intersection. Then, this edge will be added to \( E \) and treated as explored, even though the forward road has not actually been covered.

We handle this by adjusting the method by which we add edges to \( E \). If the CNN chooses a walk action and the action is processed normally, then we only add the first \( w/2 \) edges in the edge sequence corresponding to \( p^* \) to \( E \). However, if the CNN chooses a stop action, or if we stop because of a loop or exiting \( B \), then we add all of the edges.

**Parallel Search Instances and Regions Optimization.** The CNN training converges very slowly if we train on one instance of the search algorithm at a time. For example, a starting location might be at the beginning of a long straight road without any branches. In this case, \( o^* \) will point to the same angle until the road is passed, and a low learning rate is needed to prevent the CNN from simply learning to always output this angle.

To improve the convergence speed, we run multiple instances of the search algorithm in parallel. When training begins, we initialize a set of search algorithm states \( A \). On each training step, we sample a state to use uniformly from \( A \). If the search algorithm finishes (i.e., \( S \) becomes empty), then we reset the state to a new initial state.

Additionally, we divide our training data into multiple regions. Suppose that \( B \) is the bounding box of the overall training region. Each time we initialize a search algorithm state (at the start of training, and also whenever one instance finishes), we randomly select a 1 km by 1 km bounding box \( B' \) contained in \( B \), along with a random vertex \( v \in G^* \) such that \( v \) is contained in \( B' \).

**Joint Training for Segmentation Optimization.** Since choosing the best action on each step of the search is related to the task of identifying the positions of roads in the satellite imagery, we hypothesize that jointly training for segmentation will improve performance.

In a naive approach, we might jointly train the CNN to output road and non-road
Figure 4-6: Single starting location is insufficient in weakly connected cities.

labels as in Section 3. However, the search process benefits most from knowledge of the positions of only those roads that are reachable from $S_{top}$, rather than all roads in the satellite imagery. For example, if we are currently following a highway overpass, then the CNN should not output the pixels corresponding to the underpass since it is disconnected from the overpass in $G^*$. Our segmentation labels are nearby-road and not-nearby-road. We perform a breadth first search from $\ast(S_{top})$ in $G^*$, stopping after we traverse a distance $3D$ or reach a dead-end. We then label pixels close to any edge traversed by the search as nearby-road, and other pixels as not-nearby-road.

4.5 Picking Starting Locations

Unlike segmentation approaches, our iterative graph construction approach depends on starting locations for the search. In this section, we address that dependency.

We initially attempted to provide the search process with a single starting location known to be on the road network. However, we found that in cities consisting of
multiple weakly connected components, the search process would often terminate after exploring only one component. Figure 4-6 shows an example of this behavior. The search algorithm may produce accurate topology in one component. However, it fails to traverse the bridge to the other side of the city, and so roads there are left unexplored. Although we may not be able to capture the bridge due to insufficient training data on bridges, the search process should still be able to capture roads in each component.

Thus, we propose four strategies for obtaining starting locations that are useful in different contexts.

**Semi-automated Map Inference.** First, a human worker could repeatedly provide starting locations in unexplored parts of the road network. The weakly connected components are generally quite large; for example, in Figure 4-6, Manhattan is one component. So, with this method, a small number of starting locations could yield an inferred road network that covers a large region.

In this approach, the worker would first supply one starting location. The search algorithm would run until termination, i.e., execute until the search returns to the provided starting location and $S$ becomes empty. Then, the worker could scan the region of interest for portions that were not covered, and supply another starting location in such a region, and so on.

**Existing Road Map.** Existing road maps could serve as another source of starting locations. This approach is effective when road maps for the region of interest already exist, but may have poor coverage. We first densify the existing map, placing vertices $D$ away from each other. Then, we simply let $V_0$ contain every vertex in the densified map. Thus, each point along the road network in the existing map is used as a starting location, and the search process adds additional roads to the map.

Because road maps generally cover major roads, and major roads typically span across different parts of a city, this approach is likely to resolve the weakly connected component issue.

We now propose two fully automated strategies for obtaining starting locations.
Two-phase Map Inference. We observe that, if we set a low enough stopping threshold $T$, then the search process is likely to detect roads across all weakly connected components. In Figure 4-6, a low threshold would enable the search algorithm to traverse the bridge. On the other hand, a low threshold would also yield a noisy inferred road network graph, with many incorrect segments. Nevertheless, if we can prune low-confidence segments from this graph, then we may be able to produce a graph with both good coverage and high accuracy.

Two-phase map inference builds on this observation. In the first phase, we use a low stopping threshold so that the search is likely to extend across the city. We then prune the resulting graph and only retain high-confidence sequences of edges, i.e., road segments where the CNN output was much higher than the threshold. This yields $G_1$. In the second phase, we restart the search with a new, empty graph $G_2$, and use an arbitrary vertex from $G_1$ as a starting location. After the search terminates, we remove vertices from $G_1$ that are close to some edge in $G_2$. If $G_1$ is not empty, then we add an arbitrary vertex in $G_1$ to $G_2$ and restart the search from there; this is repeated until $G_1$ is empty.
We show an example of two-phase map inference in Pittsburgh in Figure 4-7. Because wide rivers separate the three components of the city, the search process in iterative graph construction may not visit all of the components if the threshold is too high. By setting a low threshold on the initial pass, we are able to obtain edges over the entire city (left). We prune these edges to retain only high confidence edges (center), and then re-execute the search process with a higher threshold using the high confidence edges as starting locations (right).

**Starting Locations from Segmentation Approach.** Finally, we could use peaks in the segmentation output as starting locations for iterative graph construction. The process is similar to two-phase map inference: we pick the pixel with the highest probability of being in the road class and use it as the starting location for the first search. After the search terminates, we unset pixels that are close to some edge in the inferred graph, and restart the search from another high probability pixel. We repeat until there are no more high probability pixels in the segmentation output.
Chapter 5

Applications to Map Making

Up to now, we have focused on the problem of inferring new road network graphs for a region of interest from scratch. In practice, however, we typically have existing road maps; in many cases, it would be more useful to improve those road maps rather than replace them. In this chapter, we discuss two issues that our iterative graph construction approach can solve in improving road maps. In Section 5.1, we discuss improving the coverage of the map. In Section 5.2, we discuss an approach for identifying road segments in the map that no longer exist.

5.1 Improving Coverage

Although the coverage of maps is very high in the urban core of cities, maps often fail to incorporate many roads on the outskirts of cities, or roads in rural regions. Because of the global availability of satellite imagery, iterative graph construction using satellite imagery can be used to add these missing roads to the map dataset.

The most basic approach for applying iterative graph construction to improving road map coverage is to apply the search process using points along the existing road map as starting locations, as we discussed in Section 4.5. However, we find that in some cases, this basic approach does not work well. Below, we discuss problems with this method and potential solutions.

Outdated Imagery. In some cities, satellite imagery lags far behind the state of the
Figure 5-1: Imagery in Doha lags behind the physical road network.

physical road network due to very frequent changes to the road network as the city rapidly expands. If the map dataset is more up-to-date than the imagery, imagery-based approaches will make changes that revert updates to the road network. Figure 5-1 shows an example of this problem in Doha, where a roundabout has been relocated from the position in the imagery to a new location to make way for the construction of a rapid transit line. The map (orange) has already reflected the change. However, iterative graph construction adds segments corresponding to the former position of the roundabout back to the map.

The simplest solution to this issue is to run iterative graph construction when the imagery is taken. However, in some cases, running iterative graph construction immediately after imagery becomes available may not be possible. For example, there may be an urgent need for maps with better coverage in a region that had a natural disaster, and it may be several months before new imagery will be released; additionally, due to the need to process the raw imagery to obtain an orthonormalized imagery layer that iterative graph construction depends on, there may be a substantial delay between the time that imagery is collected and the time that it becomes available.
To nevertheless mitigate the problem, we consider two additional data sources: the timestamp of the imagery, and the history of segment deletions and updates in the map dataset. For every edge $e$ added by iterative graph construction, we query the historical data. If there was a segment in the map dataset close to $e$ at some point after the imagery was taken, then we do not incorporate $e$ into the road map.

**Differences in Road Style.** Another issue is differences in road style between regions. For example, because roads in Doha look very different from roads in Boston, a CNN trained in Boston will perform poorly in Doha. We develop two strategies to mitigate this issue.

First, we can train different models for different regions. We first divide the world into evenly sized grid cells of a certain size; then, we train a separate CNN decision function in each cell. Lastly, we use the CNN when performing inference to improve the coverage of the map in the cell. Although the CNN is trained not to output the very missing roads that we want iterative graph construction to identify, we find that as long as the cell size is large enough and the CNN is sufficiently constrained, the CNN will learn a general model of road appearances and be able to extend the road network to these missing roads. Additionally, this system of training ensures that the road style is the same between training and inference.

However, if the quality of the map dataset is particularly poor in a cell, then we may not be able to train a CNN that performs well for that cell. Figure 5-2 shown an example in northern Alberta, where map datasets not only have poor coverage, but the polylines for roads that do appear in the dataset are often severely misaligned from
the road. This leads to the presence of both false positive examples and false negative examples during training, and the CNN after training fails to produce segments with accurate geometry.

Thus, our second approach is to train one model for the entire world. However, if we sample regions during training uniformly across the world, the vast majority of regions will come from rural areas; thus, the resulting model will not perform well in cities, especially on roads that are occluded by tall buildings or shadows. We instead divide regions into buckets based on road density; during training, we first sample a bucket uniformly, and then sample a region uniformly from the regions belonging to the selected bucket. Each bucket covers a range of the total length of roads appearing in the region.

**Semi-Automated Mapping.** In some applications, fully automated mapping may not be possible. For example, roads in OpenStreetMap must be labeled with a road type such as primary, secondary, tertiary, residential, or service, indicating how major the road is and what type of traffic it serves. Automatic labeling from imagery is often not possible because the road type may depend on use cases that are not visible from the imagery. Also, when the accuracy of road segments incorporated into the map dataset is of paramount importance, and coverage is a secondary concern, fully automated map inference may not be suitable.

We propose semi-automated mapping for these applications. Our basic process is to have human mappers validate the missing segments identified by iterative graph construction. If a segment correctly captures a missing road, the mapper might make minor modifications to improve the geometrical alignment, and then incorporate the segment into the dataset. We consider two issues in our approach.

First, iterative graph construction produces vertices every $D$ distance along a road. It is often not desirable for map datasets to contain so many vertices. We use Douglas-Peucker [13] to simplify the inferred segments before they are shown to the mapper.

Second, if this semi-automated approach is executed over a very large region, then it may take several months or even years before the mappers finish validating all of
the segments. We develop a prioritization system so that the most useful segments are validated first. We split the inferred segments into connected components, and then prioritize the connected components based on three features: (a) the diagonal length of the bounding box of the connected component, since roads that span a larger distance are more useful; (b) the probabilities associated with the directions corresponding to the segments during the search process, so that segments that are more likely to be correct are validated first; and (c) the number of points where the component connects with the existing road network, since having more such points implies providing more alternative connections between existing roads.

5.2 Removing Erroneous Roads

Errors in map datasets are common, and may arise from poor quality data sources or other errors when the incorrect road was added to the map, or changes to the physical road network after the road was added. To remove erroneous roads using iterative graph construction, we create a partial graph where the road is excluded, force the search process to follow the road, and compute the average confidence output by the CNN while following the road. If the average confidence is below a threshold, then the road is likely an error.
Chapter 6

Evaluation

In this chapter, we evaluate our iterative graph construction approach, RoadTracer, against two baseline approaches on a dataset containing imagery from forty cities.

6.1 Dataset

To evaluate our approach, we assemble a large corpus of high-resolution satellite imagery and ground truth road network graphs covering the urban core of forty cities across six countries. For each city, our dataset covers a region of approximately 24 sq km around the city center. We obtain satellite imagery from Google at 60 cm/pixel resolution, and the road network from OSM (we exclude certain non-roads that appear in OSM such as pedestrian paths and parking lots). We convert the coordinate system of the road network so that the vertex spatial coordinate annotations correspond to pixels in the satellite images.

We split our dataset into a training set with 25 cities and a test set with 15 other cities. To our knowledge, we conduct the first evaluation of automatic mapping approaches where systems are trained and evaluated on entirely separate cities, and not merely different regions of one city, and also the first large-scale evaluation over aerial images from several cities. Because many properties of roads vary greatly from city to city, the ability of an automatic mapping approach to perform well even on cities that are not seen during training is crucial; the regions where automatic
mapping holds the most potential are the regions where existing maps are non-existent or inaccurate.


The test set consists of Amsterdam, Boston, Chicago, Denver, Kansas City, Los Angeles, Montreal, New York City, Paris, Pittsburgh, Salt Lake City, San Diego, Tokyo, Toronto, and Vancouver.

6.2 Baselines

We compare RoadTracer with two baselines: DeepRoadMapper [18] and our own segmentation-based approach. Because the authors were unable to release their software to us, we implemented DeepRoadMapper, which trains a residual network with a soft intersection-over-union (IoU) loss function, extracts a graph using thresholding and thinning, and refines the graph with a set of heuristics and a missing connection classifier.

However, we find that the IoU loss results in many gaps in the segmentation output, yielding poor performance. Thus, we also implement our own segmentation approach that outperforms DeepRoadMapper on our dataset, where we train with cross entropy loss, and refine the graph using a four-stage purely heuristic cleaning process that prunes short segments, removes small connected components, extends dead-end segments, and merges nearby junctions. In our segmentation approach, we use a 20-layer U-Net-like CNN architecture. The first 10 layers downsample the input to features at 1/32 of the input resolution, and the second 10 layers upsample the features back to the original resolution. We train the model on random $256 \times 256$ crops of imagery with cross entropy loss and the AdaMax gradient descent operator; at inference time, we input $1024 \times 1024$ crops of imagery.
6.3 Metrics

We evaluate RoadTracer and the segmentation schemes on TOPO [5], SP [25], and a new junction metric defined below. TOPO and SP are commonly used in the automatic road map inference literature [6, 22, 26, 1].

The first metric, TOPO, evaluates a combination of topology (connections between roads) and geometry (alignment of individual roads). TOPO simulates a car driving a certain distance from several seed locations, and compares the destinations that can be reached in $G$ from each seed location with those that can be reached in $G^*$. More specifically, given a source location in $G^*$, we identify the closest location in $G$, and perform a search in each graph from those locations. During the searches, we place markers every $m$ meters along the search path. The search ends after walking $R$ meters from the source, for a configurable search radius $R$. We compute a precision and recall between the markers placed by the two searches, where two markers match if the distance between them is less than a matching threshold $r$. This procedure is repeated over a large number of source location, and precision and recall are averaged over the iterations.

Figure 6-1 illustrates one iteration of TOPO comparing two inferred maps (right).
against a ground truth map (left). TOPO is evaluated on the two inferred maps starting from the purple seed. On the inferred maps, green indicates a marker that matches with the ground truth map, while red indicates an unmatched marker. Precision is the number of green markers divided by the total number of markers in each of the inferred maps. For this starting location, the first inferred map has 0.8 precision, and the second has 1.0 precision. Recall is computed similarly based on markers in the ground truth map that match with the inferred map (not shown).

SP generates a large number of origin-destination pairs, computes the shortest path between the origin and the destination in both $G$ and $G^*$ for each pair, and outputs the fraction of pairs where the shortest paths are similar. For a particular origin-destination pair $(u, v)$, where $u$ and $v$ are vertices in $G^*$, SP begins by matching $u$ and $v$ with the closest vertices $u'$ and $v'$ in $G$. Then, SP computes $d^*$, the shortest distance between $u$ and $v$, and $d$, the shortest distance between $u'$ and $v'$. We say that the shortest paths are similar if the following hold: the shortest path either exists in both map or exists in neither map, and the difference between $d$ and $d^*$ is at most 5%.

We refer back to Figure 6-1 to highlight a case where TOPO gives misleading scores. We argue that in a practical mapping scenario, the first inferred map is unusable as too many edges are placed incorrectly; it would be easier to trace the roads in this region from an empty road map rather than adjust the edges in the inferred map. On the other hand, the missing edge in the second map can be quickly corrected manually. However, TOPO penalizes the second map heavily on recall: although the second map has 1.0 recall for the shown starting location, the recall is close to 0.5 when the starting location falls on the bottom of the top component or the top of the bottom component. Thus, the $F_1$ score for the first map is 0.82, but for the second map it is only 0.80.

In general, we find that both TOPO and SP tend to assign higher scores to noisier maps, and thus don’t correlate well with the usability of an inferred map. Additionally, the metrics make it difficult to reason about the cause of a low or high score.
Thus, we propose a new evaluation metric with two goals: (a) to give a score that is representative of the inferred map's practical usability, and (b) to be interpretable. Our metric compares the ground truth and inferred maps junction-by-junction, where a junction is any vertex with three or more edges. We first identify pairs of corresponding junctions \((v, u)\), where \(v\) is in the ground truth map and \(u\) is in the inferred map. Then, \(f_{v,\text{correct}}\) is the fraction of incident edges of \(v\) that are captured around \(u\), and \(f_{u,\text{error}}\) is the fraction of incident edges of \(u\) that appear around \(v\). For each unpaired ground truth junction \(v\), \(f_{v,\text{correct}} = 0\), and for each unpaired inferred map junction \(u\), \(f_{u,\text{error}} = 1\). Finally, if \(n_{\text{correct}} = \sum_v f_{v,\text{correct}}\) and \(n_{\text{error}} = \sum_u f_{u,\text{error}}\), we report the correct junction fraction \(F_{\text{correct}} = \frac{n_{\text{correct}}}{\# \text{junctions in } G}\) and error rate \(F_{\text{error}} = \frac{n_{\text{error}}}{n_{\text{error}} + n_{\text{correct}}}\).

TOPO and our junction metric yield a precision-recall curve, while SP produces a single similar path count.

In Figure 6-1, the first map has \(F_{\text{correct}} = 0.9\), \(F_{\text{error}} = 0.23\), while the second has \(F_{\text{correct}} = 0.9\), \(F_{\text{error}} = 0.00\).

6.4 Quantitative Results

We evaluate performance of the three methods on 15 cities in the test set. We supply starting locations for RoadTracer by identifying peaks in the output of our segmentation-based approach. All three approaches are fully automated.

Both RoadTracer and the segmentation approaches have parameters that offer a tradeoff between recall and error rate \((1 - \text{precision})\). We vary these parameters and plot results for our junction metric and TOPO on a scatterplot where one axis corresponds to recall and the other corresponds to error rate. For DeepRoadMapper and our segmentation approach, we vary the threshold used to produce a binary mask. We find that the threshold does not impact the graph produced by DeepRoadMapper, as the IoU loss pushes most outputs to the extremes, and thus only plot one point. For RoadTracer, we vary the walk-stop action threshold \(T\).

We report performance in terms of average \(F_{\text{correct}}\) and \(F_{\text{error}}\) across the test cities in Figure 6-2, and in terms of average TOPO precision and recall in Figure 6-4.
Figure 6-2: Average $F_{\text{correct}}$ and $F_{\text{error}}$ over the 15 test cities.

Figure 6-3: Tradeoff between error rate and recall.
On the junction metric, RoadTracer has a better $F_{\text{error}}$ for a given $F_{\text{correct}}$. The performance improvement is most significant when error rates are between 5% and 10%, which is the range that offers the best tradeoff between recall and error rate for most applications—when error rates are over 10%, the amount of noise is too high for the map to be usable, and when error rates are less than 5%, too few roads are recovered. We show this tradeoff between error rate and recall in Figure 6-3, where we overlay the inferred map in a small crop from Boston as we increase the threshold for our segmentation approach. The junction metric error rates in the crop from left to right are 18%, 13%, and 8%. The map with 18% error is too noisy to be useful. Overall, when the error rate is 5%, the maps inferred by RoadTracer have 45% better average recall ($F_{\text{correct}}$) than those inferred by the segmentation approach (0.58 vs 0.40).

On TOPO, RoadTracer has a lower error rate than the segmentation approaches when the recall is less than 0.43. Above 0.43 recall, where the curves cross, further lowering $T$ in RoadTracer yields only a marginal improvement in recall, but a sig-
nificant increase in the error rate. However, the segmentation approach outperforms RoadTracer only for error rates larger than 0.14; we show in Figure 6-3 that inferred maps with such high error rates are not usable.

We report SP results for the thresholds that yield highest number of correct shortest paths in Table 6.1. For each scheme, we only report results for the threshold that yields the highest correct shortest paths. Long, Short, and NoPath specify different reasons for an inferred shortest path being incorrect (too long, too short, and disconnected). RoadTracer outperforms the segmentation approach because noise in the output of the segmentation approach causes many instances where the shortest path in the inferred graph is much shorter than the path in the ground truth graph.

Our DeepRoadMapper implementation performs poorly on our dataset. We believe that the soft IoU loss is not well-suited to the frequency of occlusion and complex topology found in the city regions in our dataset.

We also report per-city results on the junction metric in Figure 6-5. RoadTracer’s improvement over the segmentation approach in the 5% to 10% error rate range is consistent across virtually all of the cities.

### 6.5 Qualitative Results

In Figure 6-6, we show qualitative results in crops from four cities from the test set: Chicago, Boston, Salt Lake City, and Toronto. For RoadTracer and our segmentation approach, we show inferred maps for the threshold that yields 5% average $F_{error}$. DeepRoadMapper only produces one map, with 19% $F_{error}$. In the crops, we overlay the inferred graph (yellow) over ground truth from OSM (blue).

RoadTracer performs much better on frequent occlusion by buildings and shadows...
Figure 6-5: $F_{\text{correct}}$ and $F_{\text{error}}$ in each test city.
Figure 6-6: Qualitative comparison of inferred road networks.
in the Chicago and Boston regions. Although the segmentation approach is able to achieve similar recall in Boston on the lowest threshold (not shown), several incorrect segments are added to the map. In the Salt Lake City and Toronto regions, performance is comparable. DeepRoadMapper's soft IoU loss introduces many disconnections in all four regions, and the missing connection classifier in the post-processing stage can only correct some of these.
Chapter 7

Conclusion

On the face of it, using deep learning to infer a road network graph seems straightforward: train a CNN to recognize which pixels belong to a road, produce the polylines, and then connect them. But occlusions and lighting conditions pose challenges, and such a segmentation-based approach requires complex post-processing heuristics. By contrast, our iterative graph construction method uses a CNN-guided search to directly output a graph. We showed how to construct training examples dynamically for this method, and evaluated it on 15 cities, having trained on aerial imagery from 25 entirely different cities. To our knowledge, this is the largest map-inference evaluation to date, and the first that fully separates the training and test cities. The principal experimental finding is that, at a 5% error rate, RoadTracer correctly captures 45% more junctions than our segmentation approach (0.58 vs 0.40). Hence, this thesis presents an important step forward in fully automating map construction from aerial images.
Bibliography


