First steps in traffic analysis

vehicle detection and simple tracking

Michał Drzał

michal.drzal@gmail.com, github.com/xmichaelx

Why I am interested in traffic analysis

- Combination of image processing & machine learning
- (ex) roommate works in transportation and asked me once if counting vehicles could be automated and got me thinking about the subject
- YOLO, RCNNs et consortes are so cool that I really wanted to use them in a project

Detection & tracking

Say we have algorithm able to detect vehicles in image

How to extend this to tracking vehicles using existing libraries and learn something in the process?

I hear and I forget. I see and I remember. I do and I understand.


This should be easy,
we've got detections all we need to
do is match them together.

Evaluated solutions

- Detection
  - YOLO
- Tracking
  - Trackers available in OpenCV
  - IOU tracking

All solutions tested on single test video

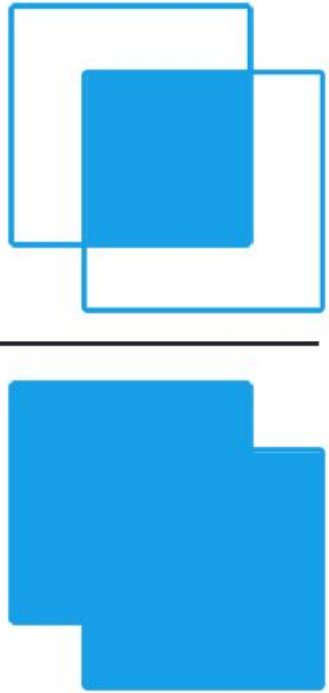
www.youtube.com/watch?v=MNn9qKG2UFI
(author Karol Majek)

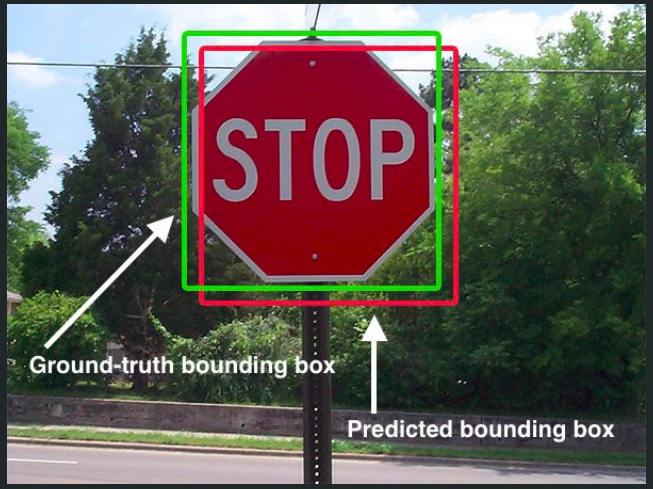
Good places to look for data

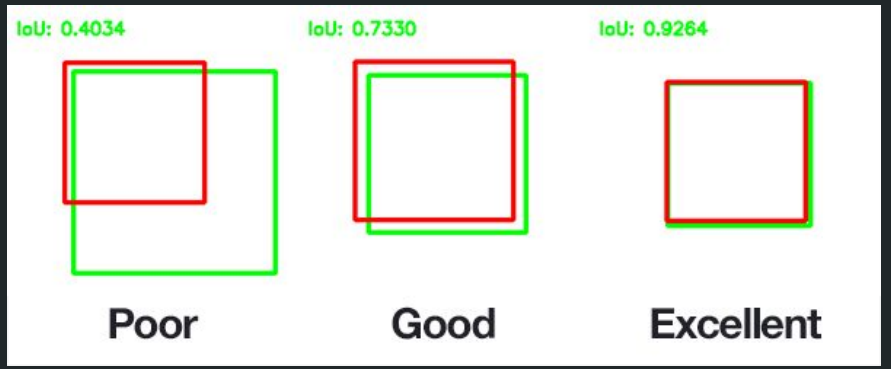
- **UA-DETRAC Benchmark Suite**
  - detrac-db.rit.albany.edu
  - Labeled, publicly available
- Brno University of Technology
  - http://www.fit.vutbr.cz/research/groups/graph/pclines/ (each entry contains links to videos)
  - https://medusa.fit.vutbr.cz/traffic/datasets
  - Partially labeled, publicly available

IOU
(Jaccard index)

$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$







Source: en.wikipedia.org/wiki/Jaccard_index

Detection

YOLO v3

https://github.com/qqwweee/keras-yolo3

Why YOLO?

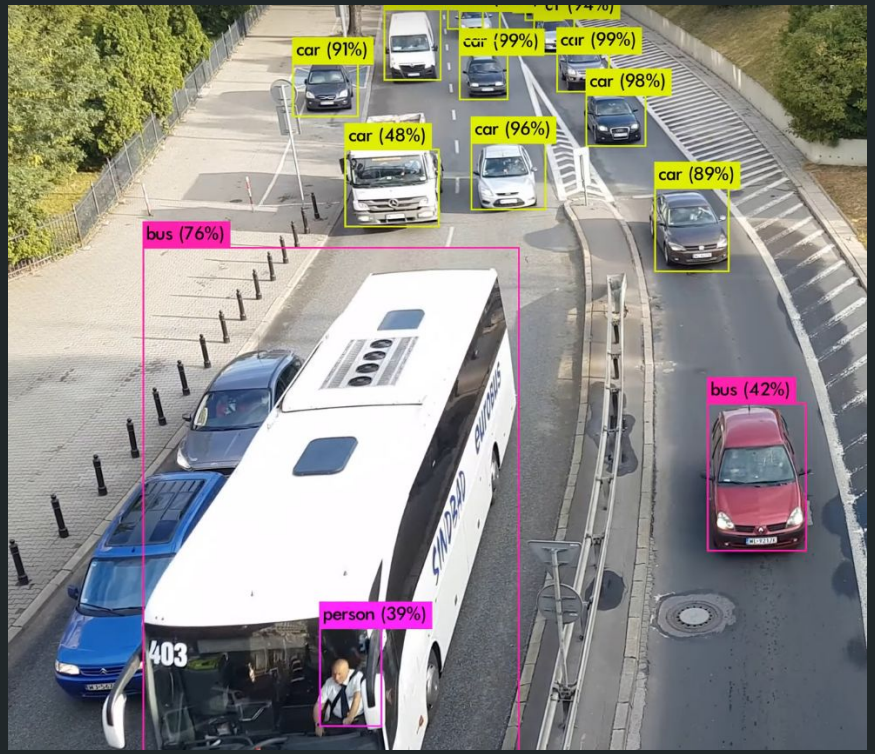
- Much faster than other detection providers, which is important for testing different tracking mechanisms
- Works really well out of the box

Why not YOLO?

- Bounding boxes cover vehicle with its surroundings
- Masks produced by Mask-RCNN are much better descriptors

For comparison of YOLO with other architectures see: https://medium.com/@jonathan_hui/object-detection-speed-and-accuracy-comparison-faster-r-cnn-r-fcn-ssd-and-yolo-5425656ae359

Also checkout BoxCars: https://arxiv.org/pdf/1703.00686.pdf



Video: www.youtube.com/watch?v=lekdUXv82xQ

Tracking using OpenCV trackers

Evaluation of top 3 trackers from:

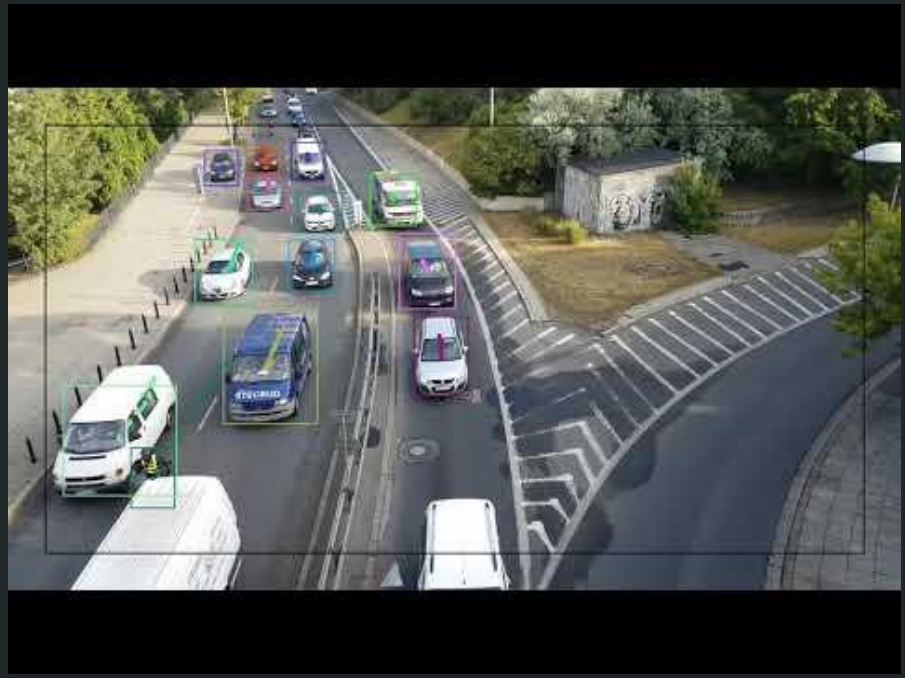
www.pyimagesearch.com/2018/07/30/opencv-object-tracking

CSRT, KCF and MOSSE

Simple algorithm:

1. Detection phase every **n** frames
2. For each frame use tracker to find new position for every track
3. For each frame compare between tracks discovered in recent detection phase and previously discovered tracks - if IOU is high enough join newly discovered track with largest IOU

CSRT



Tracking using OpenCV trackers

MOSSE

Tracking using OpenCV trackers

KCF

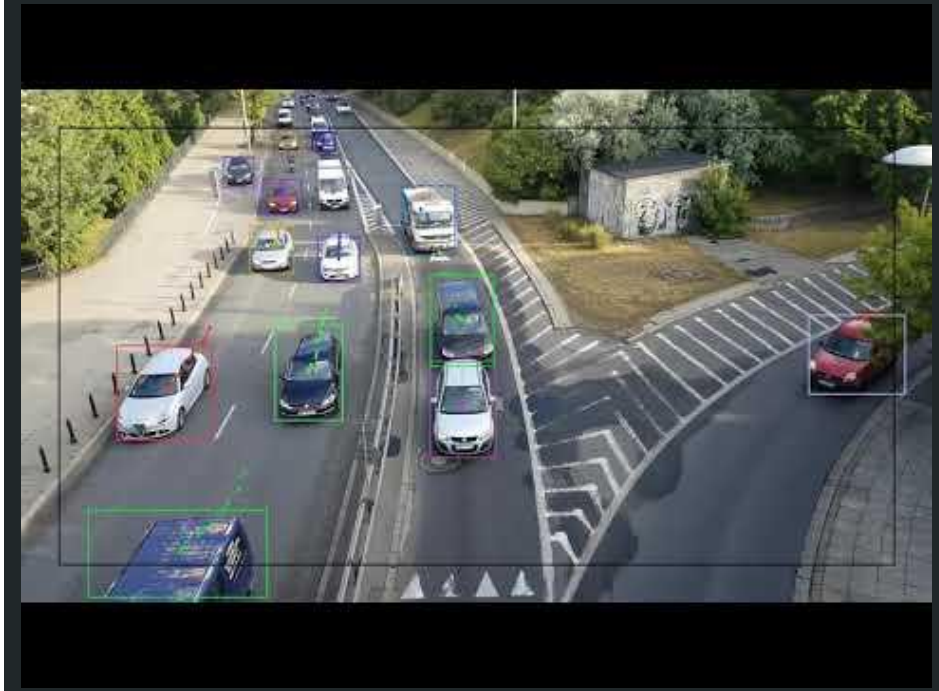
IOU tracker

Simple algorithm using tracking by detection

Algorithm 1 IOU Tracker

Inputs:
$D = \{D_0, D_1, ..., D_{F-1}\} =$
$\{\{d_0, d_1, ..., d_{N-1}\}, \{d_0, d_1, ..., d_{N-1}\}, ...\}$
Initialize:
$T_a = \emptyset, T_f = \emptyset$
$D = \{\{d_i | d_i \in D_j, d_i \geq \sigma_l\} | D_j \in D\}$
for f = 0 to F :
  for $t_i \in T_a$:
    $d_{best} = d_j$ where $max(IOU(d_j, t_i)), d_j \in D_f$
    if $IOU(d_{best}, t_i) \geq \sigma_{IOU}$:
      add d_{best} to t_i
      remove d_{best} from D_f
    else
      if $highest\_score(t_i) \geq \sigma_h$ and $len(t_i) \geq t_{min}$:
        add t_i to T_f
      remove t_i from T_a
  for $d_j \in D_t$:
    start new track t with d_j and insert into T_a
for $t_j \in T_A$:
  if $highest\_score(t_i) \geq \sigma_h$ and $len(t_i) \geq t_{min}$:
    add t_i to T_f
return T_f

http://elvera.nue.tu-berlin.de/files/1517Bochinski2017.pdf



Discoveries

- IOU tracking works surprisingly well & essentially has zero overhead compared to detector
- YOLO frequently has 1-2 frame gaps in detection for otherwise smooth tracks
- Slower trackers like KCF or CSRT yield worse FPS than tracking by detection
- Built-in OpenCV trackers have difficulty figuring out when to stop tracking

Code available at:

https://github.com/xmichaelx/TrafficAnalysis

Conclusions

- Basic tracking can be achieved using MOSSE tracker with frequent detection updates
- Adding motion model to predict next bounding box position could improve matching process
- Re-identification of tracks reappearing after occlusions is required for robust solution

What will I try next?

- Add motion model to IOU tracker (e.g. Kalman filter)
- Research better descriptors than 2D bounding boxes
- Research good descriptors for re-identification after occlusion
- Most importantly - work on bigger datasets like Detrac
- Start with Superpoint & add detection
- Finding vanishing points