Project Report

Topic - Automatic Extraction of Road Networks from Aerial Images

Members - Shriya Rai, Rahul Bhojwani

Problem Statement

The challenge we look to tackle in our project is the Automatic Extraction of Road Networks from Aerial Images. The problem has traditionally been viewed as an image semantic segmentation problem where pixels from aerial images are classified into “road” or “non-road” class which is then utilized as an input for the generation of road networks in the post-processing stage. However, our aim is to generate a road network directly from the aerial images. This approach is based on the paper - *“RoadTracer: Automatic Extraction of Road Networks from Aerial Images”.* Our task is two-fold where we implement the approach proposed in the paper on the paper’s baseline test cities to replicate and validate the results illustrated along with new test cities that are characteristically different from the original test cities.
Acquiring accurate road map data has become essential in this day and age given that location-based services are omnipresent and crucial for daily lives. Self-driving cars are no longer a thing of the future and accurately mapped road networks can be a critical guiding resource. Road maps data can also prove pivotal in efforts to get resources to lesser developed areas.
However, the creation of road maps is a time consuming, expensive, and often a labor-intensive process. Current methods to map roads automatically are incredibly error-prone and are not feasible to be implemented on a large scale especially given their low viability. Because accurately capturing the local topology is crucial for a number of applications like navigation, the performance shown in the paper suggests that RoadTracer can prove an important step forward in fully automating map construction from aerial images.

Dataset

The authors of the above-mentioned paper evaluate their approach using aerial images covering 24 square km areas of 15 cities. The model is trained in 25 other cities. We have evaluated the approach on 10 of the original test cities - Amsterdam, Chicago, Denver, Los Angeles, Montreal, Paris, Pittsburgh, Salt Lake City, Tokyo, Toronto, and 4 new cities which vary in features from the original test cities. The new cities are Nairobi, Port au Prince, Chandigarh, and New Delhi.

Aerial Imagery - Google Maps:

The dataset spans 39 cities - 25 cities for training, 10 original test cities and 4 new test cities. For each city, the dataset covers a region of approximately 24 sq km around the city center. The satellite imagery is obtained using go scripts by leveraging Google Maps API at 60 cm/pixel resolution.

Road Network - Open Street Maps:
OSM provides us with the ground truth for the road networks. The OSM data is processed such that we have data pertaining to the 39 cities. The road networks for the same are extracted. The coordinate system of the road network is converted such that the vertex spatial coordinate annotations correspond to pixels in the satellite images. Certain non-roads that appear in OSM such as pedestrian paths and parking lots are excluded.

Previous approaches:

In the previous state of the art approaches for solving the problem in question, deep learning techniques are majorly employed for the initial segmentation of the aerial images. This is followed by an algorithm that post processes the output from the segmentation step like resolving missing connections in the extracted road topology by utilizing the shortest path solution.

- Segmentation -The task is viewed as an image semantic segmentation problem where pixels from aerial images are classified into “road” or “non-road” class. The output from this step is further utilized as an input for the generation of road networks in the post-processing stage.
- Deep Road Mapper - This model utilizes the segmentation approach with an added post-processing step to enhance the initial graph by overcoming missing connections via heuristics.

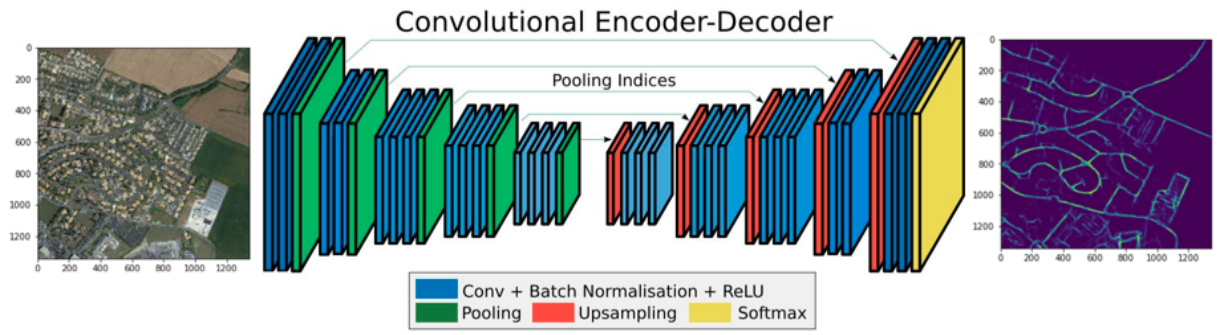


Fig1: Segnet model explaining the segmentation step in the previous state of the art approaches.

Semantic Segmentation step:
Encoder-Decoder network that takes the image as input, applies convolution to learn features and deconvolution to get the output as the segmentation map of the size of the image. Common models include: Segnet, U-net, DeepLab-v1,2,3.’]

Post-processing step:

Road pixels from the segmentation map are further used to generate the road network. Drawbacks in the previous approaches:

- As 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 from the road centerlines, and where these centerlines should be connected.
- The efficiency of post-processing decreases due to added uncertainty in the segmentation step.

Proposed approach (Iterative Graph Construction Algorithm)

The main advancement in the proposed approach is that the model generates the road graph network directly from images instead of generating a map and then post-processing. This improves the model multifold by increasing the accuracy along with the robustness of the method.

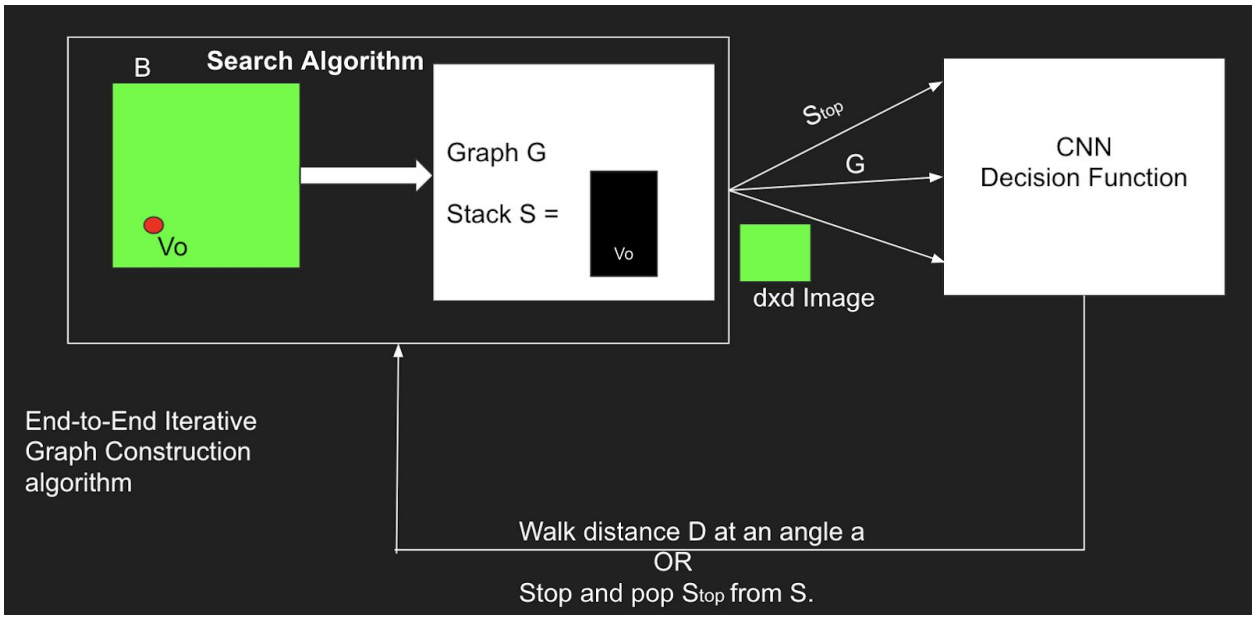


Fig 2: Roadtracer-Iterative graph construction algorithm.

The figure above explains the working of the iterative graph construction algorithm. It can be divided into two key components:

1. Search Algorithm: This part takes in the input image and a starting location on the image and maintains the constructed graph so far, G and the stack of all the vertices in the exploration phase. It passes three things to the decision function:
    a. The top vertex on the stack, S_{top}.
    b. The constructed graph G so far.
    c. A d x d cropped image centered at S_{top}.

2. Decision Function: This is the core part of the algorithm that is taking the decision on how to move along the graph. The underlying model used for the decision function is a 17-layered CNN model. The output of this function is 2-fold.
    a. Action(Walk or Stop): This is the output to decide whether the algorithm should walk to the next edge from the vertex or stop walking based on the threshold value set. This is achieved by applying a 2 neuron fully connected layer after the features from the 17th layer and applying Softmax activation function and cross-entropy loss function.
    b. Angle(Which angle to walk-in): After the value of Owalk is above a threshold, the model uses a 64 neuron fully connected layer over the 17th layer output, followed by Sigmoid activation function and RMSE loss function to get the direction of walk ranging between [0,2pi).

    Another novel part in this CNN decision function is that instead of taking in the normal image as input, it takes in the image with the current graph G rendered on it. This makes CNN be aware of the path that has been explored so far and makes the algorithm more robust. The image below explains this working of the CNN decision function.

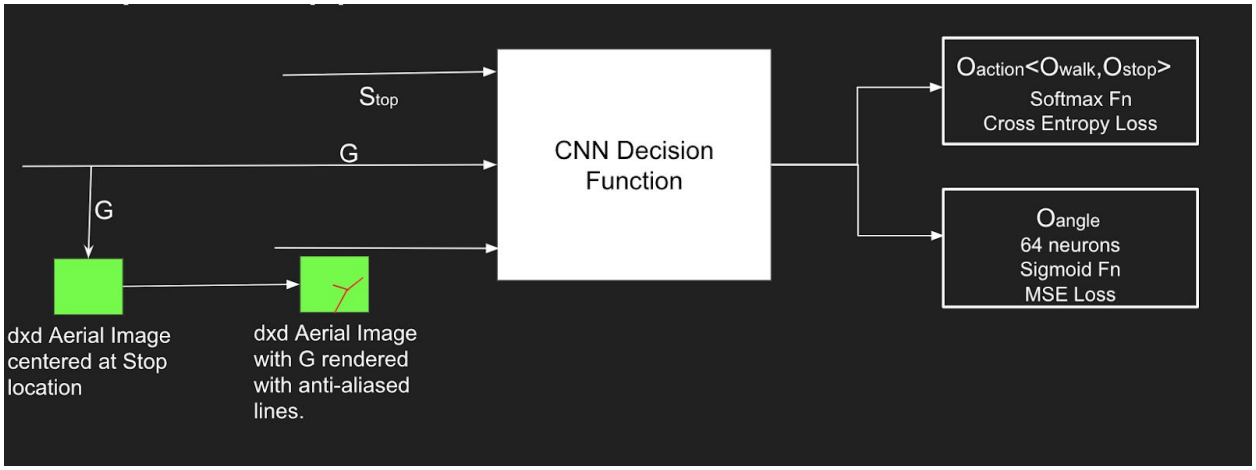


Fig 3: CNN decision function

Implementation

Technology Stack:

We utilise AWS architecture for our implementation. The instance used is EC2 wich 125GB disk space(gpt2 volume) The RAM required varies from 16GB to 128GB for computing-intensive steps. Python2 on Anaconda and TensorFlow v1.14 are employed for model code.

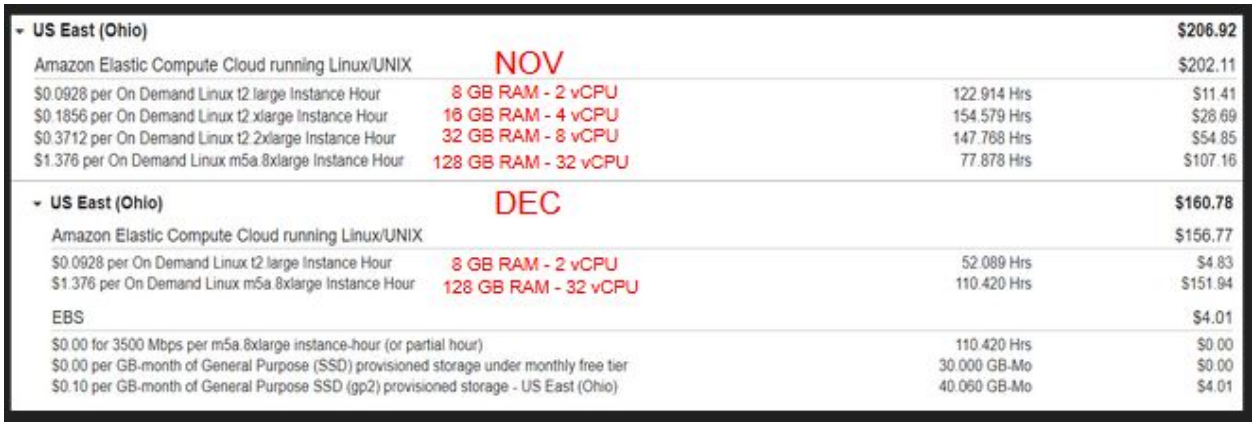


Fig 4: Our tech stack for project implementation on AWS console

Data Acquisition and Processing:

The main task of data acquisition and processing can be done to smaller tasks as follows :

- Subtask 1 : Acquiring satellite imagery data through google map API
  Google Static Maps API Images data with 60cm/pixel resolution which spanned 39 cities was downloaded. This step required the creation of a Google Console account to access Google Maps Static API.
- Subtask 2 : Acquiring and processing OSM data
  Open source OSM data was first downloaded. Then the road network graph are extracted from the downloaded OSM data. This step also prunes the data to extract data only pertaining to the cities in question Pruning is also done to get only the motorways and tunnels. Conversion of coordinate system of the road network is done so that the vertex spatial coordinate annotations correspond to pixels in the satellite images. The main challenge in this step was the size of the downloaded data which was 48 GB. This required us to increase the volume of EC2 instance. The RAM was also required to be increased to 128GB as road network graph extraction was a computing heavy step. We tried varied RAM sizes (16,32,64 and 128) to settle on 128GB finally as it was the only viable solution given the time constraint.
- Subtask 3 : Generate starting locations
  This step generates a set of seed vertices for the algorithm to utilize.
- Subtask 4 : Generate road masks
  The graphs created via OSM data (subtask 2) are utilized to generate ground truth masks.
- Subtask 5 : Create test satellite imagery files
  Test tiles consisting of four imagery tiles of a particular city are merged together. Authors utilized python 2 (code was written 2 years ago) as opposed to the popular Python 3. Because of the above reason, we had issues migrating the code to Python 3 due to multiple deprecated library functions(Eg: Using TensorFlow 1.14 instead of 2.0).

Model implementation Pipeline:

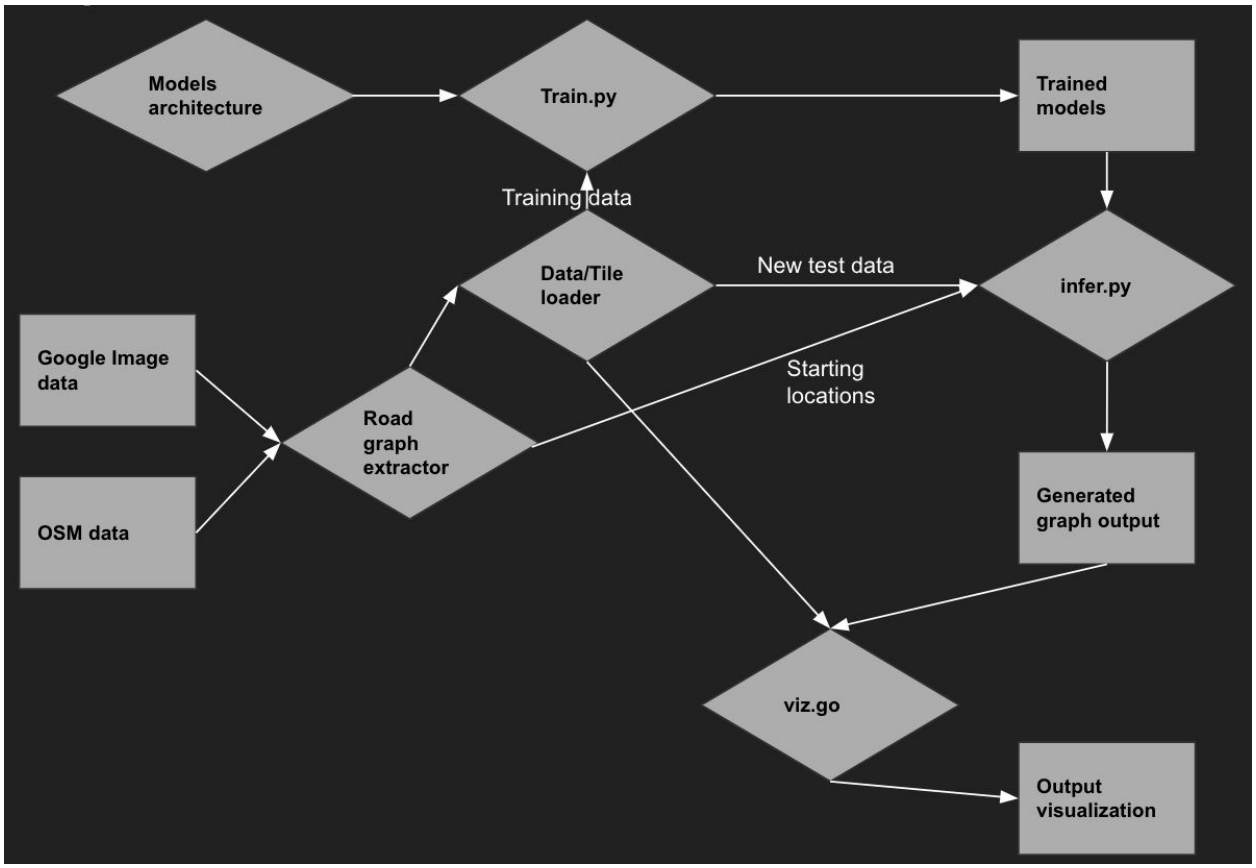


Fig 5: The model implementation pipeline

The image above explains the pipeline for the implementation of this project.

Model Training:

The right method for training the CNN decision function is very critical here. Two different kinds of training methods were explored:

- Static Training:
    - For each training example, model samples a region (v0, B) and a step count n, and initializes a search.
    - The Model generates n-next steps of the search using the “oracle” decision function.
    - Problem:
        - The entire training set is generated using the correct graph at each step
        - This makes the algorithm unable to generalize as it’s not prone to noise.
- Dynamic Training:
    - Similar to static training, the model generates the next training step using the “oracle” decision function.

- However, the input graph is the one generated by CNN decision instead of the “oracle” decision.
- This helps the model learn the right noise and hence it generalizes better on the prediction.

The model training was an expensive step in our implementation. With the 128 GB memory AWS(m5a.8xlarge) it took around 6 full days to finish only 114 epochs of training. Below we have plotted the graph for the first 8 epochs of the training process. We have also reported the results after the 114th epoch below.

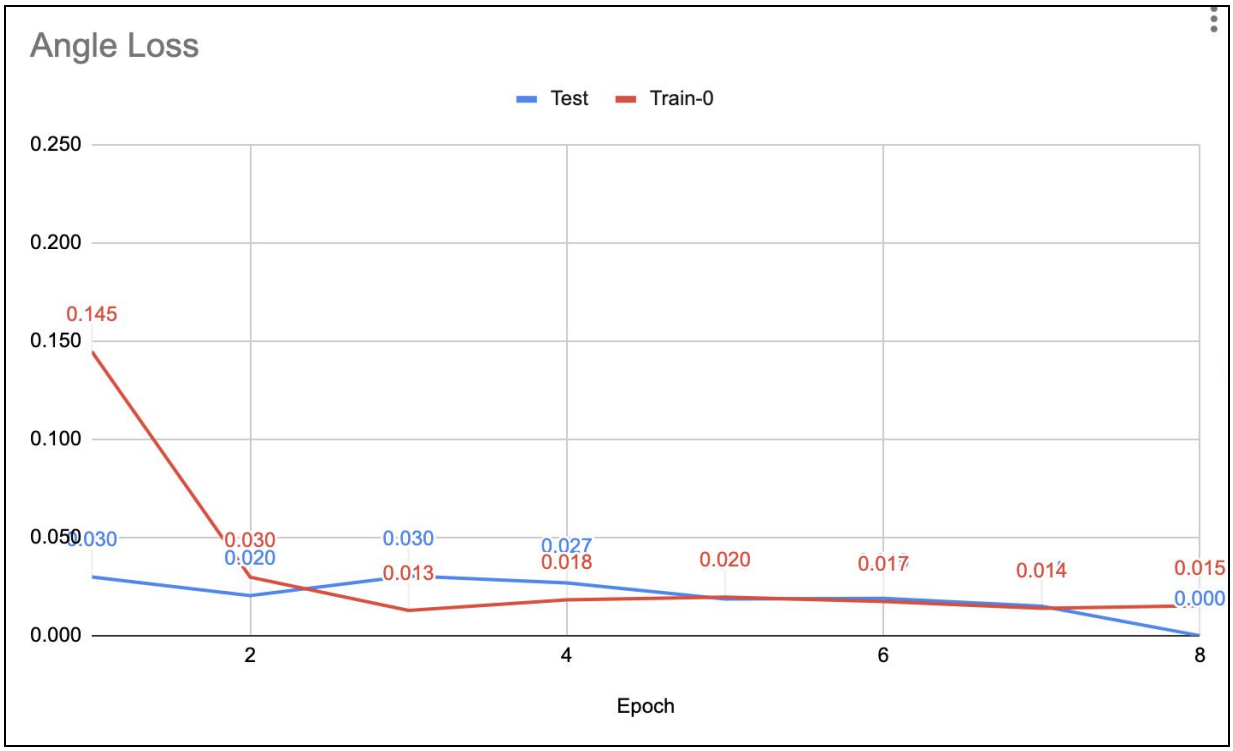


Fig 6: The train-test curve for the angle loss output from the decision function.**

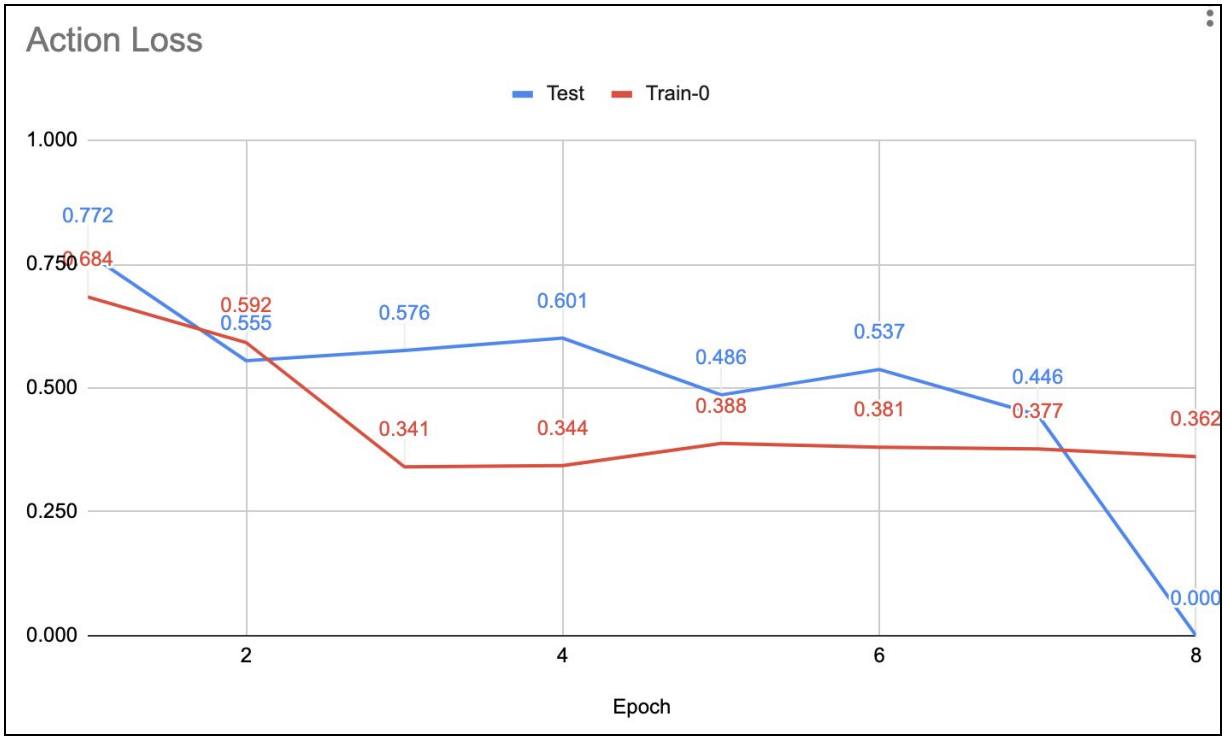


Fig 7: The train-test curve for the action loss output from the decision function.**

***Note*: In fig 6 and fig 7, the test value for the 8th epoch was not computed during the generation of the figure and hence it shows 0. Those values are actually missing(Null) and not 0.

After the 114th epoch, the *test action-loss* dropped to **0.18** and the *test angle-loss* dropped to **0.004**.

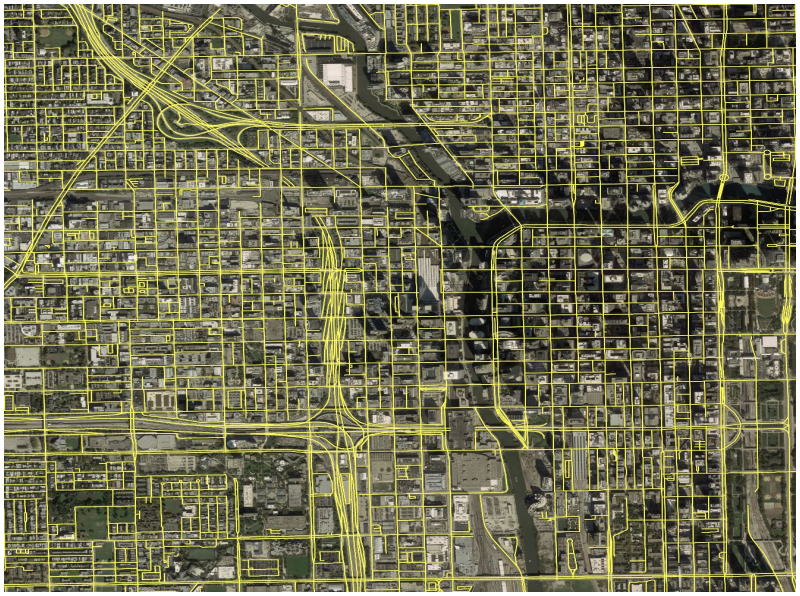
Evaluation Metric:

The measure utilised for performance evaluation is **Junction Metric** which is a measure that the authors of the paper proposed. It compares the ground truth and inferred maps junction-by-junction, where a junction is any vertex with three or more edge.
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, $f_{v,correct}$ is the fraction of incident edges of v that are captured around u, and $f_{u,error}$ is the fraction of incident edges of u that appear around v. Foreach unpaired ground truth junction v, $f_{v,correct} = 0$, and for each unpaired inferred map junction u, $f_{u,error} = 1$. Finally, if $ncorrect = \sum f_{v,correct}$ and $nerror = \sum f_{u,error}$, we report the correct junction fraction $Fcorrect = ncorrect / \#\ junctions\ in\ G*$ and error rate $Ferror = nerror / (nerror+ncorrect)$. The reasoning behind employing this measure was that it provides a score that is representative of the inferred map's practical usability and is also interpretable. The measure also yields a precision-recall curve.

Result

Results on Baseline Cities for Comparative Evaluation:

We implement the approach on 10 of the original test cities - Amsterdam, Chicago, Denver, Los Angeles, Montreal, Paris, Pittsburgh, Salt Lake City, Tokyo, Toronto. Below we visualize the result on Chicago which had the best-predicted road network out of all the other test cities along with its true road network. We have added the results along with their true networks for the remaining 9 original test cities in the appendix.



The image above shows the true road network for Chicago

The image above shows the predicted road network of Chicago

It is our hypothesis that the model performs so well relatively for Chicago as the city has an incredibly grid-like structure.

Results on new cities:

In order to get a better understanding of the usability of the model, we explored it on critical cities of interest which had different characteristics from the training cities and original test cities. We decided on multiple developing cities - Nairobi, Port Au Prince, Delhi and Chandigarh. The reasoning behind this was the paper had primarily focused on developed cities both for training and testing purposes. The cities that we decided on are on their own unique with Nairobi being developing but not as densely populated as the others in the test set. Delhi having an erratic development pattern that has a large number of high-density areas. Port au Prince is a disaster-prone city that is still rebuilding after the infamous earthquake that hit Haiti in 2010. Chandigarh has had an extremely well-planned development and is still in stages of active development.

The image above shows the true road network for Nairobi

The image above shows predicted road network for Nairobi

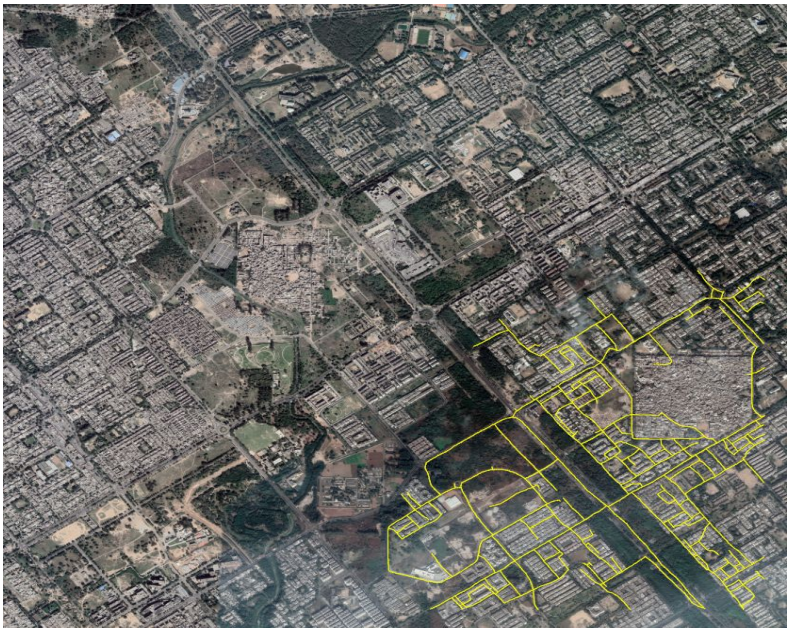
The image above shows true road network of Port Au Prince

The image above shows predicted road network of Port Au Prince

The image above shows the true road network for Delhi

The image above shows the predicted road network for Delhi

The above image shows the true road network for Chandigarh

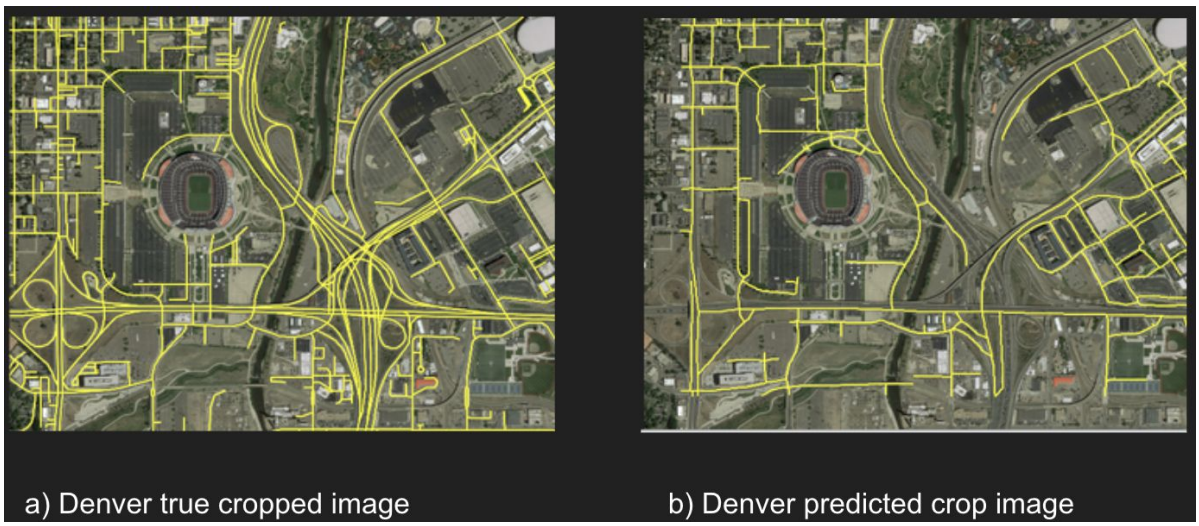


The above image shows the predicted road network for Chandigarh

The results for Nairobi and Port Au Prince are similar to the original test cities. The lack of good results for Delhi might be due to the foggy image as we tried generating the road network by trying out multiple starting locations. However, the changes in the starting location did not seem to affect the result by much. The results for Chandigarh were surprising as we expected Chandigarh to have the best results out of the 4 new test cities as it has a well-planned structure and maps most closely to the training and original test cities in terms of features.

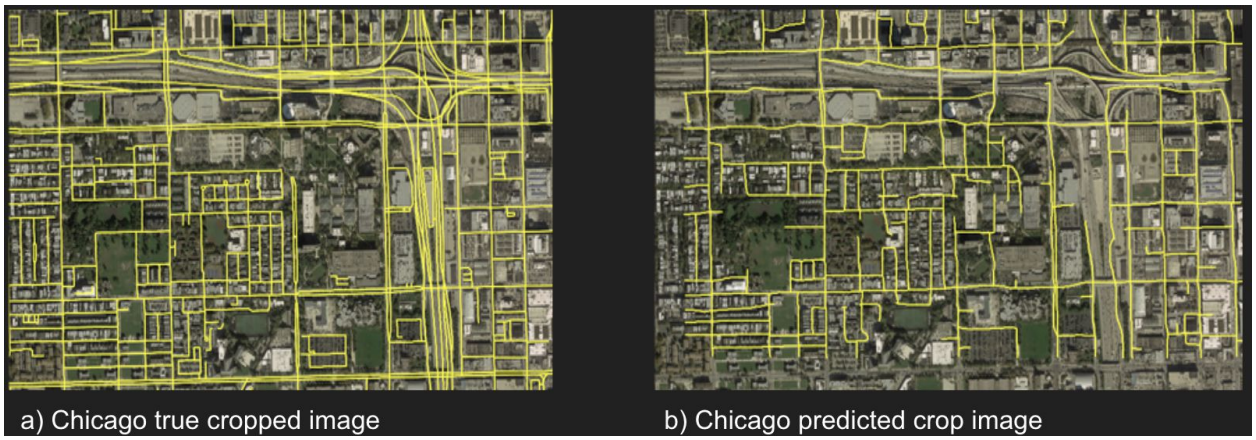
Observations and proposed solutions

Observation-1: Method does not perform well in detecting highways.



a) Denver true cropped image

b) Denver predicted crop image



a) Chicago true cropped image

b) Chicago predicted crop image

Reasoning and proposed solutions:

The highways are either:

- Case 1: A road dividing into multiple roads(lanes) and running parallel to each other.
- Case 2: A highway whose starting and ending points are beyond the scope of the image.

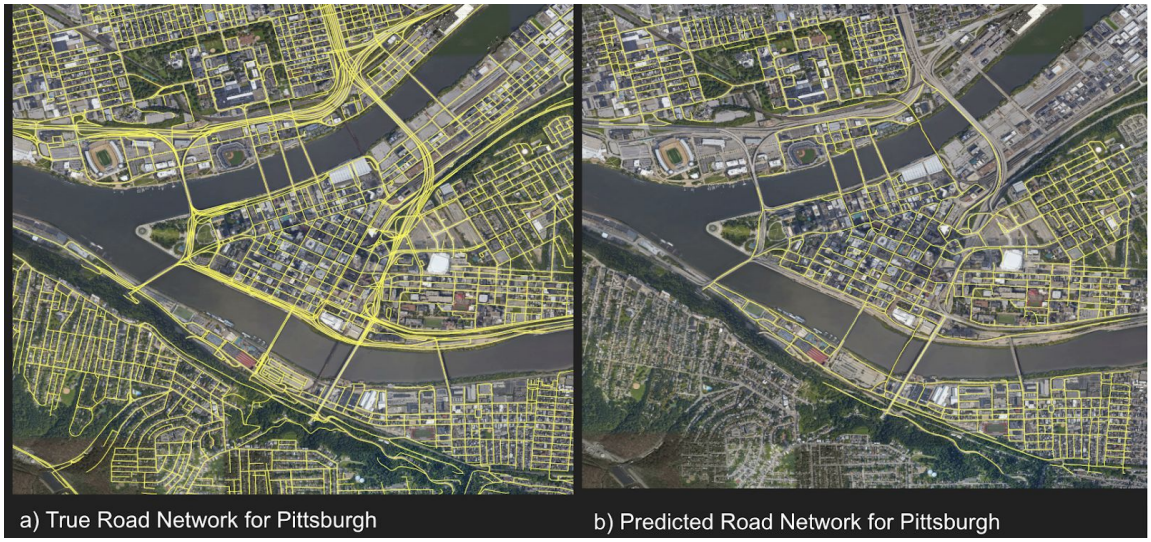
Solution for case 1:

- Due to the underlying concept of the graph-based expansion, the network proceeds for a long time when a highway is met, leading up to the problem of not being able to find parallel edges.
- This can be solved by storing the high confidence score matches separately and tracing them again.

Solution for case 2:

- This one is the most prevailing case and it can be solved by finding a starting point on a highway.
- To achieve that, we can use the segmentation output to detect wider roads[highways] and use a starting location from there.

Observation 2: Method does not perform well in case of disconnected parts in images



a) True Road Network for Pittsburgh b) Predicted Road Network for Pittsburgh

Reasoning and proposed solutions:
This is because the graph-based model cannot explore the disconnected components.

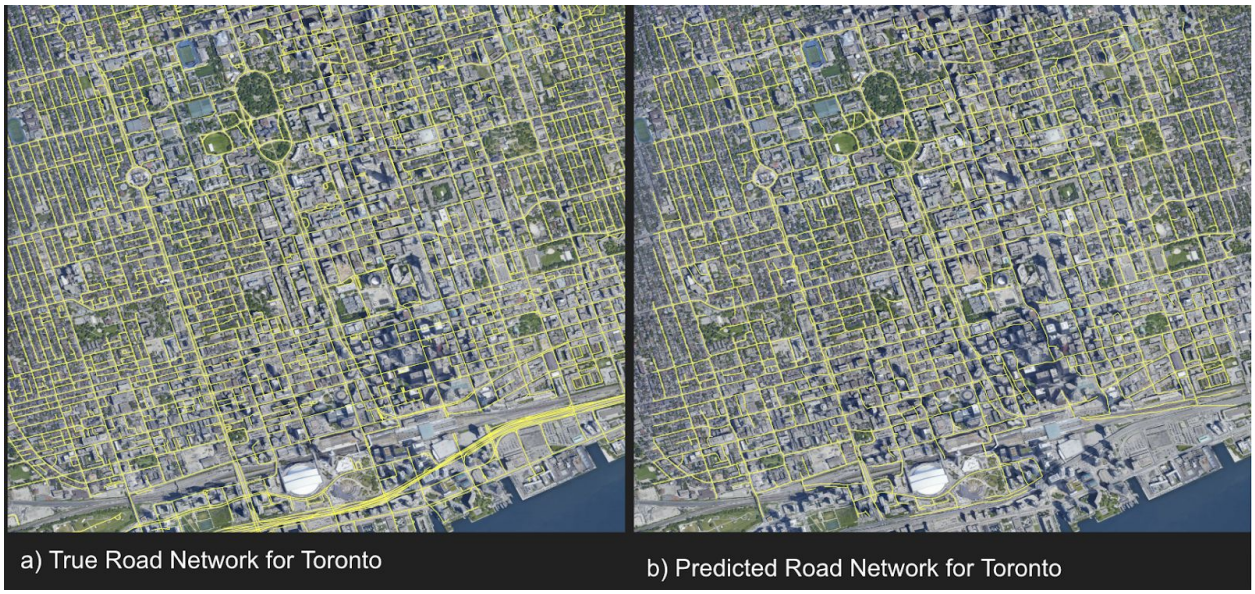
Solution:

- Multiple starting locations can be used to help the model find roads in all disjoint components.
- One optimal way of doing that is by finding the starting location in the area of the image unexplored after each complete pass.

Observation-3

The method shows poor results at image borders

a) True Road Network for Tokyo b) Predicted Road Network for Tokyo



a) True Road Network for Toronto

b) Predicted Road Network for Toronto

Reasoning and proposed solutions:

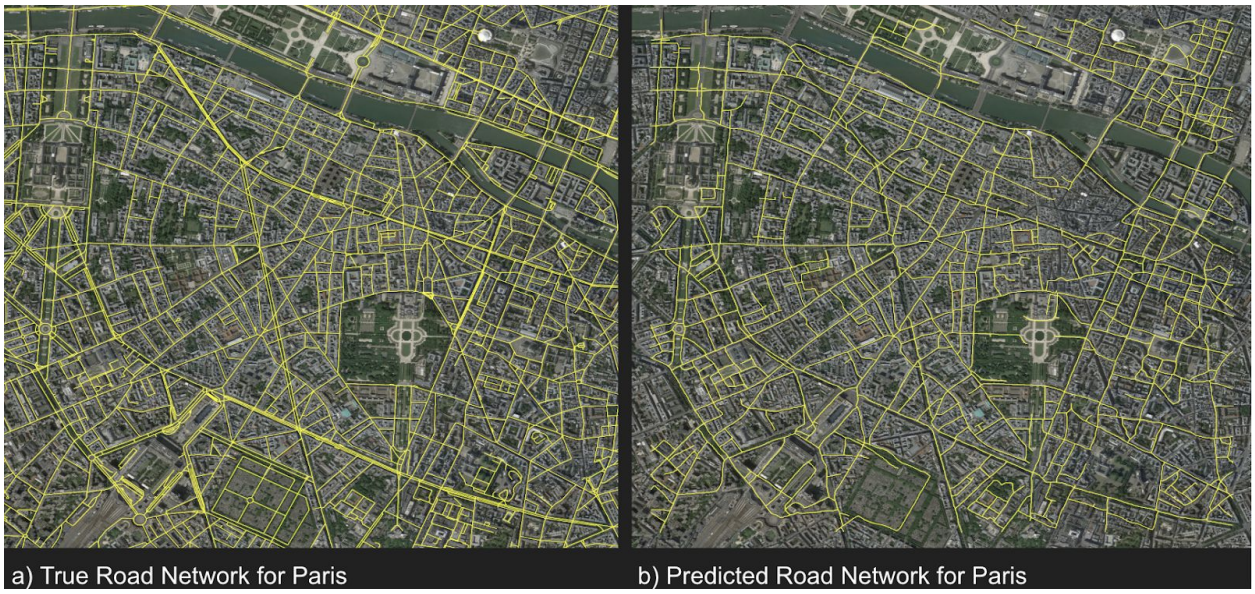
- The current CNN architecture is unable to detect the road-network at image borders because it's using 0 padding at image borders.
- This makes the pattern different(not continuous) for the border kernels to learn from it.

Solution:

- One of the novel methods to solve this is by filling in the values with mirroring the image at the borders.
- And then filter the results beyond the border part.

Observation- 4:

The Method does not perform predictions at circular loops



a) True Road Network for Paris

b) Predicted Road Network for Paris

Reasoning and proposed solutions:

- The algorithm is very susceptible to the distance hyperparameter, D.
- Taking an edge of distance D throws it out of the circle and that's the reason we notice it not working better on the tighter circular roads.

Solution:

- This can be solved by tweaking the hyperparameter, D.
- The problem is that smaller the D, the slower the model will explore/converge.

Key Learnings

- Computational resources have been a major bottleneck for utilizing this massive dataset and we gained a deeper understanding of managing the AWS tech stack.
- Google Maps Static API is very robust and very handy to access images for a required bounding box.
- The data usage and processing of OSM and Imagery to construct the graph.
- The Iterative end to end approach of generating graphs directly from images is very novel and can be used in many other domains/problems.
- The idea of dynamic training and rendering the graph into an image before inputting to CNN also broadens our perspective to expand such techniques in other domains.
- For open-source purposes, stating the hardware requirements and software versions makes the project more robust and streamlined.

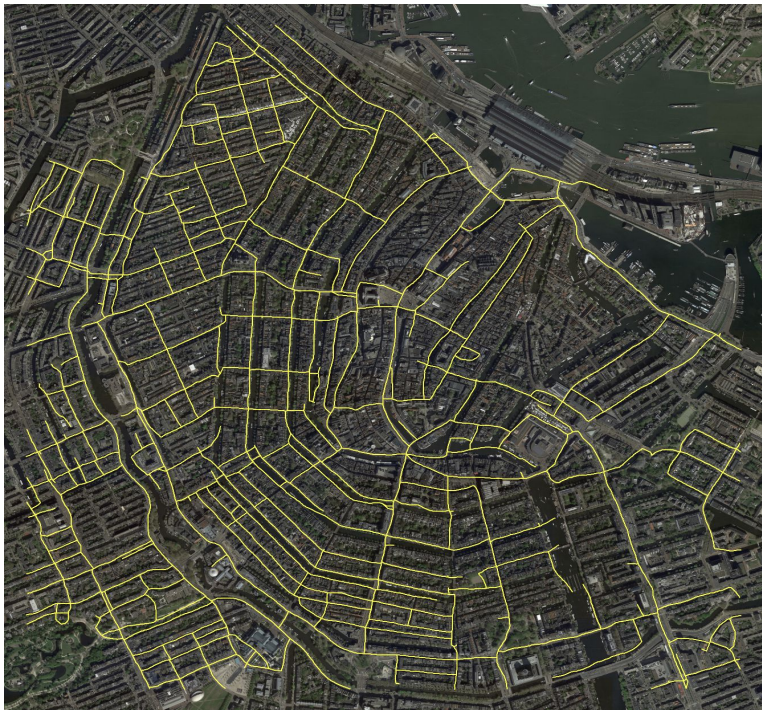
References

1. Favyen Bastani, Songtao He, Sofiane Abbar, Mohammad Alizadeh, Hari Balakrishnan, Sanjay Chawla, Sam Madden, and David DeWitt. Roadtracer: Automatic extraction of road networks from aerial images. arXiv preprint arXiv:1802.03680, 2018.
2. https://github.com/mitroadmaps/roadtracer

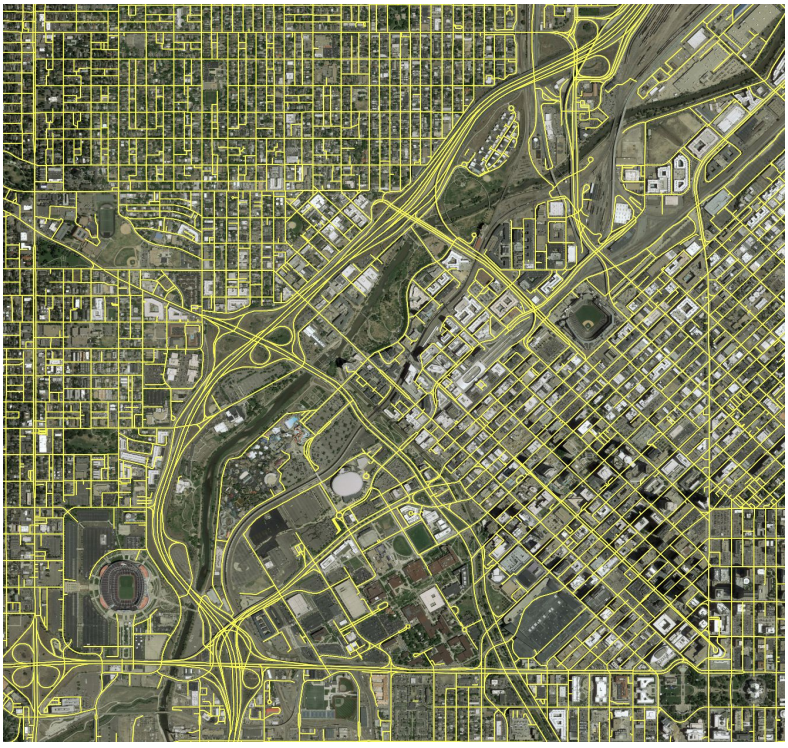
Appendix

Appendix contains results for the remaining original test cities - Amsterdam, Denver,Los Angeles, Montreal, Paris, Pittsburgh, Salt Lake City, Tokyo, Toronto.

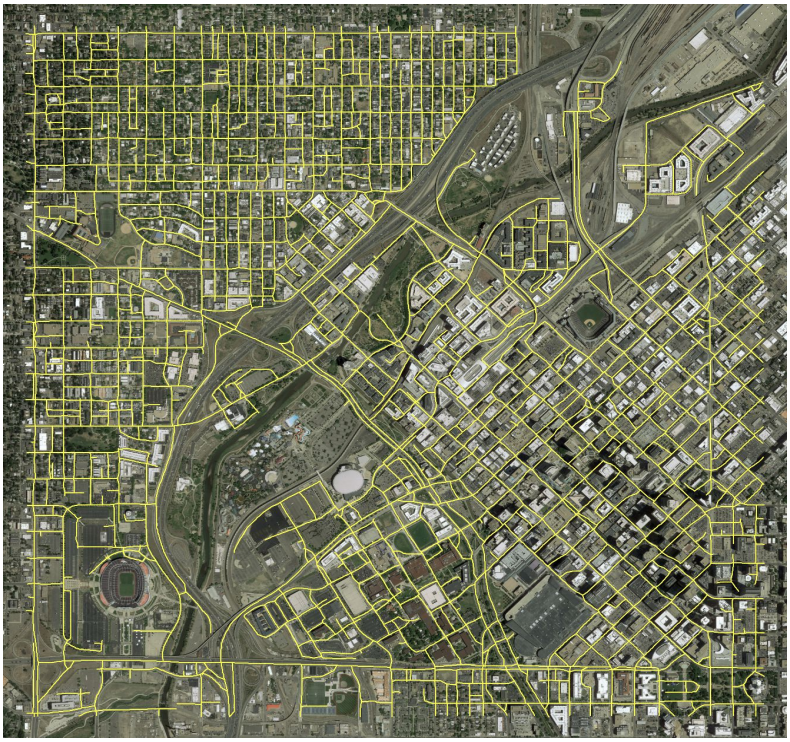
The image above shows true road network for Amsterdam



The image above shows predicted road network for Amsterdam



The image above shows true road network of Denver



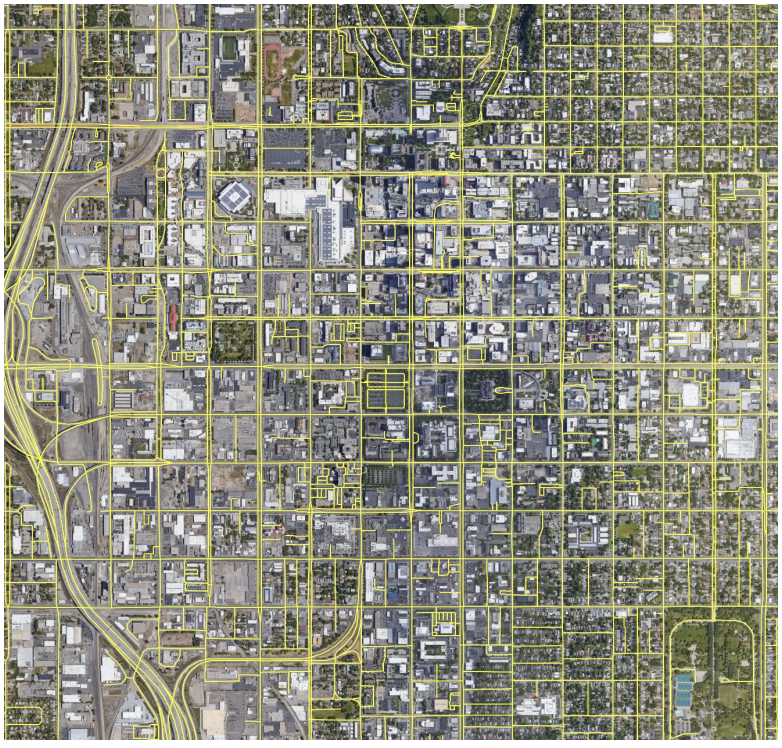
The image above shows predicted road network of Denver

The image above shows true road network of Los Angeles

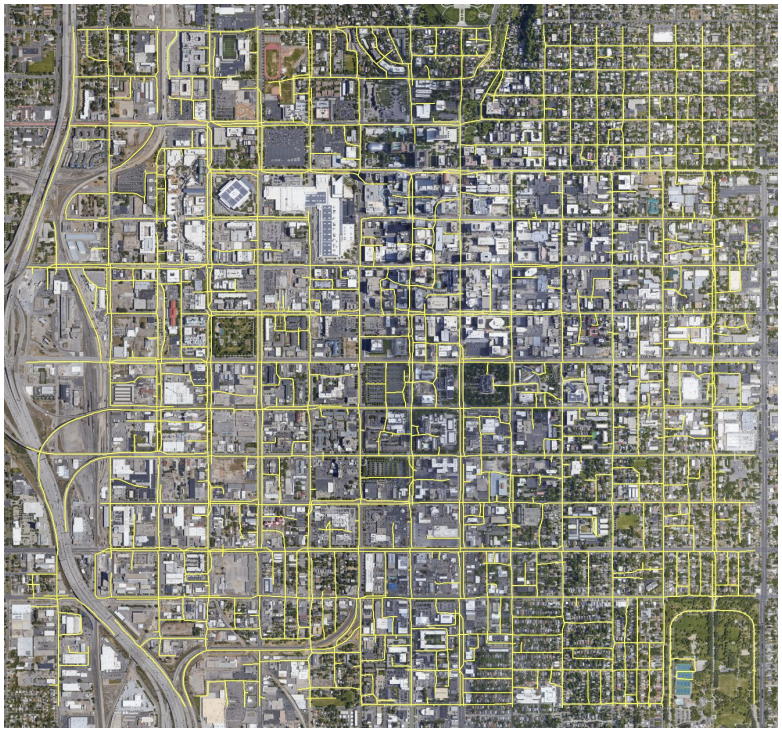
The image above shows predicted road network for Los Angeles

The above image shows true road network for Paris

The above image shows predicted road network for Paris

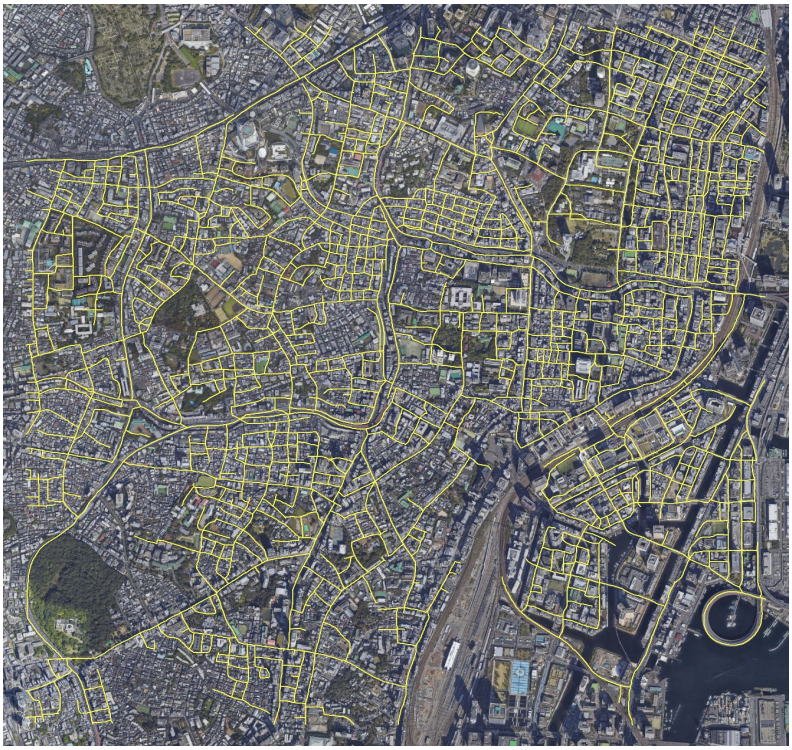


The above image shows the true road network for Salt Lake City

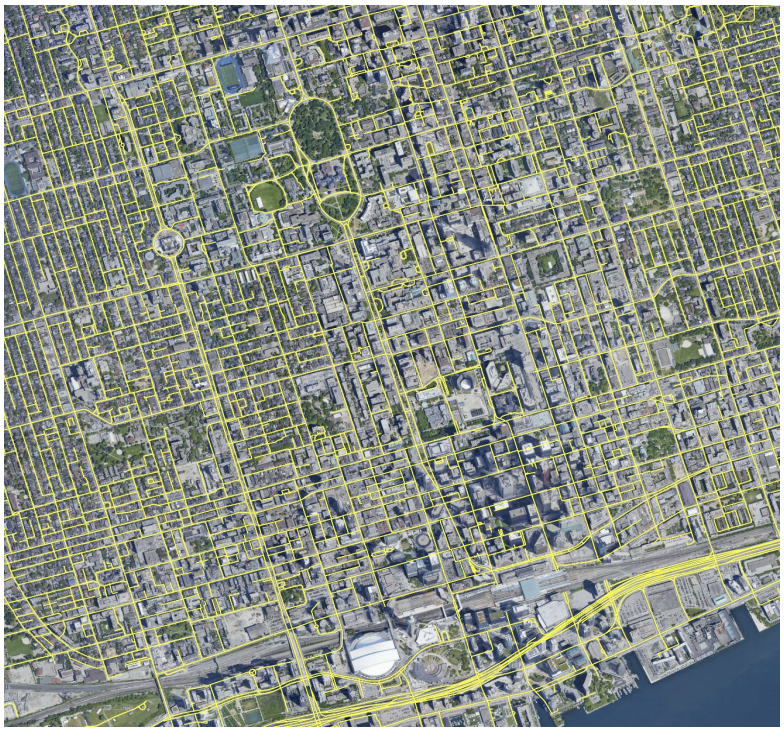


The above image shows the predicted road network for Salt Lake City

The above image shows the true road network for Tokyo



The above image shows the predicted road network for Tokyo



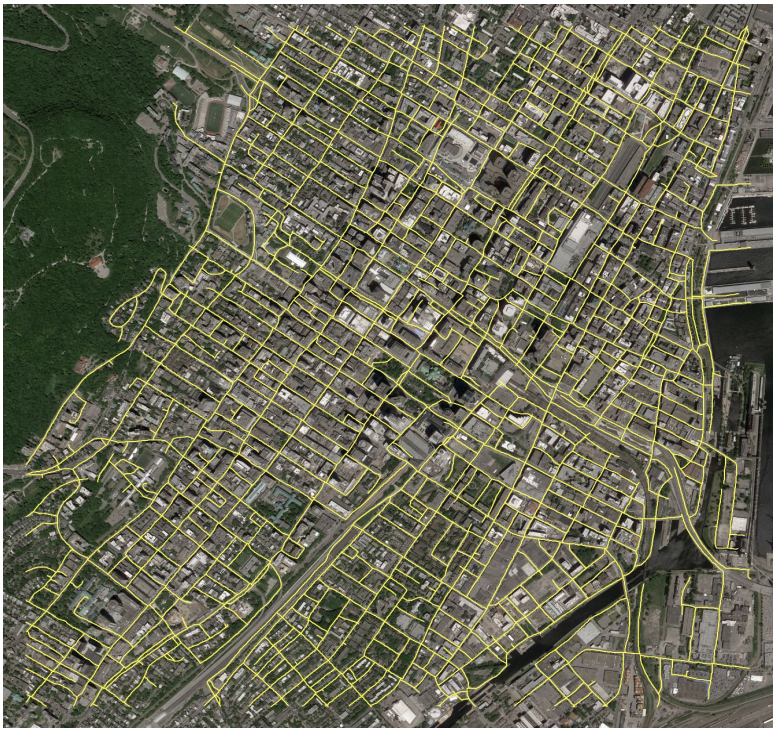
The above image shows the true road network for Toronto

The above image shows the predicted road network for Toronto

The above image shows the true road network for Pittsburgh

The above image shows the predicted road network for Pittsburgh

The above image shows the true road network for Montreal



The above image shows the predicted road network for Montreal