STREETS: A Novel Camera Network Dataset for Traffic Flow readme.pdf

Corey Snyder

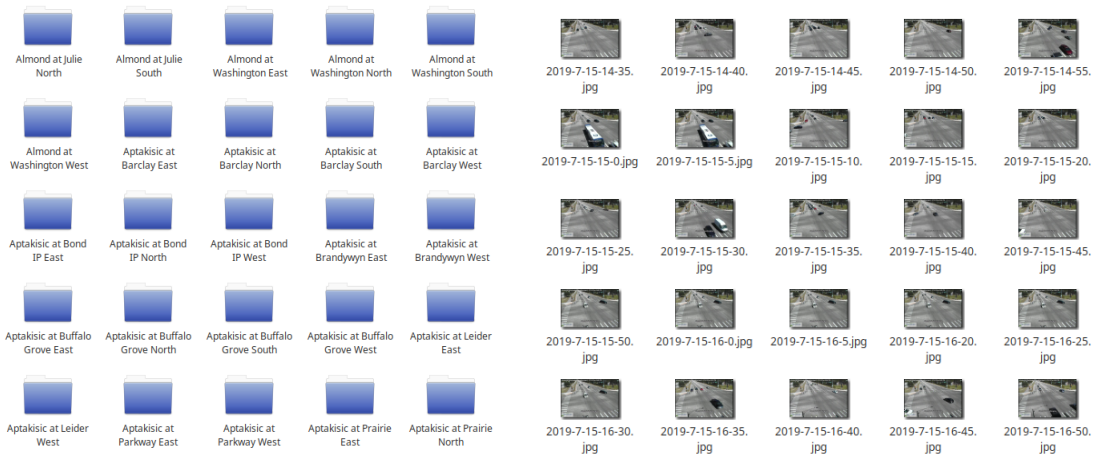
Last Updated: Oct. 27, 2019

Introduction

This file serves as a more elaborate description of the data provided in this dataset. For a briefer description, please refer to the readme.txt file. Many of these sections will show brief Python code snippets and console outputs to demonstrate how the data is loaded. Please refer to the accompanying GitHub repository for this dataset for ongoing updates to relevant codes and tools for loading or processing the dataset.

imagedata

Images are contained in separate folders for each week of images. The start and end of each zip file indicate the week of each folder. After unzipping, each day of images will be inside its own .zip. The images for each day are sorted in folder by camera view. The name of each view indicates the intersecting streets and directional leg the camera is facing, e.g. "Almond at Washington East". The name of each image timestamps when the image was pulled according to year-month-day-hour-minute.jpg. Figures 1.(a) and 1.(b) and 2 depict the structure within each day of images and within each view's folder, respectively.



(a) (b)

Figure 1: (a) Organization of folders in each day of image data and (b) a subset of images inside one of the folders in Fig. 1(a).

graphs

The graph JSONs for each community are contained in separate folders. In each folder, you will find the JSON for that community. Each JSON file is a dictionary to load the adjacency matrix, distance matrix,

and sensor dictionary to identify each camera. Let A be the adjacency matrix and D the distance matrix. The structure of these matrices is as follows, where $n_l(i,j)$ denotes the number of traffic lanes connecting sensors i and j and $d(i,j)$ the Google Maps travel distance from sensor i to sensor j:

$$A[i,j] = \begin{cases} n_l(i,j), & \exists \text{ directed edge from sensor } i \text{ to sensor } j \\ 0, & \nexists \text{ directed edge between the sensors} \end{cases}$$

$$D[i,j] = \begin{cases} d(i,j), & \exists \text{ directed edge from sensor } i \text{ to sensor } j \\ 0, & \nexists \text{ directed edge between the sensors} \end{cases}$$

Figure 2.(a) demonstrates how each key-value pair is loaded from the dictionary and Fig. 2.(b) shows the information stored in the sensor dictionary.

```
>>> import numpy as np
>>> import json
>>> from pprint import pprint
>>> with open('gurnee-graph.json', 'r') as f:
...     graph = json.load(f)
...
>>> adjacency = np.array(graph['adjacency-matrix'])
>>> distance = np.array(graph['distance-matrix'])
>>> sensor_dictionary = graph['sensor-dictionary']
>>>
```

(a)

```
>>> pprint(sensor_dictionary['28'])
[[42.363587, -87.929395], 'IL 21 at Washington East-inbound']
>>> pprint(sensor_dictionary['56'])
[[42.377277, -87.904064], 'US 41 at Delany East-inbound']
>>> pprint(sensor_dictionary['84'])
[[42.393079, -87.926813], 'US 41 at Stearns School North-inbound']
>>>
```

(b)

Figure 2: (a) How to load the graph for a particular community and (b) the stored data for each sensor index in the community. Note that the information for each sensor is a list that contains (1) the latitude and longitude coordinates of the sensor field-of-view and (2) the string describing the camera view and side of the roadway (inbound or outbound).

incidents

This folder contains the raw Excel spreadsheet with the reported incidents while "processed-incidents.json" extracts incidents that can be geographically located and are colocated with cameras in the STREETS dataset. This JSON contains the documented information for each traffic incident. Figure 3 gives examples of how incidents are stored in "processed-incidents.json". Note that empty cells in the Excel spreadsheet represent missing information for a reported incident.

```
>>> import json
>>> import numpy as np
>>> with open('processed-incidents.json', 'r') as f:
...     incidents = json.load(f)
...
>>> random_incident_keys = np.random.choice(list(incidents.keys()), 3)
>>> print(incidents[random_incident_keys[0]])
{'incidents': [{'coordinate': [42.173912, -87.9848863], 'event_type': 'Fire',
 'impact': 3, 'direction': 'North', 'time': '2018-09-20 20:43:39'}], 'affecte
d-sensors': [0, 1, 6, 7, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28,
29, 30, 31, 48, 49, 50, 51, 52, 53, 54, 55, 60, 61, 62, 63, 188, 189, 190, 19
1, 192, 193, 194, 195, 72, 73, 74, 75, 76, 77, 78, 79, 199, 196, 197, 98, 99,
 100, 101, 102, 103, 104, 105, 198, 114, 115, 116, 117]}
>>> print(incidents[random_incident_keys[1]])
{'incidents': [{'coordinate': [42.2405887, -87.9769936], 'event_type': 'Stall
', 'impact': 2, 'direction': 'North', 'time': '2018-08-29 07:45:39'}], 'affec
ted-sensors': [320, 321, 298, 299, 268, 269, 270, 271, 272, 273, 300, 301, 31
4, 315, 316, 317, 318, 319]}
>>> print(incidents[random_incident_keys[2]])
{'incidents': [{'coordinate': [42.197294, -87.98939229999999], 'event_type':
'Stall', 'impact': 2, 'direction': 'North', 'time': '2018-08-29 17:41:04'}],
'affected-sensors': [144, 145, 146, 147, 148, 149, 150, 151, 188, 189, 190, 1
91, 192, 193, 208, 209, 210, 211, 212, 213, 214, 215, 224, 225, 226, 227, 228
, 229, 230, 231]}
>>>
```

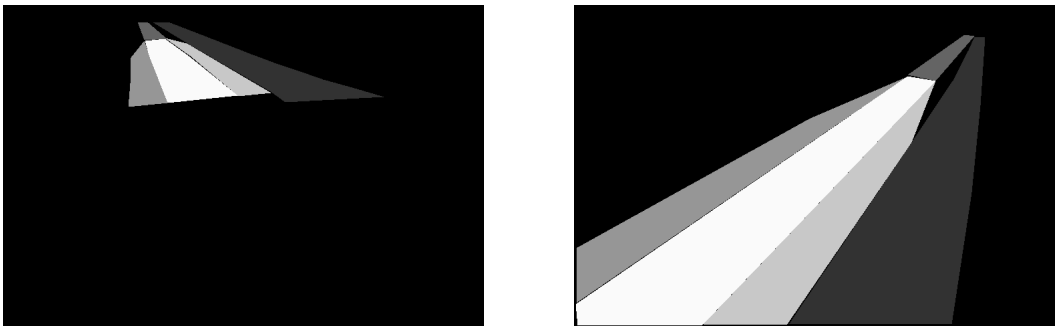
Figure 3: Python code for loading incident data and printing a few example incidents. Each incident maps to (1) a list with at least one documented occurence of coordinates, event type, traffic impact (1 to 4), direction, and timestamp and (2) a list of affected sensor indices in the area of the incident.

roadmasks

This folder contains the hand-annotated roadmasks for each camera view. The roadmasks are separated by year into two folders. Each year contains a .png image for the roadmask of each camera view. For roadmask image R, the pixel values in each roadmask give the following traffic lane information:

$$R[i,j] = \begin{cases} 0, & \text{non-roadway} \\ 50, & \text{outbound} \\ 100, & \text{non-lane specific inbound} \\ 150, & \text{inbound right turn-lane} \\ 200, & \text{inbound left turn-lane} \\ 250, & \text{inbound thru-lane} \end{cases}$$

Figure 4 shows a couple example roadmasks.



(a) Almond at Washington East (b) Delany at Continental South

Figure 4: Example roadmasks for two camera views.

trafficcounts

This folder contains the processed traffic data for each day of images. The data is separated into folders for 2018 and 2019, respectively. There is a separate JSON for each day of traffic data. Figures 5.(a) and 5.(b) show a code snippet for loading traffic data for one camera view and printing the resulting traffic information. Note how for each image we indicate the number of inbound vehicles, number of outbound vehicles, and the time of day extracted from the top of the image using Tesseract OCR.

```
>> import json
>> from pprint import pprint
>> with open('2019-7-1-trafficcounts.json', 'r') as f:
..     data = json.load(f)
..
>> view_data = data['Hunt Club at Washington North']
>>
```

(a)

```
>>> pprint(view_data)
{'2019-7-1-10-0.jpg': {'inbound': 7, 'outbound': 8, 'timestamp': [9, 55]},
 '2019-7-1-10-10.jpg': {'inbound': 2, 'outbound': 4, 'timestamp': [10, 7]},
 '2019-7-1-10-15.jpg': {'inbound': 16, 'outbound': 4, 'timestamp': [10, 11]},
 '2019-7-1-10-20.jpg': {'inbound': 4, 'outbound': 10, 'timestamp': [10, 17]},
 '2019-7-1-10-25.jpg': {'inbound': 4, 'outbound': 4, 'timestamp': [10, 21]},
 '2019-7-1-10-30.jpg': {'inbound': 10, 'outbound': 1, 'timestamp': [10, 25]},
 '2019-7-1-10-35.jpg': {'inbound': 6, 'outbound': 5, 'timestamp': [10, 33]},
 '2019-7-1-10-40.jpg': {'inbound': 16, 'outbound': 2, 'timestamp': [10, 37]},
 '2019-7-1-10-45.jpg': {'inbound': 4, 'outbound': 3, 'timestamp': [10, 41]},
 '2019-7-1-10-5.jpg': {'inbound': 5, 'outbound': 4, 'timestamp': [10, 3]},
 '2019-7-1-10-50.jpg': {'inbound': 8, 'outbound': 2, 'timestamp': [10, 45]},
 '2019-7-1-10-55.jpg': {'inbound': 7, 'outbound': 0, 'timestamp': [10, 54]},
 '2019-7-1-11-0.jpg': {'inbound': 5, 'outbound': 8, 'timestamp': [10, 57]},
 '2019-7-1-11-10.jpg': {'inbound': 7, 'outbound': 7, 'timestamp': [11, 6]},
 '2019-7-1-11-15.jpg': {'inbound': 6, 'outbound': 3, 'timestamp': [11, 14]},
 '2019-7-1-11-20.jpg': {'inbound': 10, 'outbound': 3, 'timestamp': [11, 18]},
 '2019-7-1-11-25.jpg': {'inbound': 3, 'outbound': 4, 'timestamp': [11, 22]},
 '2019-7-1-11-30.jpg': {'inbound': 10, 'outbound': 2, 'timestamp': [11, 28]},
```

(b)

Figure 5: (a) Example Python code for loading traffic counts from one day of image data at a particular camera view. (b) Resulting image data at the camera view on each side of the roadway. The timestamp is a list of (hour, minute). Note this does not show all samples for the given day, only the first 18 images printed form the dictionary.

vehicleannotations

This folder holds the vehicle annotations and images used to retrain the Mask R-CNN. The "annotations" folder contains the JSON file with the annotated polygons for each vehicle and the "images" folder contains the corresponding images. We recommend using the VGG Annotator tool as it will seemlessly load our images and annotations for visualization. Figure 6 shows how the annotations appear in the VGG Annotator.

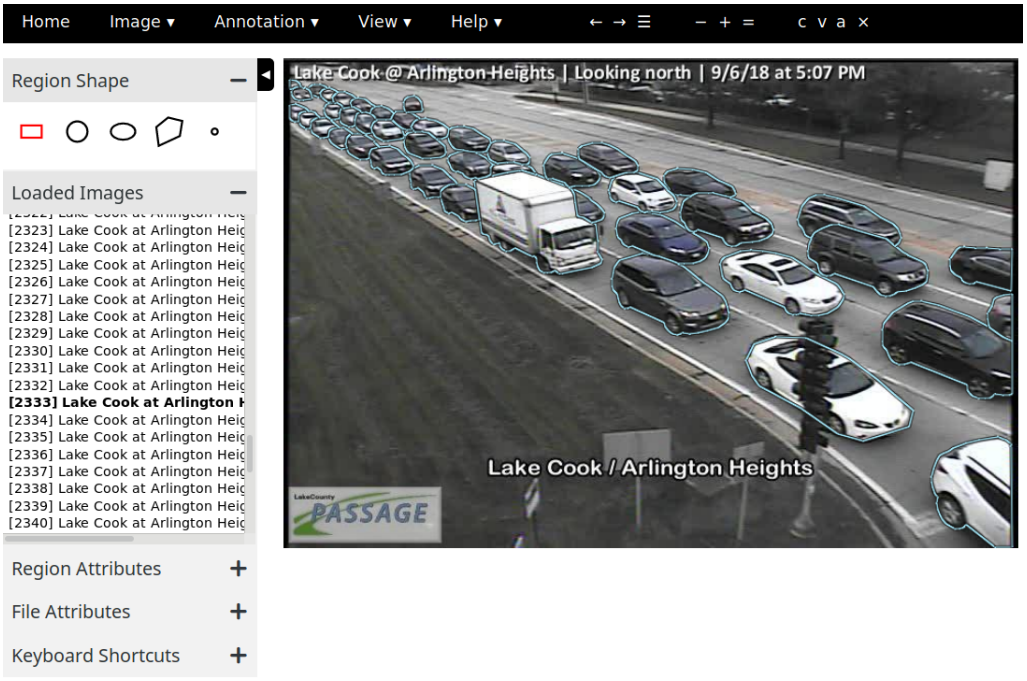


Figure 6: Example of annotations loaded in the VGG Annotator tool.

viewclassifiers

The view classifiers are separated by year. In the folder for each year, you will find a .joblib file for the K-Nearest Neighbor classifier at each camera location. There is also a "clf_sizes_year.json" file that tracks the feature vector size after preprocessing before passing to each respective classifier. These view classifiers are only of interest if you intend to work with the original image data and would like to ensure each loaded image is sorted into the correct directional view. The linked GitHub for this dataset contains more details on how to use these classifiers.