university of groningen / faculty of mathematics and natural sciences / artificial intelligence

"Who is driving around me?"
Situational Awareness in Traffic using Deep Integrated Feature Signatures and Kalman Filtering.

Master's Thesis

August 22, 2019

Author: Tim Oosterhuis (s2234831)

Primary supervisor: Prof. Dr. Lambert Schomaker

Department of Artificial Intelligence

University of Groningen, The Netherlands

Abstract

Being aware of other traffic is a prerequisite for self-driving cars to operate in the real world. In this paper, we show how the intrinsic feature maps of an object detection CNN can be used to uniquely identify vehicles from dash-cam feed. Additionally, the possibility of tracking vehicles using the 'YOLO' network extended with a Kalman filter is investigated and the resulting challenges are discussed. Feature maps of a pretrained 'YOLO' network are used to create 700 deep integrated feature signatures (DIFS) from 20 different images of 35 vehicles from a high resolution dataset and 340 signatures from 20 different images of 17 vehicles of a lower resolution tracking benchmark dataset. In the final layers of the YOLO network, all of these images fall into two classes, being classified as either a 'car' or 'truck'. 5-Fold nearest neighbor (1-NN) classification was used on DIFS created from feature maps in the early layers of the network to correctly identify instances of vehicles at a rate of 96.7% for the high resolution data and with a rate of 86.8% for the lower resolution data. We conclude that the activation patterns of a deep neural detection YOLO network trained to distinguish between different classes lend themselves to the creation of deep integrated feature signatures (DIFS), which can be successfully used to identify different instances belonging to the same class, especially when feature maps from early layers in the network are used.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

For many people it is hard to imagine their lives without the ability to drive a car. However, not all people have this privilege. For instance, due to age or disability. A self-driving car could improve the mobility and with that the quality of life for this group of people. Moreover, the vast majority of car accidents is not caused by mechanical failure or environmental factors, but by errors in human judgment. If fully developed, self-driving cars have the potential to react faster than humans could, which would greatly reduce the number of traffic accidents.
Driving and interacting with the chaos of daily traffic is a complex feat of intelligence, which integrates sensory perception with split-second decision making. From the perspective of artificial intelligence, driving is a benchmark 'human skill', like chess before it, and the first self-driving car in real life traffic will be an AI land mile.

1.1 AI in Driving

1.1.1 Early Developments

The thought of intelligent vehicles has sparked the imagination of researchers for almost half a century at this point, if not longer. The first successful application of artificial intelligence in autonomous driving was created in Japan in 1977 (Tsugawa, Yatabe, Hirose, & Matsumoto, 1979). A driverless car was equipped with a system guided by electro-magnetic signals from cables in the road. The vehicle was capable of autonomously tracking lane markers and reaching a speed of 30 km/h.
The first intelligent vehicle capable of driving on existing roads was developed in the next two decades. This was the VaMoR's-P project (Dickmanns et al., 1994). In this project the embedded electro-magnatic cables were abandoned, and sensor-based autonomous driving was pioneered. Camera images were used in combination with mathematical approximations like Kalman filtering (Kalman, 1960) to avoid obstacles and track lane markers.
Another pair of milestones within the field of autonomous driving were the successful completion of the DARPA grand (desert) challenge (Buehler, Iagnemma, & Singh, 2007) and urban challenge (Buehler, Iagnemma, & Singh, 2009) in

2005 and 2007 respectively. For the urban challenge, participating vehicles had to obey traffic rules while navigating a 60-mile race course in an urban setting.

1.1.2 State of the Art

After decades of research and development, the technology of autonomous driving has advanced to the point where we could see self-driving cars interacting in our daily lives in the very near future. Many of the semi-automatic features developed for the DARPA challenges, like collision detection and lane recognition, have since been incorporated in the newer production lines of major commercial car manufacturers.

The international society of automotive engineers has defined five levels of vehicle autonomy (international, 2016), which are shown in Figure 1.1. The first two levels of autonomy require vehicles to be outfitted with sensors integrated with low-level software architectures. With the current technology, the level of partial autonomation has been realized. Vehicles have the capacity of high level mechanical integration, like combined automated acceleration and steering. However, the human driver is required to be engaged at all times and pay constant attention to the traffic environment. In order to realize any of the higher levels of automation, vehicles would need to start taking over the monitoring of the traffic environment, and be able to react to it autonomously (see Figure 1.1). Situational awareness on the part of the vehicle is required.

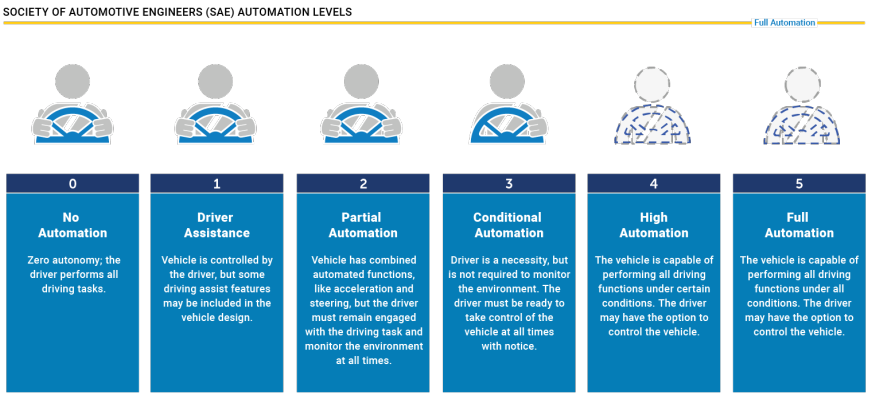


Figure 1.1: A diagram of the five levels of autonomy as defined by the international society of automotive engineers.

1.2 Situational Awareness in Traffic

1.2.1 Situational Awarenesss

Situational awareness is defined as the perception of environmental elements and events with respect to time or space, the comprehension of their meaning, and the projection of their future status (Endsley, 1995). The term situational

awareness was originally coined in human factors engineering research as a crucial skill for military aircraft crews to reduce human errors in decision making. With regard to traffic, situational awareness of an autonomous vehicle includes awareness of the road conditions, traffic rules and regulations and most difficultly the trajectories of all other traffic participants directly surrounding the autonomous vehicle and their positions in the immediate future. Within the context of this research, we will focus on the most challenging part: Situational awareness of autonomous vehicles with regard to other traffic participants. This starts with the detection of other traffic, and also encompasses the identification of detected traffic and estimation of the future position of the identified traffic participants.

1.2.2 Traffic Detection

Within the context of autonomous vehicles, reliable real-time detection of other traffic is a crucial first step. Self-driving vehicles without this ability would not be able to autonomously participate in traffic.
Reliable object detection has been a long-standing target of research within computer vision. Detection of an object in the context of computer vision means localization of the object within the larger image as well as correct classification of the object. Needless to say object detection is more challenging than mere object classification alone.
Recently however, advances in deep learning have enabled multi-class object detection on a single-frame basis within raw video-stream data with the use of deep convolutional neural network architectures such as the 'YOLO' network (Redmon & Farhadi, 2017) (See Section 3.1).
This allows for relatively accurate detection of different vehicle types and other relevant traffic related object classes from dash-cam footage (for a real-life illustrative example of state-of-the-art object detection from dash-cam footage, see Figure 1.2). This opens up the possibility for self-driving cars to perceive the other vehicles in their vicinity, which would be a critical step towards autonomous driving.

1.2.3 Necessity of Tracking

Despite being an essential step towards situational awareness in traffic, traffic detection by itself is not enough to constitute situational awareness. Even if a multitude of camera's and other sensors were used, the incoming information would need to be somehow integrated.
In order to effectively use the knowledge of the existence and locations of surrounding traffic participants for intention prediction not only detection, but also an aspect of cognition is needed to gain a deeper understanding of the traffic situation. Specifically the positions of surrounding traffic participants need to be retained in memory in order for the autonomous vehicle to track the movements of the surrounding traffic participants and be aware of their trajectories and likely future paths.
The conceptual difference between the state of awareness that can be realized though traffic detection and the desirable state of full situational awareness in traffic illustrated in Figure 1.3.

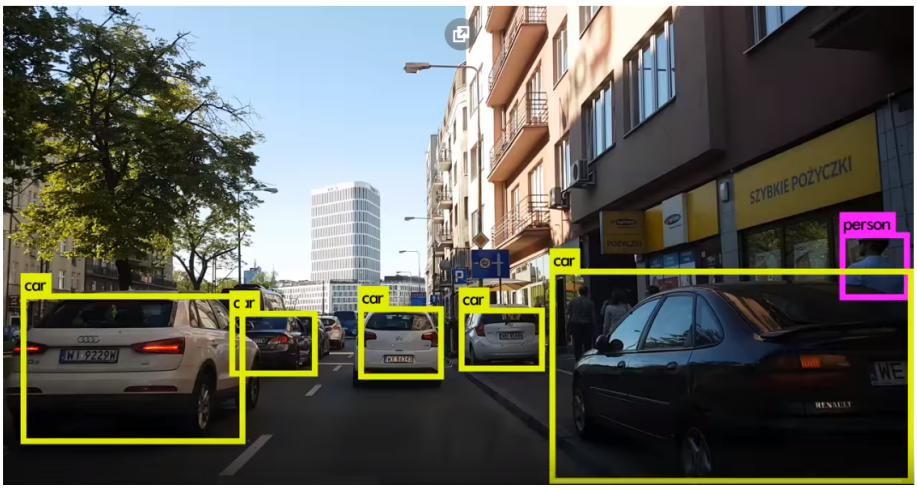


Figure 1.2: Traffic detection on a dash-cam image using the convolutional neural 'YOLO' network (Redmon & Farhadi, 2017).

The computer vision aspect of tracking a single object from video feed remains a challenging task in its own right due to the variation in scale, orientation and illumination and the possibility of temporary occlusion. In addition to this, the detection of vehicles from video-stream data is not consistent for every frame where these vehicles appear. Therefore, detections of the same traffic participant need to be linked in subsequent time frames to form a trajectory for this traffic participant. For this reason it is necessary to identify each traffic participant in each frame.

1.2.4 Necessity of Identification

In order to detect the position shifts of traffic participants from one time step to the next in a traffic environments which contains multiple traffic participants, each detected traffic participant needs to be uniquely identified. It may be worth considering that for the human brain detection and identification of moving objects are a synchronous process as well. When viewing video frames of a moving object in high enough frame rate, we perceive this as one object in continuous motion.
Identification or instance classification is a challenging classification problem due to the fact that the number of unique instances is potentially infinite and the classical approach training a classifier on all possible classes (instances in this case) is clearly impossible. Therefore, an approach which generalized to unseen instances is required.

1.3 Research Question

In this paper, the possibility of situational awareness with regard to specific other traffic participants from the perspective of a self-driving car is researched.

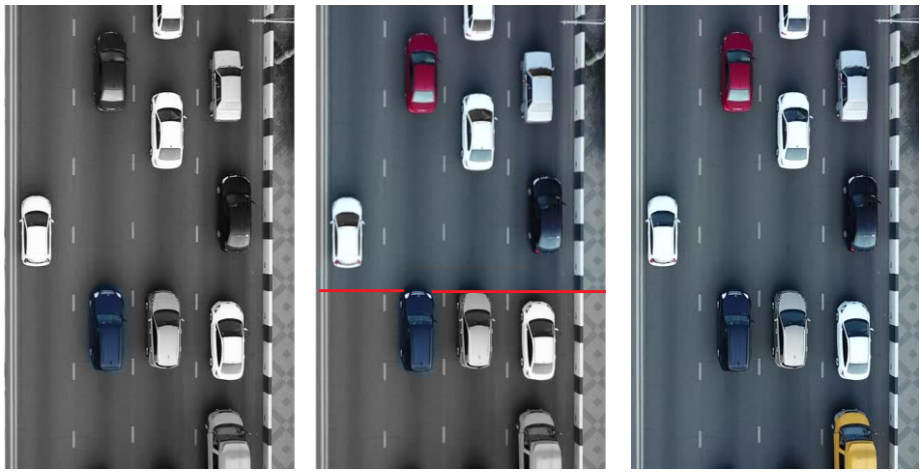


Figure 1.3: Conceptual illustration of an autonomous vehicle either with no situational awareness (left), partial awareness through traffic detection from dash-cam vehi(middle), the desirable state of full situational awareness through the combination of detection and memory retention (right). Full color symbolizes awareness.

This study aims to integrate the identification and tracking aspects of computer vision with the existing vehicle detection capabilities provided by deep convolutional neural networks and a working memory representation in order to further scientific understanding of situational awareness applied to the domain of autonomous vehicles.

1.3.1 Operationalization

To study this concept of situational awareness beyond frame-to-frame vision of autonomous vehicles based on traffic detection from camera images, an architecture has been built built to detect and uniquely identify traffic participants and track identified traffic participants from dash-cam footage over an extended time span while maintaining invariance to scale and orientation changes and robustness to periods of occlusion. To reflect a realistic traffic situation the identification and tracking architecture scales to a variable number of traffic participants. The output from this application has been subject to two experiments. The first experiment was designed to test the effectiveness of the instance classification method. The second experiment gauges the effectiveness of the multi-vehicle position tracking of the architecture. In this research the primarily focus is on vehicles, which is why the term 'vehicle' and 'traffic participant' are used interchangeably.
The remainder of this thesis is split into six more chapters. Chapter 2 lays out the theoretical framework in which this thesis and the methods used to do this research are embedded. In Chapter 3 the methods used in this research are explained in greater detail. In Chapter 4 the implementation of the identification and tracking architecture is explained. The datasets and experimental setups to test the architecture are discussed in Chapter 5. In Chapter 6 the results of the two experiments are shown, and these results and their implications are

discussed in Chapter 7, which concludes this thesis.

Chapter 2

Theoretical Framework

2.1 Deep Learning

2.1.1 Fundamental Principles

Within the field of machine learning, neural networks are systems consisting of computing units called neurons of which the outputs of some are used as weighted inputs for others, thus forming connected networks.
Generally, neural networks are organized in layers. The first layer receives data as input and is called the input layer, the last layer is trained to output the desired result and is called the output layer. In between there can be one or more layers of neurons called hidden layers. Starting from the input layer, outputs of each layer are used as weighted inputs for the next layer. See Figure 2.1 for a conceptual image of an artificial neural network.

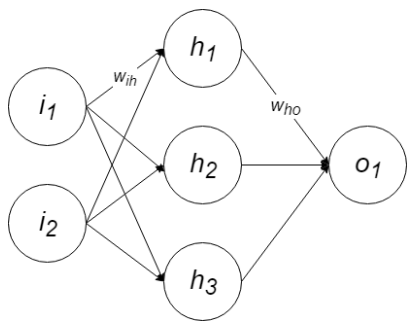


Figure 2.1: A diagram of a small neural network with two input neurons (labeled i), three hidden neurons (h), and a single output neuron (o). Each neuron in the input layer has weighted connections to all neurons in the hidden layer, which have weighted connections to the output neuron (the weights of which are referred to w_{ih} and w_{ho} respectively).

One of the key similarities between artificial neural networks and human brains is that neural networks are able to learn implicitly. They are able to approximate a function to return desired outputs after only being presented with input

and the desired targets, without being programmed with the underlying function. The learned approximated function can then be used to generalize to new examples of which the desired output is not known.
The first artificial neural networks were presented by McCulloch and Pitts in 1943 (McCulloch & Pitts, 1943). These networks had relatively simple neurons with binary inputs and outputs and a threshold as an activation function, but could nevertheless already be used to model any Boolean function.
The activation function is a key part of an artificial neuron. It determines what the output of a neuron is given a set of inputs. Starting from the 1980's, a number of different continuous non-linear activation functions were experimented with. In current practice, logistic functions such as the sigmoid function as well as Rectified Linear Units (ReLU's) are effective and commonly used neural network activation functions (Goodfellow, Bengio, & Courville, 2016).
Each connection in a neural network is weighted with weights which are initially random. The weights are tuned during the training stage of the neural network with the use of back-propagation and an optimization algorithm. For each training example presented to the network, a measure of the difference between the desired target output and the actual output of the network is calculated, called the loss.
The gradient of this loss with respect to each of the weights of the network is subsequently calculated. This is called back-propagation of the loss. The weights are updated in the gradient direction which minimizes the loss with the optimization algorithm. An example of this weight update step can be seen in Equation 2.1. Equation 2.1 shows the stochastic gradient descent algorithm (SGD). Each weight (ω), is updated by subtracting the average gradient (∇) of the loss (L) of each training example (x) with respect to that weight multiplied by a learning rate (η) which decreases over time. y denotes the target values of the examples.

$$\omega \longleftarrow \omega - \eta(\frac{1}{m}\nabla_\omega \sum_i^m L(f(x^i, \omega), y^i)) \tag{2.1}$$

The total training stage consists of a large number of iterations and ideally runs until the weights have converged to the values which best allow the neural network to generalize to new examples. In order to approximate this point, the data is commonly split in a training and a validation set, and the validation loss is used as a measure of the generalizability of the model learned by the network.
Deep neural networks are neural networks with many hidden layers. Even though a single neural layer is enough to represent any continuous function, deeper networks are commonly used. Deep networks have two main advantages over shallow networks. Firstly shallow networks need to be exponentially wider (contain many neurons per layer) to attain the same power. Secondly, deep networks tend to generalize better than comparably wide shallow networks.

2.1.2 Convolutional Neural Networks

One of the most common types of deep neural networks are the convolutional neural networks (CNN's) (LeCun, Bengio, et al., 1995). CNN's were first developed in the early 1990's, but due to the large computing power needed for CNN's to be effective, they only started coming into popularity around 2012.

CNN's are most useful for problems where the data contains a strong local dependencies and a regular grid-structures. They are most commonly applied to image data, but have also been used to analyze time series, volume and video data. CNN's have been shown to become very powerful in these cases when trained on large labeled datasets.
A CNN consists of convolutional layers. These layers are locally connected grids, whose connections are defined with the discrete convolution operation. Figure 2.2 shows how an output image is produced from an input image with the two-dimensional version of the discrete convolution operation. For the mathematical definition of the two-dimensional discrete convolution operation, see Equation 2.2, where an input matrix X is convolved with a kernel K of size $n \times m$ to produce an output matrix O.

$$O(i,j) = (X * K)(i,j) = \sum_n \sum_m X(i-n, j-m)K(n,m) \tag{2.2}$$

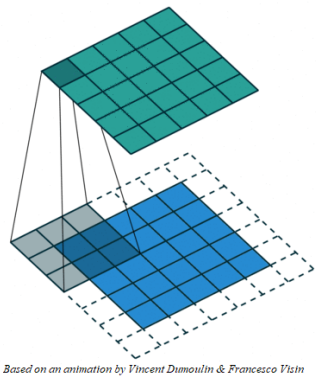


Figure 2.2: A diagram depicting the convolution of an input image (blue) with a 3×3 kernel to produce an output image (green). In this example padding is used to make the size of the output equal to that of the input.

In a convolutional layer of a CNN each input is convolved with a number of different kernels to produce just as many outputs called feature maps. The values of the convolution kernels are the trainable parameters (weights) of the CNN. The output feature maps of a convolutional layer can serve as input for a subsequent convolutional layer. Typically, a CNN consists of multiple convolutional layers and possibly subsampling layers, followed by by one or more fully connected layers leading to the output of the network (See Figure 2.3).

When compared to regular (fully connected) neural networks, CNN's have several advantages due to the fact that they have local connections between layers. Firstly, a convolutional layer has fewer parameters that need to be trained, compared to an equal sized fully connected layer, which allows for faster training of CNN's compared to their regular counterparts with equal computational resources. CNN's also have the advantage of being relatively shift-invariant, which

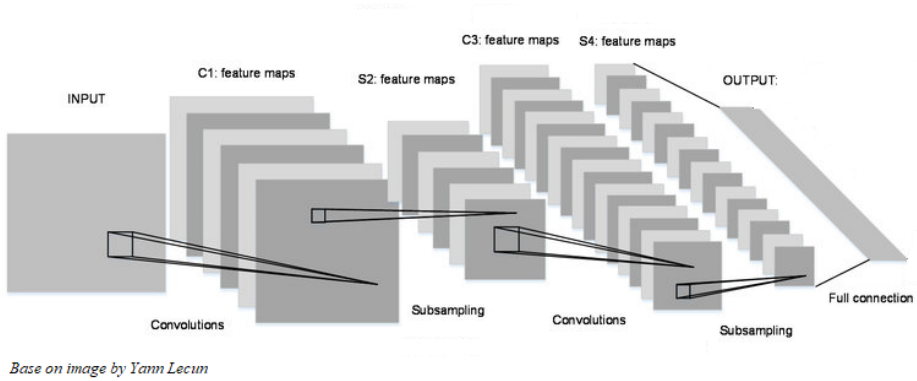


Figure 2.3: A diagram depicting a typical CNN with two convolutional layers, two subsampling layers and a fully connected layer at the end to produce a classification output vector.

is why they are commonly used for problems involving image-based data.

Special Layers and Architectures

CNN's frequently make use of subsampling in between convolutional layers, also called pooling. Maximum pooling, or max-pooling, is the most common. When max-pooling is used after a convolution, only the maximum value in each local neighborhood of each feature map is kept. See Figure 2.4 for an example. Average pooling is sometimes also used. For certain types of CNN's such as generative networks, sometimes the opposite is done, and upsampling (zooming) layers are included after convolutional layers.

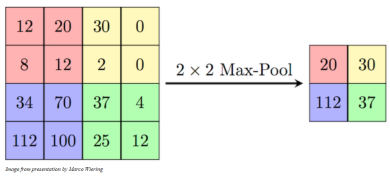


Figure 2.4: An example of max-pooling with a 2×2 pooling window and a stride of 2.

Deep convolutional neural networks can suffer from a degradation problem, where the training error goes up when more layers are added to the network (K. He, Zhang, Ren, & Sun, 2016). To combat this, some deep CNN architectures have skip connections between early layers and special residual layers later in the network. With a skip connection, the output of the early layer is added to the output of the layer previous to the residual layer, to force the residual layer output to be more similar to the output of the early layer. This has been shown to improve performance in some cases, because it has a regularizing effect on deep networks which are prone to overfitting, by forcing the later convolutional layers to be closer to the identity function.

2.1.3 Transfer learning

Pretrained deep CNN's have been previously used as fixed feature extractors on new domains. See for example (Boufenar, Kerboua, & Batouche, 2018) where a deep CNN pretrained for object recognition in images, is applied to extract features for Arabic character handwriting recognition. This practice is known as transfer learning, and it is used in applications of image recognition where are no annotated datasets large enough to effectively train a deep neural network from scratch. If done successfully tranfer learning could greatly improve the performance of learning for new domains by avoiding much expensive data labeling efforts (Pan & Yang, 2009).
When a pretrained deep CNN is used for transfer learning, typically the last layers of the network are retrained using a relatively small annotated dataset while the earlier layer weights are kept the same. The underlying assumption being that the implicit features originally learned by the lower layers of the network have a general quality, which is shared between the domain the CNN was trained on and the new transfer learning target domain.
Deep neural features have previously been used to cluster images into semantic categories on which the networks producing these features were never explicitly trained, thus demonstrating the generality of these deep features (Donahue et al., 2014).

2.1.4 Deep learning and identification

As discussed in Section 1.2.4, identification poses a challenge which can not be solved by conventional multi-class classifying techniques, because of the nearly infinite number of different possible instances. One solution is to create a feature representation of each instance, with features which maximize the differences between instances, while minimizing the variation for different examples of the same instance.
In the domain of facial recognition, a deep CNN was successfully trained to learn distinctive features which could identify unique faces while minimizing the variation for different examples of the same face (Sun, Chen, Wang, & Tang, 2014). A deep neural network has also previously been used to learn features of network traffic protocols to uniquely identify these (Wang, 2015).

2.1.5 Recurrent Neural Networks

Recurrent neural networks (RNN's) are specially designed for time dependent data and other sequence data. RNN's are frequently used to predict the next step in a sequence based on a pattern of previous steps. In an RNN there are recurrent connections with a time delay, which cause the activation in the network at each time step to be determined by the input at that time step, but also by the activation of the network in the previous time step. RNN's can be considered deep neural networks when applied to large time sequences, because in that case their time-unfolded computational graph can become very deep (Goodfellow et al., 2016).

Long Short-Term Memory networks

One of the most successfully used types of RNN's for many applications are the Long Short-Term Memory networks (LSTM's) (Hochreiter & Schmidhuber, 1997). LSTM's are recurrent neural networks with neural memory gates. The gated self-loops of LSTM's makes them more robust to the vanishing gradient problem compared to regular RNN's, because it allows LSTM's to keep the gradient flowing for a long duration.
Figure 2.5 shows a single LSTM cell. The arrows represent weighted neural connections. The connections marked with a black square are recurrent. The LSTM has an internal state with external input and a recurrent self-loop which is updated each time step. The input and output of the LSTM cell are gated with sigmoidal gates. There is also a sigmoidal gate on the recurrent self-loop weight called the forget gate. All sigmoidal gates of the LSTM are connected to the LSTM input at each time step as well as recurrently to the internal state of the LSTM at the previous time step.

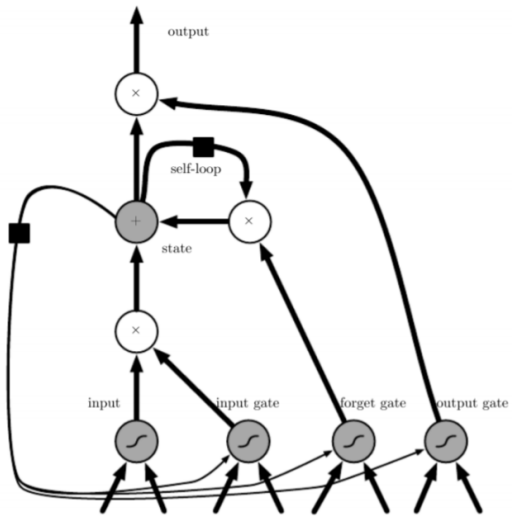


Figure 2.5: An diagram of the architecture of an LSTM cell (Goodfellow et al., 2016). Recurrent connections are denoted with black squares.

2.2 Deep Learning and Autonomous Driving

2.2.1 Object Tracking

A number of different tracking methods have been developed for visual object tracking. A common approach is tracking-by-detection, where the object to be tracked needs to be detected in each frame. This can be done with different techniques. For example, in (Ding, Chen, Zhao, Han, & Liu, 2018) a type of correlation filtering was used, whereas (Danelljan, Robinson, Khan, & Felsberg, 2016) used a deep learning approach, with a convolutional recurrent neural network. Tracking-by-detection has been previously combined with a probabilistic

model, such as (Benjaminsson & Rosenberg, n.d.), where a detection CNN was used in combination with a Poisson multi-Bernouilli mixture filter.
There are also tracking methods which do not rely on detection. (Ren, Liu, Shi, & Li, 2017) used a random forest ensemble tracker combining the outputs of many "weak" learners which try to distinguish the tracked object from the background. (Vatavu, Danescu, & Nedevschi, 2015) use a occupancy grid created from a stereo-vision elevation map as input to estimate the locations of multiple objects in a scene.
Although not specifically designed for the purpose of object tracking, detection and classification of objects within a larger scene, has been achieved at near real time speed on conventional hardware with the deep convolutional YOLO (You Only Look Once) architecture (Redmon, Divvala, Girshick, & Farhadi, 2016) (Redmon & Farhadi, 2017). The internal class representations of the YOLO architecture were also shown to generalize well: Good performance was obtained on artwork, after the network was trained on real-life data (Redmon et al., 2016).
The YOLO network has been adapted for pedestrian detection for video surveillance purposes (Molchanov, Vishnyakov, Vizilter, Vishnyakova, & Knyaz, 2017). The possibility of pedestrian detection using deep learning methods for the purpose of autonomous driving has also been studied. A deep convolutional neural network has been used to detect pedestrians on crosswalks (Guidolini et al., 2018) and on highway entrances (X. He & Zeng, 2017).

2.2.2 Vehicle Tracking

Deep neural networks have previously been applied to solve a number of problems in the specific domain of scene understanding in autonomous driving. A dynamic spatial attention RNN with LSTM cells has been deployed for accident detection based on dash-cam footage (Chan, Chen, Xiang, & Sun, 2016). (Scheidegger, Benjaminsson, Rosenberg, Krishnan, & Granstrom, 2018) used a deep Region-based CNN in combination with a Poisson multi-Bernoulli mixture filter to track vehicles in 3D based on mono camera feed, whereas (Frossard & Urtasun, 2018) developed a multi-sensor solution for tracking vehicles in 3D with the help of multiple finetuned CNN's.

2.3 Kalman Filtering

A long established solution to prediction problems based on physical trajectories is the Kalman filtering method (Kalman, 1960). Kalman filtering is an algorithm which uses a series of past observations over time to generate an estimate of the observation in the next time step, often based on a physical model.
The Kalman filtering algorithm consists of a prediction step and an update step. In the prediction step, an a priori estimate $x_{t|t-1}$ of the current state is made based on the previous state $x_{t-1|t_1}$ and a control input u_t. The control input is defined by the user and depends on the model. The prediction step equations can be seen in equation block 2.3, where A is the state transition model and B is the control input model, P is the state error covariance matrix which reflects

the certainty of the prediction and Q is the processing noise.

$$\begin{aligned} x_{t|t-1} &= A * x_{t-1|t_1} + B * u_t \\ P_{t|t-1} &= A * P_{t-1|t_1} * A^T + Q \end{aligned} \tag{2.3}$$

In the update step, a new measurement y_t is taken into account and an a postiori state estimate $x_{t|t}$ is made based on the a priori state prediction and the new measurement. The degree to which the new measurement is relied on compared to the prediction based on the model is determined by the Kalman gain K_t, which is calculated first. The update step equations can be seen in 2.4, where H is the measurement model, S_t is the measurement covariance, and R is the measurement noise. The lower R is compared to $P_{t|t-1}$ the more the measurement is trusted compared to the a priori prediction and vice versa.

$$\begin{aligned} K_t &= P_{t|t-1} * H^T * S_t^{-1} \\ S_t &= H * P_{t|t-1} * H^T + R \\ x_{t|t} &= x_{t|t-1} + K_t * (y_t - H * x_t) \\ P_{t|t} &= P_{t|t-1} - K_t * S_t * K_t^T \end{aligned} \tag{2.4}$$

Kalman filtering has previously been used in combination with neural networks. Its addition to recurrent neural networks has been successful leading to a faster and more optimal convergence during training compared to classical gradient methods (Trebatický, 2005).

Chapter 3

Methods

For the purpose of this research, a online probabilistic tracking-by-identification type multi-object tracker was developed based on the YOLO architecture. Tracking-by-identification is a novel concept, which is related to tracking-by-detection, but distinct in the sense that it uniquely identifies vehicles based on so-called activation signatures which are taken from the same network that provides the detection.
Moreover, the developed tracker is also distinct from the single object tracking approaches described in (Ding et al., 2018) (Danelljan et al., 2016) and (Ren et al., 2017) in the sense that it scales up to multiple objects, distinct from the offline approach using pre-computed detections in (Benjaminsson & Rosenberg, n.d.) because it is designed for real-time applications, and distinct from the lower level occupancy grid representation in (Vatavu et al., 2015).

3.1 YOLO

The YOLO network (Redmon & Farhadi, 2017) (short for You Only Look Once) is a deep convolutional neural architecture capable of detection and classification of objects within a larger scene, at near real time speed on conventional hardware. After successful training, the YOLO network is able to distinguish a large number of object categories. The network's output consists of class labels, confidence and bounding boxes for each detected object in an input image. Importantly, the speed of the YOLO network provides feasibility for online object tracking. Another valuable property of YOLO is its generalizability, as demonstrated by the original authors by comparing its performance on real life image data and artwork.
Successfully training very deep neural networks such as YOLO is time consuming and also arduous considering the necessary data preprocessing, data augmentation, network configuration and hyper-parameter tuning that it entails in practice. Furthermore, in real life the conditions it is hard to predict all possible use cases and the real data can differ from the training data (Redmon et al., 2016). Evaluation of performance of the existing networks is therefore necessary and can lead to a substantial gain in time and thereby lead to increased efficiency if successful. For this reason, we decided to take use a YOLO net-

work with weights pretrained on a general set of objects and evaluate it on the domain-specific task of vehicle detection in traffic. This is in contrast to (Choi, 2015) and (Benjaminsson & Rosenberg, n.d.), where the detection network was trained on the domain-specific and particular type of data of the evaluation data set. The network we use is pretrained on the COCO (Lin et al., 2014) dataset.

3.1.1 YOLO v3

The YOLO version that is used, is YOLO v3 (Redmon & Farhadi, 2018). This version of YOLO is the latest and best performing version of YOLO at the moment of this writing. YOLO v3 is especially better than previous versions of YOLO at detecting small objects, which we expect to be useful when trying to detect faraway vehicles. YOLO v3 is multi scale with three output layers for different sizes of boundary boxes and 106 layers total, all of which are convolutional. The YOLO v3 network employs two kinds of skip connections. Firstly, throughout the network there are short skip connections named routes, in which three layers are skipped to form small residual blocks. Secondly, there are two long skip connections named shortcuts, where the output from early layers in the network is concatenated to residual layers just before the two last output layers. Upsampling layers are used to make this concatenation of early layer output via skip connections possible. A layout of the network architecture can be seen in Figure 3.1.

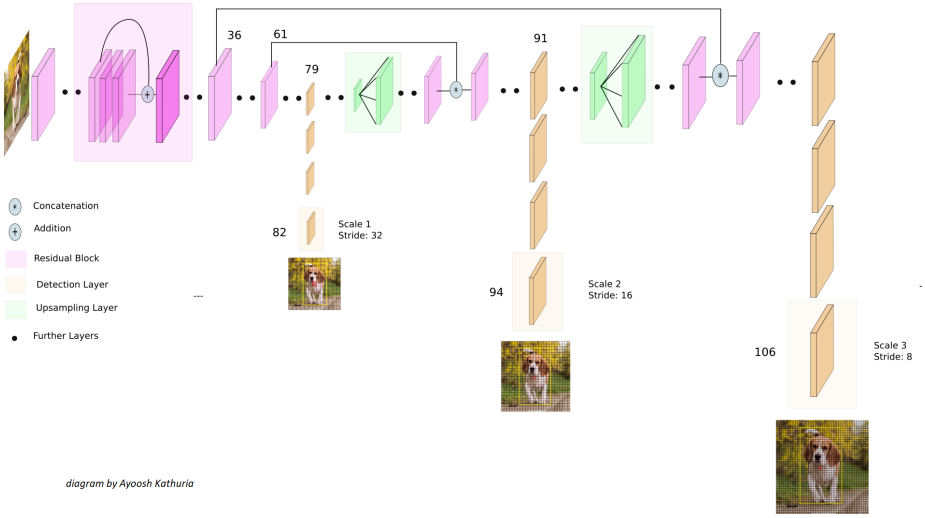


Figure 3.1: An annotated diagram of the architecture of the YOLO v3 network. Layers numbered 82, 84 and 106 are output layers (Kathuria, 2018).

3.2 Tracking

3.2.1 Tracking challenges

Deep learning architectures have the known problem of being highly sensitive to slight variations in the input data and exploratory data analysis showed that the YOLO v3 network is no exception in that regard. When looking at stream data of vehicle traffic, vehicles were frequently detected correctly in one frame, only to be incorrectly detected in the next frame after only a minimal change in position. Three types of detection artifacts can be distinguished.
Firstly, in some cases a 'blink artifact' can occur where previously detected vehicle can go completely undetected for one or multiple frames for no apparent cause, only to be detected again after that. Secondly, a 'conflation' artifact can occur, where the network groups two or more vehicles together as a single detection. Conversely, there is a 'division' artifact, where part of a vehicle is classified as being the full vehicle. Conflation and division artifacts cause the position estimate to be inaccurate, because of the incorrectly registered bounding boxes. Examples of these three types of artifacts can be seen in Figure 3.2.
Lastly, there is also a class labeling inconsistency. When a vehicle is 'flickering' in between two learned classes, because the class with the highest activation varies between frames. For example, a van which is sometimes classified as a car and sometimes classified as a truck. The various detection artifacts are problematic for tracking purposes, while the classification inconsistencies are undesirable when considering identification.

3.2.2 Tracking methods

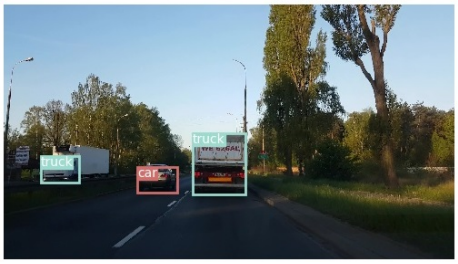
LSTM networks

To address the detection artifacts and create a stable tracker, a predictive system is needed on top of the vehicle detections generated by a the deep convolutional detection network like the YOLO v3 network. Intuitively we can expect the positions of participants in traffic to have some degree of continuity in a short time window. Recent previous detections could be used to predict the position of a vehicle for a few time steps when for instance a 'blink' artifact or an occlusion from another vehicle occurs. For this reason, LSTM networks (See Section 2.1.5) were investigated as a method of tracking the vehicles.

Kalman Filtering

To address the detection artifacts and create a robust vehicle tracker, we also investigated the possibility of stabilizing the detections with the mathematical approach of using a Kalman filter (See Section 2.3).
In using the Kalman filter two assumptions were made about the trajectories of vehicles in traffic. Firstly, inherent in the Kalman filtering approach is the assumption that the underlying process generating the observations can be modeled linearly. In the case of moving vehicles, the travelled distance and velocity can be modelled physically. The other assumption can be characterized as an assumption of continuity. When a vehicle has previously been detected and it is

(a) Conflation artifact example: Instead of two trucks, only one truck is detected and its bounding box is estimated incorrectly.



(b) Division artifact example: Only the cabin of the truck on the left is detected resulting in an inaccurate bounding box.

(c) Blink artifact example: The left car in the first (left image) frame suddenly goes undetected for a few time steps possibly due to the angle or partial occlusion (middle image) before it is detected again (right image).

Figure 3.2: Examples of common detection artifacts that occur in YOLO v3.

not on the verge of disappearing from the camera view, the assumption is that the car is still there in the next frame, and that failure to detect it would be the result of a blink artifact or a temporary occlusion.

3.3 Instance Classification

3.3.1 Challenges of Instance Classification

The YOLO image detection network significantly reduces the dimensionality of dash-cam video data from thousands of pixels per frame, to a few detected traffic participants. Despite this, the YOLO output is still subject to a big data problem, when considering the trajectories of vehicles from an input video feed, because of the exponential number of ways different detections could be combined into multi-frame trajectories.

Due to the highly dynamic nature of the traffic, the number of different unique instances of vehicles which can be encountered while driving is almost infinite. An identification solution would need to be equally dynamic. The classical multi-class-based approach with a fixed number of classes can not satisfy this requirement.

One obvious solution that was briefly explored, is to automatically read the unique signature given to each motor vehicle by the government: the vehicle's licence plate. Automatic licence plate recognition (or ALRP) is an established technique with wide usage in the fields of police work and security. However, the typical ALRP application involves a static camera identifying a specific ve-

hicle in a single frame, from which that vehicle is determined to be present in a general area around a general time. This is not quite as challenging as our goal of tracking-by-identification, where we want to identify specific vehicles during their trajectory through our field of view in as many frames as possible. Being able to (re)identify vehicles at any moment these are in view of a vehicle dash-cam, requires a flexibility which automatic licence plate recognition does not provide. A vehicle's licence plate is often not visible or readable during all parts of its trajectory due to distance, angle, occlusion, poor illumination or motion blur. In addition to this, a number-plate recognition system would not be able to generalize to bicycles and other vehicles without number plates. It is also worth considering that humans do not rely on identifiers like number plates when perceiving surrounding traffic while driving.

3.3.2 Signatures based on Feature Representations

Handcrafted Feature Signatures

Creating an identifying signature for each detected vehicle instance based on a feature representation of each vehicle could provide a more reliable means of instance classification. There are multiple ways to create such a feature representation. The traditional approach would be to manually pick handcrafted features and extract them from the detected vehicle image. Commonly used handcrafted feature descriptors include SIFT (Lowe et al., 1999) and SURF (Bay, Ess, Tuytelaars, & Van Gool, 2008). However the problem with the approach of picking handcrafted features is that what constitutes a 'good' feature is domain specific and there is no guaranteed way to avoid picking features which do not generalize well to new instances. Furthermore, handcrafted features are typically low-level (i.e. number of corners, edges etc.), which could in some cases be sufficient to distinguish completely different object classes, but would not be likely to be successful in distinguishing instances of the same object class.

Deep Feature Signatures

Another strategy is to generate distinguishing features automatically, using a CNN as a deep feature extractor. The first layers of a typical convolutional network trained on image classification contain low level feature representations, such as edges, corners and color patterns. Later layers contain more complex abstract feature maps, which finally culminate into single values relating the input image to each learned class in the last layer (See Figure 3.3 for a graphical illustration). The phenomenon of how low level features are being combined into more complex ones within a CNN, is also the guiding principle of the transfer learning paradigm applied to CNN's. The last layers are retrained to suit a new domain, while the earlier layers containing the lower level representations are kept the same (see Section 2.1.3).

In the transformation from low level feature information to a final class, quite a lot of activation is pruned by dimensionality reduction in the network and by the network weights. The feature maps in the middle layers of the CNN of may contain additional activation patterns which are irrelevant for the classification,

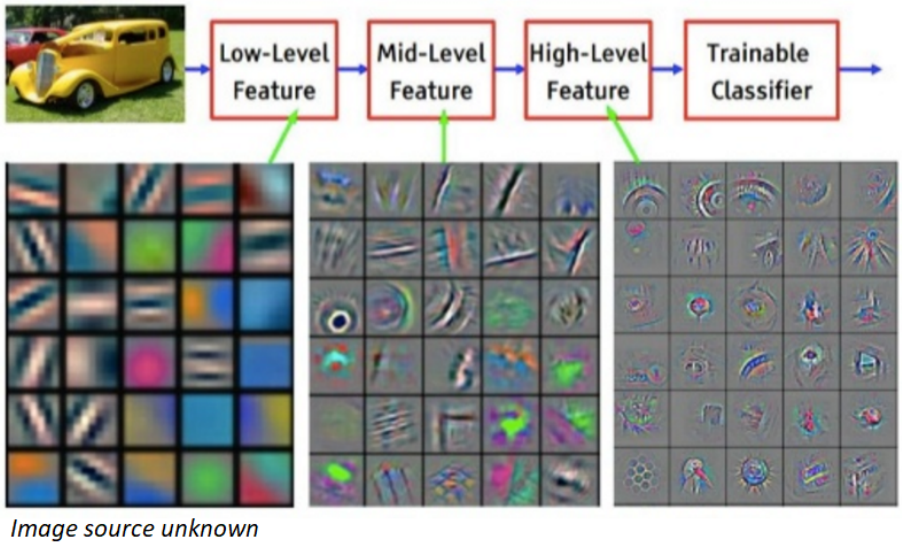


Figure 3.3: Example of low, mid, and high level CNN features.

but could be relevant for the identification of instances. Identifying signatures of traffic participants could be constructed from the mid-level feature maps of a sufficiently deep neural network.
One option would be to "cut" all objects discovered by the YOLO network out of the scene using the bounding boxes, and feed the resulting images into a second pretrained deep image classification CNN, from which the final layers have been removed. However, the addition of a second deep network would slow down the processing time of each frame considerably. The resulting tracker would lose one of the main advantages of the YOLO architecture, which is its near real-time detection speed, and would not be as usable for online tracking applications.

DIFS: Deep Integrated Feature Signatures

The ideal solution would be to extract deep integrated features directly during detection, if the feature maps produced by the YOLO network itself would lend themselves to the creation of signatures which could be used for instance classification. In the case of the YOLO network it is plausible that this could be done, because the YOLO network is not only trained for object detection, but also simultaneously for classification. For this reason, we assume low level feature representations would need to be present in the network.
We explored this possibility of using the feature maps of the convolutional layers of a YOLO network to create deep integrated feature signatures (DIFS) to uniquely identify detected traffic participant instances. Figure 3.4 shows the creation of a DIF-signature from feature map information starting from a detection by the YOLO network. From the final bounding boxes outputted by YOLO, we get the region where an object is detected. This region is then translated to the same relative region in feature map space for a predetermined layer of the network (referred to as the signature layer). Due to the fact that YOLO is fully

convolutional, we expect feature map information in this region to accurately represent the detected object. After determining the corresponding detection region in feature map space the summed activation over this region is subsequently calculated for each of the feature map images in the signature layer. This creates an n-dimensional signature for each vehicle detection, where n is the number of activation images in the signature layer. There is no trivial way of determining the optimal network layer to serve as signature layer. Therefore, performance of DIFS taken from the feature maps of signature layers at different stages in the YOLO network are compared as part of this research.

.

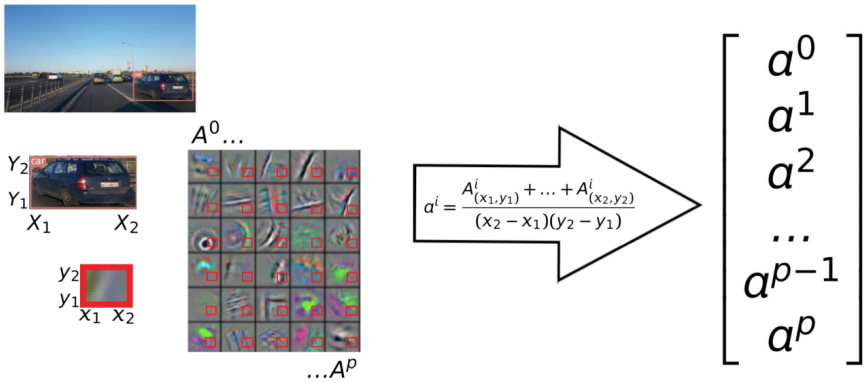


Figure 3.4: Diagram showing the process of DIFS from YOLO feature maps. Output bounding box coordinates $[X_1, X_2, Y_1, Y_2]$ of a detected object correspond to the relative region $[x_1, x_2, y_1, y_2]$ in p feature maps at layer A of the network, which are then used to create signature vector a of length p where a^i is equal to the summed activation over region $[x_1, x_2, y_1, y_2]$ of feature map A^i.

Chapter 4

Instance Tracker Implementation

4.1 Vehicle Instance Detection

For the purpose of this research, a multi-object tracker has been developed, capable of uniquely identifying and keeping track of a flexible number of surrounding vehicles. Each time step the tracking system receives an update to its visual 'scene', which is modeled by a frame from a dash-cam stream. The pretrained YOLO v3 network is applied to this 'scene' image, for the base-line detection of vehicles. DIFS for the detected vehicles are generated during the detection pass of the network. A diagram of the different components of the instance tracking framework (with the use of Kalman filtering) can be seen in Figure 4.1. The tracker is developed in Python. A Pytorch (Ketkar, 2017) implementation of the YOLO v3 network is used.

4.1.1 Vehicle Instantiation

The tracker has an activation-based memory for vehicles that it detects in its vicinity. When a vehicle is detected with a close enough signature to a vehicle that is already in the memory from a previous time step and the location of the newly detected is approximately at the location where the previously detected vehicle is expected to be by the tracker at the current time step in accordance with the assumption of continuity, the new vehicle detection is assumed to be the same vehicle. When this is not the case the detection is assumed to be a new vehicle, and a new node in the memory is dedicated to it.

4.2 Tracker Memory

The memory of the tracker is activation based and modeled loosely on reinforcement through repetition and the forgetting curve. New vehicles in memory receive a base activation. When a vehicle is perceived across multiple time steps in succession the activation increases up to a maximum, which resembles full

attention to that vehicle. When a vehicle is not perceived its memory activation value decreases. Vehicle nodes with an activation below a certain minimum threshold are dropped from memory, and no longer tracked.

4.3 Instance Tracking

Seeing as both Kalman filters and neural networks including LSTM's have a fixed number of outputs, using a single LSTM or Kalman filter to track a completely variable number of objects is likely not possible. In order to meet the goal of multi-vehicle tracking with a scalable amount of vehicles, a trained LSTM network or Kalman filter will be instantiated for each instance to be tracked, creating a dynamic flock of tracking networks or filters. Essentially, this can be seen as single object tracking in parallel.

4.3.1 Kalman-Filtering Tracker

For the Kalman-filtering tracker, measurements for each time step are the vehicle detections of the YOLO network in the 'scene' image. When a previously detected vehicle is again detected in the current time step, its position, speed and possibly its acceleration are updated with the new measurement, corrected for noise. If a vehicle is not detected in the current time step, the position of the current time step is predicted based on the previous position, speed and acceleration.

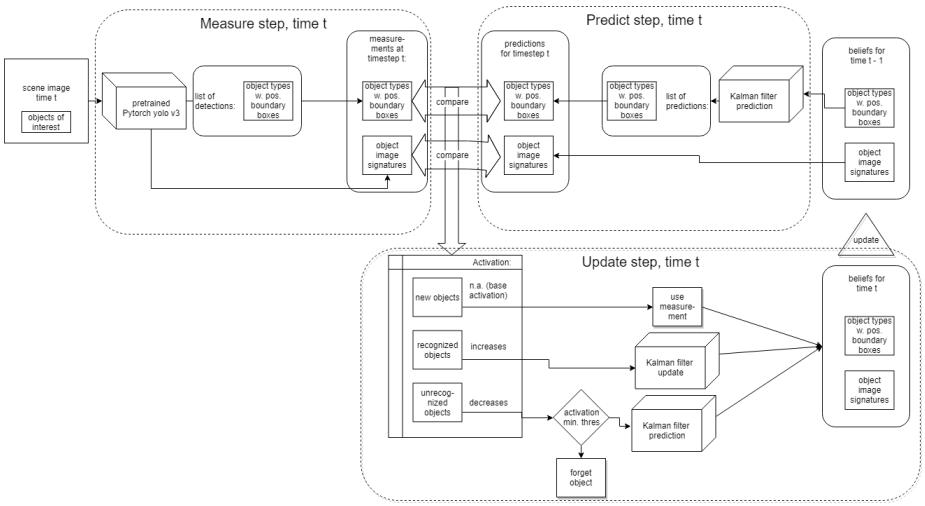


Figure 4.1: A sketch of interaction between the components of the instance tracking framework using Kalman-filtering, during a single iteration of the Kalman filtering loop.

4.3.2 LSTM-based Tracker

Like the Kalman-filtering tracker, the LSTM-based tracker also uses the observations from the pretrained YOLO v3 network. For each new vehicle instance to be tracked a trained LSTM network is instantiated, with the observation from YOLO being its initial trajectory. If a tracked vehicle instance is not observed by the YOLO network in at a certain time step, the predicted position for that instance for that time step by the LSTM will be added to the input trajectory fed to the LSTM in the next time step.

LSTM network properties

The LSTM network used for the LSTM-based tracker consists of an input layer, two LSTM layers, a hidden layer and an output layer. An input sequence length of 75 was chosen, because the shortest sequence in the training data set contained approximately this number of video frames. The other sequences in the training data were cut up into partially overlapping sequences of 75 frames.
Each input consists of 6 parameters for each vehicle to be tracked in that sequence during training, which are: a unique ID constant for that vehicle, a value between zero and one to signify whether the vehicle is being tracked, and the center coordinates and relative bounding box width and height of that vehicle.
The LSTM network is trained for 100 epochs on the KITTI suite training data (See section 5.1) using early stopping and with Adam optimization (Kingma & Ba, 2014) with a Mean squared Error loss function[1] and an initial learning rate of 0.001.

[1] A different loss function based on intersection over union was also briefly considered, but a pilot experiment showed this did not converge, likely because it turned too large a part of the parameter space into a plateau in the cost landscape.

Chapter 5

Experiments

5.1 Datasets

The tracking-by-identification tracker is evaluated on two different datasets. In order to obtain a basis for possible comparison with other multi-vehicle tracking methods, the mono camera image sequences of the KITTI benchmark data suite for autonomous driving is used (Geiger, Lenz, Stiller, & Urtasun, 2013). Additionally the tracker performance is evaluated on 4K dash-cam video data, which has been annotated by hand for the purpose of this research[1].
The KITTI suite contains 8026 images with a 1242 by 373 pixel aspect ratio, and 30601 labeled vehicles. The 4K video data contains 64073 high definition images with a 3840 by 2160 pixel aspect ratio. The 4K dash-cam video data was previously completely unlabeled. However, exploratory analysis with the YOLO v3 network revealed 165056 vehicle detections, of which 4317 have been annotated. The frame rate for the 4K video data is 30 frames per second, where as the KITTI suite data was recorded at 10 frames per second.

5.2 Experimental Setup

5.3 Instance Classification using DIF-Signatures

In this experiment we evaluate whether the deep integrated feature signatures (DIFS) from the YOLO network is an effective feature representation for the purpose of instance classifications, in the sense that the intra-instance variations of DIFS are small compared to their inter-instance differences (Sun et al., 2014). To assess the intra-instance and inter-instance similarity of the DIFS a K nearest neighbor (KNN) classification was made, using five fold cross validation.
A random sample of 20 examples for each unique vehicle with 20 or more occurrences was taken of each data set. Vehicles with less than 20 examples in the data were disregarded. This way two balanced sample sets were created. The KITTI sample set contains 340 samples (17 vehicle instances), whereas the 4K dash-cam sample set contains 700 samples (35 vehicle instances).
For selection of signature layers, the design of the YOLO v3 network with its

[1] Annotation was done with an annotation tool which was made specially for this purpose.

two residual activation bridges in the form of ‘short cut’ skip connections, and its three separate output layers played a large role. Four different sets of signature layers were chosen from the layers across the YOLO network. The selected signature layer sets are referred to in this paper as ‘early’, ‘middle’, ‘late’ and ‘last’. The ’last’ and ’late’ DIFS are taken from three different layers. The ‘last’ DIFS are taken from from the last layers before the YOLO output layers. The ‘late’ DIFS are taken from the layers after the ‘short cut’ skip connection layers where YOLO network splits into the three paths leading to the output layers. The ‘early’ and ‘middle’ DIFS are taken from two different signature layers. The ‘middle’ DIFS are taken from layers shortly before the ‘short cut’ skip connection layers. The signature layers for the ‘early’ DIFS were determined semi-randomly, with quasi-uniform distance with respect to their ‘depth’ in the network relative to their respective output layers, ‘middle’ and ‘late’ signature layers and input layer. For an overview of the position of the chosen signature layers within the YOLO v3 network architecture see Figure 5.1.

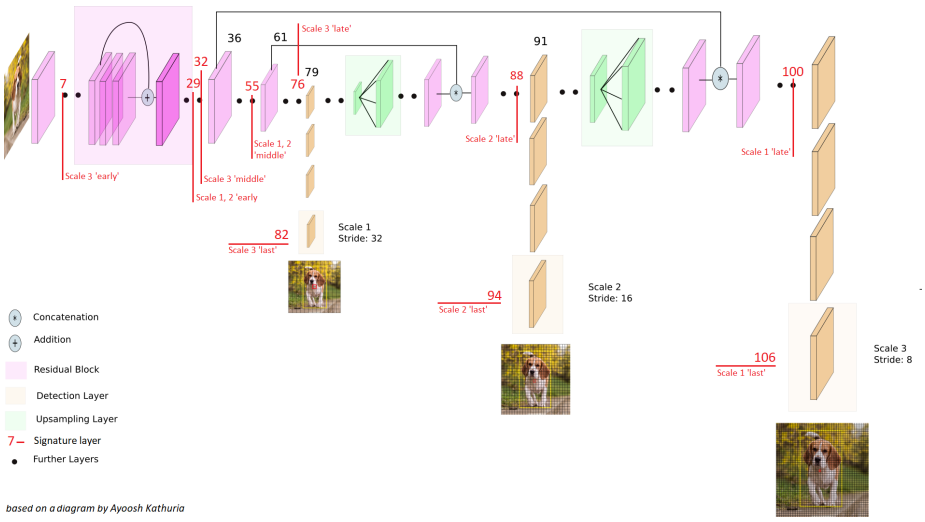


Figure 5.1: Adapted version of the YOLO v3 architecture by (Kathuria, 2018) shown in Figure 3.1 with the signature layers chosen for the DIFS evaluation highlighted in red.

The values in the feature maps of the signature layer are averaged in two different ways. Either by averaging the activation over the whole detection region, or by splitting the detection region into four equal quadrants and averaging the activation over those. The latter approach results in a four times larger DIF-signature vector.
Additionally, a number of different configurations were used for the KNN classifier. For each signature parameter setting a classification was made using a k-value of 1, 3 and 5 for each of two distance metrics: Euclidean distance and Manhattan distance.

5.4 Multi-Instance Tracking

To evaluate the multi-instance tracking performance of the YOLO-Kalman and YOLO-LSTM instance trackers, they were compared to the results of only using YOLO v3 detection on the 'car' class of the KITTI object tracking benchmark suite.

The reason we chose to use YOLO v3 detection as a benchmark for comparison, is because it represents the state-of-the-art in real time object detection, while allowing the evaluation the performance of these multi-instance trackers given the uncertainty caused by the possibility of imperfect measurements. Previous studies have used the ground truth anchors within the KITTI object tracking benchmark suite as starting points for trajectories (For example (Choi, 2015)). However we feel this presents an unrealistic picture of the capabilities of trackers evaluated this way, because no such ground truth anchors exist in the case of real life traffic situations.

Relevant performance metrics of the multi-instance tracking evaluation include metrics related to detection, which are the detection recall and the detection precision, the F1 score, which is the harmonic mean of the detection recall and precision and the false alarm rate (FAR), which is equal to the number of false positives as a percentage of the total number of detections.

Additionally, performance metrics which are developed specifically for multi-object tracking are used. These performance metrics are the CLEAR-MOT Multi-Object Tracking Accuracy and the percentage of Mostly-Tracked/Partially-Tracked/Mostly-Lost (MT/PT/ML) trajectories. These metrics also take into account ID-switches and sequence fragmentations that can occur when tracking multiple objects. The CLEAR-MOT Multi-Object Tracking Accuracy (MOTA) is a measure of successfully tracked trajectories compared to misses, mismatches and false positives. The MT/PT/ML metric are all measures of number of tracked trajectories compared to the total number of trajectories. A trajectory is classified as 'Mostly' tracked or lost if it is tracked or lost in more than 80% of the frames where it is present. (See (Bernardin & Stiefelhagen, 2008) and (Li, Huang, & Nevatia, 2009) respectively for more detailed explanations of the MOTA and MT/PT/ML metrics).

One of the parameters of the instance tracking evaluation which has a large influence on the performance is the YOLO confidence threshold. Candidate detections are accepted as true by the YOLO detection framework only if their confidence value (see Section 3.1) exceeds the YOLO confidence threshold. For this evaluation YOLO confidence thresholds of 0.2, 0.4, 0.6 and 0.8 were used, out of a possible scale from 0 to 1.

For the LSTM-based tracker, LSTM networks with 50, 100 and 200 hidden units are included in the multi-instance tracking evaluation. For the Kalman-filtering tracker, different minimum threshold values of its memory activation are used (See Section 4.2). The evaluated threshold values are 0.01, 0.5 and 1. The lower the minimum memory activation threshold of the Kalman-filtering tracker, the longer a vehicle instance remains tracked after no longer being detected, because the activation in memory for that instance decreases each time step until it reaches the threshold, at which point it is 'forgotten' by the tracker.

Chapter 6

Results

6.1 Instance Classification using DIF-Signatures

For the evaluation of instance classification using deep integrated feature signatures (DIFS) from YOLO, a KNN classifier was used to assess their intra-instance variations compared to their inter-instance differences (See Section 5.3). Table 6.1 shows the results of the DIFS instance classification evaluation. For both datasets, the highest classification accuracy was obtained with 1-nearest neighbor classification using Manhattan distance on the DIFS created by taking the activation from the middle of the YOLO network and using averaging over the whole detection region. This is 86.8% classification accuracy on the KITTI sample set, and 96.7% accuracy on the 4K dash-cam sample set.
On average across all conditions, the DIFS obtained from the 4K dash-cam data were attributed to the correct instance with 87.3% accuracy. For the DIFS from the KITTI data an average classification accuracy of 69.6% was obtained.
The average accuracy (across both datasets) for 1-nearest neighbor was 85.4%, which was higher than the average accuracy for 3-nearest neighbor classification at 77.3% and 5-nearest neighbor classification at 72.6%. For both datasets, the average accuracy was highest for 1-nearest neighbor classification and lowest for 5-nearest neighbor classification (See also Figure 6.1).
When comparing the results of DIFS extracted from different signature layers, we found that on average the 'early' DIFS were classified with 84.0% accuracy. In comparison the classification accuracy is 83.9% for the 'middle' DIFS, 80.0% for the 'late' DIFS and 65.9% for the 'last' DIFS. For the 4K dash-cam feed dataset, the highest average instance classification accuracy was obtained using the 'early' DIFS. The 'middle' DIFS resulted in the highest instance classification accuracy for the KITTI sample dataset. A breakdown of the classification accuracy by dataset and signature layer can be seen in Figure 6.2.
Lastly, the average classification accuracy when using the Manhattan distance metric is 78.7% compared to 78.2% when using Euclidean distance, and the average classification accuracy when averaging over the whole detection area is 79.0%, compared to 77.9% when splitting the detection area into four quadrants.

			1NN				3NN				5NN			
dataset	layer pos.	avg. meth.	Eucl.		Manh.		Eucl.		Manh.		Eucl.		Manh.	
			μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
KITTI	early	whole	84.4	4.0	85.3	2.9	72.1	5.5	73.5	3.1	64.1	8.5	65.3	7.8
KITTI	early	quadrants	83.2	4.6	85.3	4.2	72.1	2.8	75.0	3.4	67.1	7.4	67.9	7.5
KITTI	middle	whole	85.6	2.6	**86.8**	2.1	77.4	4.9	79.1	4.3	72.9	5.4	73.8	4.6
KITTI	middle	quadrants	82.6	4.5	84.1	4.3	72.4	2.2	71.8	2.2	65.6	3.2	66.2	2.1
KITTI	late	whole	81.2	3.2	81.8	3.0	69.4	5.7	70.6	3.7	65.0	4.8	63.2	4.0
KITTI	late	quadrants	79.1	3.8	79.4	4.5	70.0	2.0	69.7	3.0	61.2	7.5	61.5	7.3
KITTI	last	whole	62.1	3.4	59.7	4.4	47.9	2.7	50.9	2.9	50.3	3.5	50.0	4.0
KITTI	last	quadrants	66.8	3.0	67.4	4.5	57.1	3.4	56.8	4.9	53.8	4.9	52.9	6.5
4K	early	whole	95.6	0.6	95.6	1.1	92.9	1.3	93.7	1.1	91.3	1.1	92.1	0.5
4K	early	quadrants	96.1	1.6	96.6	1.3	93.1	0.6	94.1	1.1	88.7	1.5	91.0	1.9
4K	middle	whole	96.4	1.4	**96.7**	1.2	92.4	1.4	93.1	2.0	90.1	2.5	91.1	1.2
4K	middle	quadrants	94.3	2.3	95.0	2.9	87.3	2.8	88.9	2.1	83.9	3.9	85.1	3.5
4K	late	whole	94.6	1.9	94.7	1.6	90.9	2.5	90.7	2.7	87.0	1.5	86.7	1.8
4K	late	quadrants	92.4	3.3	93.0	3.2	86.9	3.3	87.7	2.6	80.9	3.6	82.6	2.7
4K	last	whole	85.1	3.6	84.1	3.9	74.6	4.4	73.0	3.0	70.1	4.0	67.9	3.3
4K	last	quadrants	84.6	4.7	84.0	4.3	73.7	4.4	73.4	5.0	67.0	4.2	67.7	3.2

Table 6.1: Accuracy (μ) and standard deviation (σ) of 5-fold KNN instance classification using YOLO DIFS. This table includes results for a value of 1, 3 and 5 for k using both Euclidean and Manhattan distance, of DIFS obtained from the four different signature layer positions ('layer pos.') with the two different averaging methods ('avg. meth.') on the 4K dash-cam feed and KITTI sample datasets. The highest obtained accuracy for both datasets is in bold.

6.2 Multi-Instance Tracking

The multi-instance tracking performance of the YOLO-Kalman and YOLO-LSTM trackers described in Chapter 4 is evaluated on the KITTI multi-object tracking benchmark dataset, and compared to the performance of plain YOLO v3 detection. The performance metrics for this evaluation were the recall, precision, F1 score, false alarm rate (FAR), Mostly-Tracked/Partly-Tracked/Mostly-Lost trajectories (MT/PT/ML) and Multi-Object Tracking Accuracy (MOTA). See Section 5.4 for an explanation of these metrics. The results of the multi-instance tracking evaluation can be seen in Table 6.2. The results include a parameter sweep of the YOLO confidence threshold for making a detection, YOLO-Kalman tracker memory activation minimum threshold for 'forgetting' an instance, as well as the number of hidden units in the LSTM networks of the YOLO-LSTM tracker (See Section 5.4).

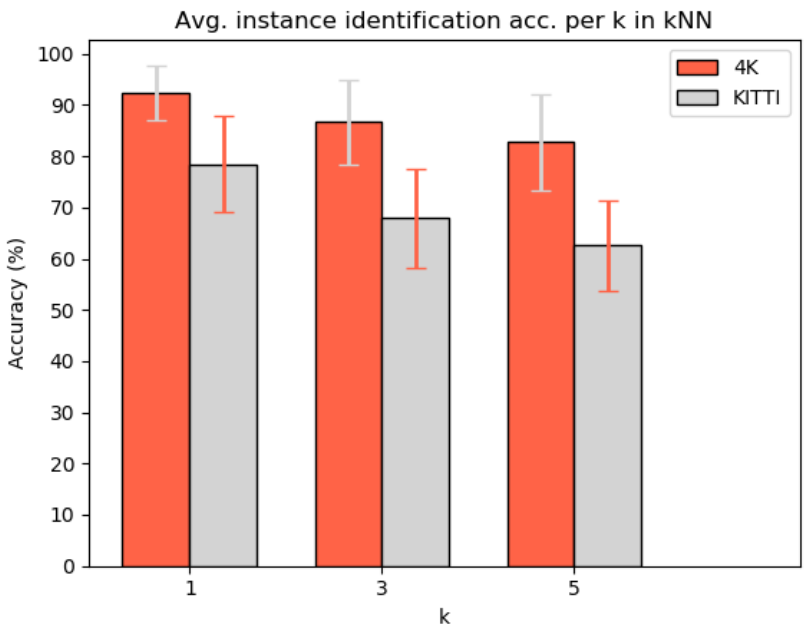


Figure 6.1: Average accuracy of the KNN instance classification of YOLO DIFS, per number of neighbors in the KNN classifier for the 4K dash-cam feed and KITTI sample datasets. Error bars of the bar graph show the standard deviation for each category.

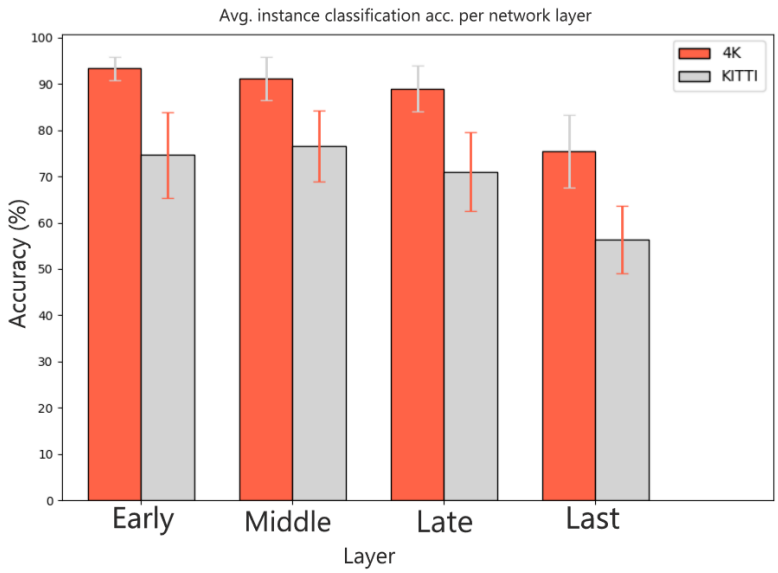


Figure 6.2: Average accuracy of the KNN instance classification of YOLO DIFS for the 4K dash-cam feed and KITTI sample datasets, per signature layer position where the deep integrated features originated. Error bars of the bar graph show the standard deviation for each category.

method	HU	Conf.	Mem.	Recall	Prec.	F1	FAR	MT	PT	ML	MOTA
plain YOLO	-	0.2	-	**63.7%**	85.4%	73.0%	38.8%	**27.3%**	56.6%	**16.1%**	45.5%
plain YOLO	-	0.4	-	61.1%	91.2%	**73.2%**	20.7%	23.7%	56.4%	19.9%	**48.9%**
plain YOLO	-	0.6	-	57.0%	94.2%	71.0%	12.1%	18.7%	57.3%	23.9%	47.7%
plain YOLO	-	0.8	-	49.7%	**97.1%**	65.7%	**5.1%**	13.5%	58.1%	28.4%	42.7%
YOLO-Kalman	-	0.2	0.01	**66.6%**	58.0%	62.0%	172.9%	**30.6%**	55.2%	**14.2%**	5.0%
YOLO-Kalman	-	0.4	0.01	64.2%	66.3%	65.2%	116.0%	26.5%	55.7%	17.8%	21.1%
YOLO-Kalman	-	0.6	0.01	60.3%	73.1%	66.1%	77.8%	23.0%	55.7%	21.3%	29.8%
YOLO-Kalman	-	0.8	0.01	54.1%	75.9%	63.1%	59.8%	15.9%	57.6%	26.5%	29.2%
YOLO-Kalman	-	0.2	0.5	66.3%	65.1%	65.7%	127.1%	30.3%	55.5%	**14.2%**	19.4%
YOLO-Kalman	-	0.4	0.5	64.1%	73.4%	68.4%	82.5%	26.5%	55.7%	17.8%	31.8%
YOLO-Kalman	-	0.6	0.5	60.2%	79.1%	68.4%	55.7%	23.0%	55.5%	21.6%	36.8%
YOLO-Kalman	-	0.8	0.5	54.0%	81.6%	65.0%	42.3%	15.6%	57.8%	26.5%	34.8%
YOLO-Kalman	-	0.2	1	66.3%	66.9%	66.6%	117.1%	30.1%	55.7%	**14.2%**	22.6%
YOLO-Kalman	-	0.4	1	64.0%	75.1%	**69.1%**	75.3%	26.5%	55.7%	17.8%	34.1%
YOLO-Kalman	-	0.6	1	60.2%	80.4%	68.8%	51.4%	23.0%	55.5%	21.6%	**38.2%**
YOLO-Kalman	-	0.8	1	54.0%	**83.0%**	65.4%	**38.5%**	15.6%	57.8%	26.5%	36.0%
YOLO-LSTM	50	0.2	-	16.6%	**84.2%**	27.7%	51.3%	1.1%	13.3%	85.6%	13.4%
YOLO-LSTM	50	0.4	-	26.8%	**84.2%**	40.7%	**50.7%**	2.0%	30.3%	67.7%	21.7%
YOLO-LSTM	50	0.6	-	40.5%	83.8%	54.6%	50.8%	4.3%	93.5%	2.1%	32.6%
YOLO-LSTM	50	0.8	-	41.7%	71.2%	52.6%	75.5%	6.0%	89.3%	4.7%	24.7%
YOLO-LSTM	100	0.2	-	18.0%	83.9%	29.7%	52.3%	3.1%	13.1%	83.7%	14.5%
YOLO-LSTM	100	0.4	-	31.5%	83.7%	45.7%	52.1%	10.4%	25.2%	64.4%	25.2%
YOLO-LSTM	100	0.6	-	49.0%	82.8%	61.6%	52.9%	24.5%	74.4%	**1.1%**	38.7%
YOLO-LSTM	100	0.8	-	54.3%	69.6%	61.0%	78.7%	33.2%	64.3%	2.6%	30.4%
YOLO-LSTM	200	0.2	-	20.7%	83.4%	33.2%	53.5%	4.5%	11.8%	83.7%	16.5%
YOLO-LSTM	200	0.4	-	35.4%	83.6%	49.7%	52.3%	11.2%	25.4%	63.4%	28.3%
YOLO-LSTM	200	0.6	-	**49.6%**	82.7%	**62.0%**	53.2%	**36.1%**	62.7%	1.3%	**39.1%**
YOLO-LSTM	200	0.8	-	47.0%	68.9%	55.9%	80.2%	26.0%	70.8%	3.2%	25.7%

Table 6.2: Results of multi-instance tracking based on YOLO detections using Kalman filtering and LSTM's, compared to the result of only using YOLO detections on the KITTI tracking benchmark. This table includes recall and precision metrics as well MT/PT/ML and CLEAR-MOT MOTA evaluation metrics explained in section 5.4. Also included are different values for the number of hidden units in the LSTM networks for YOLO-LSTM (HU), the YOLO confidence threshold (Conf.) and the memory activation threshold for the Kalman tracker (Mem.). Best performance for each tracker type on each metric is shown in bold.

Chapter 7

Discussion

7.1 Multi-Instance Tracking

When comparing the YOLO-Kalman and YOLO-LSTM trackers to the results of plain YOLO detection, we find the Kalman-filtering tracker provides an incremental increase in recall compared the plain YOLO detection (for example 60.2% compared to 57.0% for a confidence threshold of 0.6) whereas the YOLO-LSTM tracker (at best 49.6%) does not. Both the YOLO-Kalman and YOLO-LSTM tracker have a considerable drop in precision compared to plain YOLO (respectively 80.4% and 82.7% compared to 94.2% for a confidence threshold of 0.6).
When looking at the best F1 score, which is a measure which takes both recall and precision into account, the YOLO-LSTM tracker (62.0%) was not able to score as well as the plain YOLO (73.2%) or the YOLO-Kalman tracker (69.1%). Interestingly, the best result of YOLO-LSTM tracker is higher than the best result of the YOLO-Kalman tracker on the Mostly Tracked (36.1% vs 30.6%) and MOTA metrics (39.1% vs 38.2%). This is likely due to the fact that the YOLO-LSTM tracker had less ID-switches and fragmentations compared to the YOLO-Kalman tracker (see Table 6.2). Also worth noting are the high false alarm rates of both the YOLO-Kalman and YOLO-LSTM tracking systems. The main reason for the large false alarm rates for the Kalman-filtering tracker and LSTM-based trackers is a delay that occurs when the tracker still tries to track a previously tracked vehicle after it has disappeared from the view of the camera. The tracker keeps expecting a vehicle for a few frames after a vehicle was detected, creating what could be referred to as a 'safety' false positive. This is also supported by the fact that the false alarm rate seems to decrease for the YOLO-Kalman tracker when the minimum memory activation threshold before forgetting a tracked instance is higher. Which is to say, fewer false positives are generated when instances are stored in memory for a shorter number of time steps after they are no longer detected. Even though this quirk of the tracker of generating 'safety' false positives strongly impacts the score on the bench mark data, for real life applications this is not necessarily problematic, and may in some cases even be beneficial.
For both the YOLO-Kalman and YOLO-lstm tracker, as well as the plain YOLO detection the highest F1 score and MOTA was obtained with a YOLO confi-

dence threshold of 0.4 or 0.6 (See table 6.2). Based on these results it seems YOLO confidence threshold values between 0.4 and 0.6 strike a good balance between rejecting false detections and accepting true detections, at least when using an activation based dynamic memory system to filter out initial noise in the detection phase.
The low recall of the YOLO network compared to the ground truth seems to be caused in large part by the fact that YOLO has difficulty detecting vehicles which are farther away and appear smaller. This also explains the relatively low recall gain when adding a tracker, because most 'misses' occur before the tracking process can begin.
The YOLO-Kalman tracker needs two vehicle detections from the YOLO network at bare minimum to start making reliable inferences about future the positions and velocity the vehicle and the YOLO-LSTM tracker likely needs even more detections than that. On top of that, the trackers are sensitive to noise changes in position due to the jittery nature of the bounding boxes generated by YOLO. Especially at the start of a tracking sequence, because at that point the tracker predictions are based on fewer observations. It is likely that the tracking performance of the YOLO-Kalman and YOLO-LSTM trackers is heavily impacted by the relatively low YOLO detection recall. Previously, a Multi-Object Tracking Accuracy of 72.6% has been obtained on the KITTI dataset with a tracking method which incorporated Kalman filtering when reference detections were used (Choi, 2015).
The number of hidden units in the LSTM networks of the YOLO-LSTM tracker seems to be positively correlated with performance, when looking at general metrics which balance recall and precision like the F1 score, Most-Tracked trajectories and MOTA. Assuming that all LSTM networks of the multi-instance tracking evaluation converged to local optima, this seems to support the existing expert belief that larger neural networks have more local mimina with a low cost function value (Goodfellow et al., 2016). The tracking precision did decrease slightly for LSTM networks with more hidden units.
The YOLO-LSTM tracker was able to perform comparable to the YOLO-Kalman tracker on the training set, scoring slightly higher than the YOLO-Kalman tracker on the MOTA and Mostly Tracked metrics, while scoring slightly lower when looking at recall and precision.
However when compared to the YOLO-Kalman tracker, exploratory research indicated that the LSTM-based tracker seemed to generalize very poorly to the new unlabeled dash-cam data. We suspect the reason for this could be that the LSTM networks overfit to qualities of the bounding boxes that are unique to the training data such as the frame rate and the aspect ratio of the training data video feed, whereas the Kalman filtering models a linear relationship between subsequent bounding boxes which is more invariant to such constants. We therefore expect Kalman-filtering trackers to be more generalizable than the LSTM-based trackers.

7.2 Instance Classification using DIF-Signatures

When looking at the results from the evaluation of instance classification using KNN on deep integrated features signatures obtained from the YOLO v3 feature maps, we see that the highest obtained result is 96.7% accuracy (see Table 6.1).

In comparison, when using the often used SIFT key-point matching technique (See (Lowe et al., 1999)) only 6.7% classification accuracy was obtained on the 4K dash-cam video dataset, and only 5.9% classification accuracy was obtained on the KITTI dataset. Our initial conclusion is that instance classification using DIF-signatures from the YOLO feature maps highly successful, especially considering the fact that these feature maps are generated by the YOLO network during the detection pass, and the extra computational cost of generating the instance classification DIFS is therefore negligible. If we compare the different datasets, we find that while 4K dash-cam dataset could be classified correctly at 96.7% accuracy, for instances from the KITTI dataset the highest accuracy was 86.8%. Possible explanations for this, include the difference in aspect ratio and frame rate between these two data sets. Due to the lower frame rate (10 frames/second) of the KITTI data, the similarity between images in consecutive time steps is lower than with the 4K dash-cam data (30 frames/second). Due to larger difference between width and height in the KITTI images, the images require more padding to fit the square sized requirement of YOLO, which reduces the area of relevant information of the KITTI images after preprocessing more than for the 4K dash-cam images.

An interesting result to note is that 1NN classification is the most successful and the KNN classification accuracy decreases with the number of neighbors used. This indicates that the similarity of the DIFS is highly localized and not even between different examples of the same instance. This also makes sense when considering time dependence of the data and the perspective change of a tracked vehicle over time during a typical sequence. It is possible that variable qualities of a detection such as size and orientation have a large influence on the on the internal activation patterns of the YOLO network, which would mean that only the DIFS of instance examples which are adjacent on the temporal scale are reliable predictors for instance classification.

It seems that with regard to the layer from which the signature is taken, the instance classification potential is high for a large part of the network, but decreases for layers deeper in the network than the 'middle' layer, especially for the KITTI dataset (see Table 6.1). For either dataset, the 'last' layer leads to noticeably decreased results (See Figure 6.2), and should likely not be used for instance classification. A possible explanation for this is the fact that the layers used for the 'last' DIFS are directly connected to the final YOLO output layers, which could mean that their activation is already highly class based.

Based on the obtained results, the choice of distance measure for the KNN classifier seemed to be of little influence. It also seems not to make a noticeable difference to the classification accuracy whether the DIFS are created by averaging the activation across the entire detection region, or by averaging over the four quadrants. If performance is equal, the 'whole' averaging method is preferable to the 'quadrants' averaging method for efficiency reasons, because the resulting signature vectors are four times smaller.

7.3 Future Work

7.3.1 Instance Classification using DIF-Signatures

In this study three layers are used to create the deep integrated feature signatures and all DIFS are extracted from a single layer which is determined based on the output layer where the traffic participant is detected. This method of DIF-signature generation was developed experimentally and therefore remains an area of further experimentation. For instance, it is also possible to construct a signature using feature maps from multiple layers. This could be done either by using multiple layers per scale, or by using all three of the current layers regardless in which output layer the detection was made. DIFS are taken from the region in the feature maps corresponding to the region in the image where a traffic participant is detected. The effects of making the region in the feature maps used to generate the signatures larger or smaller have not yet been investigated.

In this research, instance classification based on the signature vectors was done with the KNN algorithm using Manhattan and Euclidean distance. A different classifier or similarity metric could potentially be used to distinguish the DIFS of different vehicle instances even better.

In the context of situational awareness and multi-vehicle tracking in traffic, a pertinent topic of further research is the question of how the similarity of DIFS of objects in subsequent detection measurements relates to existing consistency heuristics such as the bounding box overlap. Signature dissimilarity could play a vital role in edge case situations where the overlap heuristic fails, such as cases where one neighboring vehicle is overtaken by another. However more research is necessary in order to determine the thresholds of relative DIFS dissimilarity which could be used to determine distinct instances with certainty.

Finally, when placing this research in the context of the scientific field of deep learning, the performance of DIF-Signatures must be evaluated against other promising feature representations which could be used for the instance classification solutions in the same domain. For example, a comparison study of instance classification accuracy and feature extraction time could be made between DIFS and features from an end-to-end feature extraction CNN like the DeepID2 features of (Sun et al., 2014).

Another interesting topic for future research would be to see to what degree this ability of identification of instances based on feature map signatures generalizes to other deep neural architectures used for multi-class classification such as ResNet (K. He et al., 2016) or SqueezeNet (Iandola et al., 2016). The creation of identifying signatures from the YOLO feature maps allows us to access feature information stored in the YOLO network and utilize this information for a purpose for which the network was never explicitly trained. A better understanding of the circumstances or network properties which allow us to successfully create these deep integrated feature signatures could potentially aid in opening the 'black box' of deep neural networks (Shwartz-Ziv & Tishby, 2017), by creating a cohesive understanding of the feature information stored in deep CNN's.

7.3.2 Multi-Instance Tracking

With the limited amount of labeled vehicle trajectory data available to us, the trained LSTM networks were not able to reliably predict vehicle position estimates. A number of different parameters and hyperparameters have been tested to find the best LSTM network, including different numbers of hidden units, loss functions, and learning rates. Of course, more thorough parameter sweeps are always possible in this regard. However, we can cautiously conclude from this research that LSTM's networks tend to overfit on the exact training data trajectories, and that they were too complex to model the underlying physical speed and acceleration patterns of moving vehicles in traffic.

On the other hand, based on the results of the Kalman-filtering tracker we can also see that these patterns were not able to be modeled by position and velocity based Kalman filtering, at least when based on observations from the pretrained YOLO v3 network.

Kalman filtering has previously been successfully used to track vehicle positions obtained from a static traffic surveillance camera (Melo, Naftel, Bernardino, & Santos-Victor, 2004). However, tracking vehicles from a vehicle mounted dashcam may be more complicated mathematically. There are two different optical flows involved, firstly from the movement of the camera vehicle, and secondly from the relative movements of the other traffic participants.

It is possible that LSTM-based solution for multi-instance tracking could be realized by forcing the networks to be closer to linear with the use of strict regularization techniques, possibly with the use of residual blocks.

It is also worth investigating the potential of solutions previously used in offline vehicle tracking in combination with the YOLO detection network, such as the Poisson multi-Bernoulli mixture filter used in (Benjaminsson & Rosenberg, n.d.), or the aggregated flow descriptor used in (Choi, 2015).

Lastly, considering the success of the instance classification using DIF-signatures, another solution for multi-instance tracking could be to integrate signature information directly into the tracking procedure. This could potentially be achieved by looking at different candidate patches in line with the detected trajectory of the tracked instance, to find the bounding box with the closest matching signature to this instance. However, a model to generate these candidate patches would still be needed.

Another possibility for potential improvement in the multi-instance tracking system is to adapt the detection network. In (Molchanov et al., 2017) it was reported that for the purpose of pedestrian detection using the YOLO network, small close objects were better able to be distinguished when the YOLO architecture was altered. Most vehicle trajectory 'misses' in this research occur when vehicles are farther away, before they can be instantiated in the tracker. We expect the recall performance to increase if vehicles were detected earlier, and if there were more network detections in general, and fewer 'gaps' which need to be filled in with Kalman filtering mechanism.

For this research, we decided to use a pretrained YOLO v3 network, trained on 80 classes. One reason for this was the claim of YOLO's generalizability made in (Redmon et al., 2016). It is possible that although YOLO may generalize relatively well to some domains, the number of vehicle detections would still be higher if the network was specialized in detection vehicles instead of general objects. The best way to evaluate this is to train a YOLO v3 network on only

a select number of traffic related object classes and observe whether this boosts the number of initial vehicle detections of the tracker on the same datasets.
At the time this research started, the YOLO v3 architecture seemed like the go-to choice for video-based object detection based on both its performance and speed. However, deep neural object detection is still a developing field and alternative deep neural object detection networks could also be looked into further.

7.4 Conclusion

In this study we sought to understand how the state-of-the-art of multi-object detection could best be used in combination with instance classification and multi-instance tracking techniques in the context of furthering situational awareness of self-driving vehicles.
Instance classification of vehicles using signatures obtained from the feature maps of the deep neural YOLO network seems to be very successful. The highest results for instance classification were obtained when the signatures were created from feature maps in the beginning or the middle of the YOLO network and when using a 1-nearest neighbor classifier. The accuracy of instance classification using deep integrated feature signatures from YOLO feature maps also increased when the quality of the input video-stream data was higher.
Multi-instance tracking of surrounding vehicles from the perspective of a camera vehicle remains an open challenge. A sophisticated multi-instance solution could potentially be developed. Avenues to be explored in this light include integrating instance DIF-signature information over a time window and regularizing LSTM's. However, it is also possible that as technology develops more powerful hardware enables the creating of more sophisticated detection networks which will make relatively simple tracking models more viable, as accurate detection is currently one of the bottlenecks for successful tracking.
The KNN instance classification based on signatures and the multi-instance tracking both rely on the same pretrained YOLO network. The difference between the high accuracy obtained during the former and relative challenge of the latter indicates that the generalizability of the YOLO network differs between applications. The abstract representation of classes and objects of interest in the network may be generalizable accross different data domains, while the final detection output may not be. At least not generalizable enough to compete with state-of-the-art networks trained on specific domain data.
In summary, deep integrated feature signatures created from the feature maps of the YOLO neural network trained for detection and multi-class classification have been shown to be very promising for the purpose of instance classification. This provides us with a step towards realizing situational awareness of autonomous vehicles about surrounding participants in the dynamic environment of everyday traffic as well as potentially leading to the better understanding of the inner workings of deep neural networks in general.

Bibliography

Bay, H., Ess, A., Tuytelaars, T., & Van Gool, L. (2008). Speeded-up robust features (SURF). *Computer vision and image understanding*, *110*(3), 346–359.

Benjaminsson, J., & Rosenberg, E. (n.d.). Multiple object tracking with camera data and deep learning detectors.

Bernardin, K., & Stiefelhagen, R. (2008). Evaluating multiple object tracking performance: the CLEAR MOT metrics. *Journal on Image and Video Processing*, *2008*, 1.

Boufenar, C., Kerboua, A., & Batouche, M. (2018). Investigation on deep learning for off-line handwritten arabic character recognition. *Cognitive Systems Research*, *50*, 180–195.

Buehler, M., Iagnemma, K., & Singh, S. (2007). *The 2005 DARPA grand challenge: the great robot race* (Vol. 36). Springer.

Buehler, M., Iagnemma, K., & Singh, S. (2009). *The DARPA urban challenge: autonomous vehicles in city traffic* (Vol. 56). springer.

Chan, F.-H., Chen, Y.-T., Xiang, Y., & Sun, M. (2016). Anticipating accidents in dashcam videos. In *Proceedings of the asian conference on computer vision* (pp. 136–153).

Choi, W. (2015). Near-online multi-target tracking with aggregated local flow descriptor. In *Proceedings of the ieee international conference on computer vision* (pp. 3029–3037).

Danelljan, M., Robinson, A., Khan, F. S., & Felsberg, M. (2016). Beyond correlation filters: Learning continuous convolution operators for visual tracking. In *European conference on computer vision* (pp. 472–488).

Dickmanns, E. D., Behringer, R., Dickmanns, D., Hildebrandt, T., Maurer, M., Thomanek, F., & Schiehlen, J. (1994). The seeing passenger car 'VaMoRs-P'. In *Proceedings of the intelligent vehicles' 94 symposium* (pp. 68–73).

Ding, G., Chen, W., Zhao, S., Han, J., & Liu, Q. (2018). Real-time scalable visual tracking via quadrangle kernelized correlation filters. *IEEE Transactions on Intelligent Transportation Systems*, *19*(1), 140–150.

Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., & Darrell, T. (2014). Decaf: A deep convolutional activation feature for generic visual recognition. In *Proceedings of the international conference on machine learning* (pp. 647–655).

Endsley, M. R. (1995). Measurement of situation awareness in dynamic systems. *Human factors*, *37*(1), 65–84.

Frossard, D., & Urtasun, R. (2018). End-to-end learning of multi-sensor 3d tracking by detection. In *2018 IEEE international conference on robotics*

and automation (icra) (pp. 635–642).

Geiger, A., Lenz, P., Stiller, C., & Urtasun, R. (2013). Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press. (`http://www.deeplearningbook.org`)

Guidolini, R., Scart, L. G., Jesus, L. F., Cardoso, V. B., Badue, C., & Oliveira-Santos, T. (2018). Handling pedestrians in crosswalks using deep neural networks in the IARA autonomous car. In *2018 international joint conference on neural networks (ijcnn)* (pp. 1–8).

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770–778).

He, X., & Zeng, D. (2017). Real-time pedestrian warning system on highway using deep learning methods. In *2017 international symposium on intelligent signal processing and communication systems (ISPACS)* (pp. 701–706).

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735–1780.

Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., & Keutzer, K. (2016). Squeezenet: Alexnet-level accuracy with 50x fewer parameters and$<$ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*.

international, S. (2016). Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles. *SAE International,(J3016)*.

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, *82*(1), 35–45.

Kathuria, A. (2018). *What is new in YOLO-v3.* `https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b`.

Ketkar, N. (2017). Introduction to pytorch. In *Deep learning with python* (pp. 195–208). Springer.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

LeCun, Y., Bengio, Y., et al. (1995). Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, *3361*(10), 1995.

Li, Y., Huang, C., & Nevatia, R. (2009). Learning to associate: Hybridboosted multi-target tracker for crowded scene.

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., ... Zitnick, C. L. (2014). Microsoft coco: Common objects in context. In *European conference on computer vision* (pp. 740–755).

Lowe, D. G., et al. (1999). Object recognition from local scale-invariant features. In *Proceedings of the international conference on computer vision* (Vol. 99, pp. 1150–1157).

McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, *5*(4), 115–133.

Melo, J., Naftel, A., Bernardino, A., & Santos-Victor, J. (2004). Viewpoint independent detection of vehicle trajectories and lane geometry from uncalibrated traffic surveillance cameras. In *Proceedings of the international conference image analysis and recognition* (pp. 454–462).

Molchanov, V., Vishnyakov, B., Vizilter, Y., Vishnyakova, O., & Knyaz, V. (2017). Pedestrian detection in video surveillance using fully convolutional YOLO neural network. In *Automated visual inspection and machine vision II* (Vol. 10334, p. 103340Q).

Pan, S. J., & Yang, Q. (2009). A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, *22*(10), 1345–1359.

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 779–788).

Redmon, J., & Farhadi, A. (2017). YOLO9000: better, faster, stronger. *arXiv preprint*.

Redmon, J., & Farhadi, A. (2018). YOLO-v3: An incremental improvement. *arXiv preprint arXiv:1804.02767*.

Ren, H.-G., Liu, W.-M., Shi, T., & Li, F.-J. (2017). Compressive tracking based on online hough forest. *International Journal of Automation and Computing*, *14*(4), 396–406.

Scheidegger, S., Benjaminsson, J., Rosenberg, E., Krishnan, A., & Granstrom, K. (2018). Mono-camera 3d multi-object tracking using deep learning detections and pmbm filtering. *arXiv preprint arXiv:1802.09975*.

Shwartz-Ziv, R., & Tishby, N. (2017). Opening the black box of deep neural networks via information. *arXiv preprint arXiv:1703.00810*.

Sun, Y., Chen, Y., Wang, X., & Tang, X. (2014). Deep learning face representation by joint identification-verification. In *Advances in neural information processing systems* (pp. 1988–1996).

Trebatickỳ, P. (2005). Recurrent neural network training with the extended Kalman filter. In *Proceedings of the iit. src 2005: Student research conference* (p. 57).

Tsugawa, S., Yatabe, T., Hirose, T., & Matsumoto, S. (1979). An automobile with artificial intelligence. In *Proceedings of the 6th international joint conference on artificial intelligence-volume 2* (pp. 893–895).

Vatavu, A., Danescu, R., & Nedevschi, S. (2015). Stereovision-based multiple object tracking in traffic scenarios using free-form obstacle delimiters and particle filters. *IEEE Trans. Intelligent Transportation Systems*, *16*(1), 498–511.

Wang, Z. (2015). The applications of deep learning on traffic identification. *BlackHat USA*, *24*.