BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

DETECTION AND CLASSIFICATION OF VEHICLES FOR EMBEDDED PLATFORMS

DETEKCE A KLASIFIKACE VOZIDEL PRO VESTAVĚNÉ PLATFORMY

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **PATRIK SKALOŠ**
AUTOR PRÁCE

SUPERVISOR **Ing. JAKUB ŠPAŇHEL**
VEDOUCÍ PRÁCE

BRNO 2023

Abstract

An abstract of the work in English will be written in this paragraph.

Abstrakt

Keywords

Here, individual keywords separated by commas will be written in English.

Klíčová slova

Reference

SKALOŠ, Patrik. *Detection and Classification of Vehicles for Embedded Platforms.* Brno, 2023. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Jakub Špaňhel

Detection and Classification of Vehicles for Embedded Platforms

Declaration

I hereby declare that this Bachelor's thesis was prepared as an original work by the author under the supervision of Mr. X The supplementary information was provided by Mr. Y I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....
Patrik Skaloš
April 13, 2023

Acknowledgements

Here it is possible to express thanks to the supervisor and to the people which provided professional help (external submitter, consultant, etc.).

Contents

List of Figures

Chapter 1

Introduction

Vehicle detection in real-time is crucial for enhancing traffic safety and flow. It can be used to manage traffic at an intersection using traffic lights, spot speeding or wrong-way drivers, enhance route planning to ease congestion, alert drivers to potentially dangerous situations, and offer insightful information about driving behavior.

There are various methods for detecting vehicles, each of which has its advantages and limitations. In this paper, we focus on detection using a camera-based traffic surveillance system. This approach was chosen because of its low price, versatility, and ease of installation.

There are two main techniques for vehicle detection using a camera: image processing and convolutional neural networks (CNNs). Both approaches are based on analyzing images to identify patterns and features to detect vehicles. Image processing techniques involve applying a series of pre-defined filters and transformations to an image to extract relevant information. CNNs, on the other hand, are a type of machine learning algorithm that learns to recognize patterns and features through training. CNNs are generally more effective than image processing techniques for vehicle detection, as they can learn to recognize complex patterns and features that may be difficult to extract using pre-defined filters and transformations, but they have traditionally required a significant amount of computing power, commonly provided by a graphical processing unit (GPU). However, recent developments suggest that it may now be possible to perform CNN-based tasks in real time with reduced processing requirements.

In this paper, we attempt to train a CNN with a smaller architecture that is able to run on a single central processing unit (CPU) instead of a GPU. This would allow the system to process the camera feed on-site, rather than relying on its transmission over the internet and remote processing. This would reduce the cost of the system and the amount of network bandwidth required, as well as expand the range of purposes for which the system can be utilized.

Chapter 2

Background and Related Work

2.1 Common Approaches to Vehicle Detection

This chapter contains a summary of commonly used solutions for the problem of vehicle detection. First, we introduce systems that are not based on cameras, and then we compare CNNs with image processing techniques. **[[update]]**

Vehicle Detection Without a Camera

[[Many different types of detectors exist but none is perfect so they're used in harmony as complex systems consisting of many detectors?]]

Magnetic Sensors

One of the simplest solutions to detect vehicles is to use an induction loop.[3] The sensor, composed of a single wire, can be buried under concrete and detect the presence of metal objects passing by. With a controller, induction loops typically provide data about vehicle presence, but using more advanced algorithms, speed, approximate classification and much more can be determined from this simple sensor. Although the design is very simple, installation is not and induction loops can even get damaged over time, while repairs call for temporary shutdowns of roads. It is worth noting that there are many other types of magnetic sensors, which can provide more detailed and accurate data with simpler sensor installation, but the idea stays the same.

Ultrasonic Sensors

Another type of inexpensive and simple detector is an ultrasonic sensor.[10] These sensors can operate in a variety of conditions and are typically mounted on existing infrastructure. These distance measurement sensors are usually used to detect the presence and distance of nearby vehicles and their speed, but they can be utilized in many different ways to even provide a simple shape of a vehicle to classify it.

Radars and Lidars

Radars (Radio Detection and Ranging), which can also be mounted on existing infrastructure above ground, work similarly to ultrasonic sensors, but a single radar can oversee a much wider area of a road.[5] They are usually used to detect the presence, speed, heading

and shape of a vehicle. The shape can then be used to predict a class of the vehicle. Lidars (Light Detection and Ranging) can be used in a similar way, but are far more accurate, which makes them better at detecting shapes and locations of objects.

Vehicle Detection Based on Image Processing

Image processing techniques take a camera feed as input and in addition to detecting the presence, speed, heading and shape of a vehicle, they can also provide its color, license plate and countless other characteristics, limited mostly by the software. These camera-based systems, which are relatively inexpensive, can be mounted on existing infrastructure, do not emit energy and are highly versatile, while providing a large amount of data.

An algorithm applies a series of pre-defined filters and transformations to an image to extract patterns and features that resemble vehicles. Although modern convolutional neural networks (explained in the following section 2.2) are generally more effective at recognizing more complex patterns, image processing can still be extremely helpful for simpler tasks or as a component of a larger vehicle detection system.

A huge amount of research has been conducted on using image processing to solve the problem of vehicle detection.[12] Many of the techniques proposed can operate in real-time and are very reliable in typical environmental conditions. However, they can be sensitive to changes in illumination or have their performance affected by rain, snow, shadows, occlusions, or noise. While there are methods for addressing some of these challenges, such as shadow removal techniques[9], they add to the computational requirements of the system. Overall, our research suggests that the image processing approach is well-suited for simpler tasks or systems with lower accuracy requirements, but it is often difficult to implement and may lack versatility in more complex or demanding scenarios.

2.2 Convolutional Neural Networks

[[ako fungujú?]]

2.3 Object Detection?

2.4 Light-weight Detectors

2.5 Vehicle Detection

2.6 Tools and Libraries

LabelBox annotation app?

MMYOLO Library

In this project, we utilize an open-source library called MMYOLO**[[cite]]** which is a part of the OpenMMLab project**[[cite?]]**. It is an extension of the MMDetection**[[cite]]** library, which provides a flexible framework for various object detection tasks, supporting many state-of-the-art object detection models. The MMYOLO library instead only focuses on object detectors from the YOLO family and at the time of writing, supports the state-of-the-art YOLO detector – YOLOv8. Some of the key features of the library include:

- It is based on the PyTorch[1] machine learning framework
- It builds upon and uses other open-source libraries from the OpenMMLab project, such as MMCV[2], MMEngine[3] and MMDeploy[4]
- Library's modular design allows for easy customization and extension of the codebase
- It includes pre-trained models and their configurations which makes training and comparing different models fast and simple, while making it possible to use the transfer learning technique[[**footnote ak som nevysvetlil transfer learning v teórii**]] which significantly speeds up the process.
- Implementation of YOLO-specific components, such as the CSPDarknet and PANet backbone networks
- Support for YOLO-specific training techniques, like data augmentations or loss functions

[[**In practice - proste upraviť config a trénovať**]]

[1]PyTorch is a popular open-source machine learning framework used in computer vision and natural language processing

[2]MMCV is a library for computer vision research including building blocks for convolutional neural networks, tools for image processing, transformations and much more

[3]MMEngine library serves as the training engine for all OpenMMLab codebases, supporting hundreds of algorithms frequently used in deep learning

[4]MMDeploy library provides tools for deploying deep learning models

Chapter 3

Datasets

Big and high-quality datasets are very important when training a CNN-based detector. In this section, used datasets are listed and analyzed. First, the criteria for recognizing an appropriate dataset for our task are explained. Each chosen dataset is then analyzed individually. Finally, a summary of all used datasets is provided.

3.1 Dataset Criteria

Here, we briefly explain the most important criteria for selecting datasets to be used in a project like this.

Camera Angle

Since we're building a detector for a camera mounted on infrastructure, it is recommended to use datasets containing surveillance-type images. Datasets containing only images taken from, for example, a car dashboard camera, were therefore disregarded.

Classes

If we don't want to re-label the dataset manually, its classes must be mappable to *our* classes. In this work, 8 object classes are considered:

- Bicycle
- Motorcycle (any two-wheeled motorized vehicle)
- Passenger Car
- Transporter (or a van, pick-up truck, etc.)
- Bus (including a minibus)
- Truck
- Trailer
- Unknown

Many available datasets didn't annotate some of these classes or aggregated some of them into one and were therefore ignored.

Diversity of Images

For the trained detector to generalize well, it's important for a dataset to contain images with different camera angles, lighting conditions, weather, etc. 60 FPS continuous video does not bring much of an advantage.

Dataset Quality

We observed that many datasets contained incorrect annotations or classifications. It is important to check the dataset and either fix faulty annotations, ignore incorrectly annotated images or even disregard the whole dataset.

3.2 Individual Datasets

This section individually analyzes all datasets used in this work. Table 3.1 compares these datasets on a higher level for an overview. Example frames of each dataset are shown in figure **[[examples - na jednom obrázku example z každého datasetu]] [[pod každý dataset dať ref na example snímky?]] [[ref na example snímky]]**.

UA-DETRAC

The DETRAC dataset [11] provided by the University at Albany is the biggest and the most important dataset for this work, originally containing 1 274 055 annotations of 8250 vehicles in 138 252 images. The dataset is provided as frames from 100 video sequences of 25 fps with the resolution being 960×540 pixels. The length of these sequences varies, but is usually between 30 seconds and 90 seconds. The video is of a surveillance type and almost all sequences have a unique point of view, usually from a bridge. Many sequences were recorded in rain or at night and there is no lens flare from cars' headlights.

However, there are several issues with this dataset:

- Bicycles, motorcycles and a few vehicles that cannot be classified are not annotated at all
- Bounding boxes are often loose and do not fit tightly to the objects
- Vehicles near the edge of the frame, although fully visible, sometimes lack labels
- In several sequences, vehicles are tracked and annotated even on frames on which they are fully occluded (by other vehicles or infrastructure)
- Vehicle annotations are inconsistent in relation to masks, as some are labeled even when masked, while others are left unlabeled even when already fully visible outside of a mask. This is likely due to the camera being hand-held while the mask is static throughout the sequence

These problems were fixed by importing the dataset to a labeling application LabelBox [7], adjusting the masks (while also making them dynamic), annotating or masking bicycles, motorcycles and „unknown“ vehicles and finally, carefully repairing individual annotations if needed. Many sequences were hectic or were annotated so poorly that reannotating would be too time-consuming, so only 71 of 100 sequences were reannotated.

The reannotated dataset contains 733 833 annotations in 99 771 images and these classes (the dataset contains no trailers):

- Bicycle
- Motorcycle
- Car
- Bus
- Van - mapped to *transporter*
- Others - mapped to *truck*
- Unknown

Miovision Traffic Camera Dataset

The MIO-TCD (Miovision traffic camera dataset) [1] is another huge and very important dataset. Images are taken at different times of day by thousands of traffic cameras in Canada and the United States. Roughly 79% of all images are of resolution 720×480 pixels, but the image quality seems to be lower. The rest is of resolution 342×228 pixels. **[[ref na example fotku]]**

The dataset consists of two parts: *Classification dataset* and *Localization dataset*. Only the *Train* subset of the *Classification* part is used here because the *Test* subset is not annotated.

The part used contains 351 549 objects of these classes:

- Pedestrian - ignored
- Bicycle
- Motorcycle
- Car
- Pickup truck - mapped to *transporter*
- Work van - mapped to *transporter*
- Bus
- Articulated truck - mapped to *truck*bridge. Many sequences were recorded in rain or at night and there is no lens flare from cars' headlights.
- Single unit truck - mapped to *truck*
- Non-motorized vehicle - mapped to *trailer*
- Motorized vehicle - mapped to *unknown*

The processed dataset (without pedestrian annotations) contains 344 416 objects in 110 000 images.

The images are of low quality, there aren't many bicycles and motorcycles, and more pictures with different weather and lighting conditions are needed.

[[example fotka 720x480?]]

AAU RainSnow Traffic Surveillance Dataset

Another important dataset is the *AAU RainSnow* dataset [2]. The authors mounted two synchronized (one RGB and one thermal) cameras on street lamps at seven different Danish intersections to take 5-minute long videos at different lighting and weather conditions - night and day, rain and snow. They then extracted 2200 frames from the videos and annotated them on a pixel-level. Several different types of masks were also created and included in the dataset.

In our work, we only use the annotated frames from the RGB camera, containing 13 297 annotations in 2200 frames (before processing) of resolution 640 × 480 pixels. **[[ref na example fotku]]**

The dataset uses these 6 classes:

- Pedestrian
- Bicycle
- Motorbike
- Car
- Bus
- Truck

This creates a problem - vehicles of our internal class *transporter* (van) don't have their own class in the dataset, but are classified as trucks. However, the dataset is small and it is impossible to perfectly divide transporters and trucks into two classes as there are many different models between which a line cannot be drawn. Ignoring this issue should therefore not cause any problems.

Several other minor problems were found when processing this dataset:

- Frames in groups `Egensevej-1`, `Egensevej-3` and `Egensevej-5` are hardly usable because of the low-quality camera and challenging weather and lighting conditions, so they were dismissed
- Some frames had bounding boxes over the whole frame - this is most certainly an annotation error. These labels were ignored as well
- The mask for `Hadsundvej` intersection didn't fully cover the area that should be ignored. This was fixed by simply editing the mask

After processing, the dataset contains 10 545 objects in 1899 images.

Multi-View Traffic Intersection Dataset

For the MTID dataset, the authors [6] recorded one intersection from two points of view at 30 fps - one camera was mounted on existing infrastructure and one was attached to a hovering drone. The dataset contains 65 299 annotated objects in 5776 frames (equal share of frames for both cameras).

[[ref na example fotku]]

All annotated objects fall into one of four classes:

- Bicycle
- Car
- Bus
- Lorry - mapped to *truck*

This, at first, might not seem like enough, but a closer inspection of the annotated frames reveals that there are no pedestrians or motorcycles. However, similarly to the AAU dataset in section 3.2, there are transporters in the frames that are classified as trucks. Again, this issue is simply ignored.

When processing this dataset, two other problems were encountered:

- Vehicles that are not on the road are not annotated, so they have to be masked out. This is not as easy for the drone video because the camera is moving, but it is still simple enough
- Many frames of the drone footage lack some or all labels and have to be ignored. Ranges of images numbers which are ignored: $[1, 31]$, $[659, 659]$, $[1001, 1318]$ and $[3301, 3327]$

The processed dataset contains 64 979 objects in 5399 frames.

Night and Day Instance Segmented Park Dataset

Another useful dataset is the *NDISPark* [8], which contains images of parked vehicles taken by a camera mounted on infrastructure. Although the annotated part of the dataset only contains 142 frames after processing, there are 3302 objects annotated in total. This still makes it a tiny dataset, but it provides images of vehicles from many different points of view and also contains many occluded vehicles. **[[ref example fotku]]** Additionally, all frames are 2400 px in width and within 908 px and 1808 px in height.

This dataset does not contain any classifications, but luckily, it only contains cars, *transporters* and a few unattached car trailers. All *transporters* and *trailers* were manually classified.

VisDrone Dataset

The VisDrone dataset [13] is very different from all the previous datasets since it doesn't just contain traffic surveillance images. Images are taken by a camera mounted on a drone, from many different points of view. After processing, there are 47 720 annotated objects in 1610 frames.

This dataset might be a helpful addition to our datasets as it contains useful negative images (many annotated people and new points of view) and it often captures vehicles from a bid's-eye view.

Additionally, objects in this dataset are classified into 12 classes, which can be easily mapped to our 8 *internal* classes.

3.3 Dataset processing

[[v akom formáte sú datasety, do akého ich konvertujem, aké súbory vytváram a tak. Asi pokojne vcelku podrobne]]

Before training, all datasets used in this project must be converted to a unified format, object classes need to be mapped to be the same in each dataset, some datasets include images with regions shat should be masked out and certain subsets or images of some datasets need to be ignored.

For this, a python script was developed for each dataset. It first loads the annotations from the original format - COCO [[**ref?**]] for AAU RainSnow, MTID and NDISPark, XML for DETRAC and different CSV formats for MIO-TCD and VisDrone. The script then maps the classes to ones shown in [[**ref**]] and if needed, removes the ignored subsets or images before saving the labels in a COCO format. The processing script for NDISPark dataset also corrects the object classes before saving, as it was not done on the original ground-truth [[**footnote explain?**]] file (class corrections are defined in the script itself).

MMDetection's middle format was considered as a more efficient alternative to the COCO format, but the COCO format is more popular and is supported by most relevant application, while also being human-readable (MMDetection's middle format is saved as a pickle [[**footnote?**]] file), and most importantly, MMYOLO and the newest version of MMDetection at the time of writing this paper (v3.0.0) does not seem to fully support datasets in the middle format.

Several other python processing scripts were developed, to apply masks, combine all ground-truth files into one and split the combined ground-truth file into train, validation and test subsets. Additionally, scripts to review datasets manually were created - one to visualize a dataset by simply adding bounding boxes (with class labels) to the images and one converts the visualized images to video (or videos).

A few more scripts were written, of which two are worth mentioning - one for uploading the DETRAC dataset to LabelBox for reannotation and one for downloading the reannotated labels.

3.4 Summary of Datasets

In Table 3.1, we compare used datasets on a higher level and show the number of images and instances contained with additional comments. It is clear that the DETRAC section 3.2 dataset amounts for most of our data and might have been enough by itself, but to make this project as successful as possible, every available useful dataset should be used. The images from the selected datasets feature a great variety of lighting and weather conditions, points of view, object scales and other relevant factors. Additionally, in Table 3.2 we show the number of annotated object instances per class in all used datasets combined. [[**Povedať niečo k class inbalance?**]]

[[**Treba potom niekde povedať o train/val/test split. Tu?**]]

Dataset tag	# images	# instances	Comments
DETRAC	99 771	733 833	Large Continuous video High-quality camera Different lighting conditions
MIO-TCD	110 000	344 416	Large Low-quality images
AAU	1899	10 545	Small Different weather conditions
MTID	5399	64 979	Small Continuous video
NDISPark	142	3302	Small Occlusions
VisDrone	1610	47 720	Small Negative images New points of view
Total	218 821	1 204 795	-

Table 3.1: High-level comparison of used datasets

Class	# instances
Bicycle	14 036
Motorcycle	17 187
Passenger car	916 317
Transporter	123 585
Bus	64 529
Truck	38 069
Trailer	2360
Unknown	28 712
Total	1 204 795

Table 3.2: Numbers of class instances in all datasets combined

Chapter 4

Model Configuration and Training

In this chapter, an overview of configurations used to train the vehicle detectors is displayed. Configuration of an object detection model contains a set of parameters that define the model's behavior and the training, validation and testing pipelines. These parameters can have a significant impact on the model's speed and accuracy and tuning them is essential to achieve good results.

Most of the YOLOv8 parameters are left unchanged from the default configuration of YOLOv8-m[1], like:

Optimizer: Stochastic Gradient Descent with momentum 0.937 and weight decay 0.0005

Parameter scheduler: Linear `YOLOv5ParamScheduler` with learning rate factor of 0.01

To compensate for some datasets being smaller than others while being important and of a high quality, a dataset wrapper *RepeatDataset* is used, which makes the underlying dataset n-times more frequent when training. The repetition factor of each dataset is shown in Table 4.1.

Dataset name	# images	Repetition factor	# images after over-sampling
DETRAC	99 771	1	99 771
MIO-TCD	110 000	1	110 000
AAU RainSnow	1899	3	5697
MTID	5399	5	26 995
NDISPark	142	25	3550
VisDrone	1610	4	6440

Table 4.1: Repetition factors for each dataset used when training.

Configuration entries that were adjusted individually for each model can be found in Table 4.2.

4.1 Training Augmentation Pipeline

Data augmentations are very important when training an object detector, especially for detectors from the YOLO family. An augmentation in this context refers to the process of applying various transformations to an image to artificially increase the size and diversity of the training dataset, which helps prevent overfitting and improves the generalization ability of the trained model. To a convolutional neural network, even a tiny rotation, translation, image flip, noise or color distortion makes an input image appear to be something completely

[1] TODO yolov8_m_syncbn_fast_8xb16-500e_coco.py

Model	Deepen factor	Widen factor	Learning rate	Batch size	Epochs	Warmup epochs
YOLOv8 medium 640×384						
YOLOv8 nano 640×384						
YOLOv8 nano 512×288						
YOLOv8 pico 512×288						

Table 4.2: Training configurations for individual models.

different, so applying these transformations randomly to the input images is crucial to train a robust model.

[[nezabudnúť že posledných 10 epoch sa mení pipeline. Napísať to tu niekde]]

Resize

First, the image is resized to fit in the model's input size while, of course, keeping the aspect ratio unchanged.

Pad

If the aspect ratio of the input image is not the same as the model's input, extra pixels around the input image must be added to adapt to the model input's aspect ratio. The **color value** of the padded pixels is set to RGB(114, 114, 114).

Random Affine

The *YOLOv5RandomAffine* applies affine transformations to the image, while randomly selecting the values from configured ranges. Parameters:
Maximum translation ratio: 0.05
Maximum rotation degree: 5
Maximum shear degree: 3
Scaling ratio is set individually for each dataset as shown in Table 4.3.

Dataset name	Minimum scaling ratio	Maximum scaling ratio
DETRAC	0.8	1.0
MIO-TCD	1.0	1.1
AAU RainSnow	0.9	1.1
MTID	0.9	2.0
NDISPark	0.9	1.5
VisDrone	1.5	2.5

Table 4.3: Image scaling ratios in *RandomAffine* transformation for each dataset.

The **color value for padding** around the tranformed image (if needed) is set to RGB(114, 114, 114) to be the same as in the *Pad* 4.1 transformation.

Cut Out

The *CutOut* transformations randomly selects regions of the image and fills them by a single color. Again, parameters of this transformation are set individually for each dataset and are shown in Table 4.4.

Dataset name	Number of regions	Size of a single region
DETRAC	6	22×22 px
MIO-TCD	4	26×26 px
AAU RainSnow	8	10×10 px
NDISPark	12	20×20 px
MTID	12	10×10 px
VisDrone	20	8×8 px

Table 4.4: *CutOut* transformation parameters for each dataset.

The **fill color value** is again set to RGB(114, 114, 114), same as for padding in the previous transformations.

Custom Cut Out

A custom cut out transformation was developed, similar to *CutOut* used in the previous step. Here, the regions are selected within each bounding box with a certain probability, rather than being chosen randomly within the image. Also, the region size is specified as a range of areas in relation to the bounding box area - if the upper value is 10% and a bounding box area is 100 pixels, the maximum region area can be 10 pixels.

With the boolean option *random_pixels* toggled, the color of each pixel of a cut out region is generated randomly instead of filling it with a pre-defined color. However, it was found to have no effect after testing.

The **probability** of a region being dropped from each bounding box is set to 0.5 and the **region area** is set to be randomly selected from interval $[5\%, 35\%]$.

Albumentations

Albumentations[4] are a popular open-source library for data augmentation. The MMYOLO **[[cite or something?]]** library provides the option to use these transformations in the data augmentation pipeline. Settings are left unchanged from the original **[[ref or something?]]** *YOLOv8-m* configuration:
Blur probability: 0.01
Median blur probability: 0.01
Grayscale probability: 0.01
CLAHE probability: 0.01

HSV Random Augmentations

The *YOLOv5HSVRandomAug* simply adjusts the hue, saturation and value of the image randomly.

Random Flip

With **probability** of 0.5, the image is horizontally flipped using the *RandomFlip* augmentation.

Photometric Distortion

The *PhotoMetricDistortion* augmentation distorts an image sequentially, while each transformation is applied with a probability of 0.5. It modifies the brightness, contrast, converts color from BGR to HSV, modifies the saturation, hue, converts from BGR to HSV, modifies the contrast and finally, randomly swaps the color channels.

Filter Annotations

As the last step in the pipeline, *FilterAnnotations* is called to remove bounding boxes with **width or height** lower than 8×8 pixels.

Chapter 5

Model Optimization

Chapter 6

Inference and Deployment

Chapter 7

Evaluation

Chapter 8

Experiments

Chapter 9

Results

Chapter 10

Future Work

Chapter 11

Conclusion

Bibliography

[1] MIO-TCD: A New Benchmark Dataset for Vehicle Classification and Localization. *IEEE Transactions on Image Processing*. Institute of Electrical and Electronics Engineers (IEEE). oct 2018, vol. 27, no. 10, p. 5129–5141. DOI: 10.1109/tip.2018.2848705.

[2] Bahnsen, C. H. and Moeslund, T. B. Rain Removal in Traffic Surveillance: Does it Matter? *IEEE Transactions on Intelligent Transportation Systems*. Institute of Electrical and Electronics Engineers (IEEE). aug 2019, vol. 20, no. 8, p. 2802–2819. DOI: 10.1109/tits.2018.2872502.

[3] Bugdol, M., Miodonska, Z., Krecichwost, M. and KASPEREK, P. Vehicle detection system using magnetic sensors. *Transport Problems*. march 2014, vol. 9.

[4] Buslaev, A., Parinov, A., Khvedchenya, E., Iglovikov, V. I. and Kalinin, A. A. Albumentations: fast and flexible image augmentations. *ArXiv e-prints*. 2018.

[5] Fang, J., Meng, H., Zhang, H. and Wang, X. A Low-cost Vehicle Detection and Classification System based on Unmodulated Continuous-wave Radar. In: *2007 IEEE Intelligent Transportation Systems Conference*. 2007, p. 715–720. DOI: 10.1109/ITSC.2007.4357739.

[6] Jensen, M. B., Mogelmose, A. and Moeslund, T. B. Presenting the Multi-view Traffic Intersection Dataset (MTID): A Detailed Traffic-Surveillance Dataset. In: *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, Sep 2020. DOI: 10.1109/itsc45102.2020.9294694.

[7] Labelbox. *Labelbox* [online]. 2022. Available at: https://labelbox.com.

[8] Luca, C., Carlos, S., Paulo, C. J., Claudio, G. and Giuseppe, A. *Night and Day Instance Segmented Park (NDISPark) Dataset: a Collection of Images taken by Day and by Night for Vehicle Detection, Segmentation and Counting in Parking Areas*. Zenodo, 2022. DOI: 10.5281/ZENODO.6560822.

[9] Murali, S. and V K, G. Shadow Detection and Removal from a Single Image Using LAB Color Space. *Cybernetics and Information Technologies*. march 2013, vol. 13. DOI: 10.2478/cait-2013-0009.

[10] Stiawan, R., Kusumadjati, A., Aminah, N. S., Djamal, M. and Viridi, S. An Ultrasonic Sensor System for Vehicle Detection Application. *Journal of Physics: Conference Series*. IOP Publishing. apr 2019, vol. 1204, no. 1, p. 012017. DOI: 10.1088/1742-6596/1204/1/012017. Available at: https://dx.doi.org/10.1088/1742-6596/1204/1/012017.

[11] Wen, L., Du, D., Cai, Z., Lei, Z., Chang, M.-C. et al. UA-DETRAC: A New Benchmark and Protocol for Multi-Object Detection and Tracking. arXiv. november 2015. DOI: 10.48550/ARXIV.1511.04136.

[12] Wu, K., Xu, T., Zhang, H. and Song, J. Overview of video-based vehicle detection technologies. *ICCSE 2011 - 6th International Conference on Computer Science and Education, Final Program and Proceedings*. august 2011. DOI: 10.1109/ICCSE.2011.6028764.

[13] Zhu, P., Wen, L., Du, D., Bian, X., Fan, H. et al. Detection and Tracking Meet Drones Challenge. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Institute of Electrical and Electronics Engineers (IEEE). nov 2022, vol. 44, no. 11, p. 7380–7399. DOI: 10.1109/tpami.2021.3119563.