TALLINN UNIVERSITY OF TECHNOLOGY

School of Information Technology

Department of Software Science

Hannes Liik 142853IAPB

Vehicle detection methods and their comparison in the context of Tallinn's traffic surveillance videos

Bachelor's thesis

Supervisor: Eduard Petlenkov

PhD

Tallinn 2017

TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia teaduskond

Tarkvarateaduse instituut

Hannes Liik 142853IAPB

Sõidukite tuvastuse meetodid ja nende võrdlus Tallinna liikluskaamerate näitel

Bakalaureusetöö

Juhendaja: Eduard Petlenkov

PhD

Tallinn 2017

Author's declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Hannes Liik
22.05.2017

Abstract

This thesis examines different vehicle detection methods, with the aim of automating data collection from traffic surveillance videos. First, a survey of vehicle detection methods is given, followed by a comparison of methods applied to Tallinn's traffic surveillance videos. In order to train some of the used models, a new data set is also constructed.

To conclude the thesis, a suggestion of methods to use is made. Firstly, in case a graphics card is not available, OpenCV's Haar cascades model proved to work well. If, a capable graphics card is a available, it is recommended to use Single Shot Multibox Detector, or a similar model, as it was the most accurate tested method and is able to extract the most information from images.

This thesis is written in English and is 37 pages long, including 6 chapters and 17 figures.

Annotatsioon

Sõidukite tuvastuse meetodid ja nende võrdlus Tallinna liikluskaamerate näitel

Selles töös uuriti erinevaid sõidukite tuvastamise meetodeid, eesmärgiga automatiseerida andmete kogumist liikluse kohta. Esiteks antakse ülevaade olemasolevatest meetoditest. Seejärelt katsetatakse erinevaid meetodeid Tallinna liikluskaamerate videote peal ning antakse hinnang, kui hea iga meetod oli ning mis on selle eelised ja puudused. Kasutatavate mudelite arendamiseks moodustati ka uus andmestik. Lisaks katsetati uut alternatiivi nendele meetoditele.

Jõutakse järeldusele, et kui pole võimalik kasutada kaasaegset graafikakaarti, on hea valik kasutada *OpenCV Haar cascade* mudelit. Vastasel juhul soovitatakse kasutada *Single Shot Multibox Detector*it, või sarnast mudelit, kuna see osutus kõige täpsemaks ning võimaldab piltidelt eraldada kõige rohkem informatsiooni.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 37 leheküljel, 6 peatükki ja 17 joonist.

List of abbreviations and terms

CNN Convolutional Neural Network. 14, 34

CPU Central Processing Unit. 17

CUDA Compute Unified Device Architecture. 17

FCN Fully Convolutional Network. 27

GPU Graphics Processing Unit. 15

HOG Histogram of Oriented Gradients. 14, 15

NMS Non-Maximum Suppression. 27, 31

OpenCV Open Source Computer Vision. 4, 14, 17, 24, 34

SSD Single Shot Multibox Detector. 4, 5, 27–30

Table of contents

List of figures

1 Introduction

Traffic surveillance cameras have become commonplace, allowing the monitoring of congestion and traffic accidents. They also open up the possibility of gathering data about traffic patterns, vehicle usage, and the like, but it is not feasible for humans to manually write down everything that happens in these videos. Automation is therefore needed, but complex real life sensory data, such as video, is difficult to extract information from. Thus, much research has been done in this field and many methods have been proposed for detecting vehicles (and any other objects) in images [1, 2, 3, 4]. With so many options, it is hard to choose which method to use, and many of them are not usable out of the box, but require training of a model before they can be tested.

In this thesis, I investigate some of these methods and provide a comparison by applying them to Tallinn's traffic surveillance videos. I will also implement and test my own vehicle detection model. Finally, I will give recommendations on what methods to use based on the need to: process many video feeds in real time; extract as much information as possible; give accurate information.

2 Introduction to vehicle detection

Before giving a survey of some vehicle detection methods in section 3, I will introduce the idea of detection as it is done in the Overfeat [5] paper (which proposes a method for object detection, a superset of vehicle detection), by dividing it into two subtasks.

The simplest task is classification: classifying the contents of an image into one of many predetermined classes. While Overfeat has to consider 1000 classes, we are really interested in two: ”vehicle” and ”background” (not a vehicle). Of course, we could get some more value out of our model, by adding some classes, for example:”car”, ”truck”, ”bus”, ”pedestrian”, ”pedestrian crossing the street” etc, at the expense of model complexity.

The second, intermediary step is localization, in which the position and size of the object is predicted in addition to its class. Finally, the process of localizing several objects is called detection. I consider a single detection as the class and the bounding box of an object. While we may only be interested in the number of vehicles in an image, having the locations and sizes (bounding boxes) helps us determine if the method we use works correctly, or gives us accurate numbers by coincidence.

The easiest way to do detection is to revert to the classification task by looking at small regions of the input image separately and assume that there is either one object in the region or none. The method of going through subsections of an image is called the sliding window (Figure 1). The prediction of the bounding box of the object is the same as the window position and size. Of course, objects are unlikely to exactly fit the window, so several passes are usually done over the input image with different window sizes. Alternatively, the window size is left the same, but the input image is re-sized, making a ”pyramid” of images.

Such sliding window classification has been the standard for a long time for what I refer to as ”traditional” detection methods (anything that doesn’t use deep learning), which I cover in section 3.1. Newer artificial neural network models often look at the whole image once to produce detections [6, 7]. I will cover such ”deep learning” models in section 3.2.

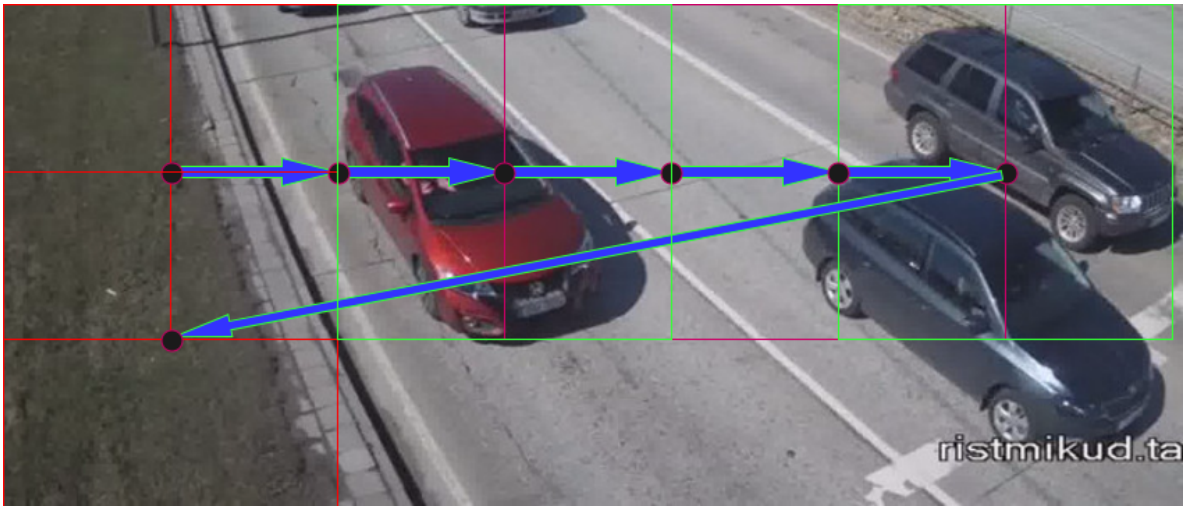


Figure 1: Sliding window classification. The image is examined a small region - a window - at a time. The window is moved by a small amount, called stride (in this case, the stride is 50% of the window size). The contents of the window is classified as a ”vehicle” (green outline) or ”background” (red outline). The position of the window in combination with the classification may be counted as a detection.

3 Survey of methods for vehicle detection

In this section, i will give an overview of existing vehicle detection methods. I will start with ”traditional” methods, machine learning methods with features engineered by researchers, in section 3.1. Then i will cover new ”deep learning” based methods in section 3.2

3.1 Methods of vehicle detection using traditional methods

3.1.1 Background subtraction

One of the easiest ways of detecting moving objects is adaptive background subtraction, in which an estimate the background of the scene is made over time. Any pixels that differ from the estimated background by a set amount are considered as foreground or a region of interest. When there is a blob of such pixels in an surveillance video frame, there is likely a vehicle there. There are two significant problems with this technique. Firstly, when the background of the image changes quickly, such as a cloud passing by, making the entire image dimmer, the algorithm will not be able to distinguish the foreground and background. Secondly, the method picks up any moving objects, including ones we are not interested in, such as pedestrians. The foreground mask and background are calculated according to equations 1 and 2.

$$M_t = (I_t - B_t) > \lambda \tag{1}$$

$$B_{t+1} = \alpha I_t + (1-\alpha) B_t \tag{2}$$

Where t = timestep, B = background, α = decay, I = image, M = foreground mask [8].

3.1.2 Colormap classification

In [1], color is used to hypothesize vehicle locations. They do this by mapping colors from the RGB colorspace to a u, v-colorspace given in equation 3, in which vehicle and non-vehicle

colors are better separable.

$$u_p = \frac{2Z_p - G_p - B_p}{Z_p} \qquad \text{and} \qquad v_p = Max\left\{\frac{Z_p - G_p}{Z_p}, \frac{Z_p - B_p}{Z_p}\right\} \tag{3}$$

Where (R_p, G_p, B_p) is the color of pixel p and $Z_p = (R_p + G_p + B_p)/3$. It is presumed that the colors for vehicle and non-vehicle classes follow Gaussian distributions, and therefore we can calculate the probability of each pixel belonging to which class. The likelihood of pixel x given class "vehicle" is given by 4.

$$p(x|vehicle) = \frac{1}{2\pi\sqrt{|\Sigma_v|}} exp((1/2)(x - m_v)\Sigma_v^{-1}(x - m_v)^T) \tag{4}$$

Where

$$\Sigma_v = \text{covariance matrix of transformed vehicle pixels}$$

$$m_v = \text{mean value of transformed vehicle pixels}$$

The mean values and covariance matrices are calculated from a data set. The computation for the probabilities of non-vehicle pixels is analogous. We classify a pixel x to the class "vehicle" according to equation 5.

$$p(x|vehicle)P(vehicle) > p(x|non - vehicle)P(non - vehicle) \tag{5}$$

Where P(vehicle) and P(non-vehicle) are priori class probabilities of vehicle and non-vehicle pixels.

Because the color distributions have overlap, this classifier alone is not accurate enough. Thus, regions of the image, which contain a high density of pixels labeled as "vehicle", are further verified using edge maps, coefficients of wavelet transforms and corners [1].

3.1.3 Haar-like features and Histograms of Oriented Gradients

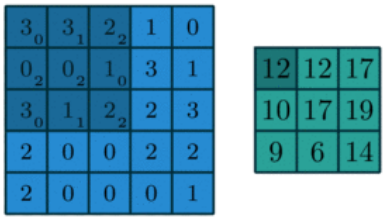
Rectangular filters or Haar-like features and Histograms of Oriented Gradients (HOGs) are frequently used as features for classifiers, as they capture spatial information. In [2], they are used by weak classifiers[1]. The outputs of these weak classifiers are used as features for another, "strong" classifier. This method is called boosting and in [2], the AdaBoost [9] algorithm is used. OpenCV [10] also has functions to train and run Haar-based detection models.

In a recent work, [3] uses context as a feature. In the proposed method, images are divided into cells, each having a number of features describing its contents (notably a 6-channel HOG), as well as features describing neighboring cells. The method is also an example of heavy feature re-use, as the classifier is run in a sliding window on the once calculated features, not the image, making a "feature pyramid" (otherwise, overlapping regions of windows would be calculated several times, redundantly). Furthermore, the features don't have to be recalculated for each scale, as intermediary scales are approximated from neighboring scales. The paper [3] also has a comparison of results with other methods including [1] and [2] in table V, displaying significant improvements.

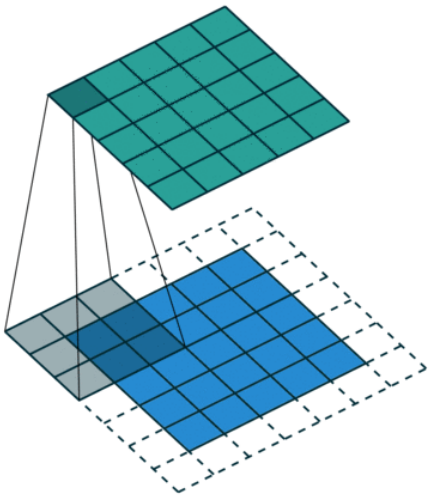
3.2 Deep learning models

Deep neural networks are currently state of the art in object detection [11, 12, 7]. This is mainly due to the advent of the Convolutional Neural Network (CNN) [13]. The convolution operation, depicted in Figure 2, is essentialy a sliding window itself, where at each window location, an output is calculated by the inner product of the window contents and a weight matrix or kernel. Note that the input usually a 3-d array and the kernel always has the same depth as the input, but in most visualizations, the input is shown as a 2-d matrix. A CNN consists of many layers of such transformations interlaced with nonlinearities, such as the rectifier $f(x) = max(0,x)$, with each layer able to detect patterns more abstract than the

[1] Weak classifiers only need to be a little better than random guesses, but their combined prediction, a "strong classifier" can be very accurate.



(a) Convolution takes the inner product of a weight matrix or kernel (small numbers in the dark blue matrix) and all possible sections of the input (blue matrix) to form a feature map (green matrix)



(b) If the size needs to be preserved, the input may be padded with zeros.

Figure 2: The convolution operation. Images adopted from [14].

last.

CNNs take the idea of adding context to features even further than [3], as each layer in the network increases the area the features describe (often referred to as the ”receptive field”). In traditional object detection methods, much of the improvements are made in feature engineering (such as the adoption of HOGs, Haar-like features and context-aware features). Neural networks are not limited by the features given by humans, as they learn to extract their own features. The main drawbacks of artificial neural networks have been 1) the amount of computational resources needed to run them, and 2) the amount of training data needed compared to traditional methods. The first problem is alleviated by efficient GPU implementations, and the second by large open data sets such as MSCOCO [15] and PASCAL [16].

In [4], a CNN is used to identify not only vehicles, but their make and model as well. There, the whole classification pipeline is run in a sliding window manner, making it simple, but with considerable overhead, as the overlapping regions of windows are evaluated several times. Because the network is small (3 conv. layers followed by 3 fully connected hidden layers), it is still fast enough to process several video streams in parallel and in real time. Overfeat [5] proposes a more efficient pipeline, in which convolutional layers are evaluated for the entire

image once, producing feature maps which are fed to the classifier with a sliding window. Furthermore, the fully-connected classifier layer is implemented by a 1x1 convolutional layer, which can be run efficiently on a GPU.

4 Comparison of vehicle detection methods in the context of Tallinn's traffic surveillance videos

In this section I will compare how some methods of vehicle detection work in practice. All the tests were on a laptop with Intel i5-4200H CPU, Nvidia 860m (2GB) GPU and 8GB RAM. Software used was: Ubuntu 16.04, python 3.5.2 (Anaconda custom), OpenCV 3.10 (Compiled with CUDA 7.5 [17]), PyTorch 0.1.11 and torch-vision 0.1.8 [18] using CUDA 8.0 and cuDNN 5.1 [19]. The used program code is available at `https://github.com/Leemur2/VehDetMethods/` [20].

4.1 Adaptive background subtraction

I have implemented adaptive background subtraction according to [8]. The algorithm keeps an estimate of the scene background according to equation 2. The foreground mask of an image I with background B is calculated with equation 1. An example of a background, an image and its mask is in Figure 3.

Detection can be done directly on the foreground masks by scanning through them with a sliding window and registering a detection when the sum of active pixels in the window is above some threshold. An example of this is given in Figure 4. We can see some problems with this approach, as multiple detections are made on the same object, and there are many non-vehicle detections. Possible fixes include merging of bounding boxes and then ignoring boxes with certain aspect ratios (tall thin objects are likely to be pedestrians, for example).
Advantages:

1. Fast.
2. Simple.
3. Does not require training.

Disadvantages:

1. Does not distinguish different objects.
2. Broken by fast weather changes.
3. Have to manually tune background decay and threshold.
4. Does not work on static images.
5. High false-positive rate.

In conclusion, background subtraction can be an easy and powerful tool for vehicle detection, but its high false positive rate makes it unusable as the sole detection algorithm. It might be more useful as a region of interest predictor.

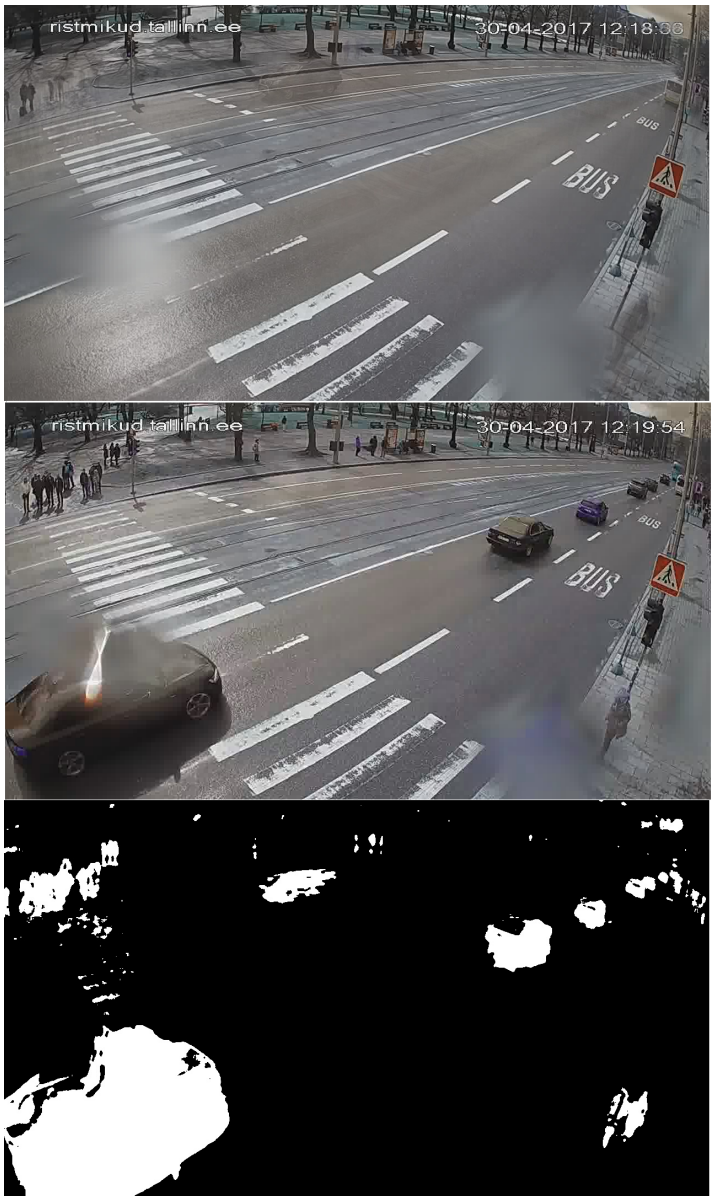


Figure 3: Background estimation, input image, foreground mask of the input.

Figure 4: Detections from a foreground mask.

4.2 Detection using colormaps

For testing the detection of vehicles from their colors as in [1], I created a data set [21] of vehicle and background pictures taken from live video feeds at `ristmikud.tallinn.ee`. Examples from the data set are in Figures 7 and 8. I re-sized all images to 64x64 resolution and transformed them to the u,v-colormaps according to equation 3, to calculate their distribution parameters:

$$m_v = \{-0.0149420, 0.035530\}$$
$$m_n = \{0.0083790, 0.0309860\}$$
$$C_v = \left\{ \begin{array}{cc} 0.0153850 & 0.0054790 \\ 0.0054790 & 0.0066460 \end{array} \right\}$$
$$C_n = \left\{ \begin{array}{cc} 0.0051850 & 0.0016780 \\ 0.0016780 & 0.002390 \end{array} \right\}$$

Where m_{class} = mean class pixel value and C_{class} = covariance matrix of class pixels. I had the best results with $P(vehicle) = P(non-vehicle) = 0.5$, but still they were not very good. Generally, non-road background was picked up, rather than vehicles, as seen in Figure 5. There were occasional good detections too, like in Figure 6. While in [1] additional verification was applied to detections, my results show that the these detections are not good enough to be useful, making verification redundant. It is possible, that the results would be better with a different data set for training. It may also be, that in the original paper, the vehicle colors were better distinguishable, as cars in Tallinn seem to be mostly dark and gray, much like the roads themselves, and even the sky.

Advantages:

1. Works in real time[1] with 0.075s processing time (without verification).
2. Simple compared to methods like neural networks.
3. Works on static images.

[1] At least 5 images should be processed in 1s.

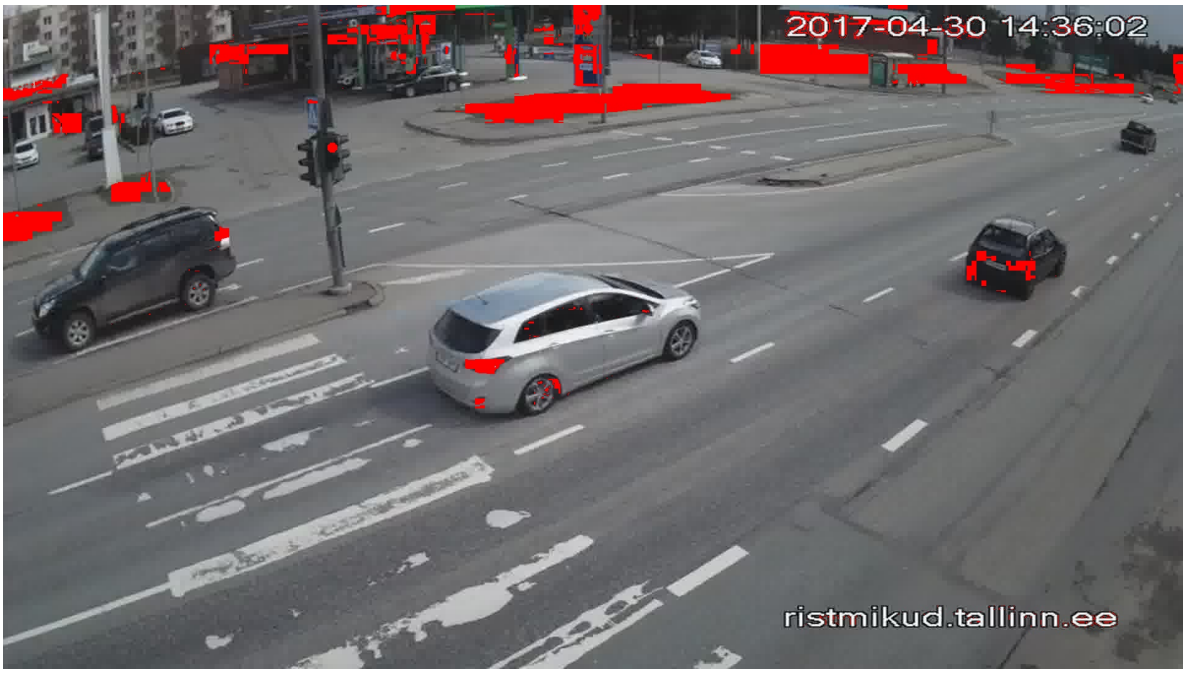


Figure 5: Colormap classification classifying background pixels as vehicle pixels (red).

Disadvantages:

1. Bad results in my tests.
2. Requires training data.
3. Requires an extra verification step.

In conclusion, this method did not work well in my experiments and I would not recommend it, but others have had decent results with it [1, 3].

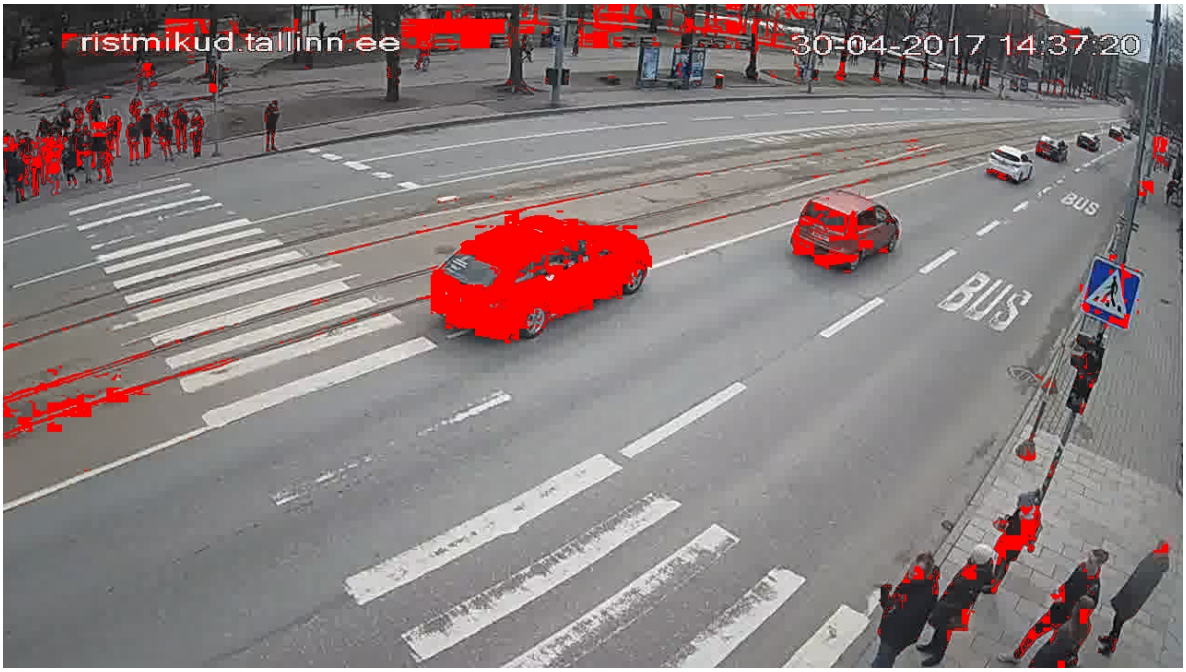


Figure 6: Vehicle pixels being recognized by colormap classification.

Figure 7: Examples of vehicles in my data set.

Figure 8: Examples of background images in my data set.

4.3 Haar cascades

I experimented with vehicle detection using OpenCV's Haar cascade detection [22]. While OpenCV has functionality for training such classifiers, I chose to test an existing model from [23]. Since the model was trained using pictures of cars only from the rear, it is expected that it may not work very well with vehicles in other orientations. Figure 9 demonstrates, that when traffic comes head-on from the camera's point of view, the detection algorithm works well. Furthermore, OpenCV's implementation does a good job pruning the detections so that each object gets counted only once. Sometimes, there are some weird false positives as well, like in Figure 11. Figure 10 shows that the model is indeed bad at detecting vehicles from the side, and produces many false positives. Training the model with a diverse data set would likely allow detection of vehicles in other orientations, but the increase in model complexity might lead to even more false detections. As the figures demonstrate, erroneous detections are often small in scale. Limiting the minimal scale size may avoid such errors.
Advantages:

1. Good implementation in OpenCV.
2. Requires less training data than neural networks.
3. Good test results with oncoming traffic.
4. Works in real time (10fps on 720p video)

Disadvantages:

1. Produced many false positives in tests.

In conclusion, using OpenCV's haar cascade implementation with a pre-trained model from [23] gave promising results, but did not work well with the variety of vehicle orientations seen in traffic surveillance videos, and gave a bit too many false positives. Training with another data set and tuning the maximum and minimum detection scales might fix these shortcomings.

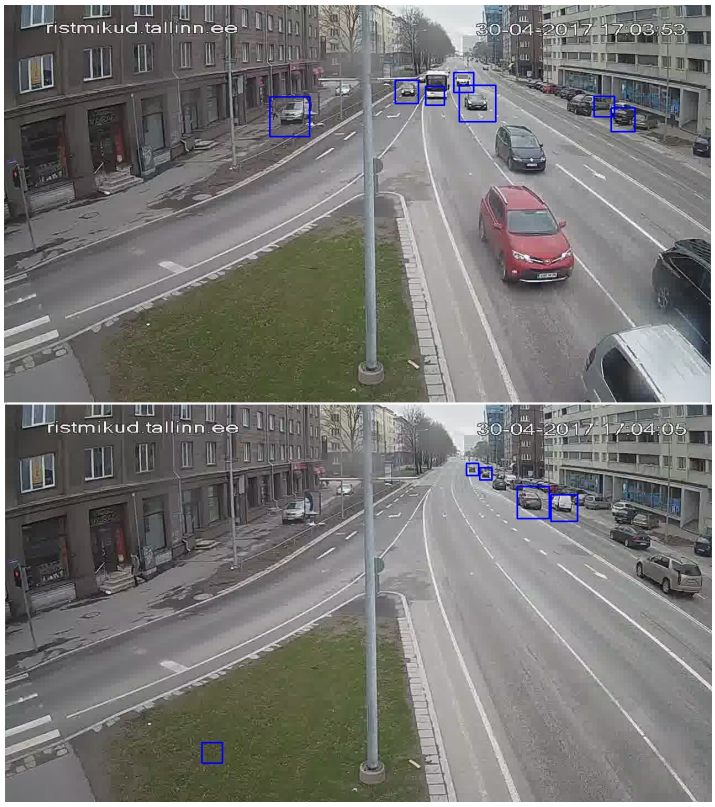


Figure 9: Haar cascades trained on rear views of cars are good at detecting vehicles in a similar orientation. The front views are similar enough.

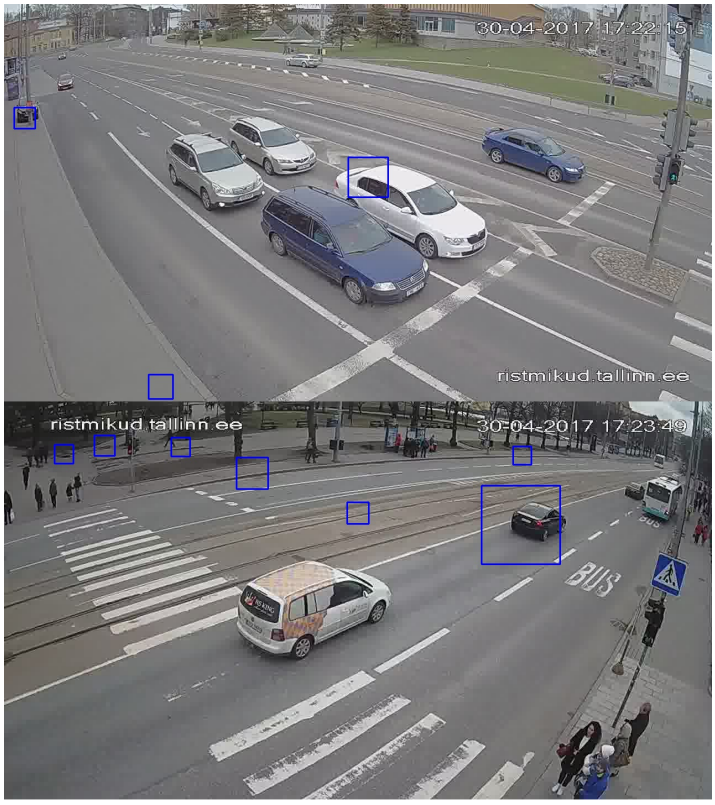


Figure 10: Haar cascades trained on rear views of cars are bad at detecting vehicles in other orientations.

Figure 11: Using Haar cascades, false-positives are often more numerous than true-positives. Because many false detections are very small in scale, limiting the minimal detection size might help.

4.4 Single Shot Multibox Detector (SSD)

I tested the SSD [7] model, which is a Fully Convolutional Network (FCN)[1] (like Ovefeat), for object detection. I used the PyTorch [24] implementation from [25]. The model only takes 300x300 resolution images, but internally produces detections on many scales. Because the aspect ratio of input images is rarely 1:1, we have the choice of warping the images to fit the model while producing distortions, or cropping the center of the image, leaving out two sides of the image. The architecture of the model is shown in Figure 12, and the detections are pruned by a Non-Maximum Suppression (NMS)[2] algorithm.

My tests showed that even with distorted images, the model worked well. This method has the best results so far, having the ability to distinguish vehicles even from dense traffic, as seen in Figure 13. Sometimes strange false detections are made, like in Figure 14, usually of classes we are not interested in, so we can ignore them. We can tune the amount of detections given, by changing the minimal acceptable confidence[3] (I used minimal confidence = 0.4, for my tests). Figure 15 shows some vehicles being left undetected, which might be fixed by lowering the minimal confidence at the risk of increase in false positives.
Advantages:

1. High accuracy.
2. Several implementations and pre-trained model parameters.
3. Low memory usage thanks to low resolution input.
4. Network can process multiple images in parallel for increased efficiency.
5. Only needs a single pass to detect many scales of objects.
6. Able to detect occluded objects.
7. Good at detecting many different types of objects.
8. Works in real time at 150ms processing time per image.

[1] FCN - a neural network consisting only of convolutional layers.
[2] NMS - if two or more detections have high overlap, keep only the one with the highest class score or confidence.
[3] The estimated probability of a detected object belonging to a class.

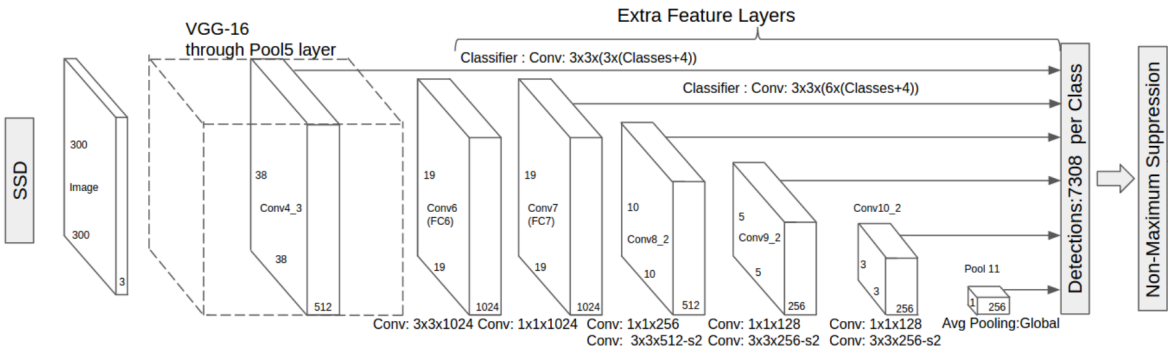


Figure 12: Architecture of the network used in SSD. Image adapted from [7].

Disadvantages:

1. Training needs images annotated with object bounding boxes.
2. A huge data set is needed for training (thousands of images).
3. Training takes a lot of time and an expensive GPU.
4. Needs a GPU to run at a reasonable speed.

In conclusion, SSD is an excellent choice for vehicle detection, if the necessary hardware is available. It is capable of distinguishing different objects and a wide range of classes, giving more accurate information than the other tested methods.

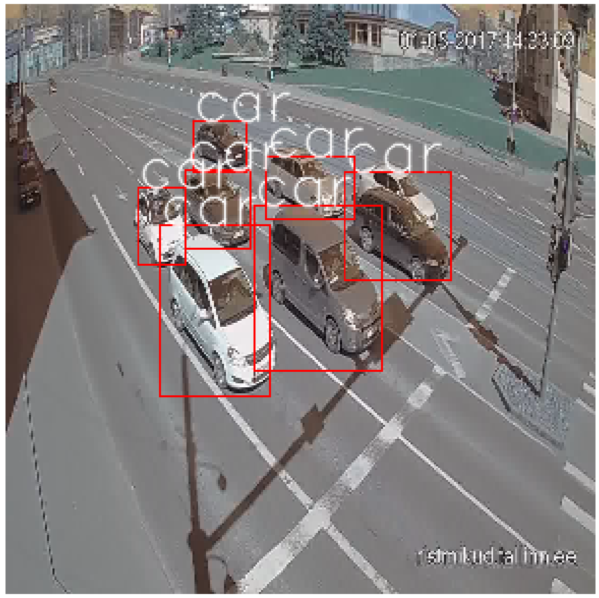


Figure 13: A pack of vehicles being detected by SSD. The model is great at distinguishing different vehicle instances.

Figure 14: A boat being incorrectly detected by SSD. Unimportant classes can be ignored.

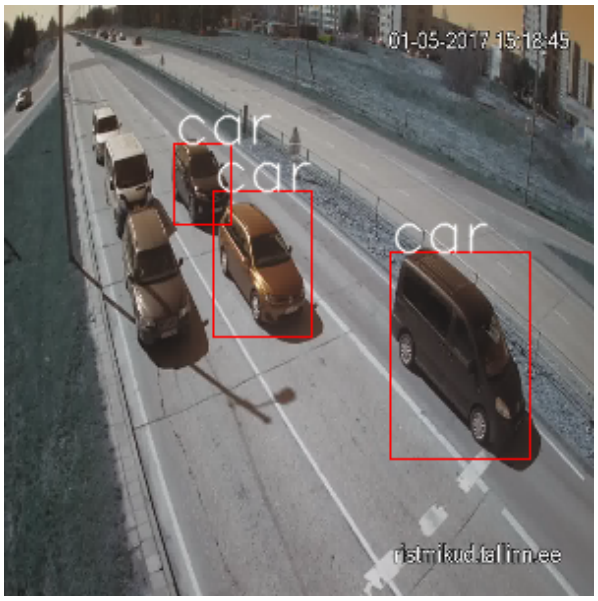


Figure 15: SSD does not always detect all vehicles. Reducing the confidence threshold might help, but at the risk of producing more false positive detections.

4.5 Proposed method

Taking into consideration the discussed methods and experiments, I propose and implement a vehicle detection method. I use a model based on SqueezeNet [26] - a convolutional neural network for image classification, as it 1) takes less resources than most popular CNNs, 2) is fully convolutional and 3) has a implementation with weights trained on ImageNet-1k [27] (with 1000 target classes) in PyTorch's model zoo [28], so I can take advantage of transfer learning[1] when training. While the details of it's architecture are not important, the way classification is done in SqueezeNet is crucial. The final convolutional layer of the network produces class scores for all regions of the image (Figure 16. To predict the class of the image, the mean class scores are taken from the entire image and squished via the softmax function (equation 6) to map scores into class probabilities. Finally, the class with the highest probability is chosen as the prediction.

$$softmax(x) = \frac{e^x}{\sum_{i=1}^{C} e_i^x} \tag{6}$$

[1] "The improvement of learning in a new task through the transfer of knowledge from a related task that has already been learned." [29]

Where C = number of classes and x = a class score.

Instead of classifying the entire image, we can run a sliding window over the map of class scores to detect objects. Training can still be done as originally intended: with single class classification over the entire image. In my model, I swapped SqueezeNet's 1000 class classifier with my own, 2-class classifier. I trained my model on my data set [21] with all images scaled to 176 pixels for the smaller dimension and then center cropped to a 176x176 px image and normalized to zero mean and $Variance = 1$ according to Imagenet means and standard deviations (as given in [30]).

The results indicate, that the main problem with the method is that it is bad at distinguishing object instances. Figure 17 demonstrates, how many detections are made on a single vehicle. One possible solution is to merge detections with high overlap, but in dense traffic, it is likely that many vehicles will be merged into one. Another is to apply non-maximum suppression, by only keeping the detections with the highest confidence. Interestingly, my model tends to give almost binary predictions - either 0% or 100%, making confidence based NMS unreliable. Furthermore, there are often detections on one vehicle that do not overlap, in which case neither method helps.

A possible improvement to the method would be to calculate the weighed average, not the mean in the sliding window, with weights penalizing high scores in the border regions of the window and boosting the scores in the middle. This would increase the confidence of detections where the vehicle is located in the middle of the window and decrease it in detections which are made only of a part of a vehicle. Currently, vehicles that are in a scale which would fit the sliding window nicely, are actually being ignored, like in the top left of Figure 17.

Advantages:

1. Works on static images.
2. Works on different vehicle orientations.
3. Trained like a classification model.
4. Can be used to detect many different classes of objects.

Figure 16: The model produces a map of class probabilities from the input image. Top image displays the predictions of class ”vehicle” from the bottom image.

5. Only about 150ms processing time on 720p images.

Disadvantages:

1. Requires GPU to run at a reasonable speed.
2. Bad at distinguishing different objects.

In conclusion, the proposed method needs more development to be useful, as it is bad at distinguishing different vehicle instances, making it inaccurate in traffic analysis.

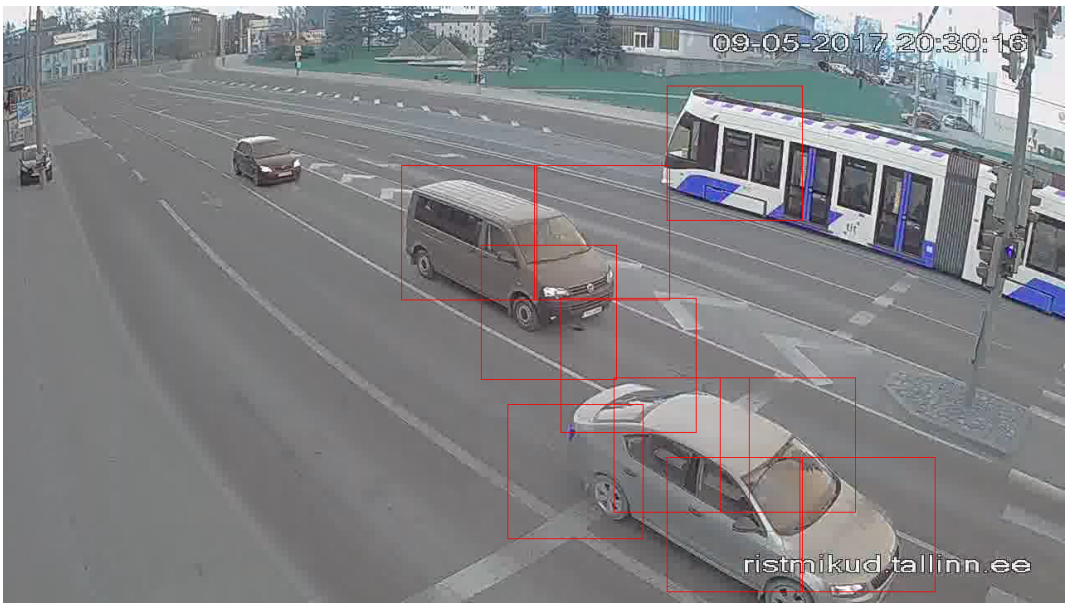


Figure 17: Detections made by the proposed model with highly overlapping detections removed. Still, there are usually many detections per vehicle.

5 Summary

As many cities have adopted traffic surveillance cameras, a new source of data about traffic has become available. Because it is not feasible for humans to constantly analyze these video feeds, vehicle detections systems are used to do it automatically, with many to choose from. In this thesis I compared different methods of vehicle detection in the context of Tallinn's traffic surveillance videos.

Pixel-wise classification methods like background subtraction and colormap classification are relatively simple and do not need much computational power, but proved to be bad at detecting vehicle instances. Using Haar cascades, implemented in OpenCV, had promising results, and is the method I would recommend using, if a GPU is not available.

If a GPU is available, CNN based methods should be considered, as they they have been shown to be very effective in object detection [11, 7, 12]. The fact that CNNs can give extra information by distinguishing many subtypes of vehicles makes them very attractive. Indeed, my tests with an implementation of one such model, SSD (implementation from [25]), produced the best results of all the tested methods and I would recommend using it, or a model like it, if possible. In addition comparing previously proposed methods, I experimented with my own CNN model, which is easier to train than SSD, but like pixel-wise methods, was not good at distinguishing vehicle instances.

6 References

[1] L.-W. Tsai, J.-W. Hsieh, and K.-C. Fan, "Vehicle detection using normalized color and edge map," *IEEE Transactions on Image Processing*, vol. 16, no. 3, p. 850–864, 2007.

[2] P. Negri, X. Clady, S. Hanif, and L. Prevost, "A cascade of boosted generative and discriminative classifiers for vehicle detection," *EURASIP Journal on Advances in Signal Processing*, vol. 2008, no. 1, p. 782432, 2008.

[3] X. Yuan, X. Cao, X. Hao, H. Chen, and X. Wei, "Vehicle detection by a context-aware multichannel feature pyramid," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, p. 1–10, 2016.

[4] C. M. Bautista, C. A. Dy, M. I. Manalac, R. A. Orbe, and M. Cordel, "Convolutional neural network for vehicle detection in low resolution traffic videos," *2016 IEEE Region 10 Symposium (TENSYMP)*, 2016.

[5] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "Overfeat: Integrated recognition, localization and detection using convolutional networks," *CoRR*, vol. abs/1312.6229, 2013.

[6] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *CoRR*, vol. abs/1506.02640, 2015.

[7] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg, "SSD: single shot multibox detector," *CoRR*, vol. abs/1512.02325, 2015.

[8] D. Ramanan, "Background subtraction." `https://www.ics.uci.edu/~dramanan/teaching/cs117_spring13/lec/bg.pdf`. Accessed: 25-04-2017.

[9] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of Computer and System Sciences*, vol. 55, no. 1, p. 119–139, 1997.

[10] Itseez, "Open source computer vision library." `https://github.com/itseez/opencv`, 2015.

[11] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," *ArXiv e-prints*, Mar. 2017.

[12] R. G. J. S. Shaoqing Ren, Kaiming He, "Faster R-CNN: Towards real-time object detection with region proposal networks," *arXiv preprint arXiv:1506.01497*, 2015.

[13] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Handwritten digit recognition with a back-propagation network," in *Neural Netwotks, current applications* (L. P.G.J., ed.), Chappman and Hall, 1992.

[14] "Theano documentation: Convolution arithmetic tutorial." `http://deeplearning.net/software/theano/tutorial/conv_arithmetic.html`. Accessed: 14-05-2017.

[15] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: common objects in context," *CoRR*, vol. abs/1405.0312, 2014.

[16] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results." http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[17] "Cuda parallel computing platform." `http://www.nvidia.com/object/cuda_home_new.html`.

[18] "Datasets, transforms and models specific to computer vision." `https://github.com/pytorch/vision`.

[19] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cudnn: Efficient primitives for deep learning," *CoRR*, vol. abs/1410.0759, 2014.

[20] H. Liik, "A variety of vehicle detection methods." `https://github.com/Leemur2/VehDetMethods/`.

[21] H. Liik, "Tallinn vehicles dataset." `https://github.com/Leemur2/TallinnVehicleDataset`, 2017. Accessed: 30-04-2017.

[22] "Haar-cascade detection in opencv." http://docs.opencv.org/2.4/modules/objdetect/doc/cascade_classification.html. Accessed: 30-04-2017.

[23] A. Sobral, "Vehicle detection with haar cascades." https://github.com/andrewssobral/vehicle_detection_haarcascades. Accessed: November, 2016.

[24] "PyTorch." https://github.com/pytorch/pytorch.

[25] "SSD: Single shot multibox object detector, in pytorch." https://github.com/amdegroot/ssd.pytorch.

[26] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size," *CoRR*, vol. abs/1602.07360, 2016.

[27] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

[28] http://pytorch.org/docs/torchvision/models.html.

[29] L. Torrey and J. Shavlik, "Transfer learning." ftp://ftp.cs.wisc.edu/machine-learning/shavlik-group/torrey.handbook09.pdf. Accessed: 29-04-2017.

[30] http://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html#sphx-glr-beginner-transfer-learning-tutorial-py.