Universitetet i Stavanger

Faculty of Science and Technology

BACHELOR'S THESIS

Study program/Specialization: Bachelor's degree in engineering / Data Science	Spring Semester, 2024 **Open** or ~~Restricted Access~~
Writer(s): Martin Sævareid Lauritsen, Alexander Kruke Bjugan	
Faculty supervisor: Naeem Khademi External supervisor(s):	
Thesis title: Using Deep Learning for Improved Automatic Incident Detection (AID) in Road Tunnels	
Credits: 20	
Keywords: deep learning, deep neural networks	Pagenumber: 76 + appendix: 79 Stavanger 15.05.2024

Abstract

Automatic Incident Detection (AID) systems in road tunnels is something that has gotten a large amount of focus because of high response times and the . This means that AID systems in road tunnels should be implemented in every tunnel, and should work without the need for humans to manually detect them. For this to be the case, existing incident detection systems need to be updated or upgraded. This is usually done with additional implementation of computer vision and/or radar. The use of radar technology is still in its very early stages, so this thesis will focus on the computer vision part.

Our thesis is a continuation of a previous thesis with closely the same goal, to improve and test different image enhancement methods, object detection methods and object tracking methods. It will take a different standpoint than to the previous one with different statistical evaluations and other methods. This will also have a greater implementation and look at queue detection. And it will conclude with a overview of the models and methods with statistics and limitations.

Contents

Chapter 1

Introduction

1.1 Background

In 2017 the Norwegian road authority presented their project for 0-goal of incidents in road tunnels [1].The consequences of incidents in road tunnels can often be more severe then normal accidents as they take place in confined spaces where they easily can escalate to larger accidents. For this reason it is important that we utilize strong and highly capable technology systems in tunnels, so that the "0-goal" can be achieved.

1.2 Automatic Incident Detection (AID)

Automatic Incident Detection (AID) systems are systems that automatically detect when incidents occur and immediately alerts the operator of traffic incidents. Several AID systems rely on congestion, vehicle speed, traffic data, such as flow, and so forth, to decide if an incident has occurred. This data is gathered by use of technologies such as inductive loops, CCTV, radar, Bluetooth, etc.
A AID system that has good performance should be able to detect most incidents that may occur in road tunnels. Incidents that appear in tunnels can be grouped in these categories:

- Stopped vehicles
- Wrong way drivers
- Pedestrians
- Objects on road
- Fire and smoke
- Queue

Moreover, AID systems are in general divided into five categories: comparative, statistical, traffic-model-based, artificial-intelligence-based, and mixed models. [2, p.1]

1.2.1 Comparative

Comparative systems use multiple thresholds depending on traffic to detect incidents. These thresholds are often based on an increase in a upstream loop detector and a decrease in a downstream loop detector following an incident. [3, p.1] California algorithms and filtering algorithms are some of the more popular ones. Take the California algorithm which tests for an incident using 3 tests or thresholds using occupancy on 2 loop detectors adjacent one another. [4, p.10] If upstream occupancy grows and downstream occupancy shrinks it could imply that a incident has taken place. [5, p.1-2]

1.2.2 Statistical

A statistical algorithm is an algorithm that aims to predict the flow of traffic in two steps. Firstly it uses historical data to predict traffic-flow parameters. It then takes the predicted traffic-flow parameters based on the historical data, and compares it to current gathered traffic-flow parameters. [5, p. 1-2] If the first value diverges to much from the currently measure value, an incident alarm is proclaimed. One of the earliest of these algorithms is the standard normal deviate model. It takes the mean and standard deviation to calculate standardized values for the traffic. When the traffic values diverge to much from the predicted values, then an incident alarm is proclaimed.[2, p. 1] Other big algorithms in this category include time series and filtering. [5, p. 1-2]

Standard Normal Deviate Algorithm
The Standard Normal Deviate Algorithm has proven to be a most effective AID system, and was used and tested in study done in Hong Kong[6]. This can be credited to it being easy to calibrate and having good transferability. The underlying principle being that a sudden change in a particular traffic parameter will be an indication of an incident. [6, p. 841]

1.2.3 Traffic-model-based

Traffic-model-based algorithms use advanced traffic-flow theories such as relationship between occupancy and volume to determine incidents. It uses the measurements from upstream and downstream loop detectors to determine traffic states like congested, uncongested and incident [5, p.] [3, p.]. An algorithm of this kind is the McMaster algorithm, and is based on the catastrophic theory. Because of traffic flow is an infinite dimension, nonlinear, stochastic, time variant and complicated dynamic system, it is difficult to specify. Hence it a traffic-model-based algorithm is rarely used to detect incidents.[5, p. 2]

1.2.4 Artificial-model-based

Artificial-model-based algorithms use historical data from both conditions with an incident and with no incident to classify traffic patterns. The increase in computational video image processing has made video-based accident detection a more and more viable option to standard loop detectors. [3, p. 1]

Fuzzy algorithms
Fuzzy algorithms are effective and helpful when data is difficult to collect or when there is not enough data. They use fuzzy logic, the concept of fuzzy boundary, and change in the occupancy or speed-density relations of two adjacent loop detectors. They are as effective as they are because of

their high robustness and their ability to overcome the boundary condition problem that normal threshold-based methods have inherited.[5, p. 2]

Neural Network Algorithms
Neural Network algorithms use historical data for training to recognize traffic patterns with incident and incident-free states. These algorithms are generally easier to use and better for real-time detection compared to model-based algorithms. They do however have downsides such as having a slow state of convergence, having difficult to understand operation meanings, caused by it being a black-box approach and them needing large historical datasets to be able to function sufficiently. Without large and wide datasets it will be no better than traditional algorithms.[5, p. 2]

Image-Based Processing Algorithms
Image-Based Processing algorithms extract information of traffic parameters from video sequence taken by video cameras by using computer vision and image-based processing technology, then verify and detect when traffic incident occur.[5, p. 2]

1.2.5 Mixed models

Mixed models use multiple different algorithms combined for detecting incidents. One of the most well known in this category is the Minnesota algorithm which combines statistical and comparative algorithms. [2, p. 1]

1.3 Challenges

There are is a lot of challenges to AID systems in road tunnels. How much light there is can affect the accuracy of detections a significant amount. It can introduce shadows and noise which in turn affects the calibration which can give false detections. The video camera lens can get dirty which affects contrast and loss in image quality, that also can give false detections. Outside interference is not the only factor either. If the video camera has low frame rate or resolution, then detection accuracy is further lowered.[7] Other factors that heavily implicates AID systems also include environmental factors. These environmental factors include snow, rain, shadows and glare. Because of these factors the system needs to be able to detect them and adjust for them.[8]

1.4 Objectives

The main objectives of automatic incident detection systems is to prevent incidents from happening, or if mentioned incident occurs, prevent them from escalating further. Because of the fast development of machine learning and computer vision, these systems have become easier to implement and can usually be installed in already existing infrastructure. Infrastructure that as an example has been using for basic information collecting with traffic monitoring purposes.[9]
Most of already existing systems primarily use simpler detection techniques such as inductive loops, while vision-based automatic incident detection still has great unexplored potential. The amount of AID systems being developed today is growing steadily with most of its focus being on highway roads and intersections, while focus on AID systems for use in tunnels is still lacking. Being based on a previous thesis, this thesis aims to further explore and improve current systems in a similar way to the previous one. That is, giving further recommendations for improving detection rate, decreasing false positives and general performance, with more statistics as base.

Chapter 2

Related Work

Approaches and methods to incident detection comes in many different flavors. AID systems today collect data from traffic cameras, dynamic sensors and static sensors. Dynamic sensors are commonly fitted to probe vehicles to generate a continuous source of data, while static sensors typically include inductive loops.[10] Traffic camera data can be used with machine learning and computer vision to further increase performance of current AID systems and reduce the amount of false alarms. Research has been done that show how a reduction in reaction time for incidents significantly reduce mortality rate and also reduces risk of further secondary incidents. It can also reduce the amounts of delay caused by incidents. [2, p. 1]
With the expanse of video surveillance, newer and better AID systems can be implemented. These new AID systems can better gather more information with the use of video, and can with that data figure out what counts as normal behavior. This normal behavior helps the system learn or detect anomalies. Anomalies usually rooted in the interactions between entities such as vehicles, pedestrians and environment. [11, p. 1] These anomalies come in three classes, point anomalies, contextual anomalies and collective anomalies. Point anomalies is usually a data point that strays far from the usual data distribution. An example of this type is a car that is stopped in a usually busy road. Contextual anomalies are related to data that could be normal in some context, but not in the current one. An example for this type of anomaly is a biker that is driving faster than the surrounding traffic. Collective anomalies are when a collection of data instances cause

an anomaly. This can for example be a group of people suddenly changing their behavior in short period of time. One of the biggest issues with detecting anomalies lies with how wide the boundary between normal and anomalous is, and where the line between them lies. The availability of collect training data and validation also greatly affects the performance of the anomaly detection. [11, p. 4]
Some studies in this field include the use of a crash detection framework that is based on three main components. Firstly the use of the Retinex image enhancement algorithm being used to enhance image quality. Then they use YOLOv3, which is a object detection algorithm know for its real-time performance, and that can detect vehicles, pedestrians and bicyclists. And lastly use a decision tree-based algorithm to determine various crash scenarios in mixed traffic flow environments. [12, p. 2]

A study on real-time wrong direction detection [13] has tested multiple detection methods and tracking methods with validation. In the study, methods such as Background Subtraction, optical flow, convolutional neural network (CNN)-based and different tracking methods was tested. Background Subtraction a technique that applies a foreground mask for static cameras. Background subtraction have already been implemented in computer vision libraries such as OpenCV, where they have 2 methods implemented. The k-nearest neighbour (KNN) algorithm and the Gaussian mixture-based background/foreground segmentation algorithm (MOG2). This method is very sensitive to changes in the lighting conditions, and would not be suited for the requirements of the study.
There is also the method of optical flow, which is the pattern learning of a moving object. Optical flow is a machine learning method with the idea being to presume that the color or brightness of a pixel remains consistent despite being shifted from one frame to another. It also assumes that the distance from frame-to-frame shifting is local or small. The most widely uses version of optical flow is called Lucas-Kanade method. The optical flow method suffers when brightness intensity happens or with object occlusion. The study ended up using a combination of YOLOv3 and a linear quadratic equation to detect and track vehicles, which showed excellent performance of 91% accuracy of wrong way driving. [13]

Chapter 3

Theory

3.1 Image enhancement methods

Image enhancement methods in computer vision are often functions or formulas created to manipulate pixel values such that detection become easier and more reliable.

3.1.1 Gray level transformation

Gray level image transformation is a image enhancement technique that transforms the gray value of pixels into other gray values by the use of a mathematical function. There are two distinct types of this method. These are gray linear transformations and gray non-linear transformations. Linear stretching is one of the fundamental methods in gray linear transformations and has the following function:

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

where $f(x, y)$ and $g(x, y)$ represent the input and output images, respectively, and C and R are the coefficients of the linear transformation. This transforms the dynamic range of the image to enhance brightness and contrast.

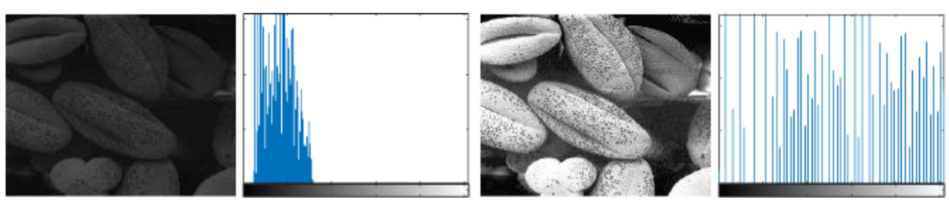
For non-linear transformations the main purpose is to enhance the image by the use of non-linear mathematical functions. A logarithmic transformation, for example, implies that the input image values and output image values have a logarithmic relationship for each pixel. This is quite useful in a case where the image is extremely dark as it can stretch the lower gray values pixels while compressing the dynamic range of higher gray value pixels. A classic formula of this kind is:

$$g(x, y) = log(1 + c \times f(x, y)) \tag{3.2}$$

where c is a control parameter. Some other functions of the non-linear kind include gamma functions. [14, p. 3-4]

3.1.2 Histogram equalization

A histogram is a graphical representation that offers a comprehensive view of an image's intensity distribution. It depicts pixel values, typically ranging from 0 to 255 on the X-axis, against the number of pixels on the Y-axis. Analyzing an image's histogram gives insight into its contrast, brightness and intensity distribution.[15] Histogram equalization is a image enhancement method that uses the cumulative distribution function (CDF) to adjust output gray levels to have a probability density function that resemble a uniform distribution. This can effectively make details that are hidden, because of high contrast and high dynamic range, reappear, and thus improve the visual effect of the input image. [14, p. 5]



(a) (b) (c) (d)

Figure 3.1: Histogram examples [14, p. 5]

3.1.3 Retinex

Retinex is a image enhancement technique that is designed to reduce the influence of illumination to enhance sharpness, color consistency, large dynamic range compression and high color fidelity of images. Based on figure 3.2 an image can be expressed as the product of a reflection component and an illumination component:

$$I(x, y) = R(x, y) \times L(x, y) \tag{3.3}$$

where $R(x, y)$ is the reflection component, and $L(x, y)$ is the reflective characteristics of the object surface. $I(x, y)$ is the received image. In figure 3.3 the general process of the Retinex algorithm is shown.

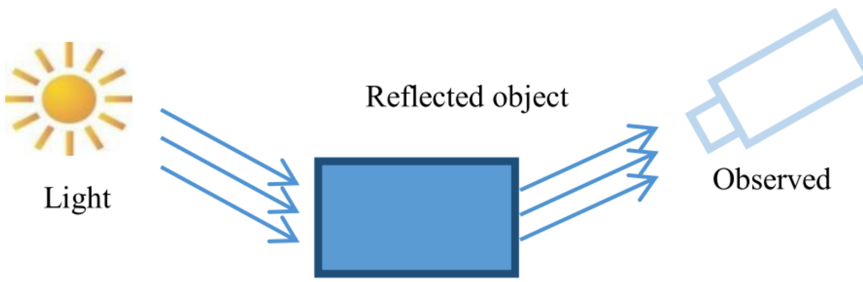


Figure 3.2: Light reflection model [14, p. 8]

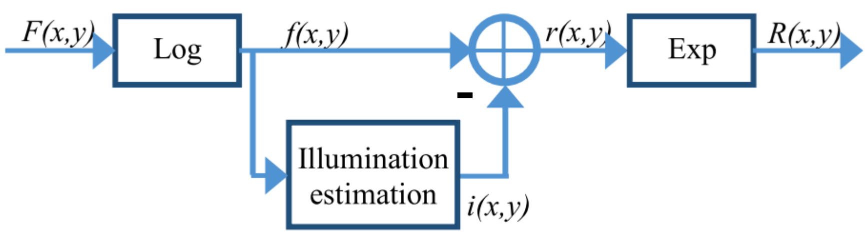


Figure 3.3: General process of the Retinex algorithm [14, p. 8]

Retinex_SSR

Retinex_SSR stands for single-scale Retinex, and is essentially an algorithm that obtains a reflection image by estimating the ambient brightness by this formula:

$$logR_i(x, j) = logI_i(x, y) - log[G(x, y) * I_i(x.y)] \tag{3.4}$$

In the formula I(x,y) represents the input image, R(x,y) represents the reflection image, i represents the various color channels, (x,y) represents the position of the pixel in the image, G(x,y) represents the Gaussian surround function, and * represents the convolution operator. The formula for the Gaussian surround function is:

$$G(x, y) = Ke^{\left(-\frac{x^2+y^2}{\sigma^2}\right)} \tag{3.5}$$

where σ is a scale parameter. [14, pp. 8–9]

Retinex_MSR

Retinex MSR, also called multiscale Retinex, is a method that extends the single-scale algorithm for maintaining balance between the dynamic range compression and color consistency. It is expressed like this:

$$MSR = logR_i(x, y) = \sum_{N}^{k=1} \omega_k logI_i(x, y) - log[G_k(x, y) * I_i(x, y)] \tag{3.6}$$

$$\sum_{k=1}^{N} \omega_k = 1 \tag{3.7}$$

where i represents the three color channels; k represents the Gaussian surround scales; N is the number of scales, generally 3; and the ω parameters are the scale weights. The biggest benefit of multiscale compared to single-scale is that it can take advantage of multiple scales. This gives the output enhanced details and contrast, and also gives better color consistency and improved visual effect. [14, p. 9]

3.1.4 Mask

Masking is an image enhancement method used to restrict object detections to a predetermined desired area of the input image. The this thesis masks were created in-part using the image alteration software called Krita [16] to draw black and white areas. White areas represents road while black areas represents tunnel walls and any other part where vehicle detection would not make sense.
The pseudo-code for masking would look something like this:

```
    if image_enhancement == "mask":
            frame = frame - mask
```

3.2 Artificial Neural Networks

Neural networks know more generally as Artificial Neural Networks (ANN) or Simulated Neural Networks (SNN) is key to deep learning algorithms. Taking inspiration from the functionality provided by biological neurons and how they signal each other, ANN's uses neurons to compute a output value based on the input value and the specific algorithm defined for that neuron. The accuracy of an Artificial neural network is dependent on the training conducted prior to its deployment. [17]

A Neural Network works by comprising neurons into layers and combining layers to form models which are known as Neural Networks. The layers used are what defines what type of neural network you are creating. An Artificial Neural Network is often compared with regular neural networks as they are built up by the same layer structure. These types of neural networks will contain the following layers:

1. One Input Layer
2. One or multiple hidden layers
3. One Output Layer

Each of these layers can contain multiple neurons where each neuron's output can connect to multiple other neurons in layers ahead of it[17].

3.2.1 Inner workings of an Artificial Neural Network

At the heart of each neuron is its own linear regression model. Comprised of input data, bias / thresholds, weights and output each neuron has its own formula that can be generalized like what we see in the equation (3.8):

$$\sum wixi + bias = w1x1 + w2x2 + w3x3 + bias \tag{3.8}$$

The general output would follow the same pattern shown in (3.9):

$$f(x) = 1if \sum w1x1 + b >= 0; 0if \sum w1x1 + b < 0 \tag{3.9}$$

After the initial input layers are constructed, individual weights are applied. Weights are an important concept in Neural Networks as they are the defining factor to how large or small of an impact each neuron has. Weights are applied to raise or lower the importance of a variable.
The output of each neuron are multiplied with their individual weights and summed to determine the output of a node.

Small Cost Function:

$$CF = \frac{1}{2}m \times (y - \hat{y}) \tag{3.10}$$

Expanded Cost Function:

$$CostFunction = \frac{1}{2}m \sum 129^{m}_{(i=1)} \times (y^i - \hat{y}^i)^2 \tag{3.11}$$

When building and training a neural network it is important to constantly get feedback on the accuracy and effectiveness of the algorithm. This is where the Cost function (3.10) and the expanded function (3.11) comes into play. The Cost Function can also be referred to as the "Mean Squared

Error". These functions use the following defined values to calculate the accuracy of the neural network model:

- i represents the sample index
- $\hat{y}$ is the model predicted value
- y real value
- m number of total samples

There are several ways one can train a neural network, one of the most common ways to do so is called "feedforward". In a feedforward neural network all values flow in one direction from input towards output. Another approach is "backpropagation". During backpropagation the flow of information is reversed flowing from output to input. This allows us and by extension the neural network itself to locate which neurons had the most effect on a wrong prediction and subsequently alter its weights and threshold. An its among other factors that this loop of feedforward followed by backpropagation that enables the neural networks to learn from its mistakes. This is also where dataset bias can play a huge factor and why a diverse dataset is necessary to develop a capable neural network[17].

3.3 Convolutional Neural Networks

CNN's use a similar approach to ANN's where the feedforward approach is at the core. CNN's are however usually used for advanced image recognition, pattern recognition and computer vision. Therefor CNN's are highly optimized and perform inherently superior with image, speech and audio inputs compared to other neural networks.
The reason for CNN's superiority is the layer composition:

- Convolutional Layer
- Pooling Layer
- Fully-Connected (FC) Layer

While Convolutional layers and pooling layers can alter as needed the fully-connected layer is always the last one and therefore it is also known as the output layer[18].

[19]

3.3.1 Layers

Convolutional layer The convolutional layer works by having a feature detector commonly also referred to as a kernel or filter move across the field of the input checking if specific features are present in a process called convolution. During the runtime the weights set in the feature detector. Each weight is fixed while the filter is moving across the dataframe (input data) and may only be changed in between analyzing the dataframe. In CNN's backpropagation is typically utilized to adjust weights along with other methods like gradient descent.

To introduce nonlinearity to the model *Rectified Linear Unit (ReLU)* transformation is applied to the feature map after each convolution[18].

Pooling layer The pooling layer much like the convolution layer works by having a filter move across the dataframe. However also known as down-sampling the pooling layer's primary objective is to reduce the amount of parameters. The filter differs from the convolutional layer by not having weights assigned, rather applying an aggregation function. Pooling have two main types:

- **Max Pooling**
- **Average Pooling**

Max Pooling works by selecting the maximum value within a field and send it to the output array. Similarly average pooling works by taking the average of each field and send that value to the output array.

While the main objective of the pooling layer is to remove information from the dataframe and therefore valuable information can be lost, the CNN benefits from the reduced complexity, improved efficiency and the reduced risk associated with overfitting[18].

Fully-connected layer In partially-connected layers as the convolutional layer and the pooling layer each node does not necessarily connect to a node in the previous layer, however in the fully-connected layer each output node connects directly to a node in the previous layer.
The fully-connected layer takes the output of the previous layers and leverage a softmax activation function in order to appropriately classify inputs. The probability produced by the fully-connected layer usually range from 0 to 1[18].

3.3.2 Hyperparameters and filter

While parameters like weights and thresholds may be adjusted during the training process, 3 parameters called "**Hyperparameter**" have to be set before running the training scenario, these hyperparameters affect all layers in the convolutional network.

1. **Number of filters**
2. **Stride**
3. **Zero-padding**

Where **Number of filters** influences the depth of the output, **Stride** defines the distance that the filter moves for each increment, and **Zero-padding** which alters the filter to fit the input image and is usually only used when the two don't align.

Figure3.4 displays how a 2x2 filter (B) would move over a 3x3 input matrix (A). The filter works by moving across the input matrix and evaluating the dot products of each subsection, which produces in this case a smaller 2x2 matrix referred to as a "feature_map"[19, p. 959].

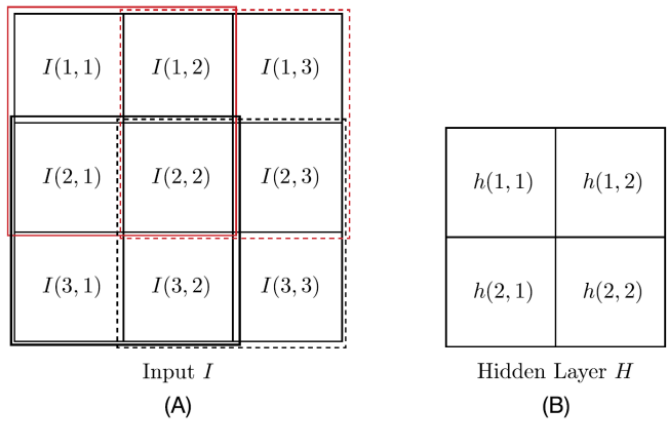


Figure 3.4: Visualization of a convolution on a 3x3 input matrix (A) with a 2x2 filter matrix(B)[19, p. 957]

During the convolution process border pixels contribute less then pixels that are more centralized. To counter this effect we can use Zero-padding to effectively add 0 values around the matrix producing a buffer zone and allowing all pixels to contribute an equal amount. In the figure3.5 [19, p. 962]

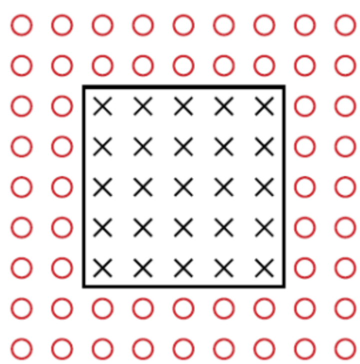


Figure 3.5: Zero-padding visualized on a 5x5 matrix[19, p. 962]

Pooling works by either selecting the max value inside the filter or by taking an average of the filtered area. Figure 3.6 shows how a filter of size 2x2 would move across an input of size 6x6 (A) producing the output of 3x3 (B)[19, p. 964].

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 3.6: Pooling example showing a filter of size 2x2 moving across a matrix of 6x6 (A) producing the output matrix 3x3 (B)[19, p. 964]

3.4 Deep Neural Network

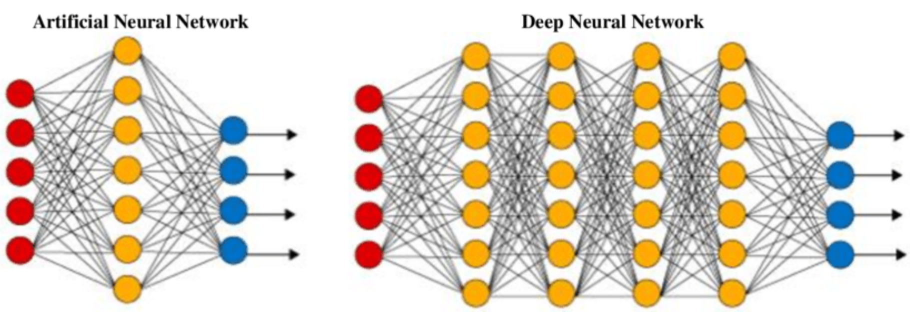


Figure 3.7: Typical size difference between ANN and DNN [20]

Deep Neural Networks or DNN's works by combining factors from both ANN's and CNN's where it usually have more hidden layers than an ANN as shown in figure 3.7 and also having the possibility of using convolutional and pooling layers in its hidden layer section. It also differs from CNN's by being able to utilize dense and other specialized layers that a conventional CNN does not have access to. Among other large models the powerful YOLO object detection algorithm is based on the usage of deep neural networks[21].

3.5 Object Detection

3.6 YOLO

3.6.1 YOLOv5

YOLOv5 is one of the most well known vision-based object detection models, and is known for its speed and accuracy. The "YOLO" in YOLOv5 stands for "You only look once", and is indicative of its real-time detection ability. YOLO is a single stage deep learning algorithm that uses a convolutional neural network for object detection. Because of this YOLO only needs a single forward propagation in its neural network to do the object detection. This gives it its very good real-time performance. The biggest change from the earlier versions of YOLO and YOLOv5 is the addition of the focus layer. Figure 3.8 shows the architecture of YOLOv5.[22]

3.6.2 YOLOv7

YOLOv7 is the successor of the YOLOv6 algorithm and improves upon its predecessor in both detection accuracy and detection speed. The model underwent training exclusively on the MS COCO dataset, starting from scratch without using any other datasets or pre-trained weights.
It is said to be better than every other object detectors before it, with an accuracy of 56.8% average precision (AP) within 30 FPS or higher on Graphics Proccessing Unit (GPU) V100. [24, p. 4-5] This Figure 3.9 shows the architecture of YOLOv7. [23, p. 21]

3.6.3 YOLOv8

YOLOv8 is the newest installment in the YOLO family. This newer version of YOLO supports multiple vision tasks such ass object detection, segmentation, pose estimation, tracking and classification. Because YOLOv8 uses a anchor-free model with decoupled head, it can independently process ob-

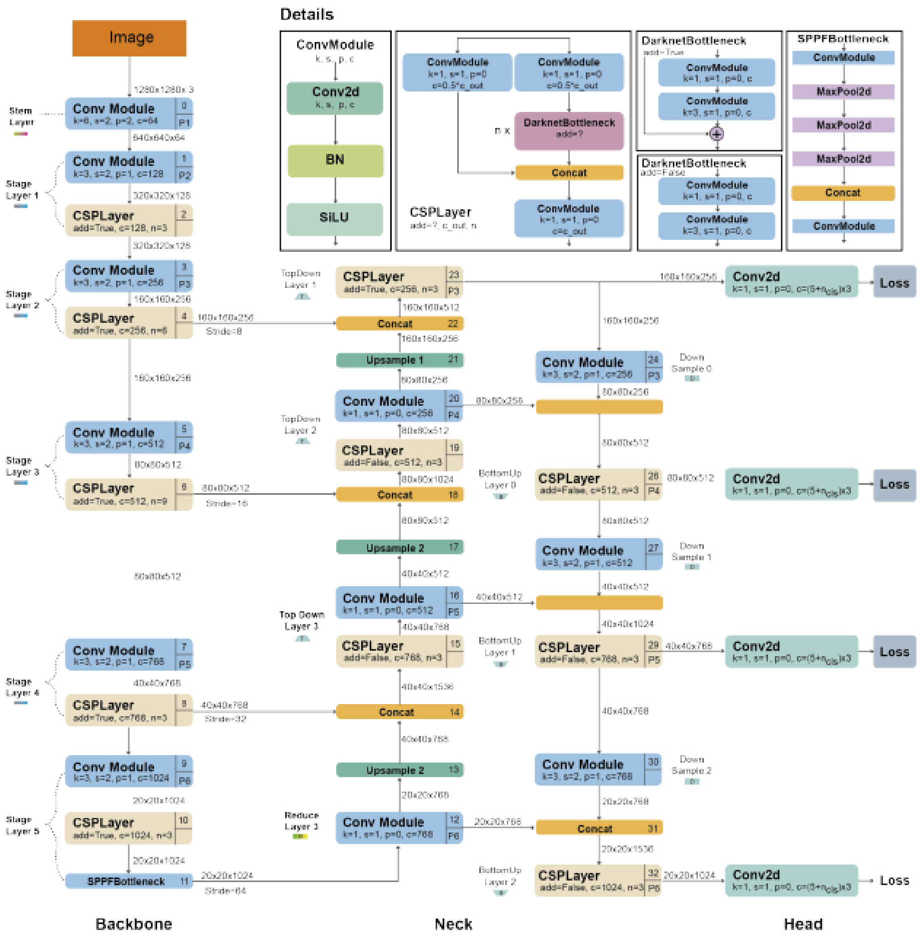


Figure 3.8: YOLOv5 Architecture. [23, p. 17]

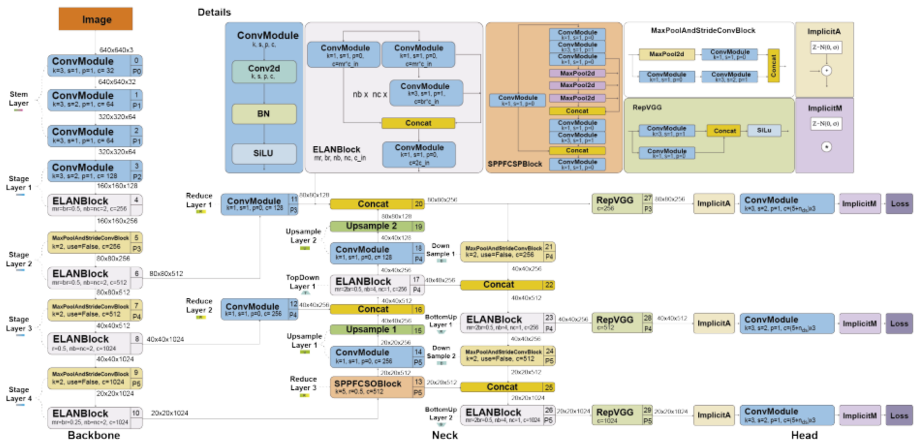


Figure 3.9: YOLOv7 Architecture

jectness, classification, and regression tasks. This allows each branch to focus on each designated task which thereby improves its overall accuracy. [23, p. 21-22] This Figure 3.10 shows the architecture of YOLOv8.

3.7 Object Tracking

[25]

3.8 Deep SORT

SORT stands for Simple Online Realtime Tracker, and Deep SORT is a extension of SORT. SORT is an algorithm for object tracking in videos, and uses 3 methods for tracking. Firstly it uses spatial data association for tracking, the baseline for the tracking-by-detection approach, where it takes the output of detector and uses it as input for the tracker. The IOU tracker, or intersection-over-union tracker, associates detection results from consecutive frames as a track using a greedy algorithm when their intersection-over-union surpasses a specific threshold. The Kalman filter is

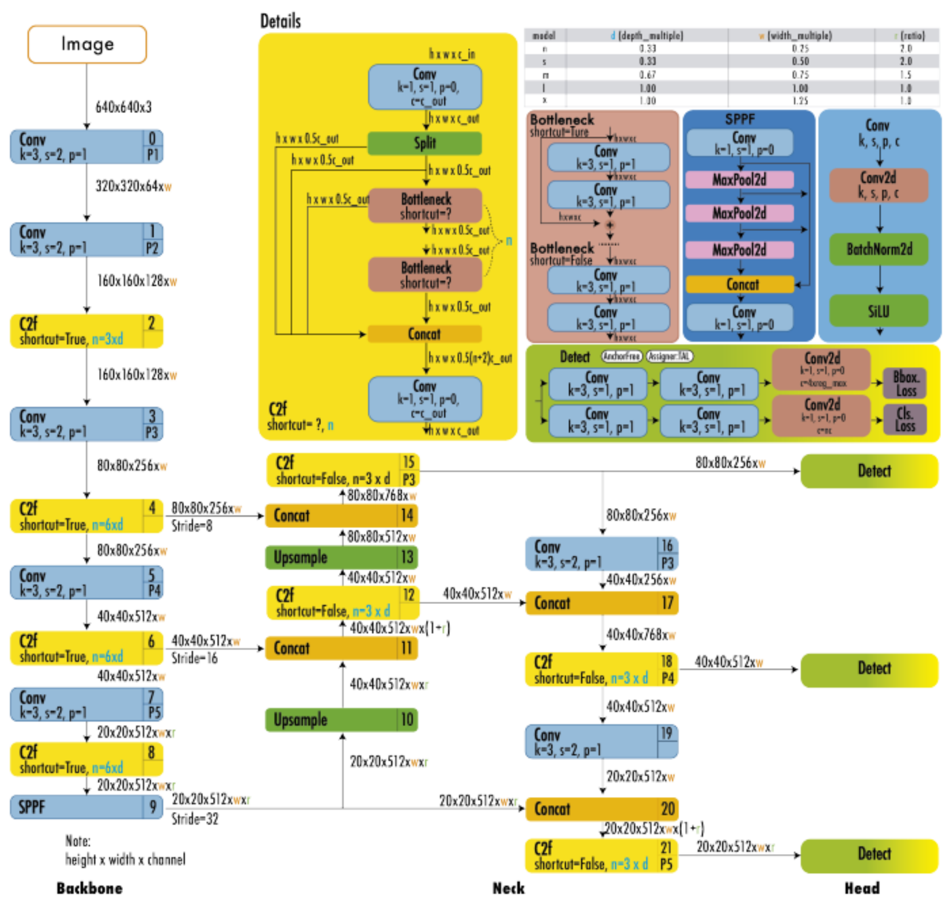


Figure 3.10: Architecture of YOLOv8

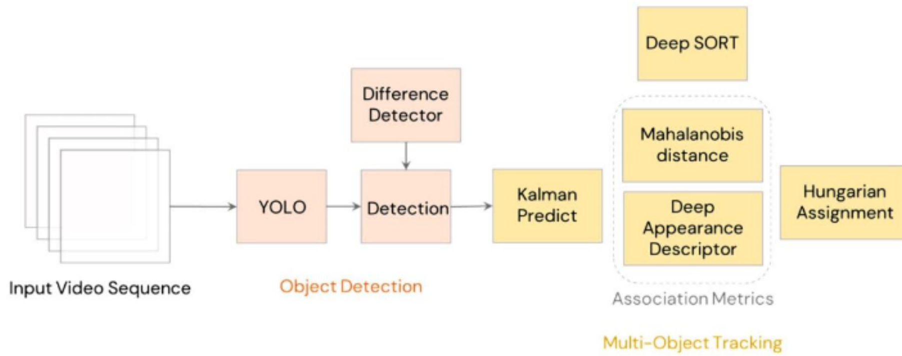


Figure 3.11: Architecture of Deep SORT[26]

used to estimate the location of the tracked object from last frame. The algorithm takes the measurements from detections and previous track states with uncertainty to determine the current states. New detection results are then assigned to the determined tracks using the Hungarian algorithm.[26] Deep SORT also uses the Kalman filter for object tracking, but it additionally integrates a deep association metric derived from appearance features learned by a deep convolutional neural network. To be able to track individual objects over multiple frames it also incorporates ID assignment. It adopts a two-stage approach, initially generating object detections, and subsequently linking these detections to existing tracks. [26] In Figure 3.11 we see the architecture of DeepSORT.

3.9 DBScan

DBScan (distance between nearest points) is a clustering algorithm used to cluster datapoints together in neighborhoods. The key to DBScan is that for any given point its neighborhood with a given radius has to contain a minimum number of points[27]. The figure3.12 shows datapoints of database 1 and 2 plotted forming clusters and noise.

DBScan provide an excellent performance and accuracy advantage when it

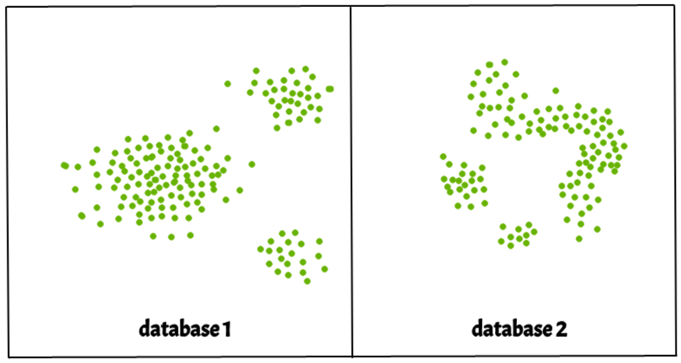


Figure 3.12: This figure displays datapoint clusters and noise for a given database 1 and 2 [27]

comes to clustering for arbitrary shapes and potential noisy data. DBScan requires two provided hyperparameters:

1. **eps**: The eps value defines the neighborhood of any given datapoint, meaning that the distance between datapoint 1 and datapoint 2 has to be less then the value of eps
2. **MinPts**: Minimal amount of points in a neighborhood to be considered a cluster

The figure3.13 shows how a DBScan would analyze a small dataset

geeksforgeeks [27] Steps used in DBScan Algorithm:

1. Find all the neighbor points within eps and identify the core points or visited with more than MinPts neighbors.
2. For each core point if it is not already assigned to a cluster, create a new cluster.
3. Find recursively all its density-connected points and assign them to the same cluster as the core point. A point a and b are said to be density connected if there exists a point c which has a sufficient number of points in its neighbors and both points a and b are within

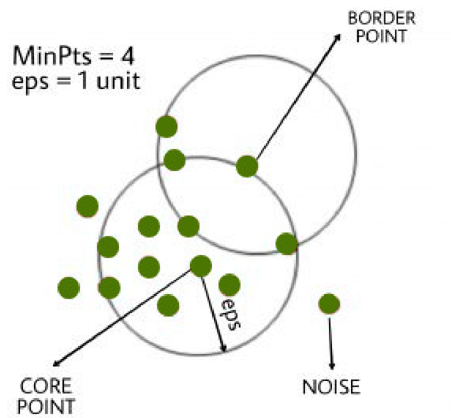


Figure 3.13: Visualizing the DBScan algorithm on a small dataset with hyper-parameters eps = 1 unit and MinPts = 4[27]

the eps distance. This is a chaining process. So, if b is a neighbor of c, c is a neighbor of d, and d is a neighbor of e, which in turn is neighbor of a implying that b is a neighbor of a.

4. Iterate through the remaining unvisited points in the dataset. Those points that do not belong to any cluster are noise.

Pseudocode for the DBScan Clustering Algorithm3.1

Kode 3.1: Pseudocode for the DBScan Clustering Algorithm[27]

```
DBSCAN(dataset, eps, MinPts){
    # cluster index
    C = 1
    for each unvisited point p in dataset {
             mark p as visited
             # find neighbors
             Neighbors N = find the neighboring points ...
                 of p

             if |N|≥MinPts:
                 N = N U N'
                 if p' is not a member of any cluster:
                     add p' to cluster C
    }
```

```
        }
```

Chapter 4

Approach

4.1 Tools

Several different tools were used during the development and research conducted during this thesis. Some like Roboflow and StreamCapture were used for the datasets while OpenCV were vital in processing and altering the visual inputs.

4.1.1 Roboflow

Roboflow is a multipupose tool for creating, training and deploying computer vision models. For our usage we only used the data annotation feature of the tool, as well as its simplification of turning videos into frames/images. As for which format was used when exporting the datasets, the Tensorflow CSV format was used.

4.1.2 OpenCV

OpenCV is the largest computer vision library in the world. [28] OpenCV was created to define a common infrastructure for all computer vision application. With over 2500 optimized algorithms OpenCV is a powerful tool in manipulating and analyzing images [29]. In this thesis OpenCV was used to manipulate images with different image enhancements and also display a preview during the runtime of the analysis. OpenCV was also utilized to draw annotation boxes over the image frame and save it to an output video file.

4.1.3 StreamCapture

StreamCapture was developed as a support software to this bachelor thesis [30]. The purpose of StreamCapture was to capture frames from a live feed from traffic cameras.
StreamCapture consists of two modules: LiveCapture and ImageCapture. LiveCapture utilizes OpenCV to access a live feed and extract frames at certain intervals. ImageCapture utilizes http requests to extract a continually updating photo on a website. The software creates a new directory where it stores the video file and every image frame captured. The decision to save every image frame was made to be able to adjust framerate of the finished video at a later point in time. The application only works correctly for macOS, Linux and any Unix based systems because of the flag "-pattern_type glob" inside the ffmpeg command which is only recognized by Unix systems:

```
ffmpeg_command = f"ffmpeg -framerate {args.fps} ...
    -pattern_type glob -i '{img_folder_name}/image_*.jpg' ...
    -c:v libx264 -pix_fmt yuv420p ...
    {video_folder_name}/timelapse.mp4"
```

To run the software use the command:

```
    py run.py -s "source of the livefeed" --spf "Interval ...
        between captured frames" --fps "framerate of the ...
        final video" -r "Runtime for the software in ...
        seconds" --output "root directory for output files"
```

StreamCapture was used to create a video from a live image feed from an Australian tunnel [31]. During the work on this thesis only the module ImageCapture required and there for the only module that was fully developed.

StreamCapture was ran on a raspberry pi over the course of 24 hours to extract video footage from the Australian tunnel4.1.

Technical Specification:	**Raspberry Pi 4 Model B**
CPU Core count	4
CPU core clock speed	1.5GHz
RAM	8GB
Architecture	64-bit

Table 4.1: Technical Specifications for a Raspberry Pi 4B 8GB[32]

4.2 Limitations

4.2.1 Dataset

The datasets being used in this thesis were sourced from the internet with most being found on YouTube. Full comprehensive list of videos sourced can be found at the associated google drive[33]. These videos were used to train and enhance the object detection models.
The dataset annotated and used in the thesis by Aleksander Vedvik[34] containing videos labeled "Video1" through 12 were used for analytic work.

4.2.2 Available Source code

As stated in the bachelor thesis written by Aleksander Vedvik[35] general source code has not been publicly published.

4.2.3 GPU resources

GPU resources used in this thesis was divided between UiS hosted GPU farms and a 2070RTX card from NVIDIA with the Technical specifications shown in table4.2 Having to rely on the 2070 limited some aspects of the

Technical Specification:	**2070 RTX**
NVIDIA CUDA-cores	2304
Boost-clock	1,71
Base-clock	1,41
Memory configuration	8 GB GDDR6
Memory Interface Width	256-bit

Table 4.2: Technical Specifications for a 2070RTX from NVIDIA[36]

training model where resolution of the dataset would be lowered to speed up training process. Reducing the resolution subsequently reduces the accuracy of the trained model. Limited GPU resources also affected the ability to perform retinex_msr and retinex_ssr as the processing time was closer to 10 seconds a frame instead of multiple frames per second. Even though this is a limitation for the thesis this would not have been practical for an in field solution as it would require high-end architecture to be in place.

4.3 Datasets

4.3.1 Dataset distribution

The dataset comprised of videos sourced through the internet and the dataset used by Aleksander Vedvik in his original thesis [35].
A full list of videos sourced through the internet considered for this thesis visit the excel spreadsheet "VideoCrashDataSet.xlsx" in the google drive containing the whole dataset used in this thesis [33].

4.3.2 Preparation of self annotated dataset

The standards set by Vedvik in his initial thesis [35] proved hard to replicate with the use of normal annotation tools without the specific theory as to what standard to follow when annotating.

4.4 Image Enhancements

In this thesis there were 6 image enhancement methods implemented to increase performance of the object detection and consequently the object tracking and queue detection models as well. These are the same methods that can be found in the bachelor thesis written by Aleksander Vedvik[35]. The 6 methods implemented were:

- Linear Gray Scale Transformation
- Non-Linear Gray Scale Transformation
- Histogram equalization (HE)
- Retinex_ssr
- Retinex_msr
- Masking

gray level transformation, both linear and non-linear, Histogram equalization (HE), Retinex_ssr, Retinex_msr and Masking.

4.5 Object Detection

In this thesis we focused on the Yolo family of object detections as their feedforward capabilities is some of the strongest among object detection algorithms. The reason for the exclusion of YOLOv6 is because it is not an official YOLO model[37] as it was independently developed by a Chinese company called meituan[38].

4.5.1 YOLOv5

The YOLOv5 model developed and published by Ultralytics[39] is an object detection model trained on the 2017 dataset COCO[40]. The weight used was YOLOv5x.pt which is the most accurate YOLOv5 model sacrificing a little speed to increase performance and accuracy. The figure 4.1 visualizes the performance of YOLOv5x compared to the other weights and EfficientDet.

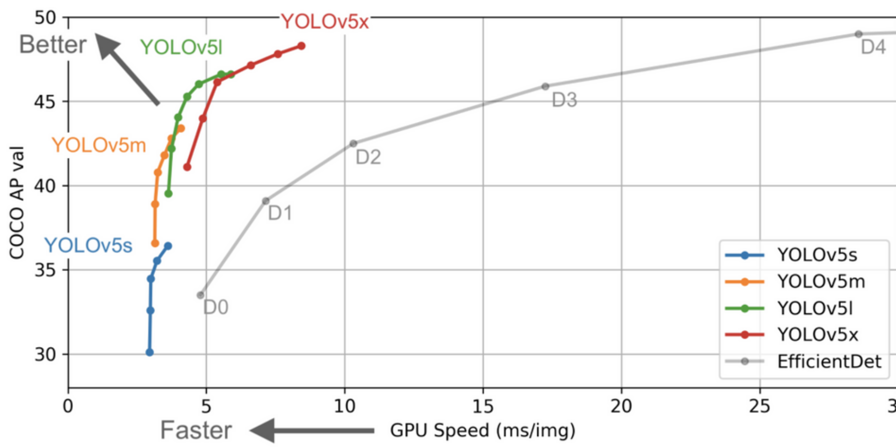


Figure 4.1: Showing the performance of different YOLO weights. With YOLOv5x represented in purple and EfficientDet represented in gray[39]

The YOLOv5 model trained on data sourced for this thesis was trained on this configuration:

- Batch size: [FILL IN THIS]
- Resolution 640x640px

4.5.2 YOLOv7

YOLOv7 Similar to YOLOv5, YOLOv7 also benefits from using the COCO dataset[41]. As visualized in the figure 4.2 YOLOv7 provides significant

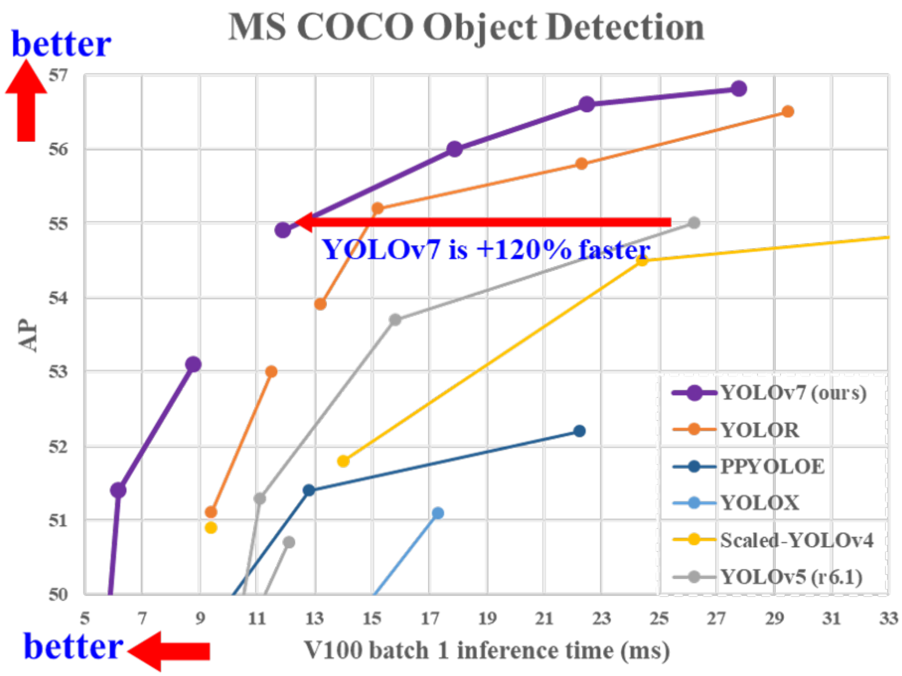


Figure 4.2: YOLOv7 displayed in purple and YOLOv5 displayed in gray[42]

improvement in terms of both speed and performance with YOLOv7 having over a 120% decrease in inference time.

4.5.3 YOLOv8

Continuing the trend YOLOv8 has also been trained on the famous COCO dataset[43]. Looking at the graphs figure 4.3 its a little unclear whether YOLOv8 or YOLOv7 provides the best experience here as v8 sacrifices a little speed for performance compared to v7 and v5. Introduced first with the YOLOv8 distribution is the ability to have built-in tracking[38], however due to time constraint this was not properly implemented.

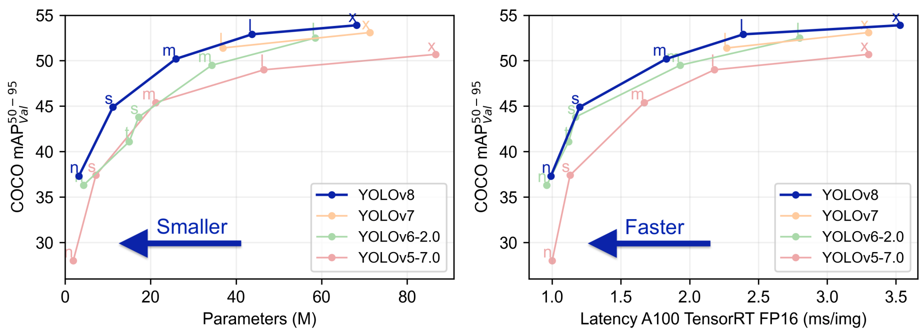


Figure 4.3: YOLOv8 represented in blue, v7 represented in yellow and v5 represented in red[43]

4.6 Tracking with DeepSORT

DeepSORT is a stateoftheart tracking algorithm using the SORT algorithm as a base and applying a Deep Neural Network on top[44]. The specific version used in this thesis is an unofficial version built to fit the YOLOv4 model [45].

The version was altered to also properly handle YOLOv5, v7 and v8.

4.7 Queue tracking

In traffic AID systems a large cause of false positive detections is falsely detecting queued vehicles as an incident. Queues can form at any time during the day and a large percentage of traffic queues are not caused by a traffic incident. Therefore accurately detecting queues can be very helpful in filtering out queued vehicles being classified as "incidents".

4.7.1 Queue Definition

“A vehicle is considered as queued when it approaches within one car length of a stopped vehicle and is itself about to stop.” (DoT [46])

The U.S Department of Transportation defines a queue as a collection of vehicles stopped within one car length of each other. For this thesis we defined a queue as a collection of at least 2 cars standing still or traveling at a slow speed.

Speed calculation Calculating the speed of every vehicle is done by taking its current point and its previous point and measuring the distance between those two points every frame 4.1. This gives us the speed metric: $pixels/frame$.

$$d = \sqrt{((x_2 - x_1)^2 + (y_2 - y_1)^2)} \tag{4.1}$$

Equation used to calculate the distance between two points in 2-Dimensional space [47]. The full implementation can be seen in the code 4.1

Kode 4.1: Full implementation of the simple speed calculation

```
    def simple_speed(self, track_id):
        if track_id not in self.objects:
            print("Track id not in self.objects")
            return -1
        track = self.objects[track_id]
        n = len(track["center_points"])
        if n ≤ self.min_number_of_frames:
            print("min number of frames not met")
            return -1

        current_point = ...
            (int(track["center_points"][-1][0]), ...
            int(track["center_points"][-1][1]))
        previous_point = track["center_points"][-self.PF]

        speed = math.sqrt((current_point[0] - ...
            previous_point[0])**2 + (current_point[1] - ...
            previous_point[1])**2)

        # self.objects[track_id]["speed"] = distance
        return speed
```

Limitations and problem Some limitations apply to this definition of queues:

- **Camera Angle**: The camera angle of the video will affect the algorithms ability to measure the speed of each vehicle. A solution to this would be to define an equation to calculate both normal speed and angular speed of any given vehicle. This could help negate the perceived deceleration of the vehicle as it moves away from the camera, and likewise the perceived acceleration as a vehicle approaches the camera.
- **Video Quality**: The quality of the analyzed video could affect the performance of the queue detection as inconsistent tracking would cause confusion for the analyzed speed.
- **Duplicate ID's**: Duplicate detections where the detection model detects a car twice could interfere with this definition of a queue as it only requires two cars to form a queue. This has been combated by creating a deadzone around the centerpoint of each car see the pseudo-code 4.2

Kode 4.2: Psuedo code for deadzone analysis of centerpoints

```
centerpoint1, centerpoint2 = car1.centerpoint, ...
    car2.centerpoint
if centerpoint2 - centerpoint1 < deadzone:
    continue
else:
    queue.append(car1, car2)
```

4.7.2 Lane Separation

Separating cars driving in different lanes is a crucial part in being able to detect queues accurately. The implementation of lane separation is based on vectors drawn between two cars. The theory is that when a vector is drawn between the centerpoints of two vehicles it will create a vector describing the angle between the cars that can be measured against a universal driving direction vector. If the difference in angle between two centerpoints relative to the horizontal line and the angle of the universal driving direction and the horizontal line differs with more than a certain margin the two represented cars cannot be in the same lane. The mathematical equation can be formulated with the two equations 4.2 and 4.3. Due to lane separation handling the logic by using a static vector it has a hard time handling turns and road irregularities that causes the road to deviate of off a straight line.

$$\theta = \arctan(\frac{x}{y}) \tag{4.2}$$

$$Angle\_difference = |\theta_1 - \theta_2| \tag{4.3}$$

The pseudo code for an implementation can be formulated like 4.3.

Kode 4.3: Psuedo code for same-lane drivng algorithm

```
vehicle_vector = ((car2.x - car1.x), (car2.y - car1.y))
vehicle_angle = angle(vehicle_vector, horizontal_line)

angle_difference = abs(vehicle_angle - ...
    universal_driving_direction)


if angle_difference > margin:
    same_lane_driving = false
else:
    same_lane_driving = true
```

The full implementation of the same-lane driving function can be found in the Code 4.4

Kode 4.4: The full implementation of same-lane driving

```
def same_lane_driving(self, center_point1, center_point2):
    lane_vector = [center_point2[0] - ...
        center_point1[0], center_point2[1] - ...
        center_point1[1]]

    angle_radians = math.atan2(lane_vector[1], ...
        lane_vector[0])
    angle_degrees = math.degrees(angle_radians)

    # print(self.common_driving_direction)
    driving_direction_angle = ...
        math.degrees(math.atan2(self.common_driving_direction[0], ...
        self.common_driving_direction[1]))
```

```
        angle_difference = abs(angle_degrees - ...
            driving_direction_angle)

        angle_difference = min(angle_difference, 360 - ...
            angle_difference)
        print("Angle difference: ", angle_difference)

        return angle_difference ≤ ...
            self.driving_direction_margin or 180 - ...
            angle_difference ≤ self.driving_direction_margin
```

4.7.3 Approaches

Two approaches were developed, one using a simple designed algorithm to sort out clustered objects and the machine learning model known as DBScan discussed in 3.9

Simple Queue Detection Algorithm

The simple queue detection algorithm would loop through every centerpoint for any given frame and look for centerpoints within a predefined radius. For every vehicle that qualifies inside the radius would be ran through the lane separation algorithm 4.4.

Kode 4.5: The code example shows the active deciding part of the queue anlysis code

```
    distance = np.sqrt((x - cx1)**2 + (y - cy1)**2)

    if distance > self.queue_detection_radius or distance ...
        < 10:
        continue
    if not self.common_driving_direction:
        continue

    if self.same_lane_driving((cx1, cy1), (x, y)):
```

DBScan Queue Detection

DBScan works by setting the two hyperparameters radius known as eps or epsilon and a number representing the minimum required nodes. Then it is given a dataset containing the x, y and speed values of all detections present in the given frame. Using these values the algorithm will through the process described in 3.9 return clusters of datapoints. As DBScan have no inherent way to separate between cars of different lanes the output clusters have to be ran through the lane separation algorithm 4.4.

Problems Due to time constraints and over-complication of the DBScan method a full working implementation could not be achieved within the timeframe of this thesis.

4.7.4 Development Challenges

Early problems with the simple queue detection method was that the bounding boxes would presumable expand and grow without any apparent reason as to why. Figure 4.4 shows a failed queue detection.

In the figure 4.4 the queue 1. can be observed stretching outside the boundaries of frame, while queue 2 is stretching high above the car that it is detecting. Throughout the video several of these queues would expand and follow the trend set by queue 1.

Due to the structure of detected objects in the incident_evaluator class inside the incident_evaluator.py there was a conflict in the initial implementation of the queue detection model. Detected objects are stored with a last_frame_number and an array of all detected centerpoints. In the initial implementation the queue detection algorithm would use the last centerpoint of every stored track. This essentially means that any track that has been detected throughout the runtime would be evaluated whether or not it

Figure 4.4: The figure is showing a failed queue detection with the queues labeled 1 and 2 being the main focus.

was being actively detected in the current frame. The simplest solution was to check if last_frame_number is equal to the current frame_number. This would ensure that only currently detected centerpoints are being evaluated. The queues are being evaluated along with the first detection of each frame. This meant that when checking for last_frame_number only the first detection would have an appropriate value while every other value would lay behind. Due to time constraints the fix implemented for this problem was to include centerpoints that were detected last frame as well. Given more time a better solution would be preferred as allowing frames detected during the last frame could potentially include centerpoints that is not detected in the current frame and therefore yield an incorrect value.

4.7.5 Possible improvements

A lot of improvements could be made to increase the performance queue detection model.

- **Fully implemented DBScan**: A fully implemented DBScan algorithm could provide significant performance boost as its model is tailored specifically to analyzing and clustering points with similar values.

- **Further statistical gathering**: Further statistical gathering could provide with a wider specter of analytics to help optimize the overall model of the simple and DBScan queue detection algorithms.

4.8 Improvements

With this thesis building on the work conducted in the bachelor thesis written by Aleksander Vedvik [35], most of the development work with this bachelor thesis was improvements of the model and algorithm already produced and publicly available at the GitHub repository [48]. The following are some of the most impactful improvements made.

- Queues as discussed in section 4.7
- Session Configurations
- JSON formatting

4.8.1 Session Configurations

Session Configurations is an improvement made increase the amount of tests and model configurations that can be tested during one runtime. There are two configuration types that can be configured.

1. **RunConfig**
2. **SessionConfig**

RunConfig A RunConfig contains the same values as those that can be used in the run command. Which means that each RunConfig effectively acts as its own run command. The config file also contains a list of dataset directories to run the configuration on. With the GitHub repository for this bachelor thesis [49] comes a boiler plate for the RunConfig and an example config file that would running a gray scale configuration can be seen in the code example 4.6.

Kode 4.6: Showing a RunConfig.json file initializing a run a gray scale configuration

```
{
    "type": "RunConfig",
    "name": "GrayLinearAnalytics",
    "dir": "GrayLinearAnalytics",
    "data": [
        "Video1",
        "Video2"
    ],
    "configurations": {
        "argOverride": true,
        "args": {
            "CommentValue": "This is a comment value ...
                and should be removed. Args are used ...
                in the case that argOverride is true ...
                for default values leave these unchanged",
            "model": "",
            "checkpoint": "",
            "pretrained": "",
            "skip_frames": 0,
            "resize": 1.0,
            "noise": "",
            "tracking": "",
            "file": "GrayLinearAnalytics",
            "img_enh": "gray_linear",
            "mode": "",
            "show": 1,
            "statistics": "statistics",
            "queue": "",
            "datamode": "json",
            "filetype": "jpg"
        }
    }
}
```

SessionConfig The SessionConfig contains an output directory to export all gathered data and a list of RunConfig.json files to run. A boiler plate for the SessionConfig is also provided in the GitHub repository [49]. A SessionConfig.json example can be seen in the code example 4.7.

Kode 4.7: Showing a Session.json file initializing 3 RunConfig.json files

```
{
    "type": "SessionConfig",
    "dir": "StandardAnalysis",
    "runConfig": [
        "GrayLinearAnalytics.json",
        "GrayNonLinearAnalytics.json",
        "NoAlterations.json"
    ]
}
```

4.8.2 JSON formatting

In the original thesis output data were restricted to a .txt file with minimal standardized layout. The output was very user friendly providing full descriptions for each value. This provides a great user friendly experience, but will be hard to utilize by an automated algorithm. Therefore a second output file was developed with a more standardized layout ordering every value in a dictionary with their description as a key. This was vital to be able to conduct a full array of analysis on the output data.

4.9 Performance evaluation

4.9.1 Evaluating the model

As the main thesis is built on top of the earlier thesis of Aleksander Vedvik [35] the theory and practice behind evaluating detections and tracking stays mainly same in this thesis. This is a introduction to every evaluation argument and its associated mathematical concept.

Detection Accuracy Each track were assigned a real_track and id based on an intersect of union IoU4.5 value between the track bounding box and the annotated bounding box in the dataset greater than 0.4 and less than the current highest IoU value.

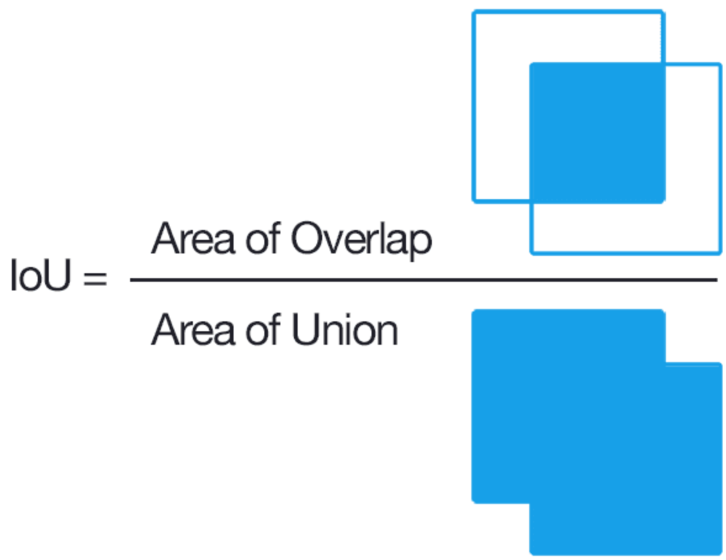


Figure 4.5: IoU formula [50]

Kode 4.8: Pseudo code implementation, found in the Vedvik bachelor [35]

```
for track in tracks:
    for real_object in real_objects:
        IoU = calculate_IoU(track, real_object) if IoU ...
            > 0.4 and IoU > max_IoU:
        max_IoU = IoU
        track['id'] = real_object['id']
```

The detection accuracy denoted as DA is the calculated like shown in the equation4.4 [35]

$$DA = \frac{avg(IoU)}{len(valid\,detections)} \tag{4.4}$$

An adjusted version of DA can be calculated called detection accuracy adjusted where occluded objects are removed excluded from the valid detections resulting in the equation 4.5 [35].

$$DAA = \frac{avg(IoU)}{valid\,detections - occluded\,objects} \tag{4.5}$$

Tracking Accuracy Tracking accuracy is calculated in the same measure as detection accuracy, however a track is considered correct when it corresponds with the id of the last detection of the detected vehicle.

$$TA = \frac{correct\,tracks}{number\,of\,tracks} \tag{4.6}$$

Time analytics Several time analytics were collected to provide a more complete picture of data and its relation to efficiency. The time measurements taken were as follows:

- Mean Tracking Time (MTT)
- Mean Time
- Mean Total Time to Detection (MTTD

Missed detections

False alarm rates were calculated by dividing the false alarms with the number of detections to generate the FAR value shown in the equation 4.7

$$FAR = \frac{number\,of\,false\,alarms}{total\,number\,of\,detections} \tag{4.7}$$

Missed Detections is calculated by dividing the difference between real objects and detected objects by the amount of real objects shown in the equation 4.8

$$MD = \frac{real\,objects - detections}{real\,objects} \tag{4.8}$$

False Positive detections are detected as objects not correlating with a real object and the rate of false positives are shown in the equation 4.9.

$$FP = \frac{false\,positive\,detections}{number\,of\,detections} \tag{4.9}$$

System analysis

System analysis were also conducted to try and better gain an understanding of the performance of model frame by frame4.9.

Kode 4.9: Implementation of system data gathering and perparation for further analysis

```
    current_time = time.time() - timeStart
    gpu = GPUtil.getGPUs()
    gpu = gpu[0]
    cpu_usage = psutil.cpu_percent(interval=None)
    computational_data = {'time': current_time, ...
        'gpu_load_percent': gpu.load*100, ...
        'gpu_memory_used': gpu.memoryUsed, ...
        'gpu_memory_usage': gpu.memoryUtil*100, ...
        'cpu_usage': cpu_usage}
```

4.9.2 Statistics Analysis

StatisticAnalyser.py is the analysis tool built to analyze the output data generated by a SessionConfig.json file. It uses the data provided to generate multiple analytical graphics.

Generated for every RunConfig:

- Confusion Matrix
- Detection Accuracy Bar plot
- Tracking Analysis Bar plot

Generated for every Video in the RunConfig:

- Detection Heatmap
- Incident analysis graph frame by frame
- Over time performance
- System load analysis

4.9.3 Graphing evaluations

As introduced in section 4.9.2 there were several type of graphs and graphical outputs generated to gain an understanding of the data output and how it might correlate with each other.

Confusion Matrix

A confusion matrix is a 2x2 grid showing the relation between the 4 possible detection states.

- **True Positive (TP)**: True positives is known as the detections correlating with a real vehicle
- **True Negative (TN)**: True negatives is the model detecting the negative class
- **False Positive (FP)**: False positives is when the model predicts a positive class incorrectly.
- **False Negative (FN)**: False negatives is when the model does not detect an object

An example of a confusion matrix can be seen in the figure4.6

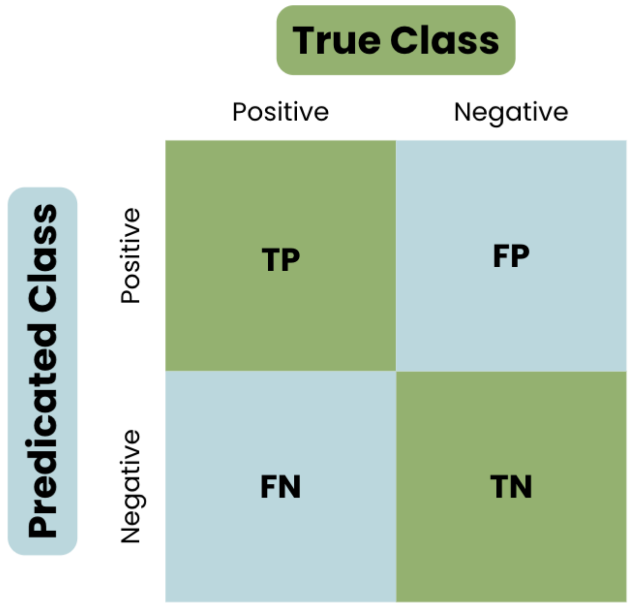


Figure 4.6: Showing the layout of a confusion matrix[51]

Detection Heatmap

Heatmaps visualize the detection densities from the video its calculated on. Through these heatmaps we will be able to gain a better understanding of how the model detects vehicles in specific frames. This could also be beneficial for a possible deployment analysis as it would highlight areas with an abnormally large amount of detections happening with a timeframe. An example of a heatmap can be seen in the figure 4.7 which has been generated from the StatisticAnalyzer.py discussed in the section 4.9.2.

Incident analysis graph frame by frame

Incident analysis graphs were generated to tell us something about the correlation between false positive detections and wrong class detections.

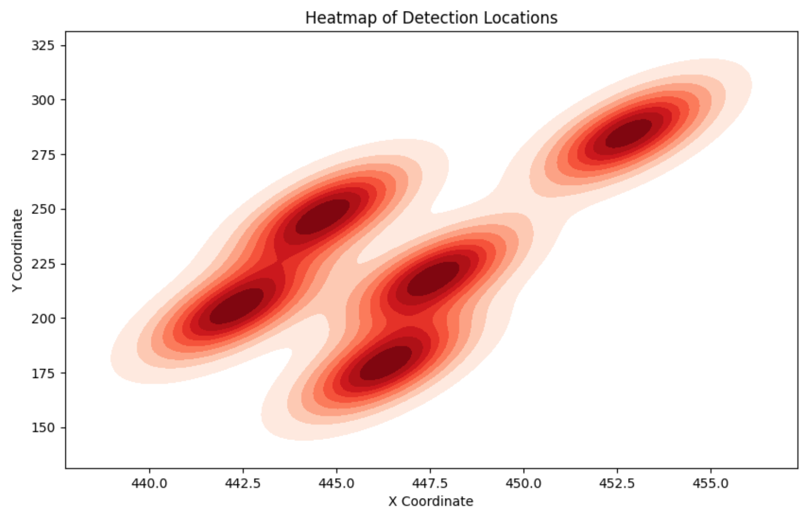


Figure 4.7: Example of a heatmap generated from the data of one of the initial test runs, with this specific one representing the heatmap of video 10

Over time performance

The time based graph is being analyzed to gain an understanding of how the detection times vary throughout the video it was recorded for.

System load analysis

The system load was considered an important part of the analysis as it could tell us something about the required performance for each yolo model and image enhancement. This could help draw a conclusion as to which model would be most suited for in-field implementation.

Chapter 5

Discussions and Results

5.1 Image Enhancements methods

Image enhancing is an important tool that could lead to improved accuracy and efficiency in the object detection phase. With infrastructure related to road tunnels being of varying quality one cannot assume high-end solutions like state of the art cameras or radar technology to be installed. By applying image enhancements different aspects that could prove confusing for the object detections can be removed. Different aspects inside a tunnel that could be removed with the help of image enhancements are:

- Glare, Reflections, and wet surfaces. These are problems that could be inside a tunnel where possibly the use of gray scale enhancement could remove these as a distraction for the model.
- The efficiency of the model could be improved by limiting its view to only the road cutting out information about its surroundings using a mask. This could lead to improved detection time and therefore tracking time. Some problems that could arise with the use of masks is that vehicles especially larger vehicles could end up having its top cut of as it would enter into the mask if the mask is drawn poorly.

5.2 Object Detection

How good an AID system performs often depends on the object detection being used. Object detection are very dependent on quality of input frames, which means that videos with bad quality needs image enhancements before moving in to the object detection model. If the image quality is good, then image enhancement are not necessarily needed, but can still help. We tested the object detection models YOLOv5, YOLOv7 and YOLOv8 because we wanted to see the performance in the YOLO family. Because of time limitations these were the only ones we managed to include in this thesis. We initially wanted to include other models such as TensorFlow, SSD MobileNet, EfficientDet and Faster-RCNN, but could not due to the time limitations.

5.3 Object Tracking

Object tracking is a essential part of detecting wrong way drivers. We decided to try and test two trackers, DeepSORT and DBScan. We did not have enough time to implement and test both, so we chose to focus on DeepSORT. Because of limited time we could not go a step further to gather statistics for performance of DeepSORT, but it is being used in all other statistics we have gathered.

5.4 Queues

A large reason for false positive rates in the current AID systems implemented in tunnels is caused by queues forming inside the tunnels. Although some queues can form as the result of an incident throughout the course of one day there will be many queues formed and dissolved that was not the result of an incident but rather inefficient driving. Implementing a system for automatic queue detection could help alleviate some of the manual labor required to evaluate an incident of stopped vehicle as a queue by possibly applying a tag of queued to every vehicle currently detected in a queue.

5.4.1 Camera Placement

Where cameras are placed in tunnels is essential for a multiple of factors. For cameras placed closer to the entrance/exit of the tunnel the environment is heavily effected by weather. The camera will have more exposure to outside elements, and will need to be able to adjust to these factors.
If a camera is further into the tunnel the lighting will usually be weaker than closer to the outside.
Camera placement is also crucial for the detection model as camera placed on an angle close to 0^o would mean that the further a vehicle gets from the camera the smaller the box becomes, while a camera placed on a 90^o angle facing downwards toward the road bounding boxes would most likely remain the same size throughout its appearance in the dataset.

5.5 Graph Analysis

While confusion matrices are based on data collected from all 12 video the rest of the graphs and heatmaps are based on data collected from video 11. The reason video 11 was chosen because it is the longest video while also showing a tunnel gradually getting filled up with smoke as can be seen in a frame in the figures analyzing the heatmaps 5.15. While only one video was chosen to analyze the meaning of, the rest of the graphs can be found in the google drive [33]. All graphs are based on video 11.

5.5.1 Confusion matrix

When analysing the confusion matrices we look at the the different squares to compare them. We want the top left (True positive) and bottom right (True negative) to be high compared to top right (false negative) and bottom left (false positive). The bigger the difference in ratio the better it performs. In each figure the matrices are from left to right, YOLOv5, YOLOv7 and YOLOv8.

Looking at the confusion matrices in figure 5.5, 5.1, 5.2, and 5.3 we can

see that yolov7 compared to yolov5 and yolov8 is able to maintain a higher amount of true positives and true negatives while also maintaining a low false positive and false negative rate. Analyzing the figures actually reveals that when it comes to pure true detections versus false detections yolov8 has the lowest ratio while in theory being the potential strongest model.
Interestingly masks as shown in 5.4 when compared to the other figures 5.5, 5.1, 5.2, and 5.3 have a much lower true positive rate cutting out almost 800 positive detections.
Just looking at the confusion matrices it becomes apparent that yolov7 outperforms yolov5 and yolov8.

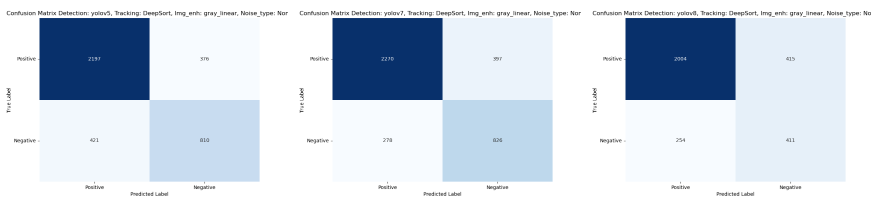


Figure 5.1: Linear gray

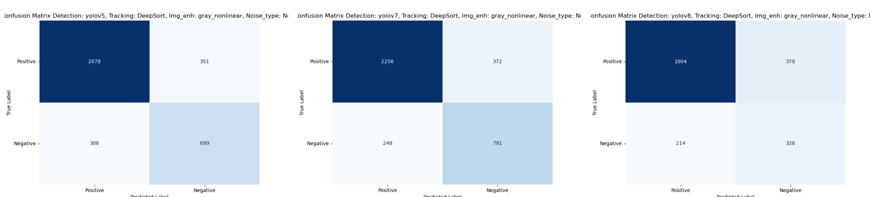


Figure 5.2: Non linear

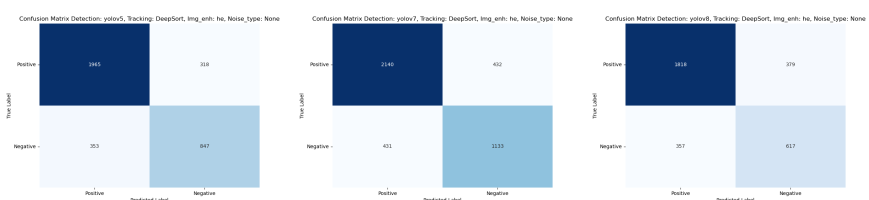


Figure 5.3: Histogram Equalization

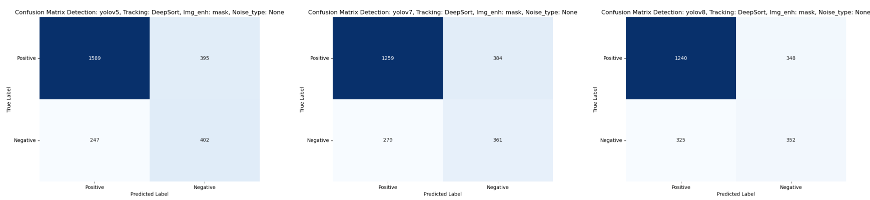


Figure 5.4: Mask

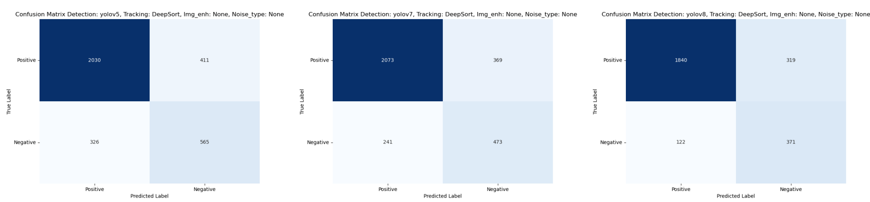


Figure 5.5: None

5.5.2 Incident analysis graph

The incident analysis graph is plotting the **Number of wrong classes** and **False positive detections** on the y-axis over **frame numbers** on the x-axis. The decision was made to use **frame numbers** over the x-axis as opposed to **seconds** as both would essentially show the same graph but a time based graph would have more decimals and could therefore be harder to read.

When looking at the graphs depicting different image enhancement methods in the different yolo models v5 5.6, v7 5.7 and v8 5.8 it starts providing a potential explanation to the interesting discoveries made with yolov8 in the subsection 5.5.1. Yolov5 5.6 and v7 5.7 show a somewhat consistent performance between the two. Featuring spikes in false positives with wrong class detections roughly following the same trend, these two also mostly follow the same distribution in time periods where v7 and v5 will react roughly the same with v7 being a little more aggressive generally giving a higher value of false positives and wrong class detections compared to v5, which holds true to the confidence matrices5.5. Looking at the same set of graphs for yolov8 5.8 you can immediately spot the increase of wrong class detections. These detections also does not follow the same trend as seen in both yolov5 5.6 and v7 5.7. With yolov8 being the newer model of the three this could point to the fact that v8 is classifying vehicles into more sub classes than what is annotated for. To confirm this it would be interesting to log the total detections over time in the same graph to see if the wrong class detections follow the same trend as the total detections.

5.5.3 Time analysis

The time was initially analyzed for the three models yolov5 5.9, v7 5.10 and v8 5.11. These graphs were produced from max time, mean time and min time as they developed over the course of the model runtime. It would have been interesting to analyze these graphs more cleaned up by excluding the initial time value since in almost exclusively the initial time value is the highest. In the graphs for yolov8 5.11 this also damages the perspective as the max value was calculated as upwards of 600 while the v5 5.9 and v7 5.10 usually keeps in the sub 100 region. The cause of this could be as simple as yolov8 having a slower initialization and therefore delaying the first detection. This data could have been analyzed to possibly support yolov8 as it is on paper supposed to be faster than the other two model 4.3.

5.5.4 System load analysis

Looking at the system load data for yolov5 5.12, v7 5.13 and v8 5.14 we actually see that v7 5.13 performs significantly better over all than both v5 5.12 and v8 5.14 having a much lower memory usage and gpu load. With the memory of the v7 5.13 at times actually being close to half of that of v5 5.12 and v8 5.14 and roughly maintaining 10% lower gpu load through out the image enhancements. It is also interesting to see that for the v5 5.12 and v8 5.14 gray linear scaling seems to have a much bigger impact on performance than on v7 5.13.

5.5.5 Heatmap

By looking at the heatmaps we can determine where in the image the most amount of detections are observed. Generally between the three models yolov5 5.15, v7 5.16 and the v8 5.17 and their respective image enhancements not to much can be said. They all follow the same pattern with the most amount of detections located around the stopped vehicle and subsequent people surrounding it. This makes sense as the vehicle and people are perfectly located to be obsereved by the model in most frames throughout the video. A line can also be seen at the left side which would indicate the left driving lane.

5.6 Model Improvements

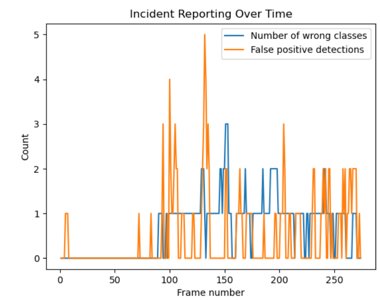
Because of a severe lack of data in datasets of tunnel footage, the model training and testing is limited to a small output for statistics. With more data models will be able to give more statistics and give a better performance overview.

5.7 Further work

More diversity in object detection models should be considered for the future. Improved annotation of sourced dataset so that it can be utilized in statistical evaluations and not only as part of the training method would could yield higher quality data that could help improve the evaluation of the detection model.
Queues have a lot of potential for further work both by finishing the DB-Scan implementation, but also fine tuning the simple model and get a real comparison between the two. Proper annotations supporting queues would enable the verification of any given queue. Expanding on the lane separation theory to index each lane and to be able to recognize any given lane and keep track of its corresponding cars could prove beneficial to the currently utilized method. Enabling queues to handle turns and other irregularities where the roads are not straight.

Yolov8 has also a built in tracker. It could be interesting comparing the differences between the Yolov8 native tracker and the more specialized DeepSORT algorithm. Further systemic analytics possibly with multiple diverse systems could help gaining an understanding of the recommended system requirements, the models efficiency and its deployability. Further research could be conducted towards how this system would be implemented and deployed, and the possible difference between a on-site analysis infrastructure or via a cloud solution.
With the rapid development within technology and road tunnel infrastructure it could be interesting to look into radar technology and look at its possible advantage or disadvantages towards the traditional live feed camera solutions already implemented in a lot of tunnels. Specific problems currently facing a traditional camera solution is the impacts of certain problems such as glare, darkness, snow and heavy rain could be mitigated or maybe even completely terminated. Analyzing the impacts of these problems and the potential sacrifice of using radar instead of camera would be an interesting comparison.
Deeper analysis of graphs and their impact, rooting out outlier values could see some graphs like the once for over time performance 5.11 could result in a better graph explaining the correlations between longer runtimes and detection time.

Gray linear



Gray non-linear

Histogram Equalization

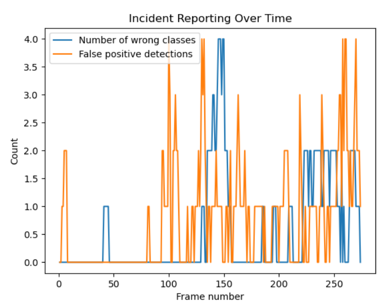
Mask

None

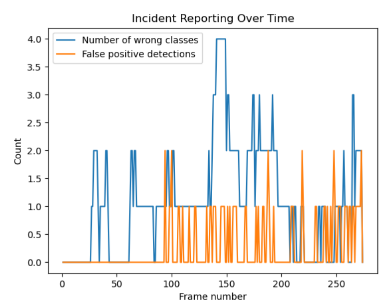
Figure 5.6: YOLOv5 Incident analysis

Gray linear

Gray non-linear



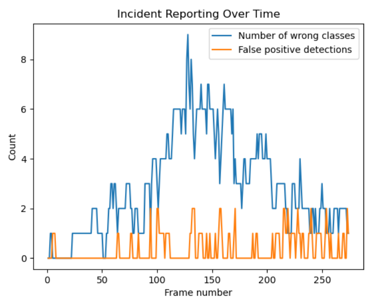
Histogram Equalization



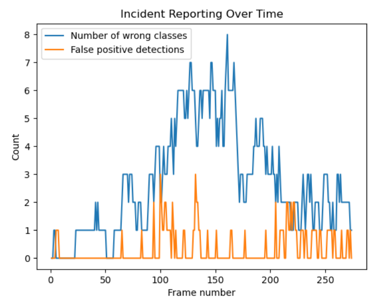
Mask

None

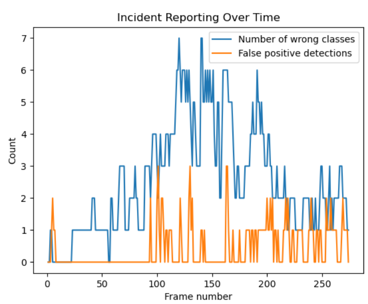
Figure 5.7: YOLOv7 Incident analysis



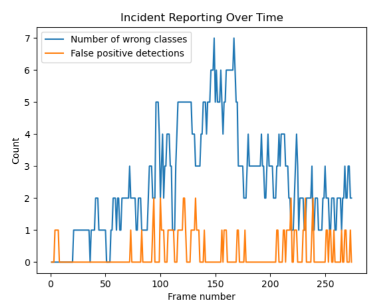
Gray linear



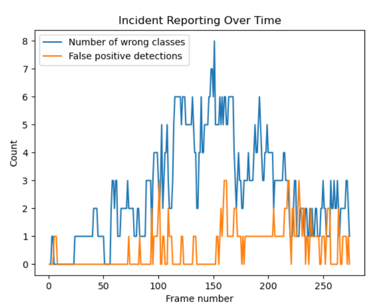
Gray non-linear



Histogram Equalization

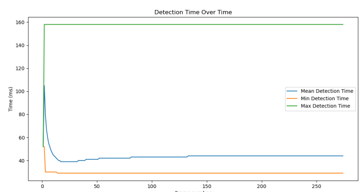


Mask

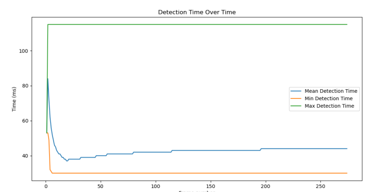


None

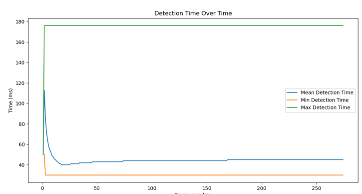
Figure 5.8: YOLOv8 Incident analysis



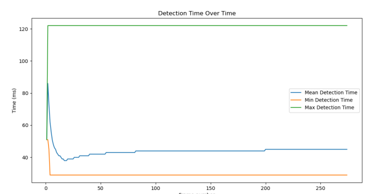
Gray linear



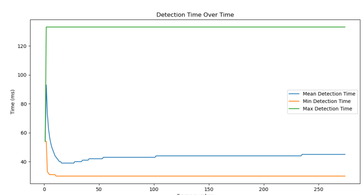
Gray non-linear



Histogram Equalization

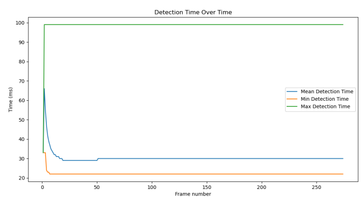


Mask

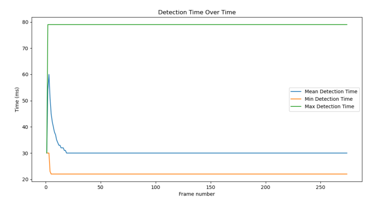


None

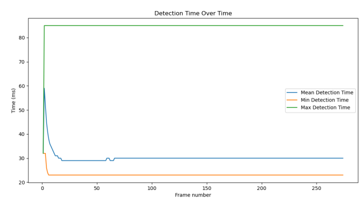
Figure 5.9: YOLOv5 time per fram analysis



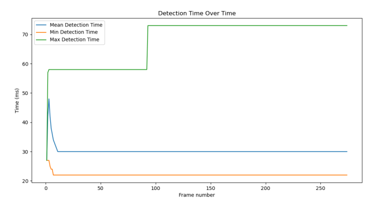
Gray linear



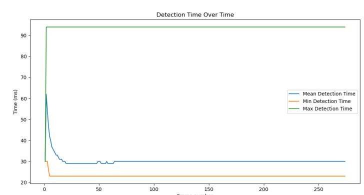
Gray non-linear



Histogram Equalization

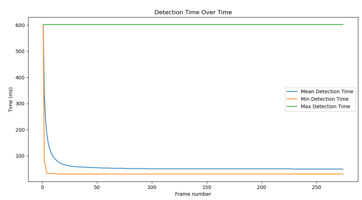


Mask

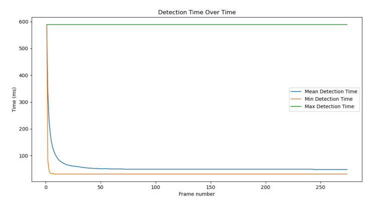


None

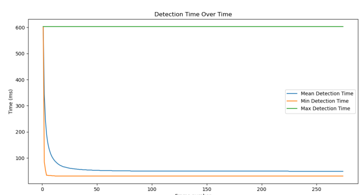
Figure 5.10: YOLOv7 time per fram analysis



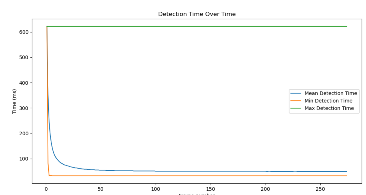
Gray linear



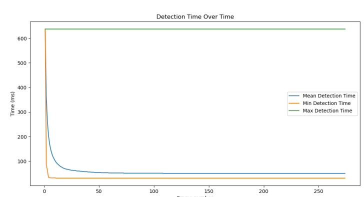
Gray non-linear



Histogram Equalization

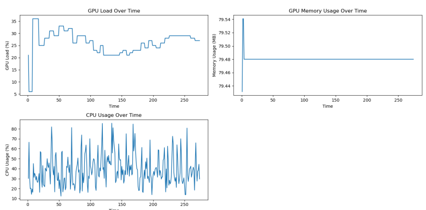


Mask

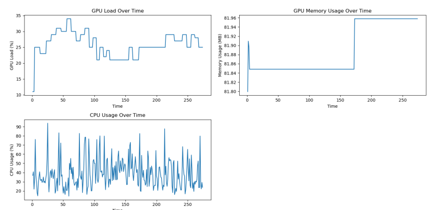


None

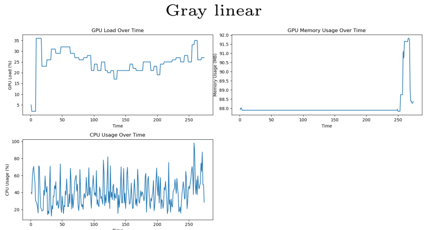
Figure 5.11: YOLOv8 time per frame analysis



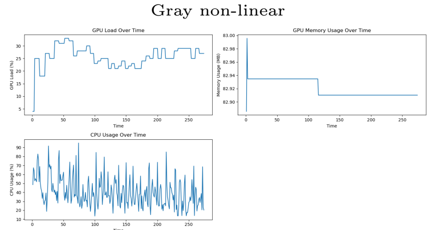
Gray linear



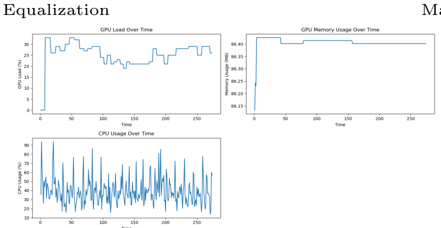
Gray non-linear



Histogram Equalization

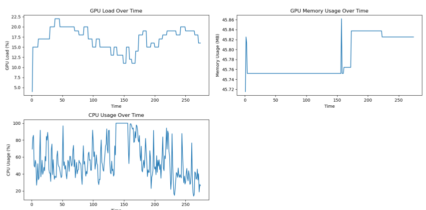


Mask

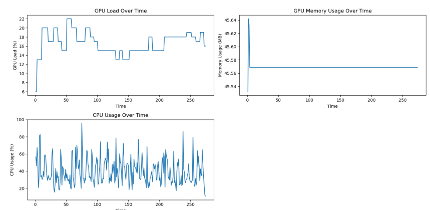


None

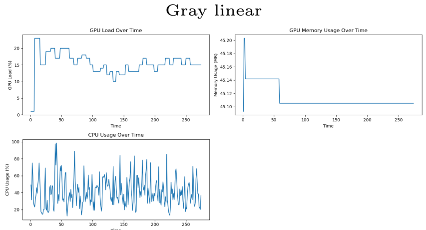
Figure 5.12: YOLOv5 System load



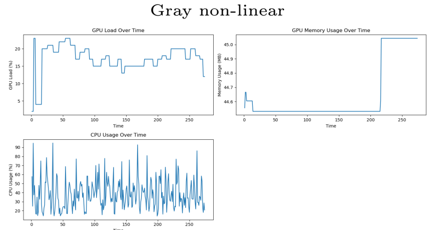
Gray linear



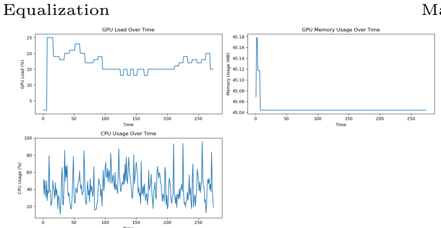
Gray non-linear



Histogram Equalization

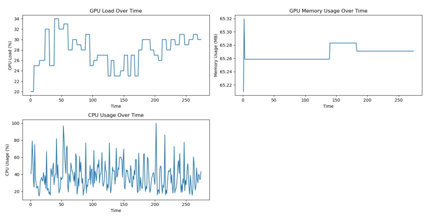


Mask

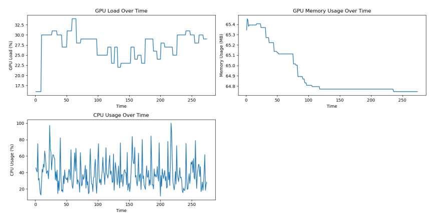


None

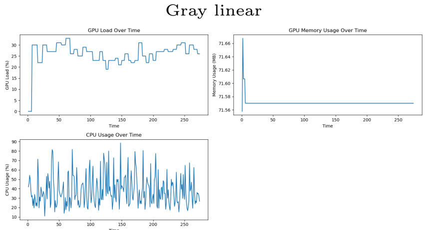
Figure 5.13: YOLOv7 System load



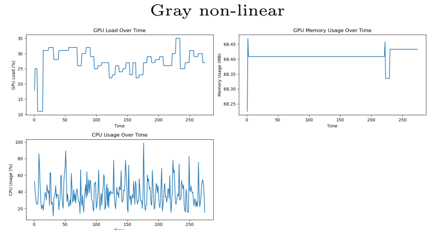
Gray linear



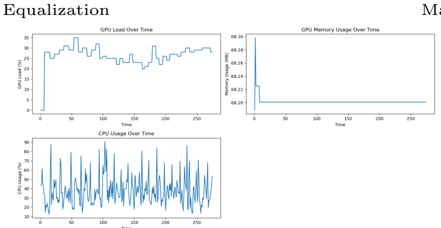
Gray non-linear



Histogram Equalization

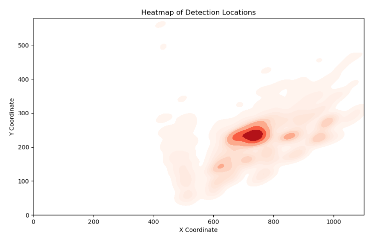


Mask

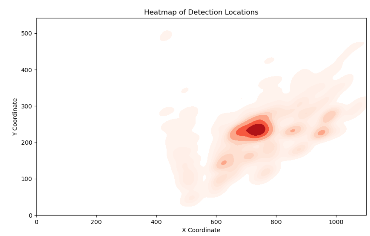


None

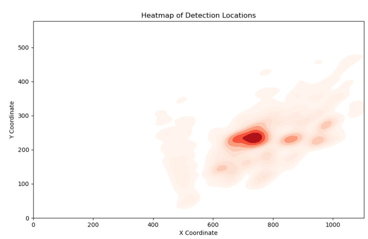
Figure 5.14: YOLOv8 System load



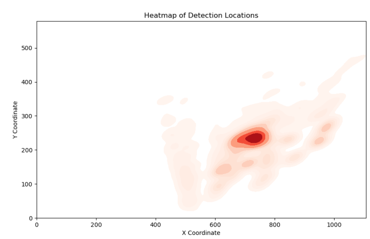
Gray linear



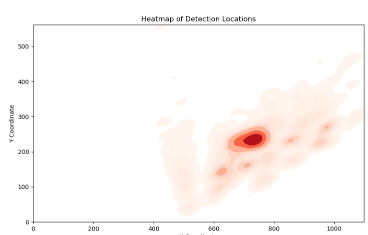
Gray non-linear



Histogram Equalization



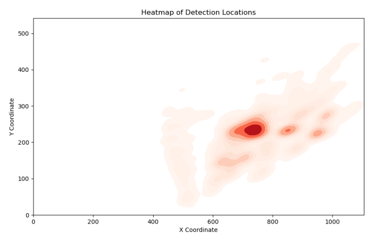
Mask



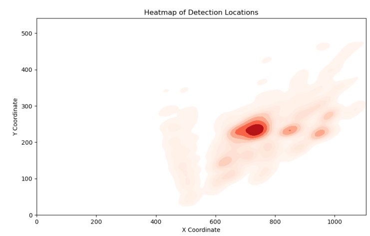
None

Reference image

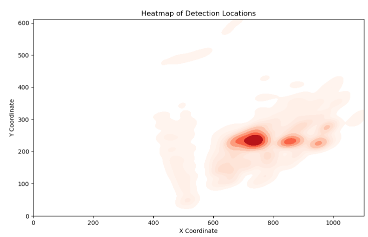
Figure 5.15: YOLOv5 Heatmaps



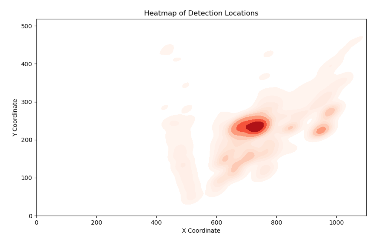
Gray linear



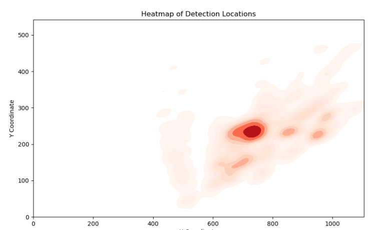
Gray non-linear



Histogram Equalization



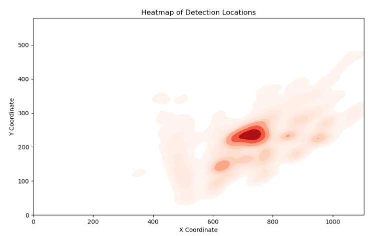
Mask



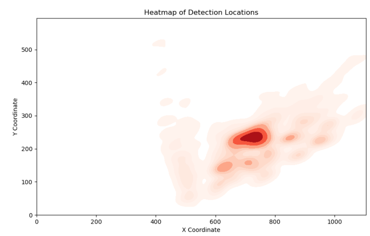
None

Reference image

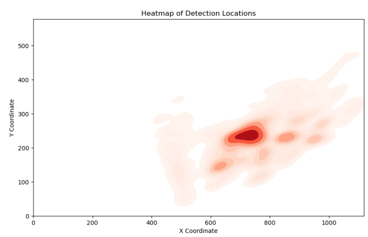
Figure 5.16: YOLOv7 Heatmaps



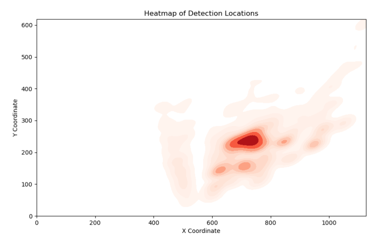
Gray linear



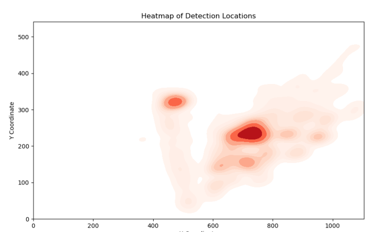
Gray non-linear



Histogram Equalization



Mask



None

Reference image

Figure 5.17: YOLOv8 Heatmaps

Chapter 6

Conclusion

Automatic Incident Detection in road tunnels is going to be a research subject for many more years to come. In our thesis we tried to further explore or test methods for AID systems in road tunnels with the background of the previous bachelor by Aleksander Vedvik. Still to this day most tunnels perform incident detection through loop detectors, which is still dependent on human interaction.
With the help of more advanced and more efficient incident detection methods that use video surveillance footage, the need for human intervention is severely lowered. This moving towards the goal of removing the need for human help. Because of the limited time period of the bachelor only certain methods were considered or used. We took comparisons of confusion matrices between image enhancement methods and object detection methods. For a method to be relatively viable it needs to be efficient and fast, and it needs to have low false positives with high detection rate.
From the confusion matrices in 5.5.1, we can see that YOLOv7 maintains a higher amount of true positives and true negatives while also maintaining a low false positive and false negative rate. It outperforms YOLOv8 even though YOLOv8 has the potential to be the best performing.
From the incident analysis in 5.5.2, we can see that YOLOv8 has a high wrong class detection that does not follow the same trend as both YOLOv5 and YOLOv7. This brings us to a conclusion that YOLOv8 most likely classifies vehicles into more sub classes than what is annotated for.
From the time analysis in 5.5.3, we don't see much because of the initial

time value, that is way higher than all other values. This is probably caused by the initialization time being included in the graph. This is taken to the extreme with YOLOv8 where it shots way ahead of the others. This could probably be cleaned up pretty easily by excluding the initial data.
From the system load analysis in 5.5.4, we can see that v7 outperforms both v5 and v8. This is by using a lower amount of memory and a lower load on the GPU. Specifically in gray linear scaling the performance impact is much larger on v5 and v8 compare to v7.
Heatmaps provided us with little data where any conclusion as to the performance could be drawn 5.5.5. It does however provide some information about where detections most commonly occur. The conclusion was that most detections were made to the right of the frame while a small lane with a smaller density could be seen on the left indicating the left driving lane.

To conclude this thesis we have looked at three of the object detection methods in the YOLO family. We have tested 4 image enhancement techniques, and used the DeepSORT algorithm for tracking. We have created a queue detection algorithm, but did not get time to gather statistics with it. Our testing with the image enhancements methods did not create a big enough impact from one another to conclude with anything for that. We have seen that YOLOv8 has the greatest potential while YOLOv7 still has the best overall performance with gray linear scaling being the most impactful image enhancement. For an in-field application in the near future yolov7 should be considered as a strong contender.

Bibliography

[1] Statens Vegvesen. "Etterlyser nye ideer om tunnel-redning". In: *Statens Vegvesen* 21.02 (2018).

[2] Yong-Kul Ki et al. "An Algorithm for Incident Detection Using Artificial Neural Networks". In: *2019 25th Conference of Open Innovations Association (FRUCT)*. 2019, pp. 162–167. DOI: `10.23919/FRUCT48121.2019.8981509`.

[3] Roger Browne et al. "Comparison and analysis tool for automatic incident detection". In: *Transportation research record* 1925.1 (2005), pp. 58–65.

[4] Peter T Martin et al. *Incident detection algorithm evaluation*. Tech. rep. Upper Great Plains Transportation Institute, 2001.

[5] Shuming Tang and Haijun Gao. "Traffic-incident detection-algorithm based on nonparametric regression". In: *IEEE Transactions on Intelligent Transportation Systems* 6.1 (2005), pp. 38–42. DOI: `10.1109/TITS.2004.843112`.

[6] Mei Lam Tam and William HK Lam. "Validation of instantaneous journey time estimates: a journey time indication system in Hong Kong". In: *Proceedings of the Eastern Asia Society for Transportation Studies Vol. 8 (The 9th International Conference of Eastern Asia Society for Transportation Studies, 2011)*. Eastern Asia Society for Transportation Studies. 2011, pp. 336–336. URL: `https://www.jstage.jst.go.jp/article/eastpro/2011/0/2011_0_336/_pdf`.

[7] Statens veivesen. "AID i tunnel Teknologisammenligning". In: *Statens veivesen* (2013). URL: `https://docplayer.me/17583116-Aid-i-tunnel-teknologisammenligning.html`.

[8] Mohamed S Shehata et al. "Video-based automatic incident detection for smart roads: The outdoor environmental challenges regarding false alarms". In: *IEEE Transactions on Intelligent Transportation Systems* 9.2 (2008), pp. 349–360.

[9] Dinesh Singh and Chalavadi Krishna Mohan. "Deep Spatio-Temporal Representation for Detection of Road Accidents Using Stacked Autoencoder". In: *IEEE Transactions on Intelligent Transportation Systems* 20.3 (2019), pp. 879–887. DOI: 10.1109/TITS.2018.2835308.

[10] Jiawei Wang et al. "A Hybrid Approach for Automatic Incident Detection". In: *IEEE Transactions on Intelligent Transportation Systems* 14.3 (2013), pp. 1176–1185. DOI: 10.1109/TITS.2013.2255594.

[11] Kelathodi Kumaran Santhosh, Debi Prosad Dogra, and Partha Pratim Roy. "Anomaly detection in road traffic using visual surveillance: A survey". In: *ACM Computing Surveys (CSUR)* 53.6 (2020), pp. 1–26.

[12] Chen Wang et al. "A vision-based video crash detection framework for mixed traffic flow environment considering low-visibility condition". In: *Journal of advanced transportation* 2020 (2020). URL: https://www.hindawi.com/journals/jat/2020/9194028/ (visited on 05/11/2024).

[13] Saidasul Usmankhujaev, Shokhrukh Baydadaev, and Kwon Jang Woo. "Real-time, deep learning based wrong direction detection". In: *Applied Sciences* 10.7 (2020), p. 2453.

[14] Wencheng Wang et al. "An experiment-based review of low-light image enhancement methods". In: *Ieee Access* 8 (2020), pp. 87884–87917.

[15] OpenCV. "Histograms - 1 : Find, plot, analyze". In: (2024). URL: https://docs.opencv.org/4.x/d1/db7/tutorial_py_histogram_begins.html (visited on 05/13/2024).

[16] krita. "home". In: (2024). URL: https://krita.org/en/ (visited on 05/12/2024).

[17] IBM. "What is a neural network?" In: (2024). URL: https://www.ibm.com/topics/neural-networks (visited on 05/12/2024).

[18] IBM. "What are convolutional neural networks?" In: (2024). URL: https://www.ibm.com/topics/convolutional-neural-networks (visited on 05/13/2024).

[19] Sergios Theodoridis. "Chapter 18 - Neural Networks and Deep Learning". In: *Machine Learning (Second Edition)*. Ed. by Sergios Theodoridis. Second Edition. Academic Press, 2020, pp. 901–1038. ISBN: 978-0-12-818803-3. DOI: https://doi.org/10.1016/B978-0-12-818803-3.00030-1. URL: https://www.sciencedirect.com/science/article/pii/B9780128188033000301.

[20] Bossy Mostafa et al. "Machine and Deep Learning Approaches in Genome: Review Article". In: *Alfarama Journal of Basic & Applied Sciences* (Aug. 2020). DOI: 10.21608/ajbas.2020.34160.1023.

[21] Bharath K. "Introduction to Deep Neural Networks". In: (July 2023). URL: https://www.datacamp.com/tutorial/introduction-to-deep-neural-networks (visited on 05/13/2024).

[22] Upesh Nepal and Hossein Eslamiat. "Comparing YOLOv3, YOLOv4 and YOLOv5 for autonomous landing spot detection in faulty UAVs". In: *Sensors* 22.2 (2022), p. 464.

[23] Juan Terven and Diana Cordova-Esparza. "A comprehensive review of YOLO: From YOLOv1 to YOLOv8 and beyond". In: *arXiv preprint arXiv:2304.00501* (2023).

[24] Oluwaseyi Ezekiel Olorunshola, Martins Ekata Irhebhude, and Abraham Eseoghene Evwiekpaefe. "A comparative study of YOLOv5 and YOLOv7 object detection algorithms". In: *Journal of Computing and Social Informatics* 2.1 (2023), pp. 1–12. URL: https://publisher.unimas.my/ojs/index.php/jcsi/article/view/5070 (visited on 05/13/2024).

[25] S. R. Maiya. "DeepSORT: Deep Learning to Track Custom Objects in a Video". In: *Nanonets* (2019). URL: https://nanonets.com/blog/object-tracking-deepsort/ (visited on 05/14/2024).

[26] Xinyu Hou, Yi Wang, and Lap-Pui Chau. "Vehicle tracking using deep sort with low confidence track filtering". In: *2019 16th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*. IEEE. 2019, pp. 1–6. URL: https://ieeexplore.ieee.org/document/8909903 (visited on 05/14/2024).

[27] geeksforgeeks. "DBSCAN Clustering in ML | Density based clustering". In: (May 2023). URL: https://www.geeksforgeeks.org/dbscan-clustering-in-ml-density-based-clustering/ (visited on 05/14/2024).

[28] OpenCV. "Home". In: (2024). [Online]. URL: https://opencv.org/ (visited on 05/10/2024).

[29] OpenCV. "About". In: (2024). [Online]. URL: https://opencv.org/about/ (visited on 05/10/2024).

[30] Martin Sævareid Lauritsen. "StreamCapture". In: (2024). URL: https://github.com/MslRobo/StreamCapture.

[31] straya.io. "Burnley Tunnel". In: (2024). URL: https://straya.io/webcams/burnley-tunnel-east/ (visited on 05/10/2024).

[32] Komplett. "Raspberry Pi 4 Model B, 8GB RAM". In: (2024). URL: https://www.komplett.no/product/1160872/datautstyr/pc-komponenter/hovedkort/integrert-cpu/raspberry-pi-4-model-b-8gb-ram# (visited on 05/15/2024).

[33] Martin Sævareid Lauritsen. "Bachelor Dataset". In: (2024). URL: https://drive.google.com/drive/folders/1GZkpp6you43HBV1QzOdCh-wQYwQwcgvV?usp=drive_link.

[34] Aleksander Vedvik. "Datasets". In: (2022). URL: https://drive.google.com/drive/folders/1hNMBL2MNyz5dWZdi1BR1o1X-3yypdCkJ?usp=sharing (visited on 05/14/2024).

[35] Aleksander Vedvik. "For-studie av automatiske hendelsesdeteksjon systemer i veitunneler basert på dype nevrale nettverk og videoovervåknings kameraer". In: (2022). URL: https://hdl.handle.net/11250/3003555.

[36] NVIDIA. "Sammenligning av spesifikasjoner for 20-serien". In: (2022). URL: https://www.nvidia.com/nb-no/geforce/graphics-cards/compare/?section=compare-20.

[37] Revca. "YOLOv6 : Explanation, Features and Implementation". In: (June 2022). URL: https://revca-technologies.medium.com/yolov6-explained-in-simple-terms-c46a0248bddc (visited on 05/14/2024).

[38] meituan. "YOLOv6". In: (Sept. 2023). URL: https://github.com/meituan/YOLOv6 (visited on 05/14/2024).

[39] Glenn Jocher. *YOLOv5 by Ultralytics*. Version 7.0. 2020. DOI: 10.5281/zenodo.3908559. URL: https://github.com/ultralytics/yolov5 (visited on 05/14/2024).

[40] COCO. ""COCO - common objects in context". In: (2022). URL: https://cocodataset.org/#home (visited on 05/14/2024).

[41] Gaudenz Beosch. "YOLOv7: A Powerful Object Detection Algorithm (2024 Guide)". In: (2024). URL: https://viso.ai/deep-learning/yolov7-guide (visited on 05/14/2024).

[42] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. "YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors". In: *arXiv preprint arXiv:2207.02696* (2022).

[43] Ultralytics. "ultralytics". In: (2024). URL: https://github.com/ultralytics/ultralytics (visited on 05/14/2024).

[44] Allan Kouidri. "Mastering Deep Sort: The Future of Object Tracking Explained". In: (Nov. 2024). URL: https://www.ikomia.ai/blog/deep-sort-object-tracking-guide (visited on 05/14/2024).

[45] theAIGuysCode. "yolov4_deepsort". In: (Aug. 2021). URL: https://github.com/theAIGuysCode/yolov4-deepsort (visited on 05/14/2024).

[46] Federal Highway Administration. *Traffic Analysis Toolbox Volume VI: Definition, Interpretation, and Calculation of Traffic Analysis Tools Measures of Effectiveness.* https://ops.fhwa.dot.gov/publications/fhwahop08054/sect3.htm. 2021. (Visited on 05/14/2024).

[47] BYJUS. *Distance Between Two Points Formula*. URL: https://byjus.com/maths/distance-between-two-points-formula (visited on 05/14/2024).

[48] Aleksander Vedvik. "Bachelor". In: (2022). URL: https://github.com/aleksander-vedvik/Bachelor (visited on 05/14/2024).

[49] Alexander Kruke Bjugan Martin Sævareid Lauritsen. "Bachelor Deep Learning For AID". In: (2024). URL: https://github.com/MslRobo/Bachelor_Deep_Learning_For_AID.

[50] Adrian Rosebrock. "Intersection over Union (IoU) for object detection". In: (2022). URL: https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/ (visited on 05/15/2024).

[51] Nisha Arya Ahmed. "What is A Confusion Matrix in Machine Learning? The Model Evaluation Tool Explained". In: (Nov. 2023). URL: https://www.datacamp.com/tutorial/what-is-a-confusion-matrix-in-machine-learning (visited on 05/15/2024).

List of Figures

List of Tables

Appendix A

GitHub repository

All code used is stored on GitHub, and datasets used stored on Google Drive:

- **GitHub**: https://github.com/MslRobo/Bachelor_Deep_Learning_For_AID
- **Datasets**

Appendix B

Code excerpts

As the thesis was built on top of the bachelor thesis of Aleksander Vedvik a lot of the source code present is from his initial thesis and proper separation of code was not possible due to having to alter source code in order to conduct the work for this thesis. His thesis can be found at [35]

```
import os
import sys
import glob
import time
import GPUtil
import psutil

PATH_TO_THIS_FILE = os.path.dirname(os.path.abspath(...
    __file__))
sys.path.insert(0, PATH_TO_THIS_FILE + '\\tools\\')
sys.path.insert(0, PATH_TO_THIS_FILE + '\\tools\\...
    deep_sort')
sys.path.insert(0, PATH_TO_THIS_FILE + '\\')
sys.path.insert(0, PATH_TO_THIS_FILE + '\\training\\')
sys.path.insert(0, PATH_TO_THIS_FILE + '\\training\\...
    tensorflowapi\\')
sys.path.insert(0, PATH_TO_THIS_FILE + '\\training\\...
    tensorflowapi\\research\\')
sys.path.insert(0, PATH_TO_THIS_FILE + '\\training\\...
    tensorflowapi\\research\\object_detection')

import cv2
```

```
import json
import numpy as np
from tools.detection_model import Detection_Model
from tools.tracking_model import Tracking_Model
from tools.incident_evaluator import Evaluate_Incidents
from tools.performance_evaluator import ...
    Evaluate_Performance
import argparse
from tools.visualize_objects import draw_rectangle, ...
    draw_text, draw_line, draw_parallelogram
from tools.tunnel_manager import Tunnel_Manager


"""
Parser setup has been influenced by the implementation ...
    used by Alexander Vedvik in his bachelor thesis that
layed the ground work for this thesis.
"""
parser = argparse.ArgumentParser(
    description="Realtime object detection and tracking"
)
parser.add_argument("-m",
                    "--model",
                    help="Choose perferred detection ...
    model to be utilized",
                    type=str)
parser.add_argument("-c",
                    "--checkpoint",
                    help="Choose checkpoint number to be...
     used, 3 will be used as default",
                    type=str)
parser.add_argument("-p",
                    "--pretrained",
                    help="Choose whether or not to use a...
     pre-trained model or not. 1 = Ture, 0 = False (0 is ...
    defautt)",
                    type=str)
parser.add_argument("--skip_frames",
                    help="Choose number of frames to ...
    skip",
                    type=int)
parser.add_argument("-r",
                    "--resize",
                    help="Define a scale factor to ...
    resize the input video",
                    type=float)
parser.add_argument("--resolution",
                    help="Set the resolution wanted for ...
    the input expecting height, width will be ...
```

```
    automatically calculated",
                       type=int)
parser.add_argument("-b",
                       "--brightness",
                       help="Set the brightness level for ...
    the input in percentage expected integer in range ...
    -100 to 100",
                       type=int)
parser.add_argument("-n",
                       "--noise",
                       help="Set the noise level to ...
    simulate a bad quality feed accepted noise types are:...
     gauss, salt, speckle",
                       type=str)
parser.add_argument("-t",
                       "--tracking",
                       help="Choose tracking model to ...
    utilize, DeepSort will be used as default")
parser.add_argument("-q",
                       "--queue",
                       help="Choose queueing algorithm, ...
    DBScan or Simple")
parser.add_argument("-f",
                       "--file",
                       help="Define the name of the saved ...
    file",
                       type=str)
parser.add_argument("-i",
                       "--img_enh",
                       help="Specify how the image should ...
    be enhanced. By default no enhancements will be ...
    applied",
                       type=str)
parser.add_argument("-s",
                       "--source",
                       help="Specify the source of the ...
    video to be analyzed",
                       type=str)
parser.add_argument("--mode",
                       help="Specify the mode of the ...
    application, by default live tracking will be used, ...
    other modes include analysis and training",
                       type=str)
parser.add_argument("--show",
                       help="Show a live feed of the ...
    tracking process, values are 1 = true, 0 = false, by ...
    default live feed will be disabled.",
                       type=int)
parser.add_argument("--datamode",
```

```
                        help="Specify how the source should ...
    be handled. json expects a json file with session ...
    configurations, and mp4 expects a single mp4 file. ...
    Default is mp4",
                        default="mp4",
                        type=str)
parser.add_argument("--filetype",
                        default="mp4",
                        help="Specify what file type the ...
    feed is should be jpg or mp4")
parser.add_argument("--iterations",
                        help="Specify how many iterations ...
    should be done, each iteration changes the value of ...
    brightness or noise depending on which is active",
                        type=int)
parser.add_argument("-a",
                        "--analysis",
                        help="Should statistics analysis be ...
    ran after completion (0 (default) or 1) (Should not ...
    be used in its current state)",
                        type=int,
                        default=0)
parser.add_argument("--downscale",
                        help="1 if downscaling should be ...
    performed, downscaling from max resolution down to ...
    360p at the lowest iteration",
                        default=0,
                        type=int)

args = parser.parse_args()

"""
Commands used in testing:
python liveTrack.py -s StandardAnalysis.json --file ...
    StatisticsTest00 --datamode json --filetype jpg --...
    show 1
python liveTrack.py -s StandardAnalysis.json --file ...
    StatisticsTest00 --datamode json --filetype jpg --...
    iterations 21 --show 1
"""

def argReplacement(file):
    if not args.model:
        args.model = file['model']
    if not args.checkpoint:
        args.checkpoint = file['checkpoint']
    if not args.pretrained:
        args.pretrained = file['pretrained']
    if not args.tracking:
```

```
        args.tracking = file['tracking']
    args.skip_frames = file['skip_frames']
    args.resize = file['resize']
    args.noise = file['noise']
    args.file = file['file']
    args.img_enh = file['img_enh']
    args.mode = file['mode']
    args.show = file['show']
    args.queue = file['queue']
    args.filetype = file['filetype']
    args.datamode = file['datamode']

# Extracts the json data
def extractJSONFile(jsonFile):
    dataCollectionDir = r'.\\SessionConfigurations'
    dataCollectionFile = os.path.join(dataCollectionDir,...
     jsonFile)

    with open(dataCollectionFile, 'r') as f:
        config = json.load(f)

    return config

def main(video, dirNames, iterationOptions=None, ...
    new_resolution=False):
    datasets = []
    percentDone = 0
    sourceDir = r'.\\data\\rawData'
    sourceFile = os.path.join(sourceDir, video)
    datasetDir = r'.\\data\\incidents'
    baseOutputDir = r'.\\data\\output'
    maskDir = r'.\\data\\tunnel_data\\masks'

    if args.datamode == "json":
        if dirNames['sessionDir']:
            baseOutputDir = os.path.join(baseOutputDir, ...
    dirNames['sessionDir'])
            baseOutputDir = os.path.join(baseOutputDir, ...
    dirNames['runDir'])
        else:
            baseOutputDir = os.path.join(baseOutputDir, ...
    dirNames['runDir'])

        if not os.path.exists(baseOutputDir):
            os.makedirs(baseOutputDir)

    if args.filetype == "mp4" or not args.filetype:
        datasets.append({"dataset": "selectedVideo", "...
    video": sourceFile})
```

```
        maskPath = os.path.join(maskDir, video.split("."...
)[0] + ".png")
        mask = maskPath
    else:
        datasetTunnelDir = os.path.join(datasetDir, ...
video)
        # print("Dataset: ", datasetTunnelDir, " Exists:...
 ", os.path.exists(datasetTunnelDir))
        image_dir = os.path.join(datasetTunnelDir, "...
images")
        anno_dir = os.path.join(datasetTunnelDir, "...
annotations.json")
        datasets.append({"dataset": video + "...
self_annotated", "images": image_dir, "annotations": ...
anno_dir})
        # print("Dataset: ", datasets)
        mask = os.path.join(maskDir, video + ".png")

    # print("Mask: ", mask)

    model_filename = os.path.join(PATH_TO_THIS_FILE, '...
tools/model_data/mars-small128.pb')

    paths = {
        "CHECKPOINT_PATH": "./training/models/ssd_mobnet...
/",
        "PIPELINE_CONFIG": "./training/models/ssd_mobnet...
/pipeline.config",
        "LABELMAP": "./training/annotations/label_map....
pbtxt",
        "DEEPSORT_MODEL": model_filename
    }

    image_enhancement_methods = ["gray_linear", "...
gray_nonlinear", "he", "retinex_ssr", "retinex_msr", ...
"mask"]
    models = ["yolov5", "yolov5_trained",  "yolov8", "...
yolov7"]
    tracking_models = ["DeepSort"]
    noise_types = ["gauss", "salt", "speckle"]
    classes = {"car": "1", "truck": "2", "bus": "3", "...
bike": "4", "person": "5", "motorbike": "6"}

    model_name = "yolov5"
    if args.model in models:
        paths["CHECKPOINT_PATH"] = "./training/models/" ...
+ args.model + "/"
        paths["PIPELINE_CONFIG"] = "./training/models/" ...
+ args.model + "/pipeline.config"
```

```
        model_name = args.model

    tracking_model_name = "DeepSort"
    if args.tracking and args.tracking in ...
   tracking_models:
        tracking_model_name = args.tracking

    ckpt_number = "3"
    if args.checkpoint is not None:
        ckpt_number = args.checkpoint

    filename = ""
    if args.file is not None:
        filename = args.file

    image_enhancement = "None"
    # print("Args Img_enh: ", args.img_enh)
    if args.img_enh is not None and args.img_enh in ...
   image_enhancement_methods:
        image_enhancement = args.img_enh

    brightness = None
    if args.brightness is not None:
        brightness = args.brightness
        brightness -= 10*iterationOptions['...
   current_iteration_index']

    if args.queue == "":
        args.queue = "simple"

    resolution = None
    if args.resolution is not None:
        resolution = args.resolution

    noise_type = None
    if args.noise is not None and args.noise in ...
   noise_types:
        noise_type = args.noise

    if args.pretrained == "1":
        paths["CHECKPOINT_PATH"] = "./training/pre-...
   trained-models/" + args.model + "/checkpoint/"
        paths["PIPELINE_CONFIG"] = "./training/pre-...
   trained-models/" + args.model + "/pipeline.config"
        paths["LABELMAP"] = "./training/annotations/...
   mscoco_label_map.pbtxt"
        model_name = "Pretrained"
        ckpt_number = "0"
        classes = {"car": "3", "truck": "8", "bus": "6",...
```

```
     "bike": "2", "person": "1", "motorbike": "4"}

    skip_frames = 1
    if args.skip_frames:
        skip_frames = int(args.skip_frames)

    resize = 1
    if args.resize:
        resize = float(args.resize)

    tunnel_manager = Tunnel_Manager()
    tunnel_data = tunnel_manager.get_tunnel_data(video....
    split(".")[0])
    driving_direction = None
    if tunnel_data:
        driving_direction = tunnel_data["...
    driving_direction"]

    model = Detection_Model(model_name, classes, paths, ...
    ckpt_number)
    tracker_model = Tracking_Model(paths["DEEPSORT_MODEL...
    "], tracker_type=tracking_model_name)
    evaluater = Evaluate_Incidents(classes, ...
    driving_direction=driving_direction)
    if args.filetype == "mp4":
        pe = Evaluate_Performance("Video", datasets, ...
    classes, model, tracker_model, mask=mask, noise_type=...
    noise_type)
    else:
        pe = Evaluate_Performance("Images", datasets, ...
    classes, model, tracker_model, mask=mask, noise_type=...
    noise_type)

    # if args.file != None or args.mode == "analysis":
    if args.mode == "analysis":
        if args.filetype == "mp4":
            cap = cv2.VideoCapture(sourceFile)
        else:
            ret, frame, new_video, mask = pe.read(resize...
    )
            cap = frame
        length = 0
        if cap and not cap.isOpened() and args.filetype ...
    == "mp4":
            print("Error: Could not open video")
        elif args.filetype == "mp4":
            width = cap.get(cv2.CAP_PROP_FRAME_WIDTH)
            height = cap.get(cv2.CAP_PROP_FRAME_HEIGHT)
            length = int(cap.get(cv2....
```

```
CAP_PROP_FRAME_COUNT))
            # print("Width: %d, Height: %d" % (width, ...
height))
        elif cap is not None:
            height, width = cap.shape[:2]

        fourcc = cv2.VideoWriter_fourcc(*'MP4V')
        # if args.mode == "analysis":
        output_video_path = os.path.join(baseOutputDir, ...
f'{video}_{new_resolution}.mp4')
        # else:
        #     output_video_path = os.path.join(...
baseOutputDir, (args.file + ".mp4"))
        frame_rate = 20
        if width is not None and height is not None:
            frame_size = (int(width), int(height))
        # print(frame_size)

        out = cv2.VideoWriter(output_video_path, fourcc,...
 frame_rate, frame_size)
    else:
        if os.path.exists(datasets[0]["images"]):
            # print("In the right place")
            path = datasets[0]["images"]
            png_files = glob.glob(f"{path}/*.png")
            jpg_files = glob.glob(f"{path}/*.jpg")
            length = len(png_files) + len(jpg_files)
        else:
            print("In the else")
            length = 0

    resolutions = {
        '720': {'width': 1280, 'height': 720},
        '648': {'width': 1152, 'height': 648},
        '576': {'width': 1024, 'height': 576},
        '360': {'width': 640, 'height': 360}
    }
    if not new_resolution:
        res = {'width': 1280, 'height': 720}
    else:
        res = resolutions[new_resolution]
    frame_size = (int(res['width']), int(res['height']))
    output_video_path = f'{baseOutputDir}/{video}_{res["...
height"]}.mp4'
    fourcc = cv2.VideoWriter_fourcc(*'MP4V')
    frame_rate = 20
    print(output_video_path)
    # raise ValueError
    # out = cv2.VideoWriter("loloutput.mp4", fourcc, ...
```

```
frame_rate, frame_size)
    out = cv2.VideoWriter(output_video_path, fourcc, ...
frame_rate, frame_size)
        # print("Length: ", length)


    frame_number = 0
    # print(f"Running Video {video} in the {dirNames['...
runDir']} configuration")
    while True:
        ret, frame, new_video, mask = pe.read(resize, ...
new_resolution)

        if frame is None:
            break

        # print("Frame: ", frame)
        # print("Frame type: ", type(frame))
        if isinstance(frame, tuple):
            # print("This is a tuple")
            frame = frame[1]
            # print("Frame[1] type: ", type(frame))
        if args.filetype != "mp4":
            height, width = frame.shape[:2]
        frame_number += 1
        if frame_number % skip_frames != 0:
            continue

        if ret:
            frame = pe.image_enhancement(frame, ...
image_enhancement, mask=mask, brightness=brightness)
        else:
            print('Video has ended!')
            break

        if new_video:
            new_tracking_model = Tracking_Model(paths["...
DEEPSORT_MODEL"], tracker_type=tracking_model_name)
            pe.tracking_model = new_tracking_model

        pe.detect_and_track(frame)

        evaluater.purge(frame_number)

        counter = 0
        queue_details = None
        tracks = pe.get_tracks()
        # queue_map = {1: [car1, car2, car3], 2: [car1, ...
car2, car3]} where 1, 2, etc is the queue index and ...
```

```
car1, car2 is the id of the car track
        queue_map = {}
        queue_colors = {1: (255, 0, 0), 2: (0, 255, 0), ...
3: (0, 0, 255), 4: (255, 255, 0)}
        for track in pe.get_tracks():
            if not track.is_confirmed() or track....
time_since_update > 1:
                continue

            # print("Track: \n", track)
            # print("Tracks: \n", pe.get_tracks())

            if counter == 0:
                color, text, current_point, next_point, ...
driving_dir, queue_details = evaluater.evaluate(track...
, frame_number, True)
                queue_stats = queue_details[1][0]
                queue_time = queue_details[1][1]
                queue_details = queue_details[0]
                if queue_details:
                    for lane in queue_details:
                        queue_map[lane] = queue_details[...
lane]["tracks"]

                if queue_stats != {}:
                    pe.queue_performance(queue_stats, ...
queue_time)
            else:
                color, text, current_point, next_point, ...
driving_dir = evaluater.evaluate(track, frame_number)

            if args.filetype == 'jpg':
                pe.performance(track, text)
            draw_rectangle(frame, track, color)

            if queue_details:
                found = False
                for lane, track_list in queue_map.items...
():
                    # print(f"Track_list: {track_list}")
                    for _, trackInfo in track_list.items...
():
                        trackId = trackInfo[0]
                        # print(f"track_info: {trackInfo...
}")
                        # print(f"track_id: {track....
track_id}")
                        # print(f"trackId = {trackId}")
                        if track.track_id == trackId:
```

```
                        found = True
                        if lane in queue_colors:
                            color = queue_colors[...
lane]
                        else:
                            color = (255, 255, 255)
                        break
                print(f"found: {found}")
                if found:
                    print(":)")
                    # draw_rectangle(frame, track, color...
)


        # draw_text(frame, track, text)
        if current_point and next_point:
            draw_line(frame, current_point, ...
next_point)
        if driving_dir:
            draw_line(frame, (int(width/2), int(...
height/2)), (int((width/2)+driving_dir[0]), int((...
height/2)+driving_dir[1])))

        if counter == 0:
            if queue_details:
                # lanes = queue_details["...
furthest_apart"]
                for lane in queue_details:
                    lane_details = queue_details[...
lane]["furthest_apart"]
                    for track in tracks:
                        if track.track_id == ...
lane_details[0]:
                            car1 = track
                        if track.track_id == ...
lane_details[1]:
                            car2 = track
                    try:
                        car1 = car1.to_tlwh()
                        car2 = car2.to_tlwh()
                    except AttributeError as e:
                        print("EXEPTION")
                        continue

                    if car1[1] > car2[1]:
                        draw_parallelogram(frame, (...
car2[0], car2[1]), (car2[0] + car2[2], car2[1]), (...
car1[0], car1[1] + car1[3]), (car1[0] + car1[2], car1...
[1] + car1[3]))
```

```
                        else:
                            draw_parallelogram(frame, (...
car1[0], car1[1]), (car1[0] + car1[2], car1[1]), (...
car2[0], car2[1] + car2[3]), (car2[0] + car2[2], car2...
[1] + car2[3]))


            counter += 1

        current_time = time.time() - timeStart
        gpu = GPUtil.getGPUs()
        gpu = gpu[0]
        cpu_usage = psutil.cpu_percent(interval=None)
        computational_data = {'time': current_time, '...
gpu_load_percent': gpu.load*100, 'gpu_memory_used': ...
gpu.memoryUsed, 'gpu_memory_usage': gpu.memoryUtil...
*100, 'cpu_usage': cpu_usage}
        frameData = {'frame': frame, 'frame_number': ...
frame_number, 'current_time': current_time, '...
computational_data': computational_data}

        if args.filetype != 'mp4':
            pe.status(frameData)

        result = np.asarray(frame)
        result = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)
        out.write(result)
        try:

            # if args.file != None or args.mode == "...
analysis":
            # if args.mode == "analysis":

            if args.show == 1:
                cv2.imshow("Output Video", result)

            if cv2.waitKey(1) & 0xFF == ord('q'):
                break
        except:
            print("BREAKING OUT")
            break

        # result = np.asarray(frame)
        # result = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR...
)

        # if args.file != None or args.mode == "analysis...
":
        #     out.write(result)
```

```

        # if args.show == 1:
        #     cv2.imshow("Output Video", result)

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

        percentStorage = percentDone
        if (frame_number / length) * 100 - ...
percentStorage > 1:
            percentDone = np.floor((frame_number / ...
length) * 100)
            print(f"Percent completed {percentDone}%")
    cv2.destroyAllWindows()

    summary, jsonFormat = pe.summary()
    print(summary)
    if filename != '':
        if iterationOptions != None:
            output_file = baseOutputDir + "/" + video....
split(".")[0] + "_" + str(iterationOptions["...
current_iteration_index"]) + ".txt"
            output_json = baseOutputDir + "/" + video....
split(".")[0] + "_" + str(iterationOptions["...
current_iteration_index"]) + ".json"
            print(":)")
        elif new_resolution:
            output_file = baseOutputDir + "/" + video....
split(".")[0] + "_" + new_resolution + ".txt"
            output_json = baseOutputDir + "/" + video....
split(".")[0] + "_" + new_resolution + ".json"
        else:
            output_file = baseOutputDir + "/" + video....
split(".")[0] + ".txt"
            output_json = baseOutputDir + "/" + video....
split(".")[0] + ".json"
        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 += f"noise_type: {noise_type}\n"
            output += f"brightness: {brightness}\n"
            output += summary
            file.write(output)

```

```
        with open(output_json, "w") as file:
            versionInfo = {
                'image_enhancement': image_enhancement,
                'detection': model_name,
                'tracking': tracking_model_name,
                'noise_type': noise_type,
                'brightness_level': brightness
            }
            outputJson = {**versionInfo, **jsonFormat}
            json.dump(outputJson, file)

if __name__ == '__main__':

    if args.datamode == 'json':

        # videoList = extractVideoFile(args.source)
        config = extractJSONFile(args.source)

        # print(videoList)

        argsStorage = args
        runConfig = []
        if config['type'] == 'SessionConfig':
            dirName = config['dir']

            i = 1
            while os.path.exists(os.path.join(r'.\\data...
   \\output', dirName)):
                dirName = config['dir'] + str(i)
                i += 1
            os.mkdir(os.path.join(r'.\\data\\output', ...
   dirName))

            for object in config['runConfig']:
                runConfig.append(extractJSONFile(object)...
   )
        else:
            # runConfig = [extractJSONFile(config)]
            runConfig = config
            dirName = runConfig['dir']

            i = 1
            while os.path.exists(os.path.join(r'.\\data...
   \\output', dirName)):
                dirName = runConfig['dir'] + str(i)
                i += 1
            os.mkdir(os.path.join(r'.\\data\\output', ...
   dirName))
            runConfig = [config]
```

```

        print(runConfig)

        # Runs the main program for each video listed in...
         the json file
        # for video in videoList:
        timeStart  = time.time()
        for file in runConfig:
            # print(file)
            for video in file['data']:

                if file['configurations']['argOverride'...
]:
                    argReplacement(file['configurations'...
]['args'])
                else:
                    args = argsStorage

                if video != None or '':
                    #try:
                    # if True: # Just a testing line
                        # print(video)
                    dirNames = {"sessionDir": None if ...
config['type'] == 'RunConfig' else dirName, "runDir":...
 file['dir'] if config['type'] == 'SessionConfig' ...
else dirName}
                    if args.iterations and args....
iterations > 0:
                        iterations = args.iterations
                        iterationOptions = {"...
max_iterations": int(args.iterations), "...
current_iteration_index": 0}

                    else:
                        iterations = 1
                        iterationOptions = None

                    downscale = False
                    if args.downscale != 0:
                        downscale = True

                    if iterationOptions:
                        for i in range(iterations):
                            iterationOptions['...
current_iteration_index'] = i
                            main(video, dirNames, ...
iterationOptions, downscale)
                        continue

```

```
                    if downscale:
                        resolutions = ['720', '648', '...
    576', '360']
                        for resolution in resolutions:
                            main(video, dirNames, ...
    iterationOptions, resolution)
                        continue

                    main(video, dirNames, ...
    iterationOptions, downscale)

                    #except Exception as e:
                        #print("This is the error: "+ ...
    str(e))
                        # print(e.message)
                        #continue
                # print(";)")
                else:
                    print("Video was None or ''")
        timeEnd = time.time()
        totalTime = timeEnd - timeStart
        print("Total time: "+ str(totalTime))

    else:
        main(args.source, args.source.split(".")[0] if ...
    args.mode == "analysis" else args.file)
```

Kode B.1: liveTrack.py was altered for this thesis and its original source code was part of the thesis written by Aleksander Vedvik and can be found under run.py in the GitHub repository [48]

```
import os
import tensorflow as tf
import torch
from ultralytics import YOLO

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
        self.init_model()

    @property
    def class_ids(self):
        return self._class_ids

    @class_ids.setter
    def class_ids(self, classes):
        class_ids = {}
        for class_name in classes:
            id = classes[class_name]
```

```
            class_ids[id] = class_name
        self._class_ids = class_ids

    def init_model(self):
        if self.model_type == "yolov5":
            model = torch.hub.load('ultralytics/yolov5',...
 'yolov5x')
            # YOLOv5 stores and use cache by default. ...
Use the line below instead if there are any problems ...
with cache.
            # model = torch.hub.load('ultralytics/yolov5...
', 'yolov5x', force_reload=True)
            self.model = model
        elif self.model_type == "yolov5_trained":
            model = torch.hub.load('ultralytics/yolov5',...
 'custom', path='training/yolov5/yolov5/runs/train/...
yolov5x_trained/weights/best.pt')
            self.model = model
        elif self.model_type == "yolov7":
            model = torch.hub.load('WongKinYiu/yolov7', ...
'custom', 'yolov7x.pt')
            self.model = model
        elif self.model_type == "yolov8":
            model = YOLO("yolov8x.pt")
            self.model = model
        else:
            """
            ...
******************************************************************************

            The code below until the END statement 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

            ...
******************************************************************************

            """
            # gpus = tf.config.experimental....
list_physical_devices('GPU')
            # for gpu in gpus:
            #     tf.config.experimental....
```

```
set_memory_growth(gpu, True)

            # self.configs = config_util....
get_configs_from_pipeline_file(self.paths['...
PIPELINE_CONFIG'])
            # self.model = model_builder.build(...
model_config=self.configs['model'], is_training=False...
)

            # self.ckpt = tf.compat.v2.train.Checkpoint(...
model=self.model)
            # self.ckpt.restore(os.path.join(self.paths...
['CHECKPOINT_PATH'], self.ckpt_no)).expect_partial()
            # self.category_index = label_map_util....
create_category_index_from_labelmap(self.paths['...
LABELMAP'])
            """
            END
            """

    def detect(self, frame, w=0, h=0):
        return_object = {"frame": frame, "boxes": [], "...
scores": [], "object_classes": []}

        if self.model_type == "yolov5" or self....
model_type == "yolov5_trained" or self.model_type == ...
"yolov7":
            results = self.model(frame)
            # print("\nYOLOV5!!!!!!!!!!!!:", results, "\...
n", type(results), "\n...
===============================")
            df = results.pandas().xyxy[0]
            # print("\nYOLOV5 DF !!!!!!!!!!!!:", df, "\n...
", type(df), "\n===============================")
            for row in df.itertuples():
                obj_class = str(row[7]).lower()
                if obj_class not in self.classes:
                    # NOTE: This continue was removed to...
 allow for the model to classify anything it found as...
 a road anomaly. This is a lackluster approach to ...
introduce Road anomaly detections.
                    # No real statistics can be drawn ...
from this as no road anomaly was annotated in the ...
dataset and therefore we have no way of verifying the...
 accuracy of the detection
                    # continue
                    obj_class = "Road anomaly"
                return_object["boxes"].append([float(row...
[1]), float(row[2]), (float(row[3])-float(row[1])), (...
```

```
float(row[4])-float(row[2]))])
                return_object["scores"].append(float(row...
[5]))
                return_object["object_classes"].append(...
obj_class)
        elif self.model_type == "yolov8":
            results = self.model.predict(source=[frame])
            result = results[0].cuda().cpu().to("cpu")....
numpy()
            for index in range(len(result.boxes.xyxy)):
                box = result.boxes.xyxy[index]
                obj_class = result.boxes.cls[index]
                return_object["boxes"].append([float(box...
[0]), float(box[1]), (float(box[2])-float(box[0])), (...
float(box[3])-float(box[1]))])
                return_object["scores"].append(float(...
result.boxes.conf[index]))
                return_object["object_classes"].append(...
obj_class)
        else:
            image_np = np.array(frame)
            input_tensor = tf.convert_to_tensor(np....
expand_dims(image_np, 0), dtype=tf.float32)
            detections = detect_fn(input_tensor, self....
model)

            num_detections = int(detections.pop('...
num_detections'))
            detections = {key: value[0, :num_detections...
].numpy()
                    for key, value in detections....
items()}
            detections['num_detections'] = ...
num_detections

            detections['detection_classes'] = detections...
['detection_classes'].astype(np.int64)

            for i, score in enumerate(detections["...
detection_scores"]):
                if float(score) ≥ self.CONFIDENCE_LEVEL:
                    x1 = float(detections['...
detection_boxes'][i][1]) * float(w)
                    x2 = float(detections['...
detection_boxes'][i][3]) * float(w)
                    y1 = float(detections['...
detection_boxes'][i][0]) * float(h)
                    y2 = float(detections['...
detection_boxes'][i][2]) * float(h)
```

```
                        class_id = str(int(detections['...
detection_classes'][i]) + 1)
                        obj_class = self.class_ids.get(...
class_id)

                        return_object["boxes"].append([x1, ...
y1, (x2-x1), (y2-y1)])
                        return_object["scores"].append(float...
(score))
                        return_object["object_classes"]....
append(obj_class)

        return return_object
```

Kode B.2: detection_model.py was altered for this thesis and its original source code was a part of the thesis written by Aleksander Vedvik for which the source code can be found at [48]

```
import math
import numpy as np
import networkx as nx
from sklearn.cluster import DBSCAN
import time
from scipy.spatial.distance import pdist, squareform

class Evaluate_Incidents:
    def __init__(self, classes, colors=None, ...
driving_direction=None):
        self.classes = classes
        self.colors = colors
        self.objects = {}
        self.driving_direction = driving_direction
        self.TTL = 240  # Number of frames before a ...
track is removed
        self.PF = 2#7  # PF = Previous Frame: Number of ...
frames used to determine direction
        self.STOPPED_DISTANCE = 3  # Distance in number ...
of pixels from current to previous frame to determine...
 stopped vehicle
        self.DIRECTION_THRESHOLD = 10  # Amount the x ...
and y vectors can deviate when determining if vehicle...
 is wrong-way driving
        self.min_number_of_frames = 2 # 24  # How many ...
frames must there be to evaluate stopped vehicle
        self.update_number_of_frames = 2#12  # How often...
 stopped vehicle should be evaluated
        self.min_number_of_driving_directions = 5 # How ...
```

```
many driving directions needed to create a general ...
vector for driving directions

        #Queue detections:
        self.queue_detection_radius = 100 # Radius value...
 used in simple queue detection
        self.dbscan_eps = 1 # Epsilon value used for the...
 DBSCAN clustering algorithm
        self.min_queue_size = 3 # Minimum number of ...
vehicles in proximity to a core point to be ...
considered a queue
        self.common_driving_direction = (93, -130) #...
(133, -100) #(93, -130) # This is the mathematical ...
vector definition for direction of traffic flow (x, y...
)
        self.secondary_driving_direction = (133, -100) #...
 NOTE: Static value that have no actual function ...
currently due to not being implemented
        self.driving_direction_margin = 10 # Tolerance ...
zone for being considered driving in the same lane
        self.queues = {}
        # self.queue_map = {}

    @property
    def colors(self):
        return self._colors

    @colors.setter
    def colors(self, colors):
        colors_default = {"alarm": (255,128,128), "ok": ...
(128,128,255), "queue": (15,255,80)}
        if colors and colors.get("alarm") and colors.get...
("ok"):
            colors_default = colors
        self._colors = colors_default

    @property
    def driving_direction(self):
        return self._driving_direction

    @driving_direction.setter
    def driving_direction(self, driving_direction):
        # Driving direction should be defined with an ...
upstream and downstream direction
        # Each direction should be defined as a vector: ...
[x, y]
        if driving_direction is None:
            driving_direction = {"Upstream": [], "...
Downstream": []}
```

```
        if driving_direction.get("Upstream") is None:
            driving_direction["Upstream"] = []
        if driving_direction.get("Downstream") is None:
            driving_direction["Downstream"] = []
        self._driving_direction = driving_direction

    def purge(self, frame_number):
        if frame_number % 24 != 0:
            return
        dict_of_objects = self.objects.copy()
        for object in dict_of_objects:
            if dict_of_objects[object]["last_frame"] < ...
    frame_number - self.TTL:
                del self.objects[object]

    # Can be used to calculate direction based on center...
     points from several frames
    def simple_linear_regression(self, track_id, ...
    frame_number):
        track = self.objects[track_id]
        n = len(track["center_points"])
        if n ≤ 5:
            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:
                next_point_x = current_point[0] + ...
    direction["distance"]
                next_point_y = direction["alpha"] + ...
    direction["beta"] * next_point_x
                next_point = (int(next_point_x), int(...
    next_point_y))

                return current_point, next_point

        if n > 10:
            n = 10
        center_points = track["center_points"][-n:]

        x_sum = 0
        y_sum = 0
        for center_point in center_points:
            x_sum += center_point[0]
            y_sum += center_point[1]

```

```
        x_mean = x_sum / n
        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))

                return current_point, next_point

        previous_point = track["center_points"][-self.PF...
]

        x_vector = current_point[0] - previous_point[0]
        y_vector = current_point[1] - previous_point[1]
        length_vector = math.sqrt(x_vector**2 + y_vector...
**2)
        try:
            x_vector /= length_vector
            y_vector /= length_vector
        except Exception as e:
            print(e)
            return None, None

        length= 50

        next_point = (int(current_point[0] + x_vector * ...
length), int(current_point[1] + y_vector * length))

        self.objects[track_id]["direction"] = {"length":...
 length, "x_vector": x_vector, "y_vector": y_vector}
        return current_point, next_point

    # TODO: Implement angular speed (Current ...
implementation only contains a simple solution for ...
distance from last center point to current point)
    def simple_speed(self, track_id):
        if track_id not in self.objects:
            print("Track id not in self.objects")
            return -1
        track = self.objects[track_id]
        n = len(track["center_points"])
        if n ≤ self.min_number_of_frames:
            print("min number of frames not met")
            return -1

        current_point = (int(track["center_points"...
][-1][0]), int(track["center_points"][-1][1]))
        previous_point = track["center_points"][-self.PF...
]
```

```

        speed = math.sqrt((current_point[0] - ...
previous_point[0])**2 + (current_point[1] - ...
previous_point[1])**2)

        # self.objects[track_id]["speed"] = distance
        return speed

    def pedestrian(self, class_name):
        if class_name == "person":
            return True
        return False

    def stopped_vehicle(self, track_id, frame_number):
        track = self.objects[track_id]
        n = len(track["center_points"])
        if n ≤ self.min_number_of_frames:
            return False

        if frame_number % self.update_number_of_frames ...
!= 0:
            stopped = self.objects[track_id].get("...
stopped")
            if stopped:
                return True
            return False

        current_point = (int(track["center_points"...
][-1][0]), int(track["center_points"][-1][1]))
        previous_point = track["center_points"][-self.PF...
]

        distance = math.sqrt((current_point[0] - ...
previous_point[0])**2 + (current_point[1] - ...
previous_point[1])**2)

        if distance ≤ self.STOPPED_DISTANCE:
            self.objects[track_id]["stopped"] = True
            return True
        self.objects[track_id]["stopped"] = False
        return False

    def wrong_way_driving(self, track_id, frame_number, ...
current_point, next_point, lane="Upstream"):
        if len(self.driving_direction.get(lane)) ≤ 0:
            return False

        track = self.objects[track_id]
        n = len(track["center_points"])
```

```
        if n ≤ self.min_number_of_frames:
            return False

        if frame_number % self.update_number_of_frames ...
!= 0:
            wrong_way = self.objects[track_id].get("...
wrong_way")
            if wrong_way:
                return True
            return False

        if not current_point or not next_point:
            return False

        # print("Next point: ", next_point)
        # print("Current point: ", current_point)

        vehicle_direction = [next_point[0] - ...
current_point[0], next_point[1] - current_point[1]]
        lane_direction = self.driving_direction.get(lane...
)

        if abs(lane_direction[0] - vehicle_direction[0])...
 < self.DIRECTION_THRESHOLD and abs(lane_direction[1]...
 - vehicle_direction[1]) < self.DIRECTION_THRESHOLD:
            self.objects[track_id]["wrong_way"] = False
            return False
        self.objects[track_id]["wrong_way"] = True
        return True
    # TODO: Claim Credit
    def same_lane_driving(self, center_point1, ...
center_point2):
        lane_vector = [center_point2[0] - center_point1...
[0], center_point2[1] - center_point1[1]]

        angle_radians = math.atan2(lane_vector[1], ...
lane_vector[0])
        angle_degrees = math.degrees(angle_radians)

        # print(self.common_driving_direction)
        driving_direction_angle = math.degrees(math....
atan2(self.common_driving_direction[0], self....
common_driving_direction[1]))
        angle_difference = abs(angle_degrees - ...
driving_direction_angle)

        angle_difference = min(angle_difference, 360 - ...
angle_difference)
        print("Angle difference: ", angle_difference)
```

```python

        return angle_difference <= self.driving_direction_margin or 180 - angle_difference <= self.driving_direction_margin

    def cars_furthest_apart(coordinates, cluster_indices):
        pairwise_distance = squareform(pdist(coordinates[cluster_indices]))

        furthest_pair_indices = np.unravel_index(np.argmax(pairwise_distance, axis=None), pairwise_distance.shape)

        return cluster_indices[furthest_pair_indices[0]], cluster_indices[furthest_pair_indices[1]]

    def cars_furthest_apart_simple(self, cars):
        max_distance = 0
        car_pair = None

        for i in range(len(cars)):
            for j in range(len(cars)):
                if i == j:
                    continue
                distance = np.sqrt((cars[i][0] - cars[j][0])**2 + (cars[i][1] - cars[j][1])**2)

                if distance > max_distance:
                    max_distance = distance
                    car_pair = [cars[i][2], cars[j][2]]

        return car_pair

    # TODO: Implement / Claim Credit
    def queue(self, frame_number):
        start_time = time.time()
        queue_map = {} # Map of all detected queues
        track_to_queue_map = {} # Simple map for tracking the lane a track belongs to {trackId: laneId}
        furthest_apart = {} # Map of the cars furthest apart in each lane {laneId: [CarId1, CarId2]}
        amount_of_queues = 0
        filtered_tracks = {key: val for key, val in self.objects.items() if frame_number - val.get('last_frame') <= 1}

        for track_id in self.objects:
```

```
            track = self.objects[track_id]
            class_to_id = {0: 'None',1: 'car', 2: '...
person', 3: 'truck', 4: 'bus', 5: 'bike', 6: '...
motorbike', 10: 'Road anomaly'}

            print(track["class"])
            if track["class"] in class_to_id:
                track["class"] = class_to_id[track["...
class"]]
            try:
                track["class"].upper()
            except AttributeError as e:
                track["class"] = "NONE"

        for track_id in filtered_tracks:
            track = self.objects[track_id]

            dif = frame_number - track['last_frame']
            if dif > 1:
                continue

            if track["speed"] == -1 or len(track["...
center_points"]) < 1:
                continue

            if track["speed"] > 10:
                continue

            vehicles = ["CAR", "BUS", "TRUCK", "STOPPED ...
VEHICLE"]
            if track["class"].upper() not in vehicles:
                continue

            cx1, cy1 = track["center_points"][-1] # ...
Center point
            if track_id in track_to_queue_map:
                lane = track_to_queue_map[track_id]
            else:
                if track_to_queue_map == {}:
                    lane = 1
                else:
                    lane = len(track_to_queue_map) + 1

            for track_id2 in filtered_tracks:
                track2 = self.objects[track_id2]
                if track_id == track_id2:
                    continue
                if track2["speed"] == -1 or len(track2["...
center_points"]) < 2:
```

```
                continue

            if track2["class"].upper() not in ...
vehicles:
                continue

            dif2 = frame_number - track2['last_frame...
']
            if dif2 > 1:
                continue

            x, y = track2["center_points"][-1]
            distance = np.sqrt((x - cx1)**2 + (y - ...
cy1)**2)

            if distance > self....
queue_detection_radius or distance < 10:
                continue
            if not self.common_driving_direction:
                continue

            if self.same_lane_driving((cx1, cy1), (x...
, y)):
                trackInfo = {"center_point": (cx1, ...
cy1), "speed": track["speed"], "track": track}
                trackInfo2 = {"center_point": (y, x)...
, "speed": track2["speed"], "track": track2}
                print(f"These cars are same lane ...
driving {track_id} and {track_id2}.")
                if track_id not in ...
track_to_queue_map:
                    if lane not in queue_map:
                        queue_map[lane] = {}
                    queue_map[lane][track_id] = ...
trackInfo
                    queue_map[lane][track_id2] = ...
trackInfo2
                    track_to_queue_map[track_id] = ...
lane
                    track_to_queue_map[track_id2] = ...
lane
                else:
                    if lane not in queue_map:
                        queue_map[lane] = {}
                    queue_map[lane][track_id2] = ...
trackInfo2
                    track_to_queue_map[track_id2] = ...
lane

```

```
        for lane in queue_map:
            cars = []
            lane_id = lane
            lane = queue_map[lane]
            for car in lane:
                cars.append((lane[car]["center_point"...
    ][0], lane[car]["center_point"][1], car))

            found_in_lane = None
            for l in self.queues:
                counter = 0
                for car in lane:
                    if car in self.queues[l]:
                        counter += 1
                if counter ≥ self.min_queue_size:
                    found_in_lane = l
                    break

            if not found_in_lane:
                n = len(self.queues) + 1
                self.queues[n] = [c[2] for c in cars]


            print(f"Cars lenght: {len(cars)}")
            if len(cars) < self.min_queue_size:
                # continue
                print(":)")
            furthest_apart[lane_id] = self....
    cars_furthest_apart_simple(cars)

        # lanes = []
        # print("Track to queue map: ", ...
    track_to_queue_map)
        # for lane in furthest_apart:
        #     lanes.append(furthest_apart[lane])
        laneDetails = {}
        # print("Track to queue map: ", ...
    track_to_queue_map)
        for lane in furthest_apart:
            queue_lane = queue_map[lane]
            # print(f"Queue_lane: {queue_lane}")
            car_ids = furthest_apart[lane]
            if not self.same_lane_driving(self.objects[...
    car_ids[0]]["center_points"][-1], self.objects[...
    car_ids[1]]["center_points"][-1]):
                continue
            tracks = {lane: [car, car_info["track"]] for...
     car, car_info in queue_lane.items()}
```

```
                laneDetails[lane] = {"furthest_apart": ...
furthest_apart[lane], "tracks": tracks}

        end_time = time.time()
        total_time = end_time - start_time

        return laneDetails, (self.queues, total_time)




    # TODO: Implement / Claim Credit
    def queue_dbscan(self):
        # features = np.array([])
        features = [] # features is a 2D array ...
containing the centerpoint and speed of a vehicle
        track_map  = [] # Track_map maps the track with ...
the key feature_id to the track and track_id since ...
not all tracks make it to the feature array
        feature_counter = 0 # Serves as the current ...
index of the feature array
        for track_id in self.objects:
            track = self.objects[track_id]
            # print("Track: ", track)
            if track["speed"] and len(track["...
center_points"]) > 2:
                x1, y1 = track["center_points"][-1]
                # x2, y2 = track["center_points"][-1]
                # features.append([x1, y1, x2, y2, track...
["speed"]])
                features.append([x1, y1, track["speed"...
]])
                track_map.append({"track_id": track_id, ...
"track": track})
                feature_counter += 1

            # print("Features: \n", features)
        if len(features) > 4:
            features = np.array(features)
            dbscan = DBSCAN(eps=self.dbscan_eps, ...
min_samples=self.min_queue_size)
            clusters = dbscan.fit_predict(features)

            unique_clusters = set(clusters) - {-1}

            refined_clusters = []

            for cluster_id in unique_clusters:
                indices = np.where(clusters == ...
```

```
cluster_id)[0]
                cluster_features = features[indices]
                cluster_tracks = track_map[indices]

                G = nx.Graph()

                for index in indices:
                    G.add_node(index)

                for i in range(len(indices)):
                    for j in range(i + 1, len(indices)):
                        if self.same_lane_driving(...
cluster_tracks[i]["track"]["center_points"][-1], ...
cluster_tracks[j]["track"]["center_points"][-1]):
                            G.add_edge(indices[i], ...
indices[j])

                for component in nx.connected_components...
(G):
                    refined_clusters_indices = list(...
component)
                    refined_clusters.append(...
refined_clusters_indices)

            # refined_clusters_track_ids is a list of ...
all clustered vehicles with their original track id
            refined_clusters_track_ids = []
            for cluster in refined_clusters:
                track_ids = [track_map[i] for i in ...
cluster]
                refined_clusters_track_ids.append(...
track_ids)

            coordinates = np.array([...])
            cluster_assignments = [...]
            for cluster_id in refined_clusters:
                cluster_inices = [i for i, x in ...
enumerate(cluster)]

            # print("Refined clusters: \n", ...
refined_clusters)


            # print("Clusters: \n", clusters)

    def evaluate(self, track, frame_number, eval_queue=...
False):
        class_name = track.get_class()
        text = f"{class_name} - {track.track_id}"
```

```
        color = self.colors["ok"]
        bbox = track.to_tlbr()
        center_point = ((int(bbox[0]) + (int(bbox[2]) - ...
int(bbox[0])) / 2), int(bbox[1]) + (int(bbox[3]) - ...
int(bbox[1])) / 2)
        speed = self.simple_speed(track.track_id)
        # print(f"Speed: {speed}")
        if track.track_id in self.objects:
            self.objects[track.track_id]["center_points"...
].append(center_point)
            self.objects[track.track_id]["last_frame"] =...
 frame_number
            self.objects[track.track_id]["speed"] = ...
speed
            self.objects[track.track_id]["class"] = ...
class_name
        else:
            self.objects[track.track_id] = {"...
center_points": [center_point], "last_frame": ...
frame_number, "speed": speed, "class": class_name}

        # Used to determine vehicle direction:
        #   Current_point is the current center location...
 of the vehicle
        #   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)
        current_point, next_point = self....
simple_direction(track.track_id, frame_number)

        # if len(self.driving_direction) ≥ self....
min_number_of_driving_directions:
        # if current_point:
        #     self.common_driving_direction = (...
next_point[0] - current_point[0], next_point[1] - ...
current_point[1])




        if self.pedestrian(class_name):
            color = self.colors["alarm"]
            text = "INCIDENT: Pedestrian"
            current_point, next_point = None, None
        elif self.stopped_vehicle(track.track_id, ...
frame_number):
            color = self.colors["alarm"]
            text = "INCIDENT: Stopped vehicle"
```

```
                current_point, next_point = None, None
            elif self.wrong_way_driving(track.track_id, ...
        frame_number, current_point, next_point):
                print("WRONG WAY DRIVER!!!")
                color = self.colors["alarm"]
                text = "INCIDENT: Wrong-way driver"
            # TODO: Introduce a queue state to this if ...
        statement

            # print(self.driving_direction["Upstream"])
            # TODO: Finish implementation
            if eval_queue:
                # print("eval_queue frame_number:", ...
        frame_number)
                queue_details, queue_stats = self.queue(...
        frame_number)
                # print("Queue details: ", queue_details)
                # return color, text, current_point, ...
        next_point, self.common_driving_direction, ...
        queue_details
                if self.driving_direction["Upstream"]:
                    return color, text, current_point, ...
        next_point, (self.driving_direction["Upstream"][0], ...
        self.driving_direction["Upstream"][1]), [...
        queue_details, queue_stats]
                else:
                    return color, text, current_point, ...
        next_point, ([]), [queue_details, queue_stats]

            # return color, text, current_point, next_point,...
         self.common_driving_direction
            if self.driving_direction["Upstream"]:
                return color, text, current_point, ...
        next_point, (self.driving_direction["Upstream"][0], ...
        self.driving_direction["Upstream"][1])
            else:
                return color, text, current_point, ...
        next_point, ([])
```

Kode B.3: incident_evaluator.py was altered for this thesis and its original source code was a part of the thesis written by Aleksander Vedvik for which the source code can be found at [48]

```
import json
import cv2
import os
import time
```

```
import numpy as np
from helpers.retinex import SSR
from helpers.retinex import MSR
from noise_manager import Noise_Manager

class Evaluate_Performance:
    def __init__(self, type, dataset_path, classes, ...
    detection_model, tracking_model, mask="", noise_type=...
    None):
        self.vid = None
        self.width = 0
        self.height = 0
        self.scale = 1
        self.entries = []
        self.next_entry_index = 0
        self.type = type
        self.detected_objects = []
        self.detected_objects_previous = {}
        self.dataset_paths = dataset_path
        self.datasets = {}
        self.classes = classes
        self.detection_model = detection_model
        self.tracking_model = tracking_model
        self.current_video = ""
        self.mask = mask
        self.queue = {}
        self.prepare()

        self.detection_time_current = 0
        self.tracking_time_current = 0
        self.total_time_current = 0
        self.fps_current = 0

        self.image_enhancement_current = 0
        self.mean_image_enhancement_time = 0

        self.mean_detection_time = 0
        self.min_detection_time = -1
        self.max_detection_time = 0

        self.mean_tracking_time = 0
        self.min_tracking_time = -1
        self.max_tracking_time = 0

        self.mean_total_time = 0
        self.min_total_time = -1
        self.max_total_time = 0

        self.missed_detections = 0
```

```
        self.total_number_of_real_detections = 0
        self.total_number_of_valid_detections = 0
        self.total_number_of_valid_detections_adjusted =...
    0
        self.false_positives_detections = 0
        self.false_positives_detections_previous = 0

        self.missed_tracks = 0

        self.detection_accuracy = 0
        self.detection_accuracy_adjusted = 0
        self.tracking_accuracy = 0
        self.tracking_id_switches = 0
        self.tracking_id_duplicates = 0
        self.incident_accuracy = 0
        self.missed_incidents = 0
        self.false_alarms = 0

        self.mean_fps = 0
        self.min_fps = -1
        self.max_fps = 0

        self.number_of_frames = 0

        self.cars_outside_mask = 0
        self.objects_inside_mask = 0

        self.queues_detected = 0
        self.queue_changes = 0
        self.mean_queue_changes = 0
        self.max_queue_length = 0
        self.min_queue_length = -1
        self.avg_queue_length = 0
        self.min_queue_detection_time = -1
        self.max_queue_detection_time = 0
        self.mean_queue_detection_time = 0
        self.queue_detection_time = []

        self.track_values = {}
        self.frame_queues = {}
        self.frameData = []

        self.noise_manager = Noise_Manager(noise_type)

    @property
    def dataset_paths(self):
        return self._dataset_paths

    @dataset_paths.setter
```

```
    def dataset_paths(self, datasets):
        dataset_paths = {}
        for dataset in datasets:
            images = dataset.get("images")
            annotations = dataset.get("annotations")
            dataset_name = dataset.get("dataset")

            video = dataset.get("video")
            if video:
                dataset_paths["video"] = video
            elif images is None or annotations is None ...
or dataset_name is None:
                continue
            else:
                dataset_paths[dataset_name] = {"images":...
 images, "annotations": annotations}

        self._dataset_paths = dataset_paths

    def prepare(self):
        if self.type == "Video":
            self.vid = cv2.VideoCapture(self....
dataset_paths.get("video"))
            self.width = int(self.vid.get(cv2....
CAP_PROP_FRAME_WIDTH))
            self.height = int(self.vid.get(cv2....
CAP_PROP_FRAME_HEIGHT))
        elif self.type == "Images":
            print("Doing this dataset thing")
            for dataset_name in self.dataset_paths:
                try:
                    if "self_annotated" in dataset_name:
                        self.prepare_self_annotated(...
dataset_name)
                except Exception as e:
                    print(e)
                    self.datasets[dataset_name] = {"...
entries": []}
            self.prepare_all_entries()

        mask = cv2.imread(self.mask, cv2....
IMREAD_GRAYSCALE)
        _, self.mask = cv2.threshold(mask, 127, 255, cv2...
.THRESH_BINARY_INV)

    def prepare_self_annotated(self, dataset_name="...
self_annotated"):
        dataset = self.dataset_paths.get(dataset_name)
        # print("Dataset value: ", dataset)
```

```
        if dataset is None:
            return
        anno_path = dataset.get("annotations")
        img_path = dataset.get("images")
        mask_path = anno_path.replace("annotations.json"...
    , "mask.png")

        if anno_path is None:
            return

        with open(anno_path, "r") as annotations:
            data = json.load(annotations)

        annotation_classes_path = anno_path.replace("...
    annotations", "classes")
        with open(annotation_classes_path, "r") as ...
    annotation_classes:
            annotation_classes = json.load(...
    annotation_classes)

        images_list = {"entries": []}
        for i, img in enumerate(data):
            if i ≤ 0:
                continue
            filename = img
            row = {"images_path": img_path, "filename": ...
    filename, "objects": [], "mask_path": mask_path }

            for object in data[img]['instances']:

                info = {}
                for class_ in annotation_classes:
                    if class_["id"] == object["classId"...
    ]:
                        class_name = class_["name"]

                        for object_attribute in object["...
    attributes"]:
                            for attribute_group in ...
    class_["attribute_groups"]:
                                if object_attribute["...
    groupId"] == attribute_group["id"]:
                                    for attribute_ in ...
    attribute_group["attributes"]:
                                        if attribute_["...
    id"] == object_attribute["id"]:
                                            info[...
    attribute_group["name"]] = attribute_["name"]

```

```
                if class_name == "people":
                    class_name = "person"

                if class_name not in self.classes:
                    continue

                x1 = float(object["points"]["x1"])
                y1 = float(object["points"]["y1"])
                x2 = float(object["points"]["x2"])
                y2 = float(object["points"]["y2"])

                row["objects"].append({"class": ...
class_name, "class_id": self.classes.get(class_name),...
  "x1": x1, "y1": y1, "x2": x2, "y2": y2, "info": ...
info})

            images_list["entries"].append(row)

        images_list['entries'] = sorted(images_list['...
entries'], key = lambda i: i['filename'])
        self.datasets[dataset_name] = images_list

    def prepare_all_entries(self):
        if self.type != "Images":
            return

        print("\nEntries:")
        classes = {}
        number_of_objects = 0
        entries = []
        for dataset in self.datasets:
            for entry in self.datasets[dataset]["entries...
"]:
                entries.append(entry)
                number_of_objects += len(entry["objects"...
])
                for obj in entry["objects"]:
                    if obj["class"] in classes:
                        classes[obj["class"]] += 1
                    else:
                        classes[obj["class"]] = 1

        print(f"Number of files: {len(entries)}")
        print(f"Number of objects: {number_of_objects}")
        for obj_class in classes:
            print(f" - {obj_class}: {classes[obj_class]}...
")
        self.entries = entries

```

```
    def performance(self, track, text):
        bbox = track.to_tlbr()
        object_class = track.get_class()
        track_id = track.track_id

        x1 = bbox[0]
        y1 = bbox[1]
        x2 = bbox[2]
        y2 = bbox[3]

        incident = False
        best_IoU = {"score": 0, "object": None, "...
real_object": None}
        for real_object in self.entries[self....
next_entry_index-1]["objects"]:
            real_object["x1"] *= self.scale
            real_object["y1"] *= self.scale
            real_object["x2"] *= self.scale
            real_object["y2"] *= self.scale
            if x1 > real_object["x1"]:
                x_min = x1
            else:
                x_min = real_object["x1"]
            if y1 > real_object["y1"]:
                y_min = y1
            else:
                y_min = real_object["y1"]
            if x2 < real_object["x2"]:
                x_max = x2
            else:
                x_max = real_object["x2"]
            if y2 < real_object["y2"]:
                y_max = y2
            else:
                y_max = real_object["y2"]

            intersection_area = (x_max - x_min) * (y_max...
 - y_min)
            if intersection_area < 0 or (x_max - x_min) ...
< 0 or (y_max - y_min) < 0:
                continue

            union_area = ((real_object["x2"] - ...
real_object["x1"]) * (real_object["y2"] - real_object...
["y1"])) + ((x2 - x1) * (y2 - y1)) - ...
intersection_area
            if union_area < 0:
                print(f"IA: {intersection_area}")
                print(f"UA: {union_area}")
```

```
                print(f"RO: x1 = {real_object['x1']}, y1...
 = {real_object['y1']}, x2 = {real_object['x2']}, y2 ...
= {real_object['y2']}")
                print(f"DO: x1 = {x1}, y1 = {y1}, x2 = {...
x2}, y2 = {y2}")
                raise ValueError

            IoU = intersection_area / union_area

            if (IoU - 1)**2 < (best_IoU["score"] - 1)...
**2:
                best_IoU["object"] = {"bbox": bbox, "...
class": object_class, "ID": track_id}
                best_IoU["real_object"] = real_object
                best_IoU["score"] = IoU

        if best_IoU["object"] is not None and best_IoU["...
score"] > 0.4:
            if best_IoU["real_object"]['info']['status']...
 == "Incident":
                incident = True
            self.detected_objects.append(best_IoU)
        else:
            self.false_positives_detections += 1

        if incident and ("Stopped vehicle" in text or "...
Pedestrian" in text):
            self.incident_accuracy += 1
        elif incident:
            self.missed_incidents += 1
        elif "Stopped vehicle" in text or "Pedestrian" ...
in text:
            self.false_alarms += 1

        if self.track_values == {}:
            self.track_values = {
                'centerpoint': {
                    'x': [(x_min + x_max) / 2],
                    'y': [(y_min + y_max) / 2]
                },
                'true_labels': [],
                'predicted_labels': []
            }
        else:
            self.track_values['centerpoint']['x'].append...
((x_min + x_max) / 2)
            self.track_values['centerpoint']['y'].append...
((y_min + y_max) / 2)

```

```



    def frame_analytics(self, frameData):
        print(":I")

    def image_enhancement(self, frame, image_enhancement...
="", mask=None, brightness=None):
        img_enh_start = time.time()
        # frame = frame[1]
        # print(frame)
        # print("Image Enhancement: ", image_enhancement...
)
        if image_enhancement == "gray_linear":
            frame = cv2.cvtColor(frame, cv2....
COLOR_BGR2GRAY)
            frame = cv2.cvtColor(frame, cv2....
COLOR_GRAY2RGB)
            # print("Gray Linear Enhancement")
            # print(frame)
        elif image_enhancement == "gray_nonlinear":
            # print("Gray Non Linear Enhancements")
            frame = cv2.cvtColor(frame, cv2....
COLOR_BGR2GRAY)
            gamma=2.0
            invGamma = 1.0 / gamma
            table = np.array([((i / 255.0) ** invGamma) ...
* 255
                for i in np.arange(0, 256)]).astype("...
uint8")
            frame = cv2.LUT(frame, table)
            frame = cv2.cvtColor(frame, cv2....
COLOR_BGR2RGB)
        elif image_enhancement == "he":
            frame = cv2.cvtColor(frame, cv2....
COLOR_BGR2GRAY)
            frame = cv2.equalizeHist(frame)
            frame = cv2.cvtColor(frame, cv2....
COLOR_GRAY2RGB)
        elif image_enhancement == "retinex_ssr":
            variance=300
            img_ssr=SSR(frame, variance)
            frame = cv2.cvtColor(img_ssr, cv2....
COLOR_BGR2RGB)
        elif image_enhancement == "retinex_msr":
            variance_list=[200, 200, 200]
            img_msr=MSR(frame, variance_list)
            frame = cv2.cvtColor(img_msr, cv2....
COLOR_BGR2RGB)
```

```
        elif image_enhancement == "mask":
            frame = cv2.bitwise_and(frame, frame, mask=...
    mask)
            frame = cv2.cvtColor(frame, cv2....
    COLOR_BGR2RGB)
        else:
            frame = cv2.cvtColor(frame, cv2....
    COLOR_BGR2RGB)

        if brightness is not None:
            brightness_percent = 1 + (brightness / 100)
            frame = cv2.convertScaleAbs(frame, alpha=...
    brightness_percent, beta=0)

        if self.noise_manager.noise_type is not None:
            frame = self.noise_manager.add_noise(frame)

        img_enh_end = time.time()
        self.image_enhancement_current = img_enh_end - ...
    img_enh_start
        self.mean_image_enhancement_time += self....
    image_enhancement_current

        return frame

    def detect(self, frame):
        detection_start = time.time()
        model_detections = self.detection_model.detect(...
    frame, self.width, self.height)
        detection_end = time.time()
        self.detection_time_current = detection_end - ...
    detection_start

        if self.detection_time_current < 10:
            self.mean_detection_time += self....
    detection_time_current

        if self.min_detection_time == -1 or (self....
    detection_time_current < self.min_detection_time and ...
    self.detection_time_current > 0):
            self.min_detection_time = self....
    detection_time_current
        if self.detection_time_current > self....
    max_detection_time and self.detection_time_current < ...
    10:
            self.max_detection_time = self....
    detection_time_current

        return model_detections
```

```

    def track(self, model_detections):
        track_start = time.time()
        self.tracking_model.track(model_detections)
        track_end = time.time()
        self.tracking_time_current = track_end - ...
track_start

        self.mean_tracking_time += self....
tracking_time_current

        if (self.min_tracking_time == -1 or self....
tracking_time_current < self.min_tracking_time) and ...
self.tracking_time_current > 0:
            self.min_tracking_time = self....
tracking_time_current
        if self.tracking_time_current > self....
max_tracking_time:
            self.max_tracking_time = self....
tracking_time_current

    def detect_and_track(self, frame):
        self.number_of_frames += 1
        model_detections = self.detect(frame)
        self.track(model_detections)

        self.total_time_current = self....
detection_time_current + self.tracking_time_current +...
 self.image_enhancement_current
        self.mean_total_time += self.total_time_current

        if (self.min_total_time == -1 or self....
total_time_current < self.min_total_time) and self....
total_time_current > 0:
            self.min_total_time = self....
total_time_current
        if self.total_time_current > self.max_total_time...
:
            self.max_total_time = self....
total_time_current

        self.fps_current = 1.0 / (self....
total_time_current)
        self.fps_current = round(self.fps_current, 3)

        self.mean_fps += self.fps_current
        if (self.min_fps == -1 or self.fps_current < ...
self.min_fps) and self.fps_current > 0:
            self.min_fps = self.fps_current
```

```
        if self.fps_current > self.max_fps:
            self.max_fps = self.fps_current

    def read(self, resize=1, new_resolution=False):
        if self.type == "Video":
            return True, self.vid.read(), False, None
        else:
            frame = None
            ret = False
            new_video = False
            mask = None
            try:
                entry = self.entries[self....
next_entry_index]
                path = entry["images_path"]
                mask_path = entry["mask_path"]
                image_path = os.path.join(path, '{}'....
format(entry["filename"]))
                frame = cv2.imread(image_path)
                self.height, self.width, _ = frame.shape
                mask = cv2.imread(mask_path, 0)
                ret = True

                if resize ≤ 1:
                    self.scale = resize
                    self.width = int(self.width * self....
scale)
                    self.height = int(self.height * self...
.scale)
                    frame = cv2.resize(frame, (self....
width, self.height), interpolation = cv2.INTER_AREA)
                    mask = cv2.resize(mask, (self.width,...
 self.height), interpolation = cv2.INTER_AREA)

                if new_resolution:
                    resolutions = {
                        '720': {'width': 1280, 'height':...
 720},
                        '648': {'width': 1152, 'height':...
 648},
                        '576': {'width': 1024, 'height':...
 576},
                        '360': {'width': 640, 'height': ...
360}
                    }
                    new_resolution = resolutions[...
new_resolution]
                    self.scale = resize
                    self.width = int(new_resolution['...
```

```
width'])
                self.height = int(new_resolution['...
height'])
                frame = cv2.resize(frame, (self....
width, self.height), interpolation = cv2.INTER_AREA)
                mask = cv2.resize(mask, (self.width,...
 self.height), interpolation = cv2.INTER_AREA)

            if self.current_video != entry['...
images_path'] and self.current_video != "":
                new_video = True
            self.current_video = entry['images_path'...
]
        except IndexError as e:
            print(e)
        self.next_entry_index += 1

        return ret, frame, new_video, mask

def get_tracks(self):
    return self.tracking_model.get_tracks()

def queue_performance(self, queue_stats, queue_time)...
:
    cars = []
    self.queues_detected = len(queue_stats)

    length = 0
    max_queue_length = 0
    min_queue_length = -1
    for queue in queue_stats:
        queue = queue_stats[queue]
        length += len(queue)

        if (self.min_queue_length == -1 or self....
min_queue_length > len(queue)):
            self.min_queue_length = len(queue)

        if self.max_queue_length < len(queue):
            self.max_queue_length = len(queue)

        if (min_queue_length == -1 or ...
min_queue_length > len(queue)):
            min_queue_length = len(queue)

        if max_queue_length < len(queue):
            max_queue_length = len(queue)

        for car in queue:
```

```
                if car not in cars:
                    cars.append(car)

        self.avg_queue_length = length / len(queue_stats...
)

        for car in cars:
            car_tracker = 0
            for queue in queue_stats:
                queue = queue_stats[queue]
                if car in queue:
                    car_tracker += 1

            if car_tracker > 1:
                self.queue_changes += car_tracker - 1
        self.mean_queue_changes = self.queue_changes / ...
len(cars)

        self.min_queue_detection_time = queue_time if ...
queue_time < self.min_queue_detection_time or self....
min_queue_detection_time == -1 else self....
min_queue_detection_time
        self.max_queue_detection_time = queue_time if ...
queue_time > self.max_queue_detection_time else self....
max_queue_detection_time
        self.queue_detection_time.append(queue_time)
        self.mean_queue_detection_time = sum(self....
queue_detection_time) / len(self.queue_detection_time...
)

        self.frame_queues = {
            'avg_queue_length': length / len(queue_stats...
),
            'max_queue_length': max_queue_length,
            'min_queue_length': min_queue_length,
            'mean_queue_changes': self.queue_changes / ...
len(cars)
        }

    def status(self, frameData):
        detection_time = int((self....
detection_time_current) * 1000)
        track_time = int(self.tracking_time_current * ...
1000)
        print(f"\nFrame: {self.number_of_frames}")
        print(f"FPS: {self.fps_current}")
        print(f"IE time: {int(self....
image_enhancement_current* 1000)} ms")
        print(f"Detection time: {detection_time} ms")
```

```
        print(f"Tracking time: {track_time} ms")
        print(f"Total time: {int(self.total_time_current...
    * 1000)} ms")

        class_to_id = {'car': 1, 'person': 2, 'truck': ...
    3, 'bus': 4, 'bike': 5, 'motorbike': 6, 'Road anomaly...
    ': 10}
        avg_score = 0
        avg_score_adjusted = 0
        number_of_detections_adjusted = 0
        number_of_correct_classes = 0
        number_of_wrong_classes = 0
        number_of_correct_ids = 0
        number_of_wrong_ids = 0
        number_of_duplicate_ids = 0
        object_ids = []
        print("Detected objects:")
        for detected_object in self.detected_objects:
            print("Detected object: ", detected_object)
            print("Class: ", detected_object['object']['...
    class'])
            print(f"\t- {detected_object['object']['...
    class']}, {round(detected_object['score']*100, 2)} %"...
    )
            avg_score += detected_object['score']
            if detected_object["real_object"]['info']['...
    occluded'] == "False":
                avg_score_adjusted += detected_object['...
    score']
                number_of_detections_adjusted += 1
            if detected_object["object"]["class"] == ...
    detected_object["real_object"]["class"]:
                print("\t\t- Correct Class")
                number_of_correct_classes += 1
            else:
                print("\t\t- Wrong Class")
                number_of_wrong_classes += 1

            x1, y1, x2, y2 = detected_object["object"]["...
    bbox"]
            bx, by = (x1 + x2) // 2, (y1 + y2) // 2
            bx, by = int(bx), int(by)
            if self.mask[by, bx] == 255:
                self.cars_outside_mask += 1

            if 'ID' in detected_object['real_object']['...
    info']:
                if detected_object['real_object']['info'...
    ]['ID'] in self.detected_objects_previous:
```

```
                if self.detected_objects_previous[...
detected_object['real_object']['info']['ID']] == ...
detected_object['object']['ID']:
                    if detected_object['real_object'...
]['info']['ID'] in object_ids:
                        print("\t\t- Duplicate ID")
                        number_of_duplicate_ids += 1
                    else:
                        print("\t\t- Correct ID")
                        number_of_correct_ids += 1
                else:
                    print("\t\t- Wrong ID")
                    if detected_object["real_object"...
]["class"] != "person":
                        number_of_wrong_ids += 1
                    self.detected_objects_previous[...
detected_object['real_object']['info']['ID']] = ...
detected_object['object']['ID']
            else:
                self.detected_objects_previous[...
detected_object['real_object']['info']['ID']] = ...
detected_object['object']['ID']
            object_ids.append(detected_object['...
real_object']['info']['ID'])

        self.track_values['true_labels'].append(...
detected_object['real_object']['class_id'])
        self.track_values['predicted_labels'].append...
(detected_object['score'])

    self.tracking_accuracy += number_of_correct_ids
    self.tracking_id_switches += number_of_wrong_ids
    self.tracking_id_duplicates += ...
number_of_duplicate_ids

    self.detection_accuracy += avg_score
    self.detection_accuracy_adjusted += ...
avg_score_adjusted
    self.total_number_of_valid_detections += len(...
self.detected_objects)
    self.total_number_of_valid_detections_adjusted ...
+= number_of_detections_adjusted
    if len(self.detected_objects): avg_score /= len(...
self.detected_objects)
    if number_of_detections_adjusted > 0: ...
avg_score_adjusted /= number_of_detections_adjusted
    print(f"Average score: {round(avg_score*100, 2)}...
 %")
    print(f"Average score adjusted: {round(...
```

```
avg_score_adjusted*100, 2)} %")

        tmp_missed = 0
        if object_ids != []:
            for real_object in self.entries[self....
next_entry_index-1]["objects"]:
                if real_object['info']['ID'] not in ...
object_ids:
                    self.missed_detections += 1
                    tmp_missed += 1
            self.total_number_of_real_detections += len(...
self.entries[self.next_entry_index-1]["objects"])

        print(f"Missed detections: {tmp_missed}")
        try:
            print(f"Missed detections: {round(100*...
tmp_missed/len(self.entries[self.next_entry_index...
-1]['objects']), 1)} %")
        except Exception as e:
            print(e)
        print(f"False positive detections: {self....
false_positives_detections - self....
false_positives_detections_previous}")

        systemAnalytics = frameData['computational_data'...
]
        frame_data = {
            'frame_number': frameData['frame_number'],
            'current_time': frameData['current_time']
        }
        if self.incident_accuracy+self.missed_incidents ...
> 0:
            incident_accuracy_accumulated = round(100*...
self.incident_accuracy/(self.incident_accuracy+self....
missed_incidents), 1)
        else:
            incident_accuracy_accumulated = 0
        frameInfo = {
            'frame_data': frame_data,
            'computational_data': systemAnalytics,
            'mean_detection_time': int(1000 * self....
mean_detection_time / self.number_of_frames),
            'min_detection_time': int(1000 * self....
min_detection_time),
            'max_detection_time': int(1000 * self....
max_detection_time),
            'mean_tracking_time': int(1000 * self....
mean_tracking_time / self.number_of_frames),
            'min_tracking_time': int(1000 * self....
```

```
min_tracking_time),
            'max_tracking_time': int(1000 * self....
min_tracking_time),
            'avg_score': avg_score,
            'number_of_valid_detections': len(self....
detected_objects),
            'number_of_detections_adjusted': ...
number_of_detections_adjusted,
            'number_of_correct_classes': ...
number_of_correct_classes,
            'number_of_wrong_classes': ...
number_of_wrong_classes,
            'number_of_correct_ids': ...
number_of_correct_ids,
            'number_of_wrong_ids': number_of_wrong_ids,
            'number_of_duplicate_ids': ...
number_of_duplicate_ids,
            'false_positive_detections': self....
false_positives_detections - self....
false_positives_detections_previous,
            'missed_detections': round(100*tmp_missed/...
len(self.entries[self.next_entry_index-1]['objects'])...
, 1),
            'incident_accuracy_accumulated': ...
incident_accuracy_accumulated,
            'queue_info': self.frame_queues
        }

        self.frame_queues = {}

        self.frameData.append(frameInfo)

        self.detected_objects = []

        self.false_positives_detections_previous = self....
false_positives_detections

    def summary(self):
        text = "\n"
        jsonFormat = {}
        try:
            total_detections = self....
total_number_of_valid_detections + self....
false_positives_detections
            text += f"Scale: {int(self.scale*100)} %\n"
            text += f"Resolution: {int(self.width*self....
scale)}x{int(self.height*self.scale)} px\n"
            text += f"Mean image enhancement time: {int...
(1000 * self.mean_image_enhancement_time / self....
```

```
number_of_frames)} ms\n"
            text += "\n"
            text += f"Mean detection time: \t{int(1000 *...
 self.mean_detection_time / self.number_of_frames)} ...
ms\n"
            text += f"Min detection time: \t{int(1000 * ...
self.min_detection_time)} ms\n"
            text += f"Max detection time: \t{int(1000 * ...
self.max_detection_time)} ms\n"
            text += "\n"
            text += f"Mean tracking time: \t{int(1000 * ...
self.mean_tracking_time / self.number_of_frames)} ms\...
n"
            text += f"Min tracking time: \t\t{int(1000 *...
 self.min_tracking_time)} ms\n"
            text += f"Max tracking time: \t\t{int(1000 *...
 self.max_tracking_time)} ms\n"
            text += "\n"
            text += f"Mean total time: \t{int(1000 * ...
self.mean_total_time / self.number_of_frames)} ms\n"
            text += f"Min total time: \t{int(1000 * self...
.min_total_time)} ms\n"
            text += f"Max total time: \t{int(1000 * self...
.max_total_time)} ms\n"
            text += "\n"
            text += f"Mean fps: \t{round(self.mean_fps /...
 self.number_of_frames, 1)}\n"
            text += f"Min fps: \t{int(self.min_fps)}\n"
            text += f"Max fps: \t{int(self.max_fps)}\n"
            text += "\n"
            text += f"False positive detections: \t{...
round(100*self.false_positives_detections/...
total_detections, 1)} %\n"
            text += f"Missed detections: \t\t\t{round...
(100*self.missed_detections/self....
total_number_of_real_detections, 1)} %\n"
            text += "\n"
            text += f"Detection accuracy: \t\t\t{round...
(100*self.detection_accuracy/self....
total_number_of_valid_detections, 1)} %\n"
            text += f"Detection accuracy adjusted: \t{...
round(100*self.detection_accuracy_adjusted/self....
total_number_of_valid_detections_adjusted, 1)} %\n"
            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"
            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"
            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"
            text += "\n"
            text += f"Incident accuracy: \t{round(100*...
    self.incident_accuracy/(self.incident_accuracy+self....
    missed_incidents), 1)} %\n"
            text += f"Missed incidents: \t{round(100*...
    self.missed_incidents/(self.incident_accuracy+self....
    missed_incidents), 1)} %\n"
            text += f"False alarms: \t\t{round(100*self....
    false_alarms/total_detections, 1)} %\n"
            text += "\n"
            text += f"Total number of valid detections: ...
    {self.total_number_of_valid_detections}\n"
            text += f"Total number of detections: {...
    total_detections}\n"

            # Own statistics
            text += "\n"
            text += f"Cars detected outside of the mask:...
     {self.cars_outside_mask}\n"
            # text += f"Foreign objects detected inside ...
    the mask: \n"

            # Queue statistics:
            text += "\n"
            text += f"Queues detected: {self....
    queues_detected}\n"
            text += f"Amount queue changes: {self....
    queue_changes}\n"
            text += f"Average queue changes per car: {...
    self.mean_queue_changes}\n"
            text += f"Max length of a queue: {self....
    max_queue_length}\n"
            text += f"Min length of a queue: {self....
    min_queue_length}\n"
            text += f"Avg length of a queue: {self....
    avg_queue_length}\n"

            text += f"Min queue detection time: {self....
    min_queue_detection_time}\n"
            text += f"Max queue detection time: {self....
    max_queue_detection_time}\n"
            text += f"Mean queue detection time: {self....
    mean_queue_detection_time}\n"
```

```

            # Json format:
            resolution = f"{int(self.width*self.scale)}x...
    {int(self.height*self.scale)}"
            jsonFormat = {
                'scale': int(self.scale*100),
                'resolution': resolution,
                'mean_image_enhancement_time': int(1000 ...
    * self.mean_image_enhancement_time / self....
    number_of_frames),
                'mean_detection_time': int(1000 * self....
    mean_detection_time / self.number_of_frames),
                'min_detection_time': int(1000 * self....
    min_detection_time),
                'max_detection_time': int(1000 * self....
    max_detection_time),
                'mean_tracking_time': int(1000 * self....
    mean_tracking_time / self.number_of_frames),
                'min_tracking_time': int(1000 * self....
    min_tracking_time),
                'max_tracking_time': int(1000 * self....
    max_tracking_time),
                'mean_total_time': int(1000 * self....
    mean_total_time / self.number_of_frames),
                'min_total_time': int(1000 * self....
    min_total_time),
                'max_total_time': int(1000 * self....
    max_total_time),
                'mean_fps': round(self.mean_fps / self....
    number_of_frames, 1),
                'min_fps': int(self.min_fps),
                'max_fps': int(self.max_fps),
                'false_positive_detections': round(100*...
    self.false_positives_detections/total_detections, 1),
                'missed_detections': round(100*self....
    missed_detections/self....
    total_number_of_real_detections, 1),
                'detection_accuracy': round(100*self....
    detection_accuracy/self....
    total_number_of_valid_detections, 1),
                'detection_accuracy_adjusted': round...
    (100*self.detection_accuracy_adjusted/self....
    total_number_of_valid_detections_adjusted, 1),
                'tracking_accuracy': round(100*self....
    tracking_accuracy/(self.tracking_accuracy+self....
    tracking_id_switches+self.tracking_id_duplicates), 1)...
    ,
                'tracking_id_duplicates': round(100*self...
    .tracking_id_duplicates/(self.tracking_accuracy+self....
```

```
tracking_id_switches+self.tracking_id_duplicates), 1)...
,
                'tracking_id_switches': round(100*self....
tracking_id_switches/(self.tracking_accuracy+self....
tracking_id_switches+self.tracking_id_duplicates), 1)...
,
                'incident_accuracy': round(100*self....
incident_accuracy/(self.incident_accuracy+self....
missed_incidents), 1),
                'missed_incidents': round(100*self....
missed_incidents/(self.incident_accuracy+self....
missed_incidents), 1),
                'false_alarms': round(100*self....
false_alarms/total_detections, 1),
                'total_number_of_valid_detections': self...
.total_number_of_valid_detections,
                'total_number_of_detections': ...
total_detections,
                'cars_detected_outside_mask': self....
cars_outside_mask,
                'queues_detected': self.queues_detected,
                'min_queue_length': self....
min_queue_length,
                'max_queue_length': self....
max_queue_length,
                'avg_queue_length': self....
avg_queue_length,
                'queue_changes': self.queue_changes,
                'mean_queue_changes': self....
mean_queue_changes,
                'min_queue_detection_time': self....
min_queue_detection_time,
                'max_queue_detection_time': self....
max_queue_detection_time,
                'mean_queue_detection_time': self....
mean_queue_detection_time,
                'frame_data': self.frameData,
                'detection_data': self.track_values
            }



        except Exception as e:
            print("Exception happened in performance ...
evaluator")
            print(e)

        return text, jsonFormat
```

Kode B.4: performance_evaluator.py was altered for this thesis and its original source code was a part of the thesis written by Aleksander Vedvik for which the source code can be found at [48]

```
import cv2
import numpy as np

class Noise_Manager:
    def __init__(self, noise_type):
        self.noise_type = noise_type

    def add_noise(self, frame, mean=0, sigma=0.5):

        noise = None

        if self.noise_type == "gauss":
            noise =  np.random.normal(mean, sigma, frame...
    .shape).astype('uint8')
            frame = cv2.add(frame, noise)
            return frame

        if self.noise_type == "salt":
            result = np.copy(frame)

            salt = np.ceil(0.01 * frame.size)
            coords = [np.random.randint(0, i - 1, int(...
    salt)) for i in frame.shape]
            result[tuple(coords)] = 255

            pepper = np.ceil(0.01 * frame.size)
            coords = [np.random.randint(0, i - 1, int(...
    pepper)) for i in frame.shape]
            result[tuple(coords)] = 0

            return result

        if self.noise_type == "speckle":
            gauss = np.random.normal(mean, sigma, frame....
    shape).astype("float32")
            frame = cv2.add(frame.astype("float32"), ...
    frame.astype("float32") * gauss)
            return frame.astype("uint8")

        return frame
```

Kode B.5: noise_manager.py was a class specifically developed for this thesis to introduce noise and be able to evaluate data with artificially reduced quality.

```
import cv2
import matplotlib

matplotlib.use('TkAgg')


def draw_circle(image, object, img_ratio, color=0):
    center_coordinates = (int((float(object["x1"]) + (...
    float(object["x2"]) - float(object["x1"])) / 2)*...
    img_ratio), int((float(object["y1"]) + (float(object[...
    "y2"]) - float(object["y1"])) / 2)*img_ratio))
    radius = 0
    if color == 0:
        color_circle = (0, 0, 255)
    else:
        color_circle = color
    thickness_circle = 10

    cv2.circle(image, center_coordinates, radius, ...
    color_circle, thickness_circle)


def draw_line(image, start_point, end_point):
    color, thickness = (255,255,255), 2
    cv2.arrowedLine(image, start_point, end_point, color...
    , thickness)


def draw_text(image, track, text):
    object = track.to_tlbr()

    coordinates = (int(object[0]), int(object[1]-10))
    font = cv2.FONT_HERSHEY_SIMPLEX
    fontScale = 1
    color_text = (255, 255, 255)
    thickness = 2

    cv2.putText(image, text, coordinates, font, ...
    fontScale, color_text, thickness)


def draw_rectangle(image, track, color):
    object = track.to_tlbr()
```

```
    0
    start_point, end_point = (int(object[0]), int(object...
    [1])), (int(object[2]), int(object[3]))
    color_rectangle = color
    thickness_rectangle = 2

    cv2.rectangle(image, start_point, end_point, ...
    color_rectangle, thickness_rectangle)

def draw_parallelogram(image, top_left, top_right, ...
    bottom_left, bottom_right):
    thickness = 2
    color = (57, 255, 20)
    # print(top_left)
    top_left = (int(top_left[0]), int(top_left[1]))
    top_right = (int(top_right[0]), int(top_right[1]))
    bottom_left = (int(bottom_left[0]), int(bottom_left...
    [1]))
    bottom_right = (int(bottom_right[0]), int(...
    bottom_right[1]))
    cv2.line(image, top_left, top_right, color, ...
    thickness)
    cv2.line(image, top_right, bottom_right, color, ...
    thickness)
    cv2.line(image, bottom_right, bottom_left, color, ...
    thickness)
    cv2.line(image, bottom_left, top_left, color, ...
    thickness)
```

Kode B.6: visualize_objects.py was altered for this thesis and its original source code was a part of the thesis written by Aleksander Vedvik for which the source code can be found at [48]. The altered code was mainly focuesed around the paralellogram visualization

```
import os
import argparse
import json
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
from sklearn.metrics import roc_curve, auc, ...
    precision_recall_curve
from sklearn.preprocessing import label_binarize

parser = argparse.ArgumentParser(
    description="Analyzing output statistics from the ...
```

```
    main software"
)
parser.add_argument("-s",
                    "--source",
                    help="Select the source directory, ...
    This is expecting a directory created from the ...
    session config file with its setup, expects only name...
     of the directory",
                    type=str)

args = parser.parse_args()

def confMatrix(confMatrix, outputDir):
    print("\nConfusion matrix")
    print(confMatrix)
    for matrix in confMatrix:
        # print(matrix)
        matrixOutputDir = os.path.join(outputDir, matrix...
    ['statInfo']['detection'], matrix['statInfo']['...
    tracking'], matrix['statInfo']['image_enhancement'])
        if not os.path.exists(matrixOutputDir):
            os.makedirs(matrixOutputDir)
        confMtx = np.array([
            [matrix['tp'], matrix['fn']],
            [matrix['fp'], matrix['tn']]
        ])
        confMtx = confMtx.astype(int)

        plt.figure(figsize=(8, 6))
        sns.heatmap(confMtx, annot=True, fmt="d", cmap="...
    Blues", cbar=False)
        title = f"Detection: {matrix['statInfo']['...
    detection']}, Tracking: {matrix['statInfo']['tracking...
    ']}, Img_enh: {matrix['statInfo']['image_enhancement...
    ']}, Noise_type: {matrix['statInfo']['noise_type']}"
        plt.title(f"Confusion Matrix {title}")
        plt.ylabel('True Label')
        plt.xlabel('Predicted Label')
        plt.xticks([0.5, 1.5], ["Positive", "Negative"])
        plt.yticks([0.5, 1.5], ["Positive", "Negative"],...
     rotation=0)
        print(matrix['statInfo']['file'])
        dataFile = matrix['statInfo']['file'].split('.')...
    [0] if matrix['statInfo']['file'].endswith('.json') ...
    else matrix['statInfo']['file']
        filename = f"{matrixOutputDir}/Confusion_matrix_...
    {dataFile}_{matrix['statInfo']['noise_type']}.png"
        plt.savefig(filename)
        plt.close()
```

```

def datasetConfMatrix(matrix, outputDir):
    filename = os.path.join(outputDir, "Confusion_matrix...
    .png")
    confMtx = np.array([
        [matrix['tp'], matrix['fn']],
        [matrix['fp'], matrix['tn']]
    ])
    confMtx = confMtx.astype(int)

    plt.figure(figsize=(8, 6))
    sns.heatmap(confMtx, annot=True, fmt="d", cmap="...
    Blues", cbar=False)
    title = f"Detection: {matrix['statInfo']['detection...
    ']}, Tracking: {matrix['statInfo']['tracking']}, ...
    Img_enh: {matrix['statInfo']['image_enhancement']}, ...
    Noise_type: {matrix['statInfo']['noise_type']}"
    plt.title(f"Confusion Matrix {title}")
    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')
    plt.xticks([0.5, 1.5], ["Positive", "Negative"])
    plt.yticks([0.5, 1.5], ["Positive", "Negative"], ...
    rotation=0)
    print(matrix['statInfo']['file'])
    dataFile = matrix['statInfo']['file'].split('.')[0] ...
    if matrix['statInfo']['file'].endswith('.json') else ...
    matrix['statInfo']['file']
    plt.savefig(filename)
    plt.close()

def brightness_graphing(graphs, outputDir):

    for graph in graphs:
        if None in graph['brightness_level']:
            continue
        graphOutputDir = os.path.join(outputDir, graph['...
    statInfo']['detection'], graph['statInfo']['tracking'...
    ], graph['statInfo']['image_enhancement'])

        if not os.path.exists(graphOutputDir):
            os.makedirs(graphOutputDir)

        for value in graph['y_value']:
            x_value = np.array(graph['brightness_level'...
    ])
            y_value = np.array(graph['y_value'][value])

            indices = np.argsort(x_value)
            sorted_x = x_value[indices]
```

```
            sorted_y = y_value[indices]

            # print(f"\n Filename: {graph['file']}, ...
    folder: {graphOutputDir}")
            # print(f"Brightness: Value: {sorted_x}, {...
    value}: {sorted_y} \n")
            plt.figure(figsize=(8,4))
            plt.plot(sorted_x, sorted_y)
            title = f"Detection: {graph['statInfo']['...
    detection']}, Tracking: {graph['statInfo']['tracking...
    ']}, Img_enh: {graph['statInfo']['image_enhancement...
    ']}, Noise_type: {graph['statInfo']['noise_type']}"
            plt.title(title)
            plt.xlabel('Brightness Value')
            plt.ylabel(value)
            filename = f"{graphOutputDir}/{graph['...
    statInfo']['noise_type']}_{value}.png"
            plt.savefig(filename)
            plt.close()

def over_time_performance(df, outputDir):
    filename = os.path.join(outputDir, f'...
    over_time_performance.png')

    plt.figure(figsize=(12, 6))
    plt.plot(df['frame_number'], df['mean_detection_time...
    '], label='Mean Detection Time')
    plt.plot(df['frame_number'], df['min_detection_time'...
    ], label='Min Detection Time')
    plt.plot(df['frame_number'], df['max_detection_time'...
    ], label='Max Detection Time')
    plt.xlabel('Frame number')
    plt.ylabel('Time (ms)')
    plt.title('Detection Time Over Time')
    plt.legend()
    plt.savefig(filename)
    plt.close()

#
def detection_accuracy_bar(df, outputDir):
    filename = os.path.join(outputDir, f'...
    detection_accuracy_bar.png')
    print(df['detection_accuracy'])
    print(df['detection_accuracy_adjusted'])
    print(df['tracking_accuracy'])
    print(df['incident_accuracy'])
    accuracy_data = df[['detection_accuracy', '...
    detection_accuracy_adjusted', 'tracking_accuracy', '...
    incident_accuracy']]
```

```
    long_format = accuracy_data.melt(value_vars=['...
    detection_accuracy', 'detection_accuracy_adjusted', '...
    tracking_accuracy', 'incident_accuracy'], var_name='...
    Metric', value_name='Percentage')


    sns.barplot(x="Metric", y="Percentage", data=...
    long_format)
    plt.xlabel('Metric')
    plt.ylabel('Percentage')
    plt.title('Detection and Tracking Accuracy Metrics')
    plt.savefig(filename)
    plt.close()

#
def tracking_analysis_bar(df, outputDir):
    filename = os.path.join(outputDir, f'...
    tracking_analysis_bar.png')

    tracking_data = df[['tracking_id_switches', '...
    tracking_id_duplicates']]
    long_format = tracking_data.melt(value_vars=['...
    tracking_id_switches', 'tracking_id_duplicates'], ...
    var_name='', value_name='Count')

    sns.barplot(x="", y="Count", data=long_format)
    plt.ylabel('Count')
    plt.savefig(filename)
    plt.close()

def incident_analysis_graph(df, outputDir):
    filename = os.path.join(outputDir, f'...
    incident_analysis_graph.png')

    plt.plot(df['frame_number'], df['...
    number_of_wrong_classes'], label='Number of wrong ...
    classes')
    plt.plot(df['frame_number'], df['...
    false_positive_detections'], label='False positive ...
    detections')
    plt.xlabel('Frame number')
    plt.ylabel('Count')
    plt.title('Incident Reporting Over Time')
    plt.legend()
    plt.savefig(filename)
    plt.close()

# def queue_analysis(df, outputDir):
#     plt.figure(figsize=(12, 6))
```

```
#     plt.plot(df['frame_number'], df['avg_queue_length...
    '], label='Average Queue Length')
#     plt.xlabel('Frame number')
#     plt.ylabel('Queue Length')
#     plt.title('Queue Length Over Time')
#     plt.legend()
#     plt.savefig(filename)
#     plt.close()

#
def detection_time_analysis(df, outputDir):
    filename = os.path.join(outputDir, f'...
    detection_time_analysis.png')

    plt.figure(figsize=(10, 5))
    sns.boxplot(data=df[['mean_detection_time', '...
    mean_tracking_time', 'mean_total_time']])
    plt.ylabel('Time (ms)')
    plt.title('Distribution of Detection, Tracking, and ...
    Total Times')
    filename = os.path.join(outputDir, f'')
    plt.savefig(filename)
    plt.close()

def detection_heatmap(df, outputDir):
    filename = os.path.join(outputDir, f'...
    detection_heatmap.png')

    plt.figure(figsize=(10, 6))
    sns.kdeplot(x=df['x_coords'], y=df['y_coords'], cmap...
    ="Reds", shade=True, bw_adjust=.5)
    plt.xlabel('X Coordinate')
    plt.ylabel('Y Coordinate')
    plt.title('Heatmap of Detection Locations')
    plt.savefig(filename)
    plt.close()

#
# def roc_recall_curves(df, outputDir):
#     rocFilename = outputDir
#     recallFilename = os.path.join(outputDir, f'recall....
    png')

#     true_labels = df['true_labels']
#     pred_score = df['pred_scores']
#     print(f'True labels length: {len(true_labels)}')
#     print(f'pred labels length: {len(pred_score)}')

#     classes = np.unique(true_labels)
```

```
#     true_labels_bin = label_binarize(true_labels, ...
    classes=classes)

#     for i in range(len(classes)):
#         class_to_id = {0: 'None',1: 'car', 2: 'person...
    ', 3: 'truck', 4: 'bus', 5: 'bike', 6: 'motorbike', ...
    10: 'Road anomaly'}
#         rocClassFilename = os.path.join(rocFilename, f...
    "_{class_to_id[i]}_roc.png")
#         print(pred_score)
#         fpr, tpr, _ = roc_curve(true_labels_bin[:, i],...
     pred_score[:, i])
#         roc_auc = auc(fpr, tpr)

#         plt.figure(figsize=(10, 5))
#         plt.plot(fpr, tpr, color='darkorange', lw=2, ...
    label='ROC curve (area = %0.2f)' % roc_auc)
#         plt.plot([0, 1], [0, 1], color='navy', lw=2, ...
    linestyle='--')
#         plt.xlabel('False Positive Rate')
#         plt.ylabel('True Positive Rate')
#         plt.title('Receiver Operating Characteristic')
#         plt.legend(loc="lower right")
#         plt.savefig(rocClassFilename)
#         plt.close()

#     precision, recall, _ = precision_recall_curve(...
    true_labels, pred_score)
#     pr_auc = auc(recall, precision)


#     plt.figure(figsize=(10, 5))
#     plt.plot(recall, precision, color='blue', lw=2, ...
    label='PR curve (area = %0.2f)' % pr_auc)
#     plt.xlabel('Recall')
#     plt.ylabel('Precision')
#     plt.title('Precision-Recall curve')
#     plt.legend(loc="lower left")
#     plt.savefig(recallFilename)
#     plt.close()



def system_load_analysis(df, outputDir):
    filename = os.path.join(outputDir, f'...
    system_load_analysis.png')

    plt.figure(figsize=(14, 7))

```

```
    plt.subplot(2, 2, 1)
    plt.plot(df['frame_number'], df['gpu_load_percent'],...
     label='GPU Load')
    plt.xlabel('Time')
    plt.ylabel('GPU Load (%)')
    plt.title('GPU Load Over Time')

    plt.subplot(2, 2, 2)
    plt.plot(df['frame_number'], df['gpu_memory_usage'],...
     label='GPU Memory Usage')
    plt.xlabel('Time')
    plt.ylabel('Memory Usage (MB)')
    plt.title('GPU Memory Usage Over Time')

    plt.subplot(2, 2, 3)
    plt.plot(df['frame_number'], df['cpu_usage'], label=...
    'CPU Usage')
    plt.xlabel('Time')
    plt.ylabel('CPU Usage (%)')
    plt.title('CPU Usage Over Time')

    plt.tight_layout()
    plt.savefig(filename)
    plt.close()



def main():
    if not args.source:
        print("No source file specified")
        return

    sourceBaseDir = r'.\\data\\output'
    baseOutputDir = r'.\\data\\graphics_output'
    sourceDir = os.path.join(sourceBaseDir, args.source)
    outputDir = os.path.join(baseOutputDir, args.source)

    i = 0
    try:
        while os.path.exists(outputDir):
            alt_path = f"{args.source}_{i}"
            outputDir = os.path.join(baseOutputDir, ...
    alt_path)
            i += 1

            if i == 10000:
                raise TimeoutError
    except TimeoutError as e:
        print(f"The directory part has timed out, i = {i...
```

```
}")
        return


    statisticList = []

    # Collecting data
    for entry in os.listdir(sourceDir):
        runDir = os.path.join(sourceDir, entry)

        for jsonFiles in os.listdir(runDir):
            # if jsonFiles.startswith("Video2"):
            #     continue
            if jsonFiles.split(".")[1] != "json":
                continue

            sourceFile = os.path.join(runDir, jsonFiles)

            try:
                with open(sourceFile, 'r') as file:
                    data = json.load(file)
                    statisticList.append({'file': file,'...
data': data, 'filename': jsonFiles})
            except FileNotFoundError:
                print(f"The file was not found. {...
sourceFile}")
            except json.JSONDecodeError:
                print("Error decoding JSON.")
            except Exception as e:
                print(f"An error occurred: {e}")


    # Processing data
    graphs = []
    graphDict = {'statInfo': {}}
    confusionMatrix = []
    dataframe = {}
    dataframes = []

    detection_accuracy = []
    detection_accuracy_adjusted = []
    tracking_accuracy = []
    incident_accuracy = []
    tracking_id_switches = []
    tracking_id_duplicates = []
    mean_detection_time = []
    mean_tracking_time = []
    mean_total_time = []
    true_labels = []
```

```
    pred_labels = []
    confMatrixValues = {
        'statInfo': None,
        'tn': 0,
        'fp': 0,
        'fn': 0,
        'tp': 0
    }
    for stat in statisticList:
        file = stat['file']
        filename = stat['filename']
        stat = stat['data']
        keys = ['brightness_level', 'mean_detection_time...
', 'mean_tracking_time', 'detection_accuracy', '...
cars_detected_outside_mask', 'resolution']
        if not all(key in stat for key in keys):
            continue

        #Statistics cleanup
        # print(stat)
        statInfo = {'file': filename,'detection': stat['...
detection'], 'tracking': stat['tracking'], '...
image_enhancement': stat['image_enhancement'], '...
noise_type': stat['noise_type'], 'resolution': stat['...
resolution']}
        statInGraph = False

        if stat['brightness_level'] == 0 or stat['...
brightness_level'] == None:
            tn = stat['total_number_of_detections'] - ...
stat['total_number_of_valid_detections']
            confusionMatrix.append({'statInfo': statInfo...
, 'tn': tn, 'fp': stat['false_positive_detections'], ...
'fn': stat['missed_detections'], 'tp': stat['...
total_number_of_valid_detections']})
            confMatrixValues['statInfo'] = statInfo
            confMatrixValues['tn'] += tn
            confMatrixValues['fp'] += stat['...
false_positive_detections']
            confMatrixValues['fn'] += stat['...
missed_detections']
            confMatrixValues['tp'] += stat['...
total_number_of_valid_detections']

        try:
            for graph in graphs:

                if graph.get('statInfo', None) == ...
statInfo:
```

```
                    graph['file'].append(file)
                    graph['brightness_level'].append(...
stat['brightness_level'])
                    graph['y_value']['...
mean_detection_time'].append(stat['...
mean_detection_time'])
                    graph['y_value']['mean_tracking_time...
'].append(stat['mean_tracking_time'])
                    graph['y_value']['detection_accuracy...
'].append(stat['detection_accuracy'])
                    graph['y_value']['...
cars_detected_outside_mask'].append(stat['...
cars_detected_outside_mask'])
                    statInGraph = True

            if not statInGraph:
                tempDict = {
                    'statInfo': statInfo,
                    'file': [file],
                    'brightness_level': [stat['...
brightness_level']],
                    'y_value': {
                        'mean_detection_time': [stat['...
mean_detection_time']],
                        'mean_tracking_time': [stat['...
mean_tracking_time']],
                        'detection_accuracy': [stat['...
detection_accuracy']],
                        'cars_detected_outside_mask': [...
stat['cars_detected_outside_mask']],
                    }
                }
                graphs.append(tempDict)
        except KeyError as err:
            print(f"Key error: {err}")
            continue


        detection_accuracy.append(stat['...
detection_accuracy'])
        detection_accuracy_adjusted.append(stat['...
detection_accuracy_adjusted'])
        tracking_accuracy.append(stat['tracking_accuracy...
'])
        incident_accuracy.append(stat['incident_accuracy...
'])
        tracking_id_switches.append(stat['...
tracking_id_switches'])
        tracking_id_duplicates.append(stat['...
```

```
    tracking_id_duplicates'])
        mean_detection_time.append(stat['...
    mean_detection_time'])
        mean_tracking_time.append(stat['...
    mean_tracking_time'])
        mean_total_time.append(stat['mean_total_time'])
        video_data = {'filename': filename, 'img_enh': ...
    statInfo['image_enhancement'], 'frame_data': {}, '...
    centerpoints': {'x': [], 'y': []}}
        for frame in stat['frame_data']:

            if video_data['frame_data'] == {}:
                video_data['frame_data'] = {
                    'frame_number': [frame['frame_data'...
    ]['frame_number']],
                    'current_time': [frame['frame_data'...
    ]['current_time']],
                    'mean_detection_time': [frame['...
    mean_detection_time']],
                    'min_detection_time': [frame['...
    min_detection_time']],
                    'max_detection_time': [frame['...
    max_detection_time']],
                    'number_of_wrong_classes': [frame['...
    number_of_wrong_classes']],
                    'false_positive_detections': [frame[...
    'false_positive_detections']],
                    'gpu_load_percent': [frame['...
    computational_data']['gpu_load_percent']],
                    'gpu_memory_usage': [frame['...
    computational_data']['gpu_memory_usage']],
                    'cpu_usage': [frame['...
    computational_data']['cpu_usage']],
                }
            else:
                video_data['frame_data']['frame_number'...
    ].append(frame['frame_data']['frame_number'])
                video_data['frame_data']['current_time'...
    ].append(frame['frame_data']['current_time'])
                video_data['frame_data']['...
    mean_detection_time'].append(frame['...
    mean_detection_time'])
                video_data['frame_data']['...
    min_detection_time'].append(frame['min_detection_time...
    '])
                video_data['frame_data']['...
    max_detection_time'].append(frame['max_detection_time...
    '])
                video_data['frame_data']['...
```

```
number_of_wrong_classes'].append(frame['...
number_of_wrong_classes'])
                video_data['frame_data']['...
false_positive_detections'].append(frame['...
false_positive_detections'])
                video_data['frame_data']['...
gpu_load_percent'].append(frame['computational_data'...
]['gpu_load_percent'])
                video_data['frame_data']['...
gpu_memory_usage'].append(frame['computational_data'...
]['gpu_memory_usage'])
                video_data['frame_data']['cpu_usage']....
append(frame['computational_data']['cpu_usage'])

        video_data['centerpoints']['x'] = stat['...
detection_data']['centerpoint']['x']
        video_data['centerpoints']['y'] = stat['...
detection_data']['centerpoint']['y']

        if statInfo['image_enhancement'] in dataframe:
            dataframe[statInfo['image_enhancement']]['...
video_stats'].append(video_data)
        else:
            dataframe[statInfo['image_enhancement']] = {
                'video_stats': [video_data],
                'dataset_stats': {}
            }

        true_labels = true_labels + stat['detection_data...
']['true_labels']
        pred_labels = pred_labels + stat['detection_data...
']['predicted_labels']

        dataframe[statInfo['image_enhancement']]['...
dataset_stats'] = {
            'img_enh': statInfo['image_enhancement'],
            'detection_accuracy': sum(detection_accuracy...
) / len(detection_accuracy),
            'detection_accuracy_adjusted': sum(...
detection_accuracy_adjusted) / len(...
detection_accuracy_adjusted),
            'tracking_accuracy': sum(tracking_accuracy) ...
/ len(tracking_accuracy),
            'tracking_id_switches': sum(...
tracking_id_switches) / len(tracking_id_switches),
            'tracking_id_duplicates': sum(...
tracking_id_duplicates) / len(tracking_id_duplicates)...
,
            'incident_accuracy': sum(incident_accuracy) ...
```

```
    / len(incident_accuracy),
            'mean_detection_time': sum(...
    mean_detection_time) / len(mean_detection_time),
            'mean_tracking_time': sum(mean_tracking_time...
    ) / len(mean_tracking_time),
            'mean_total_time': sum(mean_total_time) / ...
    len(mean_total_time),
            'true_labels': true_labels,
            'pred_labels': pred_labels
        }



    videoOutputPath = os.path.join(outputDir, )

    # Visualizing data
    confMatrix(confusionMatrix, outputDir)

    baseBaseOutputDir = outputDir
    for key in dataframe:
        dafa = dataframe[key]
        baseOutputDir = os.path.join(baseBaseOutputDir, ...
    dafa['dataset_stats']['img_enh'])
        print("\n")
        print(dafa['dataset_stats']['img_enh'])
        if not os.path.exists(baseOutputDir):
            os.mkdir(baseOutputDir)
        for video in dafa['video_stats']:
            filename = video['filename']
            outputDir = os.path.join(baseOutputDir, ...
    filename.split('.')[0])
            if not os.path.exists(outputDir):
                os.mkdir(outputDir)
            df = pd.DataFrame(video['frame_data'])

            heatmap_vals = {
                'x_coords': video['centerpoints']['x'],
                'y_coords': video['centerpoints']['y']
            }

            # heatmap_vals = pd.DataFrame(heatmap_vals)

            over_time_performance(df, outputDir)
            incident_analysis_graph(df, outputDir)
            detection_heatmap(heatmap_vals, outputDir)
            system_load_analysis(df, outputDir)

        roc_values = {
            'true_labels': dafa['dataset_stats']['...
```

```
    true_labels'],
            'pred_scores': dafa['dataset_stats']['...
    pred_labels']
        }
        outerDf = pd.DataFrame(dafa['dataset_stats'])
        datasetConfMatrix(confMatrixValues, ...
    baseOutputDir)
        detection_accuracy_bar(outerDf, baseOutputDir)
        tracking_analysis_bar(outerDf, baseOutputDir)
        detection_time_analysis(outerDf, baseOutputDir)
        # roc_recall_curves(roc_values, baseOutputDir)



    # print("Hey :)")

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

Kode B.7: StatisticAnalyser.py was specifically built for this thesis to analyze exported data into graphs

```
import os
import json

class Tunnel_Manager:
    def __init__(self):
        self.objects = {}

    def get_tunnel_data(self, tunnel):
        tunnel_data_folder = r'.\\data\\tunnel_data'
        tunnel_data_file = os.path.join(...
    tunnel_data_folder, tunnel)
        tunnel_data_file += '.json'
        if not os.path.exists(tunnel_data_file):
            return False

        with open(tunnel_data_file, 'r') as f:
            tunnel_data = json.load(f)

        return tunnel_data
```

Kode B.8: tunnel_manager.py was specifically built for this thesis to keep track of values for each video or tunnel in the dataset