**Faculty of Science and Technology**

# BACHELOR'S THESIS

| Study program/ Specialization: Computer Science - Bachelor's degree programme in computer science | Spring semester, 2022<br>**Open** / ~~Restricted access~~ |
|---|---|
| Writer: Aleksander Vedvik | ………………………………………… (Writer's signature) |
| Faculty supervisor: Naeem Khademi<br>External supervisor(s): | |
| Thesis title: Pre-Study of Deep Learning Based Automatic Incident Detection in Road Tunnels using Video Surveillance Cameras | |
| Credits (ECTS): 20 | |
| Key words: deep learning, deep neural networks, convolutional neural networks, automatic incident detection, road tunnels, object detection, object tracking, image enhancement, CCTV cameras | Pages: ……59……… + enclosure: …40……<br>Stavanger, 15.05.2022<br>Date/year |

Front page for master thesis
Faculty of Science and Technology
Decision made by the Dean October 30th 2009

## Abstract

Vehicular incidents in road tunnels can have severe consequences. Automatic incident detection (AID) systems can help reduce the severity and ensure a quick response from road authorities. Hence AID is a highly requested, but also a difficult task.

The focus of this thesis has been to provide recommendations and reduce false alarms related to AID systems using video surveillance cameras and deep neural networks. These systems mainly comprise three steps: object detection, object tracking, and incident evaluation. Hence, several state-of-the-art object detection algorithms have been evaluated, together with a state-of-the-art and a simple object tracking algorithm. Lastly, a simple incident evaluation algorithm was employed to test the whole AID pipeline.

Due to the complexity of AID systems, only the incident classes "stopped vehicle", "pedestrian" and "wrong-way drivers" were evaluated. There are not many publicly available annotated datasets containing incidents, and thus videos obtained from the internet have been annotated and used in testing.

Lastly, several image enhancement algorithms were tested and their utility assessed. Enhancing the images before the object detection step can help combat some of the challenges related to automatic incident detection in road tunnels.

# Contents

**1 Introduction** **4**
- 1.1 Background . . . 4
- 1.2 Automatic Incident Detection (AID) . . . 4
  - 1.2.1 Comparative . . . 5
  - 1.2.2 Statistical . . . 5
  - 1.2.3 Traffic-model-based . . . 6
  - 1.2.4 Artificial intelligence based . . . 6
  - 1.2.5 Mixed models . . . 7
- 1.3 Challenges of AID in Road Tunnels . . . 8
- 1.4 Objectives . . . 8

**2 Related work** **10**

**3 Theory** **13**
- 3.1 Image enhancement methods . . . 13
  - 3.1.1 Gray transformation . . . 13
  - 3.1.2 Histogram equalization . . . 14
  - 3.1.3 Retinex . . . 14
- 3.2 Artificial Neural Networks . . . 15
- 3.3 Convolutional Neural Networks . . . 16
- 3.4 Transfer learning . . . 19
- 3.5 Object detection . . . 20
  - 3.5.1 Faster RCNN . . . 20
  - 3.5.2 Single Shot Detector (SSD) . . . 22
  - 3.5.3 EfficientDet . . . 22
  - 3.5.4 YOLOv5 . . . 23
- 3.6 Object tracking . . . 23
  - 3.6.1 SORT . . . 24
  - 3.6.2 Deep SORT . . . 25

**4 Approach** **27**
4.1 Tools . . . 27
4.2 Limitations and simplifications . . . 28
4.2.1 Datasets . . . 28
4.2.2 GPU resources . . . 28
4.2.3 Code availability . . . 29
4.3 Datasets . . . 29
4.4 Preparation of data . . . 30
4.5 Image enhancement . . . 31
4.6 Object Detection . . . 31
4.6.1 TensorFlowAPI . . . 31
4.6.2 YOLOv5 . . . 34
4.7 Tracking . . . 34
4.7.1 Simple tracking . . . 34
4.7.2 Deep Sort . . . 35
4.8 Incident evaluation . . . 35
4.9 Performance evaluation . . . 36

**5 Discussion** **40**
5.1 Image Enhancement . . . 40
5.2 Detection . . . 44
5.3 Tracking . . . 47
5.4 Incident evaluation . . . 48
5.5 Performance . . . 49
5.6 Model- and Analytical Improvements . . . 50
5.7 Further work . . . 51

**6 Conclusion** **52**

**References** **54**

**A Github repository** **63**

**B Code excerpts** **64**

# Chapter 1

# Introduction

## 1.1 Background

Norwegian road authorities set a 0-goal regarding incidents in road tunnels in 2017 [1]. Incidents in road tunnels can have severe consequences due to confined spaces and thus lead to larger accidents. Hence, to achieve the 0-goal, it is important to have robust and high-level technology systems in the tunnels [2].

According to "Tunnelsikkerhetforskriften" §8 appendix 1, tunnels of length 3000m or longer with more than 2000 vehicles per lane may install a control center. All tunnels with a control center may also be equipped with CCTVs and an automatic incident detection (AID) system [3].

A detected incident from the AID system is sent to operators at the road authorities. Camera operators are given the responsibility to monitor several tunnels containing many cameras. Hence, a well-performing AID system is paramount to ensure effective monitoring and a quick response when incidents occur. [4, p. 5]

## 1.2 Automatic Incident Detection (AID)

Automatic incident detection (AID) systems are systems that incur alarms and notify camera operators when a traffic incident has occurred. Many automatic incident detection systems use traffic data, such as flow, congestion, vehicle speed, etc, to determine if there has been an incident. These traffic parameters can be obtained by using inductive loops, radars, CCTV, ultrasound, Bluetooth, etc.

A well-performing AID system should be able to detect most of the incidents that occur in road tunnels. Incidents in road tunnels can mainly be

divided into these classes:

- Stopped vehicles
- Pedestrians
- Queue
- Vehicles moving in a wrong direction
- Animals
- Stationary objects
- Smoke or fire

Furthermore, AID systems are generally divided into five categories: comparative, statistical, traffic-model-based, artificial intelligence-based, and mixed models. [5, p. 1]

### 1.2.1 Comparative

Comparative incident detection algorithms use several thresholds based on traffic occupancy to determine if there is an incident. [6, p. 1] Algorithms in this category often use the data from an upstream and a downstream loop detector. Popular algorithms in this category are the California algorithms and filtering algorithms. The California Algorithm uses three tests on the measured occupancy from two detectors to determine if there is an incident. [7, p. 10] An increase in upstream occupancy and a decrease in downstream occupancy will often signify that an incident has occurred. When these values exceed predetermined thresholds an incident is declared. [8, p. 1-2]

### 1.2.2 Statistical

Algorithms in this category perform short-term predictions of traffic variables, such as volume, speed, occupancy, etc. The implementation can be divided into two steps, where it first predicts values of traffic-flow parameters based on historical data and then compares the predicted values with the current data. [8, p. 1-2] An incident is identified if the observed values deviate enough from the predicted values. The standard normal deviate model is one of the first algorithms in this category. Based on the mean and standard deviation of the input data, the standard normal deviate algorithm calculates standardized values for the traffic. When traffic values deviate more

than a certain threshold an incident alarm is declared. [5, p. 1] Other major algorithms in this category include time-series and filtering algorithms. [8, p. 1-2]

#### Standard Normal Deviate Algorithm

The Standard Normal Deviate (SND) Algorithm has proven to be effective in AID systems, due to it having good transferability and not being too difficult to calibrate. The algorithm works by indicating a traffic incident if there is a sudden change in a particular traffic parameter. This particular traffic parameter, e.g. traffic occupancy, is given an SND value, and a traffic incident is identified when this value exceeds a predetermined threshold. The method comprises two parts: First, a time interval with preliminary detected incidents, and second, a time interval as a persistence test to confirm the detected incidents. [9, p. 841]

Newer algorithms have been built upon the original SND algorithm, such as the one used in [9]. This new algorithm has added two extensions to the previous SND algorithm. The first extension focuses on improving the reliability of detection results by adopting the weighting method. Traffic parameter values vary in different time intervals because they are derived at different data points, which in turn makes the original SND algorithm inaccurate and thus produces false alarms. The second extension addresses this problem by restricting the variation of input data within sampling periods. [9, p. 841-842]

### 1.2.3 Traffic-model-based

Traffic-model-based algorithms model the traffic flow by using the relationship between volume and occupancy to classify the upstream and downstream traffic measures into traffic states. (e.g. uncongested, congested, and incident). [8, p. 2], [6, p. 1]. The McMaster Algorithm is in this category.

It is difficult to describe traffic flow due to it being nonlinear, stochastic, time-variant, and a complicated dynamic system. Hence it is not common to use model-based detection algorithms to detect incidents. [8, p. 2]

### 1.2.4 Artificial intelligence based

Artificial intelligence-based algorithms use historic traffic data of normal and incident conditions to classify traffic patterns. Video-based incident detection systems using artificial intelligence have become increasingly popular in recent years [6, p. 1].

#### Fuzzy Algorithms

Fuzzy algorithms are based on fuzzy logic, fuzzy boundaries, and the change of occupancy or speed-density of two adjacent detector stations. These algorithms are effective when traffic data is difficult to collect or when there is not enough traffic data. This is because they have high robustness and can overcome the boundary condition problem that conventional threshold-based methods suffer from. [8, p. 2]

#### Neural-Network Algorithms

Neural network algorithms are trained on large datasets to recognize patterns in the traffic flow and identify incident or incident-free states. These algorithms are easier to use and are better suited for real-time implementation compared to model-based detection algorithms. However, there are a few drawbacks [8, p. 2]:

- Performance: The rate of convergence can be slow.
- Black-box approach: It could be difficult to understand the meanings of the neural network operations.
- Large datasets: Neural networks require large datasets that are wide enough to accurately identify incidents.

#### Image-Based Processing Algorithms

Algorithms using computer vision and image-based technology to detect incidents based on information of traffic parameters from video footage are categorized into image-based processing algorithms. Many tunnels are already equipped with video traffic surveillance, hence making this approach relatively cost-effective and available. Generally, these algorithms have a high detection rate, low false alarm rate, and a short time to detection. [8, p. 2]

### 1.2.5 Mixed models

Mixed models algorithms combine different types of methods. A popular algorithm in this category is the Minnesota algorithm, which is a combination of statistical and comparative algorithms. [5, p. 1]

## 1.3 Challenges of AID in Road Tunnels

There are several challenges related to AID systems in tunnels. Lighting conditions in the tunnel can greatly affect the detection accuracy and false alarm rate. Also, poor lighting conditions can introduce shadows and noise in images, due to ISO compensation, which can be falsely identified as vehicles or pedestrians. Furthermore, the technical specifications, such as resolution, frame rate, and ISO, and the physical conditions of the cameras can also have a great effect on the accuracy of an AID system. Dirty camera lenses, low-resolution images, and low frame rates all contribute to decreasing the detection accuracy. [4, p. 8-9]

Many AID systems are dependent on a manual configuration in order to function properly. Defining upstream and downstream directions and detection zones can be a time expensive, and therefore costly, job. Queues and rush traffic are known to produce many false alarms in a short amount of time, and thus filtering techniques have been used in order to reduce the false alarms. However, this also decreases the incident detection accuracy. This also makes detecting stopped vehicles in queues a challenge. [4, p. 11]

Another challenge related to AID systems is the weather conditions, and then especially near the entrance and exit of the tunnels. Weather conditions such as rain and snow can introduce false detections and alarms related to glare, fog, and water on the walls of the tunnel. [10]

## 1.4 Objectives

Automatic incident detection systems have been consistently developed and machine learning and computer vision have proven their success in automatic incident detection tasks until recently. Much of the research on AID systems has been focused on analyzing traffic parameters extracted from e.g. inductive loops, thus making vision-based AID systems not fully explored and utilized. Also, a lot of the research has been focused on highways and intersections, rather than road tunnels.

Vision-based detection of road accidents using traffic surveillance cameras is a highly desirable, but challenging task [11]. The computer-vision and neural network-based AID systems currently in use mainly comprise three steps: object detection, object tracking, and then incident evaluation. These systems are generally good at detecting incidents, but the main problem is the high rate of false alarms. This study will therefore be focused on improving the systems by providing recommendations on how to reduce false alarms, and increase detection and performance. Several image enhancement

methods and object detection algorithms together with a state-of-the-art object tracking algorithm will be analyzed, where pros and cons will be stated and their utility assessed.

# Chapter 2

# Related work

There are many different approaches and methods for automatic incident detection systems. Data needed for AID systems are usually collected from static sensors, dynamic sensors, and traffic cameras. Static sensors mainly include inductive loops, while dynamic sensors are usually installed on probe vehicles, which in return can provide a continuous stream of measurements of the streets they pass by. [12, p. 1-2] The most common incident detection algorithms have primarily been developed to use data from inductive loops [5, p. 1]. Furthermore, artificial neural networks have proven to yield good performance and result in the field of incident detection. Newer and better methods are constantly being tested and published.

Automatic incident detection systems can also be treated as an anomaly detection problem. Using video-surveillance cameras for anomaly detection, the different approaches can be categorized into six categories: Model-based, proximity-based, classification-based, prediction-based, reconstruction-based, and other approaches [13, p. 5]. Gaussian mixture models, regression models, Bayesian networks, and deep neural networks are classified as model-based approaches [13, p. 4].

In [5] they are using link travel speed data collected from the Korean traffic information system, which is then fed to a fully connected neural network to determine if there has been an accident. The authors of the paper determined that an incident will likely create congestion in the upstream station and reduce flow in the downstream station, leading to a high-velocity difference between the two stations. They achieved good results on their new method compared to the popular California 7 algorithm, and the method is also capable of providing real-time crash warnings to the operators [5, p. 5].

The method in [12] is based on an algorithm comprising a time-series analysis (TSA) of the traffic data collected from two adjacent detectors, where features describing the instance are generated and fed to a machine-learning

classifier [12, p. 2]. The algorithm is made up of three parts: data preprocessing, normal traffic forecasting, and incident classification. The preprocessing part filters out noise and fills out missing data, TSA is used in the traffic forecasting part, and a support vector machine model with a radial basis function is used in the incident classification stage. [12, p. 4] They achieved better mean time to detection and detection rates with similar false alarm rates as the Minnesota and CODE algorithms. [12, p. 9]

The authors of [14] proposed a vision-based framework for detecting various crash types in a mixed traffic flow environment considering low-visibility conditions. First, they used a Retinex image enhancer algorithm to improve the images, and then fed them to the object detection algorithm. YOLOv3 with Darknet-53 as the feature extractor was used for object detection. Lastly, a decision tree was used to determine if there was an incident based on the features obtained from the object detection stage. The dataset used was CCTV videos from online which the authors collected and annotated themselves. They achieved a detection rate of 92.5% and a false alarm rate of 7.5% with relatively low computational requirements [14].

A CCTV-based method using Yolov3 and an updated linear quadratic equation was used to detect and track vehicles in [15]. The authors created a unique "entry-exit" method to identify wrong-way driving vehicles, which performed well under different lighting conditions. The method works by defining entry and exit lines in the video frame. They achieved 91.98% accuracy on their self-collected dataset, while also working in real-time at about 30 frames per second. [15]

Nvidia's DeepStream model TrafficCamNet and YOLOv3 with SORT was used to detect wrong-way drivers in [16]. The system developed in the paper is also able to self-calibrate, meaning it can identify different bands on a roadway and recognize the behavior of drivers in each band. This was done by dividing each video frame into a 50 x 50 px grid and fitting a Gaussian mixture model (GMM) to the location of each grid cell and the velocity of the vehicles that had passed through that cell [16, p. 5]. The GMM could then attribute a label to each cell showing which side of the road the cell belonged. Afterward, a trained support vector machine was used to predict which side of the road a particular cell belonged, and thus filter out the noise produced from the GMM [16, p. 6]. The DeepStream model had good performance in detecting and tracking vehicles while also having low inference cost. The proposed method is highly scalable due to the self-calibration and low cost, but it suffers from a weak performance in different weather conditions and low-light conditions [16, p. 12].

By using a semi-supervised algorithm together with object detection and tracking, the authors of [17] aimed to detect incidents by classifying the

trajectories of the vehicles. They used Yolov3 and SORT for the object detection and tracking part, and Contrastive Pessimistic Likelihood Estimation (CLPE) based on Maximum Likelihood Estimation for the semi-supervised trajectory classification. [17, p. 1841] The method was able to outperform traditional semi-supervised techniques, such as Self Learning and Label Spreading, and also its supervised counterpart by a significant margin. [17, p. 1844]

The method used in [18] is based on an adaptive boosting classifier trained to detect outliers in traffic data, and a support vector machine (SVM) based method to identify the categories of the outliers. Based on their findings, SVMs can generate better AID performance than neural networks and are hence used in the proposed method. The method uses data acquired from traffic surveillance systems. Spatio-temporal signals are extracted from the video footage and smoothed with a local averaging filter to decrease training complexity. The adaptive boosting SVM then detects outliers from the ST signals. The proposed method achieves high classification accuracy, and it is also expected that slight traffic jams would be discerned with higher accuracy. [18]

In conclusion, there have been advancements in the deep learning and computer-vision-based approaches for AID systems. The YOLOv3 algorithm is popular and yields relatively good results, with Nvidia's DeepStream model TraffiCamNet as a good alternative. All methods have in common that they perform worse in terms of accuracy and false alarm rate when there are low-light or different weather conditions. Different machine learning methods have been utilized in the literature to determine incidents, based on both videos and extracted traffic parameters from road detectors. Some of these methods are only concerned with a few incident classes, while others with all incidents. Similarly, I will also only look into a few incident classes, but I will explore and compare popular image enhancement methods, object detection methods, and object tracking methods, and thus provide recommendations on how to improve detection and reduce false alarms.

# Chapter 3

# Theory

## 3.1 Image enhancement methods

### 3.1.1 Gray transformation

A gray transformation is an image enhancement algorithm where the gray values of single pixels are transformed into other gray values by using a mathematical function. This is usually called a mapping-based approach, and it enhances the image by modifying the distribution and dynamic range of the gray values of the pixels. [19, p. 3]

Gray transformations are generally divided into two groups: linear transformation and nonlinear transformation. A linear transformation, also known as linear stretching, transforms the dynamic range in an image resulting in enhanced brightness and contrast. This is done by adjusting the values of the coefficients of the linear transformation formula given in equation 3.1. [19, p. 3]

$$g(x, y) = C \cdot f(x, y) + R \tag{3.1}$$

On the other hand, a nonlinear transformation uses a nonlinear function, such as e.g. logarithmic or gamma functions, to transform the gray values of an image. Logarithmic functions are suitable for very dark images because they can stretch the lower gray values while compressing the dynamic range of the pixels with higher gray values. Gamma functions rely on a single gamma value, which can be adjusted to selectively stretch different gray regions of an image. [19, p. 4]

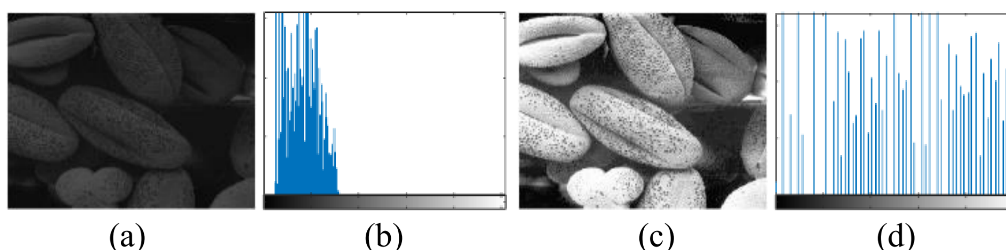


Figure 3.1: Examples of histograms [19, p. 5]

### 3.1.2 Histogram equalization

Histograms are a plot of pixel values ranging from 0 to 255 on the x-axis and the corresponding number of pixels in the image on the y-axis. The histogram can give us information about the contrast, brightness, and intensity distribution of the image. [20] Figure 3.1 shows two pictures with their corresponding histograms. If the pixel values are evenly distributed in the histogram, then the image shows high contrast and a wide dynamic range. This is the basis for the Histogram Equalization algorithm, which uses the cumulative distribution function to adjust the output gray levels to have a probability density function that corresponds to a uniform distribution. Hence details in the dark parts of the image can be made more visible. [19, p. 5]

### 3.1.3 Retinex

The Retinex algorithm is based on Retinex theory and the illumination-reflection model shown in figure 3.2. By removing the effects of the illuminating light from the image, the Retinex theory can determine the reflective nature of an object. According to the model in figure 3.2, an image can be expressed as the product of a reflection component and an illumination component as stated in equation 3.2. [19, p. 8]

$$I(x, y) = R(x, y) \cdot L(x, y) \tag{3.2}$$

In equation 3.2, R(x, y) is the reflection component that determines the inherent nature of the image. L(x, y) is the illumination component and it determines the dynamic range of the image. Lastly, I(x, y) is the received image. The reflection component can be separated from the total amount of light, and the influence of the illumination component on the image can be reduced if L(x, y) can be estimated from I(x, y). As a result, this will enhance the image. Figure 3.3 shows the general process of the Retinex algorithm,

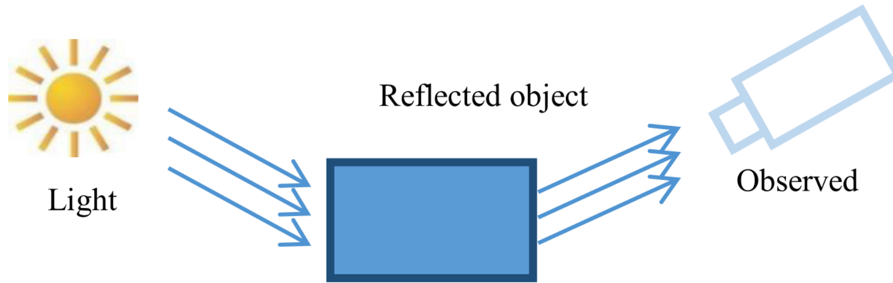


Figure 3.2: Light reflection model [19, p. 8]

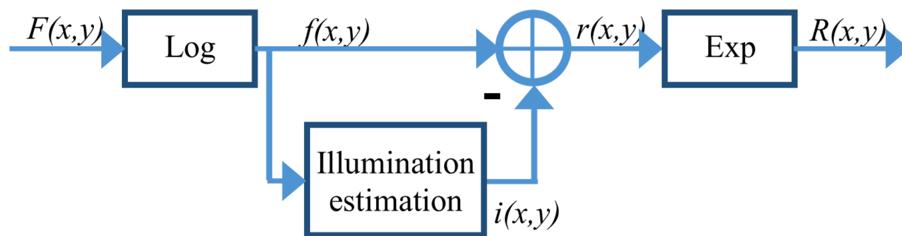


Figure 3.3: Retinex algorithm process [19, p. 8]

where Log denotes the logarithmic operation and Exp the exponential operation. The Retinex algorithm can enhance images by featuring sharpening capability, color constancy, large dynamic range compression, and high color fidelity. [19, p. 8]

## 3.2 Artificial Neural Networks

Neural networks are learning machines, comprising a large number of neurons, which are connected in a layered fashion. The human brain has been a significant inspiration to the development of neural networks, and thus these networks are built and behave similarly to the human brain. Neurons are the basic building blocks of the human brain and they are connected via synapses. [21, p. 902] This idea was borrowed in the development of neural networks, which resulted in connected nodes and synaptic weights representing the neurons and their function. Learning is thus achieved by adjusting these synaptic weights to minimize a preselected cost function. [21, p. 903]

A feed-forward neural network consists of several layers of neurons, and each neuron is determined by the corresponding set of synaptic weights and its bias term. From this point of view, a neural network realizes a nonlinear parametric function, $\hat{y} = f_\theta(x)$, where $\theta$ stands for all the weights/biases

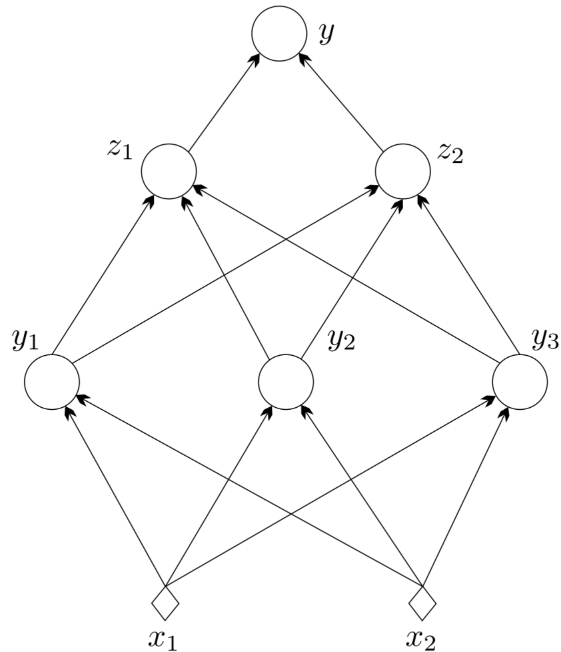


Figure 3.4: Example of a three layer feed forward network. [21, p. 911]

present in the network. All that is needed is a set of training samples, a loss function, $L(y, \hat{y})$, and an iterative scheme, such as gradient descent, to perform the optimization of the associated cost function: [21, p. 913]

$$J(\theta) = \Sigma_{n=1}^{N} L(y_n, f_\theta(x_n)) \tag{3.3}$$

Figure 3.4 shows the basic structure of the feed-forward neural network. It is a fully connected network, which means every node in one layer is directly connected to every node in the next layer. Inner products are calculated for each node, which can be defined as:

$$z_r^j = \theta_j^{rT} y^{r-1} \tag{3.4}$$

where j denotes the node in layer r, and $\theta$ denotes the vector containing the synaptic weights. T signifies that $\theta$ is a transposed vector. [21, p. 912]

## 3.3 Convolutional Neural Networks

Convolutional neural networks (CNN) are another type of neural network where the network is trained to learn the features from the data together with the parameters of the neural networks. Whereas in a fully connected

neural network, feature vectors used in training are generated from the raw data to compact the relevant information. The reason it is called a convolutional neural network is that the network performs convolutions instead of inner products. [21, p. 956] At the heart of the CNN architecture lies the weight-sharing rationale, where a set of parameters are shared among several connections. [21, p. 913] This allows the network to easily scale to images of different dimensions. [21, p. 976] Interestingly, there is also strong evidence from the visual neuroscience field that similar computations to that of a CNN are performed in the human brain. [21, p. 961]

The need for convolutions is especially evident when images are used as input in the neural network. Images comprise an array of pixel values which must be converted into a feature vector when used as input in a fully connected neural network. Every pixel in the image will then produce an input node in the network for each color channel. Since every node in one layer is connected to every node in the next layer in a fully connected network, even a small image will produce a huge amount of parameters. This results in increased computational load issues and challenges related to generalization performance. A network with large amounts of parameters is vulnerable to overfitting and would require a very large training dataset. Instead, convolutions are used in a CNN, effectively reducing the number of parameters needed and also preserving any correlations between different parts of the image. [21, p. 956]

Convolutional neural networks introduces a new method which includes three steps [21, p. 957]:

- The convolution step
- The nonlinearity step
- The pooling step

The convolution step in a neural network leverages filters or kernel matrices to produce feature maps. The nodes in a fully neural network are replaced by filters in a CNN. A filter matrix is an NxN matrix that is used to perform dot products on an input array. Figure 3.5 shows an example where a 2x2 filter (B) is used to perform a convolution on a 3x3 input array (A). The convolution step involves sliding the filter matrix across the input matrix and performing dot products, resulting in this case in a smaller 2x2 result matrix which is called a "feature map". The different dot products performed in the convolution are signified with different colors and line types in the figure. [21, p. 959] After each convolution a bias term is added to each pixel in the feature map. This term is computed under training and is in line

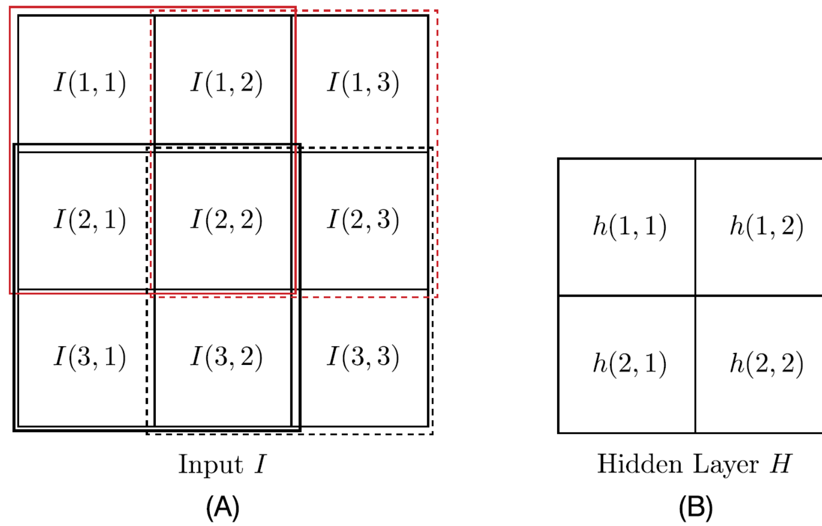


Figure 3.5: Example of a convolution on a 3x3 input matrix (A) with a 2x2 filter matrix (B) [21, p. 957]

with the weight-sharing rationale. When performing convolution the border pixels of the image matrix may contribute less to the output compared to the pixels in the center, and thus it is common to increase the size of the input matrix. The operation used in this process is called zero padding and involves padding the input matrix with zeroes around the border pixels. [21, p. 962] Figure 3.6 shows an example of zero padding a 5x5 input matrix.

The translation invariance is a significant characteristic of a CNN. A small change in e.g. position of an object in the input image would result in the same output, but with a contribution of the object moved in the same

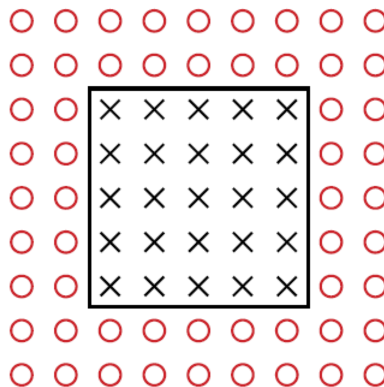

Figure 3.6: Example of zero padding a 5x5 input matrix [21, p. 962]

| 2 | 3 | **7** | 1 | 4 | 5 |
|---|---|---|---|---|---|
| 4 | **5** | 0 | 6 | **7** | 1 |
| **6** | 2 | 1 | 3 | 2 | 3 |
| 4 | 5 | 6 | **8** | 4 | **5** |
| 1 | 3 | 2 | 1 | 2 | 4 |
| **4** | 2 | 1 | **8** | **6** | 3 |

(A)

| 5 | 7 | 7 |
|---|---|---|
| 6 | 8 | 5 |
| 4 | 8 | 6 |

(B)

Figure 3.7: Example of a pooling step. The original matrix (A) is a 6x6 matrix. A window of size 2x2 and stride 2 is chosen when performing max pooling. The resulting matrix is a 3x3 matrix (B). [21, p. 964]

direction in the output. [21, p. 961]

After the convolution step, the nonlinearity or activation function is applied to every feature map array. The activation function is normally rectified linear activation function, ReLU, but other functions can be used as well. [21, p. 963]

The pooling step reduces the dimensionality of each feature map. This is done by defining a window and a stride value and then sliding the window over the feature maps. Figure 3.7 shows an example of performing max pooling on a 6x6 matrix with a 2x2 window and stride 2. The window is applied to each element of the original matrix and moved by 2 elements. Max pooling is a technique where the largest value in the window area is kept, but average pooling is also a popular alternative. When reducing the size, the information loss must be as small as possible. Pooling can be considered a special type of filtering, that acts as a summary of the information in the pooling area. This also makes the representation approximately invariant to small translations of the input. [21, p. 964]

## 3.4 Transfer learning

Transfer learning is a concept where a new model is leveraging an already trained model to improve accuracy, reduce training time and minimize the amount of data required to train the model. Deep neural networks require large amounts of labeled data to perform well and have a good generalization.

However, it is not always possible to collect such large labeled datasets, due to many being proprietary or expensive to collect, or they can be restricted due to laws and regulations such as GDPR. Limited available data, e.g. X-ray imaging of cancerous tumors, is also one of the factors that make datasets difficult to obtain. Transfer learning was thus developed to reduce these problems, and it is greatly inspired by how humans learn new things. The human brain is capable of transferring knowledge from one situation to another similar situation and hence does not need to learn new things from scratch every time it is faced with a new situation. [21, p. 1013]

The way transfer learning works is by freezing the weights of an already trained CNN model and adjusting the weights in the fully connected layer. The pre-trained model has learned to detect different shapes and features in an image, which it presents as input to the fully connected part of the network. By fixing the parameters and filters related to the feature detection and only updating the weights in the fully connected layer when training the new model, it is possible to transfer the knowledge from the pre-trained model to the new model. This way, it is possible to further develop models that are trained on large datasets to accomplish greater accuracy in more specific tasks and situations. [21, p. 1014]

## 3.5 Object detection

Support vector machines were used in most of the object detection tasks before 2012. This was when CNNs were reintroduced and proved their success in object detection tasks. [22, p. 1] Object detection is based on image classification, where image classification in computer-vision tasks predicts the class of an object in an image, while object detection also predicts the bounding boxes of the object representing its location in the image. Figure 3.8 shows an example of a swimming pool, where the left represents image classification and the right picture represents object detection with a bounding box drawn on the picture. [23]

### 3.5.1 Faster RCNN

Regions with CNN features (R-CNN) are pioneering approaches that use deep models for object detection. R-CNN models work by first selecting several proposed regions (about two thousand) from an image, such as in step 2 in figure 3.9, and then labeling their categories and bounding boxes. These labels are created based on predefined classes. Then, a convolutional neural network is used to perform forward computation to extract features

Figure 3.8: Object classification vs object detection. [23]

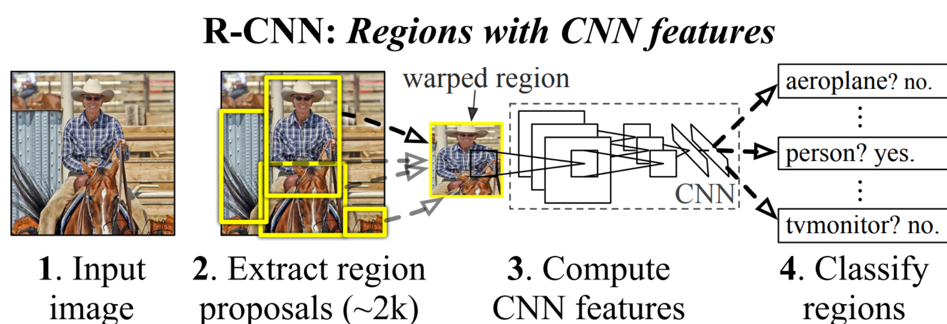


Figure 3.9: Regions with CNN features [22, p. 1]

from each proposed area (step 3 in figure 3.9). Training time is significant because it classifies and creates bounding boxes individually and a neural network is applied to one region at a time. [24]

Faster RCNN was developed to increase the speed and reduce the training time of the regions with CNN features algorithm. Instead of running a neural network on each region, the Faster RCNN algorithm only runs the neural network once on the whole image. It uses Region Proposal Network (RPN) for generating Regions of Interest. RPN takes image feature maps as an input and generates a set of object proposals, each with an objectness score as output [25].

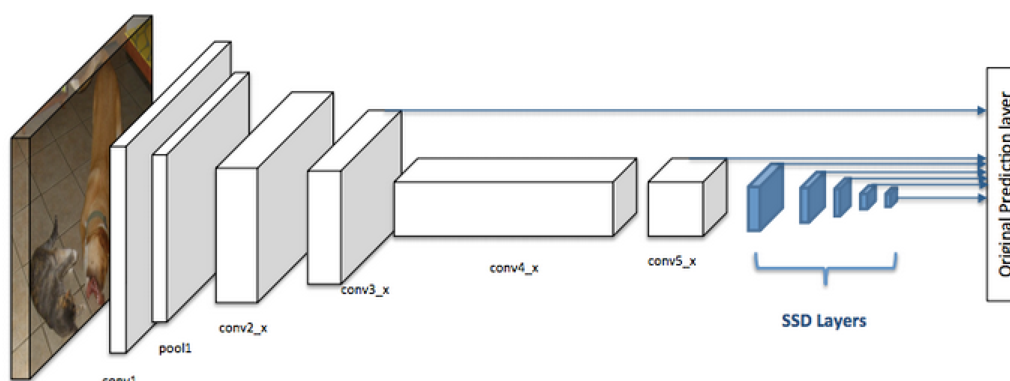


Figure 3.10: Architecture of a convolutional neural network with a SSD detector. [23]

### 3.5.2 Single Shot Detector (SSD)

The single-shot detector (SSD) is a popular one-stage detector that can predict multiple classes. It detects objects using a single deep neural network by discretizing the output space of bounding boxes into a set of default boxes over different aspect ratios and scales per feature map location. The scores are generated for the presence of each object category in each default box and then the object detector adjusts the box to better fit the object shape. Additionally, the network combines predictions from multiple feature maps with different resolutions to handle objects of different sizes. The model architecture is shown in figure 3.10. [24]

### 3.5.3 EfficientDet

The Google Brain team released their convolutional neural network called EfficientNet, which forms the backbone of the EfficientDet architecture. EfficientNet set out to define an automatic procedure for scaling CNN model architectures and optimize downstream performance given free range over depth, width, and resolution while staying within the constraints of target memory and target FLOPs.

The neural architecture search optimizes for accuracy, given a certain number of FLOPS, and results in the creation of a baseline CNN called EfficientNet-B0. Using the scaling search, EfficientNet-B0 is scaled up to EfficientNet-B1. The scaling function from EfficientNet-B0 to EfficientNet-B1 is saved and applied to subsequent scalings through EfficientNet-B7 because additional search becomes prohibitively expensive. [26]

The EfficientDet Model is evaluated on the COCO dataset, which is con-

sidered to be the general-purpose challenge for object detection. EfficientDet outperforms many of the previous object detection models under several constraints. [26]

### 3.5.4 YOLOv5

You Only Look Once (YOLO) is a single-stage object detection algorithm. It is popular for its speed and accuracy, and it detects objects in a single forward propagation through a neural network. This makes it suitable for real-time applications. The YOLOv5 implementation is different from the previous releases. Instead of utilizing Darknet, YOLOv5 uses PyTorch and CSPDarknet53 as the backbone. This backbone solves the repetitive gradient information in large backbones and integrates gradient change into a feature map that reduces the inference speed, increases accuracy, and reduces the model size by decreasing the information flow. YOLOv5 uses a path aggregation network as the neck to boost the information flow, which improves the propagation of low-level features in the model. It also improves the localization in lower levels, which enhances the localization accuracy of the object. The head in YOLOv5 is the same as YOLOv4 and YOLOv3 which generated three different output of feature maps to achieve multi-scale prediction. The Focus layer in YOLOv5 evolved from the YOLOv3 structure by replacing the first three layers with a single layer in YOLOv5. The YOLOv5 architecture is shown in figure 3.11. [27, p. 7-8]

## 3.6 Object tracking

Object tracking algorithms can be categorized into these bullet points: [28, p. 2]

- Single-object vs multiple-object trackers
- Generative vs discriminative
- Context-aware vs non-aware
- Online vs offline learning algorithms

Some of the most common challenges related to tracking problems include occlusion, clutter, variation illumination, scale variations, low-resolution targets, target deformation, target re-identification, fast motion, motion blur, in-plane and out-of-plane rotations, and target tracking in the presence of noise. [28, p. 1] The SORT algorithms are online multi-object trackers that

Figure 3.11: The YOLOv5 architecthure. [27, p. 8]

leverage Kalman filtering and the Hungarian algorithm to combat these challenges.

Kalman filtering is widely used in engineering problems involving temporal or time series prediction. The core idea behind the algorithm is to use the available detections and previous detection to arrive at the best guess of the current state while keeping the possibility of error in the process. [29] The Hungarian algorithm is a combinatorial optimization algorithm that can be used to find maximum-weight sharing matchings in bipartite graphs, also called the assignment problem [29].

### 3.6.1 SORT

Simple Online Realtime Tracking is an online, multiple object tracker with strong emphasis placed on efficiency for facilitating real-time tracking. Some of the state-of-the-art tracking algorithms are relatively slow and thus are not suitable for real-time tracking. [30, p. 1]

The SORT algorithm focuses on efficient and reliable handling of the common frame-to-frame associations. Rather than performing the object detection task itself, the SORT algorithm works together with separate object detection algorithms such as YOLOv5 or Faster RCNN. Additionally, SORT employs Kalman filtering and the Hungarian algorithm to handle the motion

prediction and data association components of the tracking problem. [30, p. 2]

The way SORT works is by approximating the inter-frame displacements of each object with a linear constant velocity model which is independent of other objects and camera motion. Each detected object is modeled as:

$$x = [u, v, s, r, \dot{u}, \dot{v}, \dot{s}]^T$$

The center pixel location is represented by $u$ (horizontal) and $v$ (vertical). Scale and aspect ratio of the target's bounding box are represented by $s$ and $r$, respectively. When there is a detection associated with a target, its state is updated by the detected bounding box, where the velocity components are solved by the Kalman filter framework. Otherwise, the state is simply predicted without correcting the linear velocity model. [30, p. 3]

Each target's bounding box is estimated by predicting its new location in the current frame when assigning detections to existing targets. The assignment cost matrix is computed as the intersection-over-union (IOU) between the detection and all predicted bounding boxes and then solved optimally by the Hungarian algorithm. To prevent target overlap, a minimum IOU is imposed to reject assignments with higher overlap than the threshold. [30, p. 3]

Unique identifiers are given to each object, which needs to either be created or destroyed. When an object is first detected, its tracker is initialized with a velocity set to zero. Since the velocity is unobserved, the covariance of the velocity component is initialized with large values. The new tracker also undergoes a probationary period where the target is associated with more detections to prevent false positives. On the other hand, tracks are removed when they are not detected for a set period $T_{Lost}$ to prevent unbounded growth of trackers. $T_{Lost}$ is normally set to 1 because the constant velocity model is a poor predictor of the true dynamics and early deletion of the tracks increased efficiency. [30, p. 3]

### 3.6.2 Deep SORT

Deep SORT is a tracking algorithm based on the SORT algorithm, but with an added deep association metric. The SORT algorithm is achieving good performance in terms of tracking precision and accuracy, but it also suffers from a lot of identity switches. The reason for this is that the employed association metric is only accurate when state estimation uncertainty is low. The deep SORT algorithm employs a convolutional neural network that replaces the association metric and appearance information. This increases the

robustness against misses and occlusions while keeping the system easy to implement. [31, p. 1]

Deep SORT is an extension of the SORT algorithm and thus the Kalman filtering framework is almost identical to the original. The tracking algorithm is based on a general tracking scenario where the camera is uncalibrated and no ego-motion information is available. Even though it is not optimal, this is the most common setup for multiple object tracking scenarios. Similar to the SORT algorithm, the tracking scenario uses the eight-dimensional state space containing the bounding box center position, aspect ratio, height, and their respective velocities in image coordinates. A standard Kalman filter with constant velocity motion and linear observation model is used, where the bounding coordinates are taken as direct observations of the object state. [31, p. 2]

The Hungarian algorithm is also used in deep SORT to solve the assignment problem, where motion and appearance information is integrated through a combination of two appropriate metrics. The first metric is found by using the (squared) Mahalanobis distance between predicted Kalman states and newly arrived measurements and then excluding unlikely associations by thresholding the distance at a 95% confidence interval from the inverse $\chi^2$ distribution. Unaccounted camera motion can introduce rapid displacements in the image plane, making the Mahalanobis distance an uninformed metric when there are occlusions. Thus a second metric is integrated, which is obtained by computing an appearance descriptor for each bounding box detection. A gallery of the last 100 associated descriptors is kept, and then the smallest cosine distance between the tracks is measured and compared to a threshold to see if the association is admissible. In practice, a pre-trained CNN is used to compute the bounding box appearance descriptors. Combined, these two metrics complement each other. The Mahalanobis distance provides information about possible object locations, and the cosine distance considers appearance information that is useful to recover identities after long-term occlusions. [31, p. 2]

Instead of solving for measurement-to-track associations in a global assignment problem, deep SORT uses a matching cascade that solves a series of subproblems. This is done by giving priority to more frequently seen objects to encode the notion of probability spread in the association likelihood. The matching cascade gives priority to tracks that have been seen more recently. To account for sudden appearance changes due to e.g. partial occlusion and to increase robustness against erroneous initialization, intersection over union association is run as in the original SORT algorithm. [31, p. 3]

# Chapter 4

# Approach

This chapter will give an overview of how the project has been implemented. The code snippets in this chapter have been provided as pseudo-code to make them easily readable. A link to the Github repository has been provided in appendix A, while the main code excerpts have been provided in appendix B.

## 4.1 Tools

### TensorFlow

TensorFlow is an open-source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries, and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML-powered applications. It is very fast and offers support for Python. TensorFlow also offers GPU support, which greatly increases the speed and performance. [32] Hence, TensorFlow has been used as the machine learning library in this thesis.

### OpenCV

The Open Source Computer Vision Library (OpenCV) was used in this thesis to visualize and process images and videos. OpenCV is an open-source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception. The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. It

supports Python and Windows, and it leans mostly towards real-time vision applications. [33]

**Superannotate**

Superannotate was used to annotate footage that was used in training and tests. Superannotate is an end-to-end platform used to annotate, version, and manage ground truth data for AI projects. It offers robust and user-friendly image, video, and text annotation tools. This makes it fast and it is feature-rich [34].

**Test system**

A personal computer with Windows 11 as operating system, i7-8700K as CPU, 32 GB RAM, and GTX 1080 Ti (Titanium) 11 GB VRAM have been used in the analysis of all AID methods. The Nvidia toolkit for machine learning was used such that the GPU could be used for model training and inference [35].

## 4.2 Limitations and simplifications

### 4.2.1 Datasets

The datasets used are a mixture of publicly available and self-annotated incident footage obtained from YouTube. A more specific dataset with images from Norwegian road tunnels would benefit the research, and could thus lead to providing more accurate and local recommendations for Norwegian tunnels.

Self-annotating images take a lot of time and thus only a limited amount of videos were annotated. Neural networks are very data-dependent, and would thus greatly benefit from being trained on a large dataset of actual incident footage. Due to the limitation of available annotated footage, the models were trained on a relatively small dataset.

### 4.2.2 GPU resources

Due to relatively low GPU VRAM, it was necessary to lower the batch size and resolution to train the models. Larger resolution models would increase the accuracy of the models, and a faster GPU would decrease the inference times.

### 4.2.3 Code availability

The code for most of the incident evaluation methods was not publicly published, and thus a simple incident evaluation method was developed due to limitations on time.

## 4.3 Datasets

The datasets used in the training and testing phase of this thesis are a combination of publicly available and self-annotated datasets. Only the classes car, truck, person, motorbike, bike, and bus are regarded from each dataset.

All the models used in this thesis were pre-trained on the COCO dataset, and thus the feature extraction part of the models is based on this dataset. The COCO dataset is a large-scale object detection, segmentation, and captioning dataset, containing over 200 000 annotated images and 80 object categories [36].

The dataset used in [37] has also been used to train the models. Data was collected from publicly available webcams located in Rogaland and manually annotated. Bounding boxes were drawn on vehicles of interest and labeled, and saved in an Extensible Markup Language (XML) file along with the source image [37, p. 29-30]. Most of the images were 800x600 pixels in RGB format [37, p. 42].

The Kitti dataset contains a suite of vision tasks built using an autonomous driving platform. The dataset contains about 15 000 annotated images and over 80 000 labeled objects [38]. Images from four classes in this dataset were used to train the models [37, p. 30].

The test dataset used consists of self-annotated videos obtained from YouTube [39]. In total there are 12 videos of tunnel incidents comprising 904 images, and 3379 objects distributed as in Table 4.1. Each video lasts between 10 and 60 seconds and is in RGB format with a resolution of 1280x720 pixels and 5 frames per second. Bounding boxes were drawn over the objects in each frame. Each object was given a class and additional information such as the type of incident, if it was in incident status or OK status, and if the object was occluded. The occluded field was used for hard to detect objects as well, e.g. vehicles that were far from the camera. Since only the incident classes stopped-vehicle, wrong-way driver, and pedestrian is regarded in this thesis, the status field was made to tell if a vehicle was involved in an incident outside of these classes. This field can thus be used in further research.

| car | person | truck | motorbike |
|---|---|---|---|
| 2524 | 551 | 264 | 40 |

Table 4.1: Distribution of objects in the test dataset

| car | person | truck | bus | bike | motorbike |
|---|---|---|---|---|---|
| 67 470 | 13 314 | 4252 | 671 | 13 | 34 |

Table 4.2: Distribution of objects in the training dataset

## 4.4 Preparation of data

To prepare all the datasets for training, I had to make a script to extract the information from both .json files and .xml files. The self-annotated data was stored in .json files, while the other datasets stored information in .xml files. After all the files were parsed, they were split in a 90-10 train-validation distribution. However, the validation dataset was not used in the actual tests. Listing 4.1 shows pseudo-code for the implementation on how to generate the tensor flow records.

Table 4.2 shows the distribution of objects in each class used in the training dataset and table 4.3 shows the validation dataset. In total there were 12 742 files and 85 754 objects in the training dataset, and 1410 files with 6653 objects in the validation dataset.

Data augmentation is used when training the models to compensate for the lack of data and thus increasing the generalization performance. The image augmentation done on the dataset includes horizontal flip, adjustments to brightness, contrast, hue, saturation, color distortion, and cropping.

```
datasets = [{"dataset1": dataset1}, {"dataset2": dataset2}]
data = Prepare(datasets)

for entry in data.get_training_entries():
    tf_example = create_tf_example(entry)
    write(tf_example)

for entry in data.get_test_entries():
    tf_example = create_tf_example(entry)
```

| car | person | truck | bus | motorbike |
|---|---|---|---|---|
| 5126 | 924 | 550 | 47 | 6 |

Table 4.3: Distribution of objects in the validation dataset

```
10     write(tf_example)
```

Listing 4.1: Pseudo-code on how to generate .tfrecords

## 4.5 Image enhancement

The image enhancement methods implemented in this thesis are as follows:

- Linear gray transformation
- Nonlinear gray transformation
- Histogram Equalization
- Retinex SSR
- Retinex MSR
- Masks

The OpenCV library provides a linear gray transformation and histogram equalization functions. A gamma function with $\gamma = 2.0$ was used in the nonlinear gray transformation. The implementation in [40] was used as the Retinex SSR and MSR image enhancement methods. Custom masks for each video clip were made using Adobe Photoshop and applied to the frames.

## 4.6 Object Detection

Four object detection models were tested and compared in this thesis. After the image enhancement step, the image was fed to the object detector class to obtain the model detections.

### 4.6.1 TensorFlowAPI

A selection of pre-trained object detection models from the TensorFlowAPI model zoo has been used in this thesis. The TensorFlow Model Garden is a repository with several different implementations of state-of-the-art models, such that Tensorflow users can take full advantage of Tensorflow for their research and product development [41]. The models used in this thesis include SSD MobileNet, EfficientDet, and Faster-RCNN, which all are trained on the COCO 2017 dataset [36]. In the transfer learning process only the classes car, truck, bus, person, bike, and motorbike were evaluated.

These models are among the state-of-the-art object detection algorithm, in terms of both speed and performance. Because incident detection systems should run in real-time, the system would benefit from lower inference times and thus faster object detection models. Hence, smaller resolution object detection models have been chosen because they are faster, but also less accurate than their larger resolution counterparts.

During the training phase of the models, I encountered out-of-memory exceptions. This was because of too large batch size and resolution of the images. Hence I lowered the batch size and resolution on some of the models. The configuration for each model is listed below:

- **SSD MobileNet**
  - Modelname: SSD MobileNet v2 320x320
  - Speed: 19 ms
  - COCO mAP: 20.2
  - Training config:
    - Batch size: 64
    - Resolution: 320x320 px
    - Figure 4.1 shows convergence after about 10 000 steps.
- **EfficientDet**
  - Modelname: EfficientDet D0 512x512
  - Speed: 39 ms
  - COCO mAP: 33.6
  - Training config:
    - Batch size: 8
    - Resolution: 512x512 px
    - Figure 4.2 shows convergence after about 15 000 steps.
- **Faster-RCNN**
  - Modelname: Faster-RCNN ResNet50 V1 640x640
  - Speed: 53 ms
  - COCO mAP: 29.3
  - Training config:
    - Batch size: 4
    - Resolution: 512x512 px
    - Figure 4.3 shows convergence after about 20 000 steps.

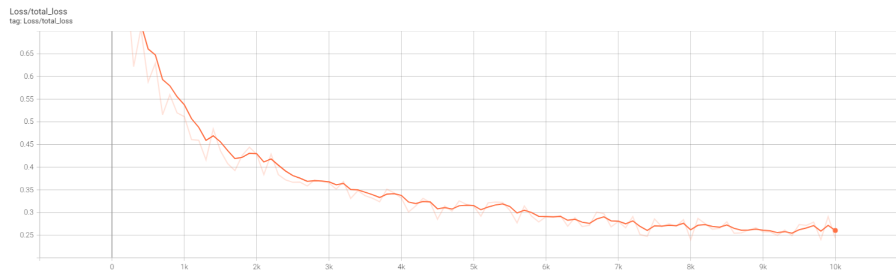


Figure 4.1: Total loss graph for SSD MobileNet.

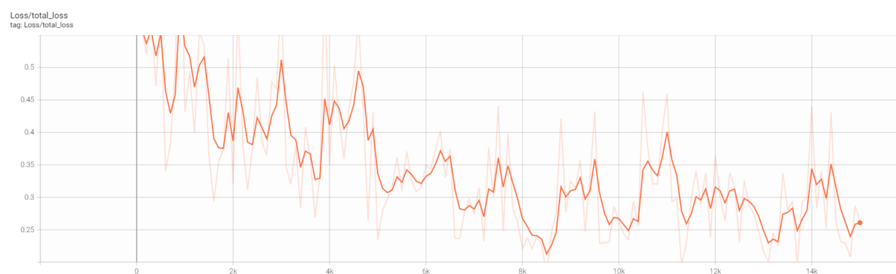


Figure 4.2: Total loss graph for EfficientDet.

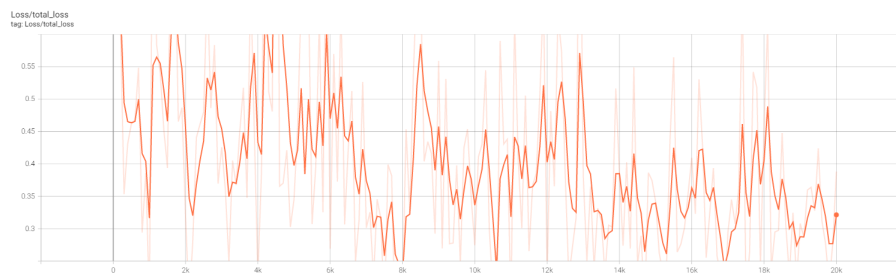


Figure 4.3: Total loss graph for Faster-RCNN.

### 4.6.2 YOLOv5

Ultralytics has released YOLOv5, which is a family of compound-scaled object detection models trained on the COCO 2017 dataset [42], [36]. The YOLOv5x implementation was used in this thesis because it offers excellent performance while also being fast. On the COCO benchmark dataset the YOLOv5x implementation managed this performance [42]:

- Resolution: 640x640px
- COCO mAP: 50.7
- Speed: 12.1 ms (Tesla V100 b1 GPU)

YOLOv5 is an improvement of the YOLOv4 implementation, as mentioned in section 3.5.4. Hence only the YOLOv5 object detection algorithm from the YOLO family was tested in this thesis.

The YOLOv5x algorithm was tested both with and without fine-tuning. The implementation without fine-tuning is used as a baseline for all models and to test the effectiveness of the image enhancement methods. The fine-tuned model was trained with this configuration:

- Batch size: 4
- Resolution: 640x640 px
- Figure 4.4 shows convergence after about 30 epochs.

## 4.7 Tracking

### 4.7.1 Simple tracking

A very simple tracking algorithm was implemented to make a baseline tracking model that could be comparable to the more advanced tracking algorithm used. The simple tracking algorithm leverages intersection over union (IoU) to determine if new detections correspond to previous detections. If IoU is greater than 0.5, then the new detection is assigned the same ID as the previous detection. Otherwise, it will be assigned a new ID. Psuedo-code for the algorithm is provided in listing 4.2.

```
1 for new_detection in new_detections:
2     for old_detection in old_detections:
3         IoU = calculate_IoU(new_detection, old_detection)
4         if IoU > 0.5:
```

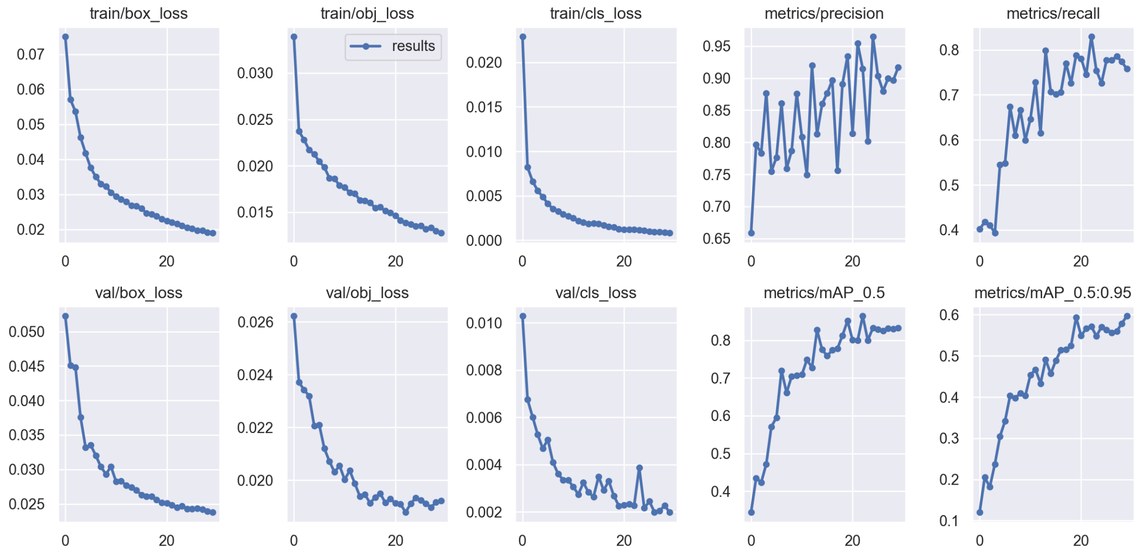


Figure 4.4: Training results from the YOLOv5 model.

```
5               track = old_track  # Assign old track object to
    current track
6           else:
7               track = new_track  # Create a new track object
```

Listing 4.2: Pseudo-code on how the simple tracker works

### 4.7.2 Deep Sort

The deep SORT algorithm is a state-of-the-art object tracking algorithm, which offers real-time multi-object tracking performance. The algorithm used in this thesis is based on the implementation in [43], which is a version of the original deep SORT model [44].

## 4.8 Incident evaluation

Three incident classes were considered in this thesis:

- Pedestrians
- Stopped vehicles
- Opposite-driving vehicles

Incidents in road tunnels usually lead to stopped vehicles or pedestrians walking out of the vehicles.

#### Opposite-driving vehicles

None of the videos in the test datasets included incidents where there were any wrong-way drivers. Hence this algorithm was not comprehensively developed. However, each vehicle was given a direction after two frames. This could then be used to determine wrong-way drivers or anomalous vehicle trajectories. Manual setup is needed, where upstream and downstream directions must be determined in advance.

#### Stopped vehicles

To determine if a vehicle has stopped, the tracks from the tracking algorithm were used. If a vehicle was at the same position for more than 2 frames, then it was classified as a stopped vehicle.

#### Pedestrians

There should not be any pedestrians in road tunnels, and thus any detections of pedestrians were determined as incidents.

## 4.9 Performance evaluation

Many performance evaluation metrics were considered to compare the different algorithms and methods used in this thesis.

#### Detection Accuracy

Intersection over Union (IoU, see figure 4.5) was calculated for each track and the real object, where the real object signifies a vehicle that has been annotated by a human and the track is a detection fed to the tracking algorithm. The highest IoU above 0.4 was selected for each track, resulting in the pseudo-code in listing 4.3.

```
1   for track in tracks:
2       for real_object in real_objects:
3           IoU = calculate_IoU(track, real_object)
4           if IoU > 0.4 and IoU > max_IoU:
5               max_IoU = IoU
6               track['id'] = real_object['id']
```

Listing 4.3: Pseudo-code on how tracks are assigned to real objects

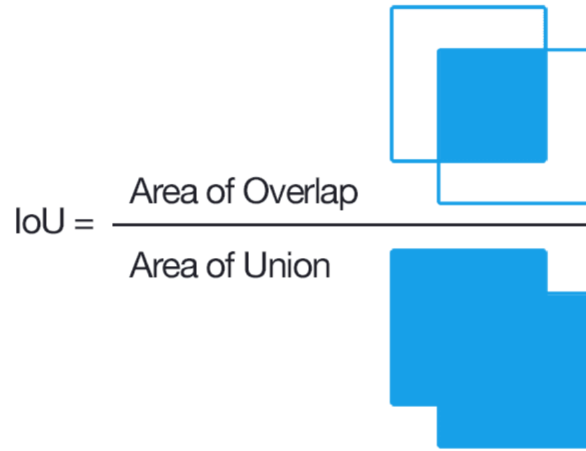


Figure 4.5: Intersection over union. [45]

The detection accuracy is then calculated as the average IoU for all valid detections. A valid detection means a detection that corresponds to a real object, and therefore is not a false positive.

$$DA = \frac{avg(IoU(real\ object, detected\ object))}{number\ of\ valid\ detections} \tag{4.1}$$

where DA denotes Detection Accuracy.

Because some of the detection was either occluded or hard to detect due to e.g. smoke or if it was truncated, all objects were annotated with a class called occluded. An object was therefore annotated with either false or true on the occluded class. Also, because these objects were particularly hard to detect, another performance metric was added to show how well an algorithm performed on rather easily detectable objects:

$$DAA = \frac{avg(IoU(real\ object, detected\ object))}{number\ of\ valid\ detections - \ number\ of\ occluded\ objects} \tag{4.2}$$

where DAA denotes Detection Accuracy Adjusted.

### Tracking Accuracy

Tracking accuracy is divided into three parts: correct tracks, duplicate tracks, and lost tracks. A track is deemed correct if the track corresponds to the same vehicle as to the last detection of that vehicle. Sometimes several detections can overlap and thus have an IoU over 0.4 with the same real vehicle. Hence some of the tracks can correspond to the same ID, resulting in duplicate

tracks corresponding to the same real vehicle. Lost tracks are counted as the number of ID switches on the detections, which is a result of lost tracks in the deep SORT model.

$$TA = \frac{correct\, tracks}{total\, number\, of\, tracks} \tag{4.3}$$

where TA denotes Tracking Accuracy.

Mean tracking time (MTT) and maximum tracking time (Max TT) were also measured for both object tracking algorithms.

### Time measurements

The mean time (MT) was calculated for each of the image enhancement methods, while the mean total time to detection (MTTD) was calculated for all systems. The MTTD metric measures how long the AID system takes from the start of image enhancement to the end of the incident evaluation algorithm. This way, the real-time performance of each method could be measured and compared.

### Detection Rate

The incident detection rate (DR) is calculated as the number of correctly detected incidents per frame divided by the number of tracks corresponding to a real incident in the frame. As a result, the detection rate is relatively low. The detection rate metric could have been adjusted such that it counted the number of correctly detected incidents per video instead. However, it was chosen because it is interesting to see how well the incident evaluation algorithm was able to detect incidents from tracks on a frame by frame level.

$$DR = \frac{number\, of\, detected\, incidents}{number\, of\, detected\, incidents + number\, of\, missed\, incidents} \tag{4.4}$$

### False alarms

The false alarm rate (FAR) was measured by comparing the incident status of each track to the actual incident status of the real vehicle corresponding to the track. If the track had a positive incident status while the real vehicle did not, then it was counted as a false alarm.

$$FAR = \frac{number\, of\, false\, alarms}{total\, number\, of\, detections} \tag{4.5}$$

where the total number of detections means all valid detections plus all false positives.

### Missed and false positive detections

As mentioned above, the intersection over union is calculated for each real object and each track. After all detections have been assigned to a real object, missed detections are counted. This is done by comparing the number of real objects in the image to the number of tracks corresponding to real objects, and calculating the difference. On the other hand, if a detection is not assigned to a real object, it is counted as a false positive detection.

$$MD = \frac{number\ of\ real\ objects - number\ of\ detections}{number\ of\ real\ objects} \tag{4.6}$$

$$FP = \frac{number\ of\ false\ positives}{total\ number\ of\ detections} \tag{4.7}$$

# Chapter 5

# Discussion

In the previous chapter several image enhancement methods, object detection models, and a tracking algorithm was tested. In this chapter, the results will be presented and discussed. All methods have been tested on a custom dataset, and thus the results should only give a relative performance between the algorithms, and not necessarily be comparable to other algorithms not used in this thesis.

## 5.1 Image Enhancement

Image enhancement (IE) could lead to big improvements in detection and tracking accuracy, and reduce false alarms. Poor image quality and lighting conditions often lead to fewer object detections. It is possible to upgrade the camera and lighting equipment in tunnels in order to improve image quality, but this is often costly and time expensive to do. Hence, algorithms have been developed to enhance images. Several image enhancement algorithms have been tested by using a pre-trained YOLOv5 object detection algorithm, called YOLOv5x, and deep SORT as the object tracking algorithm. Figures 5.1 and 5.2 show two examples with the image enhancement methods applied. The results are shown in table 5.1.

The gray transformation algorithms did not improve the overall accuracy and performance significantly. Both algorithms produce very low overhead and work best for relatively dark images, but because the test dataset comprised both dark and bright images, the overall performance went down, as can be seen from table 5.1. The gray linear transformation slightly increased the contrast in the image, but overall brightness stayed the same, while the nonlinear transformation gamma function brightened the image, but also reduced the contrast slightly. This helped detection in the darker parts of the

| **IE method** | **DA** | **DAA** | **TA** | **DR** | **MT** | **FAR** | **MD** | **FP** |
|---|---|---|---|---|---|---|---|---|
| Original | 75.1% | 78.2% | 73.9% | 61.4% | 0 ms | 16.9% | 45.0% | 19.6% |
| Gray linear | 76.0% | 78.5% | 72.4% | 59.0% | 0 ms | 19.2% | 41.0% | 23.5% |
| Gray nonlinear | 76.1% | 78.6% | 71.2% | 63.1% | 1 ms | 17.8% | 43.9% | 22.6% |
| Histogram Equalization | 74.3% | 76.5% | 71.6% | 63.4% | 1 ms | 20.6% | 46.9% | 25.2% |
| Retinex SSR | 73.1% | 75.1% | 75.7% | 60.4% | 5491 ms | 11.9% | 49.2% | 16.3% |
| Retinex MSR | 73.7% | 75.6% | 74.3% | 61.5% | 11606 ms | 12.9% | 50.8% | 18.1% |
| Mask | 76.5% | 78.6% | 73.3% | 67.2% | 1 ms | 6.5% | 56.2% | 10.5% |

Table 5.1: Image enhancement methods comparison. Pre-trained YOLOv5x with deep SORT was used.

image, such as at the sides of the road, but it lowered the detection rates in the brighter parts of the image, such as on the road. By converting the image to gray, color noise can be reduced, such as chromatic aberrations, and thus reduce false positive detections. On the other hand, information is also lost, which can affect object detection performance and thus lead to both missed detections and false-positive detections. In table 5.1 the gray transformation algorithms had higher false-positive detections, but also had fewer missed detections. However, the false alarm rates increased significantly. Hence these enhancement methods should only be used in darker tunnels where fast image enhancement is important.

Similar to the gray transformation algorithms, the histogram equalization (HE) algorithm works better for darker images. The HE algorithm increased brightness in all parts of the images, which led to a brighter overall image, but it also overexposed the highlights. Because the middle of the road is normally brighter than the sides, the middle became overexposed while the sides were nicely lit, which can easily be seen in figure 5.1. As a result, the image quality normally goes down and thus negatively affects most of the video clips. This can also be seen in table 5.1 where the HE algorithm performed worse on almost all metrics compared to the original image.

Applying masks to the video clips can improve the overall detection performance in road tunnels, by helping the AID system only focus on important parts. Lights and other appliances on the ceiling and the tunnel walls can be detected as vehicles because these can resemble vehicles, as seen in figure 5.3. Water dripping down on walls and ceilings is also known to produce false alarms. By masking away walls and ceilings, the system can effectively reduce false-positive detections and false alarms. Table 5.1 shows that missed detections and false positives decreased significantly without significantly affecting the detection accuracy. However, masking the image can also lead

Figure 5.1: Image enhancements methods

Figure 5.2: Image enhancement methods

Figure 5.3: Example of false positive detections.

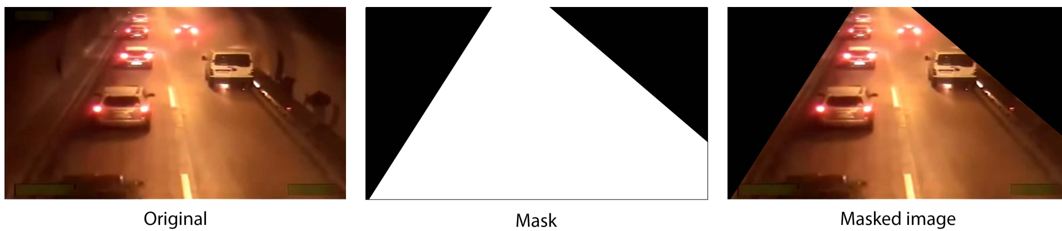


Figure 5.4: Masked image: Camera mounted on the ceiling.

to the loss of important detections, which can also be seen from the large increase of missed detections in table 5.1. E.g. a vehicle or pedestrian can extend outside of the mask region, as seen in figure 5.4 and 5.5, and thus be truncated which leads to a missed detection. Also, if a vehicle crashes into one of the walls of the tunnels, then the vehicle can be masked out of the image. For the masks to be effective, the cameras should be mounted on the ceiling directly above the road and only considered the closest part of the image, such that a simple mask of the road does not truncate any vehicles or exclude important regions of the road. Using masks on videos from wall-mounted cameras and cameras with a shallow angle, have a high probability of truncating vehicles, such as trucks and buses. As a last remark, masks reduce the number of pixels that need to be evaluated by the object detection system, which makes the object detection slightly faster.

The Retinex SSR and MSR algorithms improve the quality of the im-

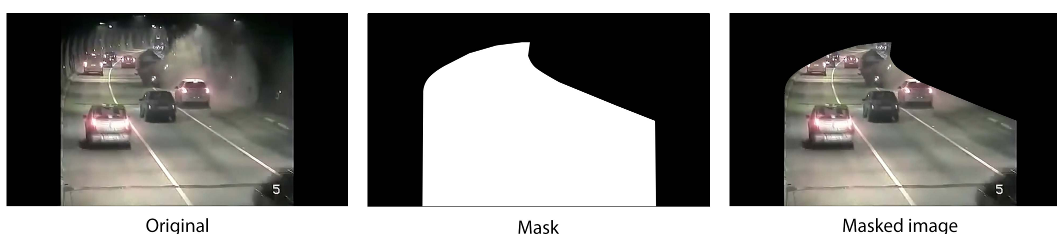


Figure 5.5: Masked image: Camera mounted in the wall.

ages, by increasing the contrast and brightness in shadow areas while also not overexposing the highlights. Also, the algorithms improve the white balance making the colors more accurate. Both algorithms improved the performance of the AID system by lowering both false-positive detections and the false alarm rate by a significant amount. However, the algorithms are computationally expensive, meaning they could not perform real-time detections, as table 5.1 shows.

It is possible to use deep neural networks to improve image quality, such as in [46] and [19]. Compared to other image-enhancing algorithms, this has proven to be significantly better. However, it is also significantly slower due to high computational complexity. Furthermore, deep neural network approaches also have a strong dependence on the datasets [19, p. 26]. As a result, deep neural networks are generally not fast enough for real-time detection applications [46, p. 8].

One image enhancement method may work for one clip but worsen another clip. Each clip must therefore be evaluated and relevant enhancement methods applied. E.g. the nonlinear gray transformation in 5.2 works well for that tunnel scene, but worsen the image in 5.1 by lowering the contrast to much. Computational speed is also important to evaluate because AID systems are dependent on being able to perform in real-time.

## 5.2 Detection

A well-performing object detection algorithm is imperative in an AID system. The results of four of the most common object detection algorithms are listed in table 5.2.

A pattern present in all methods is that vehicles located far from the camera are harder to detect than vehicles closer to the camera. Many of the missed detections, false alarms, and inaccuracies were due to this. E.g. vehicles far from the camera would move slowly relative to the camera and thus be classified as stopped vehicles. Hence it would be beneficial to define

| Algorithm | DA | DAA | TA | DR | MTTD | FAR | MD | FP |
|---|---|---|---|---|---|---|---|---|
| SSD MobileNET | 80.7% | 82.3% | 90.8% | 63.8% | 62 ms | 2.5% | 55.2% | 3.7% |
| YOLOv5x pre-trained | 75.1% | 78.2% | 73.9% | 61.4% | 58 ms | 16.9% | 45.0% | 19.6% |
| YOLOv5x fine-tuned | 82.3% | 83.8% | 90.6% | 71.8% | 56 ms | 7.7% | 15.8% | 6.3% |
| EfficientDet | 80.2% | 81.3% | 90.8% | 64.6% | 79 ms | 1.3% | 65.4% | 1.3% |
| Faster RCNN | 74.7% | 77.1% | 88.7% | 69.5% | 126 ms | 9.8% | 28.0% | 9.4% |

Table 5.2: Performance evaluation. Deep SORT was used with all models.

an object detection area close to the camera and ignore detections outside this area. This way, the system can ensure good and reliable detections, with fewer false alarms and false positives. On the other hand, it is important to have enough cameras to cover the whole tunnel and not leave any blind spots.

Vehicles involved in incidents often lead to smoke and damaged vehicles, which obstructs the view of the cameras, as shown in figure 5.6. This can lead to the loss of important detections. The pre-trained YOLOv5 implementation has not been trained on damaged vehicles, and thus struggled to detect the vehicles figure 5.6. The other object detection models were trained on incident footage containing damaged vehicles and vehicles partially obscured by smoke. These models were therefore able to easier detect vehicles obstructed by smoke or damaged vehicles, which can be seen by the slightly higher detection rates in table 5.2. Figure 5.7 also shows an example where the SSD MobileNet algorithm was able to detect an upside-down vehicle, while the pre-trained YOLOv5 model could not.

All models performed close to real-time with relatively high detection and tracking accuracy. The Faster RCNN model had the highest inference time and performance measures should therefore be evaluated, such as reducing input resolution or skipping frames, in order to be able to perform in real-time. The SSD MobileNet and EfficientDet systems had by far the lowest false alarm rates and false positives. However, they also had much higher missed detections compared to the other models. This could be a problem in an AID system because it could lead to the loss of important detections. The Faster RCNN model, on the other hand, provided a better trade-off between missed detections and false alarms. The false alarms incurred by the model can instead be reduced by other means, such as masks. A confidence level of 60% was used on the models from the TensorFlowAPI. Allowing lower confidence levels can lead to more false positives and false alarms, and an excessive confidence level can lead to missed detections. The confidence level chosen provided a decent trade-off.

The YOLOv5 algorithm performed very well without being trained on the

Figure 5.6: Crashed vehicles produce smoke which obstructs the view of the camera.

Figure 5.7: Comparison between pre-trained YOLOv5x and trained SSD Mobnet .

incident footage from tunnels. It had more false positive detections, but it was faster and provided similar detection rates as the other models. Allowing the models to be trained on specific datasets that represent the road tunnels where the AID system should be used, greatly improves false alarms, false positives, detection rate, and missed detections. This is evident from the fine-tuned YOLOv5x implementation, which outperformed the other models in terms of speed, missed detection, and detection rate. Hence, the YOLOv5 model proves to be a good object detection algorithm for use in an AID system.

## 5.3 Tracking

Similar to object detection, a well-performing tracking algorithm is also very important in order for the AID system to correctly identify incidents. As can be seen from the missed detections in table 5.2, the object detection systems are not able to detect every vehicle in every image. In many cases, vehicle detections are lost for a few frames. A good tracking algorithm should still be able to keep the tracks and assign the same ID to the detected vehicles. Detecting stopped vehicles and wrong-way drivers heavily depends on whether the system is able to keep track of different vehicles and their IDs.

Different positions and angles of the CCTV cameras may have an impact on the tracking performance. Cameras that are mounted low and thus have a shallow angle, have a higher chance of getting occluded vehicles. Less advanced trackers will easily lose occluded tracks, making incident evaluation less accurate. Also, a longer duration of occluded vehicles could obstruct the view of the vehicles involved in incidents, leading to slower incident evaluation times. Cameras mounted on the wall of the tunnel can also lead to worse tracking performance due to occlusion, compared to ceiling-mounted cameras. This is shown in figure 5.8.

Cameras are mounted differently in many tunnels concerning position and camera angles, which means that the tracking algorithm must have a good generalization performance. From table 5.3 it is evident that the deep SORT algorithm has a much higher tracking accuracy and higher mean tracking time compared to the simple tracking algorithm. Furthermore, it is interesting to see that there are fewer initial detections that were kept after the deep SORT algorithm had evaluated them. This is because the algorithm tries to remove duplicate and false tracks. We thus can conclude that the deep SORT algorithm proved to be fast and very accurate, and thus generally suitable for real-time multi-object tracking. However, the algorithm did not perform in real-time when there were many tracks. This could be

Figure 5.8: The white truck obstructs the view of the wall mounted camera, leading to occlusions and thus less detections.

| Algorithm | TA | MTT | Max TT | Valid detections |
|---|---|---|---|---|
| Deep SORT | 90.8% | 12 ms | 455 ms | 1473 |
| Simple tracking algorithm | 66.1% | 0 ms | 7 ms | 1800 |

Table 5.3: Performance evaluation of deep SORT and a simple tracking algorithm using SSD MobileNet as detection algorithm.

a problem in congested traffic, and skipping frames would most probably be needed. The simple tracking algorithm was very fast, but suffered from many ID switches and duplicate IDs, making it less suitable for general object tracking problems. Hence the simple tracking algorithm would need to be customized for each tracking problem.

## 5.4 Incident evaluation

In almost all incidents in road tunnels, vehicles will eventually stop and people may exit their vehicles. This makes evaluating stopped vehicles or pedestrians a good option but could lead to slower detection times and missed detections. More advanced algorithms that can detect damaged vehicles, anomalous vehicle trajectories, smoke, fire, and crashes would make the incident detections more accurate and copious.

As can be seen from the result in table 5.2, there were many missed incident detections. Due to how the detection rate is calculated, it is relatively low for all models. Most of the methods were able to correctly identify if there had been an incident in all video clips, but only a few of the incidents were detected on a frame-by-frame level. Hence, it is evident from the results that the incident evaluation algorithm used can be significantly improved, and it is also not sufficient to only consider stopped vehicles, wrong-way drivers, and pedestrians as incident classes.

False alarms could be reduced by evaluating longer parts of the videos before incurring alarms. In the tests, every false alarm in every frame was counted. Some of these false alarms were due to temporary shadows or light reflecting on the ground or walls, but these detections were only lasting for a few frames. Evaluating longer parts of the video would increase incident detection time, but would also increase the confidence of the system, resulting in fewer false alarms.

## 5.5 Performance

All detection algorithms, except faster RCNN, combined with the deep SORT algorithm can currently handle about 20-25 frames per second on the test system used in this thesis, which is close to real-time. This means that there is little overhead for computationally expensive image enhancement algorithms or incident evaluation methods. A solution could be to skip frames, e.g. every second or third frame. Unless it is a high-speed tunnel, vehicles do not move very far in a few frames, which makes skipping frames a viable option. As seen from the test results, the detection and tracking algorithms were able to keep most of the tracks with a frame rate as low as 5 fps.

The object detection speed was not affected much when scaling down the input images, but the image enhancement methods had a large improvement which can be seen in table 5.4. However, the number of missed detections is prohibitively high when reducing the image resolution by 50%. Less aggressive resize parameters could be considered to increase speed.

Smaller resolution object detection models are faster than larger models. Furthermore, single-stage object detectors are faster than two-stage detectors. Hence, smaller single-stage object detectors would be preferred in weaker or lower-end systems. Larger object detection models are more accurate than the smaller detectors, and should thus be used if the system can handle them in real-time.

| Algorithm | MT | MD | Valid detections |
|---|---|---|---|
| Retinex SSR 50% | 1197 ms | 82.8% | 549 |
| Retinex SSR 100% | 5491 ms | 49.2% | 1820 |
| Retinex MSR 50% | 2380 ms | 83.0% | 547 |
| Retinex SSR 100% | 11 606 ms | 50.8% | 1765 |

Table 5.4: Differences in speed and missed detections using different resolution. The number next to the names signifies the scale percent used. Pre-trained YOLOv5x was used as object detection model.

## 5.6 Model- and Analytical Improvements

The datasets used in this thesis could have consisted of more footage from actual incidents and preferably from Norwegian tunnels. Also, the dataset could have been larger and included more incidents in more classes. E.g. none of the datasets contained videos of wrong-way drivers. With more data, the models could also have been trained on a completely different dataset than the test dataset, and therefore the generalization performance could have been evaluated more accurately.

The test dataset should have contained more frames per video clip. The video clips were down-sampled to 5 frames per second, instead of the regular 24 or 30 frames per second. The accuracy and performance of an AID system could have been tested in more detail, with more headroom to fine-tune performance metrics. Also, longer clips of the incidents would be beneficial. Most of the clips ended shortly after the incident occurred, and thus sometimes it did not show that vehicles arriving later stopped. Longer clips would also mean that the system could have had more frames to analyze before deciding on whether it was an incident or not. E.g. it could spend another 2 seconds to evaluate an incident and thus reduce false alarms. Furthermore, footage from several cameras within the same tunnel would be better in terms of testing the extent of AID systems. Then it would be possible to make more distinct detection zones for all cameras and determine more accurately the effect camera positions and angles have on the AID system.

The incident evaluation method used was not particularly good, which can be seen from the detection rate. More advanced incident evaluation algorithms should be tested, compared, and assessed to determine the best AID systems. E.g. neural networks, SVMs, and decision trees could have been tested as the incident evaluation algorithm, but due to limitations on time these options were not explored.

The object detection and tracking models could have been more fine-tuned, by e.g. using different resolutions, batch sizes, image augmentations,

balancing the datasets, etc. The deep SORT association metric could also be trained on a more specific dataset containing damaged vehicles, in order to improve tracking accuracy. Another way of detecting incidents could be to extract traffic parameters from the detection and tracking results, which could then be analyzed to evaluate incidents. This method would allow testing of the majority of the different AID algorithms developed (such as e.g. statistical or comparative algorithms), as mentioned in the introduction.

The models could also have been tested on different equipment, both worse and better, to test the limits of a real-time AID system. The systems could e.g. be tested on a raspberry pi or higher-end GPUs such as a Tesla P100, in order to see the economic impact an AID system can impose.

## 5.7 Further work

Evaluating more incident classes would be a natural next step. In this thesis, only the classes "stopped vehicle", "wrong-way driver" and "pedestrian" have been evaluated. Stationary objects, anomalous trajectories, smoke, fire, and stopped vehicles in queues would increase the overall AID performance.

Another improvement would be to look into other algorithms and methods. The Nvidia Deepstream toolkit looks promising, due to excellent detection, tracking, and computation performance. Better occlusion handling for object tracking, such as the algorithm proposed in [47], could be implemented to test the performance in e.g. queues.

Automatic lane detection and configuration, similar to the one proposed in [16], that can decide the upstream and downstream direction would decrease a lot of the initial manual work needed to set up an AID system. Some of the manual work needed include defining masks, and upstream and downstream direction to detect wrong-way drivers.

Finally, more image enhancement methods could also be explored, such as the deep neural network image enhancing method used in [46]. Because deep neural networks are computationally expensive, it would also be necessary to look into methods that decrease detection, tracking, and evaluation time. Otherwise, the AID system would struggle to be able to operate in real-time.

# Chapter 6

# Conclusion

There has been a large demand for good AID systems that can help reduce vehicular incidents on roads. Hence there has been a lot of research on AID systems throughout the years. Most of the systems are based on analysis of traffic parameters extracted by e.g. inductive loops. Recently, machine learning techniques and computer-vision approaches have been developed and tested in AID systems. This thesis has therefore looked at how to improve AID systems using computer-vision and neural networks.

Deep neural network-based AID systems using traffic surveillance cameras can achieve great performance in detecting incidents in road tunnels accurately and fast. However, these systems also often produce a lot of false alarms which reduces their effectiveness.

Image enhancement methods have the potential to increase detection accuracy and reduce false alarms. A downside to this is that one image enhancement method may work for one clip, but worsen another clip. Therefore it is necessary to evaluate each clip and determine which image enhancement method works the best. Some image enhancement methods, such as the Retinex algorithms or neural network-based methods, have better generalization performance but also add computational complexity to the system.

All object detection models tested, performed close to real-time with relatively high detection and tracking accuracy. The Faster RCNN model had fewer missed detections, but also slightly higher false positives and false alarms compared to the other models. The fine-tuned YOLOv5x algorithm was the best model in terms of detection accuracy, detection rate, missed detection, and speed. By defining masks and detection zones, the object detection algorithms could perform better. This way only vehicles closer to the camera would be evaluated, and thus false positives and alarms related to walls, ceilings, and objects in the distance would be reduced. Overall the YOLOv5 algorithm offered a good trade-off between speed, detection

accuracy, and false alarms, making it well suited for use in an AID system.

The deep SORT algorithm is fast and very accurate, and thus generally suitable for real-time multi-object tracking. However, the algorithm did not perform in real-time when there were many tracks, and performance measures should be taken. A simpler tracking algorithm could be used in specific scenarios but would need customization for each scenario.

The incident evaluation algorithm was mostly capable of detecting incidents in the incident classes "stopped vehicle", "wrong-way driver" and "pedestrian". However, it is evident from the results that more advanced methods and algorithms should be used. Incidents in more classes should also be considered to increase the detection rates. As a final remark, the incident evaluation algorithm should evaluate longer parts, e.g. 2 seconds, of the video before incurring an alarm. This measure would further reduce the number of false alarms.

Computational speed is important to evaluate because an AID system should be able to operate in real-time. An important factor to evaluate is congestion and queues because these increase the latency due to the tracking algorithm not being able to perform in real-time when there were many tracks. Reduction of latency can be achieved by using smaller object detection models, less advanced tracking algorithms, simpler image enhancement methods, or less advanced incident evaluation algorithms. However, this affects the incident detection accuracy and is thus not preferable. Another way is to either skip frames or lower the resolution of the input images. The more advanced tracking algorithms can keep tracks even with several skipped frames. It is thus a better option to skip frames than to use less advanced enhancement, detection, tracking, and incident evaluation methods. Lowering the resolution of the input images greatly affects the detection by increasing the missed detections. Hence, the resolution should only be reduced by a small amount and used if the image enhancement method is computationally expensive.

# References

[1] "Nullvisjonen i norske vegtunneler," Statens vegvesen. (2022), [Online]. Available: https://www.vegvesen.no/fag/fokusomrader/forskning-innovasjon-og-utvikling/innovasjonspartnerskap/nullvisjonen-i-norske-vegtunneler/ (visited on 05/01/2022).

[2] "Utfordring," Statens vegvesen. (2022), [Online]. Available: https://www.vegvesen.no/fag/fokusomrader/forskning-innovasjon-og-utvikling/innovasjonspartnerskap/nullvisjonen-i-norske-vegtunneler/utfordring/ (visited on 05/01/2022).

[3] "Forskrift om minimum sikkerhetskrav til visse vegtunneler (tunnelsikkerhetsforskriften)," Lovdata. (2007), [Online]. Available: https://lovdata.no/dokument/LTI/forskrift/2007-05-15-517 (visited on 05/01/2022).

[4] "AID i tunnel teknologisammenligning," ViaNova. (2013), [Online]. Available: https://docplayer.me/17583116-Aid-i-tunnel-teknologisammenligning.html (visited on 05/01/2022).

[5] Y.-K. Ki, W.-T. Jeong, H.-J. Kwon, and M.-R. Kim, "An Algorithm for Incident Detection Using Artificial Neural Networks," in *2019 25th Conference of Open Innovations Association (FRUCT)*, ISSN: 2305-7254, Nov. 2019, pp. 162–167. DOI: 10.23919/FRUCT48121.2019.8981509.

[6] R. Browne, S. Foo, S. Huynh, B. Abdulhai, and F. Hall, "Comparison and analysis tool for automatic incident detection," *Transportation Research Record: Journal of the Transportation Research Board*, vol. 1925, pp. 58–65, Jan. 1, 2005. DOI: 10.1177/0361198105192500107.

[7] D. P. T. Martin, J. Perrin, B. Hansen, R. Kump, and D. Moore, "INCIDENT DETECTION ALGORITHM EVALUATION," p. 54, [Online]. Available: https://www.ugpti.org/resources/reports/downloads/mpc01-122.pdf.

[8] S. Tang and H. Gao, "Traffic-incident detection-algorithm based on nonparametric regression," *IEEE Transactions on Intelligent Transportation Systems*, vol. 6, no. 1, pp. 38–42, Mar. 2005, Conference Name: IEEE Transactions on Intelligent Transportation Systems, ISSN: 1558-0016. DOI: 10.1109/TITS.2004.843112.

[9] X. Li, W. Lam, and M. Tam, "New automatic incident detection algorithm based on traffic data collected for journey time estimation," *Journal of Transportation Engineering*, vol. 139, pp. 840–847, Aug. 1, 2013. DOI: 10.1061/(ASCE)TE.1943-5436.0000566.

[10] M. S. Shehata, J. Cai, W. M. Badawy, *et al.*, "Video-based automatic incident detection for smart roads: The outdoor environmental challenges regarding false alarms," *IEEE Transactions on Intelligent Transportation Systems*, vol. 9, no. 2, pp. 349–360, Jun. 2008, Conference Name: IEEE Transactions on Intelligent Transportation Systems, ISSN: 1558-0016. DOI: 10.1109/TITS.2008.915644.

[11] D. Singh and C. K. Mohan, "Deep spatio-temporal representation for detection of road accidents using stacked autoencoder," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 3, pp. 879–887, Mar. 2019, Conference Name: IEEE Transactions on Intelligent Transportation Systems, ISSN: 1558-0016. DOI: 10.1109/TITS.2018.2835308.

[12] J. Wang, X. Li, S. S. Liao, and Z. Hua, "A Hybrid Approach for Automatic Incident Detection," *IEEE Transactions on Intelligent Transportation Systems*, vol. 14, no. 3, pp. 1176–1185, Sep. 2013, Conference Name: IEEE Transactions on Intelligent Transportation Systems, ISSN: 1558-0016. DOI: 10.1109/TITS.2013.2255594.

[13] S. K. Kumaran, D. P. Dogra, and P. P. Roy, "Anomaly Detection in Road Traffic Using Visual Surveillance: A Survey," *ACM Computing Surveys*, vol. 53, no. 6, pp. 1–26, Nov. 2021, arXiv: 1901.08292, ISSN: 0360-0300, 1557-7341. DOI: 10.1145/3417989. [Online]. Available: http://arxiv.org/abs/1901.08292 (visited on 04/19/2022).

[14] C. Wang, Y. Dai, W. Zhou, and Y. Geng, "A Vision-Based Video Crash Detection Framework for Mixed Traffic Flow Environment Considering Low-Visibility Condition," en, *Journal of Advanced Transportation*, vol. 2020, e9194028, Jan. 2020, Publisher: Hindawi, ISSN: 0197-6729. DOI: 10.1155/2020/9194028. [Online]. Available: https://www.hindawi.com/journals/jat/2020/9194028/ (visited on 03/15/2022).

[15] S. Usmankhujaev, S. Baydadaev, and K. J. Woo, "Real-Time, Deep Learning Based Wrong Direction Detection," en, *Applied Sciences*, vol. 10, no. 7, p. 2453, Jan. 2020, Number: 7 Publisher: Multidisciplinary Digital Publishing Institute, ISSN: 2076-3417. DOI: 10.3390/app10072453. [Online]. Available: https://www.mdpi.com/2076-3417/10/7/2453 (visited on 04/19/2022).

[16] A. Haghighat and A. Sharma, "A computer vision-based deep learning model to detect wrong-way driving using pan–tilt–zoom traffic cameras," en, *Computer-Aided Civil and Infrastructure Engineering*, vol. n/a, no. n/a, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/mice.12819, ISSN: 1467-8667. DOI: 10.1111/mice.12819. [Online]. Available: http://onlinelibrary.wiley.com/doi/abs/10.1111/mice.12819 (visited on 03/15/2022).

[17] P. Chakraborty, A. Sharma, and C. Hegde, "Freeway Traffic Incident Detection from Cameras: A Semi-Supervised Learning Approach," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, ISSN: 2153-0017, Nov. 2018, pp. 1840–1845. DOI: 10.1109/ITSC.2018.8569426.

[18] L.-L. Wang, H. Y. T. Ngan, and N. H. C. Yung, "Automatic incident classification for large-scale traffic data by adaptive boosting SVM," en, *Information Sciences*, vol. 467, pp. 59–73, Oct. 2018, ISSN: 0020-0255. DOI: 10.1016/j.ins.2018.07.044. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025518305681 (visited on 03/15/2022).

[19] W. Wang, X. Wu, X. Yuan, and Z. Gao, "An experiment-based review of low-light image enhancement methods," *IEEE Access*, vol. 8, pp. 87 884–87 917, 2020, Conference Name: IEEE Access, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.2992749.

[20] "Histograms - 1 : Find, plot, analyze," OpenCV. (), [Online]. Available: https://docs.opencv.org/4.x/d1/db7/tutorial_py_histogram_begins.html (visited on 05/04/2022).

[21] S. Theodoridis, *Machine learning: a Bayesian and optimization perspective*, 2nd edition. London: Elsevier, Academic Press, 2020, 1131 pp., ISBN: 978-0-12-818803-3.

[22] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *arXiv:1311.2524 [cs]*, Oct. 22, 2014. arXiv: 1311.2524. [Online]. Available: http://arxiv.org/abs/1311.2524 (visited on 05/04/2022).

[23] "How single-shot detector (SSD) works?" ArcGIS Developer. (2019), [Online]. Available: https://developers.arcgis.com/python/guide/how-ssd-works/ (visited on 05/04/2022).

[24] G. Boesch. "Object detection in 2022: The definitive guide," viso.ai. (2022), [Online]. Available: https://viso.ai/deep-learning/object-detection/ (visited on 05/04/2022).

[25] PulkitS. "A step-by-step introduction to the basic object detection algorithms," Analytics Vidhya. (Oct. 11, 2018), [Online]. Available: https://www.analyticsvidhya.com/blog/2018/10/a-step-by-step-introduction-to-the-basic-object-detection-algorithms-part-1/ (visited on 05/04/2022).

[26] J. Solawetz. "A thorough breakdown of EfficientDet for object detection," Medium. (Apr. 30, 2020), [Online]. Available: https://towardsdatascience.com/a-thorough-breakdown-of-efficientdet-for-object-detection-dc6a15788b73 (visited on 05/04/2022).

[27] U. Nepal and H. Eslamiat, "Comparing YOLOv3, YOLOv4 and YOLOv5 for autonomous landing spot detection in faulty UAVs," *Sensors*, vol. 22, no. 2, p. 464, Jan. 2022, Number: 2 Publisher: Multidisciplinary Digital Publishing Institute, ISSN: 1424-8220. DOI: 10.3390/s22020464. [Online]. Available: https://www.mdpi.com/1424-8220/22/2/464 (visited on 05/04/2022).

[28] M. Fiaz, A. Mahmood, and S. K. Jung, "Tracking noisy targets: A review of recent object tracking approaches," *arXiv:1802.03098 [cs]*, Feb. 13, 2018. arXiv: 1802.03098. [Online]. Available: http://arxiv.org/abs/1802.03098 (visited on 03/11/2022).

[29] S. R. Maiya. "DeepSORT: Deep learning to track custom objects in a video," Nanonets. (2019), [Online]. Available: https://nanonets.com/blog/object-tracking-deepsort/ (visited on 05/04/2022).

[30] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple Online and Realtime Tracking," *2016 IEEE International Conference on Image Processing (ICIP)*, pp. 3464–3468, Sep. 2016, arXiv: 1602.00763. DOI: 10.1109/ICIP.2016.7533003. [Online]. Available: http://arxiv.org/abs/1602.00763 (visited on 04/20/2022).

[31] N. Wojke, A. Bewley, and D. Paulus, "Simple Online and Realtime Tracking with a Deep Association Metric," *arXiv:1703.07402 [cs]*, Mar. 2017, arXiv: 1703.07402. [Online]. Available: http://arxiv.org/abs/1703.07402 (visited on 04/20/2022).

[32] "TensorFlow." (2022), [Online]. Available: https://www.tensorflow.org/ (visited on 05/04/2022).

[33] "About," OpenCV. (2022), [Online]. Available: https://opencv.org/about/ (visited on 05/04/2022).

[34] "Build SuperData for your AI," SuperAnnotate. (2022), [Online]. Available: https://www.superannotate.com/ (visited on 05/04/2022).

[35] "CUDA Toolkit," NVIDIA Developer. (Jul. 2, 2013), [Online]. Available: https://developer.nvidia.com/cuda-toolkit (visited on 05/04/2022).

[36] "COCO - common objects in context." (2022), [Online]. Available: https://cocodataset.org/#home (visited on 05/01/2022).

[37] B. A. Eide, "Detecting and classifying vehicles entering and exiting a tunnel," 2021, Accepted: 2021-09-29T16:26:48Z Publisher: uis. [Online]. Available: https://uis.brage.unit.no/uis-xmlui/handle/11250/2786181 (visited on 05/02/2022).

[38] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," The KITTI Vision Benchmark Suite, in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012. [Online]. Available: http://www.cvlibs.net/datasets/kitti/eval_object.php?obj_benchmark=2d (visited on 05/04/2022).

[39] Red D Film, *Tunnel crash compilation! over 19 minutes!* Nov. 7, 2015. [Online]. Available: https://www.youtube.com/watch?v=IOxxEJpXZGU (visited on 05/04/2022).

[40] A. S. Krishnan, *Retinex image enhancement*, Apr. 5, 2022. [Online]. Available: https://github.com/aravindskrishnan/Retinex-Image-Enhancement (visited on 05/01/2022).

[41] H. Yu, C. Chen, X. Du, *et al.*, *Tensorflow model garden*, https://github.com/tensorflow/models, 2022.

[42] *YOLOv5*, May 1, 2022. [Online]. Available: https://github.com/ultralytics/yolov5 (visited on 05/01/2022).

[43] T. A. Guy, *Yolov4-deepsort*, Apr. 29, 2022. [Online]. Available: https://github.com/theAIGuysCode/yolov4-deepsort (visited on 05/01/2022).

[44] N. Wojke and A. Bewley, "Deep cosine metric learning for person re-identification," in *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, IEEE, 2018, pp. 748–756. DOI: 10.1109/WACV.2018.00087.

[45] "Intersection over union (IoU) for object detection," PyImageSearch. (Nov. 7, 2016), [Online]. Available: https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/ (visited on 05/04/2022).

[46] C. Chen, Q. Chen, J. Xu, and V. Koltun, "Learning to see in the dark," *arXiv:1805.01934 [cs]*, May 4, 2018. arXiv: 1805.01934. [Online]. Available: http://arxiv.org/abs/1805.01934 (visited on 04/30/2022).

[47] M. H. Nasseri, H. Moradi, R. Hosseini, and M. Babaee, "Simple online and real-time tracking with occlusion handling," *arXiv:2103.04147 [cs]*, Mar. 6, 2021. arXiv: 2103.04147. [Online]. Available: http://arxiv.org/abs/2103.04147 (visited on 05/05/2022).

# List of Figures


3.1 Examples of histograms [19, p. 5] . . . . . . . . . . . . . . . . . . 14
3.2 Light reflection model [19, p. 8] . . . . . . . . . . . . . . . . . . . 15
3.3 Retinex algorithm process [19, p. 8] . . . . . . . . . . . . . . . . . 15
3.4 Example of a three layer feed forward network. [21, p. 911] . . 16
3.5 Example of a convolution on a 3x3 input matrix (A) with a 2x2 filter matrix (B) [21, p. 957] . . . . . . . . . . . . . . . . . . 18
3.6 Example of zero padding a 5x5 input matrix [21, p. 962] . . . 18
3.7 Example of a pooling step. The original matrix (A) is a 6x6 matrix. A window of size 2x2 and stride 2 is chosen when performing max pooling. The resulting matrix is a 3x3 matrix (B). [21, p. 964] . . . . . . . . . . . . . . . . . . . . . . . . . . 19
3.8 Object classification vs object detection. [23] . . . . . . . . . . . 21
3.9 Regions with CNN features [22, p. 1] . . . . . . . . . . . . . . . 21
3.10 Architecture of a convolutional neural network with a SSD detector. [23] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22
3.11 The YOLOv5 architecthure. [27, p. 8] . . . . . . . . . . . . . . . 24

4.1 Total loss graph for SSD MobileNet. . . . . . . . . . . . . . . . . 33
4.2 Total loss graph for EfficientDet. . . . . . . . . . . . . . . . . . . 33
4.3 Total loss graph for Faster-RCNN. . . . . . . . . . . . . . . . . . 33
4.4 Training results from the YOLOv5 model. . . . . . . . . . . . . 35
4.5 Intersection over union. [45] . . . . . . . . . . . . . . . . . . . . . 37

5.1 Image enhancements methods . . . . . . . . . . . . . . . . . . . . 42
5.2 Image enhancement methods . . . . . . . . . . . . . . . . . . . . . 42
5.3 Example of false positive detections. . . . . . . . . . . . . . . . . 43
5.4 Masked image: Camera mounted on the ceiling. . . . . . . . . . 43
5.5 Masked image: Camera mounted in the wall. . . . . . . . . . . . 44
5.6 Crashed vehicles produce smoke which obstructs the view of the camera. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 46

5.7 Comparison between pre-trained YOLOv5x and trained SSD Mobnet . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 46
5.8 The white truck obstructs the view of the wall mounted camera, leading to occlusions and thus less detections. . . . . . . . . 48

# List of Tables

4.1 Distribution of objects in the test dataset . . . . . . . . . . . . 30
4.2 Distribution of objects in the training dataset . . . . . . . . . . 30
4.3 Distribution of objects in the validation dataset . . . . . . . . . 30

5.1 Image enhancement methods comparison. Pre-trained YOLOv5x with deep SORT was used. . . . . . . . . . . . . . . . . . . . . . . 41
5.2 Performance evaluation. Deep SORT was used with all models. 45
5.3 Performance evaluation of deep SORT and a simple tracking algorithm using SSD MobileNet as detection algorithm. . . . . . 48
5.4 Differences in speed and missed detections using different resolution. The number next to the names signifies the scale percent used. Pre-trained YOLOv5x was used as object detection model. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 50

# Appendix A

# Github repository

All code has been uploaded to this Github repository:

- **Github:** `https://github.com/aleksander-vedvik/Bachelor`
- **Datasets**

An explanation of the code and how to run is provided in the README.md file in the repository. A link to the datasets is also provided in the repository.

# Appendix B

# Code excerpts

```
1  import os
2  import sys
3
4  PATH_TO_THIS_FILE = os.path.dirname(os.path.abspath(__file__)
       )
5  sys.path.insert(0, PATH_TO_THIS_FILE + '\\tools\\')
6  sys.path.insert(0, PATH_TO_THIS_FILE + '\\tools\\deep_sort')
7  sys.path.insert(0, PATH_TO_THIS_FILE + '\\')
8  sys.path.insert(0, PATH_TO_THIS_FILE + '\\training\\')
9  sys.path.insert(0, PATH_TO_THIS_FILE + '\\training\\
       tensorflowapi\\')
10 sys.path.insert(0, PATH_TO_THIS_FILE + '\\training\\
       tensorflowapi\\research\\')
11 sys.path.insert(0, PATH_TO_THIS_FILE + '\\training\\
       tensorflowapi\\research\\object_detection')
12
13 import cv2
14 import numpy as np
15 from tools.detection_model import Detection_Model
16 from tools.tracking_model import Tracking_Model
17 from tools.incident_evaluator import Evaluate_Incidents
18 from tools.performance_evaluator import Evaluate_Performance
19 import argparse
20 from tools.visualize_objects import draw_rectangle, draw_text
       , draw_line
21
22 parser = argparse.ArgumentParser(
23     description="Real-time detection")
24 parser.add_argument("-m",
25                     "--model",
26                     help="Choose what model to use. If not
       provided, SSD will be used as default.",
27                     type=str)
```

```
28
29 parser.add_argument("-c",
30                     "--checkpoint",
31                     help="Choose what checkpoint number to
   use. If not provided, 3 will be used as default.",
32                     type=str)
33
34 parser.add_argument("-p",
35                     "--pretrained",
36                     help="Choose whether to use a pre-trained
    model or not. 1 = True, 0 = False (0 is default).",
37                     type=str)
38
39 parser.add_argument("-s",
40                     "--skip_frames",
41                     help="Choose how many frames should be
   skipped.",
42                     type=int)
43
44 parser.add_argument("-r",
45                     "--resize",
46                     help="Define a scale factor to resize the
    input image.",
47                     type=float)
48
49 parser.add_argument("-t",
50                     "--tracking",
51                     help="Choose what model to use. If not
   provided, DeepSort will be used as default.",
52                     type=str)
53
54 parser.add_argument("-f",
55                     "--file",
56                     help="A file will be saved with the
   filename provided. Default is no file saved.",
57                     type=str)
58
59 parser.add_argument("-i",
60                     "--img_enh",
61                     help="Specify which image enhancement
   method. Default is none.",
62                     type=str)
63
64 args = parser.parse_args()
65
66
67 def main():
68     datasets = []
69     org_path = r'.\\data\\Incidents\\Video'
```

```
70     for i in range(1, 13):
71         image_dir1 = org_path + str(i) + "\\images\\"
72         anno_path1 = org_path + str(i) + "\\annotations.json"
73         dataset_name = "self_annotated" + str(i)
74         datasets.append({"dataset": dataset_name, "images":
    image_dir1, "annotations": anno_path1})
75 
76     model_filename = os.path.join(PATH_TO_THIS_FILE, 'tools/
    model_data/mars-small128.pb')
77 
78     paths = {
79         "CHECKPOINT_PATH": "./training/models/ssd_mobnet/",
80         "PIPELINE_CONFIG": "./training/models/ssd_mobnet/
    pipeline.config",
81         "LABELMAP": "./training/annotations/label_map.pbtxt",
82         "DEEPSORT_MODEL": model_filename
83     }
84 
85     image_enhancement_methods = ["gray_linear", "
    gray_nonlinear", "he", "retinex_ssr", "retinex_msr", "mask
    "]
86     models = ["ssd_mobnet", "faster_rcnn", "yolov5", "
    yolov5_trained", "efficientdet"]
87     classes = {"car": "1", "truck": "2", "bus": "3", "bike":
    "4", "person": "5", "motorbike": "6"}
88 
89     model_name = "yolov5"
90     if args.model in models:
91         paths["CHECKPOINT_PATH"] = "./training/models/" +
    args.model + "/"
92         paths["PIPELINE_CONFIG"] = "./training/models/" +
    args.model + "/pipeline.config"
93         model_name = args.model
94 
95     tracking_model_name = "DeepSort"
96     if args.tracking:
97         tracking_model_name = args.tracking
98 
99     ckpt_number = "3"
100    if args.checkpoint is not None:
101        ckpt_number = args.checkpoint
102 
103    filename = ""
104    if args.file is not None:
105        filename = args.file
106 
107    image_enhancement = "None"
108    if args.img_enh is not None and args.img_enh in
    image_enhancement_methods:
```

```
109         image_enhancement = args.img_enh
110 
111     if args.pretrained == "1":
112         paths["CHECKPOINT_PATH"] = "./training/pre-trained-
    models/" + args.model + "/checkpoint/"
113         paths["PIPELINE_CONFIG"] = "./training/pre-trained-
    models/" + args.model + "/pipeline.config"
114         paths["LABELMAP"] = "./training/annotations/
    mscoco_label_map.pbtxt"
115         model_name = "Pretrained"
116         ckpt_number = "0"
117         classes = {"car": "3", "truck": "8", "bus": "6", "
    bike": "2", "person": "1", "motorbike": "4"}
118 
119     skip_frames = 1
120     if args.skip_frames:
121         skip_frames = int(args.skip_frames)
122 
123     resize = 1
124     if args.resize:
125         resize = float(args.resize)
126 
127     model = Detection_Model(model_name, classes, paths,
    ckpt_number)
128     tracker_model = Tracking_Model(paths["DEEPSORT_MODEL"],
    tracker_type=tracking_model_name)
129     evaluater = Evaluate_Incidents(classes)
130     pe = Evaluate_Performance("Images", datasets, classes,
    model, tracker_model)
131 
132     frame_number = 0
133     while True:
134         ret, frame, new_video, mask = pe.read(resize)
135         frame_number +=1
136         if frame_number % skip_frames != 0:
137             continue
138 
139         if ret:
140             frame = pe.image_enhancement(frame,
    image_enhancement, mask)
141         else:
142             print('Video has ended!')
143             break
144 
145         if new_video:
146             new_tracking_model = Tracking_Model(paths["
    DEEPSORT_MODEL"], tracker_type=tracking_model_name)
147             pe.tracking_model = new_tracking_model
148 
```

```python
        pe.detect_and_track(frame)

        evaluater.purge(frame_number)

        for track in pe.get_tracks():
            if not track.is_confirmed() or track.time_since_update > 1:
                continue

            color, text, current_point, next_point = evaluater.evaluate(track, frame_number)

            pe.performance(track, text)

            draw_rectangle(frame, track, color)
            draw_text(frame, track, text)
            if current_point and next_point:
                draw_line(frame, current_point, next_point)


        pe.status()

        result = np.asarray(frame)
        result = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)
        cv2.imshow("Output Video", result)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    cv2.destroyAllWindows()

    summary = pe.summary()
    print(summary)
    if filename != "":
        output_file = "./data/output/" + filename + ".txt"
        with open(output_file, "w") as file:
            output = f"Image enhancement: {image_enhancement}\n"
            output += f"Detection: {model_name}\n"
            output += f"Tracking: {tracking_model_name}\n"
            output += summary
            file.write(output)

if __name__ == '__main__':
    main()
```

Listing B.1: Code excerpt from the file run.py

```python
import os
import tensorflow as tf
import torch
```

```
import numpy as np
from object_detection.utils import label_map_util
#from object_detection.utils import visualization_utils as viz_utils
from object_detection.builders import model_builder
from object_detection.utils import config_util


"""
********************************************************************************
The function "detect_fn" below is taken from the source below:

Title: TFODCourse
File: 2. Training and Detection.ipynb
Author: Nicholas Renotte
Date: 03.04.2021
Code version: 1.0
Availability: https://github.com/nicknochnack/TFODCourse

********************************************************************************
"""
@tf.function
def detect_fn(image, detection_model):
    image, shapes = detection_model.preprocess(image)
    prediction_dict = detection_model.predict(image, shapes)
    detections = detection_model.postprocess(prediction_dict, shapes)
    return detections


class Detection_Model:
    def __init__(self, model_type, classes, paths={}, ckpt_number=3):
        self.model_type = model_type
        self.classes = classes
        self.class_ids = classes
        self.paths = paths
        self.ckpt_no = 'ckpt-' + str(ckpt_number)
        self.configs = None
        self.ckpt = None
        self.category_index = None
        self.CONFIDENCE_LEVEL = 0.6  # Confidence level for the TensorFlowAPI models

        self.model = None
```

```
46         self.init_model()
47
48     @property
49     def class_ids(self):
50         return self._class_ids
51
52     @class_ids.setter
53     def class_ids(self, classes):
54         class_ids = {}
55         for class_name in classes:
56             id = classes[class_name]
57             class_ids[id] = class_name
58         self._class_ids = class_ids
59
60     def init_model(self):
61         if self.model_type == "yolov5":
62             model = torch.hub.load('ultralytics/yolov5', '
   yolov5x')
63             # YOLOv5 stores and use cache by default. Use the
    line below instead if there are any problems with cache.
64             # model = torch.hub.load('ultralytics/yolov5', '
   yolov5x', force_reload=True)
65             self.model = model
66         elif self.model_type == "yolov5_trained":
67             model = torch.hub.load('ultralytics/yolov5', '
   custom', path='training/yolov5/yolov5/runs/train/
   yolov5x_trained/weights/best.pt')
68             self.model = model
69         else:
70             """
71
   *************************************************************************
72             The code below until the END statement is taken
   from the source below:
73
74             Title: TFODCourse
75             File: 2. Training and Detection.ipynb
76             Author: Nicholas Renotte
77             Date: 03.04.2021
78             Code version: 1.0
79             Availability: https://github.com/nicknochnack/
   TFODCourse
80
81
   *************************************************************************
82             """
83             gpus = tf.config.experimental.
```

```
list_physical_devices('GPU')
84            for gpu in gpus:
85                tf.config.experimental.set_memory_growth(gpu,
 True)
86
87            self.configs = config_util.
get_configs_from_pipeline_file(self.paths['PIPELINE_CONFIG
'])
88            self.model = model_builder.build(model_config=
self.configs['model'], is_training=False)
89
90            self.ckpt = tf.compat.v2.train.Checkpoint(model=
self.model)
91            self.ckpt.restore(os.path.join(self.paths['
CHECKPOINT_PATH'], self.ckpt_no)).expect_partial()
92            self.category_index = label_map_util.
create_category_index_from_labelmap(self.paths['LABELMAP'
])
93            """
94            END
95            """
96
97     def detect(self, frame, w=0, h=0):
98         return_object = {"frame": frame, "boxes": [], "scores
": [], "object_classes": []}
99
100        if self.model_type == "yolov5" or self.model_type ==
"yolov5_trained":
101            results = self.model(frame)
102            df = results.pandas().xyxy[0]
103            for row in df.itertuples():
104                obj_class = str(row[7]).lower()
105                if obj_class not in self.classes:
106                    continue
107                return_object["boxes"].append([float(row[1]),
 float(row[2]), (float(row[3])-float(row[1])), (float(row
[4])-float(row[2]))])
108                return_object["scores"].append(float(row[5]))
109                return_object["object_classes"].append(
obj_class)
110        else:
111            image_np = np.array(frame)
112            input_tensor = tf.convert_to_tensor(np.
expand_dims(image_np, 0), dtype=tf.float32)
113            detections = detect_fn(input_tensor, self.model)
114
115            num_detections = int(detections.pop('
num_detections'))
116            detections = {key: value[0, :num_detections].
```

```
    numpy()
117                             for key, value in detections.items()}
118             detections['num_detections'] = num_detections
119 
120             detections['detection_classes'] = detections['
    detection_classes'].astype(np.int64)
121 
122             for i, score in enumerate(detections["
    detection_scores"]):
123                 if float(score) >= self.CONFIDENCE_LEVEL:
124                     x1 = float(detections['detection_boxes'][
    i][1]) * float(w)
125                     x2 = float(detections['detection_boxes'][
    i][3]) * float(w)
126                     y1 = float(detections['detection_boxes'][
    i][0]) * float(h)
127                     y2 = float(detections['detection_boxes'][
    i][2]) * float(h)
128                     class_id = str(int(detections['
    detection_classes'][i]) + 1)
129                     obj_class = self.class_ids.get(class_id)
130 
131                     return_object["boxes"].append([x1, y1, (
    x2-x1), (y2-y1)])
132                     return_object["scores"].append(float(
    score))
133                     return_object["object_classes"].append(
    obj_class)
134 
135         return return_object
```

Listing B.2: Code excerpt from the file detection_model.py

```
1 import os
2 
3 os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
4 import tensorflow as tf
5 physical_devices = tf.config.experimental.
    list_physical_devices('GPU')
6 if len(physical_devices) > 0:
7     tf.config.experimental.set_memory_growth(physical_devices
    [0], True)
8 import numpy as np
9 
10 from deep_sort import preprocessing, nn_matching
11 from deep_sort.detection import Detection
12 from deep_sort.tracker import Tracker
13 from helpers import generate_detections as gdet
14 
15 
```

```
16 class Tracking_Model:
17     def __init__(self, model_filename, tracker_type="DeepSort
   ", max_cosine_distance=0.4, nn_budget=None,
   nms_max_overlap=1.0):
18         self.model_filename = model_filename
19         self.max_cosine_distance = max_cosine_distance
20         self.nn_budget = nn_budget
21         self.nms_max_overlap = nms_max_overlap
22         self.encoder = None
23         self.tracker = None
24         self.tracker_type = tracker_type
25         self.init_tracker()
26 
27     def init_tracker(self):
28         if self.tracker_type == "DeepSort":
29             """
30 
   ********************************************************************
31             The code below until the END statement is taken
   from the source below:
32 
33             Title: yolov4-deepsort
34             File: preprocessing.py
35             Author: The AI Guy
36             Date: 21.08.2021
37             Code version: 1.0
38             Availability: https://github.com/theAIGuysCode/
   yolov4-deepsort
39 
40 
   ********************************************************************
41             """
42             encoder = gdet.create_box_encoder(self.
   model_filename, batch_size=1)
43             self.encoder = encoder
44             metric = nn_matching.
   NearestNeighborDistanceMetric("cosine", self.
   max_cosine_distance, self.nn_budget)
45             tracker = Tracker(metric)
46             """
47             END
48             """
49         else:
50             tracker = Simple_Tracker()
51         self.tracker = tracker
52 
53     def track(self, model_detections):
```

```
54         if self.tracker_type == "DeepSort":
55             """
56
    ***************************************************************************
57             The code below until the END statement is taken
    from the source below:
58
59             Title: yolov4-deepsort
60             File: preprocessing.py
61             Author: The AI Guy
62             Date: 21.08.2021
63             Code version: 1.0
64             Availability: https://github.com/theAIGuysCode/
    yolov4-deepsort
65
66
    ***************************************************************************
67             """
68             frame, boxes, scores, object_classes =
    model_detections["frame"], model_detections["boxes"],
    model_detections["scores"], model_detections["
    object_classes"]
69             bboxes = np.array(boxes)
70             scores = np.array(scores)
71             object_classes = np.array(object_classes)
72
73             features = self.encoder(frame, bboxes)
74             detections = [Detection(bbox, score, class_name,
    feature) for bbox, score, class_name, feature in zip(
    bboxes, scores, object_classes, features)]
75
76             boxes = np.array([d.tlwh for d in detections])
77             scores = np.array([d.confidence for d in
    detections])
78             classes = np.array([d.class_name for d in
    detections])
79             indices = preprocessing.non_max_suppression(boxes
    , classes, self.nms_max_overlap, scores)
80             detections = [detections[i] for i in indices]
81
82             self.tracker.predict()
83             self.tracker.update(detections)
84             """
85             END
86             """
87         else:
88             self.tracker.update(model_detections)
```

```
89
90      def get_tracks(self):
91          return self.tracker.tracks
92
93
94  class Simple_Tracker:
95      def __init__(self):
96          self.vehicle_count = 0
97          self.all_tracks = []
98          self.tracks = []
99          self.age = 0
100
101     def calculate_IoU(self, new_detection, old_detection):
102         if new_detection[0] > old_detection[0]:
103             x_min = new_detection[0]
104         else:
105             x_min = old_detection[0]
106         if new_detection[1] > old_detection[1]:
107             y_min = new_detection[1]
108         else:
109             y_min = old_detection[1]
110         if new_detection[2] < old_detection[2]:
111             x_max = new_detection[2]
112         else:
113             x_max = old_detection[2]
114         if new_detection[3] < old_detection[3]:
115             y_max = new_detection[3]
116         else:
117             y_max = old_detection[3]
118
119         intersection_area = (x_max - x_min) * (y_max - y_min)
120         if intersection_area < 0 or (x_max - x_min) < 0 or (
    y_max - y_min) < 0:
121             return 0
122
123         union_area = ((old_detection[2] - old_detection[0]) *
     (old_detection[3] - old_detection[1])) + ((new_detection
    [2] - new_detection[0]) * (new_detection[3] -
    new_detection[1])) - intersection_area
124
125         IoU = intersection_area / union_area
126         return IoU
127
128     def update(self, model_detections):
129         detected_objects, object_classes = model_detections["
    boxes"], model_detections["object_classes"]
130
131         self.tracks = []
132         for detected_object, object_class in zip(
```

```
    detected_objects, object_classes):
133             detected_object = [detected_object[0],
    detected_object[1], detected_object[0]+detected_object[2],
     detected_object[1]+detected_object[3]]
134             for track in self.all_tracks:
135                 bboxes = track.to_tlbr()
136                 IoU = self.calculate_IoU(detected_object,
    bboxes)
137                 if IoU > 0.5:
138                     track.boxes = detected_object
139                     track.class_name = object_class
140                     track.age = self.age
141                     break
142             else:
143                 self.vehicle_count += 1
144                 track = Simple_Track(detected_object, self.
    vehicle_count, object_class, self.age)
145                 self.all_tracks.append(track)
146             self.tracks.append(track)
147 
148         for track in self.all_tracks:
149             if self.age - track.age > 10:
150                 del track
151         self.age += 1
152 
153 
154 class Simple_Track:
155     def __init__(self, boxes, track_id, class_name, age):
156         self.time_since_update = 0
157         self.boxes = boxes
158         self.track_id = track_id
159         self.class_name = class_name
160         self.age = age
161 
162     def get_class(self):
163         return self.class_name
164 
165     def is_confirmed(self):
166         return True
167 
168     def to_tlbr(self):
169         return self.boxes
```

Listing B.3: Code excerpt from the file tracking_model.py

```
1 import math
2 
3 
4 class Evaluate_Incidents:
```

```
5      def __init__(self, classes, colors=None, driving_direction=None):
6          self.classes = classes
7          self.colors = colors
8          self.objects = {}
9          self.driving_direction = driving_direction
10         self.TTL = 240  # Number of frames before a track is removed
11         self.PF = 2#7  # PF = Previous Frame: Number of frames used to determine direction
12         self.STOPPED_DISTANCE = 3  # Distance in number of pixels from current to previous frame to determine stopped vehicle
13         self.DIRECTION_THRESHOLD = 10  # Amount the x and y vectors can deviate when determining if vehicle is wrong-way driving
14         self.min_number_of_frames = 2#24  # How many frames must there be to evaluate stopped vehicle
15         self.update_number_of_frames = 2#12  # How often stopped vehicle should be evaluated
16
17     @property
18     def colors(self):
19         return self._colors
20
21     @colors.setter
22     def colors(self, colors):
23         colors_default = {"alarm": (255,128,128), "ok": (128,128,255)}
24         if colors and colors.get("alarm") and colors.get("ok"):
25             colors_default = colors
26         self._colors = colors_default
27
28     @property
29     def driving_direction(self):
30         return self._driving_direction
31
32     @driving_direction.setter
33     def driving_direction(self, driving_direction):
34         # Driving direction should be defined with an upstream and downstream direction
35         # Each direction should be defined as a vector: [x, y]
36         if driving_direction is None:
37             driving_direction = {"Upstream": [], "Downstream": []}
38         if driving_direction.get("Upstream") is None:
39             driving_direction["Upstream"] = []
```

```
40         if driving_direction.get("Downstream") is None:
41             driving_direction["Downstream"] = []
42         self._driving_direction = driving_direction
43 
44     def purge(self, frame_number):
45         if frame_number % 24 != 0:
46             return
47         dict_of_objects = self.objects.copy()
48         for object in dict_of_objects:
49             if dict_of_objects[object]["last_frame"] <
   frame_number - self.TTL:
50                 del self.objects[object]
51 
52     # Can be used to calculate direction based on center
   points from several frames
53     def simple_linear_regression(self, track_id, frame_number
   ):
54         track = self.objects[track_id]
55         n = len(track["center_points"])
56         if n <= 5:
57             return None, None
58 
59         current_point = (int(track["center_points"][-1][0]),
   int(track["center_points"][-1][1]))
60         if frame_number % 12 != 0:
61             direction = self.objects[track_id].get("direction
   ")
62             if direction:
63                 next_point_x = current_point[0] + direction["
   distance"]
64                 next_point_y = direction["alpha"] + direction
   ["beta"] * next_point_x
65                 next_point = (int(next_point_x), int(
   next_point_y))
66 
67                 return current_point, next_point
68 
69         if n > 10:
70             n = 10
71         center_points = track["center_points"][-n:]
72 
73         x_sum = 0
74         y_sum = 0
75         for center_point in center_points:
76             x_sum += center_point[0]
77             y_sum += center_point[1]
78 
79         x_mean = x_sum / n
80         y_mean = y_sum / n
```

```
        numerator = 0
        denominator = 0
        for center_point in center_points:
            x = center_point[0]
            y = center_point[1]
            numerator = (x - x_mean) * (y - y_mean)
            denominator = (x - x_mean) ** 2

        try:
            beta = numerator / denominator
        except Exception as e:
            print(e)
            beta = 0

        alpha = y_mean - beta * x_mean

        d = 1
        if (center_points[-1][0] - center_points[-2][0]) < 0:
            d = -1
        distance = d * math.sqrt((center_points[-1][0] -
    center_points[-2][0])**2 + (center_points[-1][1] -
    center_points[-2][1])**2)
        next_point_x = center_points[-1][0] + distance
        next_point_y = alpha + beta * next_point_x
        next_point = (int(next_point_x), int(next_point_y))

        self.objects[track_id]["direction"] = {"alpha": alpha
    , "beta": beta, "distance": distance}
        return current_point, next_point

    # Can be used to calculate direction based on center
    points from current and previous frame
    def simple_direction(self, track_id, frame_number):
        track = self.objects[track_id]
        n = len(track["center_points"])
        if n <= 8:
            return None, None

        current_point = (int(track["center_points"][-1][0]),
    int(track["center_points"][-1][1]))
        if frame_number % 12 != 0:
            direction = self.objects[track_id].get("direction
    ")
            if direction:
                x_vector = direction["x_vector"]
                y_vector = direction["y_vector"]
                length = direction["length"]
                next_point = (int(current_point[0] + x_vector
```

```
 * length), int(current_point[1] + y_vector * length))
124
125                 return current_point, next_point
126
127         previous_point = track["center_points"][-self.PF]
128
129         x_vector = current_point[0] - previous_point[0]
130         y_vector = current_point[1] - previous_point[1]
131         length_vector = math.sqrt(x_vector**2 + y_vector**2)
132         try:
133             x_vector /= length_vector
134             y_vector /= length_vector
135         except Exception as e:
136             print(e)
137             return None, None
138
139         length= 50
140
141         next_point = (int(current_point[0] + x_vector *
    length), int(current_point[1] + y_vector * length))
142
143         self.objects[track_id]["direction"] = {"length":
    length, "x_vector": x_vector, "y_vector": y_vector}
144         return current_point, next_point
145
146     def pedestrian(self, class_name):
147         if class_name == "person":
148             return True
149         return False
150
151     def stopped_vehicle(self, track_id, frame_number):
152         track = self.objects[track_id]
153         n = len(track["center_points"])
154         if n <= self.min_number_of_frames:
155             return False
156
157         if frame_number % self.update_number_of_frames != 0:
158             stopped = self.objects[track_id].get("stopped")
159             if stopped:
160                 return True
161             return False
162
163         current_point = (int(track["center_points"][-1][0]),
    int(track["center_points"][-1][1]))
164         previous_point = track["center_points"][-self.PF]
165
166         distance = math.sqrt((current_point[0] -
    previous_point[0])**2 + (current_point[1] - previous_point
    [1])**2)
```

```
167
168         if distance <= self.STOPPED_DISTANCE:
169             self.objects[track_id]["stopped"] = True
170             return True
171         self.objects[track_id]["stopped"] = False
172         return False
173
174     def wrong_way_driving(self, track_id, frame_number,
    current_point, next_point, lane="Upstream"):
175         if len(self.driving_direction.get(lane)) <= 0:
176             return False
177
178         track = self.objects[track_id]
179         n = len(track["center_points"])
180         if n <= self.min_number_of_frames:
181             return False
182
183         if frame_number % self.update_number_of_frames != 0:
184             wrong_way = self.objects[track_id].get("wrong_way
    ")
185             if wrong_way:
186                 return True
187             return False
188
189         vehicle_direction = [next_point[0] - current_point
    [0], next_point[1] - current_point[1]]
190         lane_direction = self.driving_direction.get(lane)
191
192         if abs(lane_direction[0] - vehicle_direction[0]) <
    self.DIRECTION_THRESHOLD and abs(lane_direction[1] -
    vehicle_direction[1]) < self.DIRECTION_THRESHOLD:
193             self.objects[track_id]["wrong_way"] = False
194             return False
195         self.objects[track_id]["wrong_way"] = True
196         return True
197
198     def evaluate(self, track, frame_number):
199         class_name = track.get_class()
200         text = f"{class_name} - {track.track_id}"
201         color = self.colors["ok"]
202         bbox = track.to_tlbr()
203         center_point = ((int(bbox[0]) + (int(bbox[2]) - int(
    bbox[0])) / 2), int(bbox[1]) + (int(bbox[3]) - int(bbox
    [1])) / 2)
204         if track.track_id in self.objects:
205             self.objects[track.track_id]["center_points"].
    append(center_point)
206             self.objects[track.track_id]["last_frame"] =
    frame_number
```

```
207         else:
208             self.objects[track.track_id] = {"center_points":
    [center_point], "last_frame": frame_number}
209
210         # Used to determine vehicle direction:
211         #   Current_point is the current center location of
    the vehicle
212         #   Next point is calculated by creating a vector
    from the current center point and the previous center
    point, and then multiplying it with a length. (Used to
    draw an arrow in the vehicle direction)
213         current_point, next_point = self.simple_direction(
    track.track_id, frame_number)
214
215         if self.pedestrian(class_name):
216             color = self.colors["alarm"]
217             text = "INCIDENT: Pedestrian"
218             current_point, next_point = None, None
219         elif self.stopped_vehicle(track.track_id,
    frame_number):
220             color = self.colors["alarm"]
221             text = "INCIDENT: Stopped vehicle"
222             current_point, next_point = None, None
223         elif self.wrong_way_driving(track.track_id,
    frame_number, current_point, next_point):
224             color = self.colors["alarm"]
225             text = "INCIDENT: Wrong-way driver"
226
227         return color, text, current_point, next_point
```

Listing B.4: Code excerpt from the file incident_evaluator.py

```
1 import json
2 import cv2
3 import os
4 import time
5 import numpy as np
6 from helpers.retinex import SSR
7 from helpers.retinex import MSR
8
9 class Evaluate_Performance:
10     def __init__(self, type, dataset_path, classes,
    detection_model, tracking_model):
11         self.vid = None
12         self.width = 0
13         self.height = 0
14         self.scale = 1
15         self.entries = []
16         self.next_entry_index = 0
17         self.type = type
```

```
18         self.detected_objects = []
19         self.detected_objects_previous = {}
20         self.dataset_paths = dataset_path
21         self.datasets = {}
22         self.classes = classes
23         self.detection_model = detection_model
24         self.tracking_model = tracking_model
25         self.current_video = ""
26         self.prepare()
27 
28         self.detection_time_current = 0
29         self.tracking_time_current = 0
30         self.total_time_current = 0
31         self.fps_current = 0
32 
33         self.image_enhancement_current = 0
34         self.mean_image_enhancement_time = 0
35 
36         self.mean_detection_time = 0
37         self.min_detection_time = -1
38         self.max_detection_time = 0
39 
40         self.mean_tracking_time = 0
41         self.min_tracking_time = -1
42         self.max_tracking_time = 0
43 
44         self.mean_total_time = 0
45         self.min_total_time = -1
46         self.max_total_time = 0
47 
48         self.missed_detections = 0
49         self.total_number_of_real_detections = 0
50         self.total_number_of_valid_detections = 0
51         self.total_number_of_valid_detections_adjusted = 0
52         self.false_positives_detections = 0
53         self.false_positives_detections_previous = 0
54 
55         self.missed_tracks = 0
56 
57         self.detection_accuracy = 0
58         self.detection_accuracy_adjusted = 0
59         self.tracking_accuracy = 0
60         self.tracking_id_switches = 0
61         self.tracking_id_duplicates = 0
62         self.incident_accuracy = 0
63         self.missed_incidents = 0
64         self.false_alarms = 0
65 
66         self.mean_fps = 0
```

```
67         self.min_fps = -1
68         self.max_fps = 0
69 
70         self.number_of_frames = 0
71 
72     @property
73     def dataset_paths(self):
74         return self._dataset_paths
75 
76     @dataset_paths.setter
77     def dataset_paths(self, datasets):
78         dataset_paths = {}
79         for dataset in datasets:
80             images = dataset.get("images")
81             annotations = dataset.get("annotations")
82             dataset_name = dataset.get("dataset")
83 
84             video = dataset.get("video")
85             if video:
86                 dataset_paths["video"] = video
87             elif images is None or annotations is None or
   dataset_name is None:
88                 continue
89             else:
90                 dataset_paths[dataset_name] = {"images":
   images, "annotations": annotations}
91 
92         self._dataset_paths = dataset_paths
93 
94     def prepare(self):
95         if self.type == "Video":
96             self.vid = cv2.VideoCapture(self.dataset_paths.
   get("video"))
97             self.width = int(self.vid.get(cv2.
   CAP_PROP_FRAME_WIDTH))
98             self.height = int(self.vid.get(cv2.
   CAP_PROP_FRAME_HEIGHT))
99         elif self.type == "Images":
100            for dataset_name in self.dataset_paths:
101                try:
102                    if "self_annotated" in dataset_name:
103                        self.prepare_self_annotated(
   dataset_name)
104                except Exception as e:
105                    print(e)
106                    self.datasets[dataset_name] = {"entries":
    []}
107            self.prepare_all_entries()
108 
```

```
109     def prepare_self_annotated(self, dataset_name="
    self_annotated"):
110         dataset = self.dataset_paths.get(dataset_name)
111         if dataset is None:
112             return
113         anno_path = dataset.get("annotations")
114         img_path = dataset.get("images")
115         mask_path = anno_path.replace("annotations.json", "
    mask.png")
116 
117         if anno_path is None:
118             return
119 
120         with open(anno_path, "r") as annotations:
121             data = json.load(annotations)
122 
123         annotation_classes_path = anno_path.replace("
    annotations", "classes")
124         with open(annotation_classes_path, "r") as
    annotation_classes:
125             annotation_classes = json.load(annotation_classes
    )
126 
127         images_list = {"entries": []}
128         for i, img in enumerate(data):
129             if i <= 0:
130                 continue
131             filename = img
132             row = {"images_path": img_path, "filename":
    filename, "objects": [], "mask_path": mask_path }
133 
134             for object in data[img]['instances']:
135 
136                 info = {}
137                 for class_ in annotation_classes:
138                     if class_["id"] == object["classId"]:
139                         class_name = class_["name"]
140 
141                         for object_attribute in object["
    attributes"]:
142                             for attribute_group in class_["
    attribute_groups"]:
143                                 if object_attribute["groupId"
    ] == attribute_group["id"]:
144                                     for attribute_ in
    attribute_group["attributes"]:
145                                         if attribute_["id"]
    == object_attribute["id"]:
146                                             info[
```

```
attribute_group["name"]] = attribute_["name"]
147
148                 if class_name == "people":
149                     class_name = "person"
150
151                 if class_name not in self.classes:
152                     continue
153
154                 x1 = float(object["points"]["x1"])
155                 y1 = float(object["points"]["y1"])
156                 x2 = float(object["points"]["x2"])
157                 y2 = float(object["points"]["y2"])
158
159                 row["objects"].append({"class": class_name, "
class_id": self.classes.get(class_name),  "x1": x1, "y1":
y1, "x2": x2, "y2": y2, "info": info})
160
161             images_list["entries"].append(row)
162
163         images_list['entries'] = sorted(images_list['entries'
], key = lambda i: i['filename'])
164         self.datasets[dataset_name] = images_list
165
166     def prepare_all_entries(self):
167         if self.type != "Images":
168             return
169
170         print("\nEntries:")
171         classes = {}
172         number_of_objects = 0
173         entries = []
174         for dataset in self.datasets:
175             for entry in self.datasets[dataset]["entries"]:
176                 entries.append(entry)
177                 number_of_objects += len(entry["objects"])
178                 for obj in entry["objects"]:
179                     if obj["class"] in classes:
180                         classes[obj["class"]] += 1
181                     else:
182                         classes[obj["class"]] = 1
183
184         print(f"Number of files: {len(entries)}")
185         print(f"Number of objects: {number_of_objects}")
186         for obj_class in classes:
187             print(f" - {obj_class}: {classes[obj_class]}")
188         self.entries = entries
189
190     def performance(self, track, text):
191         bbox = track.to_tlbr()
```

```
192         object_class = track.get_class()
193         track_id = track.track_id
194 
195         x1 = bbox[0]
196         y1 = bbox[1]
197         x2 = bbox[2]
198         y2 = bbox[3]
199 
200         incident = False
201         best_IoU = {"score": 0, "object": None, "real_object"
    : None}
202         for real_object in self.entries[self.next_entry_index
    -1]["objects"]:
203             real_object["x1"] *= self.scale
204             real_object["y1"] *= self.scale
205             real_object["x2"] *= self.scale
206             real_object["y2"] *= self.scale
207             if x1 > real_object["x1"]:
208                 x_min = x1
209             else:
210                 x_min = real_object["x1"]
211             if y1 > real_object["y1"]:
212                 y_min = y1
213             else:
214                 y_min = real_object["y1"]
215             if x2 < real_object["x2"]:
216                 x_max = x2
217             else:
218                 x_max = real_object["x2"]
219             if y2 < real_object["y2"]:
220                 y_max = y2
221             else:
222                 y_max = real_object["y2"]
223 
224             intersection_area = (x_max - x_min) * (y_max -
    y_min)
225             if intersection_area < 0 or (x_max - x_min) < 0
    or (y_max - y_min) < 0:
226                 continue
227 
228             union_area = ((real_object["x2"] - real_object["
    x1"]) * (real_object["y2"] - real_object["y1"])) + ((x2 -
    x1) * (y2 - y1)) - intersection_area
229             if union_area < 0:
230                 print(f"IA: {intersection_area}")
231                 print(f"UA: {union_area}")
232                 print(f"RO: x1 = {real_object['x1']}, y1 = {
    real_object['y1']}, x2 = {real_object['x2']}, y2 = {
    real_object['y2']}")
```

```
233                 print(f"DO: x1 = {x1}, y1 = {y1}, x2 = {x2},
    y2 = {y2}")
234                 raise ValueError
235 
236             IoU = intersection_area / union_area
237 
238             if (IoU - 1)**2 < (best_IoU["score"] - 1)**2:
239                 best_IoU["object"] = {"bbox": bbox, "class":
    object_class, "ID": track_id}
240                 best_IoU["real_object"] = real_object
241                 best_IoU["score"] = IoU
242 
243         if best_IoU["object"] is not None and best_IoU["score
    "] > 0.4:
244             if best_IoU["real_object"]['info']['status'] == "
    Incident":
245                 incident = True
246             self.detected_objects.append(best_IoU)
247         else:
248             self.false_positives_detections += 1
249 
250         if incident and ("Stopped vehicle" in text or "
    Pedestrian" in text):
251             self.incident_accuracy += 1
252         elif incident:
253             self.missed_incidents += 1
254         elif "Stopped vehicle" in text or "Pedestrian" in
    text:
255             self.false_alarms += 1
256 
257     def image_enhancement(self, frame, image_enhancement="",
    mask=None):
258         img_enh_start = time.time()
259         if image_enhancement == "gray_linear":
260             frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
261             frame = cv2.cvtColor(frame, cv2.COLOR_GRAY2RGB)
262         elif image_enhancement == "gray_nonlinear":
263             frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
264             gamma=2.0
265             invGamma = 1.0 / gamma
266             table = np.array([((i / 255.0) ** invGamma) * 255
267                 for i in np.arange(0, 256)]).astype("uint8")
268             frame = cv2.LUT(frame, table)
269             frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
270         elif image_enhancement == "he":
271             frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
272             frame = cv2.equalizeHist(frame)
273             frame = cv2.cvtColor(frame, cv2.COLOR_GRAY2RGB)
274         elif image_enhancement == "retinex_ssr":
```

```
275                 variance=300
276                 img_ssr=SSR(frame, variance)
277                 frame = cv2.cvtColor(img_ssr, cv2.COLOR_BGR2RGB)
278             elif image_enhancement == "retinex_msr":
279                 variance_list=[200, 200, 200]
280                 img_msr=MSR(frame, variance_list)
281                 frame = cv2.cvtColor(img_msr, cv2.COLOR_BGR2RGB)
282             elif image_enhancement == "mask":
283                 frame = cv2.bitwise_and(frame, frame, mask=mask)
284                 frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
285             else:
286                 frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
287 
288         img_enh_end = time.time()
289         self.image_enhancement_current = img_enh_end -
    img_enh_start
290         self.mean_image_enhancement_time += self.
    image_enhancement_current
291 
292         return frame
293 
294     def detect(self, frame):
295         detection_start = time.time()
296         model_detections = self.detection_model.detect(frame,
     self.width, self.height)
297         detection_end = time.time()
298         self.detection_time_current = detection_end -
    detection_start
299 
300         if self.detection_time_current < 10:
301             self.mean_detection_time += self.
    detection_time_current
302 
303         if self.min_detection_time == -1 or (self.
    detection_time_current < self.min_detection_time and self.
    detection_time_current > 0):
304             self.min_detection_time = self.
    detection_time_current
305         if self.detection_time_current > self.
    max_detection_time and self.detection_time_current < 10:
306             self.max_detection_time = self.
    detection_time_current
307 
308         return model_detections
309 
310     def track(self, model_detections):
311         track_start = time.time()
312         self.tracking_model.track(model_detections)
313         track_end = time.time()
```

```
314         self.tracking_time_current = track_end - track_start
315 
316         self.mean_tracking_time += self.tracking_time_current
317 
318         if (self.min_tracking_time == -1 or self.
    tracking_time_current < self.min_tracking_time) and self.
    tracking_time_current > 0:
319             self.min_tracking_time = self.
    tracking_time_current
320         if self.tracking_time_current > self.
    max_tracking_time:
321             self.max_tracking_time = self.
    tracking_time_current
322 
323     def detect_and_track(self, frame):
324         self.number_of_frames += 1
325         model_detections = self.detect(frame)
326         self.track(model_detections)
327 
328         self.total_time_current = self.detection_time_current
     + self.tracking_time_current + self.
    image_enhancement_current
329         self.mean_total_time += self.total_time_current
330 
331         if (self.min_total_time == -1 or self.
    total_time_current < self.min_total_time) and self.
    total_time_current > 0:
332             self.min_total_time = self.total_time_current
333         if self.total_time_current > self.max_total_time:
334             self.max_total_time = self.total_time_current
335 
336         self.fps_current = 1.0 / (self.total_time_current)
337         self.fps_current = round(self.fps_current, 3)
338 
339         self.mean_fps += self.fps_current
340         if (self.min_fps == -1 or self.fps_current < self.
    min_fps) and self.fps_current > 0:
341             self.min_fps = self.fps_current
342         if self.fps_current > self.max_fps:
343             self.max_fps = self.fps_current
344 
345     def read(self, resize=1):
346         if self.type == "Video":
347             return True, self.vid.read(), False, None
348         else:
349             frame = None
350             ret = False
351             new_video = False
352             mask = None
```

```
353             try:
354                 entry = self.entries[self.next_entry_index]
355                 path = entry["images_path"]
356                 mask_path = entry["mask_path"]
357                 image_path = os.path.join(path, '{}'.format(
    entry["filename"]))
358                 frame = cv2.imread(image_path)
359                 self.height, self.width, _ = frame.shape
360                 mask = cv2.imread(mask_path, 0)
361                 ret = True
362 
363                 if resize <= 1:
364                     self.scale = resize
365                     self.width = int(self.width * self.scale)
366                     self.height = int(self.height * self.
    scale)
367                     frame = cv2.resize(frame, (self.width,
    self.height), interpolation = cv2.INTER_AREA)
368                     mask = cv2.resize(mask, (self.width, self
    .height), interpolation = cv2.INTER_AREA)
369 
370                 if self.current_video != entry['images_path']
     and self.current_video != "":
371                     new_video = True
372                 self.current_video = entry['images_path']
373             except IndexError as e:
374                 print(e)
375             self.next_entry_index += 1
376 
377             return ret, frame, new_video, mask
378 
379     def get_tracks(self):
380         return self.tracking_model.get_tracks()
381 
382     def status(self):
383         detection_time = int((self.detection_time_current) *
    1000)
384         track_time = int(self.tracking_time_current * 1000)
385         print(f"\nFrame: {self.number_of_frames}")
386         print(f"FPS: {self.fps_current}")
387         print(f"IE time: {int(self.image_enhancement_current*
     1000)} ms")
388         print(f"Detection time: {detection_time} ms")
389         print(f"Tracking time: {track_time} ms")
390         print(f"Total time: {int(self.total_time_current*
    1000)} ms")
391 
392         avg_score = 0
393         avg_score_adjusted = 0
```

```
394         number_of_detections_adjusted = 0
395         number_of_correct_classes = 0
396         number_of_wrong_classes = 0
397         number_of_correct_ids = 0
398         number_of_wrong_ids = 0
399         number_of_duplicate_ids = 0
400         object_ids = []
401         print("Detected objects:")
402         for detected_object in self.detected_objects:
403             print(f"\t- {detected_object['object']['class']},
     {round(detected_object['score']*100, 2)} %")
404             avg_score += detected_object['score']
405             if detected_object["real_object"]['info']['
    occluded'] == "False":
406                 avg_score_adjusted += detected_object['score'
    ]
407                 number_of_detections_adjusted += 1
408             if detected_object["object"]["class"] ==
    detected_object["real_object"]["class"]:
409                 print("\t\t- Correct Class")
410                 number_of_correct_classes += 1
411             else:
412                 print("\t\t- Wrong Class")
413                 number_of_wrong_classes += 1
414 
415             if detected_object['real_object']['info']['ID']
    in self.detected_objects_previous:
416                 if self.detected_objects_previous[
    detected_object['real_object']['info']['ID']] ==
    detected_object['object']['ID']:
417                     if detected_object['real_object']['info'
    ]['ID'] in object_ids:
418                         print("\t\t- Duplicate ID")
419                         number_of_duplicate_ids += 1
420                     else:
421                         print("\t\t- Correct ID")
422                         number_of_correct_ids += 1
423                 else:
424                     print("\t\t- Wrong ID")
425                     if detected_object["real_object"]["class"
    ] != "person":
426                         number_of_wrong_ids += 1
427                     self.detected_objects_previous[
    detected_object['real_object']['info']['ID']] =
    detected_object['object']['ID']
428             else:
429                 self.detected_objects_previous[
    detected_object['real_object']['info']['ID']] =
    detected_object['object']['ID']
```

```
430             object_ids.append(detected_object['real_object'][
    'info']['ID'])
431 
432         self.tracking_accuracy += number_of_correct_ids
433         self.tracking_id_switches += number_of_wrong_ids
434         self.tracking_id_duplicates +=
    number_of_duplicate_ids
435 
436         self.detection_accuracy += avg_score
437         self.detection_accuracy_adjusted +=
    avg_score_adjusted
438         self.total_number_of_valid_detections += len(self.
    detected_objects)
439         self.total_number_of_valid_detections_adjusted +=
    number_of_detections_adjusted
440         if len(self.detected_objects): avg_score /= len(self.
    detected_objects)
441         if number_of_detections_adjusted > 0:
    avg_score_adjusted /= number_of_detections_adjusted
442         print(f"Average score: {round(avg_score*100, 2)} %")
443         print(f"Average score adjusted: {round(
    avg_score_adjusted*100, 2)} %")
444 
445         tmp_missed = 0
446         for real_object in self.entries[self.next_entry_index
    -1]["objects"]:
447             if real_object['info']['ID'] not in object_ids:
448                 self.missed_detections += 1
449                 tmp_missed += 1
450         self.total_number_of_real_detections += len(self.
    entries[self.next_entry_index-1]["objects"])
451 
452         print(f"Missed detections: {tmp_missed}")
453         try:
454             print(f"Missed detections: {round(100*tmp_missed/
    len(self.entries[self.next_entry_index-1]['objects']), 1)}
     %")
455         except Exception as e:
456             print(e)
457         print(f"False positive detections: {self.
    false_positives_detections - self.
    false_positives_detections_previous}")
458 
459         self.detected_objects = []
460 
461         self.false_positives_detections_previous = self.
    false_positives_detections
462 
463     def summary(self):
```

```
464         text = "\n"
465         try:
466             total_detections = self.
    total_number_of_valid_detections + self.
    false_positives_detections
467             text += f"Scale: {int(self.scale*100)} %\n"
468             text += f"Resolution: {int(self.width*self.scale)
    }x{int(self.height*self.scale)} px\n"
469             text += f"Mean image enhancement time: {int(1000
    * self.mean_image_enhancement_time / self.number_of_frames
    )} ms\n"
470             text += "\n"
471             text += f"Mean detection time: \t{int(1000 * self
    .mean_detection_time / self.number_of_frames)} ms\n"
472             text += f"Min detection time: \t{int(1000 * self.
    min_detection_time)} ms\n"
473             text += f"Max detection time: \t{int(1000 * self.
    max_detection_time)} ms\n"
474             text += "\n"
475             text += f"Mean tracking time: \t{int(1000 * self.
    mean_tracking_time / self.number_of_frames)} ms\n"
476             text += f"Min tracking time: \t\t{int(1000 * self
    .min_tracking_time)} ms\n"
477             text += f"Max tracking time: \t\t{int(1000 * self
    .max_tracking_time)} ms\n"
478             text += "\n"
479             text += f"Mean total time: \t{int(1000 * self.
    mean_total_time / self.number_of_frames)} ms\n"
480             text += f"Min total time: \t{int(1000 * self.
    min_total_time)} ms\n"
481             text += f"Max total time: \t{int(1000 * self.
    max_total_time)} ms\n"
482             text += "\n"
483             text += f"Mean fps: \t{round(self.mean_fps / self
    .number_of_frames, 1)}\n"
484             text += f"Min fps: \t{int(self.min_fps)}\n"
485             text += f"Max fps: \t{int(self.max_fps)}\n"
486             text += "\n"
487             text += f"False positive detections: \t{round
    (100*self.false_positives_detections/total_detections, 1)}
     %\n"
488             text += f"Missed detections: \t\t\t{round(100*
    self.missed_detections/self.
    total_number_of_real_detections, 1)} %\n"
489             text += "\n"
490             text += f"Detection accuracy: \t\t\t{round(100*
    self.detection_accuracy/self.
    total_number_of_valid_detections, 1)} %\n"
491             text += f"Detection accuracy adjusted: \t{round
```

```
    (100*self.detection_accuracy_adjusted/self.
    total_number_of_valid_detections_adjusted, 1)} %\n"
492             text += f"Tracking accuracy: \t\t\t\t{round(100*
    self.tracking_accuracy/(self.tracking_accuracy+self.
    tracking_id_switches+self.tracking_id_duplicates), 1)} %\n
    "
493             text += f"Tracking ID duplicates: \t\t{round(100*
    self.tracking_id_duplicates/(self.tracking_accuracy+self.
    tracking_id_switches+self.tracking_id_duplicates), 1)} %\n
    "
494             text += f"Tracking ID switches: \t\t\t{round(100*
    self.tracking_id_switches/(self.tracking_accuracy+self.
    tracking_id_switches+self.tracking_id_duplicates), 1)} %\n
    "
495             text += "\n"
496             text += f"Incident accuracy: \t{round(100*self.
    incident_accuracy/(self.incident_accuracy+self.
    missed_incidents), 1)} %\n"
497             text += f"Missed incidents: \t{round(100*self.
    missed_incidents/(self.incident_accuracy+self.
    missed_incidents), 1)} %\n"
498             text += f"False alarms: \t\t{round(100*self.
    false_alarms/total_detections, 1)} %\n"
499             text += "\n"
500             text += f"Total number of valid detections: {self
    .total_number_of_valid_detections}\n"
501             text += f"Total number of detections: {
    total_detections}\n"
502
503         except Exception as e:
504             print(e)
505
506         return text
```

Listing B.5: Code excerpt from the file performance_evaluator.py

```
1 import os
2 import io
3 import sys
4
5 PATH_TO_THIS_FILE = os.path.dirname(os.path.abspath(__file__)
   )
6 sys.path.insert(0, PATH_TO_THIS_FILE + '\\')
7 sys.path.insert(0, PATH_TO_THIS_FILE + '\\..\\')
8 sys.path.insert(0, PATH_TO_THIS_FILE + '\\..\\tensorflowapi\\
   ')
9 sys.path.insert(0, PATH_TO_THIS_FILE + '\\..\\tensorflowapi\\
   research\\')
10 sys.path.insert(0, PATH_TO_THIS_FILE + '\\..\\tensorflowapi\\
   research\\object_detection')
```

```
11
12 os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
13 import tensorflow.compat.v1 as tf
14 from PIL import Image
15 from object_detection.utils import dataset_util
16
17 from prepare_data import Prepare
18
19 def create_tf_example(example):
20     path = example["images_path"]
21     image_path = os.path.join(path, '{}'.format(example["
    filename"]))
22     with tf.gfile.GFile(image_path, 'rb') as fid:
23         encoded_jpg = fid.read()
24     encoded_jpg_io = io.BytesIO(encoded_jpg)
25     image = Image.open(encoded_jpg_io)
26     width, height = image.size
27
28     filename = example["filename"].encode('utf8')
29     image_format = b'jpg'
30
31     x1s = []
32     x2s = []
33     y1s = []
34     y2s = []
35     classes = []
36
37     for obj in example["objects"]:
38         x1s.append(obj['x1'] / width)
39         x2s.append(obj['x2'] / width)
40         y1s.append(obj['y1'] / height)
41         y2s.append(obj['y2'] / height)
42         classes.append(obj['class'].encode('utf8'))
43
44     tf_example = tf.train.Example(features=tf.train.Features(
    feature={
45         'image/height': dataset_util.int64_feature(height),
46         'image/width': dataset_util.int64_feature(width),
47         'image/filename': dataset_util.bytes_feature(filename
    ),
48         'image/source_id': dataset_util.bytes_feature(
    filename),
49         'image/encoded': dataset_util.bytes_feature(
    encoded_jpg),
50         'image/format': dataset_util.bytes_feature(
    image_format),
51         'image/object/bbox/xmin': dataset_util.
    float_list_feature(x1s),
52         'image/object/bbox/xmax': dataset_util.
```

```
    float_list_feature(x2s),
53          'image/object/bbox/ymin': dataset_util.
    float_list_feature(y1s),
54          'image/object/bbox/ymax': dataset_util.
    float_list_feature(y2s),
55          'image/object/class/text': dataset_util.
    bytes_list_feature(classes),
56          'image/object/class/label': dataset_util.
    int64_list_feature([1]),
57      }))
58      return tf_example
59
60
61 def main(_):
62      image_dir_night = r'..\\..\\data\\Training\\raw_night\\'
63      anno_path_night = r'..\\..\\data\\Training\\raw_night\\'
64
65      image_dir_raw = r'..\\..\\data\\Training\\raw\\'
66      anno_path_raw = r'..\\..\\data\\Training\\raw\\'
67
68      image_dir_kitti = r'..\\..\\data\\Training\\kitti\\'
69      anno_path_kitti = r'..\\..\\data\\Training\\kitti\\'
70
71      datasets = [ {"dataset": "other1", "images":
    image_dir_night, "annotations": anno_path_night},
72                   {"dataset": "other2", "images":
    image_dir_raw, "annotations": anno_path_raw},
73                   {"dataset": "other3", "images":
    image_dir_kitti, "annotations": anno_path_kitti}]
74
75      org_path = r'..\\..\\data\\Incidents\\Video'
76      for i in range(1, 13):
77          image_dir1 = org_path + str(i) + "\\images\\"
78          anno_path1 = org_path + str(i) + "\\annotations.json"
79          dataset_name = "self_annotated" + str(i)
80          datasets.append({"dataset": dataset_name, "images":
    image_dir1, "annotations": anno_path1})
81
82      train_test_distribution = 0.9
83      classes = {"car": "1", "truck": "2", "bus": "3", "bike":
    "4", "person": "5", "motorbike": "6"}
84
85      preparer = Prepare(train_test_distribution, datasets,
    classes)
86
87      output_path = "../annotations/"
88
89      train_output = output_path + "train.record"
90      writer = tf.python_io.TFRecordWriter(train_output)
```

```
91      for example in preparer.get_all_train_entries():
92          tf_example = create_tf_example(example)
93          writer.write(tf_example.SerializeToString())
94      writer.close()
95      print(f'Successfully created the TFRecord file: {
        train_output}')
96  
97      test_output = output_path + "test.record"
98      writer = tf.python_io.TFRecordWriter(test_output)
99      for example in preparer.get_all_test_entries():
100         tf_example = create_tf_example(example)
101         writer.write(tf_example.SerializeToString())
102 
103     print(f'Successfully created the TFRecord file: {
        test_output}')
104 
105 if __name__ == '__main__':
106     tf.app.run()
```

Listing B.6: Code excerpt from the file generate_records.py