TALLINN UNIVERSITY OF TECHNOLOGY
Faculty of Information Technology
Department of Software Science

Estimating object detection reliability for TTU "Iseauto" self-driving car

Master's thesis

Artur Vainola
153015

Supervisor
Juhan-Peep Ernits, PhD

Tallinn 2018

Declaration

I declare that this thesis is the result of my own research except as cited in the references. The thesis has not been accepted for any degree and is not concurrently submitted in candidature of any other degree.

Artur Vainola

June 1, 2018

....

(Signature)

Abstract

Emergence of convolutional neural networks has sparked the computer vision field, resulting in abundance of different object detecting methods. Performance of object detectors is evaluated using benchmark datasets, but metrics they use are very general and reflect ability to detect wide range of classes. In a context of safety-critical applications like autonomous vehicles, minimizing misdetection rate is more important than being able to differentiate bus from truck. There is no known method for evaluating performance of object detectors in a context of self-driving car.

We develop an object detector evaluation method that considers aspects of the context of autonomous vehicles. Proposed method requires building a benchmark dataset by recording and annotating context-specific data and makes it possible to estimate reliability of different detectors by comparing the output against ground truth.

We then use our developed method to analyze performance of Fast Region-based Convolutional Neural Network (Fast-RCNN), Single-Shot Detector (SSD) and You Only Look Once v2 (YOLOv2) in the context of TTÜ "Iseauto" project, estimating reliability and providing basis for choosing the best option.

The thesis is in English and contains 34 pages of text, 5 chapters, 16 figures, 2 tables.

Annotatsioon

Objektituvastuse töökindluse hindamine TTÜ "Iseauto" kontekstis

Konvolutsiooniliste närvivõrkude kasutuselevõtt on suurendanud huvi masinnägemise valdkonna vastu ja loonud kasvupinnase erinevate objektituvastuse meetodite tekkeks. Erinevate meetodite võrdlemiseks kasutatakse võrdlusandmekogusid, aga nendes kasutatuav meetrika on väga üldine ja mõõteb pigem tuvastajate võimekust tuvastada suurt hulka objektide kategooriaid. Autonoomsete sõidukite kui turvakriitiliste rakenduste kontektstis on olulisem minimeerida tuvastamata objektide hulka kui teha vahet bussil ja veoautol. Teadaolevalt pole ühtegi head meetodit, kuidas mõõta objektituvastajate töökindlust isesõitvate autode kontekstis.

Käesolevas väitekirjas arendatakse välja objektituvastajate töökindluse mõõtmise meetod, mis võtab arvesse autonoomsete sõidukite konteksti. Pakutav meetod baseerub kontekstispetsiifilise võrdlusandmekogu loomisel, mille loomiseks on tarvis koguda videosalvestisi ja neid annoteerida. Meetod võimaldab hinnata erinevate objektituvastajate töökindluste võrreldes nende poolt tuvastatud objekte annoteeritud objektidega.

Seejärel kasutatakse arendatud meetodit, et analüüsida Fast-RCNN, SSD ja YOLOv2 objektituvastajate täpsust TTÜ "iseauto" projekti kontekstis, hinnates nende töökindlust ja luues aluse parima objektituvastaja valikuks.

Lõputöö on kirjutatud Inglise keeles ning sisaldab teksti 34 leheküljel, 5 peatükki, 16 joonist, 2 tabelit.

List of Acronyms

AP average precision

Caffe Convolutional architecture for fast feature embedding

CNN Convolutional Neural Network

COCO Common Objects in Context

DPM Deformable Parts Model

Fast-RCNN Fast Region-based Convolutional Neural Network

FN false negative

FP false positive

FPS frames per second

HOG Histogram of Oriented Gradients

GPS Global positioning system

GPU graphics processing unit

GT ground truth

GUI graphical user interface

ILSVRC ImageNet Large Scale Visual Recognition Competition

IMU Inertial measurement unit

IoU Intersection over Union

mAP mean average precision

R-CNN Region-based Convolutional Neural Networks

ROS Robot Operating System

SSD Single-Shot Detector

SVM Support vector machine

TN true negative

TP true positive

VATIC Video Annotation Tool from Irvine, California

VOC PASCAL Video Object Classes

WHO World Health Organization

YOLOv2 You Only Look Once v2

Contents

List of Figures

List of Tables

1. Introduction

According to World Health Organization (WHO) [2], more than 1.2 million people die each year as a result of road traffic crashes. Nearly half of those dying on the world's roads are "vulnerable road users": pedestrians, cyclists, and motorcyclists. Main cause of accidents is human error: speeding, distracted driving, driving under the influence, etc.

There is a strong belief that artificial intelligence can help reduce the impact of traffic accidents on mortality. In addition to making traffic safer it could bring many more benefits like greater accessibility, better road efficiency, positive impact on the environment by helping reduce the energy footprint of transportation, increased productivity, reduced time spent on everyday commuting by shortening the journey and making it possible to use the time for other tasks.

Reduced cost and increased availability of sensing technologies and computing power has brought about dramatic advancements in autonomous systems research and built a solid foundation for autonomous vehicle development. Nowadays, most car manufacturers already have "drivers assistant" systems installed into their cars, helping drivers with tasks of different complexity from simple parking assistance to so called "semi-autonomous" driving systems. Furthermore, many of them are putting lots of resources and effort into shifting more and more tasks from human driver to onboard computers, hoping to eventually build cars that drive without the need of human intervention.

However, building a self-driving car that exceeds human driving performance is far from an easy task. In order to be able to operate autonomously, it needs to have a good understanding of surrounding world. To achieve this, complex sensory systems have been developed, utilizing different sensors like monocular video cameras, stereo-cameras, IR-cameras, TOF-cameras, RADARs, LIDARs and ultrasonic sensors. While LIDARs are very good for localization using 3D point cloud maps, it alone does not provide a performant object detection solution. Therefor most systems use video cameras, alone or in combination with LIDARs, for object detection and tracking.

When talking about object detection we usually refer to a task of localizing and classifying multiple objects of single or multiple classes on an image or video frame. While computers have been used for decades to detect objects on video images, demand for fast and reliable object detection method and recent advancements in machine learning technologies have brought much attention to computer vision field. This has resulted in new and better object detecting methods emerging every year. Abundance of methods [3] [4] [5] [6] have made it possibility to choose an algorithm that best fits your application, but making a choice requires understanding of performance metrics. For object detection, most popular metric used in all known evaluation datasets is mean average precision (mAP), which generalizes well, but has few slightly different implementations making it difficult to interpret. With available benchmark datasets [7] [8] [9] we can measure general performance of each method, but when the context is specific, general score might not give you the best information. To find best method for specific task, we need to evaluate each method with relevant data and metrics.

TTÜ in collaboration with Silber Auto is building a autonomous self-driving vehicle called "Iseauto". It is an educational project that challenges the ability to build a self-driving car within one year, using mainly open-source software and very limited resources. Initially the car must be able to drive autonomously only on a fixed route inside TTÜ campus. It is a custom built car based on Mitsubishi i-MiEV electric car and has different sensors (LIDARs, cameras, Global positioning system (GPS), Inertial measurement unit (IMU), etc) providing information about surrounding world and an onboard computer to process the data and provide high level driving commands.

With TTÜ “Iseauto“ project we are facing the problem of finding suitable object detection method for detecting vehicles and pedestrians. While fixing the route reduces the complexity of the problem as we can focus on performance within certain surrounding terrain, we still have to deal with changes in environment due to weather conditions and seasons. In addition we need to consider the speed and computation complexity of possible method as it has to run in real-time and with multiple cameras simultaneously.

Autoware, an open-source software platform for urban autonomous driving, is used as the main software platform for TTÜ “Iseauto” self-driving car. Autoware has integration with different object detection methods [3] [4] [5] [6] and we need to evaluate how well these methods performs in a safety-critical application with different light and weather conditions. Based on this we formulate following research question:

- REQ1: How to measure the performance of an object detector for a self-driving car?

- REQ2: Which metrics should be used to evaluate the performance considering the context of "Iseauto project"?
- REQ3: How does each of the available detectors performs on our data?
- REQ4: How compute-intensive is each object detector and how fast they can process real-time video stream on available graphics processing unit (GPU)?

General goal is to develop a method for estimating object detection reliability in different conditions. To achieve the general goal we will:

- develop a method to easily analyse performance of existing methods with different configurations and scenes
- record scenes from actual the route in different weather and light conditions to increase the variation of the sample set
- use the developed method to create ground truth data by labelling recorded videos,
- analyse which performance metrics are relevant in the context of self-driving cars,
- use the developed method to perform ananlysis of different scenes to understand the reliability of existing object detectors,
- analyze the speed and efficiency of each object detector running on two different GPUs.

Using this approach, we will be able to suggest object detection method and configuration for detecting vehicles and pedestrians in a fixed route inside TTÜ campus considering detection reliability, speed and efficiency of the methods.

In next chapter, Chapter 2, we introduce the software platform that we are using for development of the car, give a brief overview about the available object detectors and benchmark datasets, and explain what metrics are used to compare performance of different object detectors. Chapter 3 describes the process of building our own benchmark dataset. In Chapter 4 we explain what metrics we used for analyzing the data, present the results of performance analysis and show the records of speed and efficiency observations.

2. Background and related work

2.1 Autoware

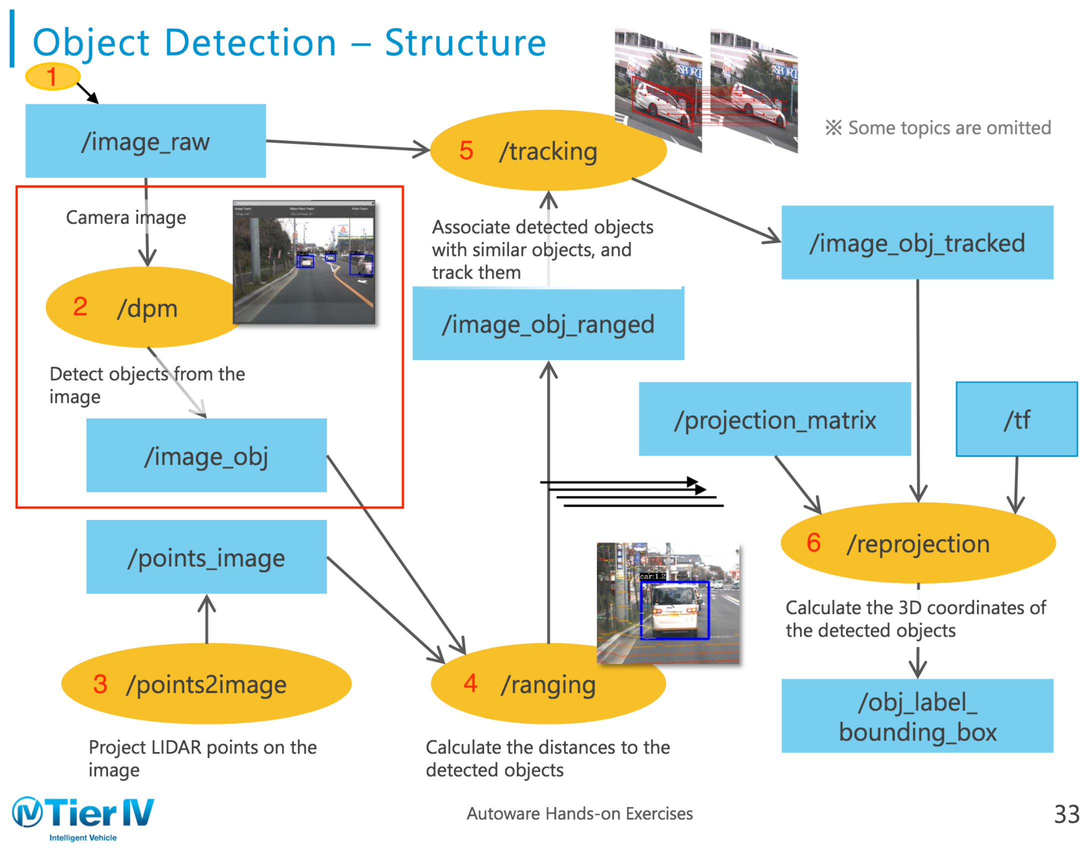
As a development platform, "Iseauto" project is using Autoware [10], an open-sourced software platform for urban autonomous driving. Autoware is built on top of Robot Operating System (ROS) [11], an open-source meta-operating system for robots. ROS provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers. The Robot Operating System (ROS) runtime "graph" is a peer-to-peer network of processes (called ROS nodes) that are loosely coupled using the ROS communication infrastructure. ROS implements several different styles of communication, including synchronous communication over services, asynchronous streaming of data over topics, and storage of data on a Parameter Server [12].

Autoware offers solutions for different subtasks required for autonomous driving. Every subtask is separate ROS node that subscribes to all topics it needs as an input and publishes its output to another topic that other nodes use as input. Object detection and tracking subsystem on Autoware consists of 6 different nodes:

1. camera node - for mapping raw camera data to ROS topic;
2. object detection node - for detecting objects on a video frame;
3. points2image node - for projecting LIDAR points on the video image;
4. ranging node - for calculating distance to detected objects;
5. object tracking node - for tracking detected objects and;

6. re-projection node - for calculating 3D coordinates of detected objects;

, and are illustrated with relations to each-other on the Figure 2.1.



Object Detection – Structure
1
/image_raw
5 /tracking
※ Some topics are omitted
Camera image
Associate detected objects with similar objects, and track them
/image_obj_tracked
2 /dpm
/image_obj_ranged
Detect objects from the image
/projection_matrix
/tf
/image_obj
6 /reprojection
/points_image
Calculate the 3D coordinates of the detected objects
3 /points2image
4 /ranging
/obj_label_ bounding_box
Project LIDAR points on the image
Calculate the distances to the detected objects
Tier IV Intelligent Vehicle
Autoware Hands-on Exercises
33

Figure 2.1: Object tracking module structure in Autoware [1]

This work focuses mainly on object detection node (2), which uses output of camera node (1) as input and publishes its output to a topic that is input for ranging node (4). Ranging node (4) output together with camera node (1) output are inputs for object tracking node (5), which in turn produces input for re-projection node (6). Current Autoware implementation has integration with four different object detectors: DPM [3], Fast-RCNN [4], SSD [5] and YOLOv2 [6]; and next section gives a brief overview of each.

2.2 Object detectors

Object detectors use a trained model to detect objects on an image or a video frame. Training a model is a method of supervised learning requiring considerable amount of labeled data. Usually the context where people need to detect objects is in some way specific and self-trained model could have an advantage when designed for specific task and trained with relevant data. But annotating enough data to train a performant object detection model is expensive. Even though usage of bounding-box is relatively inexpensive compared to pixel-based annotation, it still requires much more effort than tagging an image with a class. A rough rule of thumb is that a supervised deep learning algorithm will generally achieve acceptable performance with around 5,000 labeled examples per category and will match or exceed human performance when trained with a dataset containing at least 10 million labeled examples [13]. Recording and annotating[1] such dataset of representative sample of at least 3 categories[1] did not fit into the time-frame of this work and we had to use one of the pre-trained models.

2.2.1 DPM

Before the arrival of Convolutional Neural Network (CNN)s, state-of-the-art object detection method was DPM [3]. It is an object detection algorithm that uses a sliding window method to reduce the complex object detection task to easier binary classification task. It is based on a Dalal-Triggs detector [14] that uses a single filter on Histogram of Oriented Gradients (HOG) features to represent an object category and a linear Support vector machine (SVM) classifier with a sliding window approach, where a filter is applied at all positions and scales of an image.

Emergence of CNN and deep learning brought object detectors performance to a new level and earlier state-of-the-art models like DPM [3] aren't able to compete with available CNN object detectors [4][5][6]. Furthermore, it is being deprecated in the Autoware in the near future and therefor is not evaluated in this work.

2.2.2 Fast-RCNN

Fast-RCNN [4] is an object detector that is built on previous work of Region-based Convolutional Neural Networks (R-CNN) [15] and SPP-net [16]. Compared to its predecessor R-CNN, it improves training and testing speed while also increasing detection accuracy. It uses a single-stage training with multi-task loss that backpropagates to all network layers. Instead of sliding window method it uses a Selective Search algortihm [17] for region proposal, reducing the number of prior bounding-boxes typically close to

[1] 100 frames per hour and 2 objects per frame is a rough average measured during annotation.

[1] 3 object categories: cars, pedestrians and cyclist; is an absolute minimum required set.

2000. Rather than feeding all the region proposals to CNN, it uses SPP-net [16] approach to calculate CNN representation of the image only once and use that to calculate the CNN representation for each patch. It is implemented in Python and C++ using Convolutional architecture for fast feature embedding (Caffe) [18], framework for state-of-the-art deep learning algorithms and a collection of reference models.

Fast-RCNN was state-of-the-art on VOC 2012 when the article was released, but newer methods [6] [5] have shown better results. Furthermore, it is much slower compared to single-shot object detectors, making it questionable option for self-driving car.

2.2.3 SSD

SSD [5] is a method for detecting objects in images using a single deep neural network. It discretizes the output space of bounding boxes into a set of default boxes over different aspect ratios and scales per feature map location. At prediction time, the network generates scores for the presence of each object category in each default box and produces adjustments to the box to better match the object shape. Additionally, the network combines predictions from multiple feature maps with different resolutions to naturally handle objects of various sizes. SSD is simple relative to methods that require object proposals. It completely eliminates proposal generation and subsequent pixel or feature resampling stages and encapsulates all computation in a single network, making it easy to train and straightforward to integrate into systems that require a detection component.

SSD uses a weighted sum of the localization loss and the confidence loss as overall objective loss function and hard negative mining to achieve faster optimization and more stable training. To make the model more robust to various input object sizes and shapes, each training image is randomly sampled one of the following options:

- Use the entire original input image.
- Sample a patch so that the minimum jaccard overlap with the objects is 0.1, 0.3, 0.5, 0.7, or 0.9.
- Randomly sample a patch.

Compared to methods that utilize an additional object proposal step, SSD has competitive accuracy on VOC, COCO, and ILSVRC but is much faster.

2.2.4 YOLOv2

YOLO9000 is a single-shot object detector that can detect over 9000 object categories. It is built on YOLOv2, a model with various improvements to the YOLOv2 detection method.

Structure of the YOLOv2 allows it to change the input image size on the fly. Instead of fixing the input image size, they change the network every few iterations, forcing the network to learn to predict well across a variety of input dimensions. This novel method called multi-scale training allows YOLO9000 to run at varying sizes, offering an easy tradeoff between speed and accuracy.

YOLOv2 uses a method to jointly train on object detection and classification. Using this method it trains simultaneously on the COCO detection dataset [8] and the ImageNet [9] classification dataset. As classification dataset has much wider and deeper range of labels, they use a so called hierarchical classification instead of simple softmax. Usage of WordTree, a hierarchical tree built from WordNet [19] concepts, allows it to train classification on both detection and classification data. For data containing only classification information they backpropagate only classification error. This joint training allows YOLOv2 to predict detections for object classes that don't have labelled detection data.

For predicting bounding-boxes YOLOv2 uses anchor-boxes, but instead of choosing priors by hand, it runs k-means clustering on the training set bounding-boxes to automatically find good anchor-boxes. Use of anchor-boxes gives it increase in recall from 81% to 88% compared to predecessor. Objectness prediction predicts the Intersection over Union (IoU) of the ground truth and the proposed box and the class predictions predict the conditional probability of that class given that there is an object.

YOLOv2 runs significantly faster the other state-of-the-art object detectors like Fast-RCNN [4] and SSD [5], making it good option for real-time applications like self-driving car.

2.3 Benchmark datasets

There are several freely available datasets like [7] [8] [9] [20] that anyone can use for training or evaluating their model. In addition to training and evaluation images, they usually contain also tools for measuring performance. Often providers of the datasets hold yearly challenges for different computer vision task like classification, detection and tracking. These challenges are considered as benchmarks for measuring performance of a model and top performers are titled as state-of-the-art.

2.3.1 VOC

One of the popular benchmark datasets for evaluating object detector performance is VOC Challenge [7]. It was object detection benchmark standard until 2012. It consists of two components: (1) a publicly available dataset of images and annotation, together with standardised evaluation software; and (2) an annual competition and workshop. A new dataset with ground truth annotation was released each year from 2005 until 2012 when it retired. It had two principal challenges:

- classification — does the image contain any instances of a particular object class? (where the object classes include cars, people, dogs, etc.)
- detection — where are the instances of a particular object class in the image (if any)?

There is complete annotation for twenty classes. For annotation they initially used Mechanical Turk, but as the quality of location data was too low, trained annotators were used for annotating ground-truth bounding-boxes. All images are annotated with axis-aligned bounding-boxes for every instance of the twenty classes. In addition to a bounding box for each object, attributes such as: ‘orientation’, ‘occluded’, ‘truncated’, ‘difficult’; are specified.

IoU with threshold of 0.5 is used for deciding if the prediction was a hit or miss. Detections output are assigned to ground truth object annotations satisfying the IoU criterion in order ranked by the (decreasing) confidence output. Ground truth objects with no matching detection are false negatives and multiple detections of the same object in an image are considered false detections, e.g. 5 detections of a single object count as 1 correct detection and 4 false detections. For performance evaluation it uses a mAP metric described in Section 2.4

2.3.2 COCO

Another popular dataset for training general object detection models is COCO [8]. It is a dataset with the goal of advancing the state-of-the-art in object recognition by gathering images of complex everyday scenes containing common objects in their natural context. Dataset contains 91 common object categories with 82 of them having more than 5,000 labeled instances. In total the dataset has 2,500,000 labeled instances in 328,000 images. COCO is significantly larger in number of instances per category than the VOC and ILSVRC. Objects are labeled using per-instance segmentations instead of bounding-boxes to aid in precise object localization. For annotation they used Amazon Mechanical Turk, but required workers to pass a training to increase the quality of annotations.

For performance evaluation COCO uses a slightly different mAP metric (Section 2.4) that averages the usual mAP over multiple IoU values. This mean that COCO mAP score shouldn't be directly compared with other datasets as they have different interpretations.

2.3.3 ILSVRC

ILSVRC [9] is a object category classification and detection benchmark dataset based on ImageNet, containing hundreds of object categories and millions of images. It is a object detection benchmark standard since 2012 when the VOC [7] retired. Similarly to VOC, ILSVRC consists of two components: a publically available dataset and an annual competition and corresponding workshop. The challenge has been held annually since 2010 and has become a standard benchmark for large-scale object recognition.

ILSVRC annotations are divided into two categories:

1. image-level annotation of a binary label for the presence or absence of an object class in the image,
2. object-level annotation of a tight bounding box and class label around an object instance in the image.

It has 60,658 training images with 478,807 training objects from 200 object classes. Images are annotated using Amazon Mechanical Turk with 3 stage quality pipeline, containing 'drawing', 'quality verification' and 'coverage verification' stages.

It penalizes the algorithm for missing object instances, for duplicate detections of one instance, and for false positive detections. The final metric for evaluating an algorithm on a given object class is average precision over the different levels of recall achieved by varying the threshold for confidence score (Section 2.4). To measure overall performance of an object detector, mean average precision across all categories is used (Section 2.4).

2.3.4 KITTI

KITTI [20] object detection benchmark consists of 7481 training images and 7518 test images, comprising a total of 80.256 labeled objects. Datsets are captured by driving around the mid-size city of Karlsruhe, in rural areas and on highways. Up to 15 cars and 30 pedestrians are visible per image. Dataset provides accurate 3D bounding-boxes for object classes such as cars, vans, trucks, pedestrians,cyclists and trams. It is a 3D object detection benchmark evaluating both 3D location and orientation predictions. Ground-truth data is obtained by manually labeling objects in 3D point clouds produced by Velodyne LIDAR, and projecting them back onto the image. This results in tracklets with accurate 3D poses, which can be used to asses the performance of algorithms for 3D orientation estimation.

First, average precision (explained in Section 2.4) is used to evaluate 2D object detection performance. Detections are iteratively assigned to ground truth labels starting with the largest overlap, measured by bounding box intersection over union. Detections with IoU (explained in Section 2.4) > 0.5 count as true positives and multiple detections of the same object count as false positives. Finally they evaluate performance of jointly detecting objects and estimating their 3D orientation using their own measure called average orientation similarity (AOS) [20].

2.4 Metrics

Performance metrics are used to determine how well the output of an algorithm matches the ground truth. Different metrics capture different aspects of performance. Using different metrics help developers to optimize on specific performance aspects. In order to evaluate object detection performance, we need to understand how different metrics relate to different contexts, and focus only on the relevant metrics.

Back in 2002 Mariano, Vladimir Y., et al [21] defined 7 metrics that could be used for performance evaluation:

- Area-based recall for frame - a pixel-count-based metric that measures how well the algorithm covers the pixel regions of the ground-truth. Computed for each frame, and it is the weighted average for the whole data set.
- Area-based precision for frame - a pixel-count-based metric that measures how well the algorithm minimized false alarms. Computed for each frame, and it is the weighted average for the whole data set.
- Average fragmentation - a metric measuring how well the ground-truth object is not broken into pieces. Every output box that overlaps the ground-truth object is counted as a fragment for that object, even if the overlap is a single pixel. Intended to penalize an algorithm for multiple output boxes covering a ground-truth object.
- Average object area recall - a metric measuring the average area recall of all the ground-truth objects in the data set. The recall for an object is the proportion of its area that is covered by the algorithm's output boxes. The objects are treated equally regardless of size.
- Average detected box area precision - this metrics is a counterpart of the previous metric where the output boxes are examined instead of the ground truth objects. Precision is computed for each output box and averaged for the whole frame. The precision of a box is the proportion of its area that covers the ground truth objects.
- Localized object count recall - in this metric, a ground-truth object is considered detected if a minimum proportion of its area is covered by the output boxes. Recall is computed as the ratio of the number of detected objects with the total number of ground-truth objects. The ground-truth objects are treated equally regardless of size.
- Localized output box count precision - counterpart of previous metric. Counts the number of output boxes that significantly covered the ground truth. Output boxes are treated equally regardless of size.

IoU, also known as Jaccard Index or Jaccard similarity coefficient, is used in most object detectors and evaluation tools. It measures similiarities between two sample sets (in object detection context most commonly the bounding-box representations of location on an image) and is calculated by dividing the area of intersection with the area of union.

While in object detectors it is used to train the model to predict the confidence score, in evaluation tools it is used to decide if the prediction is a hit or miss.

All common benchmark datasets use mAP to evaluate the performance of object detector. average precision (AP) [22] is a metric that considers precision as a function of recall and computes average value of precision over the interval of recall from `r=0` to `r=1`. Sometimes an interpolated average precision [23] (often also called 11-point interpolated average precision) is used to reduce the impact of "wiggles" by defining the precision function as a the maximum precision over all recalls greater than `r`. For object detectors, AP is computed for each class and different levels of recall is achieved by varying treshold of confidence score. While mAP is usually simply a mean of the APs over all classes, COCO [8] averages it over IoU tresholds from 0.5 to 0.95 with a step of 0.5, to penalize high number of bounding-boxes with wrong classification.

3. Acquisition of validation data

To find out which of the available methods works best in our context, we decided to evaluate performance of the available object detectors on our own dataset. For this we needed to record video frames and annotate recorded videos.

3.1 Recording video frames

In order to create our own dataset, we first needed to gather sufficient amount of data. There was no platform available yet at that time, so data collecting was done using a regular passenger car with laptop running Autoware and sensors attached to the roof rack. Without special vehicle for data acquisition, we had to prepare and set up all the equipment for every session. This made it very time consuming and required presence of several team members, To get a good respresentative sample of data we wanted to drive around the planned route multiple times in different weather and light conditions and managed to organize 3 sessions: in November, in January and in April.

Unfortunately the selection of available cameras was very limited and videos were recorded with two different cameras:

- Pointgrey Flea3 USB3 camera @ 1280x1024(1,3MP) and
- Pointgrey Bumblebee2 stereo camera @ 648x488(0.3MP).

Furthermore, the quality of the videos recorded with Bumblebee2 was so low that we were not able to use it for the analysis and were forced to using only the Flea3.

All videos were recorder at 7 frames per second (FPS) to provide more variance while keeping the total number of frames from growing too high and not to overwhelm the annotation task with too much similar frames. Cameras were connected to ROS using `pointgrey_camera_driver` node, which published all frames into `/camera/image_raw` ROS topic. Messages from camera and other sensors were saved into ROS data file (rosbag).

3.2 Annotating

Annotating, often also called ground-truthing, requires a human operator to manually mark the boundaries of target objects, making pixel-level annotation very expensive task. Use of simple bounding-box instead of pixel-level representation allows less expensive ground-truthing and enables large volume and variety of video data to be ground-truthed. Therefor, majority of datasets and object detection algorithms use axis-aligned bounding-box to represent location of an object.

For annotating our recorded videos we considered different tools [24] [25], but decided to use docker image version [26] of Video Annotation Tool from Irvine, California (VATIC) [27]. It is an open-source video annotation tool built on MySQL database and Apache HTTP server, making it possible to run it on a server and access it anywhere, without the need of installing anything locally. It has integration with Amazon Mechanical Turk to oursource the annotation work, but in this work we used it in offline mode and did all the annotation by ourselves. Despite few minor shortcomings it has a simple and intuitive graphical user interface.

VATIC uses jpg images for each frame and provides tool to extract the frames from a video. It keeps the images in specific directory structure to make it convenient to read the frames in correct order. If you already have you frames as images, it has a tool to place the sequence of images into desired structure. As our videos were recorded into rosbag file as ros messages, we used `image_saver` node in ros `image_view` packages to extract each frame as jpg image and `turkic formatframes` tool to divide images into suitable directory structure for vatic.

VATIC uses class tag, axis-aligned bounding-box as representation of object location, and boolean flags to mark object as occluded or outside of the view frame. Despite the different way of representing the rectangle, usage of axis-aligned bounding-boxes made it easier to analyze the results, because all object detectors use the same abstraction for representing the predicted object location.

To create good quality annotations we focused on three properties: consistency, accuracy, exhaustiveness. Consistency was sustained by doing all of the annotations by single person. For high accuracy we looked through all the annotated recordings multiple times. Exhaustiveness was achieved by annotating everything that was interpretable as an object of one of the relevant classes.

One of the drawbacks of VATIC in a context of autonomous vehicles turned out to be the linear interpolation of bounding-boxes on intermediate frames. When annotating two non-consecutive frames, it uses linear interpolation to add annotations to all the frames inbetween. Linear interpolation works good on linearly moving objects, but on a moving vehicle where camera is also moving, even otherwise linearly moving object do not appear to be moving in a straight line with respect to camera.

At the time of annotating we were not aware of it, but there exists an external tool called vatic-tracker [28] that can be integrated with VATIC. It replaces the linear interpolation with a object tracking algorithm using OpenCV and should help with increasing the annotation speed noticeably when dealing with non-linear movements.

As VATIC had graphical user interface (GUI) only for annotation, other tasks had to be done using command-line interface. To add new video frame into database we used `turkic load <identifier> </path/to/frames/directory> -offline -blow-radius 0`. Option `-offline` was used to mark the annotation job as offline job instead of using Mechanical Turk. Option `-blow-radius 0` was used to tell VATIC not to discard bounding-box values on neightbouring frames when the object is annotated on a new frame. To publish frames to webserver for annotaion we used `turkic publish -offline` and to export annotations into text file we used `turkic dump <identifier> -o <output-file-name>`.

In total, we annotated 6 different scenes, 4 from the session in January and 2 from the session in April (due to the bad quality of the videos recorded with Bumblebee2 camera, we were not able to use any recording from the session in November). Each scene contained 280-321 frames with an average of 2.4 objects on a frame.

3.3 Running detectors on acquired data

In order to perform the analysis, we stored the output of all object detectors on all 6 annotated recordings. We used Autoware nodes to run the object detectors, `rosplay` to play the recorded rosbag and `rostopic echo` to save messages published by object detector into yaml file. Playing the rosbags in real-time resulted in more compute-intensive detectors to struggle with proccessing all the frames on the recording. As our goal was to isolate the performance factor and analyse that separately, we slowed down the playback of the rosbags to get output for each frame from all detectors.

4. Analysis of the results

4.1 Object-based statistics

To get an overview of how many objects were detected and missed, we computed a so called object-based statistics, using IoU as a measure for accepting predicted object as true positive (TP). This is basically implementation of a "localized object count recall" introduced in [21]. On each frame we counted TPs, FNs and FPs using following rules:

- all ground truth (GT) objects which have any detected object with IoU above threshold count as TP
- all GT objects without any detected object with IoU above threshold count as FN
- all detected objects without any GT object with IoU above threshold count as FP
- all detected objects that had a GT object with IoU above threshold, but were not counted as TP due to some other detected object having higher IoU (multiple detection of same object), were disregarded.

In this metric the concept of true negative (TN) is not used as the amount of possible TN could be very high and have no statistical significance.

As in the context of our detection task multiple detections of the same GT object doesn't possess any threats, disregarding them just reduced the noise. Counting them as TPs would be problematic as sum of TPs and FNs wouldn't match the GT object count any more. Counting them as FPs would not give us any valuable information, but rather make it more difficult to detect more important FPs.

Finally we summed up all TPs, FNs and FPs to get the statistics for the whole scene. All the figures representing object-based scores show the total values of each group for each detector for the whole scene. To make thing clear, when we talk about recall in the following chapters, while not explicitly stated on the charts, we mean the propostion of TPs to GTs (TPs + FNs).

4.2 Pixel-based statistics

To get an overview of relative area covered by hit and error we also computed a so called pixel-based statistics by marking every pixel on each frame as TP, FN, FP or TN following these rules:

- pixels that are not inside any object, GT or detected, are marked as TN;
- pixels that are inside at least one GT object and one detected object are marked as TP;
- pixels that are inside at least one GT object but not inside any detected object are marked as FN;
- pixels that are not inside any GT objects but are inside at least one detected object are marked as FP.

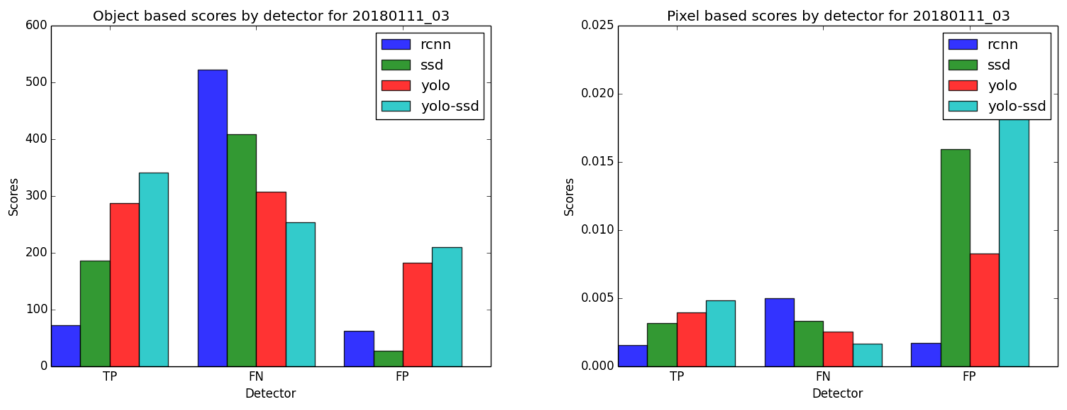
Compared to just adding up errors on each object, this removes the effect of "double counting" caused by overlapping areas and every pixel is only counted once. Then we counted them on each frame and divided by total number of pixels (in out case 1280x1024 = 1310720) to get relative area of the frame covered by each group. Finally we averaged the relative areas of each group to get the statistics for the whole scene.

This metric is somewhat similar to "area-based recall for frame" and "area-based precision for frame" introduced in [21], but we use it together with the object base metric to understand the average sizes of detected and missed objects.

4.3 Combining detectors

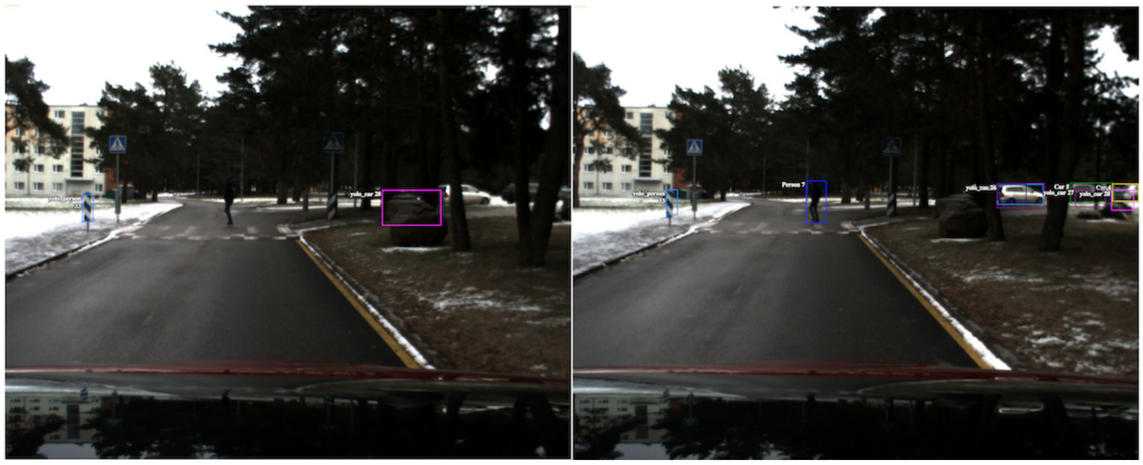
In addition to comparing the performance of different detectors, we wanted to see if using multiple detectors simultaneously could improve the results. Therefor we merged to outputs of YOLOv2 and SSD and performed the object-based analysis on that. Simply merging the outputs of detectors meant that we should see an increase in positives, both TP and FP, but as we disregard all double detections with IoU over threshold, the increase should be more noticeable in FPs and TPs should still represent portion of GT objects.

4.4 Performance



(a) Object based TP, FN and FP counts (b) Share of TP, FN and FP pixels (TN left out)

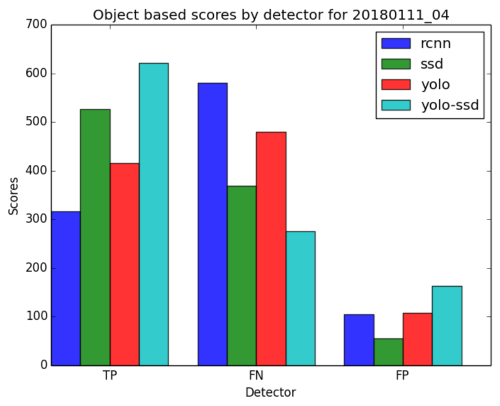
Figure 4.1: Object and pixel based results of scene 20180111_03



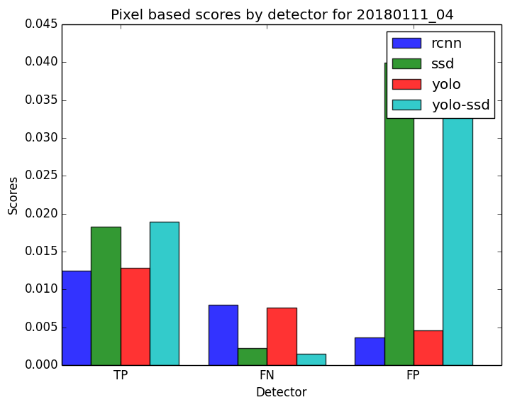
(a) Examples of YOLO FPs (b) SSD missing a pedestrian on crossing

Figure 4.2: Frames from scene 20180111_03

First scene was recorded in January with little snow and cloudy weather, containing total of 3 different pedestrian and 14 different cars. From Figure 4.1 we can see that none of the detectors performed well. Although there is significant difference between TP and FN counts of detectors, only combined YOLOv2 and SSD was able to achieve more FNs than TPs. Pixel based scores show that while the number of FNs was higher than number of TP, area covered by them was not so different, which indicates that FNs were smaller than TPs. From single trackers YOLOv2 managed to show the best recall, but also had much higher amount of FPs with 2 examples visible on Figure 4.2(a). Probably the most important observation is that SSD was not able to detect the pedestrian crossing the road Figure 4.2(b) until he was already half way across. Another thing to notice is that while SSD had relatively low amount of FPs, area covered by them was very large.

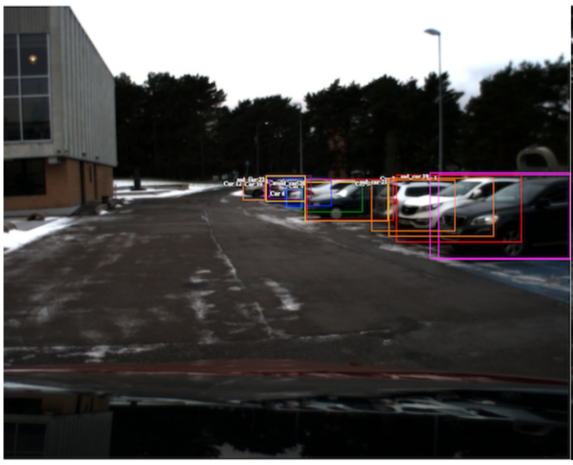


(a) Object based TP, FN and FP counts

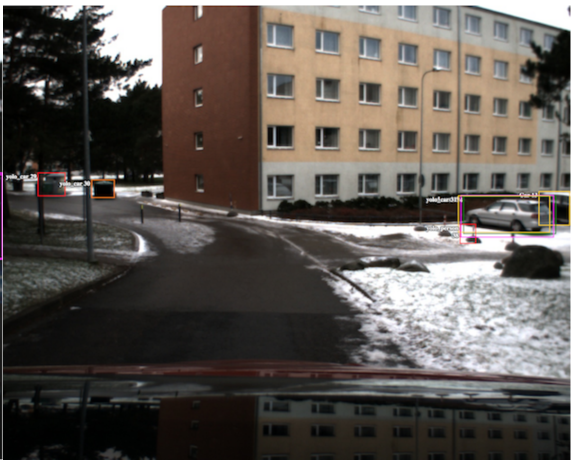


(b) Share of TP, FN and FP pixels (TN left out)

Figure 4.3: Object and pixel based results of scene 20180111_04



(a) Parking lot with lots of cars



(b) Examples of YOLO FPs

Figure 4.4: Frames from scene 20180111_03

Second scene was recorded on the same day with similar conditions but different location and setting. Scene started in parking lot with lots of cars (Figure 4.4(a)) and contained total of 17 different cars and only 1 pedestrian. In general, object analysis show slightly better results (Figure 4.3) but none of the detectors was performing remarkably

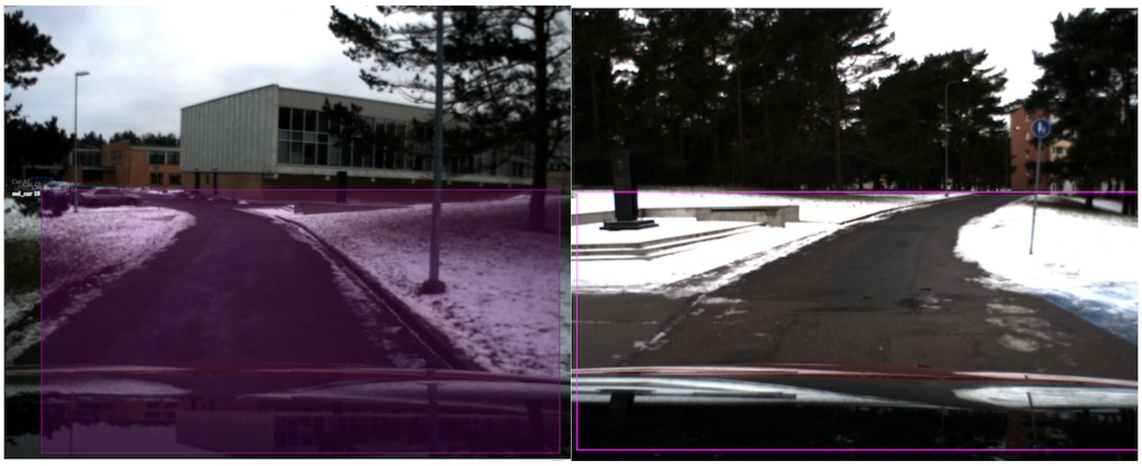
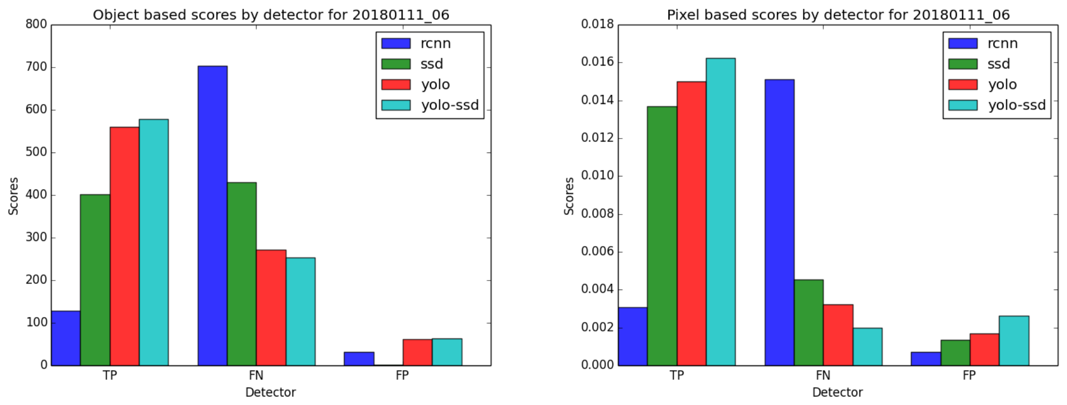

(a) Frame from scene 20180111_03 (b) Frame from scene 20180111_04

Figure 4.5: Large FP caused by the visibility of the front of our test car

better. Different from first scene is that here SSD managed to achieve best recall, with amount of FNs still very high, but the small area of FNs on Figure 4.3(b) indicates that average size of the missed object was relatively small and thus the objects were probably far away. Figure 4.4(b) shows again examples of FPs triggered by YOLOv2 detecting trash bins as cars, causing much higher amount of FPs than SSD. Similarly to the previous scene, we can see that SSD had lowest number of FPs, but average area was very large. When visualizing the outputs, we discovered that it in both scenes it was caused by the front of the car being visible on the frames and SSD detecting it as a car (Figure 4.5). We could crop the bottom part of the frames before passing them to detector to avoid these FNs, but we can just ignore them, because camera placement on the final vehicle eliminates this problem.



(a) Object based TP, FN and FP counts (b) Share of TP, FN and FP pixels (TN left out)

Figure 4.6: Object and pixel based results of scene 20180111_06

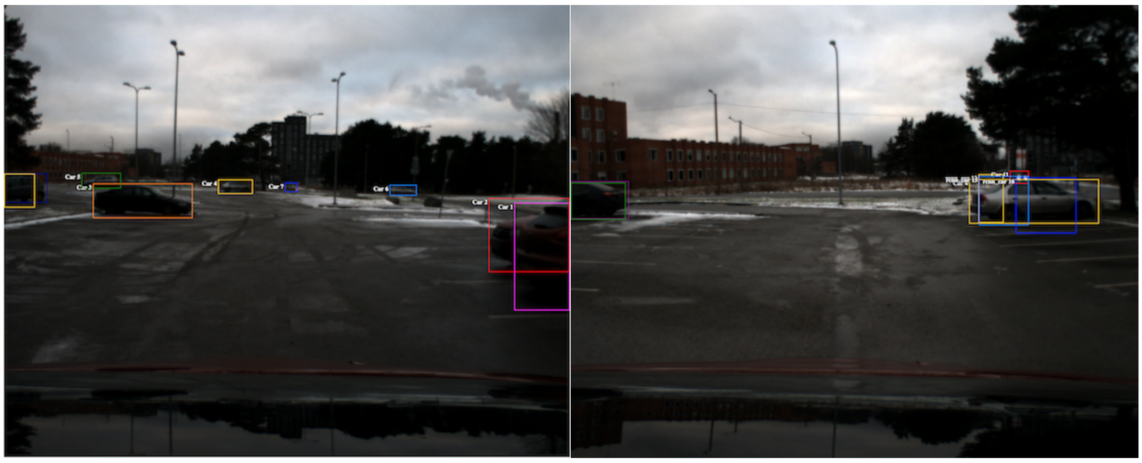
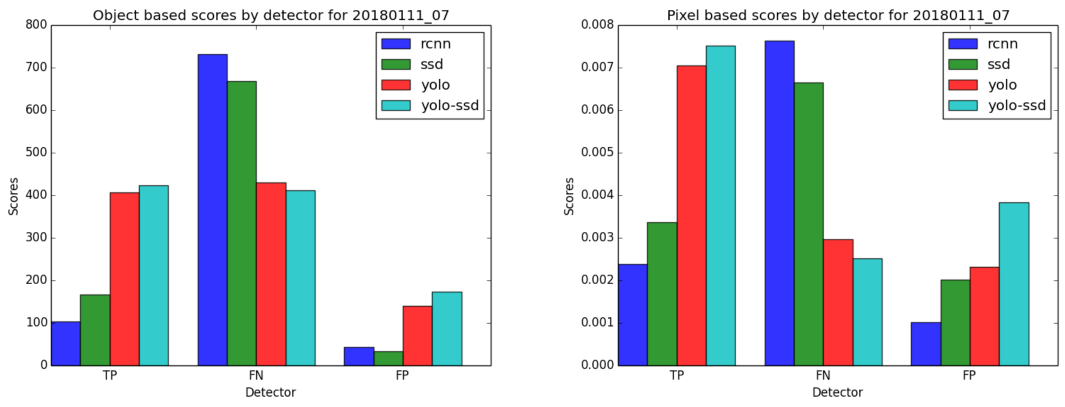

(a) Examples of Fast-RCNN FPs (b) Examples of Fast-RCNN FPs

Figure 4.7: Frames from scene 20180111_06

Third scene again was recorded on the same day as first and second, but the setup was a parking lot with cars parked sparsely, no pedestrian and few cars passing by on the road behind the parking lot. From Figure 4.6 we can see that for some reason Fast-RCNN was performing very poorly as seen on Figure 4.7(a), failing to detect all cars except one on the left edge of the screen, and Figure 4.7(b), detecting one car as three smaller ones and causing 3 FNs instead of 1 TP. YOLOv2 managed to show again the best results and also noticeable is that here combined detector had very little edge over YOLO in recall. In general results were similar to previous scenes (with an exception of Fast-RCNN), giving approximately same amount of TPs and FNs, but Figure 4.6(b) indicates that FNs were on an average 3-4 times smaller.



(a) Object based TP, FN and FP counts (b) Share of TP, FN and FP pixels (TN left out)

Figure 4.8: Object and pixel based results of scene 20180111_07

Fourth scene was selected as a test how consistently detectors perform. It was

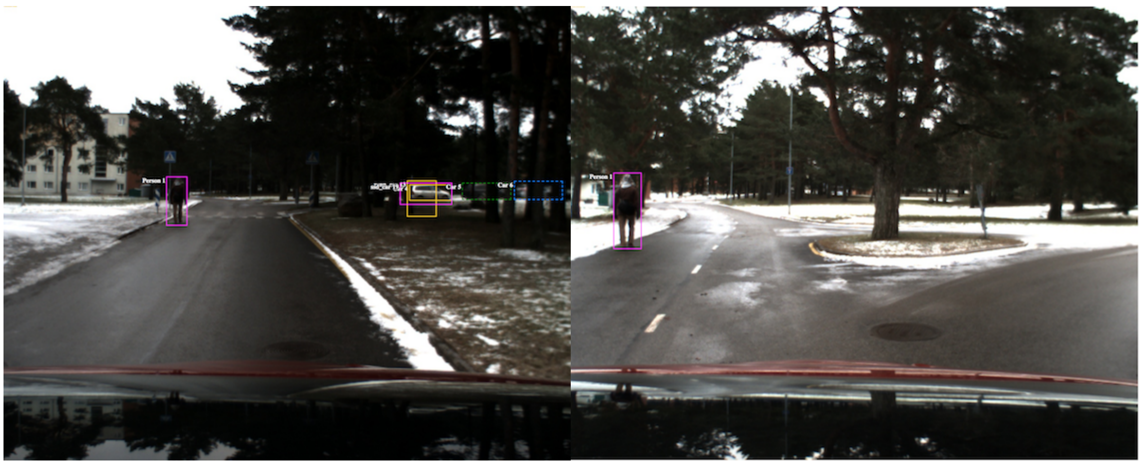
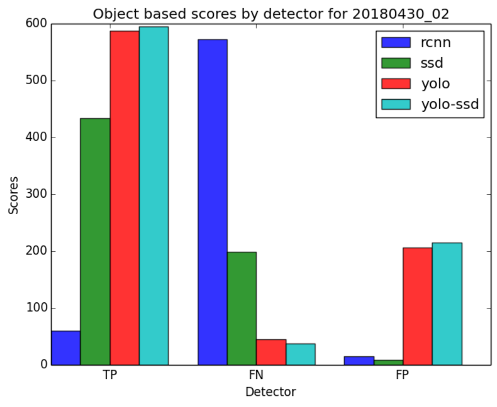
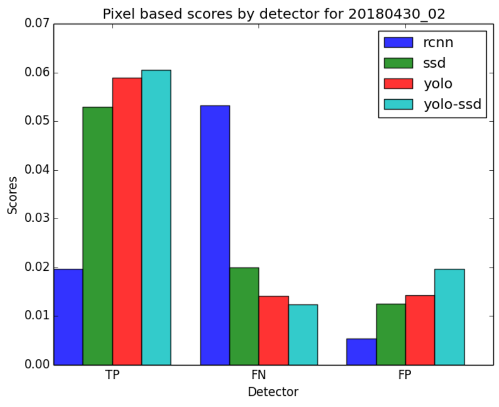

(a) (b)

Figure 4.9: Frames from scene 20180111_07. Examples of Fast-RCNN and SSD failing to detect a pedestrian

recorded in same conditions, mostly in same location and with similar setup as the first scene, having total of 9 cars and 3 pedestrians. In general, as seen on Figure 4.8, results are quite similar to the first scene having more FNs than TPs, except here YOLOv2 showed even more significant advantage over others and and combining it with SSD didn't have much effect. Figure 4.8(b) shows that Fast-RCNN and SSD were missing large GT objects with examples of missed pedestrian on Figure 4.9.



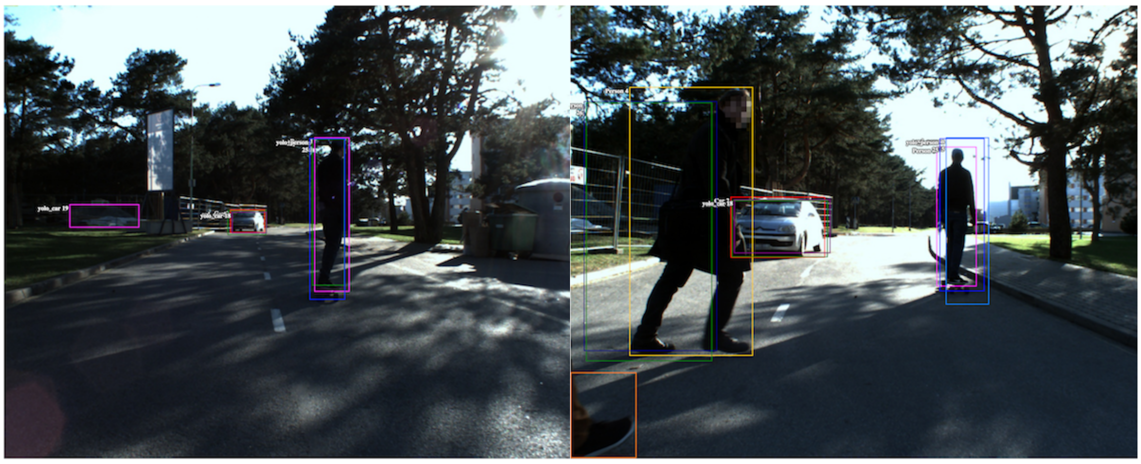
(a) Object based TP, FN and FP counts



(b) Share of TP, FN and FP pixels (TN left out)

Figure 4.10: Object and pixel based results of scene 20180430_02

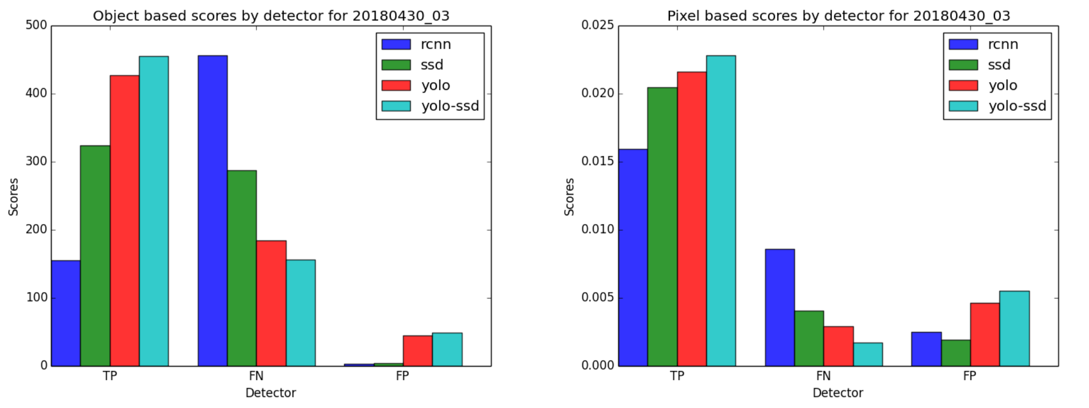
Fifth scene was recorded in April with dry and sunny weather. Setup was easier than in previous scenes, containing only total of 3 persons and 1 car, but frontal sun and high contrasts increased difficulty of the task. With an exception of Fast-RCNN, results (Figure 4.10) were pretty good. While YOLO managed to show strong recall, it came



(a) Example of YOLO FP (b) SSD and YOLO fail to accurately locate

Figure 4.11: Frames from scene 20180430_02

at an expense of high FPs count, mainly caused by a detection of same non-existent car (Figure 4.11(a)) on more than half of the frames. In contrast, SSD had just few FPs, but also noticeably lower recall. Another example of this scene on Figure 4.11(b) shows how SSD (green box) and YOLO (blue box) both failed with accurately locating the pedestrian (yellow box) very close to the camera.



(a) Object based TP, FN and FP counts (b) Share of TP, FN and FP pixels (TN left out)

Figure 4.12: Object and pixel based results of scene 20180430_03

Sixth scene was recorded on the same day and with similar weather and light conditions as fifth scene. Setup was more difficult with total of 4 pedestrians and 3 cars, but this time with rear sun. Figure 4.12 shows that again there is significant difference between object-based values of detectors, amount of FNs was still quite high ranging from 0.7 for YOLO to 0.25 for Fast-RCNN. Pixel-based values on Figure 4.12(b) show that FNs were on average smaller than TPs. One thing to notice on this scene is that rear sun dropped the

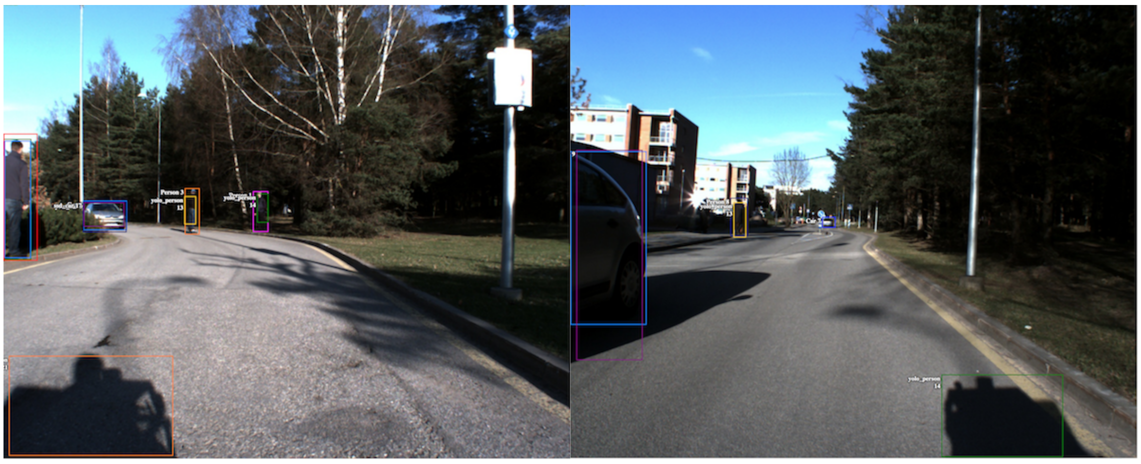

(a) SSD FP caused by a shadow (b) YOLO FP caused by a shadow

Figure 4.13: Frames from scene 20180430_03

shadow of ourself in front of us, causing both SSD (Figure 4.13(a)) and YOLO (Figure 4.13(b)) to mistakenly predict it to be a person.

4.5 Tuning of the metric

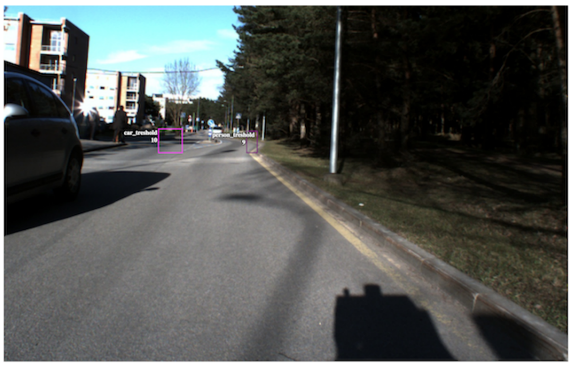
Observing statistics of all the scenes together, we noticed that an average size of FN detections was, with few exceptions for Fast-RCNN and SSD, smaller than average size of TP detections. As object further from the camera appear smaller on the video, there was a reason to believe that many on the FNs are far from the camera. This brought us to a hypothesis, that considerable amount of FNs were small object that are far from us and therefor we don't care if we fail to detect them.

With a help of dynamics and lens optics, we were able to test our hypothesis. Knowing maximum speed, reaction time and braking deceleration, we can calculate maximum stopping distance for the car and consider all the object farther away as "don't care". We know that the maximum speed for the vehicle will be 20km/h, but as the vehicle is still being built, we don't yet know the breaking deceleration for it. Therefor we used an average passenger car breaking deceleration of 5.4m/s2. Reaction time in out case is basically the time it will take for a detector node from taking an input until publishing output. Due to detectors having different speeds, we decided to use average human reaction time of 0.7sec which should be safe for even the slowest Fast-RCNN. This gave us 6.8m for stopping distance for the vehicle moving at its maximum speed of 20km/h. So, with a little buffer, everything that is farther than 10m can be considered as "don't care".

To figure out how far objects are by looking at the video frame, we need to know details about the camera and the actual object. More precisely, if we know:

- lens focal length;
- sensor size in mm;
- sensor size in pixels;
- size of the object on image (in pixels, width or height);
- approximate width or height of the object in real life in meters;

, we can calculate approximate distance between the camera and the object in real life. Estimating approximate minimum size of a car and a pedestrian and using above-mentioned principles, we computed the threshold size for both and assumed that all undetected objects below the threshold size are at least certain distance away from the camera. Both objects with respective threshold size and frequent aspect ratio are visualized on Figure 4.14. As distant "don't care" objects are not safety-critical in our context, we trimmed the FN count by removing all instances where area of the GT object is below the threshold.



(a)

Figure 4.14: Threshold sizes for "don't care" car and pedestrian objects

Results (Figure 4.15) show that our hypothesis was correct and with eliminating small undetected GT objects we were able to reduce FN count remarkably. While the percentage of decrease was different on scenes, results show that all detectors were struggling with detecting small objects. With an exception in scene 20180430_02, number of eliminated FNs was almost equal for all detectors. On an average, we were able to remove around 50% of false-negatives.

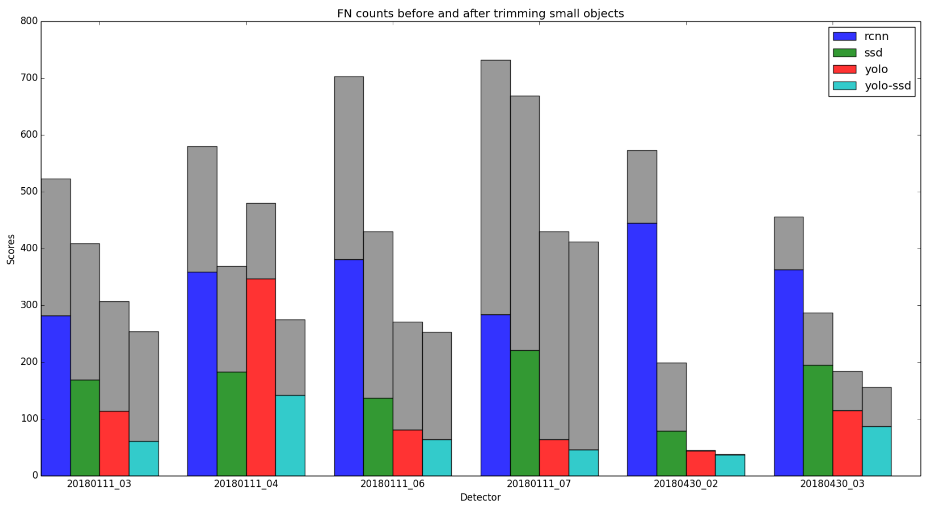


Figure 4.15: Object based statistics with small FNs trimmed off

In a safe-critical application like self-driving car, we need to be extremely cautious with false-negatives. Despite the considerable decrease, the amount of remained false-negatives is still substantial. Object tracking can help with filling gaps in sequential frames, but it still depends on detecting the object first. Considering these results, further steps need to be done to improve the recall of the detector.

4.6 Speed and efficiency

In real-time applications like self-driving car almost as important as performance of the detectors is speed and efficiency. Ability to process frames faster translates into faster reaction times. As longer reaction time prolongs the stopping distance, it is safety critical to be able to process the frame within a certain time-frame, whereas length of the time-frame depends on the speed of the car.

To understand GPU load of each detector, we used `rosplay` to play recorded videos in real-time (1280x1024 @ 7 FPS) and Autoware nodes to run the object detector. During the playback we used NVidia System Management Interface (nvidia-smi) to log out the GPU load. To examine how well each object detector can process real-time input, we counted how many frames each detector was able to process. We ran this experiment on 2 different GPUs: NVidia Tesla K40m (Table 4.1) and NVidia Geforce GTX 1080 Ti (Table 4.2).

Table 4.1: GPU load and processed frame count on NVidia Tesla K40m

Object detector	GPU load	processed/total frames
Fast-RCNN	100%	240/961
SSD	70-90%	961/961
YOLOv2	30-50%	960/961

Table 4.2: GPU load and processed frame count on NVidia Geforce GTX 1080 Ti

Object detector	GPU load	processed/total frames
Fast-RCNN	80-100%	734/961
SSD	18-36%	960/961
YOLOv2	10-24%	960/961

From the results we can see that there is noticeable difference of GPU load between object detectors on both GPUs. Fast-RCNN was utilizing the whole GPU on both experiments and still wasn't able to process all frames. Managing to process only 76% of the frames on the more powerful Geforce GTX 1080 Ti clearly indicated that Fast-RCNN might not be the best option for a real-time application like self-driving car. While SSD and YOLOv2 both managed to process all[1] frames on both GPUs, there was considerable difference between the load even on the Geforce GTX 1080 Ti, making YOLOv2 the best choice for speed and efficiency.

[1] 1 frame deficit was caused by the initialization proccess and was not considered important in the statistics.

5. Conclusion

With emergence of CNNs the field of computer vision has advanced vastly resulting in an abundance of different methods for detecting object on a video. Quality of a method is usually measured using one of the benchmark datasets which evaluate how well the method can perform with large number of different object classes and contexts. In case of a specific context, a state-of-the-art method that is trained and evaluated on a context-free dataset might not be the top performer. If you have enough context specific data you can train you own object detector, but annotating ground truth data for object detection is relatively expensive task. Therefor using a pre-trained model is often the only option, but there was no developed method that helps to decide which of the available methods is best considering specific context and application.

To understand how reliably the available methods can detect cars and pedestrians, we developed a method for analyzing and evaluating performance of different object detectors in autonomous self-driving car context. The general idea of our method is using our own benchmark dataset containing relevant data and relevant, context specific metrics to measure the performance of different methods. Using this method, we first recorded scenes on the planned route in different weather and light conditions and selected 6 scenes that give us a good sample set. Next we created ground truth data by annotating selected scenes, ran each object detector on each annotated scene and gathered performance statistics by comparing the output to the ground truth using context specific metrics. Finally we compared the statistics of different object detectors and used annotation tool to visualize prediction of object detectors together with ground truth data to get better understanding of the statistics.

While out of the three available detectors YOLOv2 [6] showed best performance and efficiency, results clearly indicate that none of the analysed object detectors is able to detect objects with reliability required for safety-critical applications like self-driving cars. To guarantee safety, we can not rely only on the object detector, but need to use it in combination with other sensors, that can fill in the weak spots.

There are several ideas how to further improve the performance. As we only performed the analysis with one camera, using different cameras and camera configurations could result in better performance and should be studied. Fine tuning a pre-trained detector is widely used method to adapt it to certain context, but requires more training data. Futhermore, there could be new and better methods that were not studied in this thesis, but this requires developing integration with Autoware. In fact integration with new version of YOLO, YOLOv3, is in development and could potentially perform better than studied detectors.

References

[1] Autoware TierIV Academy. Autoware hands-on exercises, May 2018. URL `https://github.com/CPFL/Autoware-Manuals/blob/master/en/Autoware_TierIV_Academy_v1.1.pdf`.

[2] World Health Organization. *Global status report on road safety 2015*. World Health Organization, 2015.

[3] Pedro Felzenszwalb, David McAllester, and Deva Ramanan. A discriminatively trained, multiscale, deformable part model. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[4] Ross Girshick. Fast r-cnn. *arXiv preprint arXiv:1504.08083*, 2015.

[5] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[6] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. *arXiv preprint*, 2017.

[7] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[8] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[9] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Chal-

lenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

[10] Shinpei Kato, Eijiro Takeuchi, Yoshio Ishiguro, Yoshiki Ninomiya, Kazuya Takeda, and Tsuyoshi Hamada. An open approach to autonomous vehicles. *IEEE Micro*, 35 (6):60–68, 2015.

[11] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[12] Ros wiki, May 2018. URL `http://wiki.ros.org/ROS/Introduction`.

[13] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[14] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

[15] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *european conference on computer vision*, pages 346–361. Springer, 2014.

[17] Jasper RR Uijlings, Koen EA Van De Sande, Theo Gevers, and Arnold WM Smeulders. Selective search for object recognition. *International journal of computer vision*, 104(2):154–171, 2013.

[18] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014.

[19] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[20] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[21] Vladimir Y Mariano, Junghye Min, Jin-Hyeong Park, Rangachar Kasturi, David Mihalcik, Huiping Li, David Doermann, and Thomas Drayer. Performance evaluation of object detection algorithms. In *Pattern Recognition, 2002. Proceedings. 16th International Conference on*, volume 3, pages 965–969. IEEE, 2002.

[22] Mu Zhu. Recall, precision and average precision. *Department of Statistics and Actuarial Science, University of Waterloo, Waterloo*, 2:30, 2004.

[23] Hinrich Schütze, Christopher D Manning, and Prabhakar Raghavan. *Introduction to information retrieval*, volume 39. Cambridge University Press, 2008.

[24] Language and University of Maryland Media Processing Laboratory. A video metadata markup tool: Viper-gt, May 2018. URL `http://viper-toolkit.sourceforge.net/products/gt/`.

[25] Anting Shen. Beaverdam: Video annotation tool for computer vision training labels. Master's thesis, EECS Department, University of California, Berkeley, Dec 2016. URL `http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-193.html`.

[26] NPSVisionLab. Vatic in a box. wrapped up in a docker container., May 2018. URL `https://github.com/NPSVisionLab/vatic-docker`.

[27] Carl Vondrick, Donald Patterson, and Deva Ramanan. Efficiently scaling up crowdsourced video annotation. *International Journal of Computer Vision*, pages 1–21, 2012. ISSN 0920-5691. URL `http://dx.doi.org/10.1007/s11263-012-0564-1`. 10.1007/s11263-012-0564-1.

[28] John Doherty. Vatic tracking module, May 2018. URL `https://github.com/johndoherty/vatic_tracking`.