

Traffic Analysis in Original Video Data of Ayalon Road

Ido Greenberg

2019



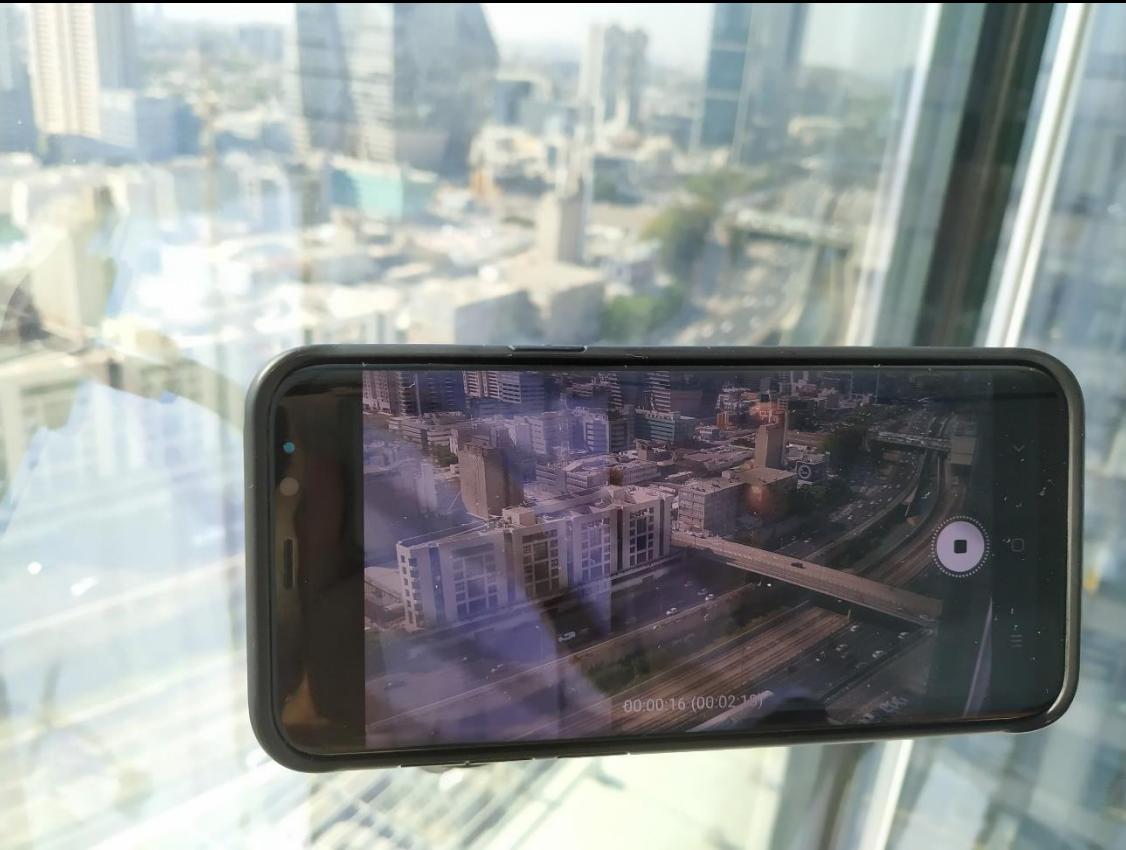
Background



Looking down: data are everywhere

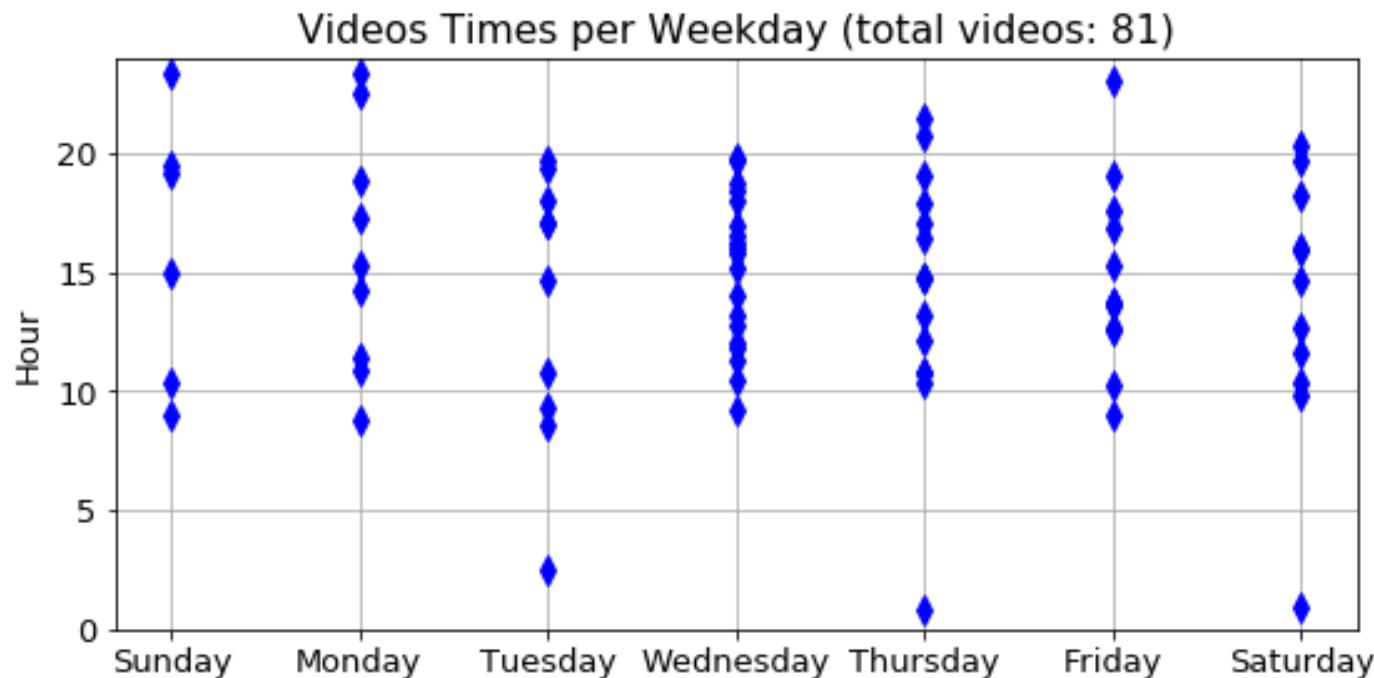
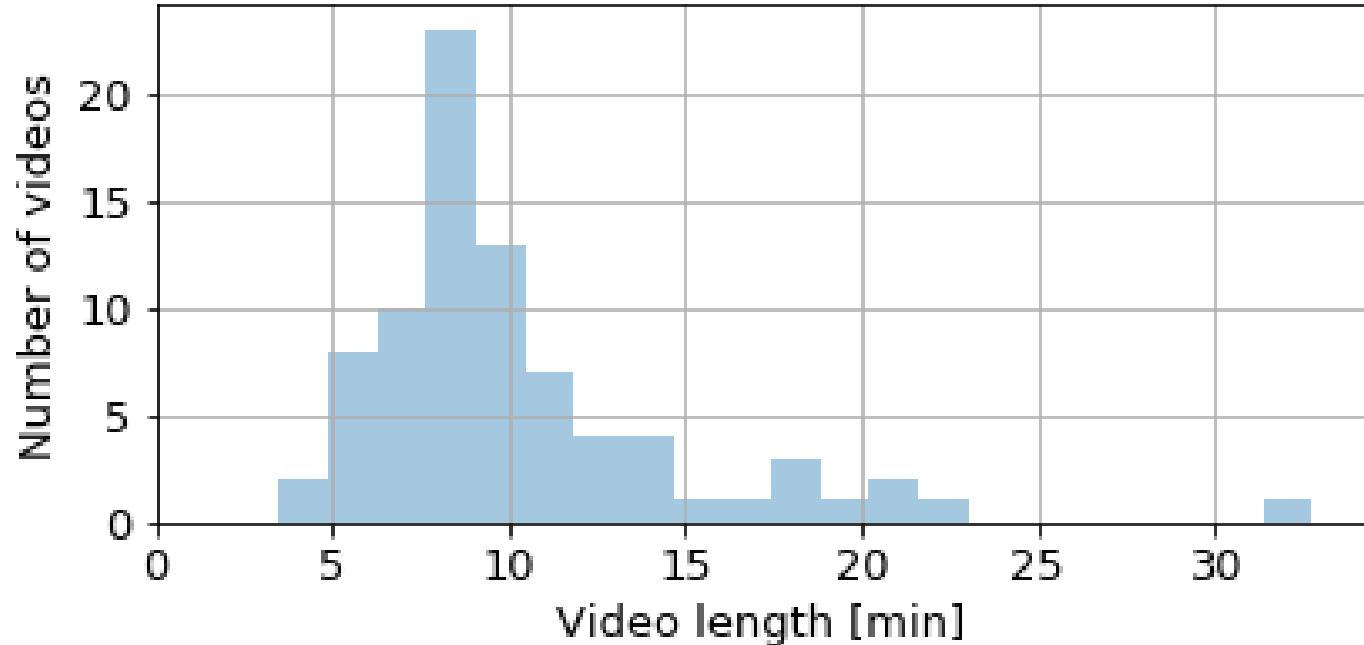


- Hyperlapse, x8 speed, Full-HD
- 120MB for 8 minutes



Collected data

- 81 videos X 8 minutes over 2 months



What we need

- **Detection:** find vehicles in a frame
- **Tracking:** connect the same vehicle in different frames
- **Analysis of traffic**



Detection

Out-of-the-box YOLO





Out-of-the-box SSD (better with small, crowded objects)

car: 55.08%

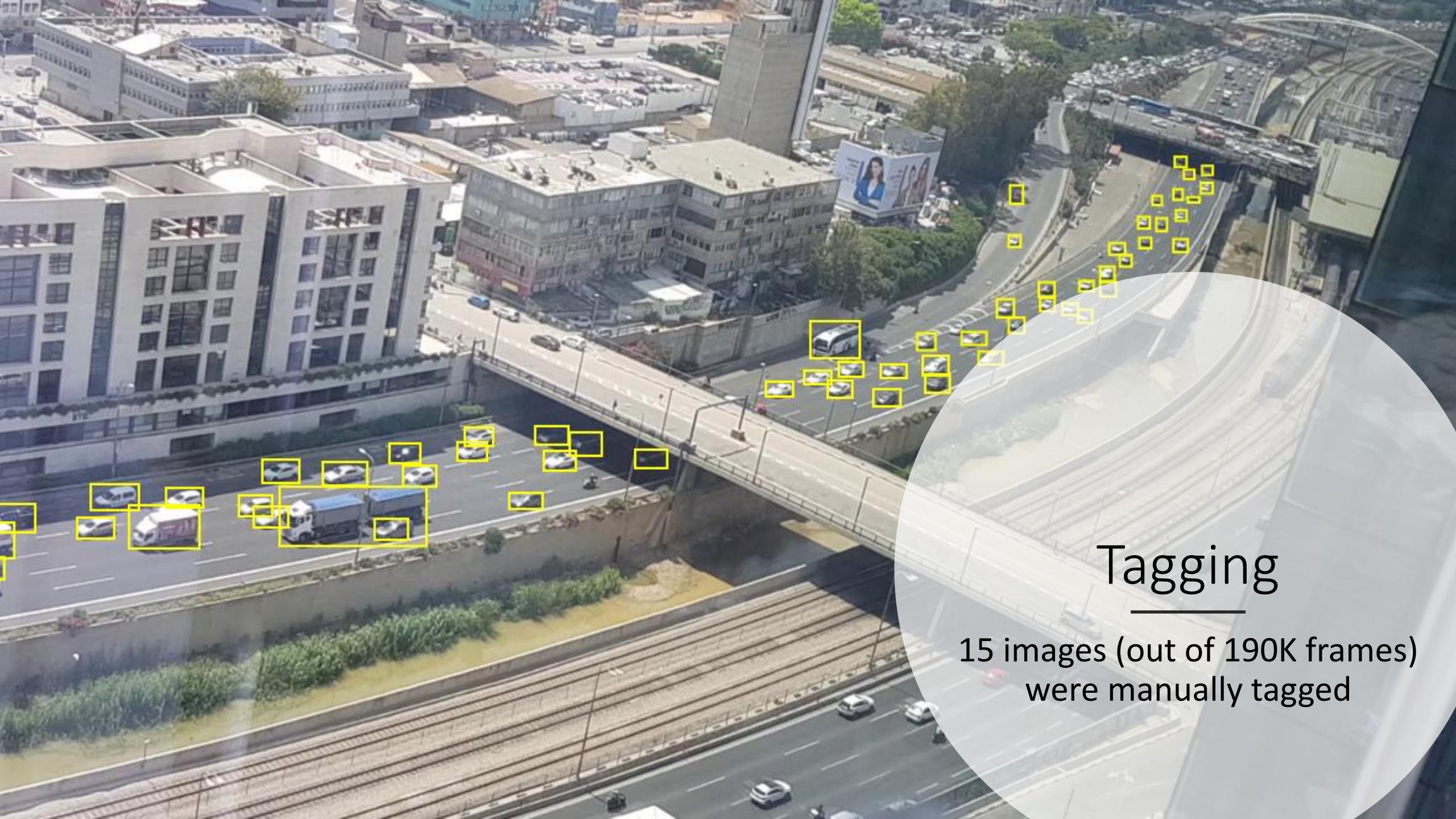
Zoom in?

No one's gonna train for me

- **Train:** tune the detection according to the data
- **Design:** adjust models to fit typical sizes & density
 - Also simplify – no need for multiple object classes



Having to train by myself - illustration

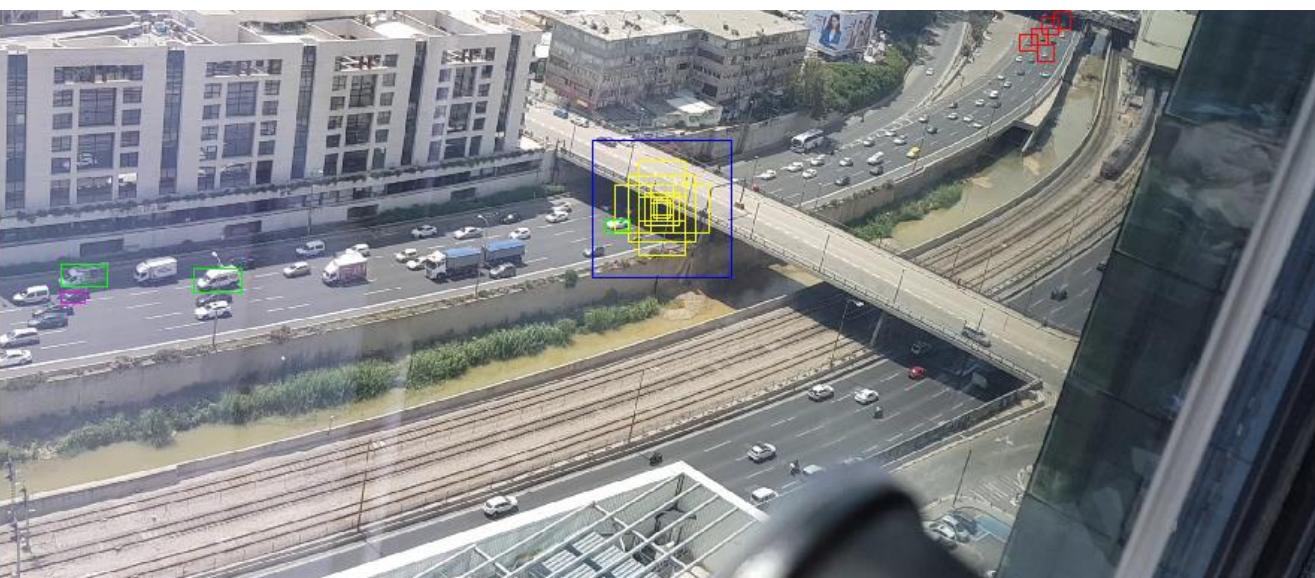


Tagging

15 images (out of 190K frames)
were manually tagged

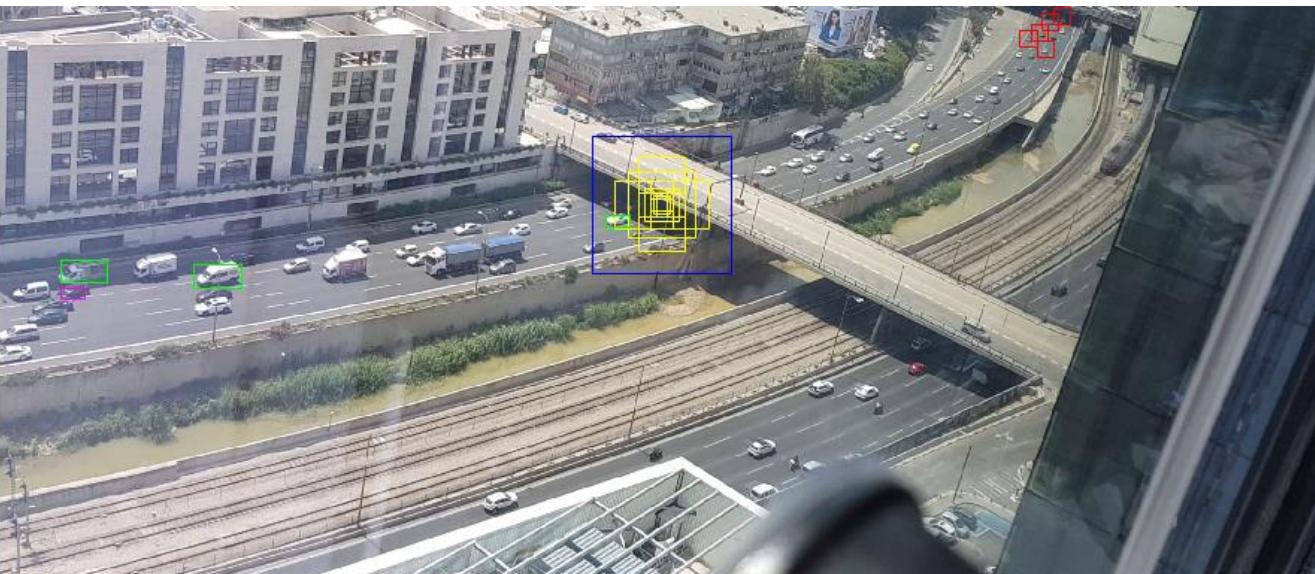
Tagged vehicles → network-compatible output

- Split image to 290K (overlapping) rectangles (***anchor boxes***)
 - 9 boxes per location – various scales & shapes
 - A box receives info from pixels around it (***receptive field***)

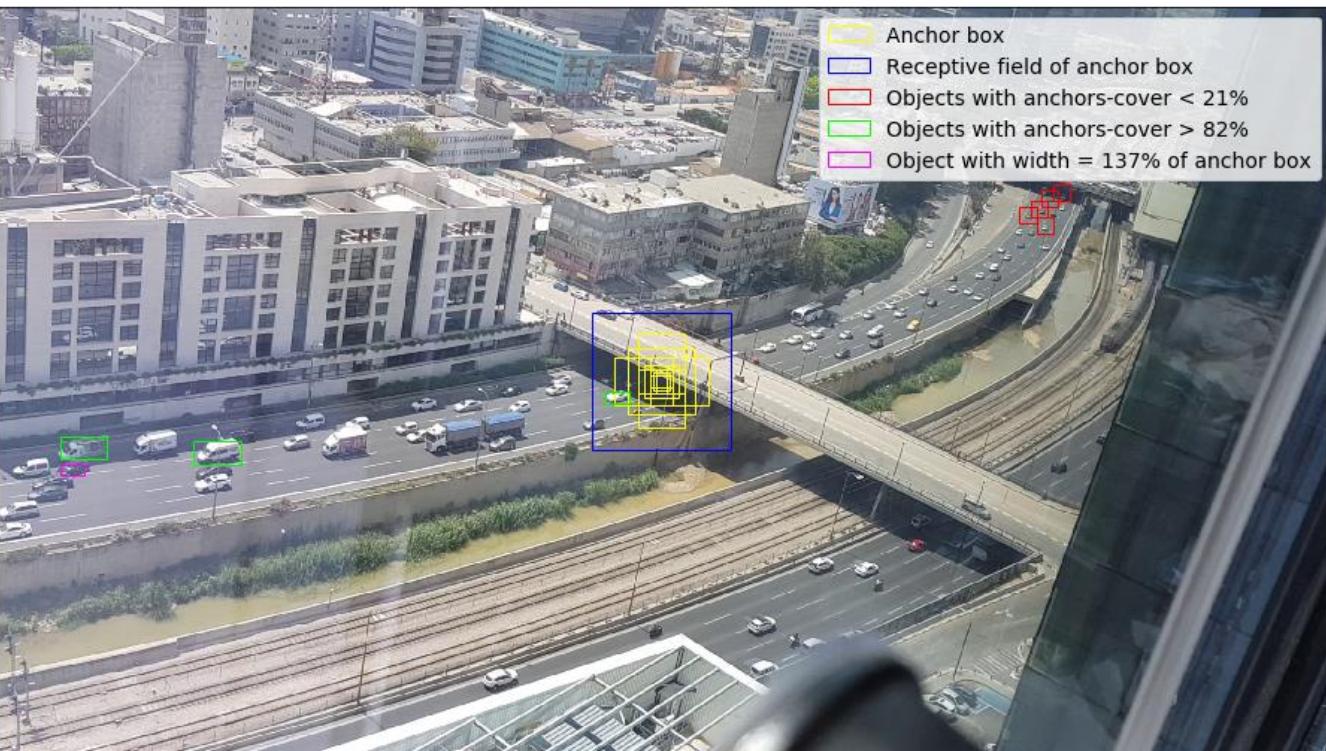


Tagged vehicles → network-compatible output

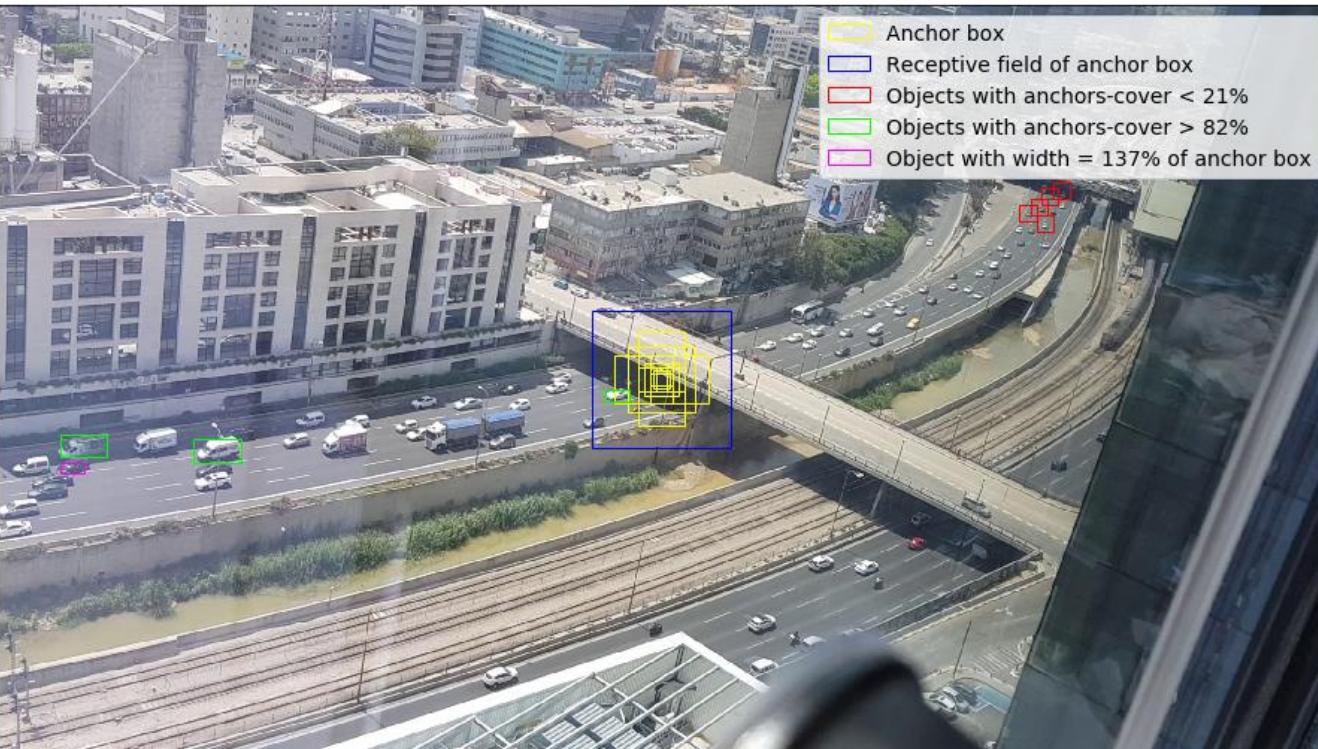
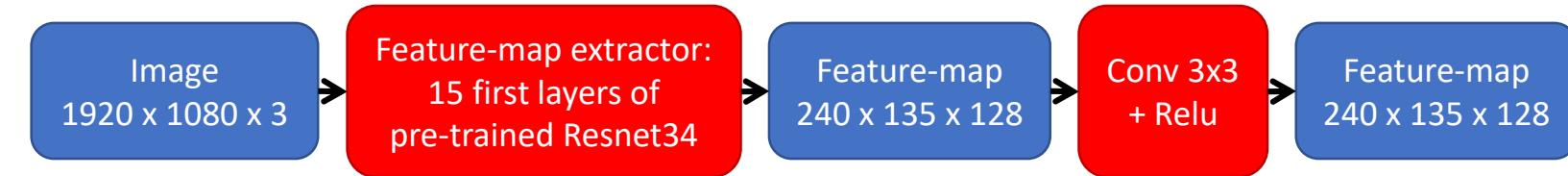
- Split image to 290K (overlapping) rectangles (***anchor boxes***)
 - 9 boxes per location – various scales & shapes
 - A box receives info from pixels around it (***receptive field***)
- Output per box:
 - Highly-overlapping object → (1, object location)
 - Little/no overlap → (0, ...arbitrary...)
 - Somewhere in the middle → drop from training data



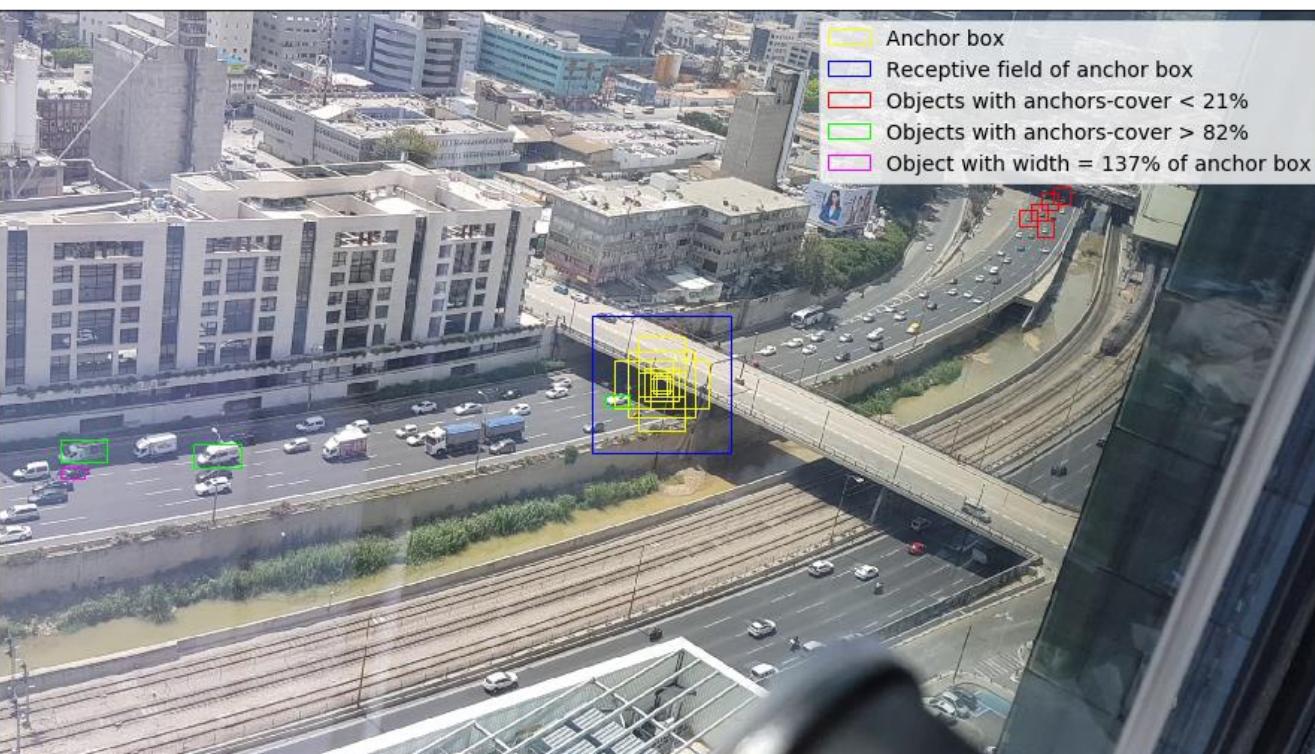
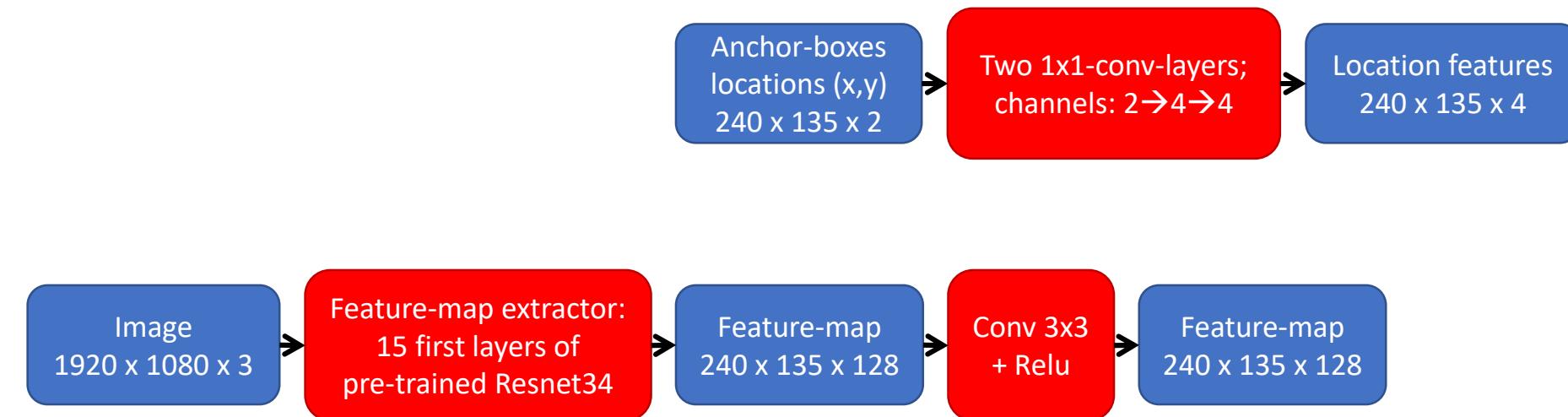
Network architecture



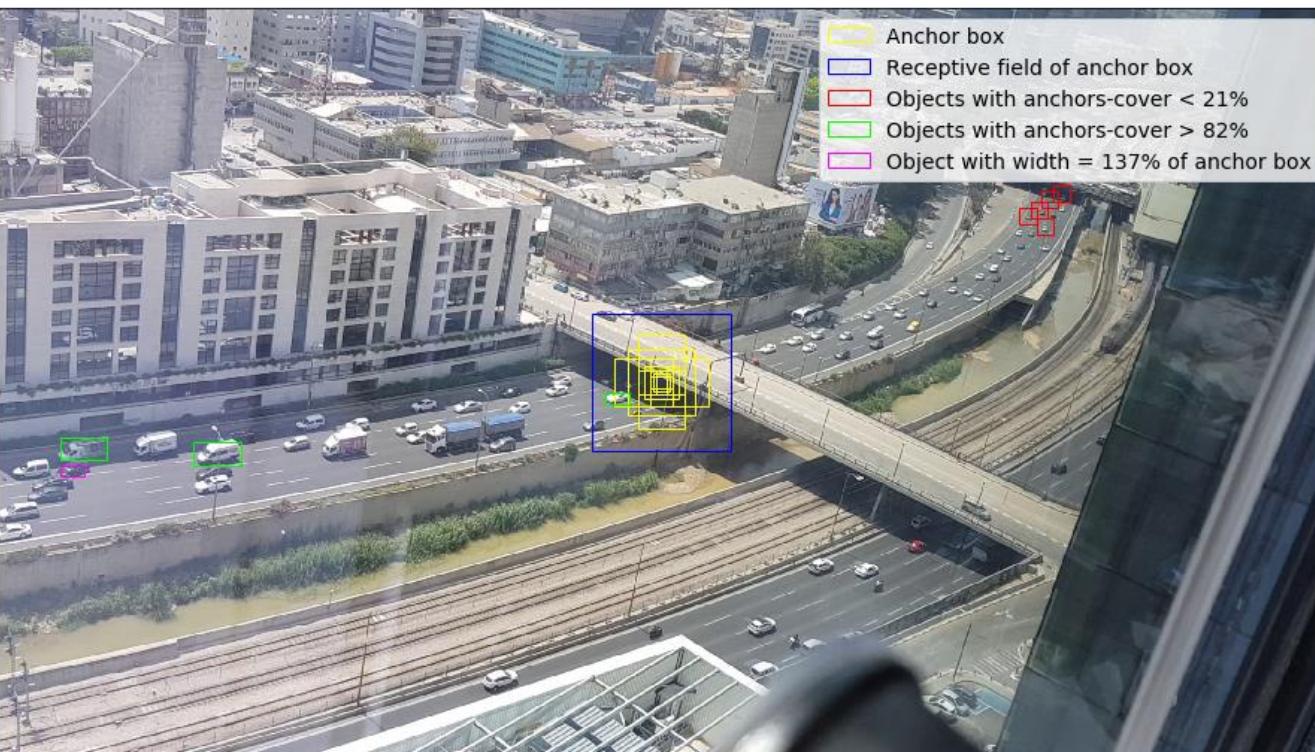
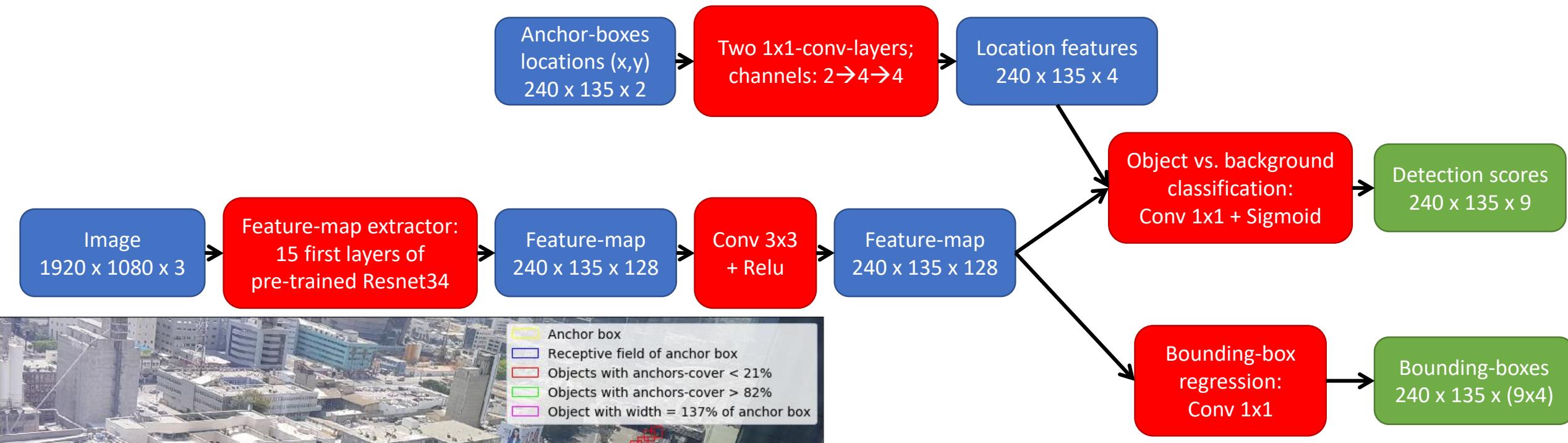
Network architecture



Network architecture

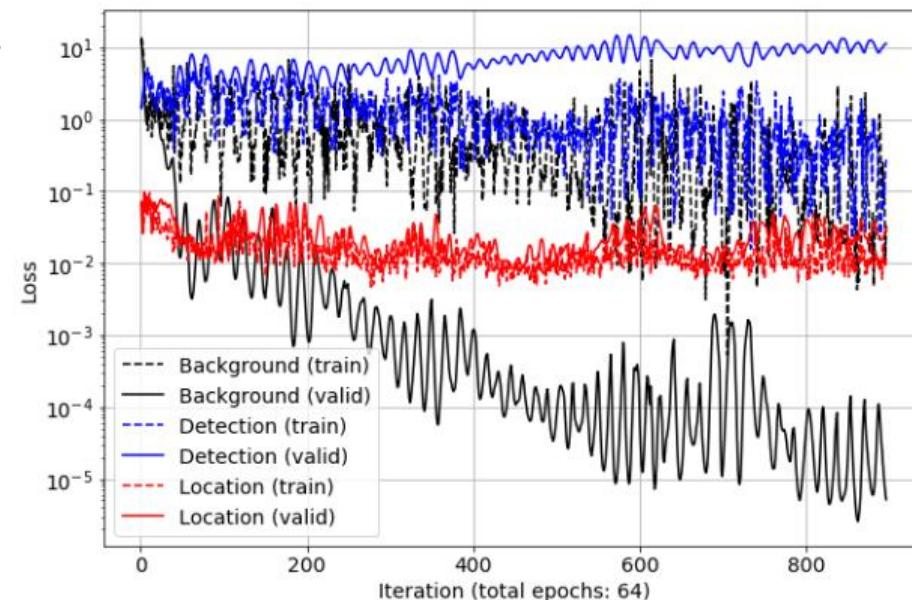


Network architecture



Training

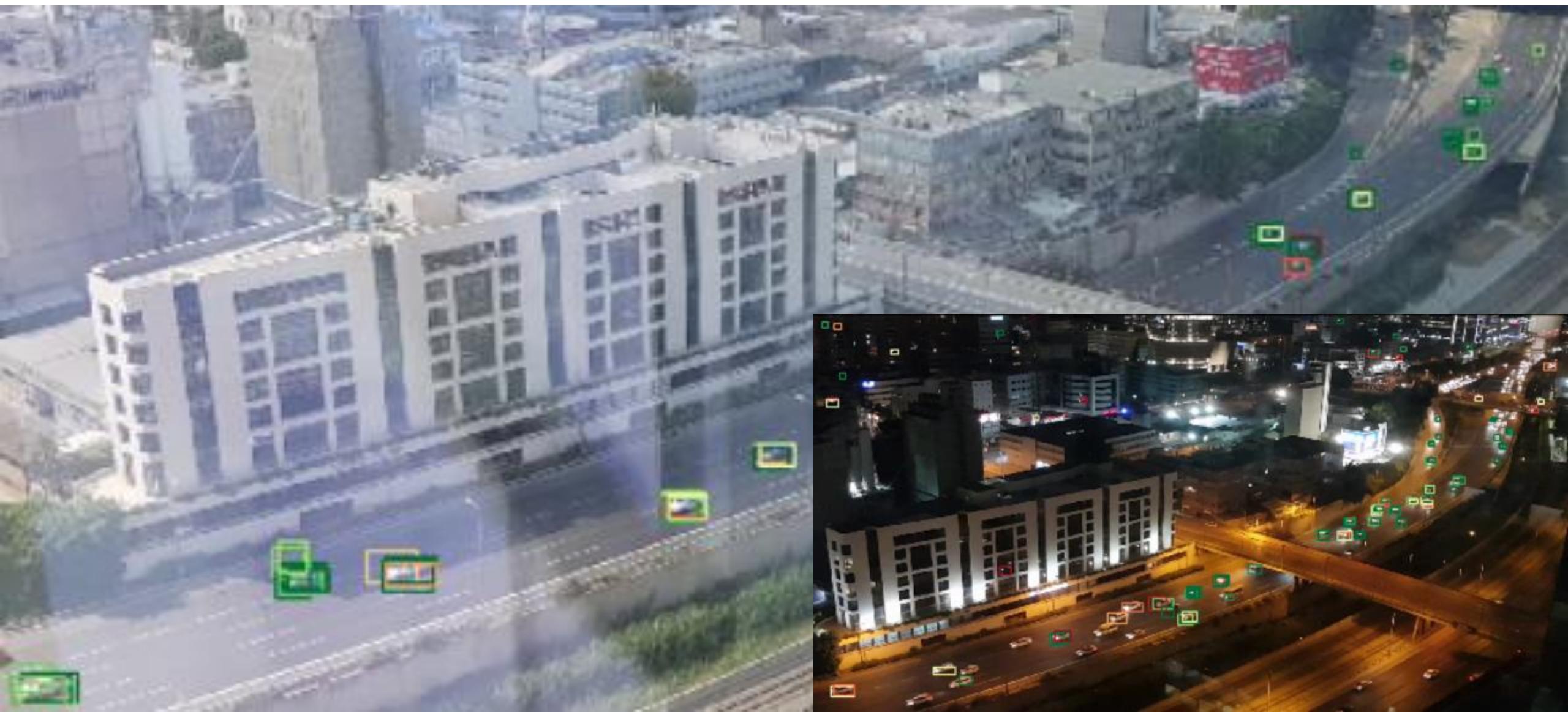
- **Optimizer:** Adam
- **Loss:** Binary cross-entropy (detection) + L1 (location)
 - Manual tuning of the FP/FN tradeoff
- **Transfer learning:** begin with freezed pre-trained Resnet layers
- **Hard negative mining:** up-sample boxes with large loss
- Architecture & hyper-params empirical tuning
- Running time on my laptop:
 - Freezed Resnet-layers: 5 minutes
 - Unfreezed Resnet-layers: +7 minutes



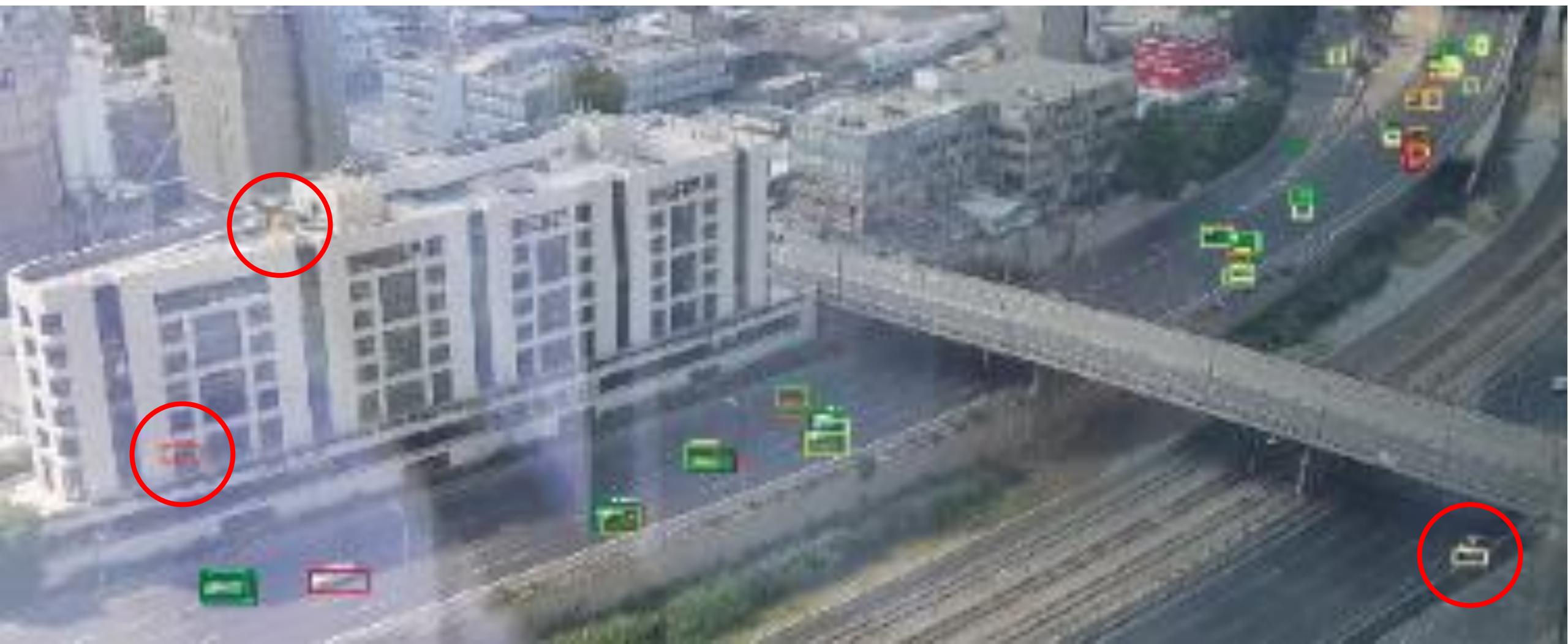
Without Hard Negative Mining



3 training images



Middle of training (15 training images)



Middle of training

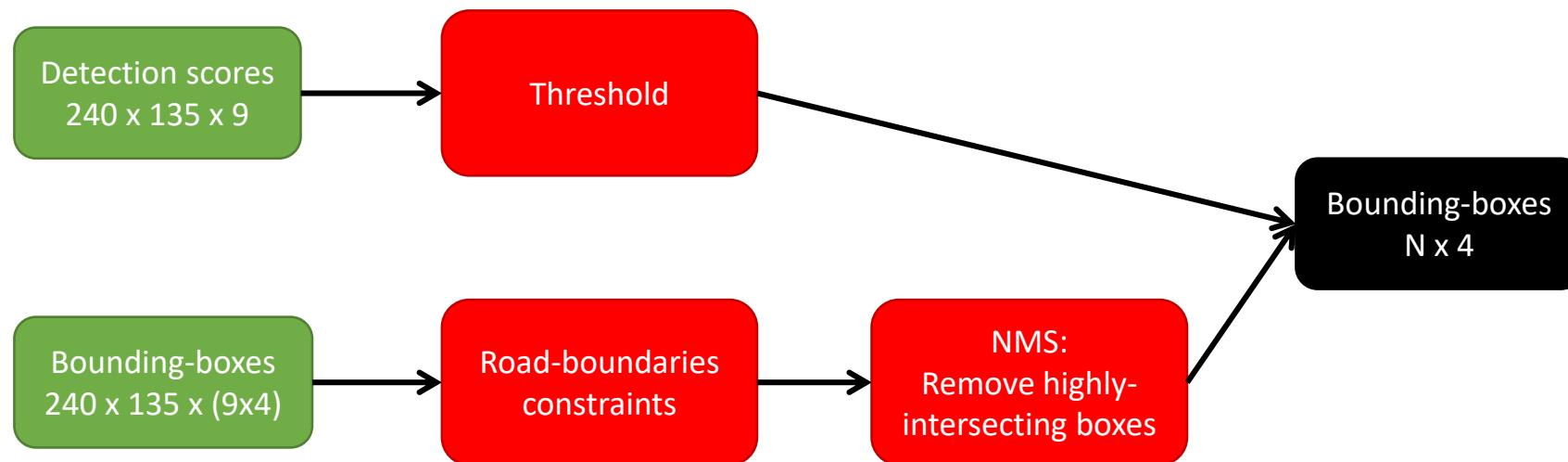


End of training

(15 training images)



Network output → predicted boxes



122 objects (0.315 = red <= certainty <= green)

Road
boundaries

*All results are out-of-sample

42 objects (0.360 = red <= certainty <= green)

Night +
reflections

*All results are out-of-sample



22 objects (0.379 = red <= certainty <= green)

Shavuot

*All results are out-of-sample



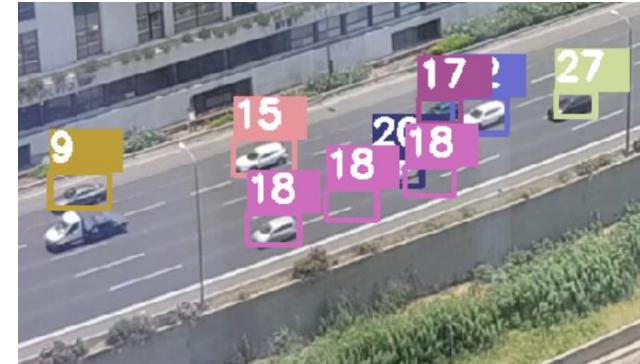
Tracking

SORT: Simple, Online and Realtime Tracking

- New detections \leftrightarrow "active tracks": assigned by intersection (**IOU**)
 - (Hungarian algorithm for linear assignment)



What the engineer had in mind



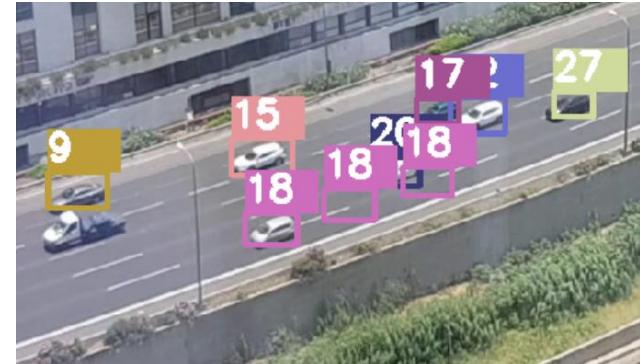
What the customer had in mind

SORT: Simple, Online and Realtime Tracking

- New detections \leftrightarrow "active tracks": assigned by intersection (**IOU**)
 - (Hungarian algorithm for linear assignment)



What the engineer had in mind

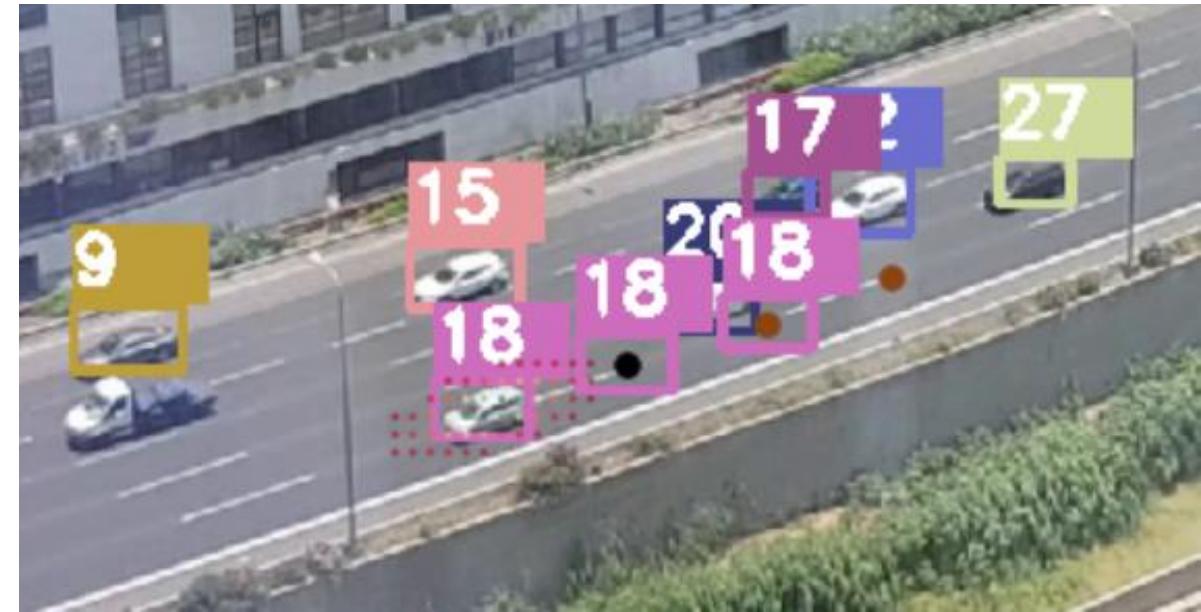


What the customer had in mind

- Active track – represented by ***Kalman Filter***:
 - Iteratively merge info from motion extrapolation & new observations
 - At any time, holds the track location and its uncertainty
- What can we possibly do?

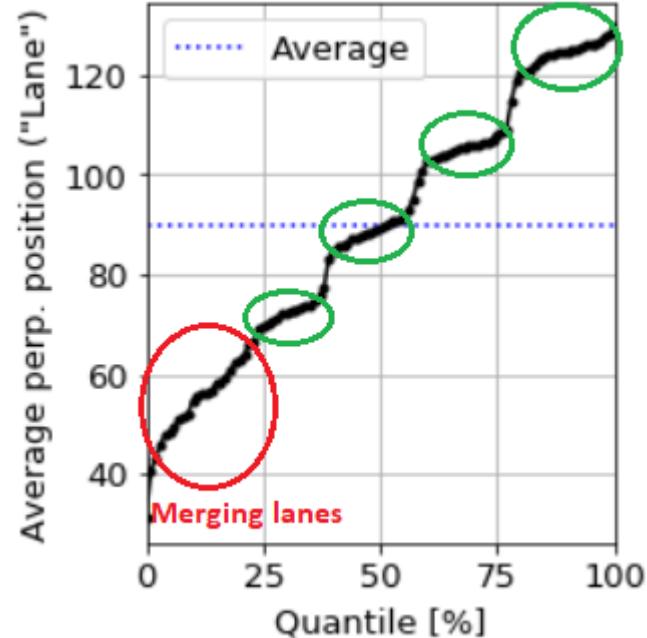
KF-based objects assignment

- Only location-based (no size & shape)
- Predicted-location error: 10 pixels in motion direction (u_1), 2 in perpendicular (u_2)
 - Motion direction: globally-estimated once in a while
 - **Covariance** = $U^T \cdot \begin{pmatrix} 10^2 & 0 \\ 0 & 2^2 \end{pmatrix} \cdot U$ $(U := (u_1, u_2))$
 - \Rightarrow Likelihood (any location | predicted location, covariance)
- [New detections \leftrightarrow tracks] assigned by likelihood



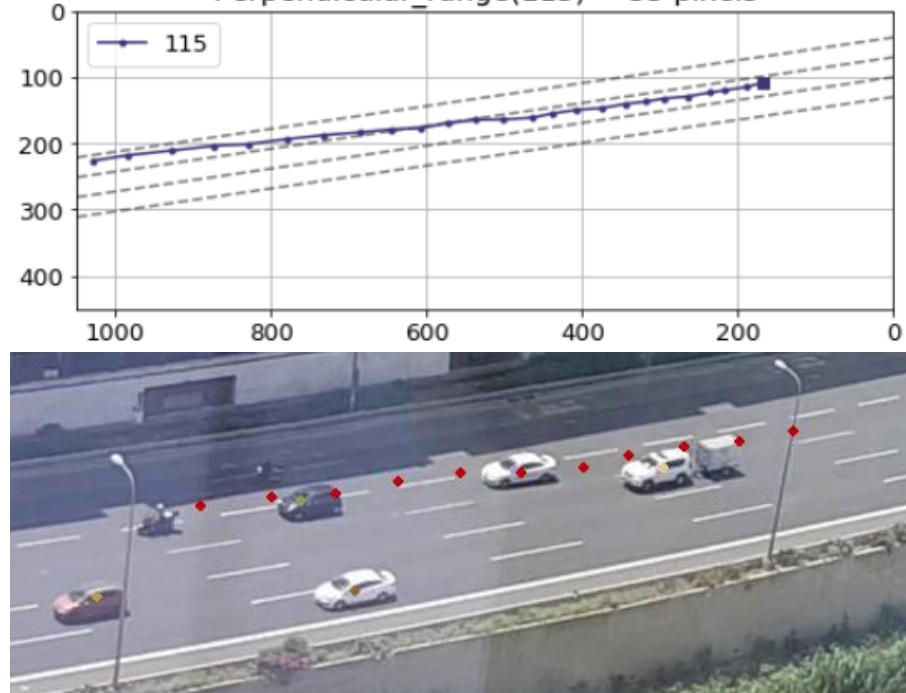
Does it work?

Vehicles distribution among lanes



Detected lane transition

Perpendicular_range(115) = 35 pixels

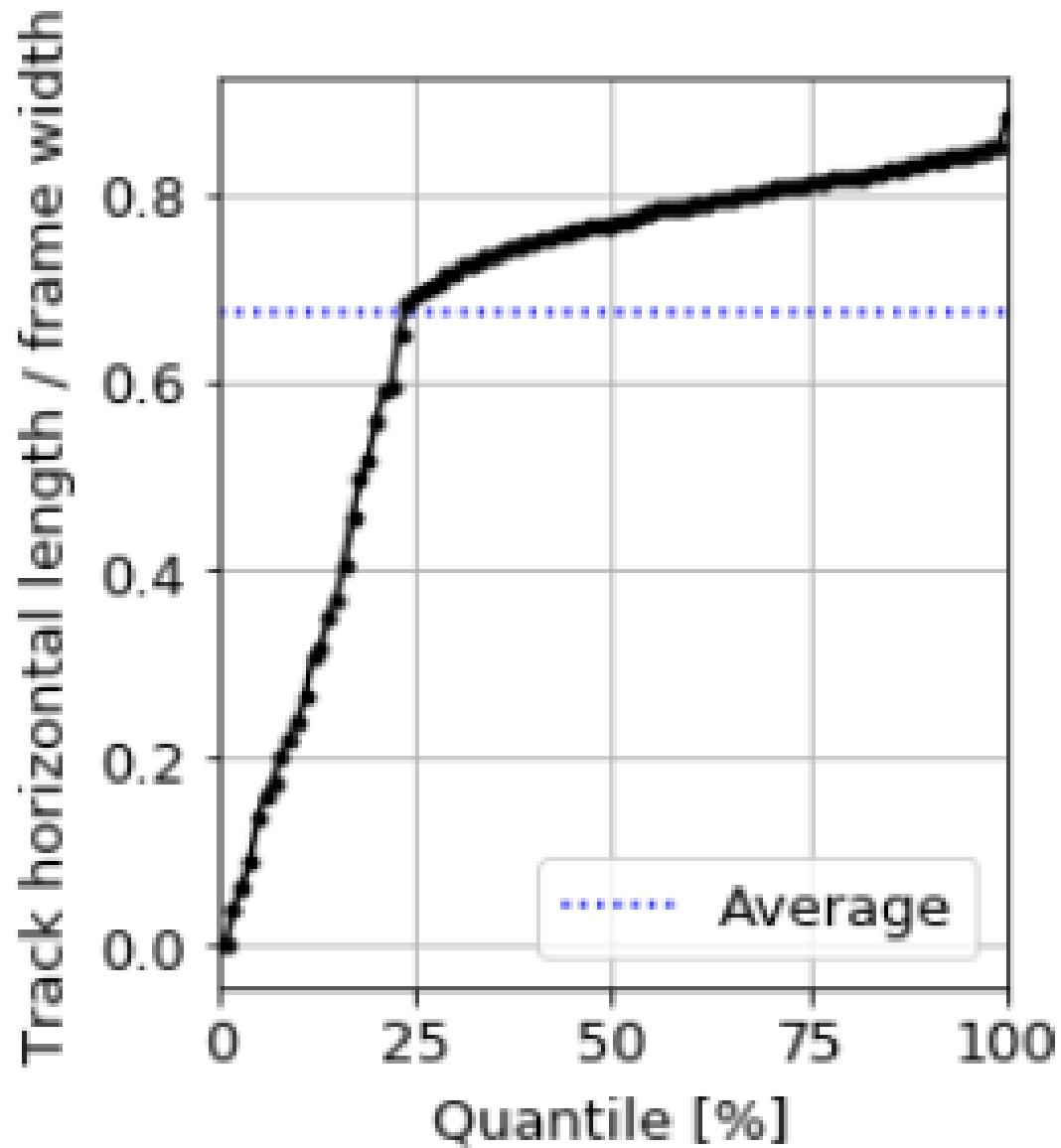


Track over gaps w/o detection



Does it work?

- Do we track the whole path?

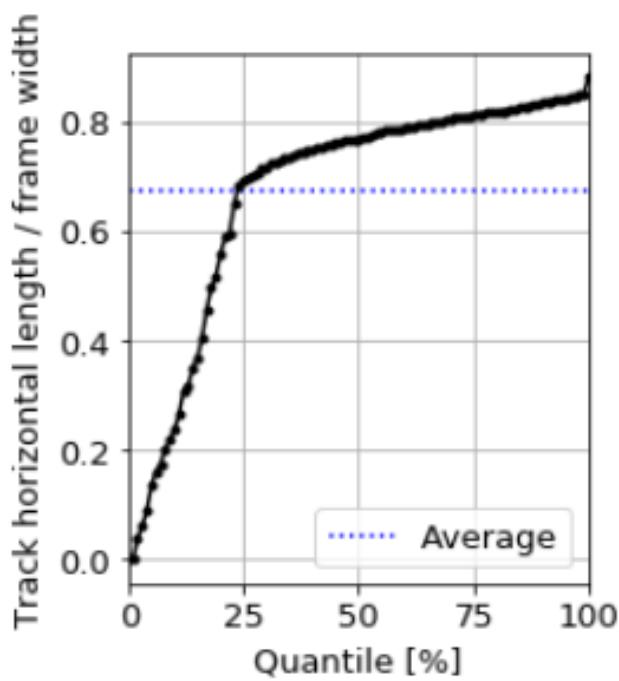


Does it work?

- Do we track the whole path?

- Fake tracks: false detections

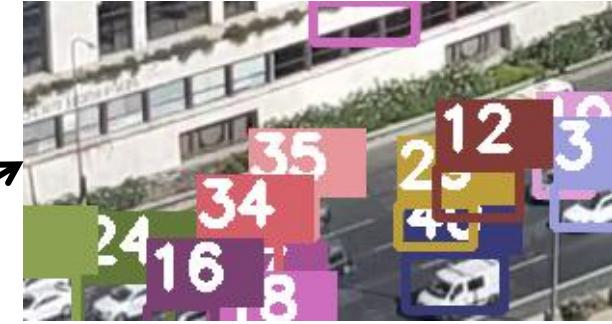
- Failed tracks: missing detections, hidings



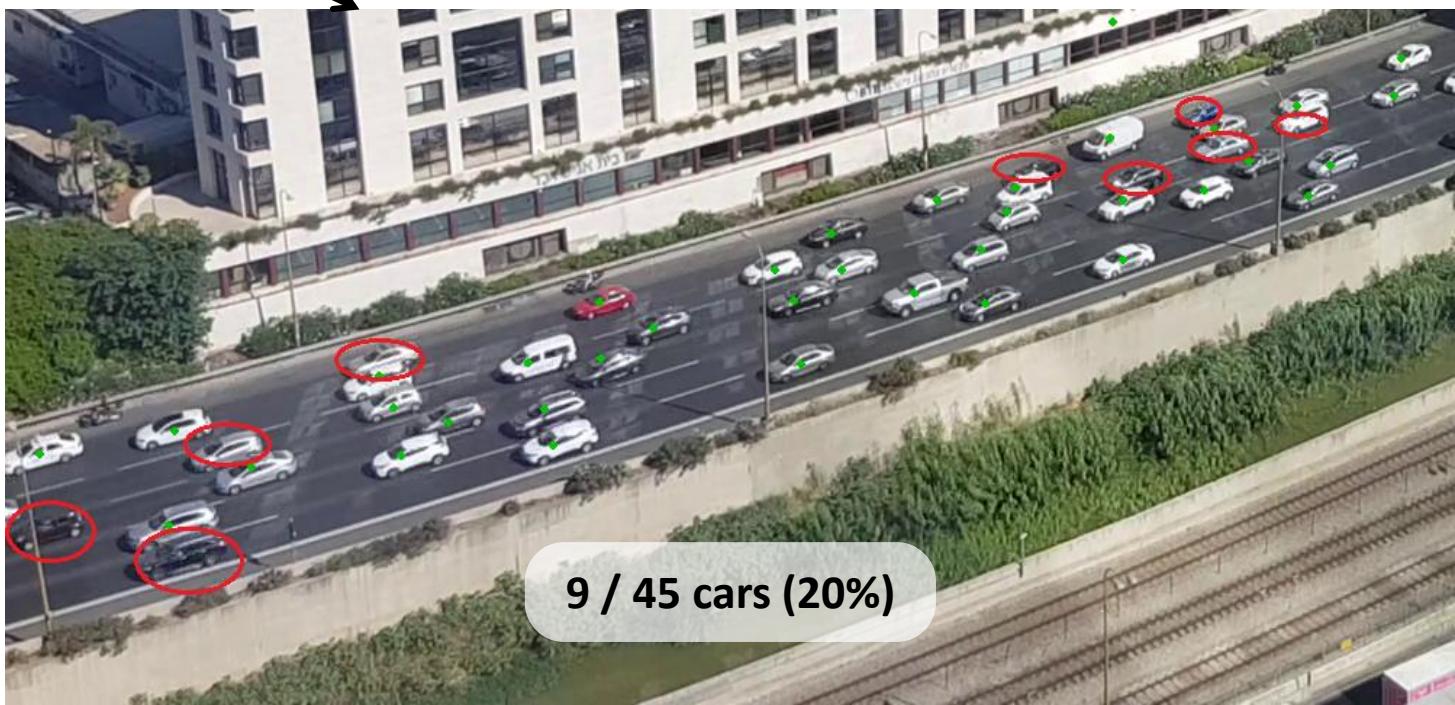
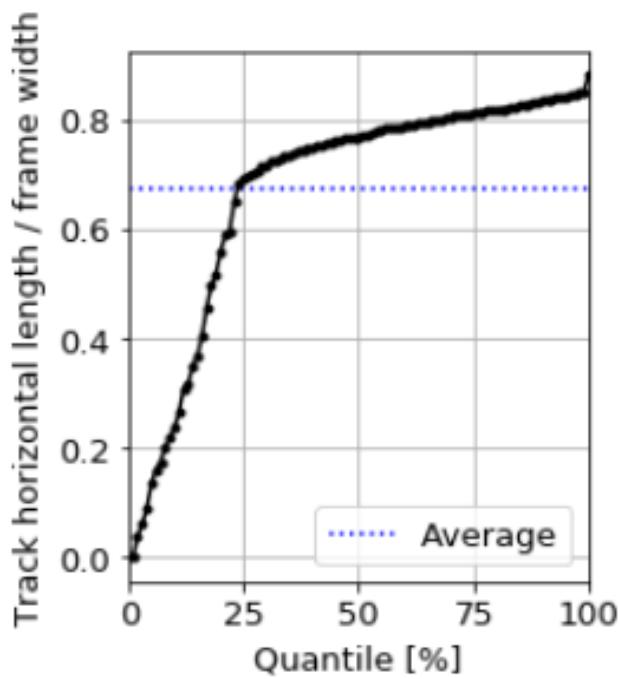
Does it work?

- Do we track the whole path?

- Fake tracks: false detections



- Failed tracks: missing detections, hidings



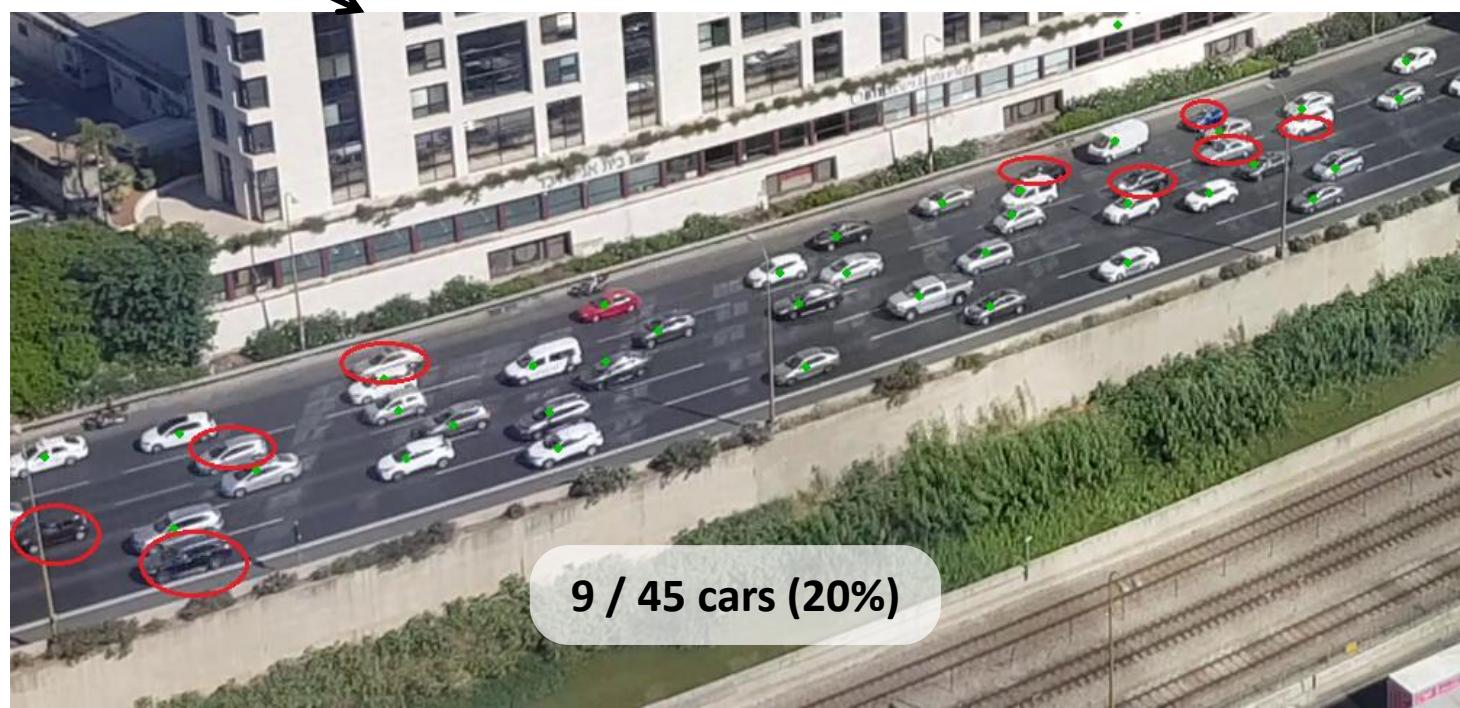
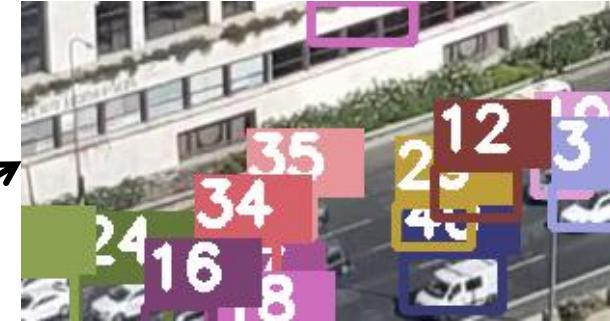
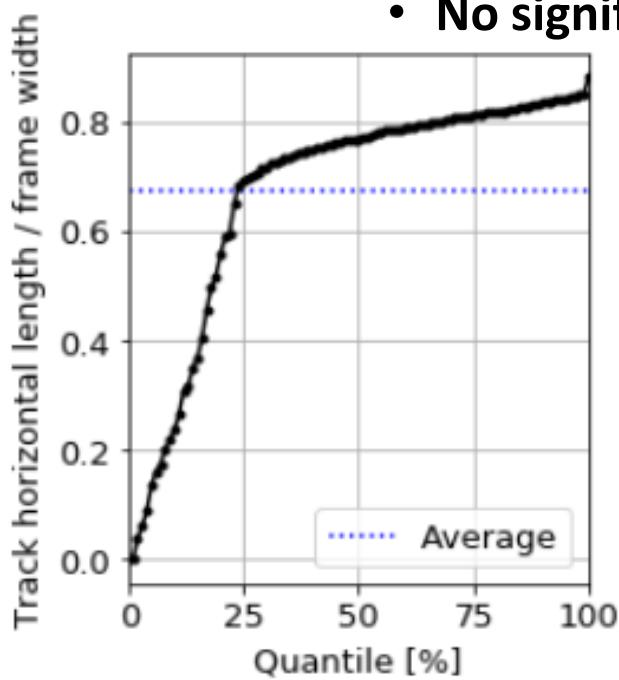
Does it work?

- Do we track the whole path?

- Fake tracks: false detections
- Failed tracks: missing detections, hidings

- **Eliminated by removing short tracks**

- **No significant bias expected**



Does it work?

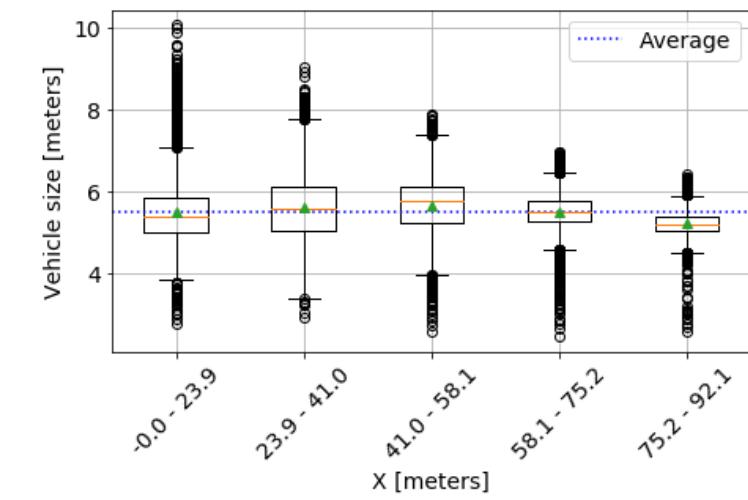
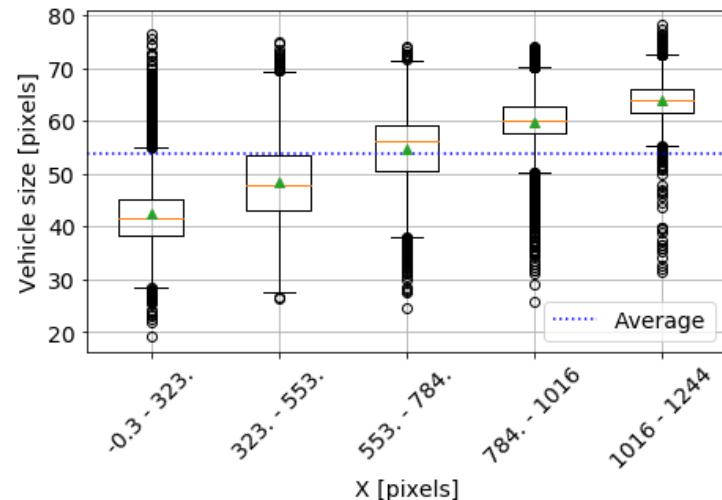
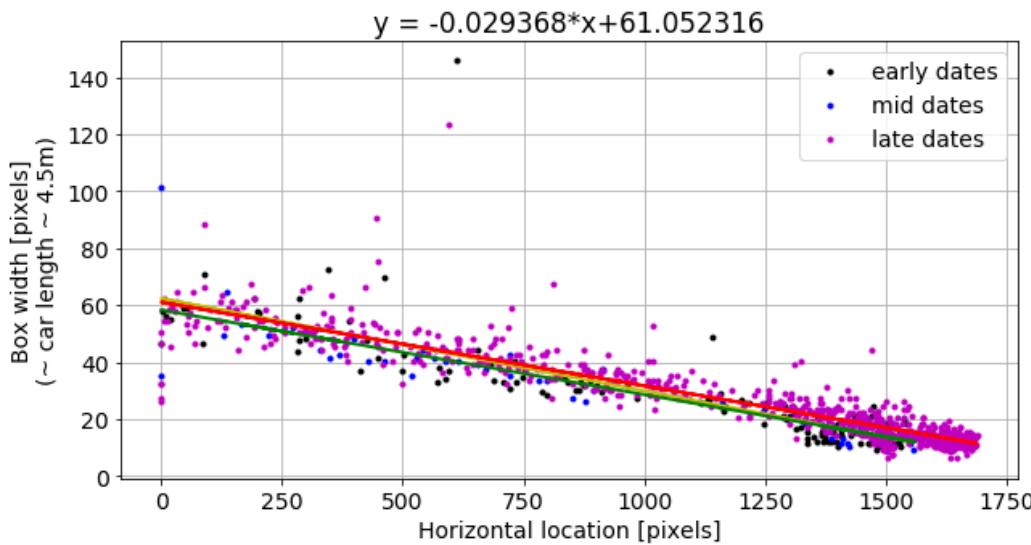
- **Running time** on my laptop:
 - Full frames: 1.2 frame/sec (~25 min/video)
 - **Cropped frames:** 3 frame/sec (~10 min/video)
- **Code profiling:**
 - **Do it** – there're always unexpected pitfalls
 - E.g. data-frames reallocation, GPU memory cleaning
 - **%lprun** in Jupyter Notebook
 - Sync GPU before measuring time



Analysis

Data representation

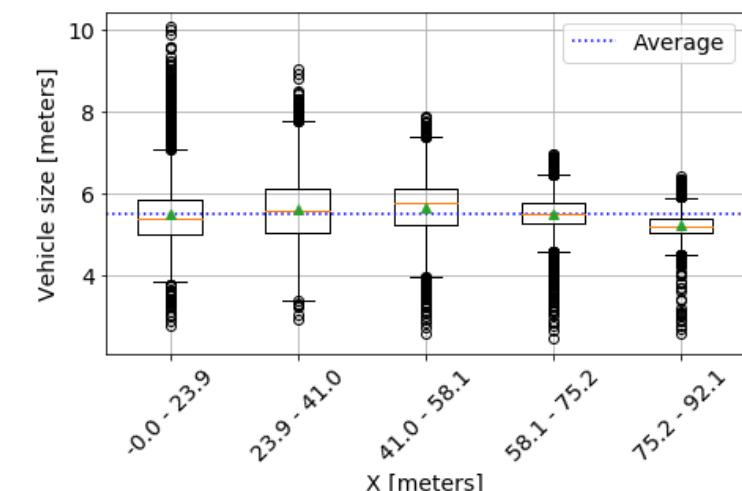
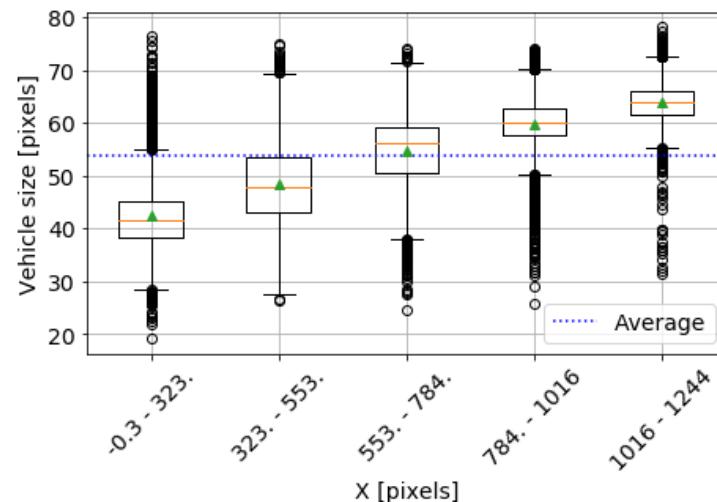
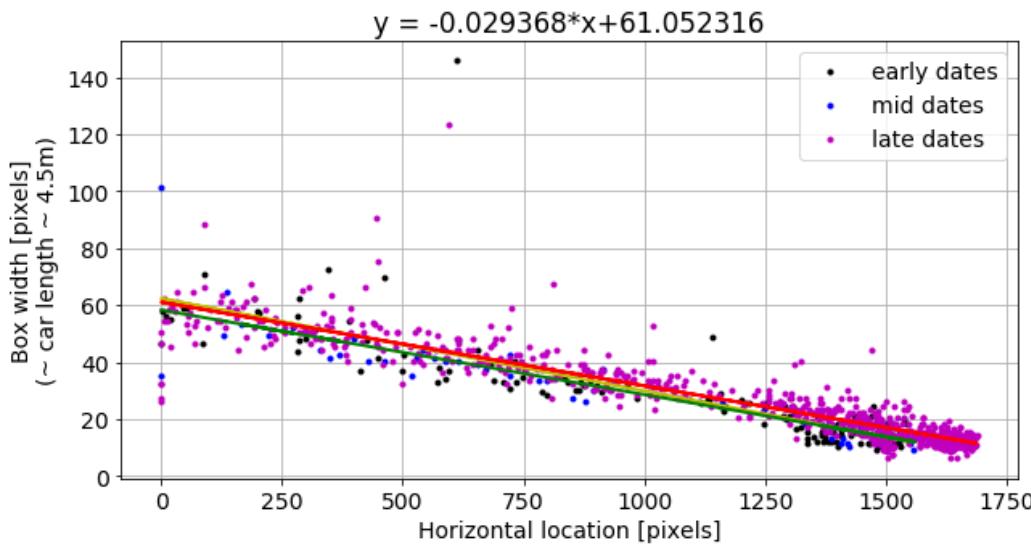
| Data structure | Keys | Values | Processing needed |
|-------------------|---------------|--------|--|
| Raw tracking logs | time, vehicle | x, y | Non-linear transformation: pixels → meters |
| | | | |



Data representation

| Data structure | Keys | Values | Processing needed |
|-------------------|------------------|----------------|--|
| Raw tracking logs | time, vehicle | x, y | Non-linear transformation: pixels → meters |
| Per-vehicle* | vehicle, x-range | time, y, speed | Interpolation to grid points |
| | | | |

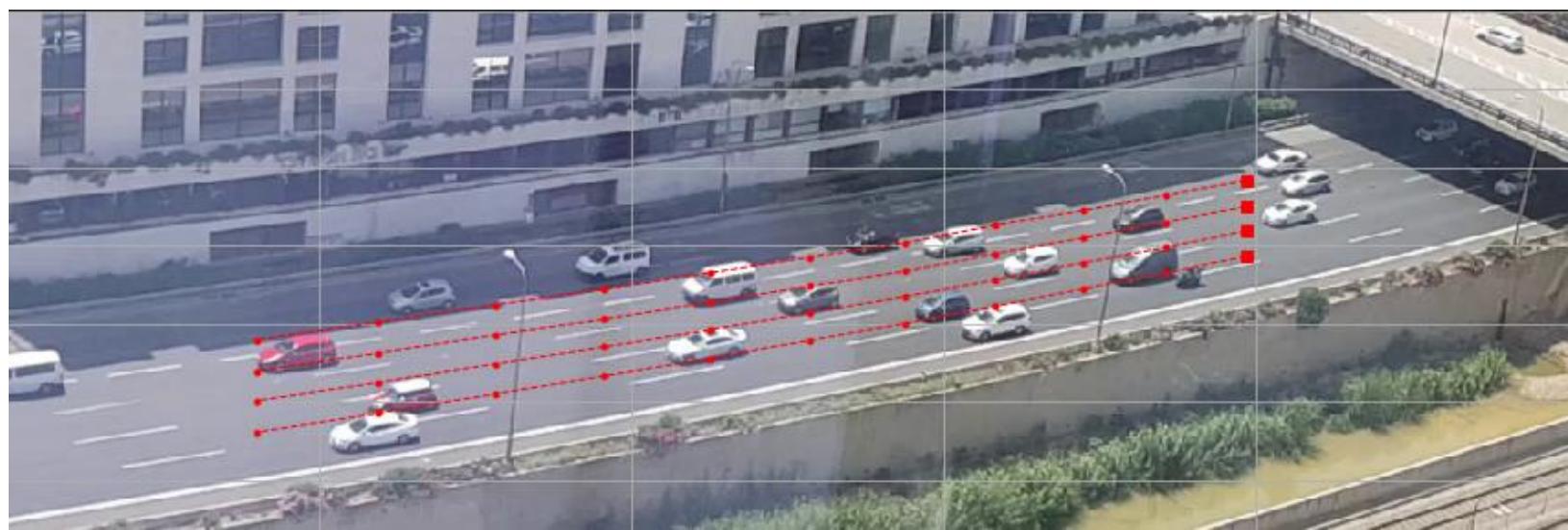
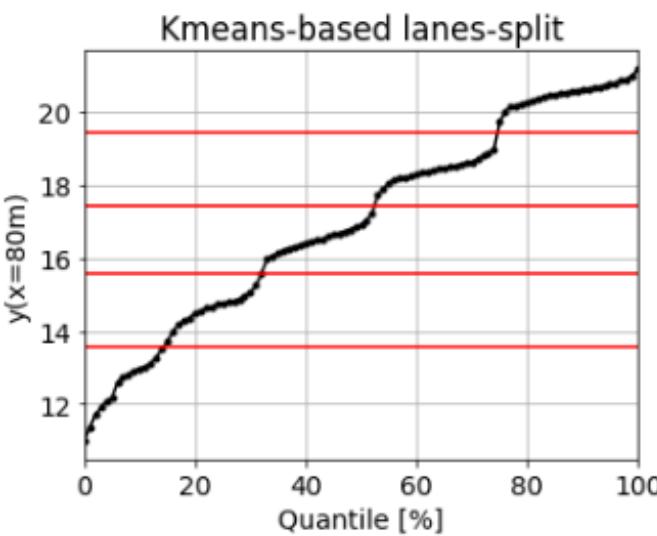
* Can be concatenated over all videos



Data representation

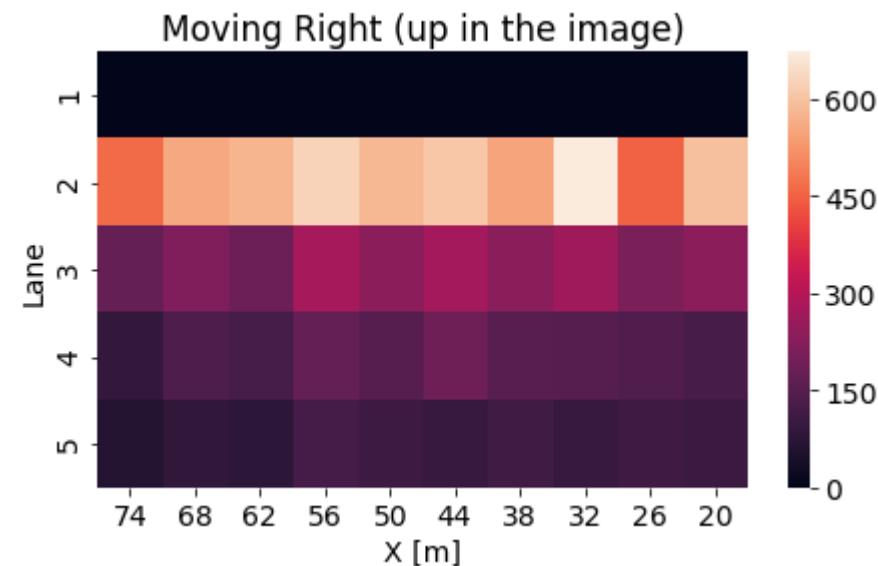
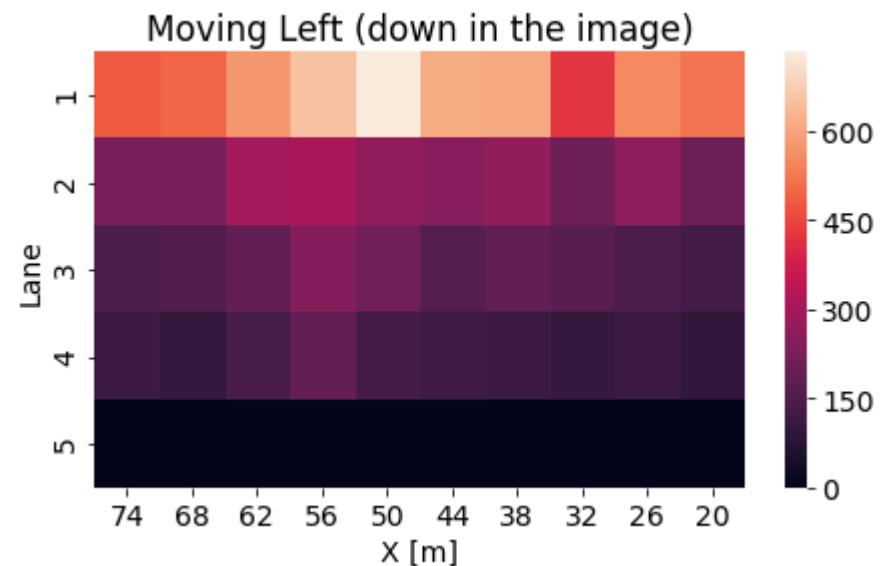
| Data structure | Keys | Values | Processing needed |
|-------------------|---------------------|-------------------|---|
| Raw tracking logs | time, vehicle | x, y | Non-linear transformation: pixels → meters |
| Per-vehicle* | vehicle, x-range | time, y, speed | Interpolation to grid points |
| Spatial* | time, lane, x-range | n_vehicles, speed | Clustering to lanes (K-means with uniform split of the vertical locations range as initial centers) |

* Can be concatenated over all videos



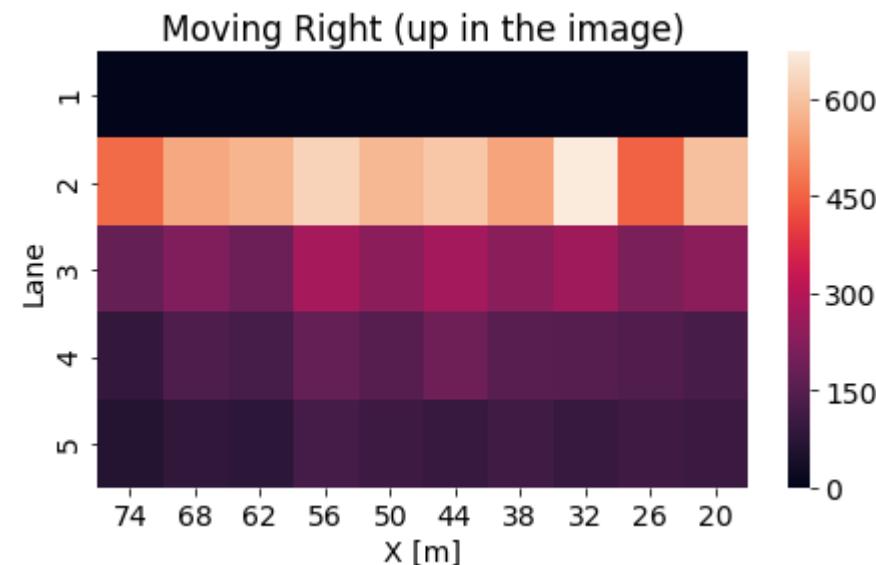
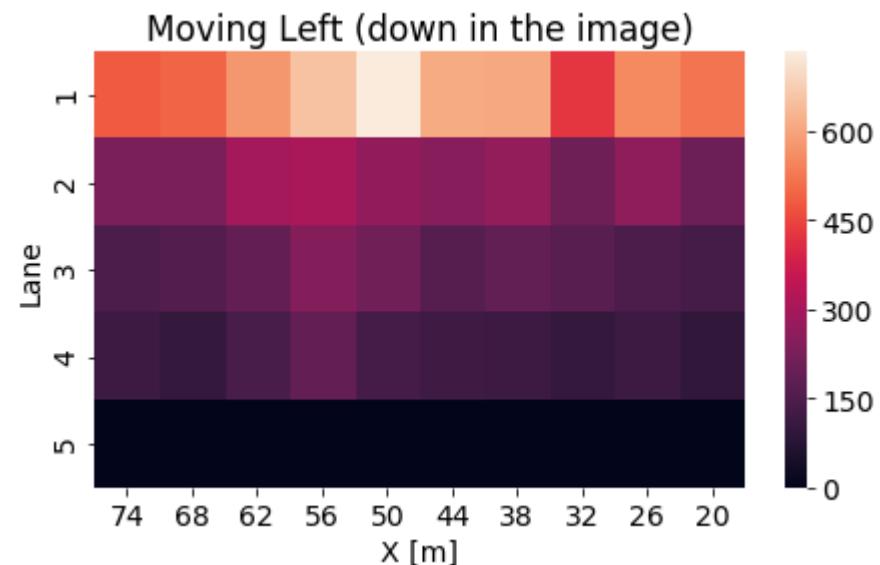
Lane transitions

- 21K lane-transitions detected
- Mostly in the right – near the entrances & exits



Lane transitions

- 21K lane-transitions detected
- Mostly in the right – near the entrances & exits
- Manual sanity check: 5/10 FP transitions ☹
 - Close to edge of a lane (e.g. high vehicles)
 - Tracking confusions on dense traffic



Summary

Main contributions

- Original dataset of traffic in a major road
- Detection
 - Small, crowded objects in noisy images
 - Trainable from little data
- Tracking
 - Fast-moving objects in low frame-rate videos
 - Robustness to missing detections
- TODO analysis contribution

Methodological insights (nothing surprising...)

- Data are everywhere
- **Draw/plot/print everything:** it's often not what you expected
- **Transfer learning** is insanely powerful
- **Sampling** may be crucial for training
- Have a **pipeline** for anything useful – you'll always run it again
- **Running time profiling:** many unexpected computational pitfalls

Possible extensions

- Data:
 - Tag before & after Hashalom interchange
- Detection:
 - Tag more data from "difficult" videos
 - Further tuning of hyper-params & architecture
 - Further output filtering (e.g. stricter manual constraints)
- Tracking:
 - Exploit size in addition to location
 - Non-linearly-additive models instead of KF:
 - Model parameters in meters (not pixels) – requiring location-dependent noise matrix
 - Large negative acceleration is more probable in higher speeds and in direction of motion
 - *Deep SORT*: exploit visual info
- Analysis:
 - TODO

References

- [Project repo in GitHub](#)
- [VGG Image Annotator](#) / Abhishek Dutta et al., Oxford
- [Guide to build Faster RCNN in PyTorch](#) / Fractal research group
- [Object detection and tracking in PyTorch](#) / Chris Fotache
- [Simple Online and Realtime Tracking](#) / Alex Bewley et al.
- [Kalman Filter](#) / Wikipedia



Thank you