Traffic Control Elements Inference using Telemetry Data and Convolutional Neural Networks

Deeksha Goyal
dgoyal@lyft.com
deeksha@stanford.edu
Lyft Inc. and Stanford University
San Francisco and Stanford, California

Albert Yuen
ayuen@lyft.com
Lyft Inc.
San Francisco, California

Han Suk Kim
hkim@lyft.com
Lyft Inc.
San Francisco, California

James Murphy
jmurphy@lyft.com
Lyft Inc.
San Francisco, California

ABSTRACT

Stop signs and traffic signals are ubiquitous in the modern urban landscape to control traffic flows and improve road safety. Including them in digital maps of the road network is essential for geospatial services, e.g., assisted navigation, logistics, ride-sharing and autonomous driving. This paper proposes to infer them by exclusively relying on large-scale anonymized vehicle telemetry data, which is available for companies offering such services. Vehicle patterns from telemetry data at each intersection are extracted, and we employ a convolutional neural network for the task of labeling these driver patterns. We train our neural network in San Francisco, and choose to test the model in Palo Alto, whose urban layout is significantly different from the urban layout of San Francisco, in order to prove the generality of the algorithm. At a confidence threshold of 90%, our classifier achieves 96.6% accuracy and 66.0% coverage in detecting three classes of traffic control elements: stop signs, traffic signals, and neither. Our work paves a way for inferring traffic control elements for automated map updates.

KEYWORDS

statistical learning, machine learning, deep learning, CNN, neural networks, kernel density estimator, map making, map inference, telemetry, OSM, stop signs, traffic lights, traffic, routing, ETA, locations, mapping, ride-sharing

ACM Reference Format:
Deeksha Goyal, Albert Yuen, Han Suk Kim, and James Murphy. 2019. Traffic Control Elements Inference using Telemetry Data and Convolutional Neural Networks. In *Proceedings of (SIGKDD '19).* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

SIGKDD '19, , Anchorage, Alaska, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Mapping is at the core of the daily operations of ride-sharing companies such as Lyft, Uber, Didi Chuxing or Ola. Accurate digital maps enable smooth pick-ups and drop-offs of passengers by providing, for example, precise building exits, suggested routes to assist drivers, and optimal dispatch by accurately computing estimated time arrival using roads conditions and traffic data.

Building and maintaining an accurate map is challenging in many aspects. One can rely on a fleet of mapping cars to drive around cities and collect road locations and street-level imagery. This method provides a map of the highest quality, but is onerous and expensive to scale because of the operational costs of the fleet of mapping cars. Alternatively, one can rely on satellite or aerial imagery to infer map features. Unfortunately, satellite and aerial imagery can only provide the most basic map features — e.g., roads — but not stacked roads or traffic control elements (TCEs) as aerial images offer no visibility for street-level map features, and those are often imprecise due to occlusion. Another approach is to leverage large-scale proprietary telemetry data generated by ride-sharing companies for each ride. Telemetry data does not suffer from the cost of operating a fleet of mapping cars and their intrinsic low coverage. It also has the potential to infer more types of map features than satellite or aerial imagery.

In this paper, we rely on telemetry data from Lyft rides to infer map features, with an emphasis on TCEs. More broadly, this paper shows that we can infer an accurate map at a reduced cost for all cities where large-scale telemetry data is available.

TCEs are signaling devices or signs that are located at road ends, in the vicinity of a road junction or pedestrian crossing in order to control the flows of traffic. Including TCEs in maps is valuable: a) for accurate routing calculations in order to add a possible time penalty to go from a road segment to the next one, b) for driver position prediction to improve market decisions, c) safety, and d) autonomous driving softwares to plan the behavior of vehicles on the road.

The intuition behind this paper is that, given a large amount of telemetry data, histograms of telemetry data with speed and distance from road ends indicate the behavior of drivers at a road end and can guide the inference of the TCEs at the end of each road segment. The task is treated as a three-class classification problem, with the classes being $C = (\textit{traffic signal}, \textit{stop sign}, \textit{neither})$, and

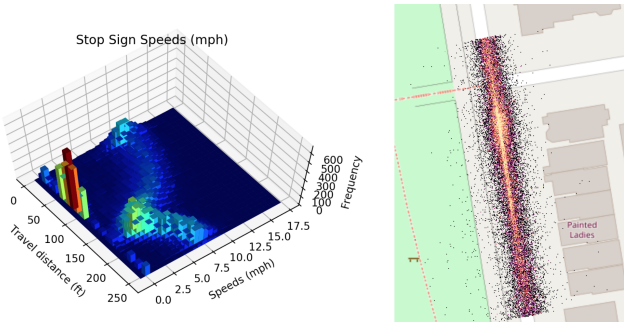

Figure 1: (Left) Histogram of vehicle speed and distance from the road end in North-West direction. The road ends with a stop sign, located at around 175 feet. (Right) Scatter plot of the vehicle data points used to build the histogram

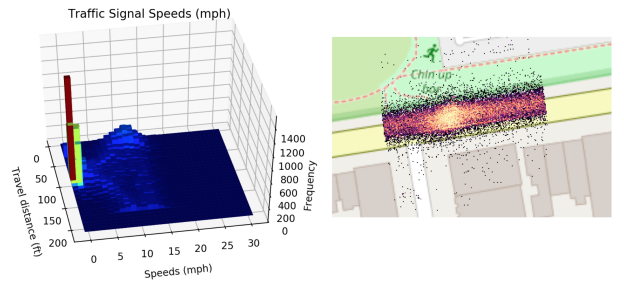

Figure 2: (Left) Histogram of vehicle speed and distance from the road end in South-West direction. The road ends with a traffic signal, located at 80 feet. (Right) Scatter plot of the vehicle data points used to build the histogram

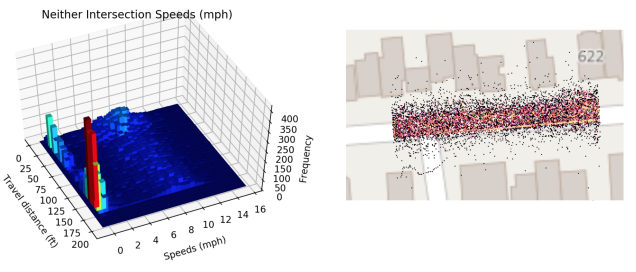

Figure 3: (Left) Histogram of vehicle speed and distance from the road end in South-West direction. The road ends with neither a stop sign, nor a traffic signal. (Right) Scatter plot of the vehicle data points used to build the histogram

we show in Fig. 1, 2 and 3, examples of such histograms for each of the three classes. Fig. 1 represents the histogram for a road segment ending with a stop sign, and we observe that the speed of the drivers decreases as they approach the junction and then increases afterwards, displaying a v shape in the histogram. Fig. 2 represents the histogram for a road segment ending with a traffic signal, and we see not only the similar v shape from the stop sign histogram, but also a line of constant speed, which indicates drivers going through on a green light. An example of a road end with neither is shown in Fig. 3, and this idiosyncratic example reveals that many drivers go through with constant speed. The neither class is an interesting case because it encompasses many different types of junction. For the most part, however, their driver patterns differ noticeably from those of stop signs and traffic signals.

We propose to use a convolutional neural network, a neural network architecture that is shift-invariant and specialized for processing data with a grid-like structure — e.g., images or histograms — to classify the three classes C. More ambitiously, by leveraging large-scale telemetry data, we can use this approach to successfully predict a larger variety of map features, e.g., number of lanes or speed bumps.

This paper is organized as follows: We first provide an overview of related work on telemetry and map inference. Geospatial applications of telemetry data range from traffic understanding down to the level of computing precise locations of vehicles. Map inference is the process of inferring map features — roads, buildings, traffic control elements — mainly from telemetry data and imagery. We then describe the publicly available datasets as well as our anonymized Lyft proprietary dataset employed in this paper. We present the methodology of inferring TCEs from the problem formulation and data processing to the chosen neural network architecture. We subsequently apply our trained deep learning model to San Francisco and Palo Alto, and describe the performance of our model as well as its limitations. Last, we suggest what can be done next to extend our present work.

2 RELATED WORK

Telemetry data from smartphones has been used for more than a decade to develop intelligent transportation systems: to understand road usage and traffic [9, 16, 40], or to estimate arrival time from a place to another [20, 21]. Combined with an accurate map of the road network, the path of a vehicle on the road network can be reconstructed using telemetry data, even in the case of noisy and sparse data [29]. Crucial for the development of intelligent transportation systems, this class of algorithms is called map-matching [11, 19, 30]. While map-matching can be generalized for cases when the map is wrong [28], map-matching algorithms — as well as all arrival time estimation and routing algorithms — work best when the map is correct. Here lies the value of map inference.

Most map inference work in the literature has been focused on inferring and updating the road network [1, 5, 12, 26, 33] from GPS traces — e.g., using kernel density estimators [3, 4], iterative methods [15], clustering algorithms [8] or graph spanners [38]. With the availability of inexpensive satellite and aerial imagery data, GPU computing power, as well as progress in convolutional neural network (CNN) for computer vision [36, 37], CNN-powered map inference has been used to infer road segments at varying degrees of success [2, 6, 10, 17, 22, 27].

Inferring traffic control elements with telemetry data has been explored in the past by using ruled-based methods [7, 39] in a set of manually selected candidate locations, or by using more traditional statistical learning approaches, both supervised (random

forest, Gaussian mixture models, SVM, naive Bayes) and unsupervised (spectral clustering) [18]. However, the study relied on hired drivers who probably know the purpose of the study (which may unconsciously bias their driving behavior) and clever but extensive feature engineering which may not be straightforward because of noisy or sparse sensor data, e.g., the number of the times the vehicle is stopped and the final stop duration.

This paper infers traffic control elements for map updates using telemetry data and a CNN-based computer vision approach. This differs from past map inference work as a) most of the past work in map inference focused on inferring the road segments for map updates, b) it relies too much on prior knowledge about the map and driver behavior which means that it cannot be easily deployed in all cities, and c) the work on telemetry data was based on more traditional statistical methods.

While this present paper focuses only on the inference of traffic control elements, the current approach can be generalized to many types of map features.

3 DATASET

This paper relies on three data sources:

San Francisco Open Data - Our Ground Truth: Launched in 2009, the City of San Francisco opened hundreds of datasets pertaining to the urbanism of San Francisco as well as its urban life. Among the available datasets are the stop signs dataset [31] — 10,527 stop signs — and the traffic signal dataset [32] — 1,397 traffic signals, each of which, in that dataset, can represent multiple traffic signals depending on the direction of the road. While OSM also contains traffic signals and stop signs, after benchmarking the coverage and accuracy of the OSM traffic light and stop signs datasets, we decided not to rely on the OSM traffic signal and stop signs datasets because of its poor quality. The datasets from San Francisco Open Data proved to be much more reliable, and are used as ground truth to train our classifier.

The Road Network in Open Street Map (OSM) - What we want to label: OSM is an open source map whose first contributions started in 2004 [13]. We mostly rely on the road network, defined as a directed graph in which the vertices are road junctions and the edges are roads. Our task is to label the end of each directed edge with one of the three classes in $C = (\textit{traffic signal}, \textit{stop sign}, \textit{neither})$. We assume that the road network correctly models the road segments in our physical world, which is an appropriate assumption as the road segments in the road network should first be well modeled before inferring other elements in the road network (e.g., TCE, speed limits, number of lanes, turn restrictions). In practice, an algorithm to detect road segment errors is first run. Then, the algorithm of the present paper is run.

Anonymized proprietary Lyft telemetry data: We leverage Lyft vehicle telemetry data collected for forty days in the summer of 2018 in San Francisco and Palo Alto, collected from smartphones. Each data point contains the *latitude*, *longitude*, *accuracy*, *speed* and *bearing*. A future iteration of this work could make use of more features like *acceleration*, *gyroscope*, and *timestamp*. Contrary to Refs. [7, 18, 39], apart from building histograms, we do not engineer any additional features. Collecting driver data over a short period of time ensures that we are capturing only the TCEs that are currently in the road network. We want, for example, avoid the situation where we collect data for a long period of time (e.g., years), infer that a given road segment possesses a TCE, but realize that the TCE was actually removed during that period of time.

We create bounding boxes over the end of each road segment in San Francisco. We assign vehicle data points to each bounding box, from which we extract driver patterns. 20% of bounding boxes we looked at are reserved for testing, 20% for validation, and the remaining 60% are for training, all of which are selected randomly. We ensured that the training set and the test set are drawn from the same distribution.

4 METHODOLOGY

4.1 Problem Formulation

The TCE prediction task is a multi-label classification problem for labeling traffic flows at junctions. The inputs are diagrams generated by applying a Kernel Density Estimator (KDE) on data points in each bounding box (see Fig. 4), which display the frequency of data points found at certain speeds and distances near junctions. Note that, for example, a four-way junction would have four diagrams for each traffic flow into the junction. The output is a prediction for the traffic being controlled by one of the three classes in C: a *stop sign*, a *traffic signal*, or *neither*.

We optimize for the mean cross-entropy loss function:

$$L = \frac{1}{N}\sum_{i=1}^{N} H(X_i, \mathbf{y_i}) = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{3} -y_{i,j}\log(\hat{y}_{i,j}), \tag{1}$$

where N is the size of the dataset to compute the cross-entropy loss function, H is the cross-entropy between the output of the model applied to the sample X_i (the i-th diagrams) and the ground truth $\mathbf{y}_i$, $\hat{y}_{i,j}$ is the predicted probability output of the model for the sample i and the label j. $y_{i,j} = 1$ if j is the label for i, else $y_{i,j} = 0$.

4.2 Training Data Generation

Our pipeline for generating predictions involves three data processing steps.

4.2.1 Generate Bounding Boxes. In order to know where to collect driver telemetry data from, we create bounding boxes over the end of each road segment using the graph of the road network provided by OSM [13]. Those bounding boxes are defined by the *length*, the *width*, the position of the *center* as well as the *bearing* of the box. We arbitrarily decided to use a length of 57 m and a width of 47 m, and those values can be revisited in a future iteration of this work. The bearing is given by OSM. These boxes also make sure to cover 10 m after the junction so that we can collect the vehicle data as vehicles enter, cross, and leave the junctions. Figures 4 and 5 show examples of the bounding boxes.

We filter out bounding boxes for junctions that have more than four segments as those cases are rare (around 0.01% in San Francisco). We also filter out bounding boxes for junctions that are too close to each other, so that we do not have any bounding boxes cover more than one junction. Even after removing these cases, we retain a coverage of junctions othat is over 90% with 33,532 bounding boxes as shown in Fig. 5.

We then label the bounding boxes using the San Francisco Open Datasets [31, 32]. They are labeled by checking which traffic control element in the dataset is closest to the junction in the bounding box. If there are none, then we label the bounding box as not having any traffic control elements (*neither*). We have manually benchmarked this approach by randomly selecting 100 junctions from the San Francisco Open Datasets and validated that the 100 selected junctions are correct.

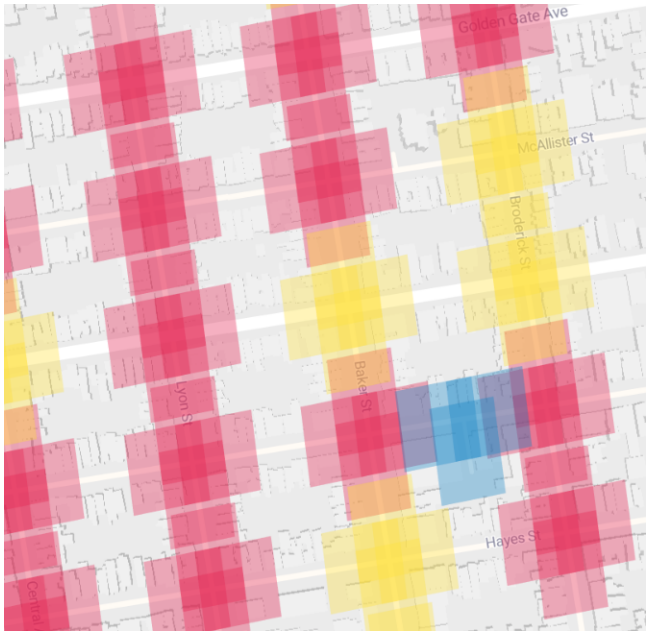

Figure 4: Examples of bounding boxes around Baker Street and Hayes Street in San Francisco with the following labels: Red = *stop sign*, Yellow = *traffic signal*, Blue = *neither*.

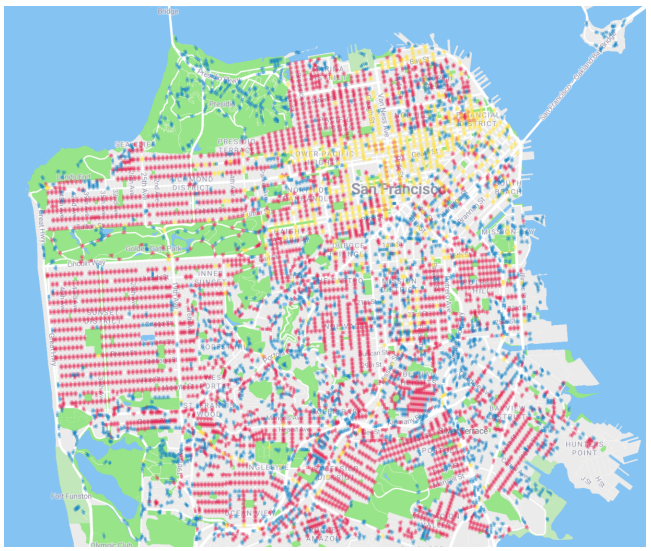

Figure 5: The bounding boxes in San Francisco with the labeling defined above.

4.2.2 Collect Telemetry Data. For each bounding box, we collect driver telemetry data inside of them from 40 days in the summer of 2018 for San Francisco. These days are deliberately chosen to be different days of the week to ensure that we are not overfitting for traffic patterns on certain days. We then place each of these data points into the correct bounding boxes by aligning the *bearing* of the telemetry data and the *bearing* of the bounding box. This prevents collecting driver data in the opposite direction of traffic flow. Note that some data points may be assigned to multiple bounding boxes, since all bounding boxes at a junction cover the center part of the junction.

It is likely that some data points would display low GPS accuracy because of tall buildings in San Francisco as well as low smartphone quality. Thus, we make sure to only collect data points with high GPS *accuracy*.

In order to have enough data points to see clear driver patterns, we only keep bounding boxes that have at least 1,000 data points. This reduces the number of bounding boxes to 24,339. For future iterations, rather than cutting out bounding boxes with fewer data points, we could continue to collect telemetry in each bounding box until we reach the lower bound.

4.2.3 Kernel Density Estimators. For each bounding box, we create a diagram over *speed* and *distance* from junction by applying a 2D Kernel Density Estimator (KDE) [35] on the data with a Gaussian kernel function. The bandwidth of the KDE is determined by Silverman's rule of thumb [35].

At lower speeds, we are likely to see more location data points than at higher speeds because of the sampling rate. This leads to a noticeable amount of driver data points at speed zero at all distances from the junction. These points are not indicative of any driver pattern and add noise. In order to mitigate the effect of this noise, we normalize the diagram with a cube root and min/max normalization. These normalization steps moreover help with surfacing the driver patterns we are searching for.

4.3 Training

We use a deep learning model in order to discern the driver patterns from each diagram. We choose to train a convolutional neural network as this class of neural networks is shift-invariant and specialized for processing data with a grid-like structure, and has been particularly successful in solving computer vision problems [24, 25].

Preprocessing of Diagrams. Before we train, we resize the diagrams to dimensions 224 x 224 x 3. We further normalize the three channels in the images using mean = [0.485, 0.456, 0.406] and std = [0.229, 0.224, 0.225] in order to properly train the neural network.

VGG19. Our classifier uses the VGG19 architecture [37], developed in 2014, which improves upon AlexNet [24] by developing a deeper architecture which leverages small filters. By doing so, the model is able to have the same effective receptive field as a shallower network with larger filters, but is able to have more non-linearities and fewer parameters which increases its performance. We tried using the ResNet architecture [14] and found that there was a negligible difference in performance. This might be because the input images simply contain one pattern per image, which is

much simpler than images that are often fed into convolutional neural networks, making a change in architecture unnecessary.

Transfer Learning. We tried a custom neural network architecture initialized with random weights. However, we found that when we initialized our network with VGG19 pretrained on ImageNet, there was a significant boost in accuracy. Despite our input images being very different from the ImageNet images, transfer learning has demonstrated the ability to detect underlying patterns effectively [34].
We initialize with the weights from VGG19 pretrained on the ImageNet dataset and do not freeze any layers. We add an additional fully connected layer to the end of the network that has randomly initialized weights and outputs three scores.

Parameters. We decrease the learning rate by gamma after 14 epochs. We also use the Adam adaptive learning update [23] with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 1e - 8$. We use a batch size of 8.

5 EXPERIMENTAL RESULTS

5.1 Evaluation Metrics

Our approach is to first get a well-trained model based on accuracy in San Francisco. We then test the model in Palo Alto to see its generalizability. We evaluate this model by accuracy, precision, recall, F1 score, and coverage of dataset.

5.2 Hyperparameter Tuning

For hyperparameter tuning, we use the aforementioned diagrams and model architecture, and we tune the learning rate and gamma decay rate values. We find that the best learning rate and gamma values are 0.001 and 0.1 respectively.

5.3 Train on San Francisco

We trained our model using San Francisco data.

Accuracy. After hyperparameter tuning, our best model for the three classes has a validation accuracy of 91.26%.

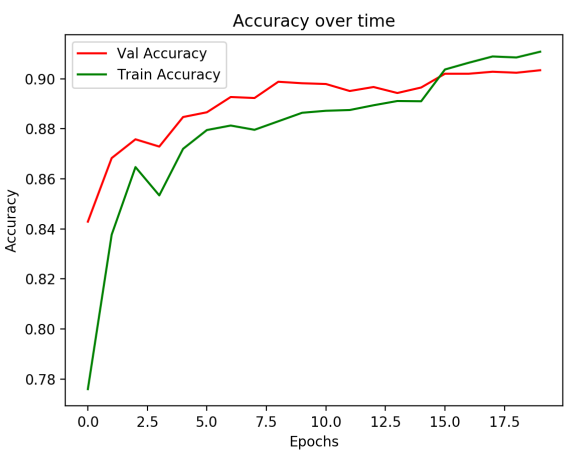

Figure 6: Training: Accuracy over Epoch

The validation and training accuracies in Fig. 6 tend to stay around the same, suggesting that there is little overfitting.

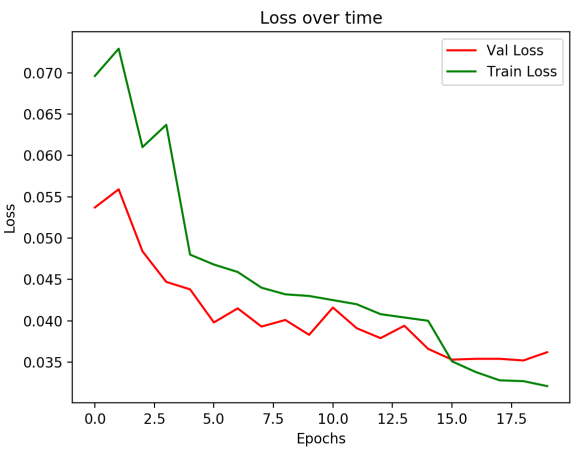

Figure 7: Training: Cross-Entropy Loss over Epoch

Loss. The loss curves in Fig. 7 show a gradual decrease in loss.

Saliency Maps. Saliency maps [36] are a useful technique for visualizing what pixels in the input image are most essential to the classifier in making its prediction. In the saliency maps for this model shown in Fig. 8, we see that the model is able to identify the driver patterns we are searching for in each class. The stop sign saliency maps show that the model is looking for the v pattern. The traffic signal saliency maps show that the model is looking for a normalized v pattern since they also tend to exhibit patterns for constant speed. Finally, the neither case shows that the model is learning multiple cases for neither junctions.

5.4 Test on Palo Alto

In order to evaluate how general the classifier is, we test the classifier in Palo Alto. Palo Alto is a good candidate because it is not as urban as San Francisco, so one might expect a drop in performance of the classifier. Moreover, in our dataset, San Francisco's traffic control elements comprise around 40% stop signs, 20% traffic signals, and 40% neither. Palo Alto, on the other hand, is comprised of around 15% stop signs, 8% traffic signals, and 77% neither. These predictions are then compared against manual curation by human experts.

Data Processing. We followed the same data processing pipeline as before to create bounding boxes around each junction in Palo Alto, collect millions of driver data points, and then assign the data points to each bounding box. This created 3,641 bounding boxes for Palo Alto, which are shown in Fig. 9.

Prediction. We apply the VGG19 classifier trained on the San Francisco data on the Palo Alto data to output confidence scores in each class for each image.

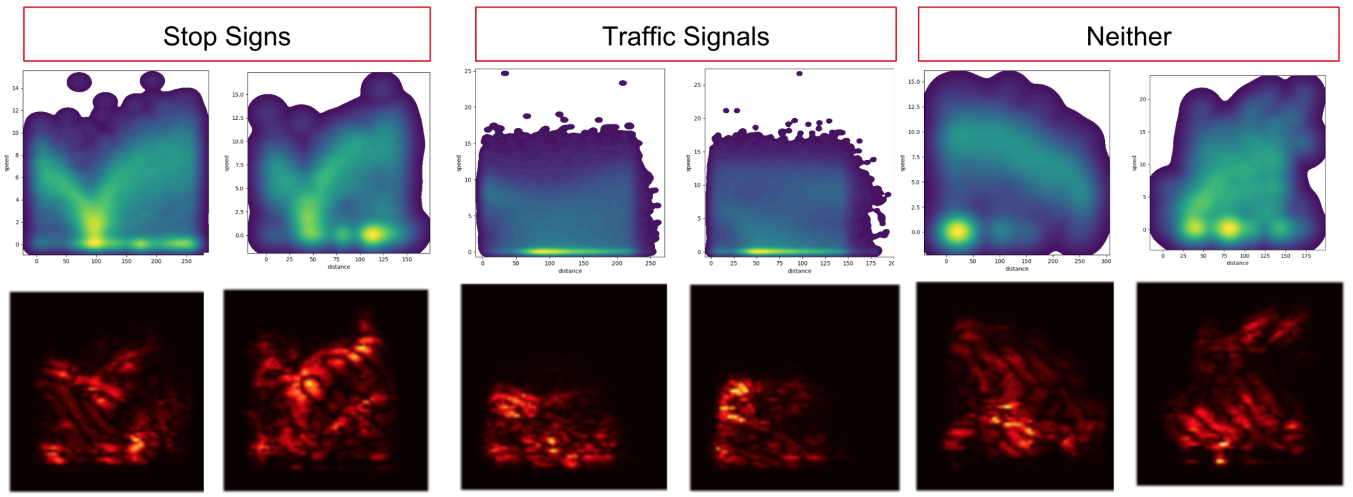

Figure 8: (Top) Examples of the input diagram for training for the three classes. (Bottom) Saliency maps for the three classes

Figure 9: The bounding boxes in Palo Alto. The orange labeling of the boxing boxes signifies that we do not currently know their labels at prediction time.

Evaluation. A 10% quality control check on the human curated ground truth showed 97% accuracy. Compared to the ground truth, the San Francisco classifier applied to Palo Alto has a total accuracy of 90.1%. Its precision, recall, and F1 scores are 0.90.
These results are promising because the model's accuracy is similar to that in San Francisco despite Palo Alto's traffic patterns being widely different.

Confidence Thresholding. We can further increase the total accuracy and F1 score if we threshold by confidence as shown in Table 1. The confidence threshold t only keeps data points that the model assigns a confidence value greater or equal to t. By increasing the confidence threshold, we are able to increase total accuracy. However, this comes at the cost of decreasing the coverage of the dataset.

If we choose a confidence threshold of $t = 80\%$, for example, the model achieves a total accuracy of 94.874% and it covers 82.10% of the bounding boxes we provide.

Should we want to contribute our results to the open source community, by thresholding, we can reduce the load on manual curation to only samples that the model is very confident in.

Accuracies for each class. Table 1 shows that the classifier performs best at predicting neither junctions. This could be because the San Francisco dataset had a large proportion of neither junctions. In fact, the model was able to predict all yield signs it encountered as *neither*. We are confident that with more stop sign and traffic signal training data, the classifier can perform better in those classes as well.

5.5 Incorrect Predictions

We look at a random sample of diagrams and their predictions and find that the incorrect predictions can be grouped into three categories: label limitations, outliers, and lack of data.

5.5.1 Label Limitations. While the San Francisco open dataset covers many traffic control elements, there are still some limitations to using it. For example, it does not label implied stops, such as when a minor street intersects with a major street. Our classifier often predicts this as having a stop sign, since the driver at the minor street exhibits stop-sign-like behavior. This can explain why the stop sign accuracy is lowest, while it is an effectively correct prediction.

5.5.2 Outliers. As shown in Fig. 10, some junctions have outliers that skew the histograms and thus obscure the driver patterns. This is resolved by filtering out extreme points.

Palo Alto Metrics								
Confidence Threshold t	Coverage	Total Accuracy	Stop Sign Accuracy	Traffic Signal Accuracy	Neither accuracy	Average Precision	Average Recall	Average F1 Score
0%	100%	90.11%	74.63%	81.493%	96.86%	0.90	0.90	0.90
20%	100%	90.11%	74.63%	81.493%	96.86%	0.90	0.90	0.90
50%	98.38%	90.96%	75.63%	83.20%	97.38%	0.91	0.91	0.91
80%	82.10%	94.87%	82.46%	88.42%	98.66%	0.95	0.95	0.95
85%	77.19%	95.58%	85.26%	89.03%	98.97%	0.96	0.96	0.96
90%	66.06%	96.61%	89.29%	89.13%	99.18%	0.97	0.97	0.97
95%	46.19%	97.26%	92.02%	88.69%	99.27%	0.97	0.98	0.97
99%	12.63%	98.83%	91.30%	91.67%	100%	0.99	0.99	0.99

Table 1: Palo Alto Confidence Thresholding

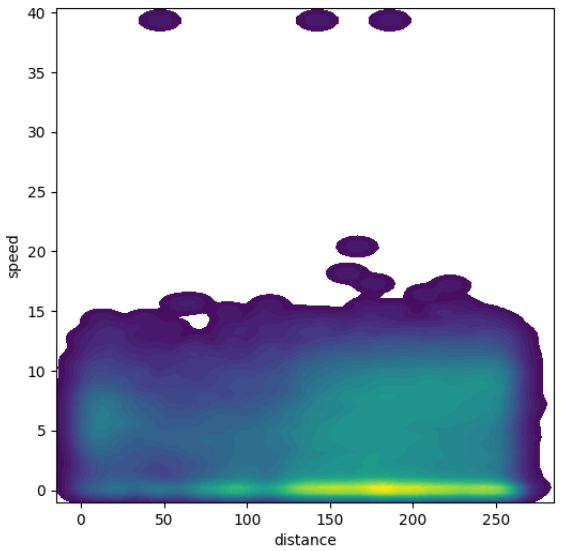

Figure 10: A few data points at high speeds are causing skew

5.5.3 Insufficient Number of Data Points. Some histograms do not show a clear pattern and these tend to have fewer data points. We can resolve this by raising the lower bound on number of data points per bounding box. This can also be corrected by the first improvement that we propose in the next section in order to reduce the noise level, and therefore the need to collect a larger amount of data point/

6 FUTURE WORK AND CONCLUSION

So far, accurate and thorough data on traffic control elements has been incorporated into digital maps by using street-level observations, usually from human beings or cameras. We prove in this paper that large-scale telemetry data can be used in conjunction with deep learning to infer traffic control elements for automated map updates, as demonstrated by the high accuracy and F1 score in a region different from the region where the model was trained.

This is a promising step forward into wide-scale traffic control element detection. However, we believe that further improvements can be made.

First and foremost, the most direct improvement of this work is to use map-matched drivers locations. A map-matching algorithm [28, 30] takes as input the map of the road network and a sequence of possibly sparse and noisy locations, and outputs a trajectory on the road network. By map-matching each ride of the drivers, we can attribute a location in the road network for each driver location (mostly collected from the GPS unit of the drivers' smartphones). That would solve two problems: 1) Our diagram-based approach will then successfully work in places with overlapping roads, short roads and urban canyon, where the GPS location is notoriously less accurate, as we will directly collect map-matched locations data on a given road segment, and not noisy GPs locations using bounding boxes. All kind of road segments will therefore be captured. 2) Because the data collection is no longer done in a bounding box for each road segment, but directly on each road segment, the size and shape of bounding boxes being arbitrary will no longer be an issue. We would also no longer encounter the possible issue of overlapping bounding boxes. We would also be able to apply our current algorithm in cities with a large amount of curved roads (e.g., Paris).

Another important improvement of our model would be to add more sensory input to the model, such as acceleration. Road type and neighborhood type should also be added as inputs to the model, as they are often correlated with the type of traffic control elements.

This paper only explores three classes, but there are other types of traffic control elements — for example, yield signs — that we are interested in. We found that the model correctly interpreted all yield signs as *neither* in Palo Alto. In future work, we may add *yield* as another class.

Taking into account the hour of the day is another important improvement as some traffic control elements change their behavior at certain hours. For example, some traffic signals blink red in the middle of the night instead of following the green-yellow-red cycle. This suggests a *stop sign* behavior at those hours.

We may also fuse this approach with street imagery data to improve the confidence of our predictions.

The bounding boxes were generated after filtering out junctions with more than 4 segments and junctions that are too close to each other. In order to build a more inclusive model, creating bounding boxes of adaptive sizes that change their geometry based on each road segment's geometry is another potential improvement. This way, we can cover all roads and junctions.

In summary, large-scale telemetry vehicle data, collected by large logistics, routing and ride-hailing companies, combined with deep learning, can be used to detect various types of traffic control elements. This paper paves the way for robust updates of digital maps at scale.

ACKNOWLEDGMENTS

The authors are grateful for the fruitful interactions with their Mapping coworkers at Lyft, especially Barak Michener, Yuanyuan Pao, Timothy Brathwaite, Marie Douriez and RenÃľe Park. We also thank the Lyft Data Curation Team lead by Alex Kazakova for helping us assess the accuracy of our algorithm. We also thank the City of San Francisco for releasing and updating their traffic control element datasets, and the OSM community for the formidable digital map that Open Street Map has become over the years.

REFERENCES

[1] Favyen Bastani, Songtao He, Sofiane Abbar, Mohammad Alizadeh, Hari Balakrishnan, Sanjay Chawla, and Sam Madden. 2018. Machine-assisted Map Editing. In *Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL '18)*. ACM, New York, NY, USA, 23–32. https://doi.org/10.1145/3274895.3274927

[2] Favyen Bastani, Songtao He, Mohammad Alizadeh, Hari Balakrishnan, Samuel Madden, Sanjay Chawla, Sofiane Abbar, and David DeWitt. 2018. RoadTracer: Automatic Extraction of Road Networks from Aerial Images. In *Computer Vision and Pattern Recognition (CVPR)*. Salt Lake City, UT.

[3] James Biagioni and Jakob Eriksson. 2012. Inferring Road Maps from Global Positioning System Traces: Survey and Comparative Evaluation. *Transportation Research Record* 2291, 1 (2012), 61–71. https://doi.org/10.3141/2291-08

[4] James Biagioni and Jakob Eriksson. 2012. Map Inference in the Face of Noise and Disparity. In *Proceedings of the 20th International Conference on Advances in Geographic Information Systems (SIGSPATIAL '12)*. ACM, New York, NY, USA, 79–88. https://doi.org/10.1145/2424321.2424333

[5] R. Bruntrup, S. Edelkamp, S. Jabbar, and B. Scholz. 2005. Incremental map generation with GPS traces. In *Proceedings. 2005 IEEE Intelligent Transportation Systems, 2005.* 574–579. https://doi.org/10.1109/ITSC.2005.1520084

[6] Alexander V. Buslaev, Selim S. Seferbekov, Vladimir I. Iglovikov, and Alexey A. Shvets. 2018. Fully Convolutional Network for Automatic Road Extraction From Satellite Imagery. In *CVPR Workshops*.

[7] R. Carisi, E. Giordano, G. Pau, and M. Gerla. 2011. Enhancing in vehicle digital maps via GPS crowdsourcing. In *2011 Eighth International Conference on Wireless On-Demand Network Systems and Services*. 27–34. https://doi.org/10.1109/WONS.2011.5720196

[8] Chen Chen, Cewu Lu, Qixing Huang, Qiang Yang, Dimitrios Gunopulos, and Leonidas Guibas. 2016. City-Scale Map Creation and Updating Using GPS Collections. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*. ACM, New York, NY, USA, 1465–1474. https://doi.org/10.1145/2939672.2939833

[9] Cynthia Chen, Jingtao Ma, Yusak Susilo, Yu Liu, and Menglin Wang. 2016. The promises of big data and small data for travel behavior (aka human mobility) analysis. *Transportation research part C: emerging technologies* 68 (2016), 285–299.

[10] G. Cheng, Y. Wang, S. Xu, H. Wang, S. Xiang, and C. Pan. 2017. Automatic Road Detection and Centerline Extraction via Cascaded End-to-End Convolutional Neural Network. *IEEE Transactions on Geoscience and Remote Sensing* 55, 6 (June 2017), 3322–3337. https://doi.org/10.1109/TGRS.2017.2669341

[11] C. Y. Goh, J. Dauwels, N. Mitrovic, M. T. Asif, A. Oran, and P. Jaillet. 2012. Online map-matching based on Hidden Markov model for real-time traffic sensing applications. In *2012 15th International IEEE Conference on Intelligent Transportation Systems*. 776–781. https://doi.org/10.1109/ITSC.2012.6338627

[12] T. Guo, K. Iwamura, and M. Koga. 2007. Towards high accuracy road maps generation from massive GPS Traces data. In *2007 IEEE International Geoscience and Remote Sensing Symposium*. 667–670. https://doi.org/10.1109/IGARSS.2007.4422884

[13] M. Haklay and P. Weber. 2008. OpenStreetMap: User-Generated Street Maps. *Pervasive Computing* 7, 4 (Oct. 2008), 12–18. https://doi.org/10.1109/MPRV.2008.80

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[15] Songtao He, Favyen Bastani, Sofiane Abbar, Mohammad Alizadeh, Hari Balakrishnan, Sanjay Chawla, and Sam Madden. 2018. RoadRunner: Improving the Precision of Road Network Inference from GPS Trajectories. In *Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL '18)*. ACM, New York, NY, USA, 3–12. https://doi.org/10.1145/3274895.3274974

[16] Juan C. Herrera, Daniel B. Work, Ryan Herring, Xuegang (Jeff) Ban, Quinn Jacobson, and Alexandre M. Bayen. 2010. Evaluation of traffic data obtained via GPS-enabled mobile phones: The Mobile Century field experiment. *Transportation Research Part C: Emerging Technologies* 18, 4 (2010), 568 – 583. https://doi.org/10.1016/j.trc.2009.10.006

[17] Jiuxiang Hu, Anshuman Razdan, John C. Femiani, Ming Cui, and Peter Wonka. 2007. Road network extraction and intersection detection from aerial images by tracking road footprints. *IEEE Transactions on Geoscience and Remote Sensing* 45, 12 (12 2007), 4144–4157. https://doi.org/10.1109/TGRS.2007.906107

[18] Shaohan Hu, Lu Su, Hengchang Liu, Hongyan Wang, and Tarek F. Abdelzaher. 2015. SmartRoad: Smartphone-Based Crowd Sensing for Traffic Regulator Detection and Identification. *ACM Trans. Sen. Netw.* 11, 4, Article 55 (July 2015), 27 pages. https://doi.org/10.1145/2770876

[19] Timothy Hunter, Pieter Abbeel, and Alexandre Bayen. 2014. The Path Inference Filter: Model-Based Low-Latency Map Matching of Probe Vehicle Data. *IEEE Transactions on Intelligent Transportation Systems* 15, 2 (April 2014), 507–529. https://doi.org/10.1109/TITS.2013.2282352

[20] Timothy Hunter, Ryan Herring, Pieter Abbeel, and Alexandre Bayen. 2009. Path and travel time inference from GPS probe vehicle data. *NIPS Analyzing Networks and Learning with Graphs* 12, 1 (2009).

[21] Timothy Hunter, Aude Hofleitner, Jack Reilly, Walid Krichene, Jerome Thai, Anastasios Kouvelas, Pieter Abbeel, and Alexandre Bayen. 2013. Arriving on time: estimating travel time distributions on large-scale road networks. *arXiv e-prints*, Article arXiv:1302.6617 (Feb. 2013), arXiv:1302.6617 pages. arXiv:cs.LG/1302.6617

[22] Vladimir Iglovikov, Sergey Mushinskiy, and Vladimir Osin. 2017. Satellite Imagery Feature Detection using Deep Convolutional Neural Network: A Kaggle Competition. *arXiv e-prints*, Article arXiv:1706.06169 (June 2017), arXiv:1706.06169 pages. arXiv:cs.CV/1706.06169

[23] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations (ICLR)* (2015).

[24] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. (2012), 1097–1105. http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[25] Yann LeCun, Yoshua Bengio, and Geoffrey E. Hinton. 2015. Deep Learning. *Nature* 521 (2015), 436–444.

[26] Xuemei Liu, James Biagioni, Jakob Eriksson, Yin Wang, George Forman, and Yanmin Zhu. 2012. Mining Large-scale, Sparse GPS Traces for Map Inference: Comparison of Approaches. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '12)*. ACM, New York, NY, USA, 669–677. https://doi.org/10.1145/2339530.2339637

[27] Gellert Mattyus, Wenjie Luo, and Raquel Urtasun. 2017. DeepRoadMapper: Extracting Road Topology From Aerial Images. In *The IEEE International Conference on Computer Vision (ICCV)*.

[28] James Murphy and Yuanyuan Pao. 2018. Map matching when the map is wrong: Efficient vehicle tracking on- and off-road for map learning. *arXiv e-prints*, Article arXiv:1809.09755 (Sept. 2018), arXiv:1809.09755 pages. arXiv:math.OC/1809.09755

[29] James Murphy, Yuanyuan Pao, and Asif Haque. 2017. Image-based Classification of GPS Noise Level Using Convolutional Neural Networks for Accurate Distance Estimation. In *Proceedings of the 1st Workshop on Artificial Intelligence and Deep Learning for Geographic Knowledge Discovery (GeoAI '17)*. ACM, New York, NY, USA, 10–13. https://doi.org/10.1145/3149808.3149811

[30] Paul Newson and John Krumm. 2009. Hidden Markov Map Matching Through Noise and Sparseness. In *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS '09)*. ACM, New York, NY, USA, 336–343. https://doi.org/10.1145/1653771.1653818

[31] SF OpenData. 2018. Stop Signs. https://data.sfgov.org/Transportation/Stop-Signs/ddtz-jevd/data

[32] SF OpenData. 2018. Traffic Signals. https://data.sfgov.org/Transportation/Traffic-Signals/4abk-vggy

[33] Zhangqing Shan, Hao Wu, Weiwei Sun, and Baihua Zheng. 2015. COBWEB: A Robust Map Update System Using GPS Trajectories. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp '15)*. ACM, New York, NY, USA, 927–937. https://doi.org/10.1145/2750858.2804286

[34] Hoo-Chang Shin, Holger Roth, Mingchen Gao, Le Lu, Ziyue Xu, Isabella Nogues, Jianhua Yao, Daniel J. Mollura, and Ronald M. Summers. 2016. Deep Convolutional Neural Networks for Computer-Aided Detection: CNN Architectures, Dataset Characteristics and Transfer Learning. *IEEE Transactions on Medical Imaging* 35 (2016), 1285–1298.

[35] B. W. Silverman. 1986. *Density Estimation for Statistics and Data Analysis*. Chapman & Hall, London.

[36] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. 2013. Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps. *CoRR* abs/1312.6034 (2013). arXiv:1312.6034 http://arxiv.org/abs/1312.6034

[37] Karen Simonyan and Andrew Zisserman. 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. *CoRR* abs/1409.1556 (2014). arXiv:1409.1556 http://arxiv.org/abs/1409.1556

[38] Rade Stanojevic, Sofiane Abbar, Saravanan Thirumuruganathan, Sanjay Chawla, Fethi Filali, and Ahid Aleimat. 2017. Kharita: Robust Map Inference using Graph Spanners. *CoRR* abs/1702.06025 (2017). arXiv:1702.06025 http://arxiv.org/abs/1702.06025

[39] D. Wang, T. Abdelzaher, L. Kaplan, R. Ganti, S. Hu, and H. Liu. 2014. Exploitation of Physical Constraints for Reliable Social Sensing. In *2013 IEEE 34th Real-Time Systems Symposium(RTSS)*, Vol. 00. 212–223. https://doi.org/10.1109/RTSS.2013.29

[40] Pu Wang, Timothy Hunter, Alexandre M. Bayen, Katja Schechtner, and Marta C. González. 2012. Understanding Road Usage Patterns in Urban Areas. *Scientific Reports* 2 (20 12 2012), 1001 EP –. https://doi.org/10.1038/srep01001