Vehicle Counting with Object Detection on Traffic Webcams

Ana Schclar Leitão
madalena.schclar@tecnico.ulisboa.pt
Instituto Superior Técnico, Universidade de Lisboa
Lisbon, Portugal

Abstract

As the population residing in urban areas continues to grow, the need for understanding the traffic flow of a city is becoming increasingly more essential. Several cities have invested in the installation and deployment of Intelligent Transportation Systems (ITS). Some of the most commonly used sensors in ITS are traffic cameras used for analysis and monitoring purposes. Among the existing methods that can perform this analysis based on low-resolution camera footage, Object Detection is a compelling and yet to be specifically adapted approach. In this work, we are using a state-of-the-art general object detector, Faster-RCNN [1], and adapting it to real-world low-resolution traffic footage. The traditional applications of this type of object detectors present considerable differences to our intended application on traffic data. These differences pose the main challenge of this approach and are mentioned throughout our work. On the low-resolution traffic dataset we are using, our implementation of Faster-RCNN achieved a mean Average Precision (mAP) of 70%. Other state-of-the-art general object detectors achieved results 10% or 20% lower on the same testing conditions. In another traffic dataset, our approach achieved results comparable to those achieved by density estimation, a standard object and vehicle counting technique.

Keywords: Vehicle Counting, Object Detection, CNNs, Traffic Monitoring

1 Introduction

With the ever-growing size of the world population, and particularly the increase of those living in urban areas, urban planning and management have become an essential part of any large metropolis. Streets, roads and traffic are one of the main components of a city, hence, its population can only achieve a satisfactory lifestyle if proper traffic flow conditions are maintained. For this reason, cities are placing a large interest in traffic monitoring and planning. This interest has led to the development and evolution of Intelligent Transport Systems (ITS). The aim of ITS is to provide accurate information to road users, allowing them to make better use of the transportation networks. Recently, research on ITS has been going in the direction of computer vision and machine learning. Both these fields have seen tremendous advances, reaching an unprecedented level of accuracy and speed in image analysis and object detection. However, the transition from general computer vision techniques to traffic monitoring applications is not trivial. These approaches require large amounts of training data to perform correctly and the traffic camera data is available is not extensive and varies considerably in terms of quality and resolution.

Our main goal with this work is to adapt a general object detection model to real-world traffic data. Through the application of object detection, our aim is to perform vehicle counting, one of the essential tasks of traffic monitoring. The key challenge lies in the use of real-world traffic data. Most traffic cameras were not installed to be analysed through computer vision techniques, presenting serious problems to these approaches: **low resolutions, cluttered and busy scenes, small and vehicles**, among others. Considering all the impediments above, we intend to evaluate if object detection is a feasible method of vehicle counting to be used with the existing traffic camera networks.

2 Background

The ITS application we are focusing on is **vehicle counting**. To understand a region's traffic, it is essential to know how many vehicles occupy the roads and at what times. Vehicle counting provides the information needed to understand the use and capacity of the current road infrastructure and also to plan new roads. Large quantities of data are required for this purpose and they tend to be analysed offline. There are two main methods to count vehicles in camera footage: detection and density estimation. Our primary concern is vehicle detection.

Vehicle detection is a specific application of object detection. **Object detection** is one of the fundamental tasks of computer vision. It is the task of determining whether an object of a particular class is present in a given image and returning its spatial location. With **vehicle detection**, the systems analyse input traffic photos and provide as output the locations and classes of vehicles in the image.

There are some **challenges** in applying general object detection techniques directly to traffic monitoring applications. Most arise from the differences in data type and quality. Most traffic footage is of **low-resolution** and **low frame-rate**. The images present very **dense** and **cluttered** scenes with vehicles at severely **different scales**. A substantial number of vehicles have **extremely small dimensions** in the images. Another characteristic of these images is the **occlusion** of the vehicles. In heavy traffic, several vehicles are almost completely hidden by others. Vehicle detection is concerned with only two main classes, vehicles and, less often, pedestrians. However, systems should be capable of recognizing the small distinctions between **sub-classes** of vehicles. Two other influencing factors are **illumination**, mainly the daytime vs nighttime differences, and **weather**, harsh weather conditions (heavy rain, snow, fog) are quite detrimental to the quality of the video captured.

Related Work - Here we present some works on vehicle detection that use deep learning methodologies. However, unlike with object detection, there are still several studies based on traditional detection techniques being proposed. This is especially evident for works focusing on low-resolution traffic camera data. In vehicle detection, researchers have mainly focused on applying deep learning detectors to autonomous driving problems. In general, a detector, such as Faster RCNN [1] and YOLO [2], is trained and tested on a high-resolution dataset, captured with on-vehicle cameras. The main benchmark for these systems is the KITTI dataset [3] and very successful results have been achieved. In [4] Faster RCNN is

trained and reported on KITTI and in [5] it is applied to Stanford Cars Dataset [6].

Regarding traffic camera footage, in [7] the authors propose a three module system to track and efficiently query the obtained data, using YOLOv2 [8] as the detector. They use their own traffic camera data (720p resolution) to train and report results on vehicle counts. In [9], the authors present a new highway traffic dataset with vehicles at very distinct scales. The system they propose begins by dividing the road area in the video into two, the proximal area and the remote area. Both areas are separately submitted through YOLOv3 [10] detector. The obtained vehicles are merged and the ORB algorithm [11] is used to obtain features from each object. These features are used to track vehicles and compute their trajectories in the video stream.

As mentioned, another method to count vehicles is through **density estimation**. The idea is to find a mapping between local image features and the corresponding density map. On said map, the count of vehicles is obtained by summing the pixel values. In this method it is unnecessary to find individual vehicles. Guerrero-Gomez-Olmedo et al. [12] analysed two counting by density models [13, 14] on their dataset, TRANCOS, a dataset especially designed for counting severely overlapped vehicles. In 2017, Zhang et al. [15] published a work focusing on low-resolution traffic webcam data. They propose two solutions to the problem. The first solution is an improvement based on [13], where they introduce geometry information by dividing a target region into blocks. Their second solution is a FCN model that simultaneously learns vehicle density and vehicle count, using convolutional and deconvolutional layers. Both these solutions achieved positive results and adapt well to different vehicle scales.

3 Vehicle Counting By Detection

Our work focuses on the traffic monitoring task of vehicle counting through detection. The goal is to apply a well-known general object detection architecture to webcam traffic data. In this section we present the detection architecture and the datasets used. as well as the metrics used for evaluation.

3.1 Faster RCNN Architecture

In this project, we chose to implement the Faster RCNN Architecture [1]. It is one of the most precise detection models while also being one of the most adaptable due to its modular architecture. The model can be divided into three parts: (1) feature extraction, (2) region proposal, and (3) classification. Figure 1 is a representation of this architecture.

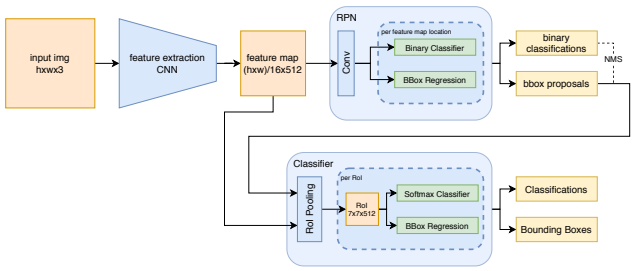

Figure 1. Representation of Faster RCNN architecture.

To briefly describe the architecture: an input image goes through a feature extraction CNN (VGG16 [16]), this CNN is known as the shared layers of the network. A feature map is computed and used as input to the region proposal network (RPN). This module uses a sliding window-like technique to extract a fixed number of box proposals in all image locations. The number and dimensions of the proposals at each location is also fixed. The RPN module applies two simultaneous loss functions to each box proposal, a binary classifier to attribute the probability of an object being represented by that box and a bounding box regressor which improves the proposal's location. The last section of the network, the classifier, receives as input the feature map from the CNN and the best region proposals from the RPN. Each of those proposals is extracted from the feature map and warped to a fixed size. A softmax function is used to produce class probability scores and another box regression is applied to further improve the proposal's location. The final output of the network are all these class probabilities and bounding boxes.

Training Details - The network has 4 losses, two in the RPN and two in the classifier. Also, it has a modular architecture, in which two separate parts of the network share the convolutional layers of the feature extraction network. This demands particular training techniques. Three different techniques are proposed by Ren *et al.* in [1]. In this project we will adopt the method applied in the paper, the 4 step alternating training:

1. The RPN module is trained with an ImageNet pre trained VGG16 model. Only the altered RPN layers are saved at the end of this step, as well as the region proposals.
2. The classifier module is trained, with the same ImageNet pre trained VGG16 model as the previous step and using the regions proposed by it. Both parts of the network are trainable and their updated weights are saved.
3. The RPN module is again trained, initializing the VGG layers with those obtained in step 2 and the RPN layers with those obtained in step 1. The VGG layers, also known as shared layers, remain fixed. This step is used to fine-tune the RPN to the shared layers trained with the classifier, actually accomplishing the sharing of the feature extraction layers.
4. On the last step, the classifier layers are fine-tuned using the region proposals from the previous step and the same fixed shared layers.

To implement this training technique, we joined the 4 steps in 2 phases. Phase 1 contains steps 1 and 2, while phase 2 contains the remaining steps.

Implementation Details - We will be using as foundation a Faster RCNN implementation available at [17] on GitHub. It is a Python project using the Keras deep learning framework. This implementation is quite similar to the original one by Ren et al. in [18] written in Matlab and using the Caffe [19] framework. As backend we will use Tensorflow.

3.2 Traffic Datasets

For this project, we need real-world cluttered scenes captured by traffic cameras. Preferably, footage of lower resolution as that is the norm for most traffic monitoring video systems. For these reasons, we will be using the CityCam dataset for training. Additionally,

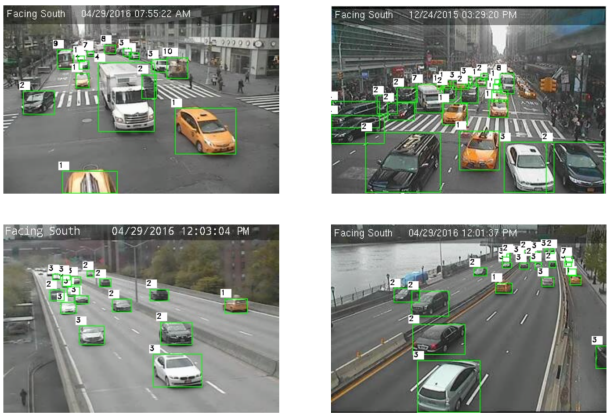

Figure 2. Frames from the CityCam dataset with annotations. The two top frames represent an urban traffic scene and the bottom two represent highway traffic.

we will be analysing non-annotated traffic data from the city of Tallinn.

CityCam - CityCam is a large labeled traffic webcam dataset created for traffic analysis by [15] (Figure 2. The dataset is composed of 60 million frames obtained from 212 webcams over a 4 week period. 60,000 of those frames have been annotated, containing around 900,000 labeled vehicles. It depicts real world urban traffic scenes on a large city, New York City. The annotated frames belong to 16 different cameras, 11 are placed on typical urban intersections while 5 are on higher traffic and higher speed roads, similar to highways. The annotations provide information on the vehicle type and count, bounding box locations, camera orientation, the time and the weather when the footage was obtained. In the next section we present a more thorough analysis of the CityCam data.

Tallinn data - To perform further traffic analysis, we have available a dataset comprised of traffic footage from over one hundred cameras installed throughout the city of Tallinn. This data is not annotated for detection, however, there is a density estimation score for each frame. The purpose of analysing this data is to compare both methods of Vehicle Counting and to assess our work on a different dataset.

3.3 Evaluation Metrics

The performance of object detection algorithms is evaluated based on three criteria: precision, recall and speed. Generally, precision and recall are represented by a single metric, mean Average Precision (mAP).

There is one more measure essential to object detection, the **Intersection over Union** (IoU) overlap score. IoU is a measure of overlap between two bounding boxes. Having two bounding boxes, their area of overlap (intersection) and area of union are computed, they are then divided giving

$$IoU = \frac{Area\,of\,Intersection}{Area\,of\,Union} \tag{1}$$

To determine whether a prediction is suitable, a threshold is applied to the IoU score. We will be testing several threshold values to determine a suitable one for our problem.

Precision is the fraction of the detections made that are correct. **Recall** is the fraction of correct detections made compared to the detections that should have been returned. In more general terms, precision is the number of TP (True Positives) over the number of TP + FP (False Positives) and recall is the number of TP over the number of TP + FN (False Negatives). For each class and each image i, the detector returns predictions as (b_{ij}, s_{ij}), where b_{ij} is the predicted location and s_{ij} is the confidence of prediction j. The locations predicted are matched to ground-truth bounding boxes, using an algorithm described in [20], and binarily scored according to a certain threshold applied to the IoU overlap result, the score (z_{ij}) is 1 if the boxes IoU is greater than the threshold, and 0 otherwise. The formulas for precision and recall are as follow:

$$Precision(t) = \frac{\sum_{ij} 1[s_{ij} \geq t] z_{ij}}{\sum_{ij} 1[s_{ij} \geq t]} \tag{2}$$

$$Recall(t) = \frac{\sum_{ij} 1[s_{ij} \geq t] z_{ij}}{N} \tag{3}$$

where N is the number of ground-truth instances of a certain class, across all images.

These two metrics are combined to compute the **Average Precision**. For each class, AP is calculated as the average of the maximum precision over the different levels of recall, or, in other words, AP is the area under the Precision-Recall curve. We sort the predictions in a descending order according to confidence score, incrementally compute the precision and recall and plot all these values, obtaining the Precision-Recall curve. Before computing the area under the curve, the precision in interpolated. The precision at each recall value becomes the maximum precision at that recall or higher (4). The AP is then computed as the mean of the interpolated precision at all the recall values (5).

$$p_{interp}(r) = \max_{\tilde{r}:\tilde{r} \geq r} (p(\tilde{r})) \tag{4}$$

$$AP = \int_0^1 p_{interp}(r) dr \tag{5}$$

The **mean Average Precision** (mAP) is simply the average of the AP over all classes. In our evaluation we will be using three levels of mAP according to bounding box dimension. The box dimension values are adapted to the CityCam dataset and as follows:

- small: width $\leq$ 13px;
- medium: 13px $<$ width $\leq$ 29px;
- large: width $>$ 29px.

These values are explained in 4.1. From these values, the difficulty levels are: (1) easy - large boxes; (2) medium - medium and large boxes; (3) hard - small, medium and large boxes.

Detection speed is measured in FPS (Frames per Second), *i.e.* how many images the detector can evaluate in one second. The constraints to this measure depend immensely on the application. Vehicle counting is usually employed as an offline method. This implies that time constraints are not too severe.

4 CityCam: Analysis and Training

The most significant part of our work is the adaptation of a general object detection architecture, Faster RCNN, to the low-resolution traffic dataset CityCam. To achieve this we begin by analysing the data in the first section. In the second section, we adapt the network's parameters with the results obtained by studying CityCam.

4.1 Analysis of CityCam Dataset

In this section we analyse the CityCam dataset. The information given is on the following topics: data format, scene type, annotation box statistics, box scales and ratios, class distribution and finally, weather conditions.

The CityCam dataset is available as video files and .xml annotation files. The majority of the videos present a **frame rate** of around 1 frame/second and all the frames have a **resolution** of 240x352 pixels. We converted the videos to individual .jpg frames and the annotations from a per frame .xml file to a general .txt file. Each line in this annotations file represents one vehicle instance in the following format: frame_path, x1, y1, x2, y2, class. The full dataset was partitioned as subsets of 90% for training+validation and of 10% for testing. The training+validation set was further partitioned into 80% for the training set and 20% for validation.

The CityCam dataset is composed of images from 16 different cameras, most of these are installed in urban intersections, depicting regular urban traffic. Observing the footage from each camera, we have decided to divide them into three **scene types**: urban (78.26%), highway (16.24%) and other (5.5%).

To understand the **dimensions of the vehicles** in this data, we computed the median values of the areas, widths and heights of all bounding boxes per camera. In Table 1, we show these and other statistics per scene type.

With these values we can confirm two expected characteristics from this type of traffic footage: (1) A frame tends to contain a very large number of annotations, 14 vehicles per image, on average. (2) The size of the annotations is extremely small for common object detection applications. The average width and height don't reach $25px$. Compared to the KITTI [3] traffic dataset where annotations must have a height greater than $25px$ and the MSCOCO challenge [21] where areas inferior to $1024px$ are classified as *small*. Also, we note that the highway cameras present considerably smaller median dimensions than the rest of the dataset. This could imply that the detector's performance will be lower on the highway scenes of CityCam.

To study the discrepancy in object **scales** in CityCam, we performed a clustering of the widths and the heights of all the annotations in the training+validation set. We used Kmeans to cluster each of the measures into three groups with the intention of obtaining three scales: small, medium and large. We also computed the three width/height ratios that best represent the overall set. The results attained from this analysis are the widths 13, 29 and 65 as **bounding box scales** and the **box ratios** [0.8 0.6], [1.1 0.8] and [1.4 1.1], where the first number in brackets is for the width and the second for the height. These values will be used during training given that the Faster-RCNN architecture requires the specification of bounding box scales and ratios in the RPN stage.

The **class distribution** in CityCam is imbalanced, as expected in real traffic conditions. Table 2 presents the class counts on the training+validation data, The three first classes solely constitute over 85% of the entire dataset. These are the vehicle subclasses that represent the *car* class. All other vehicle types occur a noticeably smaller number of times, which might become an obstacle to the correct classification of instances belonging to these classes. Comparing the class distributions between the different traffic scenes and the total set, we can note that the urban cameras present the most similar distribution to the total set one. This is expected given that most of the cameras are installed in typical urban roads. We also see that the *highway* set is almost entirely composed of black sedans and other cars (91%) and the *other* set has considerably more trucks than the rest.

To determine the **weather conditions** affecting the CityCam data we observed each of the videos, dividing them into four weather categories: normal (151 videos), rain (23 videos), heavy rain (5 videos), shadows (21 videos). The frames classified as *heavy rain* present a much more severe distortion and it is unlikely that an object detector will perform acceptably on these images.

This dataset contains a few **errors** in its annotations, mostly by having extremely small areas that don't represent any vehicle. These are unlikely to have any effect on the detection algorithm, due to the way in which the training data is generated. Also, these don't occur enough to be significant.

4.2 Network Parameters for CityCam

The Faster RCNN architecture was created having general object detection tasks in mind. Considering the main differences between this type of task and vehicle counting, it is likely that adapting the parameters to the CityCam traffic data will improve the detection results. In this section, we present three of the most relevant network parameters and how we adapt them to our data: input size, anchor scales and ratios, IoU thresholds.

The **input image size** is one of the most significant parameters in this architecture. It affects the dimensions of the feature map produced by the Feature Extraction module (the VGG) which will, in turn, affect the number of anchor locations to be analysed. The VGG used, outputs a feature map with dimensions 16 times smaller than the input's. In table 3, we show how different rescalings of the input image (from 1x to 3x) alter these details, assuming the image dimensions of CityCam, 352x240px. By performing a scaling of 2.5 times, i.e., equivalent to the parameters in [1], the network has 6.25 times more anchor locations to evaluate, which will have a considerable impact on the amount of time the network spends per image. However, while expanding the image size decreases the time efficiency, it is practically a necessity in the CityCam dataset. In Table 1, we can observe that over a third of the cameras have annotations with a median area smaller than 256px = 16x16. This means that, when not performing a scaling of the image, half of the annotations' areas are smaller than the windows considered, and consequently not captured by the network. When using an input two times larger that the image, the sliding window area corresponds to a 64px = 8x8 area in the original image, which is an area small enough to capture most of the vehicles in the dataset.

In the RPN module, Faster RCNN uses **fixed-size anchor boxes** to extract region proposals from each section of the feature map. During training, these proposals are considered positive only when they have an IoU higher than a threshold with a ground-truth (GT) box. For this to be the case, the dimensions of the anchors have to be representative of the GT object's dimensions. In the original implementation, the anchor scales and ratios were defined to possibly represent large objects from diverse classes. They applied nine bounding box proposals per feature map location. These are combinations of the scales 128, 256, 512 with the ratios 1:1, 2:1, 1:2. We will be comparing these to the values computed for our data in the previous section (4.1): [13, 29, 65] and [0.8:0.6, 1.1:0.8 , 1.4:1.1] for the scales and ratios, respectively.

Table 1. Analysis of CityCam training set. The table shows the distribution of frames and annotations per scene type, as well as the median of annotations in each image. The last three columns present the median area, width and height.

	Num Frames		Num Annotations per camera		Num Annotations per frame	Median Annotation Dims		
Cam id	count	%	count	%	median	area	width	height
urban	39935	78.27	518476	72.25	13	349	21	16
highway	8284	16.24	152630	21.21	18	127	13	9
other	2830	5.5	46841	6.53	16	387	26	15
Total	51049	100%	717947	100%	-	-	-	-
Avg	-	-	-	-	14.38	319.56	20.19	15.13

Table 2. Class counts in training+validation set. The table contains the class distributions for the total set and for each of the three types of scene. Values above 15% are shown in bold font.

class id	1 taxi	2 black sedan	3 other car	4 small truck	5 medium truck	6 big truck	7 van	8 medium bus	9 big bus	10 other	total
total count (x1000)	113	240	259	12	19	4	27	7	17	19	718
total %	**15.72**	**33.48**	**36.05**	1.72	2.7	0.61	3.7	1.04	2.32	2.66	100%
urban %	**19.1**	**30.8**	**32.76**	1.81	3.04	0.49	4.03	1.26	3.12	3.61	100%
highway %	4.99	**44.27**	**46.76**	1.22	0.35	0.03	1.85	0.29	0.03	0.19	100%
other %	13.28	**28.04**	**37.55**	2.35	6.57	3.82	6.04	1.05	1.01	0.28	100%

Table 3. Effect of different image scalings on the region proposal stage. This table presents network details using five possible scalings of CityCam frames, from 1x to 3x. The feature map size is 16x smaller than the image size. Using a sliding window, anchors are applied to each feature map location. The location's size and area are shown relative to the original image size. The number of anchors is calculated assuming nine anchors per window location.

Scale	1x (original)	1.5x	2x	2.5x	3x
Input img size	352x240	528x360	704x480	880x600	1056x720
Feature map size	22x15	33x22.5	44x30	55x37.5	66x45
Num window locations	330	726	1320	2035	2970
Location size	16x16	10.7x10.7	8x8	6.4x6.4	5.3x5.3
Location area	256	114	64	41	28
Num anchors (k=9)	2970	6534	11880	18315	26730

During the RPN training, **IoU thresholds** are applied. IoU overlap is the main metric to assess how appropriate a box proposal is to a ground-truth bounding box. In RPN, these thresholds which determine whether a box proposal is positive or negative, i.e., if it contains an object or not. To do this, two thresholds are used, and upper and a lower one. The following are the criteria used by the network:

- Positive proposal:
  - highest IoU score with a GT box
  - IoU > upper threshold with any GT box
- Negative proposal:
  - IoU < lower threshold with all GT boxes

All other boxes are irrelevant during training and are thus eliminated. An analysis performed on a sample of the dataset (around 10%), showed that, using the anchor dimensions previously defined in this section, the average IoU is of around 0.55. As such, it is more appropriate for this data to use small thresholds. Such as 0.5 and 0.1, for the upper and lower threshold respectively. At the end of the RPN module, **non-maximum suppression** is applied using another IoU threshold, set at 0.7. If two proposals have an IoU overlap higher than this value, then only the one with the highest class confidence score is kept.

5 Results

This section presents the results achieved by training and testing our adapted Faster RCNN model on the CityCam dataset and testing on the Tallinn traffic data.

5.1 Training on CityCam

The training on CityCam was performed in two stages: initial parameter testing on a single camera and analysis of results on the full dataset. We adopted this method to minimize the amount of training time spent on the entire dataset.

Single Camera

The camera chosen for the initial parameter testing is the one with the highest amount of annotations and it's taken from a typical urban intersection (Cam id = 398). Around 4000 frames from this camera were used for training, the class distribution resembles the

total one (in Table 2) and the median dimensions of the annotations are similar to the total average.

To test and perform parameter adaptation we trained several models, introducing only one change in each new one. The base model follows the initial parameters, the changes introduced in each model are: (a) image input size to 240px and our anchor scales and ratios, (b) adam optimizer, (c) our iou thresholds, (d) amsgrad, (e) weight decay (f) 2000 regions of interest, (g) and image input size to 480 px. The models were trained from an Imagenet pre-initialised VGG16 and following the technique in Section 3.1 for 8 epochs per phase. As preprocessing, the images are normalised and the mean pixel per depth channel is subtracted. Table 4 contains the testing results of each model in the detection and classification task. All testing in this section was performed on the validation set.

The first model provided very poor results. As expected, barely any correct detections were made due to the size and scale of the anchor box proposals. These dimensions were specified for general object detection, where the focus is on a few large objects. Unlike traffic vehicle detection, where images contain a large number of vehicles at different scales. This is the only model trained with an input image height of 600px (shortest image side). The size of the input affects tremendously the model's training time, however, in this architecture, it has a great positive effect on the results. As explained in Section 4.2, the input image size dictates the number of anchor box locations and sizes: a larger image implies more and smaller locations. For the purpose of tuning the parameters, we trained most models without resizing the images (height = 240px), only on the last model did we upscale them to twice the size (height = 480px) to compare the results. As we expected, the greatest improvement resulted with the upscaling of the input image in model *g*. However, this increased the testing time, confirming the obvious trade-off between detection accuracy and time. In this traffic monitoring application, a frame rate of two images per second is entirely acceptable so it is preferable to have higher accuracy and lower frame rate.

With this analysis on only one camera, we concluded that our parameter definition in the previous chapter (Section 4.2) positively affects the performance of Faster RCNN on the CityCam traffic dataset. Having achieved these results, we proceeded to train a model with the parameters equal to model *g* on the full CityCam training set.

Full Training set

After optimizing and testing on a single camera, the network was trained simultaneously on all 16 cameras using the parameters of model *g* in Table 4. As before, the frame rate during testing is around 2 frames per second. In Table 5, we show the detection and classification results obtained on the full CityCam dataset. The results are presented by scene type: urban, highway and other.

Observing the table, we note that, as expected, the best values overall are obtained on urban scenes. This occurs because most of our training data portrays this type of traffic scene, as such, the model is more adapted to typical urban footage and performs better.

On the detection results, we can see that the mAP is slightly better for the urban scene frames, 83.88% vs 80.56% for all cameras. However, in terms of recall, all camera types show similar results at the medium and easy difficult levels. The highway scenes obtained worse recall levels only at the hardest difficulty. This is justifiable by the smaller average size of annotations in this scene type, seen in Table 1. On the classification results, we see that the mAP of the main three classes is slightly higher than the total one, on all scene types. Comparing the detection results to the classification ones, there is a difference in the mean APs, this is however, much less severe in the urban cameras. This indicates that vehicles in the urban traffic footage are not commonly misclassified. The recall values show the same as the mAP, for detection only, the value is slightly higher than for both localization and classification.

Table 6 contains the per class Average Precision for the three scene types, and the all over average. In bold font we highlight APs above 80%. We can see that only urban cameras obtained results above 80%. We can also observe that, when considering all the cameras, the highest scores occur on two of the three main classes: taxis and black sedans. These two classes form around 50% of the dataset.

Figure 3 contains some examples of detections made on CityCam by our model. The annotated bounding boxes are colour coded as: green - true positive, red - false negative and blue - false positive.

To study the **effect of weather conditions** on the performance of our implementation, we divided the test set into 5 subsets: (1) rain, (2) heavy rain, (3) shadows, (4) whole set minus rain, (5) whole set minus rain and shadows. Table 7 contains the obtained classification results. These correspond accurately to what would be expected. The worst results occur on the *heavy rain* set. As we can see, this type of rain severely affects the detection, the medium difficulty mAP is 25% lower than the mAP of the whole set. In the *rain* set, the effect is less intense. The mAP is 6% lower. The presence of shadows seems to have a minor effect, with a mAP difference of less than 2%. Considering these three conditions had a negative influence on the performance, the best results occur on the set with no rain or shadows. Overall, the results on this set are only slightly better than those achieved on the entire testing set. This indicates that the unfavourable weather or illumination conditions don't occur frequently enough to have a high impact on the general performance scores.

Due to the discrepancy in results between urban cameras and **highway cameras**, we decided to train a model uniquely with the three highway cameras. Table 8 shows the results obtained by this highway model and compares them to the ones obtained by the general one. From these results, we see that the general model behaves better when taking into account the smaller annotations. However, the highway model provides better results at the medium a easy difficulties. The results are very similar in both implementations, but, we believe that with a larger and more extensive highway training set, the highway only model would provide more satisfying results.

One of the challenges of vehicle detection is the lack of benchmarks and systems to compare and evaluate models. Vehicle detection systems are very dependent on the application and most publications use private datasets. To sidestep this issue, we decided to **compare our model to state-of-the-art object detectors**: Mask RCNN [22], Faster RCNN [1] and RetinaNet [23]. We are using the pre-trained implementations available on Detectron2 [24], a detector software from Facebook AI Research. All of these models have ResNet50 [25] as the backbone network and are trained on the MS COCO dataset, with over 200 thousand images. To do this comparison we selected a random sample of 500 images from the CityCam test set and evaluated the three Detectron2 models and

Table 4. Detection and classification results using different Faster RCNN parameters. The table shows, for the detection only task, the mean Average Precisions (mAP) and mean Recall (mRec) at the three difficulty levels defined in 3.3. The next column contains the detection rate in seconds per image. The final two columns contain the mAP and mRec for the classification task at medium difficulty.

	mAP			mRec			rate	classification (m+l)	
model	s+m+l	m+l	l	s+m+l	m+l	l	(sec/img)	mAP	mRec
base	15.19	22.25	43.47	19.31	28.28	55.25	0.7	8.51	23.91
a	13.64	18.81	37.98	25.34	34.95	70.57	0.4	15.77	26.82
b	34.82	45.07	68.11	45.97	59.5	89.93	0.2	43.69	54.83
c	42.29	54.04	73.5	54.2	69.26	94.2	0.2	53.63	66.32
d	43.31	55.38	76.42	53.55	68.48	94.5	0.2	53.65	65.42
e	43.25	55.34	76.47	53.17	68.02	94	0.2	53.34	64.94
f	44.07	56.15	74.05	56.62	72.15	95.15	0.2	54.7	69.35
g	**79.96**	**88.32**	**90.87**	**87.73**	**96.9**	**99.7**	0.5	**86.62**	**94.67**

Table 5. Detection and classification results on the full CityCam dataset by traffic scene type. The table shows, for the detection only task, the mean Average Precisions (mAP) and mean Recall (mRec) at the three difficulty levels defined in 3.3. The final three columns contain the mAP of all classes, the mAP of the main three classes and the mRec for the classification task at medium difficulty. The best values for each metric are presented in bold.

	mAP			mRec			classification (m+l)		
scene type	s+m+l	m+l	l	s+m+l	m+l	l	mAP	mAP top 3 cls	mRec
all cams	69.56	80.56	85.35	80.01	93.15	98.77	72.49	74.15	84.88
urban	**74.54**	**83.88**	**88.57**	**83.22**	**93.74**	**99.01**	**79.64**	**80.87**	**88.8**
highway	50.99	70.95	75.87	66.06	92.17	98.76	56.83	58.27	78.03
other	69.99	76.75	81.83	83.31	91.36	97.42	54.68	61	73.65

Table 6. Classification results on CityCam per traffic scene type and class.

scene type	taxi	black sedan	other car	small truck	medium truck	big truck	van	medium bus	big bus	other
all cams	78.62	78.92	67.3	45.27	68.56	40.53	65.02	61.73	78.57	40.65
urban	**84.1**	**83.84**	74.11	59.82	**83.66**	45.13	77.8	74.09	**88.13**	45.09
highway	66.26	67.36	52.42	14.3	24.48	4.76	33.45	39.55	-	38.69
other	66.98	69.22	52.23	18.95	51.6	35.42	42.09	27.01	26.03	18.17

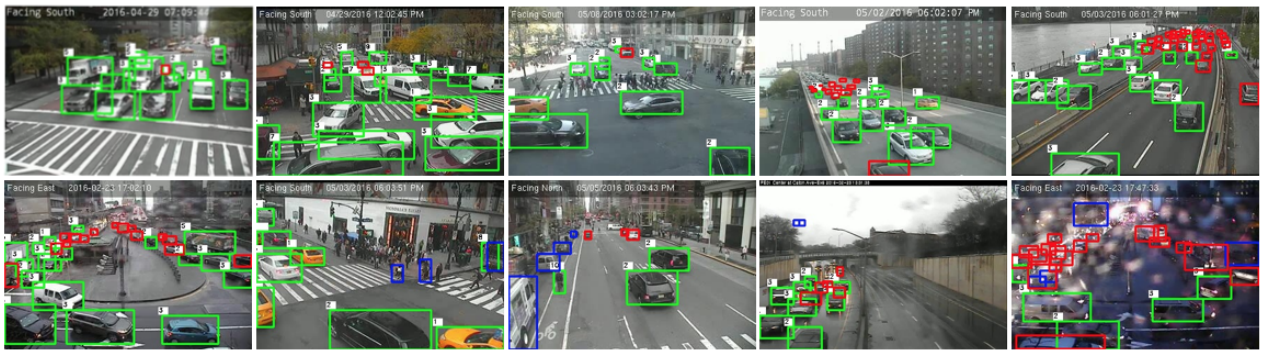

Figure 3. Detections made by our model on the CityCam dataset.

our implementation on the detection only task. Table 9 contains the results obtained. Surprisingly, our model fared better on all metrics. While the Detectron2 models have not been trained on this data, they were submited to a much more extensive training and all three are the newest and best implementations of each network. The biggest difference in results occured in the mAP at the medium difficulty, where medium and large vehicles are considered. As mentioned before, this is the metric we consider the most relevant. Our implementation achieves a mAP close to 10% higher than the second best model. All other results are fairly similar. As expected, no network performs well when the smallest vehicles

Table 7. Results of weather conditions on detection.

	mAP (%)			mRec (%)	
	s+m+l	m+l	l	s+m+l	m+l
rain	59.96	71.81	79.41	71.05	85.38
heavy rain	41.14	51.97	62.34	58.88	75.41
shadows	59.28	76.16	81.83	69.99	90.54
no rain	63.91	81.33	85.69	72.36	92.37
no rain and no shadows	**64.41**	**81.86**	**86.07**	**72.61**	**92.57**
all	61.82	77.7	83.5	70.79	89.41

Table 8. Comparison of an implementation trained on highway only versus the general training.

	mAP (%)			mRec (%)	
	s+m+l	m+l	l	s+m+l	m+l
highway	38.7	**62.43**	**69.51**	51.4	**83.43**
general	**40.15**	60.26	68.72	**53.52**	80.67

are considered and all achieve acceptable results if only the largest vehicles are contemplated.

5.2 Testing on Tallinn Traffic Data

We tested our fully-trained vehicle detection model on different traffic data to better understand how it would behave on other footage. For this, we are using the Tallinn Traffic dataset mentioned in 3.2. This dataset was not created for object detection and has no bounding box annotations. However, each frame is accompanied by an estimated density score (number of vehicles in image) and density map. The lack of detection annotations implies that the assessment of our model's performance was done by manual observation and by comparison to the density scores.

We selected three cameras from the dataset: on a typical urban scene, on a highway and on a parking lot. In figure 4, we compare the detection counts we obtained with our model to the existing density counts in the urban location. In the graph we can see that both techniques provided similar results. This indicates that our detection model is as effective as the density method in the task of vehicle counting. Furthermore, detection can contribute with additional information such as vehicle type, size and location.

As we saw previously when testing on CityCam, our model has a slightly worse performance on highway scenes. The same occurs on the Tallinn highway data tested. Figure 5.a) shows the comparison between our detection counts and the data's density vehicle counts. Our model tends to detect less vehicles than the density method in this type of scene.

As for the parking lot camera, this is a type of scene not previously seen by our model during training nor testing. As such, we expect the results to be slightly less reliable. Figure 5.b) compares the density vehicle counts and our detection counts. As we can see, there is a discrepancy in the number of vehicles detected by each method.

Figure 6 shows examples of images detected by our implementation on the Tallinn traffic dataset.

6 Conclusions

The aim of this project was to assess the use of object detection to perform the task of vehicle counting on typical low resolution traffic data. To accomplish this, we adapted and trained a Faster RCNN model on the CityCam dataset and later tested it on this same dataset and on traffic footage from the city of Tallinn. We found that one of the main aspects that affects the results of our model is the road type. Most of the training data used represents urban streets and intersections. As such, the performance is much better in this type of setting versus a highway road, as we saw in both CityCam and the Tallinn data. Similarly, we also found that the results are better at daytime and with mild weather conditions. We are certain that all of these issues would be solved if more training data for these situations were available. Besides this, we also found that the size of the vehicles has a great influence on the outcome. Smaller vehicles, as expected, are less frequently detected. However, if we are using this method to count vehicles in a video sequence, it is not necessary to find the smallest vehicles in each frame, as they will be larger in the ones that follow. Finally, we found the counting results on the Tallinn urban footage very comparable to the ones obtained through density estimation. We thus conclude that both techniques are equally effective for vehicle counting and detection can provide additional information such as size, location and class. With this, we conclude that vehicle counting can be performed by a state-of-the-art object detector on low resolution traffic data. However, this method is not advisable if there is a necessity in detecting the smallest of vehicles. Due to the construction of object detectors, extremely small objects are missed or ignored. Additionally, this method requires some domain adaptation. Urban cameras and highway cameras produce considerably different scenes, as such, to perform well on both, the model has to be trained on both. The same can be said for unusual roads such as roundabouts and different illumination and weather conditions, within reason.

References

[1] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, 2015, pp. 91–99.

[2] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[3] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[4] Q. Fan, L. Brown, and J. Smith, "A closer look at faster r-cnn for vehicle detection," in *2016 IEEE intelligent vehicles symposium (IV)*. IEEE, 2016, pp. 124–129.

[5] A. Tourani, S. Soroori, A. Shahbahrami, S. Khazaee, and A. Akoushideh, "A robust vehicle detection approach based on faster r-cnn algorithm," in *2019 4th International Conference on Pattern Recognition and Image Analysis (IPRIA)*. IEEE, 2019, pp. 119–123.

[6] J. Krause, M. Stark, J. Deng, and L. Fei-Fei, "3d object representations for fine-grained categorization," in *Proceedings of the IEEE international conference on computer vision workshops*, 2013, pp. 554–561.

[7] L. Huang, W. Xu, S. Liu, V. Pandey, and N. R. Juri, "Enabling versatile analysis of large scale traffic video data with deep learning and hiveql," in *2017 IEEE International Conference on Big Data (Big Data)*. IEEE, 2017, pp. 1153–1162.

[8] J. Redmon and A. Farhadi, "Yolo9000: better, faster, stronger," *arXiv preprint*, 2017.

[9] H. Song, H. Liang, H. Li, Z. Dai, and X. Yun, "Vision-based vehicle detection and counting system using deep learning in highway scenes," *European Transport Research Review*, vol. 11, no. 1, p. 51, 2019.

[10] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv*, 2018.

[11] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *2011 International conference on computer vision*. Ieee, 2011, pp. 2564–2571.

[12] R. Guerrero-Gómez-Olmedo, B. Torre-Jiménez, R. López-Sastre, S. Maldonado-Bascón, and D. Onoro-Rubio, "Extremely overlapping vehicle counting," in *Iberian*

Table 9. Comparison between our implementation and other object detectors. The highest values of each metric are in bold.

		mAP			Recall		
		s+m+l	m+l	l	s+m+l	m+l	l
Detectron2	Mask RCNN	45.52	62.44	75.65	56.13	76.99	93.28
	Faster RCNN	44.69	61.23	74.47	55.61	76.19	92.66
	RetinaNet	38.88	52.98	62.88	58.72	80.02	94.87
	our	**48.54**	**70.09**	**78.36**	**58.78**	**84.88**	**94.89**

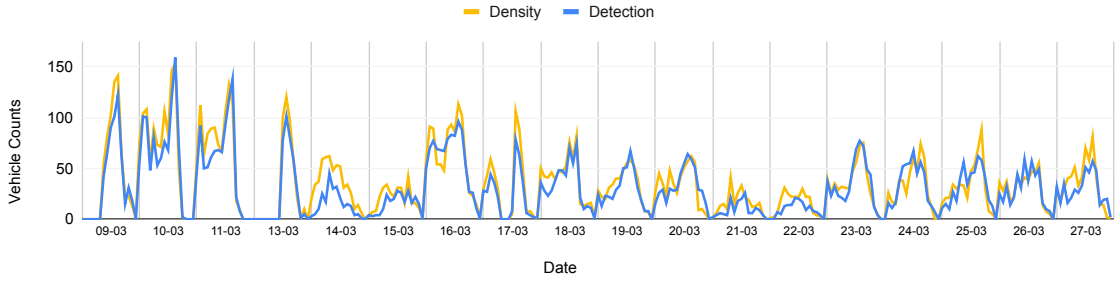

Figure 4. Our detection counts versus the density counts of the Tallin dataset. Both density and detection counts are plotted per hour. Each section in the horizontal axis corresponds to one day.

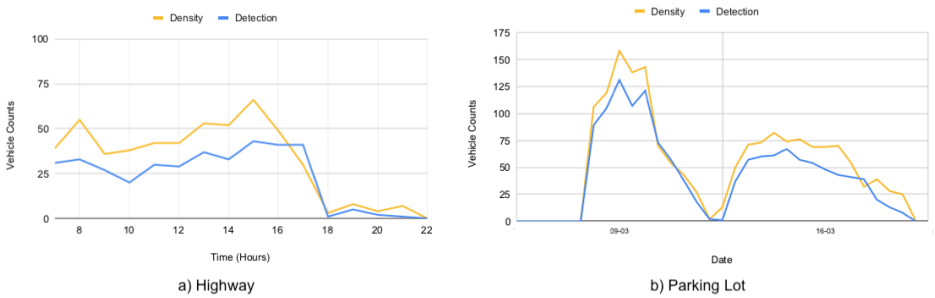

Figure 5. Detection vs density vehicle counts on a highway (left) and a parking lot (right) camera from the Tallinn traffic dataset.

Figure 6. Detections made by our model on the Tallinn traffic dataset.

Conference on Pattern Recognition and Image Analysis. Springer, 2015, pp. 423–431.

[13] V. Lempitsky and A. Zisserman, "Learning to count objects in images," in *Advances in neural information processing systems*, 2010, pp. 1324–1332.

[14] L. Fiaschi, U. Köthe, R. Nair, and F. A. Hamprecht, "Learning to count with regression forest and structured labels," in *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012).* IEEE, 2012, pp. 2685–2688.

[15] S. Zhang, G. Wu, J. P. Costeira, and J. M. Moura, "Understanding traffic density from large-scale web camera data," *arXiv preprint arXiv:1703.05868*, 2017.

[16] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[17] "Keras-fasterrcnn." [Online]. Available: https://github.com/you359/Keras-FasterRCNN

[18] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks." [Online]. Available:

https://github.com/ShaoqingRen/faster_rcnn
[19] Y. Jia, "Caffe: An open source convolutional architecture for fast feature embedding." [Online]. Available: http://caffe.berkeleyvision.org/
[20] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.
[21] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*. Springer, 2014, pp. 740–755.
[22] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick, "Mask R-CNN," *CoRR*, vol. abs/1703.06870, 2017. [Online]. Available: http://arxiv.org/abs/1703.06870
[23] T. Lin, P. Goyal, R. B. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," *CoRR*, vol. abs/1708.02002, 2017. [Online]. Available: http://arxiv.org/abs/1708.02002
[24] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, and R. Girshick, "Detectron2," https://github.com/facebookresearch/detectron2, 2019.
[25] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.