Vehicle Collision Detection based on Synthetic Data using Deep Learning

Master of Science Thesis
University of Turku
Department of Computing
2021
Timo Jokela

TURUN YLIOPISTO
Tietotekniikan laitos

TIMO JOKELA: Vehicle Collision Detection based on Synthetic Data using Deep Learning

Pro gradu -tutkielma, 66 s.
Tekoäly ja data-analytiikka
5 2021

Konenäkö ja visuaalista dataa käsittelevät koneoppimismenetelmät ovat kehittyneet merkittävästi kuluneen vuosikymmenen aikana. Tämä edistys on myös näkynyt myös niin kutsuttujen autonomisten ajoneuvojen tuotekehityksessä, eli kehitettäessä liikennevälineitä, jotka kykenevät toimimaan liikenteessä itsenäisesti.
Eräs merkittävä hidaste mille tahansa koneoppimisen sovellutukselle on saatavilla olevan laadukkaan datan määrä. Datalla tarkoitetaan sitä tietoaineistoa, jonka avulla koneoppimisen mallit oppivat uusia taitoja. Laadukkaan aineiston puute on usein merkittävin este, jonka moni koneoppimiseen liittyvä projekti kohtaa.
Autonomisista ajoneuvoista sekä yleisesti liikenteestä puhuttaessa edellä mainittu koskee erityisesti onnettomuuksia, joista ei juurikaan ole tarjolla yhdenmukaista ja hyvälaatuista dataa julkista käyttöä ja tutkimusta varten. Tämä opinnäytetyö esittelee ratkaisun, jossa todellinen onnettomuusdata korvataan videopeliympäristössä luodulla datalla.
Tutkielmassa esiteltävä ratkaisu kykenee oppimaan törmäyksen tunnistuksen keinotekoisesta datasta ja sen jälkeen soveltamaan opittua tietoa todellisiin törmäyksiin. Tutkielman ratkaisu koostuu kolmesta erillisestä osa-alueesta. Kaksi ensimmäistä osa-aluetta ovat kohteiden tunnistaminen ja seuranta, joiden avulla jäljitetään videoaineistossa liikkuvia ajoneuvoja. Tiedot liikkuvista ajoneuvoista siirretään törmäyksiä tunnistavalle mallille, joka pyrkii päättelemään, liikkuuko seurattava ajoneuvo normaalisti vai onko se osallisena törmäyksessä.
Mallinnukset tuottavat lupaavia tuloksia, mutta yhteys keinotekoisen ja todellisen datan välillä jää osin vaillinaiseksi. Esitetty törmäyksiä tunnistava malli onnistuu hienoisesti parantamaan tuloksia verrattuna triviaaliin vertailukohtamalliin. Keinotekoinen data ei kuitenkaan täysin vastaa todellisia törmäyksiä, mistä johtuen mallin on erittäin vaikea tunnistaa joitakin törmäystilanteita täsmällisesti.

Asiasanat: koneoppiminen, konenäkö, kohteiden tunnistus, törmäysten tunnistus, kohteiden seuranta, konvolutionaaliset neuroverkot

UNIVERSITY OF TURKU
Department of Computing

Timo Jokela: Vehicle Collision Detection based on Synthetic Data using Deep Learning

Master of Science Thesis, 66 p.
Artificial intelligence and data analytics
May 2021

Computer vision and deep learning methods that process visual data have considerably improved during the last decade. This progress has also affected the development of so-called autonomous vehicles, which are able to act independently in the traffic.
One notable hindrance facing any deep learning application is the amount of quality data that is available. Data means the corpus of information from which the models learn new skills. Lack of good data is often the most significant hurdle a deep learning project faces.
When considering autonomous vehicles and traffic generally, this problem is particularly evident in a collision context, as there is very little accident data available for public use and research, particularly when the data should be both consistent and of good quality. This thesis presents a solution in which real data is substituted with data that is generated in a video game environment.
The solution proposed in this thesis can learn collision detection by looking at the synthetic data and then apply the learned information in detecting real collisions. The presented solution consists of three phases. The first two phases are object detection and object tracking which are used to identify and follow vehicles moving the video footage using deep learning. Information obtained in these phases is then transferred to the third phase, a collision detector, which attempts to infer if the tracked vehicle is moving normally or if it is participating in a collision.
Initial results indicate a promising although limited connection between synthetic and real-world data, and the proposed model is able to slightly surpass the performance of a trivial baseline. However, the generated synthetic training data is not entirely representative of its real-world counterpart, which results in some of the collision events being very difficult to detect properly.

Keywords: deep learning, computer vision, object detection, collision detection, object tracking, convolutional neural network

Contents

1 Introduction

A textbook example of a deep learning problem is one that has a decent amount of good quality data with which to train whatever model is suitable for tackling the problem. Real-life problems, on the other hand, are often very different from the ones in the examples and tutorials, and this is particularly true for traffic accidents.

Accidents, unlike many other events in a machine learning context, are hopefully observed as rarely as possible. This means that we have less opportunity to obtain well-rounded accident data and therefore less opportunity to train better models that would have a better understanding of accidents. Intentionally creating more or better accident footage could be an expensive undertaking in any controlled environment, or a very questionable one in a non-controlled environment, and possibly one not well-received by the public.

The purpose of this thesis is to investigate the idea of substituting real data with synthetic data in order to overcome the hurdle. More specifically, this means that a virtual environment is used to generate proper synthetic data for training a deep learning model for vehicle collision detection. A virtual environment, if it is an adequately accurate representation of the real world, could offer a suitable alternative to any approach that requires real-world data.

The idea of using a virtual environment in a traffic-related context is not unheard of. One such environment, known commercially by the name AILiveSim, is a virtual environment for training autonomous vehicles, developed on top of a video game engine known as Unreal Engine [1]. It can be considered as a heavyweight tool that incorporates many features essential to autonomous traffic, for example, simulation of multiple sensor types.

However, occasionally one can make do with less. The approach presented in this thesis is to generate synthetic data in an environment that is inexpensive and readily available, namely a video game known as Grand Theft Auto V (GTA V). The attempted approach relies solely on visual data and therefore many of the features offered in specially tailored products are not needed.

For years video games have been pushing the envelope for better and more realistic visuals. Apart from possible artistic choices, many games of the last decade can be said to have very believable graphics. GTA V, even though originally published in 2013 and thus already several years old, continues to look reasonably modern on contemporary systems with capable hardware. A screenshot of the game is shown in Figure 1.1.

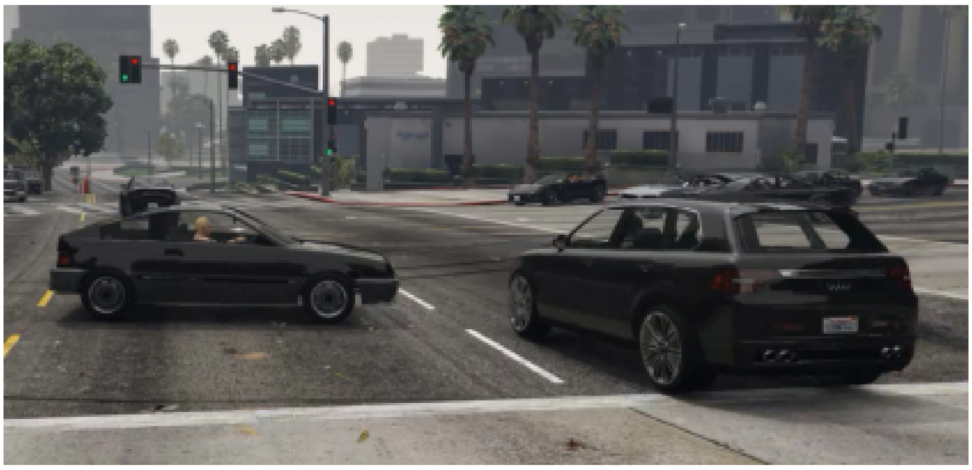


Figure 1.1: A screenshot of Grand Theft Auto V.

In addition to suitable visuals, the game has other perks such as reasonable price and easy availability, rich traffic system with concurrently moving vehicles, and an active modding community. The term *modding* refers to the act of enthusiasts modifying or expanding a game beyond its original confines. In the case of GTA

V modding has resulted in a programming library Script Hook V [2] which makes it possible to script events in the game, thus allowing for traffic accidents to be programmatically caused and recorded.

1.1 Research questions and objectives

This research is centered around two connected research questions:

1. Are modern superficially lifelike virtual environments visually and physically accurate enough to be utilized in developing machine learning models that can detect vehicle collisions happening in reality?

2. Are there specific techniques that translate well between real and virtual environments in a vehicle collision context and how do these techniques perform when evaluated using real collision data?

To further elaborate on the first research question, it should be mentioned that focusing solely on the visual similarity of a video game and the real world is most probably not enough for accurate real-life collision detection results. In addition, the collision model of the game should also be a sufficiently accurate representation of its physical counterpart. Typical modern video games focused on driving generally incorporate believable collision models, as aspects like vehicle handling and driving physics are constant topics in video game reviews and unusual or awkward physics could easily drive players off the game. It remains to be seen if this superficial resemblance is enough to result in good real-world performance of the developed model.

The objectives of this research can be stated as follows:

1. Identify techniques of contemporary computer vision, initially designed for and trained on real-life scenarios, which perform well when applied to data obtained from traffic scenarios in a computer game.

2. By utilizing these techniques to devise a model which, when trained on synthetic data, can successfully predict real-life vehicle collisions happening in video footage.

Thus, the research hypothesis for this thesis is: when trained on synthetic collision footage and tested on real-life footage, the performance of a suitable model surpasses that of a trivial baseline. (The baseline is presented in section 3.1.3.)

1.2 Research background and methodology

The training footage will be subjected to various computer vision techniques including but not limited to object detection and object tracking. Even though these methods as such have very little to do with collisions, it is nevertheless vital to identify all vehicle instances participating in the crash. Therefore, all other applied techniques will be constrained by the detected vehicles, as it is only useful to consider collision detection in such areas of a video frame.

The framework will consist of three separate and consecutive phases:

1. Object detector detects vehicles in the video frames.
2. Object tracker tracks the cars across the video.
3. Collision detector attempts to detect collisions based on data received from the previous phases.

The first two phases utilize established methods that have been proven to work. The collision detector model will be developed during this research. As the video clips are annotated the approach falls into the category of supervised learning.

A collision is neither a one-frame event in a video clip nor an infinitesimally short instant of time in the real world. Therefore, the model in the third phase should be one with a capability to understand sequential data (i.e., how the collision progresses through time). For this reason, visual processing of the data will be coupled with techniques that can understand the temporal aspect of the phenomenon.

The research methodology can be summarized as follows:

1. Using a virtual environment to collect synthetic collision data.
2. Using deep learning methods for computer vision tasks such as object detection and tracking and creating a framework that can understand the visual aspect of the data.
3. Using temporal methods to allow the framework to handle sequential data.
4. Using common performance metrics to evaluate the proposed framework for collision detection on real-life data.

1.3 Research motivation

This research assumes the point of view of an observer moving in the traffic flow. That is, collisions are detected much like a driver of a car would detect them. The observer, however, is not assumed to be a participant of the collision. In other words, the vehicle that detects the collision is not one of the colliding vehicles.

This is the first motivating element. There is much research on collision detection, but due to lack of suitable data many papers focus on the point of view of a stationary observer, for example a surveillance camera. This is further discussed in section 1.4.

An example situation illustrates the idea behind the viewpoint of this thesis: Assume an autonomous vehicle is approaching a busy intersection. It detects a green traffic light and knows it is permitted to proceed through the intersection. Another car in front it is traveling in the same direction and has already reached the center of the intersection. However, almost instantaneously another vehicle approaching from the right disobeys the red light and speeds through the intersection, crashing into the car in front of the autonomous vehicle. If the autonomous vehicle does not understand a collision when it sees one it could easily end up as a participant in the crash.

This is the second motivating element. The better an autonomous vehicle is in detecting collisions, the better it is at avoiding the. A vehicle that detects collisions sooner than later makes traffic safer for all users of the road. It can take needed measures to avoid the collision and it can alert the officials about what has happened.

1.4 Related literature

Over the past years, several different methods have been applied to the problem of collision detection. For example, in [3] the authors gathered a small set of surveillance videos off the internet. These videos, recorded by stationary observers, were then processed by a pipeline that first detected and extracted vehicles and their bounding boxes from the frames, after which the authors applied a method consisting of Violent Flow descriptors and Support Vector Machines to detect crashes in the videos. A similar data set was used in [4] as the authors obtained a set of closed-circuit television (CCTV) surveillance videos recorded in the city of Hyderabad in India, after which deep representations of the events were extracted by using denoising autoencoders which were trained with non-collision traffic videos.

In [5] a set of traffic accident videos collected from the police officials in China were used in training a crash detection method that is similar to the detector presented in thesis in the sense that both of the methods utilize an image classifier network to extract crash-related appearance features, which in turn are processed by a spatiotemporal model capable of handling both types of data.

A different approach was taken in [6], which is an attempt to provide a novel dataset suitable for analysing traffic accidents. The videos were downloaded from the YouTube online video service and the resulting set consists of CCTV-recorded accident 230 videos along with annotations.

One publication this thesis is particularly indebted to is [7] in which accident footage was used, not for collision detection but for collision prediction purposes by utilizing what the author call dynamic-spatial-attention Recurrent Neural Network. As a by-product, the team published an accident data set [8] consisting of footage recorded in six major cities in Taiwan, and it is this data this paper also relies on when attempting to establish a connection between virtual and real collisions.

All of the aforementioned research highlights one or more of the problems related particularly to data involving traffic accidents:

- The availability of large, consistent and good-quality datasets is extremely weak. The videos are downloaded one-by-one from various places all over the internet or they are obtained from a non-public source (e.g., local officials) not available to the research community in general.
- Many of used videos are recorded by stationary surveillance cameras, which means the visual information in the videos is very different from what an autonomous car moving on a road observes. Some research uses onboard videos such as dashcam recordings, but the quality of such footage varies considerably as a result of there being many different kinds of dashboard cameras.
- Many of the datasets are heavily characteristic of a specific geographical region and may contain types of traffic or behavior rarely encountered elsewhere.

All these observations underline the difficulty of evaluating the methods published in the studies: when the data is not shared, it can be impossible to accurately compare new research with what has been published before. This problem is more pronounced because any phenomenon involving the possibility for a loss of life should be measured as unambiguously as possible.

Resorting to synthetic data is definitely not a silver bullet solution capable of fixing all the mentioned problems, but it can help alleviate some of the issues. In a sufficiently complex virtual environment, it is possible to create virtual data that is consistent and versatile. A virtual environment can also be suited to represent any traffic culture anywhere in the world.

The use of synthetic data generally in traffic-related models is not unheard of. In [9] the authors generated a virtual collision data set consisting of crashes involving the observer and another vehicle in order to train a model to identify vehicles that can be considered to move dangerously with respect to the observer. The virtual environment is in fact the same one as used in this thesis, Grand Theft Auto V. This is a promising indication of the possible suitability of GTA V data for collision detection purposes. In scripting the accidents, the authors relied on the same library that was used in this thesis, Script Hook V [2]. This is, in fact a more credible approach than one might initially think: a Google search with the string *GTA V autonomous cars* shows that this is not the first time this possibility has been considered.

The research of [9] is particularly interesting in its approach as it extends models trained using synthetic data with other methods. Specifically, the authors noticed that synthetic data alone is noisy and does not offer an adequately accurate representation of the way a real driver would behave in a situation in which a collision is imminent. This is very much in line with the observations made during the implementation phase of this thesis: the artificial intelligence drivers of GTA V do not (and most likely are not intended to) behave exactly the way real human drivers do; they are different in many subtle and sometimes strikingly different ways, and naturally lack the complexity of their human counterparts. The authors state that the behavior was due to their implementation of the driving algorithm, but the observations during the implementation of this thesis indicated that it is very difficult to fully script away all the AI functionalities. However, some difference should also be accounted to what was scripted: in the mentioned study the observer was a participant in the crash, whereas in the context of this thesis the observer is merely an observer. This most likely explains some of the difference in the behavior of the drivers.

This is not a problem in the context of this thesis, as collisions can be modeled even if the preceding behavior could not be considered believable. However, for [9] this posed a more serious issue as the path of the dangerous vehicle simulated by the synthetically trained model was not a very accurate representation of the paths real drivers would take. The authors came up with an insightful method that couples the synthetically created labels of dangerous and non-dangerous moving vehicels with a real-world path prediction model in order to reduce the bias of skewed driving behavior present in the synthetic data.

This is indeed an interesting avenue of future research, where synthetically created data is acknowledged as flawed, but can be corrected with approriate real-world data. It should be considered how a similar approach could be integrated into any framework that attempts to detect, predict or avoid an imminent traffic accident. After all, any solution that is able to offer even the slightest improvement in traffic security is worth exploring.

1.5 Thesis organization

This thesis is divided into five chapters as follows:

- Chapter 1 gives an overview of the idea and the motivation behind this thesis, along with a brief glance at related literature and research

- Chapter 2 presents the related background information for two of the main tasks proposed in the thesis: object detection and object tracking.
- Chapter 3 is dedicated to the structure of the implemented collision detection framework.
- Chapter 4 presents the obtained collision detection results.
- Chapter 5 gives a summary of the topic and offers the essential conclusions that have been reached during the thesis.

2 Background

This chapter introduces the methods and concepts on which the proposed framework is built during chapter 3. The discussion starts with an overview of concepts related to some of the standard methods in deep learning: convolutional and recurrent neural networks. The two later sections focus on object detection and object tracking methods.

2.1 Convolutional Neural Networks

The last decade has marked unforeseen progress in tackling many of the problems related to computer vision. If there is one method this success can be attributed to, it is without a doubt a form of neural network known as Convolutional Neural Network (CNN). It lies in the heart of many ground-breaking techniques which have pushed the boundaries of what is possible in tasks including but not limited to image classification and object detection.

CNNs are neural networks that have evolved from earlier concepts inspired by the structure of the visual cortex [10]. CNNs consist of several layers of different functionality which can be stacked in a consecutive fashion. When the the number of consecutive layers is very high the network is typically called a *deep* CNN. The following section provides a brief summary of these layers.

2.1.1 Layers of a CNN

This section presents the typical layers used in CNNs. The first two layer types, a convolutional layer and a pooling layer, are the core building blocks of a CNN. The last subsection discusses some of the methods used for regularizing CNNs.

Convolutional layer

The core building block of a CNN is a layer known as the convolutional layer. The basic principle of the layer is to perform filtering operations on the areas of the received input. (Even though the name of the layer suggests a mathematical operation of convolution, the actual operation is in fact cross-correlation [10].) The purpose of a filter is to capture some feature it has learned during training and to produce a feature map of this feature as an output. In other words, the filter traverses the input and outputs a map of all the places where a feature detected by the filter is present. Each element in a feature map is only connected to a small segment of the input known as the receptive field.

The shift from one receptive field to the following one is a configurable parameter of the layer. This parameter is called the stride. A stride larger than 1 would mean the size of the feature map would be reduced along with the computational complexity of the model. It is also possible for the feature map to be smaller in size than the original input even if the stride equals 1. This depends on the dimensions of the image and the size of the filter. The reduction in size is prevented by using zero-padding to expand the edge regions of the image so that the size of the feature map equals the size of the image after the filtering process has been completed.

A rough visualization of this process can be seen in figure 2.1.

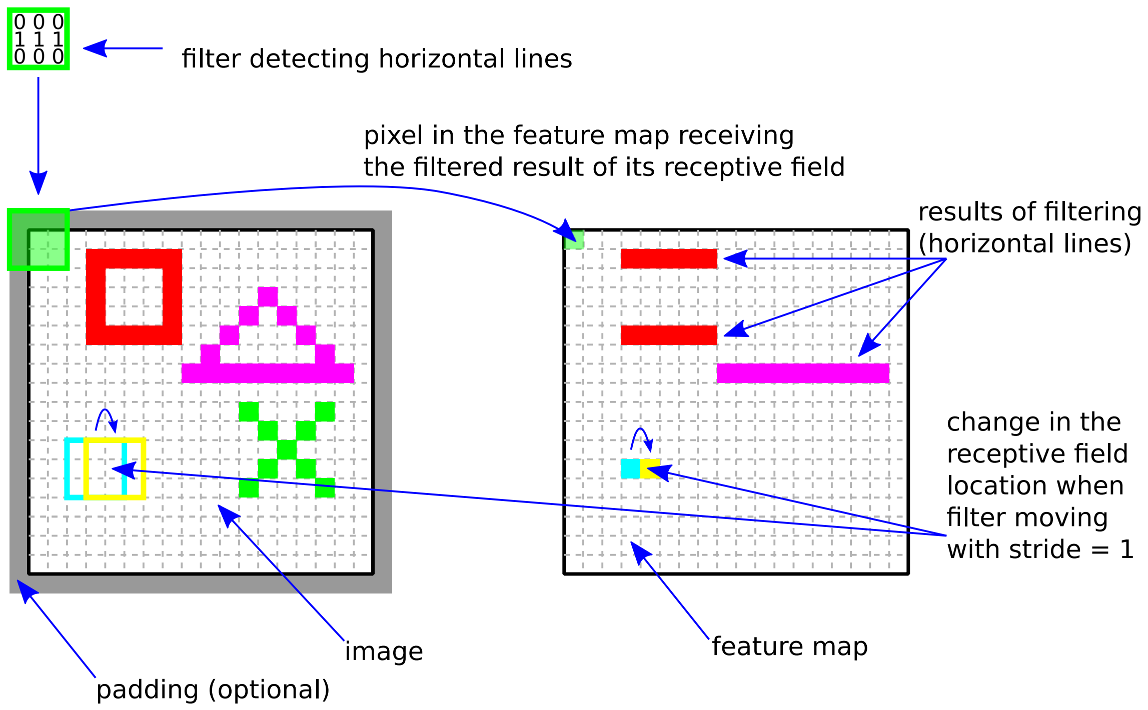

0 0 0
1 1 1
0 0 0
filter detecting horizontal lines
pixel in the feature map receiving the filtered result of its receptive field
results of filtering (horizontal lines)
change in the receptive field location when filter moving with stride = 1
image
feature map
padding (optional)

Figure 2.1: A convolutional layer producing a feature map of an input.

The behavior of a convolutional layer means that it utilizes what is known as parameter sharing to reduce the complexity of the model. In a traditional fully connected layer, each neuron of a preceding layer is connected to every neuron on the successive layer, but in a convolutional layer each neuron shares the same weights of the filter. This also plays a part in what is known as translational invariance: a filter detecting a feature in the upper-left corner of an image is also capable of detecting the same feature in the lower-right corner of an image.

The most typical activation used with a convolutional layer is the Rectified Linear Unit Function (ReLU)

$$ReLU(z) = max(0, z) \tag{2.1}$$

where z is the weighted sum of the inputs, that is the value of the node before the activation function has been applied. *ReLU* it is typically able to produce good results and is faster to compute than many of the alternatives [10].

Pooling layer

The size (in terms of *width* · *height* · *depth*) of the output produced by the final convolutional layer rarely matches the size of the original input. A convolutional layer (or a series of convolutional layers) is typically followed by a pooling layer which shrinks the input to reduce the computational load and memory usage of the model [10]. This also has a regularizing effect and adds to the translational invariance of a CNN. The two different pooling operations typically available for a CNN are max pooling and average pooling. Max pooling keeps the largest number in the examined region and discard the rest. Average pooling computes the mean of the numbers in the examined region. Of these max pooling is the most common one used as it helps to select the regions where a feature is most clearly present.

Regularization

CNNs like any other neural networks, are also prone to overfitting. Two typical techniques used to alleviate this problem, are dropout and batch normalization.

Dropout is the act of setting a random set of neurons in a layer to output zero during each training iteration [10]. Effectively, this means that these neurons are *dropped out* from the computation and cannot participate in making the prediction. As a result, the model has to learn alternative network paths for the sample it has just seen, reducing the intensity of overfitting. It should be noted that dropout is only active during the training phase and all neurons work normally after the training has been concluded.

Batch normalization is primarily an optimization technique, but it also has a regularizing effect. It works by zero-centering and scaling the input, then by shifting the result based on specific learned parameters [10]. This happens on a per-batch basis.

2.1.2 Image classification with CNNs

The typical usage of a CNN network is image classification, the act of assigning a label to an image. When assigning a label the essential content of an image is associated with some noun describing the content. Image classification networks typically utilize very deep CNNs with multiple layers.

However, a CNN need not produce a prediction of an image class as an outcome. A common use case is one in which a CNN is stripped from its fully connected layers, only leaving the trained convolutional blocks (convolutional and pooling layers) in place. This retained part is then connected to other networks and functions for further processing. In fact, this is the way CNNs are used throughout this thesis in various places: a CNN extracts feature maps from an image containing a vehicle, and instead of passing these feature maps to a set of fully connected layer for prediction, the feature maps are further processed by whatever layers are suitable in the given situation.

The collision detection framework presented in this thesis relies on a network named VGG16 [11] for image classification tasks. It is a deep CNN that was originally developed in 2014 but remains useful to this day. The structure of VGG16 is shown in Figure 2.2.

VGG16 receives as input an image of size 224 by 224 pixels with three channels. The input is sequentially processed by both convolutional and pooling layers. Convolutional layers extract increasingly high-level feature maps from the image while pooling layers are used to downsample the information that is being feeded deeper into the network. The result produced by the final pooling layer can be processed by a fully connected network that can translate the feature maps into a classification. The result can also be manipulated in other ways, for example by processing it with a recurrent layer as is done in this thesis.

VGG16 is not the most recent or advanced competitor on the market, and there

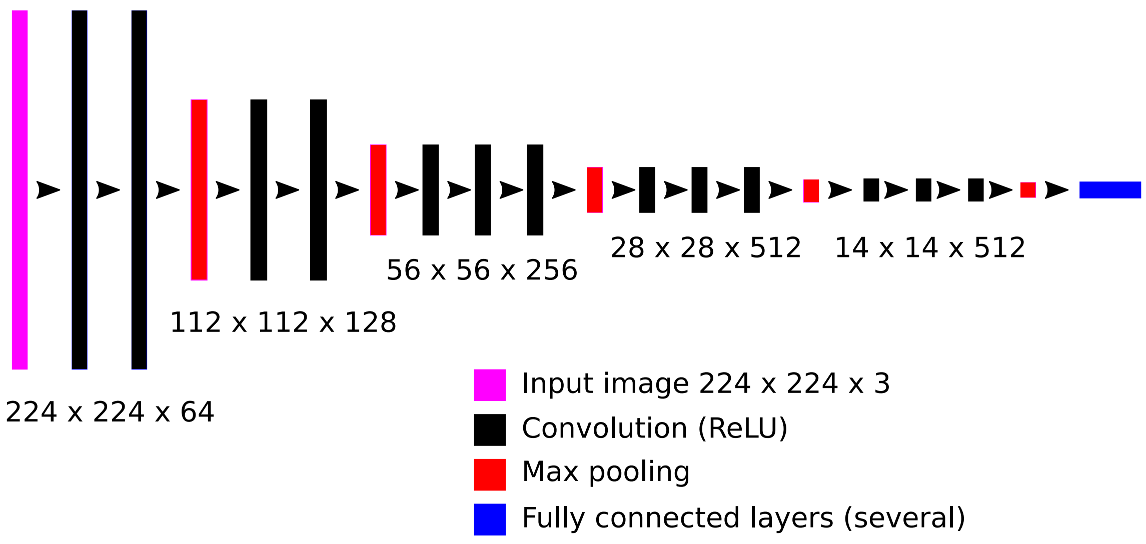


Figure 2.2: The convolutional layers of VGG16. The annotated dimensions are the same for each convolutional layer of the same size.

have been newer and more effective networks, but initial tests suggested VGG16 is both effective and adequate for purposes of this thesis and easy to fine-tune. It is also the network which the selected object detector, Faster R-CNN in section 2.3.1 originally relied on and therefore a natural choice as a classifier.

2.2 Recurrent Neural Networks

Due to the reasons discussed in section 1.2, the framework should not rely explicitly on processing static images individually. The framework should also be able to understand the temporal aspects of a collision and the way the event progresses from one frame to another. This requires tooling that is suitable for handling data that is sequential in nature.

A recurrent neural network (RNN) is a form of network specifically designed for sequential data. A basic building block of RNN is a layer that consists of recurrent cells. A recurrent cell is much like any typical neuron in a fully connected network, except for the fact that in addition to the normal input a recurrent cell also receives a second input. This second input provides the cell information about the previous state of the layer. A simplified illustration is shown in Figure 2.3.

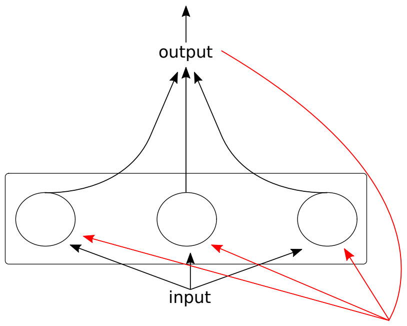


Figure 2.3: A very simple RNN layer with recurrent cells.

The following equation is used in computing the output of an RNN layer for a single sample [10].

$$\mathbf{y}_{(t)} = \phi\left(\mathbf{W}_x^T \mathbf{x}_{(t)} + \mathbf{W}_y^T \mathbf{y}_{(t-1)} + \mathbf{b}\right) \tag{2.2}$$

where ϕ is the activation function, t is the time step, $\mathbf{b}$ is the bias vector, $\mathbf{x}$ and $\mathbf{y}$ are the sample and the output at the given time step t, respectively. $\mathbf{W}$ indicates the weights associated with the given vector.

A recurrent network can generate two kinds of output:

1. An output for each time step: the layer outputs an entire sequence with all the time steps included.

2. Single output after the final time step: only the final result is returned after all the time steps have been completed. This is the desired choice for collision detection purposes. The purpose of a collision detector is not to output an entire sequence but only the binary result (collision or no collision) that has been deduced from the sequence.

The basic RNN cell suffers from several issues, particularly when dealing with long sequences [10]. Admittedly a collision sequence consisting of a couple of frames is not long when measuring the length in the number of frames, but more recent alternatives generally provide better performance in nearly all situations and there is rarely any reason to resort to the old-fashioned basic RNN cells.

The Long Short-Term Memory (LSTM) is a type of RNN cell that attempts to mitigate some of the problems experienced with basic cells. The detailed internal structure and theory of an LSTM cell is beyond the scope of this introduction, but on a superficial level it can be described as a cell that has two input vectors that depend on the previous state of the cell: short-term state **h** and long-term state **c** [10]. These can be observed in Figure 2.4.

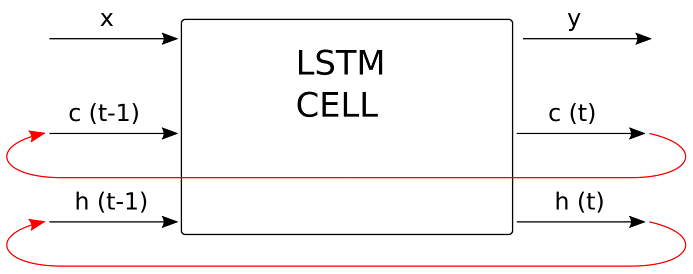


Figure 2.4: The inputs and outputs of an LSTM cell.

The high-level principle is that with LSTM as the cell structure, the network learns what information should be stored in the long-term state and what information should be discarded. This is a very useful property, and there is typically very little reason to subject a model to the limitations of a basic RNN cell.

As a last note, it should be mentioned that a very typical activation function for a recurrent cell is the hyperbolic tangent function

$$tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} = \frac{e^{2z} - 1}{e^{2z} + 1} \tag{2.3}$$

where z is the weighted sum of the inputs. The hyperbolic tangent function is the preferred activation used in many publications dealing with RNNs, although ReLU is also possible [10].

2.3 Object detection

Object detection is the process of identifying an object in a sub-region of an image and associating a label to the object from a set of known classes. As shown in Figure 2.5, the location of object is determined with a bounding box (a colored rectangle) which fits around the object as tightly as possible.

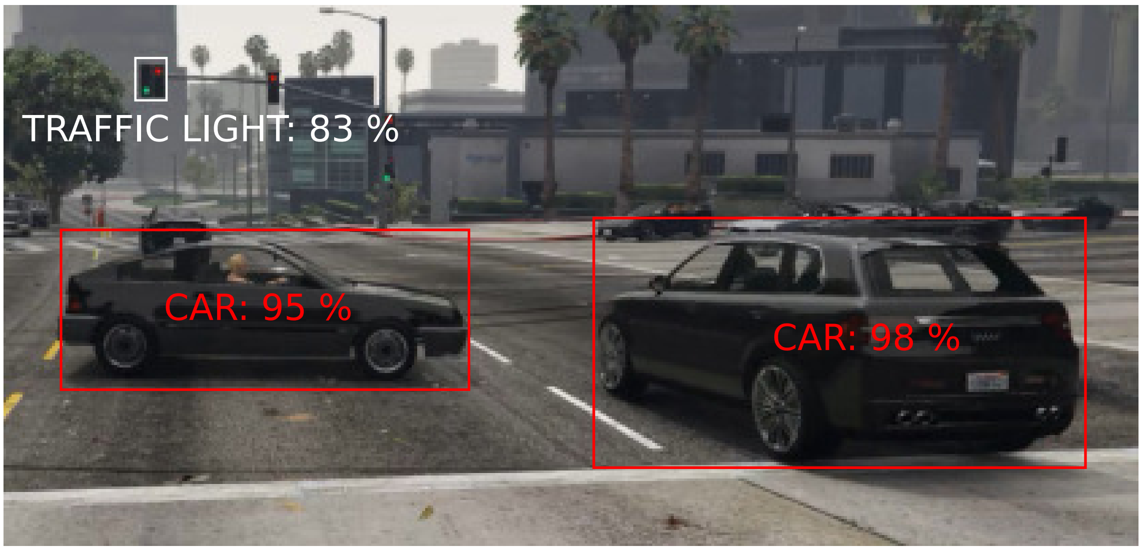


Figure 2.5: An image image detection example with bounding boxes around objects. The percentages indicate how confident the detector is about the content of a given bounding box. Very small or occluded objects often remain undetected, such as the vehicles in the background.

This section presents a class of networks known as region-based convolutional neural networks, one of the most widely used group networks for image classification purposes.

2.3.1 Region-based Convolutional Neural Networks

Several solutions exist for detecting objects in an image. They can be classified into two major categories, one-stage and two-stage, depending on the number of steps they take when performing the detection process [10]. A rough characterization is that a one-stage detector is faster but less accurate than a two-stage detector. This ambivalence is a problem inherently tied to collision detection: the detection should be fast, but it should also be accurate. Performance speed is definitely an issue to be addressed, but as the focus of this thesis is primarily in investigating *if it is even possible to model collisions with synthetic data*, more emphasis was put on accuracy and less on performance.

Region-based Convolutional Neural Networks (R-CNN) are a family of two-stage object detectors that rely on so-called region proposals (regions in the image) on which the network attempts to do object classification. That is, a prior component of the network suggests a region that *possibly* contains an object, and a latter component then decides if there is an object in this region and what is the class of that object.

The family of network receives its name from the pioneering network of the same name, R-CNN, that uses a method known as selective search for generating the region proposals [12]. The selective search algorithm assigns regions of an image into groups based on specific features of similarity (e.g., color and shape). From every processed image the algorithm extracts approximately 2000 different regions.

After this the network executes a deep CNN on each of these regions. The CNN acts as a feature extractor and the extracted features are reshaped into the form of a dense vector. Finally, a support vector machine makes a classification based on the contents of the vector and, in case the region contains an object, linear regressor adjusts the bounding box (originally created by selective search) to tightly fit around the detected object.

The one considerable drawback of the original R-CNN is the speed: there are approximately 2000 region proposals, and the latter components of the network have to process every one of the proposals before the ultimate decision can be made. Selective search is also a fixed algorithm: it cannot be trained, and it does not learn to generate better proposals.

The next version of the network, Fast R-CNN, made considerable improvement by executing a CNN on the whole image and generating region proposals from the final set of feature maps that the CNN outputs [13]. Selective search is still used for generating the region proposals, but it is only used on the generated feature maps instead of the whole image, resulting as a considerably improved execution time. A pooling layer then resizes the proposed regions and softmax layer is responsible for making the classifications.

Fast R-CNN was again later superseded by another improved version known as Faster R-CNN [14], which is the network used in the implementation phase of this thesis. Like its predecessors Faster R-CNN is also a two-stage detector. It has two central components: Region Proposal Network (RPN), and a convolutional detector that uses the output of the RPN. The network structure is shown in Figure 2.6.

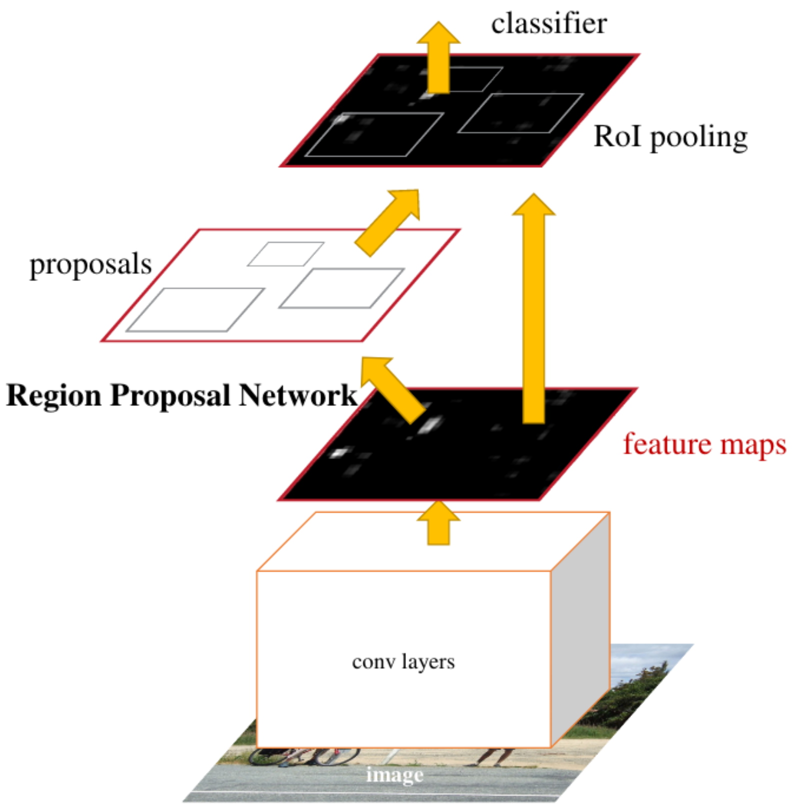


Figure 2.6: The Faster R-CNN network (Ren, He, Girshick, *et al.* [14], 2017).

The basic principle by which the network functions is as follows:

1. RPN receives a set of feature maps from the convolutional layers. RPN uses these feature maps to predict Regions of Interest (RoI) and then determines which of the regions actually contain an object. If a region is detected to contain an object the RPN generates a bounding box proposal.

2. RoI pooling layer classifies the object in the proposed bounding box and fine-tunes the location of the object.

Faster R-CNN differs from the earliear implementations in that it does not use selective search at all. Instead, the RPN creates the proposed regions by using sliding a window that moves over the feature maps and generates so-called anchor boxes of different sizes as it goes. A binary classifier learns which of the anchor boxes contain foreground and which contain background. All anchor boxes detected to contain background are discarded. The remaining anchor boxes receive a score that represents the probability of there being an object in the box.

This effectively means that the RPN learns the regions that should be proposed (as opposed to untrainable selective search). As a two-stage detector Faster R-CNN among the faster ones, but its predecessors have little hope of achieving performance that would be acceptable in any real-time scenario involving collision detection. More effective solutions undeniably exist, but for the current research questions Faster R-CNN provides a suitable balance between speed and accuracy, as the two-stage structure makes sure as few vehicles as possible go undetected.

2.4 Object tracking

Object tracking is the process of following a visual object moving in sequential data. When the object is being followed, awareness of the identity of the object should be maintained. Sequential data in which the object moves is typically a stream of video frames and the identity is any peace of information that distinguishes the object from other objects. Figure 2.7 is a simple illustration of this idea. The moving vehicle is tracked through a series of frames and the vehicle is associated with a constant identity number.

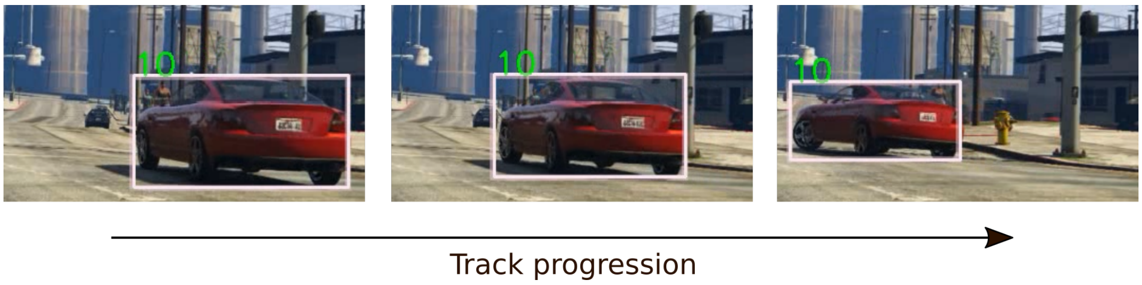


Figure 2.7: An example of tracking a vehicle from frame to frame while maintaining awareness of the identity (the number displayed above the vehicle).

Each detected and tracked object receives a unique identify if the tracking process is successful. Figure 2.8 highlights this idea: three different vehicles have been identified in the frame and each of the vehicles has received a unique identity. Without this requirement it would be impossible for the system to track specific objects reliably.

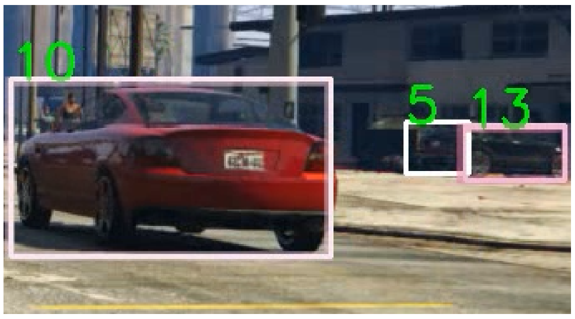


Figure 2.8: Each tracked object gets a unique identity number (shown above the object).

To a human observer, the process of identifying a moving object is intuitive and happens subconsciously, but to a computer an object detected in a frame is an object detected in that frame only. Without identity it is not associated to any preceding or subsequent detection. An identity is needed to group information from different frames as a cohesive whole.

In the simplest case a successive tracking method can be implemented with a very naive approach [15]. For example, it is possible to associate the centroid of an object in a later frame with an object centroid that was closest to the same position in the preceding frame. Another approach is to associate objects with most overlapping bounding boxes in different frames.

A different approach to tracking comes from a family of object trackers known colloquially as model-free trackers [16]. These trackers rely only on motion cues and do not rely on object detection techniques. That is, a car can be tracked without any car detection being made by an object detector. The strength of these trackers is also their drawback, as the object has to be moving instantaneously or it cannot be tracked. Additionally, as object detection is already a pre-requisite of collision detection, model-free trackers are not particularly useful for the framework that is presented in this thesis.

Many naive attempts are susceptible to be hampered by commonplace visual phenomena, particularly when the tracked object gets partially or completely occluded by another object [17]. In many cases a situation such as this will result in an identity switch: The tracker loses track of the object and, upon next observation, considers it to be a different object, meaning that the tracked identify of the factually same object is incorrectly switched to another, different identity. That is, the object remains the same, but the system thinks it is not.

2.4.1 SORT

Simple Online and Realtime Tracking (SORT) [18] is an effective tracking method that is built on two central components, Kalman filtering and data association using the Hungarian algorithm. It also has some low-level functionality that alleviates the identity switching problem in some short-lived occlusion events. On a superficial level the operation of SORT can be described as follows:

1. Given a frame in a video stream, every detected and tracked object (target) in the frame has some state including location, velocity, bounding box and a predicted state in the next frame.

2. When a known target is redetected in the current frame, its state is updated based on the state it had in the previous frame and the detection box of the target in the current frame. Kalman filtering [19] is used to update the velocity of the target, based on the motion of the target from the previous to the current frame.

3. If a known target was not redetected, an estimation model computes an estimated location of the target in the current frame. The target is assumed to move at a constant velocity that it is supposed to have, based on the latest detection update.

4. Before targets and detection boxes are associated, each target has a predicted state (including a predicted detection box) in the current frame. The predicted boxes and the detection boxes are compared to each other by computing the intersection-over-union (IOU) [20] of every prediction with every detection. From these values an assignment cost matrix is computed, which in turn is used allocate every detection with a prediction (and hence, a target) with the help of the Hungarian algorith [21].

5. A minimum IOU threshold is used to discard any allocations for which the computed IOU is below the threshold. That is, a detection and a prediction must have overlap larger than the threshold, or the allocation is rejected.

6. A detection is considered to be a new and unseen target, if the detection box cannot be allocated to any prediction.

7. If a target goes undetected for a specified number of frames, it is terminated. This means that the target has lost its identity even if it reappears in a later frame.

The problem with SORT is that it too suffers from frequent identity switches, regardless of the attempted measures to improve accuracy. This would be particularly evident in a scene with much traffic where occlusions could last for several frames, resulting in an unavoidable identity switch. Because of this, an even more advanced solution was considered for the role of the object tracker.

2.4.2 Deep SORT

Deep SORT, or SORT with a deep association metric [22], is an extension to the original SORT approach. It incorporates SORT techniques with deep learning measures, resulting in improved tolerance to occlusions and hence a reduced number of identity switches.

The deep learning component of Deep SORT is a CNN model trained on samples of the tracked object class. This CNN model, stripped from its final layer, does not produce a prediction. Instead, the last remaining layer is a fully connected layer that outputs a vector representation of the tracked object. This is what the authors call a deep appearance descriptor. A high-level overview of this network is shown in Figure 2.9.

The appearance descriptor obtained in this fashion is then used in comparing feature similarity with seen targets. It is used in conjunction with a matching cascade algorithm that relies on Mahalanobis distance [23] and attempts to associate targets over multiple frames, favoring more frequently seen objects. The result is an

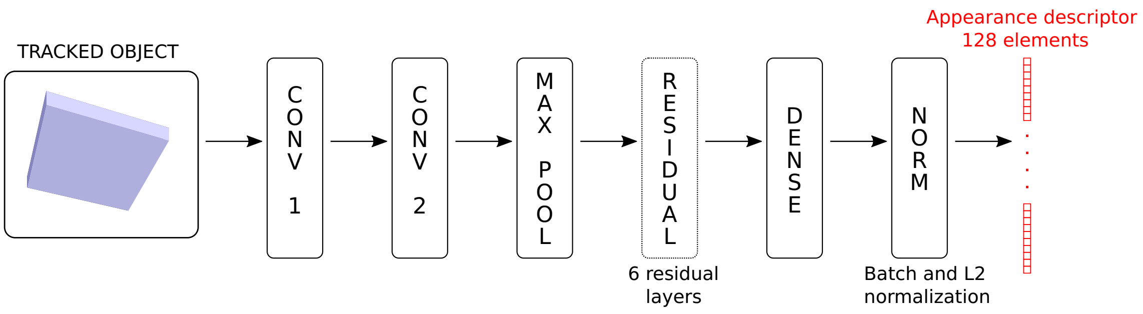


Figure 2.9: The Deep SORT CNN generating an appearance descriptor.

algorithm that can withstand longer periods of occlusion while still managing to be effective in real-time.

Such characteristics are virtually mandatory when tracking objects in traffic, where occlusions happen all the time, all over the place. In a collision detection context, the duration of the event and hence the time needed to track a vehicle is short, but an ill-timed occlusion can still be crucially detrimental to the detection process. This is the main reason of selecting Deep SORT in this thesis.

3 Implementation and evaluation

This chapter discusses the proposed collision detection framework. The first three sections describe the separate phases of the process. Then, a description of the used datasets and evaluation metrics is provided. Finally, some of the technical aspects related to the implementation are highlighted.

3.1 Proposed framework

At the abstract level, the proposed framework can be described as a pipeline that receives a video as input and produces collision detections as output. As shown in Figure 3.1, the framework uses three main phases as follows:

1. Object detector: detects the vehicles in a video frame and associates each detected vehicle with a bounding box.

2. Object tracker: tracks the detected vehicles and maintains identities of the vehicles from a current frame to the next frame. It also extracts the track images for the collision detector (see Figures 3.3 and 3.4).

3. Collision detector: detects if a sequence of track images from tracked vehicle is a collision event or not.

In this context *frame* refers to a complete, full-sized image that represents a one time step in a video (Figure 3.1, phase 1). *Track image* represents a small sub-region of a frame in an area where a tracked vehicle is located (Figure 3.1, phase 2).

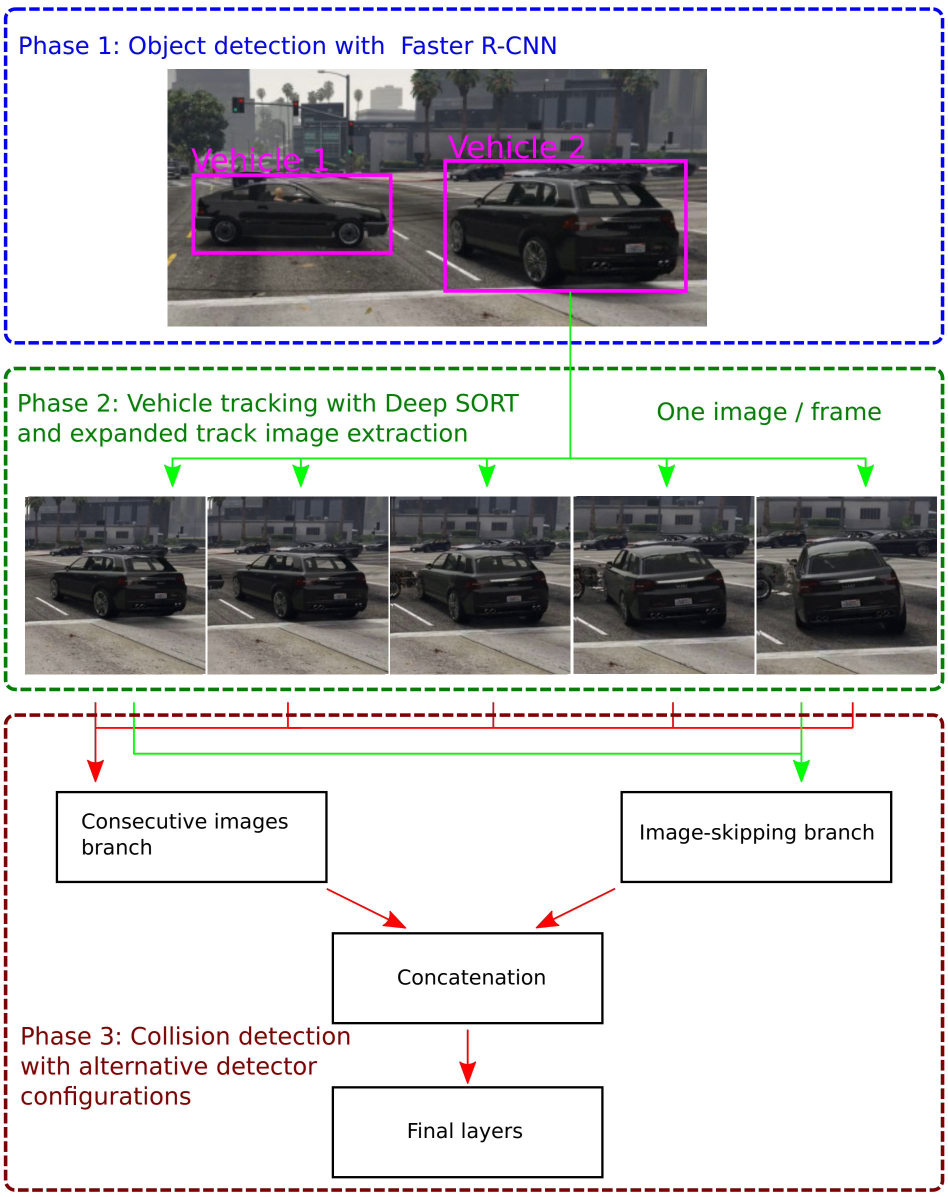

Phase 1: Object detection with Faster R-CNN
Vehicle 1
Vehicle 2
Phase 2: Vehicle tracking with Deep SORT and expanded track image extraction
One image / frame
Consecutive images branch
Image-skipping branch
Concatenation
Final layers
Phase 3: Collision detection with alternative detector configurations

Figure 3.1: A high-level overview of the framework. Every vehicle is processed in the same way as Vehicle 2 in this illustration.

The proposed framework is modular: the implementation of one phase can be changed while leaving other phases untouched. A latter phase does not depend on the exact implementation of the previous phase, nor does it depend on the libraries the previous phase depends on. Only the output produced by the preceding phase is significant. For example, when improved real-time performance is needed, Faster R-CNN can be substituted with a faster one-pass detector architecture.

3.1.1 Object detection

The task of identifying and locating an object in an image or a video frame is known as object detection. The result is a bounding box, a rectangular sub-region in the image containing an object of interest (a vehicle).

In this phase the detector receives a video frame or an image as input. The task of the detector is to locate the bounding boxes of all the vehicles in each frame. The detector returns the detections as an array of bounding box coordinates in a top-left/bottom-right format. An overview of the detection phase is shown in Figure 3.2. The library responsible for frame extraction is OpenCV [24] and each frame is presented as a Numpy array [25].

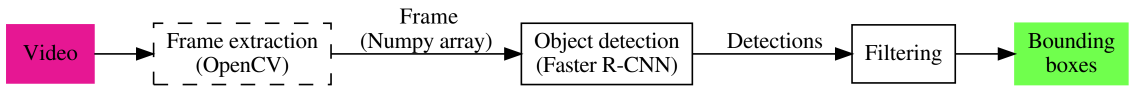


Figure 3.2: An overview of the object detection process

Most available object detectors, including Faster R-CNN, are general detectors. Therefore, in addition to vehicles, they can detect other objects such as humans and animals. The specific model used in this thesis was pre-trained Faster R-CNN with backbone ResNet 101 [26] from the Detectron2 model zoo. The model was not fine-tuned in any way for this task.

In this project only detections involving vehicles are retained while other detections are dropped as irrelevant for the task at hand. An alternative way would be to replace the current detector with another one that is capable of detecting only vehicles. This could improve real-time performance but has no effect on accuracy.

It should also be emphasized that any other detector capable of extracting vehicle bounding box information would be just as suitable a candidate. This is possible due to the modularity of the framework.

3.1.2 Object tracking

Sometimes a vehicle crash is evident in a single static image. Other times one crashing vehicle at least partially occludes the other one and it is impossible to tell if the vehicles are in contact or not. That is why some notion of movement should be included in the detection process, which means that the vehicles must be assigned some kind of identity. With this identity, the detector can follow a vehicle from one frame to another. To a human observer, this is a nearly effortless and intuitive process. To a computer vision system, it is a fundamentally challenging problem known as object tracking.

Vehicles are tracked frame by frame: During each frame detections made by the object detector and the actual frame are passed on to the tracker. The tracker uses the received parameters along with its own internal state (see chapter 2.4) to decide if the observed vehicle is a new and previously unseen vehicle or if it is a vehicle that has been seen earlier and is now continuing on its track. Each vehicle (and its track) is associated with an abstraction named *VehicleTrack*, briefly characterized in Listing 1.

Listing 1 A simplified version of the VehicleTrack class.

```
class VehicleTrack:
    # a vehicle ID matching the ID given by the tracker
    vehicle_id: int
    # track images of a tracked vehicle
    images: Dict[int, np.ndarray]  # key := frame number
    # fill-in algorithm for fixing missing detections
    def fill_in(self, frame_id: int, frame: np.ndarray,
                detection_box: List[int]) -> None:
        # see Listing 2 for details
```

The purpose of this class is to be a container for whatever images are associated with a particular vehicle during its observed lifetime in the video. The track of a vehicle is the full sequence of images in which a vehicle has been detected. For example, if a video is 100 frames long and a specific vehicle has been detected in frames 20-80, then the track of this vehicle consists of 61 track images. This extraction process is illustrated in Figure 3.3.

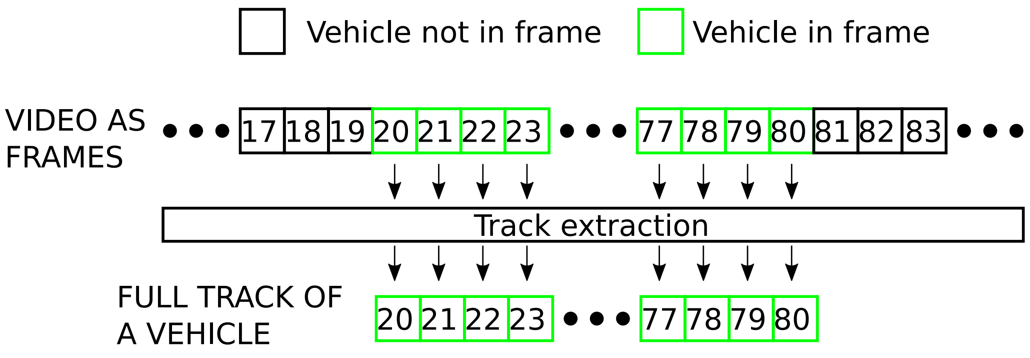


Figure 3.3: Extraction of the full track of a vehicle that is detected in video frames 20-80.

Every track image is an image of the tracked vehicle expanded to include not only the vehicle itself but also its immediate surroundings in the current frame, as shown in Figure 3.4. The purpose of this expansion is to include information about any object that is close to the tracked vehicle and, possibly, also coming into contact with it. The track images are also scaled to the size of 224 x 224 pixels, while maintaining aspect ratio of the vehicle region, to keep the image dimensions constant.

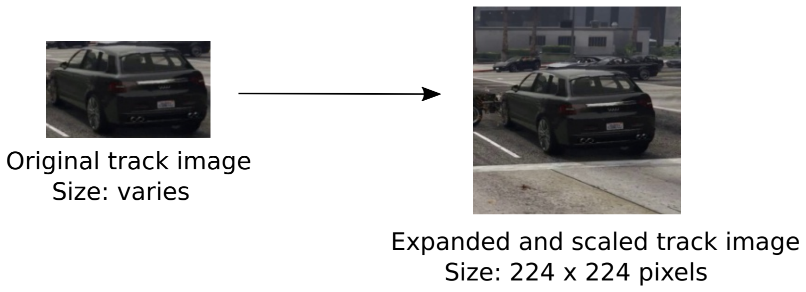


Figure 3.4: An example of expanded track image which consists of the contents of the bounding box along with the nearby pixels outside the box. The aspect ratio of the box is preserved.

Faster R-CNN and Deep SORT are not infallible and a vehicle that is detected in an earlier frame often goes undetected for a couple of frames before being detected again. When an image is added to an existing track the *VehicleTrack* object always checks to see if there has been a gap in the detection process. In case a gap exists, the algorithm in Listing 2 is used to fill in the missing images.

Listing 2 Fill-in algorithm for fixing tracks with missing images

```
change($B_a, B_b$) := the change in the sides between boxes a and b
distance($F_a, F_b$) := the number of frames between frames a and b
displacement($B_a, B_b$) := centroid displacement between boxes a and b
$F_0,\ B_0 \leftarrow$ last detected frame and bounding box
$F_1,\ B_1 \leftarrow$ currently detected frame and bounding box
$s \leftarrow$ displacement($B_0, B_1$) / distance($F_1, F_0$)
$r \leftarrow$ change($B_0, B_1$) / distance($F_1, F_0$)
**for** each missing frame $F_i$ and bounding box $B_i$ **do**
    $i \leftarrow$ distance($F_i,\ F_0$)
    $B_{i,x1} \leftarrow B_{0,x1} + (i \cdot s_x) - (i \cdot r_x)/2$
    $B_{i,y1} \leftarrow B_{0,y1} + (i \cdot s_y) - (i \cdot r_y)/2$
    $B_{i,x2} \leftarrow B_{0,x2} + (i \cdot s_x) + (i \cdot r_x)/2$
    $B_{i,y2} \leftarrow B_{0,y2} + (i \cdot s_y) + (i \cdot r_y)/2$
**end for**
```

In short, the algorithm computes the displacement and resize values which are then used to transform an actual detection into an artificial detection so that the track is complete without any missing images.

The overall view of the tracking phase can be observed in Figure 3.5. The detection processing involves the use of the fill-in algorithm in Listing 2 (if needed), in addition to extracting the track image from the original frame.

As an implementation detail it should be noted that the original Deep SORT model was trained with data consisting of pedestrians walking on a street [22]. It was therefore not usable for vehicle detection purposes. Instead, this thesis relied on an alternative implementation trained on vehicles [27]. However, the original weights of this model are for an old version of PyTorch (1.0.1) and therefore unusable as such in any up-to-date environment. Because of this the weights were retrained for a more recent version of PyTorch (1.7.0) using the process outlined in the repository of [27].

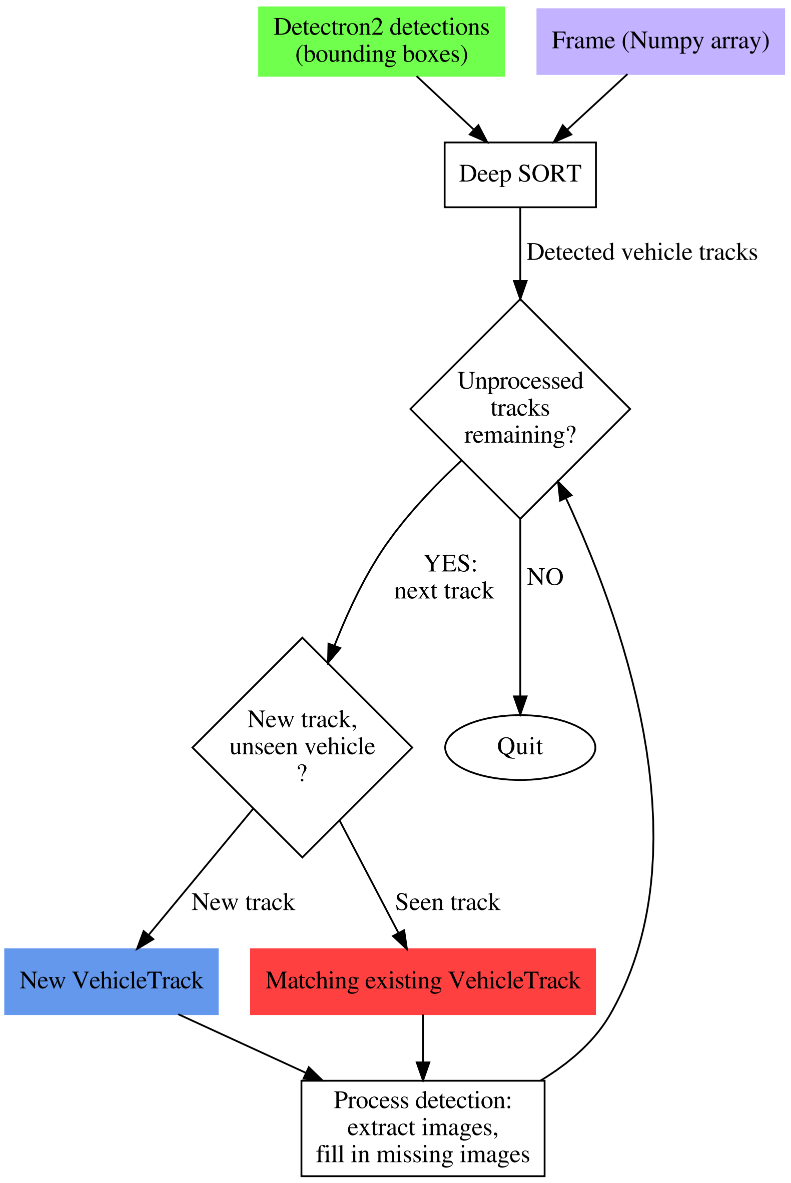

Detectron2 detections (bounding boxes)
Frame (Numpy array)
Deep SORT
Detected vehicle tracks
Unprocessed tracks remaining?
YES: next track
NO
New track, unseen vehicle ?
Quit
New track
Seen track
New VehicleTrack
Matching existing VehicleTrack
Process detection: extract images, fill in missing images

Figure 3.5: An overview of track processing.

3.1.3 Collision detection

When a vehicle has been tracked for a sufficient number of frames, the observed track images can be processed by the collision detector. The task of the detector is to evaluate whether the observed sequence of images is a normal (non-collision) or a collision sequence. The high-level principle is shown in Figure 3.1 (phase 2 and phase 3).

For this project, several alternative models were built using the TensorFlow deep learning framework [28] and all the alternatives employ VGG16 [11] as a backbone. The models can be grouped as follows:

- Sequence-of-5 models: models based on sequences of 5 track images.
- Sequence-of-10 models: models based on sequences of 10 track images.

Therefore, a sequence-of-n model bases its prediction on $n - 1$ previous track images and the current track image. All the models (excluding the baseline model) utilize the same basic component, referred to as the Temporal Base Model (TBM). It is a simple neural network with both convolutional and recurrent properties, intended to capture some meaningful visual and temporal interactions in the sequence it processes.

The TensorFlow framework offers a special layer named TimeDistributed [29], which allows for a layer to be applied to a temporally sliced input. When using TimeDistributed the operation that is being performed shares same set of weights for each of the temporal slices. TimeDistributed offers an easy way to handle a sequence of images as temporal data and is an integral part of the TBM. The high-level principle of operation of the TBM is shown in Figure 3.6.

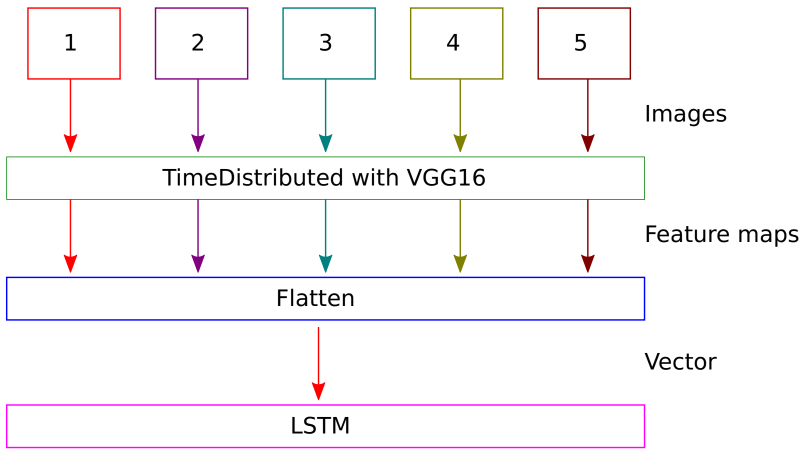


Figure 3.6: The structure of the TBM.

The collision detection process can be summarized as follows:

1. An image sequence is sliced by a TimeDistributed layer.
2. VGG16 extracts features from each slice (track image).
3. The features are flattened into a single long vector per track image.
4. The vectors are grouped as a temporal sequence, maintaining image order.
5. The LSTM layer processes the sequence.
6. Dropout is used on the LSTM output for regularization.
7. Fully connected layers are used in making the final prediction.

On an abstract level, the LSTM layer attempts to generate the next suitable vector in the sequence. It predicts what kind of a vector of feature maps would be extracted from an image that is a suitable addition to the sequence the model has just seen. The LSTM layer outputs only the final sequence, as discussed in section 2.2. In the TensorFlow framework, this kind of output is achieved by using the argument *return_sequences=False*. This resulting vector is then processed by the final layers, after which the model can decide if the sequence is a collision or not.

The configuration of the VGG16 backbone is the following:

- The fully connected layers are dropped (*include_ top = False*).
- Four last CNN layers are set as trainable.
- All input values are preprocessed using *vgg16.preprocess_ input* [30].
- (Optional) All input values are rescaled.

It is noteworthy that neither *vgg16.preprocess_ input* performs any rescaling nor was any external rescaling utility used. Therefore, a separate rescaling layer was added to the TBM. The configuration is shown in Listing 3.

Listing 3 The layers of the TBM.

```
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.applications import vgg16
backbone = vgg16.VGG16(include_top=True, weights='imagenet',
                       input_shape=(224, 224, 3))
for layer in backbone.layers[:-4]:
    layer.trainable = False
SEQUENCE_LENGTH = 5  # or 10
i = layers.Input((SEQUENCE_LENGTH, (224, 224, 3))
x = vgg16.preprocess_input(tf.cast(i, tf.float32))
x = layers.experimental.preprocessing.Rescaling(
    1. / 255, 0.0, name='Rescaling')(x)
x = layers.TimeDistributed(backbone)(x)
x = layers.TimeDistributed(tf.keras.layers.Flatten())(x)
x = layers.LSTM(256, activation='tanh')(x)
```

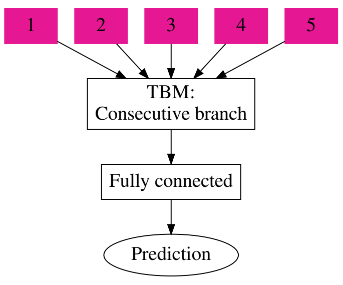
The models were compiled as shown in Listing 4. Stochastic Gradient Descent (SGD) [31] with the learning rate of 0.0025 was used as an optimizer, as it provided the most consistent results during initial tests.

Listing 4 Model compilation.

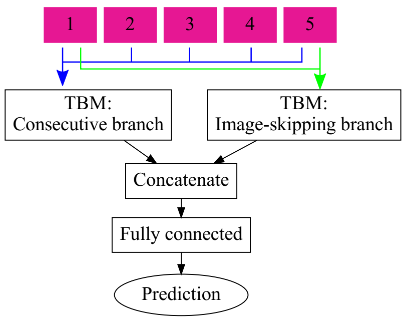
```
from tensorflow.keras import optimizers

optimizer = optimizers.SGD(learning_rate=0.0025)

model.compile(optimizer,

              loss='binary_crossentropy',

              metrics=['binary_accuracy'])
```

Proposed collision models

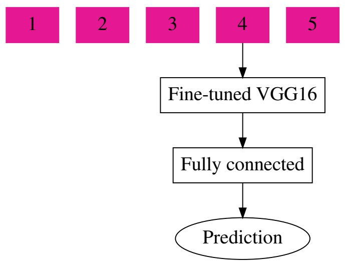
This section proposes four different models and a baseline model for the collision detection phase. The proposed Collision Models (CM) are named as *CM:m-n*, where m is the length of the image sequence and n is the number of branches in the model. The structures of the sequence-of-5 models and the baseline model are shown in Figure 3.7. The sequence-of-10 models are virtually the same except for the sequence length which is ten images instead of five.



(a) CM:5-1.



(b) CM:5-2.



(c) Baseline.

Figure 3.7: Overview of the sequence-of-5 models and the baseline model.

All models utilize the same configuration for the fully connected layers (see Listing 5). It was tested with both *tanh* and *ReLU* activations (section 3.2.2).

Listing 5 The fully connected layers.

```
from tensorflow.keras import layers

# x is the output of the previous layer

x = layers.Flatten()(x)  # if the form of x requires

x = layers.Dense(512, activation='tanh')(x)  # or 'relu'

x = layers.Dropout(0.5)(x)

x = layers.Dense(256, activation='tanh')(x)  # or 'relu'

o = layers.Dense(1, activation='sigmoid')(x)
```

The details of the models are as follows:

CM:5-1

The model CM:5-1 uses one branch for processing a sequence of five consecutive track images, where each image is one temporal slice processed by the TBM.

The last image in the sequence is the most recent image that was extracted during the tracking phase. It has originated from the frame the tracker has just seen. In addition to this track image, four previous images are used for detection.

CM:5-2

The model CM:5-2 has two different branches: one branch for processing five consecutive track images, and one branch with image-skipping behavior.

The image-skipping branch ignores all detections and track images between the first and the last image. The purpose of this branch is to capture the drastic changes between the beginning and the ending of the sequence, whereas the consecutive branch focuses on the more distinguished image-to-image changes.

CM:10-1

The model CM:10-1 is very similar to the model CM:5-1 except that sequences of ten consecutive images are used for detection. The models uses only a single branch when processing the track images.

CM:10-2

The model CM:10-2 is very similar to the model CM:5-2. The first branch processes a sequence of ten consecutive images. The second branch is an image-skipping branch and only considers the first (1) and the last (10) image in the ten images long sequence while the other images are ignored.

The baseline model

A baseline model is needed to evaluate the performance of the collision models. A fine-tuned VGG16 network is used for this purpose, and as VGG16 is an image classifier, the detection of the sequence type (normal or collision) is based on only a single image while the other images in the sequence are ignored.

The baseline model uses the fourth image in the sequence. If any single image represents a collision properly, it is typically neither one from the beginning nor one from the end but one in the middle of a sequence. Additionally, the fourth image is observed by both the sequence-of-5 and the sequence-of-10 models and is a feature every model sees exactly once, as the fourth image is in the middle of the sequence and not processed by the image-skipping branch.

As with the other models, the VGG16 backbone is stripped from the fully connected top layers after which the final convolutional layers are set as trainable. The configuration of the baseline model is shown in Listing 6.

Listing 6 The baseline model.

```
import tensorflow as tf

from tensorflow.keras import layers

from tensorflow.keras.applications import vgg16

backbone = vgg16.VGG16(include_top=True, weights='imagenet',

                       input_shape=(224, 224, 3))

for layer in backbone.layers[:-4]:

    layer.trainable = False

i = Input((224, 224, 3))

x = vgg16.preprocess_input(tf.cast(i, tf.float32))

x = layers.experimental.preprocessing.Rescaling(

    1. / 255, 0.0, name='Rescaling')(x)

x = backbone(x)

# x is then processed by the layers in Listing 5.
```

Like the CM models that rely on temporal features, the baseline model also requires a network of convolutional layers that are used to extract the feature maps. However, unlike the temporal models, the baseline model does not use an LSTM layer. Instead, the output from the final VGG16 layer is passed on to a series of fully connected layers (Listing 5), which ultimately produce a classification result.

3.2 Dataset

This section describes the dataset that is used for the experiments in this thesis. It is important to note that the training dataset was only used to train the collision detector. The object detector or the object tracker were not trained with this data.

Only collisions involving passenger cars were generated in GTA V. Therefore, the framework was only trained and tested with passenger car data. Larger vehicles such as trucks or buses were not included in the datasets. This was done to limit number of different collision scenarios that had to be created in GTA V.

3.2.1 Training dataset

Obtaining diverse quality data of vehicle collisions is difficult, particularly if the point of view has to be that of an observer moving in the traffic flow. This excludes data from any stationary source such as a surveillance camera. For this reason, synthetic data is first used for training the proposed collision detection models (CM models and the baseline model). The models are then evaluated on a real data.

The training dataset was obtained from the video game GTA V with the help of an unofficial scripting library named ScriptHookV [2]. Several different accident scenarios were created in the game environment. The collision types varied from subtle bumps to violent crashes. Several different types of lighting was used, ranging from daylight to dusk. The weather type was limited to dry weather.

This resulted in a collection of 300 collision videos, approximately 150 frames long each. However, not all of these frames were used for training the collision detector, as most of the frames do not contain any information that can help the detector to learn what a collision looks like.

None of the videos consists entirely of collisions. A typical video is one that first contains some 100 frames of non-collision frames, followed by some number of collision frames. Therefore, the collection was annotated with two details for every video:

- the frame number when the collision begins, and
- the approximate X/Y coordinates of the collision in the frame in which the collision begins.

The videos were then processed by the first two phases of the framework, the object detector and the vehicle tracker. When coupled with the annotation data, this resulted in 580 extracted tracks where a vehicle is in collision with another vehicle. However, some of the tracks were too short to be used for training: only tracks having at least 15 normal and 15 collision images were used due to the sampling method (see Figure 3.9).

The process of extracting track images was as follows:

1. Object detector processes a frame in a training video.
2. Object tracker tracks vehicles across the frames.
3. Track images are extracted for every vehicle observed in the video.
4. If a vehicle is located in the annotated X/Y coordinates at the right time it is a participant in the collision and the track is a collision track.
5. Sufficiently long collision tracks are retained, other tracks are discarded.

The fourth step in the process means that the entire track is considered to be a collision track if a collision happens in any of the images. However, even if a track is a collision track it does not mean that all the images in the track contain a collision. In fact, most of the images are collision-free. For example, a vehicle might be moving completely normally for the first 80 track images, after which another vehicles suddenly crashes into it. The collision that happens from the image 81 onwards means that the complete track is considered as a collision track.

The reason for keeping only collision tracks is to feed the collision detector as challenging samples as possible. Many of the non-collision tracks involve a single vehicle traveling the road without any interaction with other vehicles, but the real challenge is in telling the difference between a collision and non-collision while two vehicles are very close to each other. Figure 3.8 is an example of a particularly challenging track, where the collision is nearly unnoticeable.

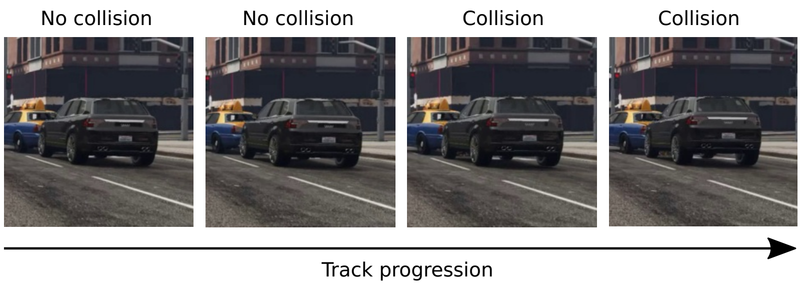


Figure 3.8: A very challenging track changing from non-collision to collision. The collision is nearly undetectable in any of the static images.

In order to train the model with as challenging samples as possible, the training sequences were sampled as shown in Figure 3.9 (for sequence-of-5 models). For the sequence-of-10 models the approach was similar but each sample consisted of ten images instead of five.

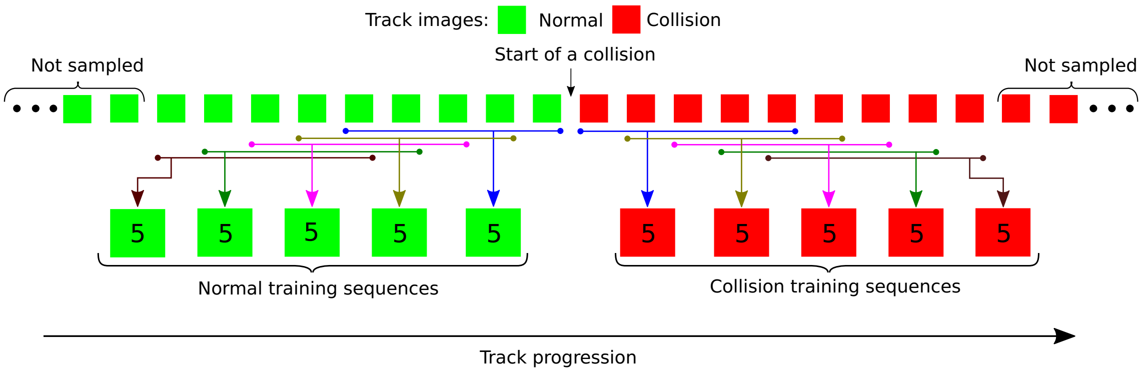

Track images: 🟩 Normal 🟥 Collision
Start of a collision
Not sampled
Not sampled
5
5
5
5
5
5
5
5
5
5
Normal training sequences
Collision training sequences
Track progression

Figure 3.9: An overview of how the train sequences were sampled for the sequence-of-5 models. Each sequence of five images represents one sample. Each collision track contributes five normal and five collision samples to the training set.

The sampling method can be described as one the selects sequences close to the annotated beginning of the collision and marks them appropriately. Sequences selected prior to the collision frame are marked as normal sequences. Sequences selected from the collision frame onward are marked as collision sequences. In all the cases the sampling was cut so that normal images would not leak into collision sequences or vice-versa. Furthermore, only track images close to the moment of collision were used. Images far away in the beginning or in the end of the track were not used.

The collision detector should also be able to understand normal traffic scenery, that is, sequences that at least to a human observer are very obviously non-collision sequences. This is not in conflict with the sampling method. In addition to challenging tracks (such as Figure 3.8), the tracks also contain less challenging examples where the moment of collision is easily identifiable. In these cases the images prior to the collision represent typical non-collision scenarios, such as the sequence in Figure 3.10, where the observed vehicle is not seen to be close to any other vehicle.

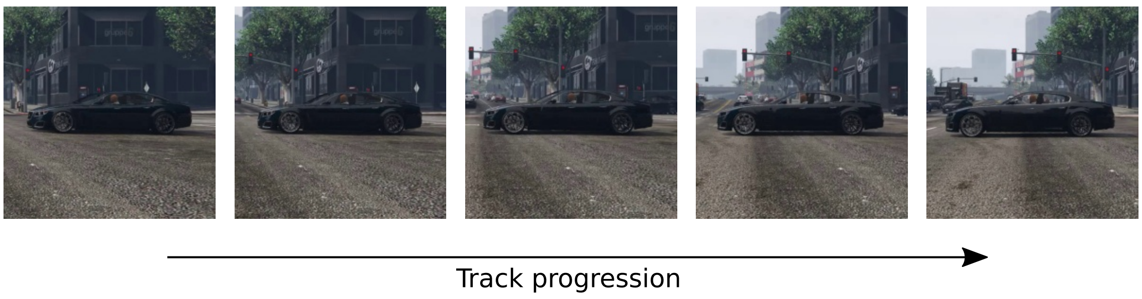


Figure 3.10: Example of a normal (non-collision) training sequence without challenging images. The last image in the sequence represents the time step right before a collision.

In addition to the sampling method, horizontal flip augmentation was used to increase the size of the training set. That is, that the size of the training set was doubled by creating a copy of each track and then horizontally mirroring each image in the copied track. Other forms of augmentation (rotation, shifting, contrast changes) were tested but they proved to be either ineffective or even harmful to the training process.

Lastly, the dataset was balanced so that the collision detector would see an equal number of normal and collision sequences during training. The final result is a balanced training set consisting of 1890 normal and 1890 collision sequences.

3.2.2 Test dataset

In [7] research was conducted by using accident footage in predicting (instead of detecting) collisions by utilizing what the authors call dynamic-spatial-attention Recurrent Neural Network. As a by-product the team published a dashcam-recorded accident dataset consisting of footage recorded in six major cities in Taiwan [8]. This thesis uses the dashcam dataset when attempting to establish a connection between virtual and real collisions.

However, due to the nature of the dataset only a small subset of it is relevant for the purposes of this thesis. The majority of the videos in the set consists of events that are not represented in the generated synthetic training data (e.g., collisions involving only two-wheeled scooters, which are very numerous in Taiwanese traffic but absent in the training data). Therefore, most of the videos were not used in testing the accuracy of the collision detector. Instead, only 30 most suitable videos involving passenger car collisions were selected and the efficacy of the models was then evaluated on these videos.

The video clips were processed by the object detector and the object tracker in a similar fashion as with the training set. However, all non-collision tracks were also retained and only a small number of low-quality tracks (e.g., very blurry or dark images) were discarded. The result is a test set of the following characteristics: 38 normal tracks, 52 collision tracks, 90 tracks in total.

Most of the collision tracks include both normal and collision images. Each of the tracks has been annotated image-by-image, marking each image as being either a normal image or a collision image. However, the test set is not balanced: there are more normal tracks than collision tracks. The motivation for this is the fact that collisions are rare events when compared to the number of normal traffic events. The suitability of a collision detection method cannot be accurately evaluated on an articially balanced set and the test setup should at least partially reflect the real world where most of the traffic is collision-free.

The obvious danger here is that a model can have fairly decent performance by predicting normal sequences all the time. Missing an occasional collision event would degrade the score of the model only marginally. For this reason, several metrics were used to help capture the overall performance of the model with both normal and collision events. The details of this approach are discussed next.

3.3 Performance evaluation and metrics

Measuring the performance of a collision detector is a problem in itself. For example, it is often not unambiguous when a collision can be said to have ended. Does a collision end when the participating vehicles are no longer in contact with each other? Or does a collision end when the participating vehicles have ceased to move?

From a traffic security point of view detecting the end of a collision is much less important than detecting the beginning. The sooner the observer can identify an event as a collision, the sooner it can take the needed measures such as quick maneuvering to avoid ending up as a participant in the crash.

Therefore, the test tracks were segmented to a maximum of two halves:

1. First segment containing normal images, and
2. second segment containing collision images (or no collision segment at all).

The performance of the collision detector was not evaluated past the annotated collision images. As an example, let us consider a track that is 15 images long and where a collision happens at the end.

Table 3.1: An example track with normal (0) and collision (1) images.

Track image	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1

Here, each number represents a single image in the track and is either a collision image or a normal image. Suppose a track like this is processed with the model CM:5-1. In that case, five images are needed before the model can make its first prediction. Additionally, an area identified as Gray Zone (G) should be considered, which is a sequence in the track where normal and collision images mix. It is also an area where the detection could go either way. In this kind of a scenario the track and the predictions are similar to what is presented in Table 3.2.

Table 3.2: A prediction example with Gray Zones (G) using a sequence-of-5 model. Colors indicate the starting images of sequences and where the corresponding prediction happens.

Track image	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
Moment of prediction					🟦	🟥	🟩	🟨	⬜	🟪	🟦	🟧	⬜	🟫	🟥
Correct predictions					0	0	0	0	0	0	0_G	0_G	1_G	1_G	1
Model's predictions					0	0	0	1	0	0	0	0	0	1	1
Model's score					1	1	1	0	1	1	1	1	0	1	1

The way in which a Gray Zone is scored depends on the number of normal and collision images. For the final metrics the following approach was selected:

- If a sequence starting at a gray zone G has more normal images it is correct to predict the sequence as a normal sequence. A collision detection is an incorrect detection.

- If a sequence starting at a gray zone G has more collision images it is correct to predict the sequence as a collision sequence. Not detecting a collision means a real collision is missed.

- In the case of sequence-of-10 models the prediction is omitted and not scored if a sequence starting at a gray zone G has an equal number of normal and collision images. The sequence is ambiguous, and no correct prediction can be made.

The list of metrics with which the models are evaluated is the following:

- Total, normal (non-collision) and collision accuracy.
- The number of true positives (TP): collision event detected as collision.
- The number of false positives (FP): normal event detected as collision.
- The number of true negatives (TN): normal event detected normal.
- The number of false negatives (FN): collision event detected as normal.
- F_1 score.

In accordance with the above metrics F_1 score is defined as:

$$F_1 = \frac{TP}{TP + \frac{FN+FP}{2}}. \tag{3.1}$$

3.4 Technical remarks

The models were trained on a Linux desktop with Intel Xeon 1230v3 (CPU), 20 GB of RAM and Nvidia 1080Ti 11 (GPU). At the time of writing this thesis it was not possible to run an end-to-end test of the entire framework due to multiple library incompatibilities. Particularly, the PyTorch-dependend and the TensorFlow-dependend methods refused to execute in the same environment. This was partly due to the unfortunate early decision to experiment with different frameworks, the repercussion of which was noticed too late.

As a result, the framework was split into two docker containers, each of them encompassing suitable library versions required by the tools contained. In order to pass information from the tracker to the collision detector an intermediate phase was needed, during which extracted track images were transferred from one container to another. The library configuration of the Docker containers was the following:

- Detectron2 / Deep SORT container: Cuda 10.2, cuDNN 7.6
- TensorFlow container: Cuda 11.2, cuDNN 8.1

4 Results

This chapter presents the results obtained by the framework. First, object detection and object tracking performance is briefly evaluated. This is followed by the results of the collision detector on the training set. The primary focus is on the last section, which presents the results on the test set obtained by the collision detector.

4.1 Preliminary checks

The performance of the first two phases of the framework, object detection and object tracking, was briefly evaluated in order to better understand their effect on the performance of the collision detector. This procedure was performed by manually checking how many of the collision events these methods managed to detect in the test set. The results are as follows:

1. Faster R-CNN successfully detected every colliding vehicle in the test set.
2. Deep SORT successfully tracked every colliding test set vehicle detected by Faster R-CNN.

The results do not mean that the models correctly detected and tracked every vehicle in every single collision frame correctly. Instead, they managed to a capture a a suitably long section of each collision track and no track was lost due to these methods. Therefore, the performance of the framework depends entirely on the collision detector.

The functionality of the fill-in algorithm in Listing 2 was assessed by inspecting some of the cases where a track of a vehicle was temporarily lost. Subjective observation confirmed that, when needed, the algorithm worked in an expected fashion and produced consistent results. Figure 4.1 demonstrates a fixed track produced by the algorithm.

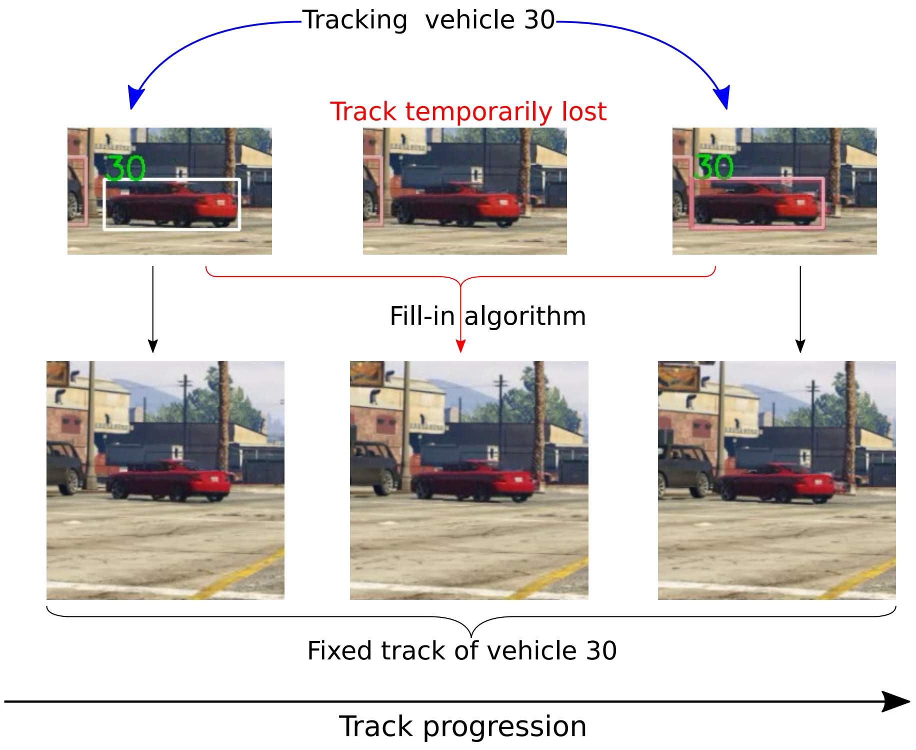


Figure 4.1: Real result by the fill-in algorithm in Listing 2.

The track extraction process was assessed by manually checking a selection of extracted tracks. Generally speaking, the process is able produce consistent results if the processes it relies on (the object detector and the object tracker) produce consistent results. The extraction process is not flawless, however. It is susceptible to sudden and drastic changes in the object detection process, which results in the extracted images exhibiting the same behavior.

Figure 4.2 is an example of this phenomenon. Here, the object detector incorrectly expands the bounding box of the tracked vehicle to include both the tracked vehicle and the other vehicle coming into contact. This error is reflected in the image extraction process in two ways:

- The enlargened bounding box causes the track image to be zoomed out.
- The tracked vehicle is no longer located in the center of the track image.

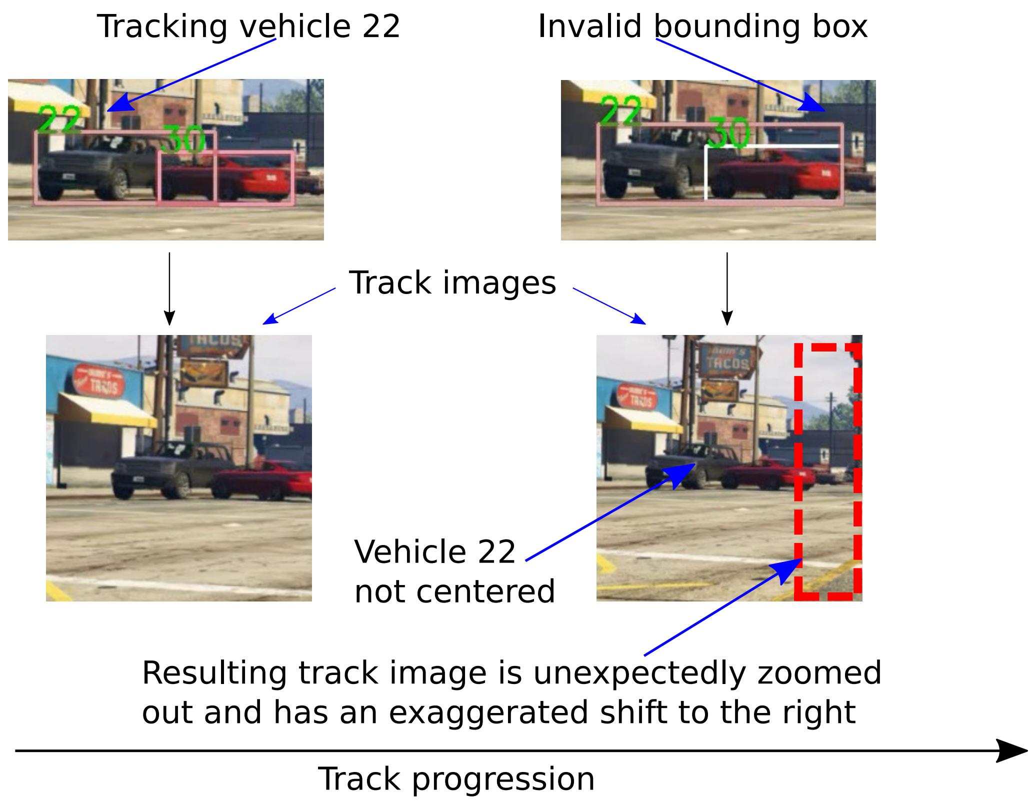


Figure 4.2: When tracking vehicle 22, a dramatic change in the bounding box (generated by Faster R-CNN) results as a dramatic change (dashed area) in the extracted track image.

As of now this behavior is an innate property of the framework and its effects cannot be evaluated separately. This means that the behavior of the track extraction process in irrevocably tied to the accuracy of the collision detector, as the detector receives whatever track images was extracted by the extraction process. Effectively, the results in sections 4.2 and 4.3 include this behavior.

4.2 Training set results

The models were first trained and validated with splitted training data in order to get a cursory impression of the performance. The purpose was not to provide an accurate benchmark but to get a feel as to what could be expected when evaluating the models with the test set. Additionally, the intention was to gain useful insights as to how the models should be trained with full training data, when no validation set would be available for monitoring the metrics during the training process.

For this section the training set was split by using 70 % of the data for training and 30 % for validation. The size of the acquired training set was increased with flipping augmentation. No augmentation was performed on the validation set. Preliminary tests suggested that input rescaling would not necessarily have the expected effect on performance, and therefore the training process was conducted both with and without input scaling. The results obtained with input scaling are shown in Figure 4.3.

Curiously enough all the models have very similar performance in terms of both accuracy and loss, albeit that the baseline model has considerably slower progression than the rest of the models. Accuracy on the training set comes very close to 100 % while validation accuracy is capped at approximately 70 %. A critical observation is that validation loss takes a turn for the worse soon after a model surpasses 90 % accuracy on the training set, indicating that some overfitting begins to take place. This is not unexpected, as collisions are a very complex group of events and the number and the variety of training samples is most probably lower than ideal.

The results were also computed without input scaling (Figure 4.4). The situation is strikingly similar between the rescaled and the non-rescaled variants, apart from slightly altered training progression. This suggests that the test set should also be evaluated both with and without rescaling.

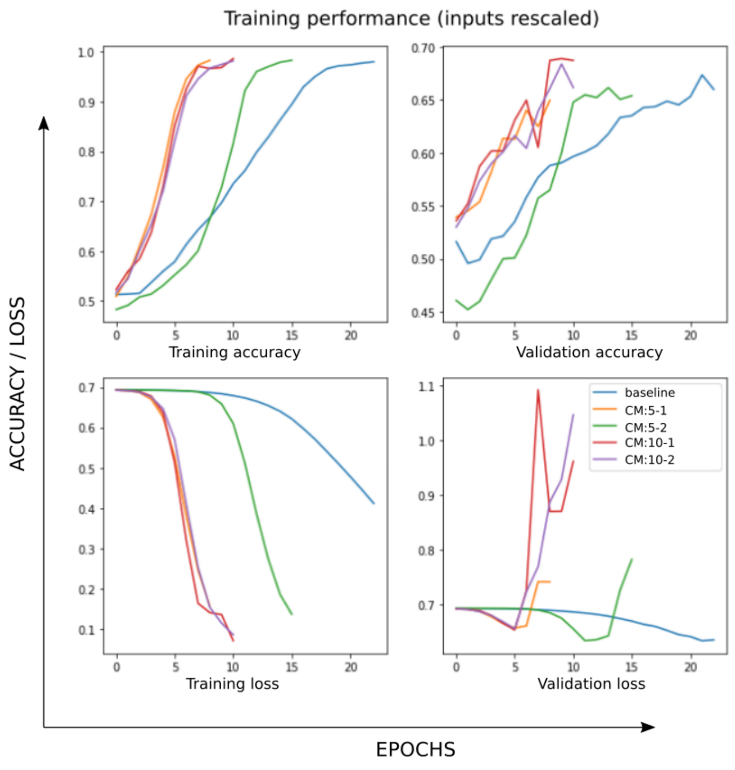


Figure 4.3: Training with *tanh* activation and rescaled inputs (70 % / 30 % split).

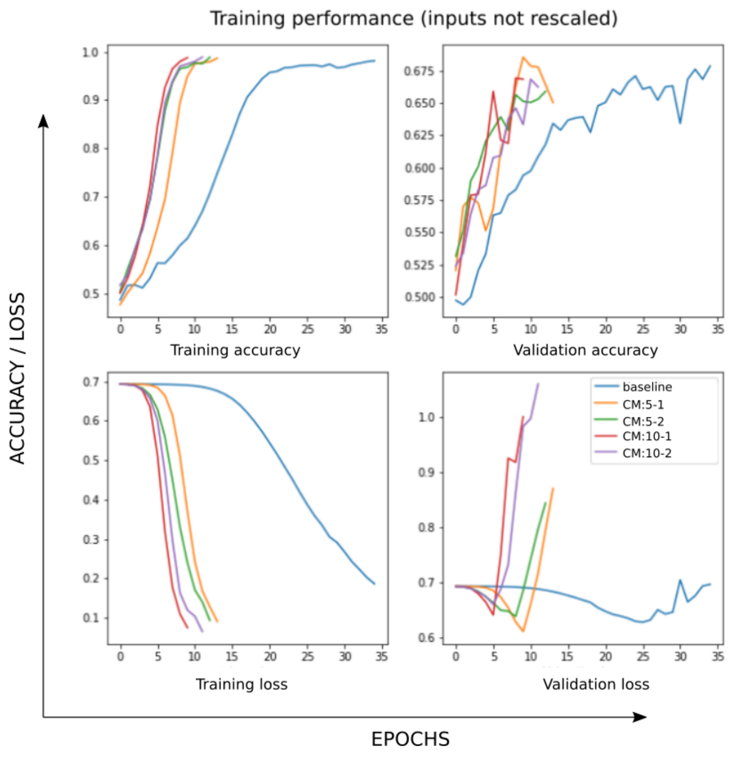


Figure 4.4: Training with *tanh* activation, no input rescaling (70 % / 30 % split).

4.3 Test set results

To achieve best possible performance on the test set the 70 % / 30 % split of the training set was revoked and the models were trained with the complete training dataset (100 % of the set used for training). In order to avoid overfitting, the number of epochs was limited by what was observed in section 4.2 with the consideration that the size of the training set was increased after dropping the validation split. In reality all the models were able to achieve a training accuracy of roughly 98 - 99 % after approximately 6 epochs, after which the training was stopped.

Earlier observations on the test set indicated some degree of variance related to model performance. That is, a specific model configuration could receive different test scores after two separate training iterations. Therefore, each model was fully trained and evaluated for ten repetitions, after which means and standard deviations (SD) of the scores were computed. To elaborate, the process was as follows:

1. Create a new model instance with a specific configuration.
2. Load pre-trained weights for the VGG16 backbone.
3. Train the model for a specified number of epochs.
4. Evaluate the performance of the model on the test set.
5. Store metrics.
6. Repeat phases 1 - 5 for 10 times in total.
7. Compute the mean and SD of every metric.

F_1 and accuracy results of the models are shown in Table 4.1.

Table 4.1: F_1 score and accuracy (acc.) over 10 full train/test cycles. Total accuracy is the accuracy on all the sequences that could be extracted from the data. Normal accuracy is accuracy on the normal sequences only. Collision accuracy is accuracy on the collision sequences only. All models utilize the *tanh* activation except where noted. Most consistent model in green.

	F_1		Total acc.		Normal acc.		Collision acc.	
Model	Mean	SD	Mean	SD	Mean	SD	Mean	SD
Without rescaling								
baseline (ReLU)	0.34	0.03	0.57	0.04	0.53	0.04	0.7	0.04
baseline	0.39	0.02	0.64	0.06	0.6	0.09	0.74	0.07
CM:5-1	0.43	0.01	0.65	0.03	0.63	0.04	0.77	0.04
CM:5-2	0.43	0.02	0.66	0.04	0.64	0.06	0.75	0.04
CM:10-1	0.39	0.03	0.71	0.03	0.74	0.07	0.59	0.11
CM:10-2	0.41	0.01	0.71	0.02	0.73	0.04	0.63	0.07
With rescaling								
baseline (ReLU)	0.37	0.02	0.65	0.05	0.66	0.08	0.64	0.06
baseline	0.38	0.01	0.64	0.03	0.65	0.04	0.67	0.04
CM:5-1	0.4	0.01	0.65	0.02	0.65	0.04	0.67	0.04
CM:5-2	0.39	0.01	0.64	0.02	0.64	0.03	0.68	0.03
CM:10-1	0.41	0.02	0.68	0.03	0.69	0.04	0.67	0.06
CM:10-2	0.41	0.01	0.69	0.03	0.69	0.05	0.67	0.08
CM:10-2 (ReLU)	0.4	0.02	0.63	0.04	0.6	0.06	0.77	0.04

A slightly unexpected result is that the CM:10-2 model without input rescaling seems to produce the most consistent results. Particularly, this model demonstrates good accuracy with least variance while sacrificing only a fraction of its F_1 score.

The temporal CM models outperform the baseline model at least in terms of accuracy. F_1 score is not notably better, though. Table 4.2 gives further insight into the matter. It describes how well the models do in terms true/false and positive/negative. Particularly interesting in the column FP/N which shows the ratio of collision detections on normal sequences; in other words, how eager a model is to interpret a normal sequence as a collision sequence.

Table 4.2: True (TP) / False (FP) Positives and True (TN) / False (FN) Negatives divided by the total number of positive (collision) sequences (P) or the total number of negative (normal) sequences (N). Mean values computed over 10 full train/test cycles. All models utilize the *tanh* activation except where noted.

Model	TP/P	TN/N	FP/N	FN/P
	Higher is better		Lower is better	
Without rescaling				
baseline (ReLU)	0.70	0.53	0.47	0.30
baseline	0.74	0.60	0.40	0.26
CM:5-1	0.77	0.64	0.37	0.23
CM:5-2	0.77	0.64	0.36	0.25
CM:10-1	0.59	0.74	0.26	0.41
CM:10-2	0.63	0.73	0.27	0.37
With rescaling				
baseline (ReLU)	0.64	0.66	0.34	0.36
baseline	0.67	0.65	0.35	0.33
CM:5-1	0.67	0.65	0.35	0.33
CM:5-2	0.68	0.64	0.35	0.32
CM:10-1	0.67	0.69	0.31	0.33
CM:10-2	0.67	0.69	0.31	0.33
CM:10-2 (ReLU)	0.77	0.60	0.40	0.23

It seems that the baseline model is able to reach a relatively good F_1 score by tilting very heavily towards collision sequences: the baseline model alternatives have a high number of false positives, that is, they often see collisions where there are none. Admittedly similar behavior is present in the temporal CM models also, but the model CM:10-2 gives the least number of false positives while not sacrificing much of the other metrics.

The overall observation is that all the model CM:10-2 seems to be best in tolerating the noise in the dataset and produces consistent results across training iterations. Also, rescaling has surprisingly little effect, which goes against the common convention of the trade. However, the *ReLU* activated baseline suffers massively from non-scaled inputs, which seems to indicate that the effectiveness of rescaling depends highly on the model configuration.

In general, all the temporal CM models outperform the baseline model but have very similar performance when compare with each other. This is consistent with what was observed during the validation phase in the previous chapter. It is also noteworthy that the baseline model is not without its merit and has a performance fairly close to the other models. In other words, the baseline model does not perform considerably worse than the other models.

The above observations are an indication of the fact that the single most important feature is a good static image, regardless of all the temporal aspects discussed in this thesis. Temporal aspects do increase the performance of the model and particularly work to reduce the number of incorrect collision detections, but they cannot surpass the significance of a single image that is a good representation of a collision.

Examples of two processed real-life test tracks are seen in Figure 4.5. Each track is approximately 100 frames long and split into sequences of suitable length. The framework processes the sequences, classifying every sequence it sees as either a normal (0) or a collision (1) sequence. The shown images are selected evenly throughout the track and the numbers above the images indicate both the truths and the detections in the vicinity (before and after) of the displayed image.

For example, the string *True 000...111* above the image 067 in the upper row indicates that there are no collision sequences prior to the image, but a collision happens soon after that image is passed. The more similar the *Pred* string is to the *True* string, the better the score of the model in the neighborhood of that image.

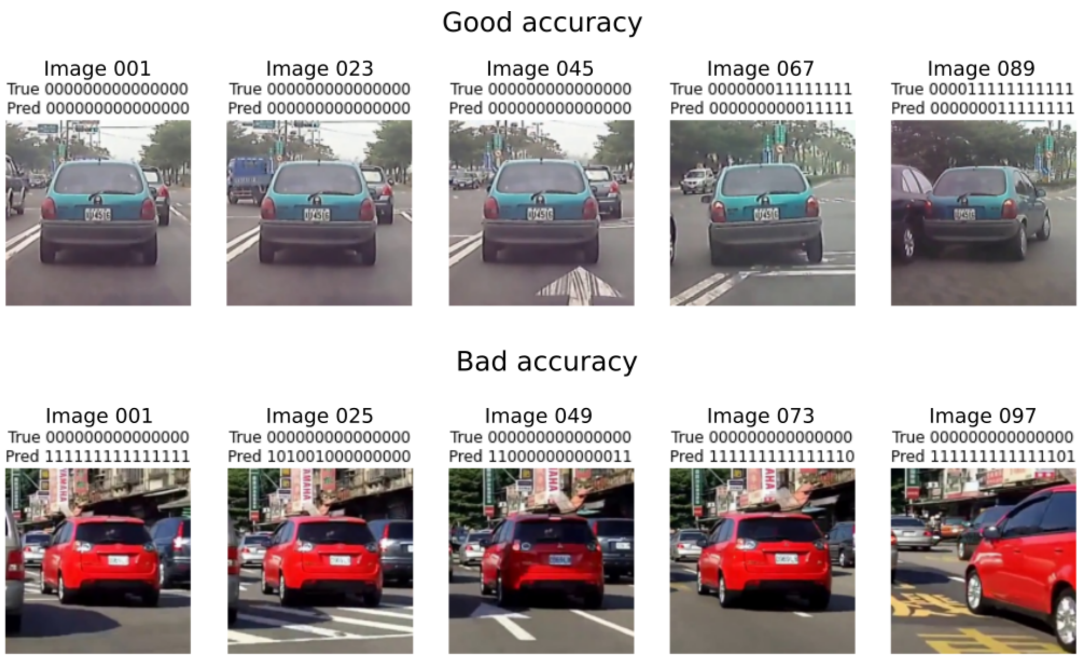


Figure 4.5: Example tracks with truths (True) and corresponding predictions (Pred) by CM:10-2 (without input scaling). Normal (0) and collision (1) sequences in the vicinity of the shown image. Example images are selected evenly throughout the track (original videos from [8]).

In the case of the upper track the model performs fairly well. The detection happens a few frames too late, but other than that the model seems to have a reasonably consistent understanding of the track. On the other hand, the lower track highlights a situation in which the model is very wrong and detects numerous collision sequences when there is in fact no collision happening in the track. The poor performance is a combination of many things, but the numerous occluded vehicles is one of the contributing factors.

Finally, a subjective evaluation of track characteristics affecting detection performance was conducted. A rough summary is displayed in Table 4.3. If the track demonstrates a quality listed in the *Better performance* column, then it is more likely the detector receives a better score. On the other hand, characteristics better fitted in the *Worse performance* column usually lead to degraded performance.

Table 4.3: Subjective observation on track characteristics and performance.

	Better performance	Worse performance
Nature of collision	**Obvious**	**Subtle**
Point of contact	**Visible**	**Occluded**
Number of visible vehicles	**Less**	**More**
Occluded vehicles	**Less**	**More**

The results in Table 4.3 have the same characteristics that should make a collision more difficult to detect for a human observer also: a violent high-speed crash is more easily identifiable as a collision than a minor nearly undetectable bump.

5 Discussion and conclusions

This chapter focuses on providing a conclusion for the thesis. The chapter is opened by a short discussion about what was attempted and what was achieved. Next, some initial thoughts about possible future avenues for continued research are suggested. Finally, a conclusive summary is provided in the ending section.

5.1 Discussion

Discussion of the achieved results is best approached by considering the initial research questions and the research hypothesis.

Research question 1.

Are modern superficially lifelike virtual environments visually and physically accurate enough to be utilized in developing deep learning models that can detect vehicle collisions happening in reality?

It seems plausible that modern computer vision techniques, initially designed for and trained on real-life data, are indeed able to process synthetic data also. This deduction stems from the observartions by which:

1. Object detection techniques can detect virtual counterparts of real objects in synthetic data.

2. Object tracking techniques can track virtual counterparts of real objects.

3. Collision detector, when trained on synthetic data, using a backbone initially trained on real data, is able to detect at least some of the collisions happening in real-life footage.

All these facts show that the feature sets and characteristics of real and virtual events are not entirely disjoint. Had any of the mentioned bullet points failed, the synthetically trained framework would have also failed to make any sense at all about the accidents in the dashcam dataset.

Research question 2.

Are there specific techniques that translate well between real and virtual environments in a vehicle collision context and how do these techniques perform when evaluated using real collision data?

All the phases on the presented framework rely on some form of transfer learning, and all underlying utilities were initially trained with real data. This is true for Faster R-CNN as the object detector, Deep SORT as the object tracker and VGG16 as the backbone of a collision detector.

There were no actual earlier results with which to compare a framework such as the one proposed in this thesis, and it is therefore difficult to make a definite judgment if the framework has reasonable performance or not. The consistent total accuracy was around the 70 % mark when evaluated with real-world data.

Research hypothesis

When trained on synthetic collision footage and tested on real-life footage, the performance of a suitable model surpasses that of a trivial baseline.

In this case the trivial baseline was an image classifier fine-tuned with collision images and similar non-collision images. The baseline acted by selecting a specific image from a sequence of images, relying only on that single image when trying to decide if the observed vehicle was experiencing a collision or not. This means that the baseline discarded all notion of movement and temporality.

All of the presented collision models were able to surpass the performance of the baseline. That is, a model utilizing visual information arranged in a temporal fashion was able to outperform a baseline that relied only on one static image. On the other hand, the performance of the baseline was not strikingly worse than any of the other models: in terms of accuracy the gap was less than 10 %. This suggests that a single static image was the most contributing factor in the detection and the temporal aspects were secondary.

The models suffered from a level of variance meaning that a model with the same configuration produced different results after two different training iterations. The models also suffered from a high number of false positives (i.e., they incorrectly identified normal sequences as collision sequences). The detailed structure of the model affected the efficacy of the model, and typically a more complex model provided more consistent results than a simple model. A more complex model also performed better in terms of false positives.

5.2 Future works

Future work on this topic can be broadly divided into two categories: speed improvements and accuracy improvements.

Speed improvements

The presented framework is surely not the fastest possible and many implementation decisions are based more on convenience than on efficiency. Particularly, utilizing a one-stage object detector in place of a two-stage detector would surely result in a framework with more reasonable inference times. The essential measure of speed for any framework is the time from observation to detection. In the current context this means the time it takes for the model to detect a vehicle in a video, track it for a required time and then notice a collision if the vehicle is experiencing one. Unfortunately, due to issues mentioned in section 3.4 it was not possible to compute such statistics, at least without considerable investment in getting the models in different containers to communicate in real-time. This was outside of the scope of this thesis, however.

Accuracy improvements

It is obvious that the collision detection framework is not ready for any real-world environment in its current state. An accuracy of 70 % is not an approvable score for a model from which near 100 % level of performance is required. There are surely many available avenues with which the accuracy could possibly be improved. These include, but are not limited to:

1. Better training data that is a better representation of real-world situations.
2. More expressive models that are better able to detect subtle movement hinting at a collision.
3. Utilization of untrainable features such as optical flow.

5.3 Conclusion

This thesis has presented an approach with which synthetic data can be used to generate models that have better understanding of collision happening in the real world. The suggested framework consists of three phases. The first phase is responsible for object detection, that is, detecting vehicles in a video footage. The second phase is responsible for tracking the objects long enough so that their track of movement can be established. The final phase performs the actual collision detection based on the results provided by the preceding phases.

The results indicate that the approach has at least some merit. The accuracy of the proposed collision detection models surpasses that of a trivial baseline. However, a maximum accuracy of roughly 70 % on real-life data means that the model as such is not suitable for a production environment, and more development would be needed in order to use a framework such as this for actually detecting collisions in any real environment.

References

[1] J. Leudet, F. Christophe, T. Mikkonen, and T. Mannisto, “Ailivesim: An extensible virtual environment for training autonomous vehicles”, in *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, Los Alamitos, CA, USA: IEEE Computer Society, Jul. 2019, pp. 479–488. DOI: 10.1109/COMPSAC.2019.00074. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/COMPSAC.2019.00074.

[2] A. Blade, *Script Hook V*, Mar. 2021. [Online]. Available: https://www.dev-c.com/gtav/scripthookv/.

[3] V. Machaca Arceda and E. Laura Riveros, “Fast car crash detection in video”, in *2018 XLIV Latin American Computer Conference (CLEI)*, 2018, pp. 632–637. DOI: 10.1109/CLEI.2018.00081.

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

[5] Z. Lu, W. Zhou, S. Zhang, and C. Wang, “A new video-based crash detection method: Balancing speed and accuracy using a feature fusion deep learning framework”, *Journal of Advanced Transportation*, vol. 2020, p. 8 848 874, Nov. 2020, ISSN: 0197-6729. DOI: 10.1155/2020/8848874. [Online]. Available: https://doi.org/10.1155/2020/8848874.

[6] A. P. Shah, J.-B. Lamare, T. Nguyen-Anh, and A. Hauptmann, “Cadp: A novel dataset for cctv traffic camera based accident analysis”, in *2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, 2018, pp. 1–9. DOI: `10.1109/AVSS.2018.8639160`.

[7] F.-H. Chan, Y.-T. Chen, Y. Xiang, and M. Sun, “Anticipating accidents in dashcam videos”, in *Computer Vision – ACCV 2016*, S.-H. Lai, V. Lepetit, K. Nishino, and Y. Sato, Eds., Cham: Springer International Publishing, 2017, pp. 136–153, ISBN: 978-3-319-54190-7.

[8] F.-H. Chan, *Anticipating accidents in dashcam videos*, Mar. 2017. [Online]. Available: `https://aliensunmin.github.io/project/dashcam/`.

[9] H. Kim, K. Lee, G. Hwang, and C. Suh, “Crash to not crash: Learn to identify dangerous vehicles using a simulator”, *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, pp. 978–985, Jul. 2019. DOI: `10.1609/aaai.v33i01.3301978`. [Online]. Available: `https://ojs.aaai.org/index.php/AAAI/article/view/3887`.

[10] A. Géron, *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O’Reilly Media, Aug. 2019, ISBN: 1492032646.

[11] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition”, in *International Conference on Learning Representations*, 2015.

[12] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation”, *CoRR*, vol. abs/1311.2524, 2013. arXiv: `1311.2524`. [Online]. Available: `http://arxiv.org/abs/1311.2524`.

[13] R. Girshick, “Fast r-cnn”, in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1440–1448. DOI: 10.1109/ICCV.2015.169.

[14] S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-cnn: Towards real-time object detection with region proposal networks”, *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 6, pp. 1137–1149, Jun. 2017, ISSN: 0162-8828. DOI: 10.1109/TPAMI.2016.2577031. [Online]. Available: https://doi.org/10.1109/TPAMI.2016.2577031.

[15] A. Rosebrock, *Simple object tracking with opencv*, Jul. 2018. [Online]. Available: https://www.pyimagesearch.com/2018/07/23/simple-object-tracking-with-opencv/.

[16] B. Schiele, “Model-free tracking of cars and people based on color regions”, eng, *Image and vision computing*, vol. 24, no. 11, pp. 1172–1178, 2006, ISSN: 0262-8856.

[17] W. Luo, J. Xing, A. Milan, X. Zhang, W. Liu, and T.-K. Kim, “Multiple object tracking: A literature review”, *Artificial Intelligence*, vol. 293, p. 103 448, 2021, ISSN: 0004-3702. DOI: https://doi.org/10.1016/j.artint.2020.103448. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0004370220301958.

[18] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, “Simple online and realtime tracking”, in *2016 IEEE International Conference on Image Processing (ICIP)*, 2016, pp. 3464–3468. DOI: 10.1109/ICIP.2016.7533003.

[19] Y. Kim and H. Bang, “Introduction to kalman filter and its applications”, in *Introduction and Implementations of the Kalman Filter*, F. Govaers, Ed., Rijeka: IntechOpen, 2019, ch. 2. DOI: 10.5772/intechopen.80600. [Online]. Available: https://doi.org/10.5772/intechopen.80600.

[20] A. Rosebrock, *Intersection over union (iou) for object detection*, Nov. 2016. [Online]. Available: https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/.

[21] H. W. Kuhn and B. Yaw, "The hungarian method for the assignment problem", *Naval Res. Logist. Quart*, pp. 83–97, 1955.

[22] N. Wojke, A. Bewley, and D. Paulus, "Simple online and realtime tracking with a deep association metric", in *2017 IEEE International Conference on Image Processing (ICIP)*, 2017, pp. 3645–3649. DOI: 10.1109/ICIP.2017.8296962.

[23] "Mahalanobis distance", in *The Concise Encyclopedia of Statistics*. New York, NY: Springer New York, 2008, pp. 325–326, ISBN: 978-0-387-32833-1. DOI: 10.1007/978-0-387-32833-1_240. [Online]. Available: https://doi.org/10.1007/978-0-387-32833-1_240.

[24] G. Bradski, "The OpenCV Library", *Dr. Dobb's Journal of Software Tools*, 2000.

[25] *Numpy.ndarray*, Jan. 2021. [Online]. Available: https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html.

[26] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, and R. Girshick, *Detectron2*, https://github.com/facebookresearch/detectron2, 2019.

[27] M. R. Shishira, *Object tracking*, 2021. [Online]. Available: https://github.com/abhyantrika/nanonets_object_tracking.

[28] M. Abadi, A. Agarwal, P. Barham, *et al.*, *TensorFlow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015. [Online]. Available: http://tensorflow.org/.

[29] *tf.keras.layers.TimeDistributed | TensorFlow Core v2.4.1*, Mar. 2021. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/layers/TimeDistributed.

[30] *tf.keras.applications.vgg16.preprocess_ input | TensorFlow Core v2.4.1*, Mar. 2021. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/applications/vgg16/preprocess_input.

[31] *tf.keras.optimizers.SGD | TensorFlow Core v2.4.1*, Mar. 2021. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/SGD.