Goethe University Frankfurt am Main

Department of Computer Science and Mathematics

Towards Explainable AI Systems for Traffic Sign Recognition and Deployment in a Simulated Environment

Project Report

Systems Engineering Meets Life Science

Pascal Fischer and Alen Smajic

Supervisor 1: Prof. Dr. Visvanathan Ramesh

Supervisor 2: Dr. Michael Rammensee

18th April 2021

Abstract

Recent advances in Machine Learning are pushing the boundaries of real world AI applications and are becoming more and more integrated in our daily lives. As research advances, the long-term goal of autonomous driving becomes rather reality than fiction, but brings with it numerous challenges for our society and the system engineers. Topics like ethical AI, privacy and transparency are becoming increasingly relevant and have to be considered when designing AI systems with real world applications. In this project we present our first attempt at tackling the problem of traffic sign recognition by taking a systems engineering approach. We focus on the development of an explainable and transparent system with the use of concept whitening layers inside our deep learning models, which allow us to train our models with handcrafted features and make predictions based on predefined concepts. We also introduce a simulation environment for traffic sign recognition, which can be used for model deployment, performance benchmarking, data generation and further domain model research.

Contents

1 User View

Autonomous driving is one of the most anticipated technologies of the 21st century and one of the most active research topics at the moment. Autonomous driving attempts navigating roadways without human intervention by sensing and reacting to the vehicle's immediate environment [1]. It includes major challenges for Computer Vision and Machine Learning with a great focus on explainable AI. One important aspect of the vehicle's environment are traffic signs, which highly influence the decision making process when driving a car. The problem of traffic sign recognition becomes more and more important as we move towards stage 4 "highly automated" and stage 5 "fully automated" car driving, since there is no human supervision anymore. In this chapter, we want to give you an overview over the context of traffic sign recognition and formally describe our task. Furthermore, we want to give an overview over the different performance constraints, which have to be fulfilled in order to enable autonomous real-time traffic sign recognition with explainable deep learning models.

1.1 Context of Traffic Sign Recognition

Traffic signs play an important role in our road infrastructure and provide the driver with critical information like current road regulations, which in fact alters the driver's behaviour on the roads and ensures the safety of all road users. Therefore, a good visibility and distinguishability is highly crucial and may prevent potential car accidents [2]. For this purpose, the physical appearance of traffic signs is pretty unique and contains some beneficial characteristics for deploying traffic sign recognition algorithms [3]. In most cases, traffic signs are rigidly positioned in clear sight to the driver at different height positions on the right side of the road, but can sometimes also be located on the left road side or above the middle of the road. They are mostly placed on traffic sign poles but can also be located on other structures like traffic lights, buildings, walls etc. Traffic signs can also appear as a single sign or a group of different signs which are placed next to each other in good sight. The signs itself contain a very small object variation and their colors are very well contrasted against the environment [3]. They come in different sizes and scales and contain different colors, shapes and symbols, which also differ depending on the legislative countries they are located in. Many traffic signs have the problem that if you occlude some parts of the sign, it can represent more than one class which means that they are often very similar. They consist of reflective materials which ensures good visibility at low lighting conditions if there is any illumination positioned in their direction. The biggest problem in traffic sign recognition is the outdoor environment which highly influences the visibility and physical appearance of traffic signs. Weather and lighting conditions vary significantly and even the sign installation itself can physically change over time due to accidents, vandalism or weather influence which can lead to sign occlusion, sign rotation, damaged sign surface and degenerated colors [3]. It is also important to

mention that there is another class of traffic signs, namely the digital variable signs which are mostly located on highways. We limit ourselves to recognition of non digital traffic signs and leave these for future work.

1.2 Task

Our task consists of developing a system which recognizes traffic signs in real-time by using a camera sensor mounted on a driving car. To extract the traffic signs from the scene, we need to consider using a two stage pipeline which consists of a traffic sign detection algorithm in the first stage and a traffic sign classification algorithm in the second stage. The traffic sign detection model should generate detection proposals which need to be resolved by the traffic sign classification model. Furthermore, we want to develop a robust system with focus on transparency and explainability, which means we want to be able to explain certain system decisions and the features it uses for predictions. The system should be able to recognise different traffic signs in various conditions and environments as described in the context part. For this purpose we constrained ourselves only to German traffic signs, without loss of generality. Furthermore, we also constrained our task to cover only the recognition of traffic signs in front of the car and leave the further processing of this information (like estimating the road area where the sign is valid) for future work. To support our model decisions and to validate our performance, we also want to develop a realistic simulation which visualizes the contextual information of traffic sign recognition and gives us the opportunity to deploy our system inside a realistic car driving environment.

1.3 Performance Constraints

Since the traffic sign recognition should be used in a driving car environment, our system must perform predictions in real-time to deliver the information right in time in order to take the appropriate action. The recognition and classification of a traffic sign must be done at least as fast as for a human driver. Furthermore, the system needs a very high prediction performance with high accuracy. To achieve this, we want our traffic sign detection model to have high recall and are willing to sacrifice some precision. The traffic sign detection model must also be able to detect traffic signs on great distances. In the next step we want a very robust and accurate traffic sign classification algorithm which resolves the proposals from the detection model and can also perform well on false positive proposals. Our goal is to develop a system which outperforms a human driver given the same conditions.

2 Modeling View

To better understand the problem of traffic sign recognition, we derived a contextual world model which focuses on the invariances that we can exploit in our system design. This starts with the sensor modeling, which is particularly important for capturing the input information and highly influences the further processing steps. In the next step we analysed the physical appearance of traffic signs in their natural environment and the differences between the respective sign classes. For this purpose, we tried to describe a set of concepts which we want to be used for prediction by our model. Furthermore, we derived a probabilistic world model to get a better understanding of the environmental influence on traffic sign recognition. Finally, we combined all this information into a modular estimation pipeline from which we derived our system design. We also used the contextual world model to get a rough idea, in which direction we want to develop our traffic sign simulation. In the last section, we give some instructions on how to implement our system with respect to privacy and transparency.

2.1 Contextual World Model

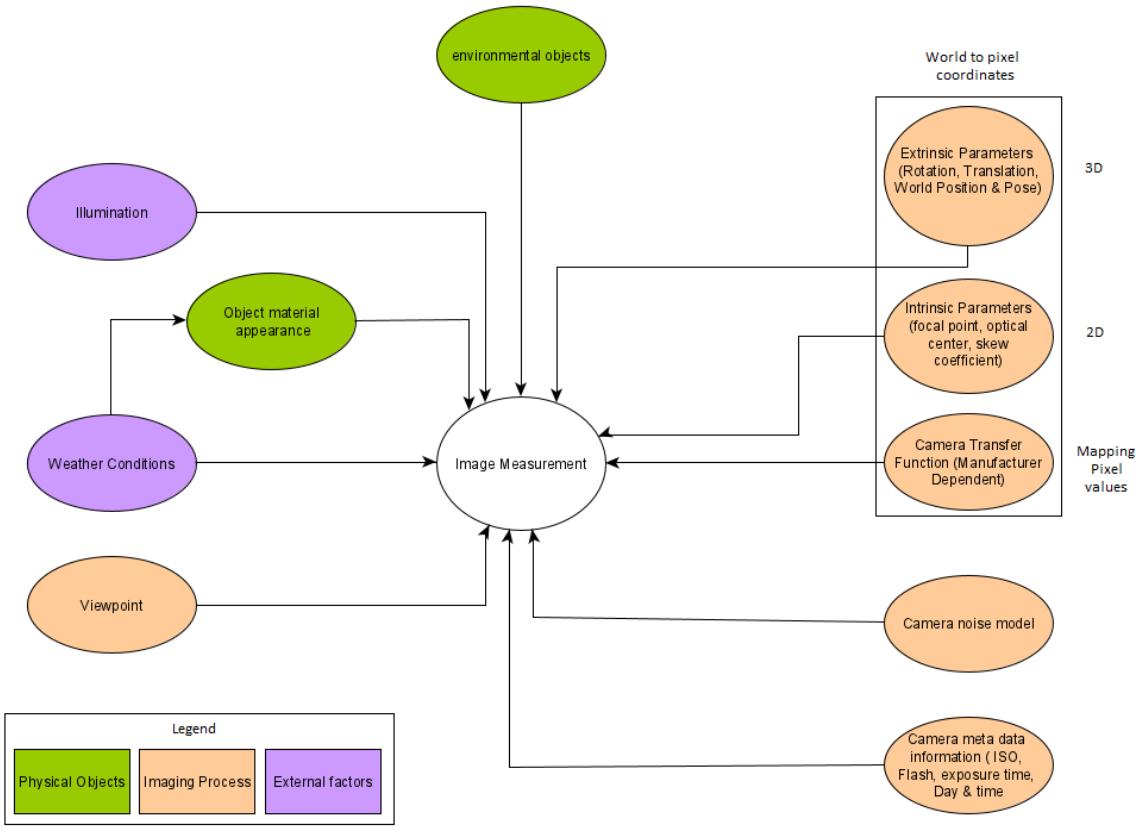


Figure 2.1: Rough visualisation of our contextual world model

Our world model 2.1 is centered around the image measurement which is being captured by a camera sensor. We differentiate between 3 different types of sources, which influence the image measurement. The first one are the physical objects including the traffic signs, which are positioned somewhere in the real world. These objects consist of different surface materials, which are visible to the camera sensor. The second type of sources are the external factors, which consist of the weather condition and illumination. It is also important to mention that the weather condition can change the physical appearance of some object materials due to weather exposure. The last source of influence is the imaging process itself, which highly influences the image measurement based on different factors. These factors are mainly related to the choice of a specific camera sensor and its position in the real world.

2.1.1 Camera Sensor Model

Since we want to detect and classify traffic signs, our system needs to use a camera sensor to capture images of its environment. The sensor should be mounted on the car roof facing to the front, where it has the best visual overview of the environmental scene and the least possibility of being occluded by other objects like other cars. The sensor must also have a big field of view to reduce the space of blind spots and offer the best possible viewpoint. Figure 2.2 offers a visualisation of this problem. We leave this task of determining the optimal camera position and field of view to be later determined by testing and evaluating the different viewpoints inside our simulation. Further

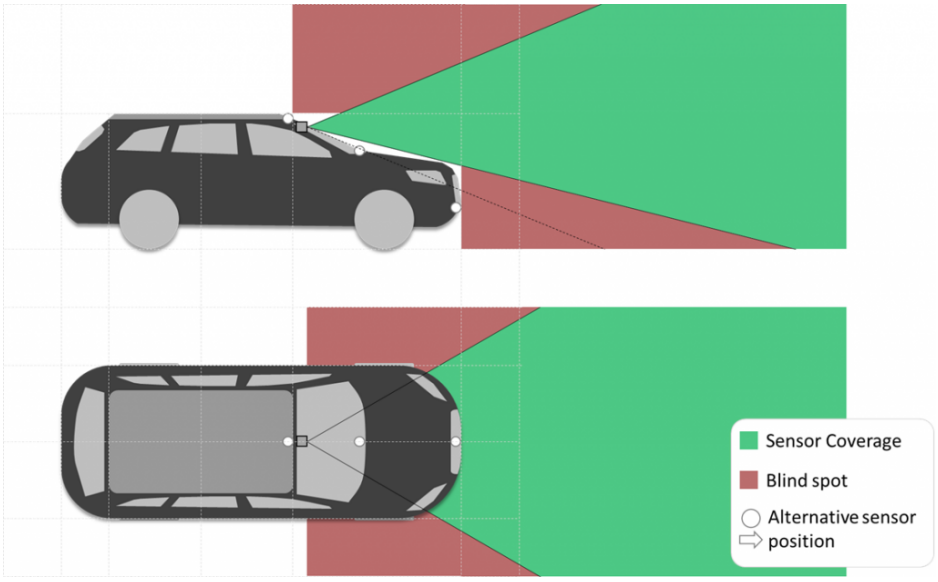


Figure 2.2: Front camera positioning visualisation [4]

requirements for our camera sensor include high resolution images and robustness to camera movement. The latter requirement is particularly important, since the camera used for capturing the traffic signs will be moving along with the car which can lead to abrupt contrast images and image distortions due to motion blur [3]. This requirement also includes robustness in suboptimal driving conditions like uneven roads, where the camera is disposed to vehicle shaking. Furthermore, we require our camera sensor to perform well in low illumination environments, where it only has the head beam lights

of the car at its disposal. At the same time we require our sensor to be robust against strong illumination facing in the direction of the camera (like the head beam lights of other cars), which means that our camera sensor needs to make clear high resolution images at any given point under any external condition. We constrain ourselves to these camera specification for this project iteration and leave further camera modeling aspects for future work

2.1.2 The Traffic Sign Domain

Traffic signs are part of the road equipment and are positioned by the local authorities. All details on regulations regarding traffic signs in Germany can be found in the "Verwaltungsvorschrift zur Straßenverkehrsordnung" (VwV-StVO) [5]. In the following sections we will give an overview over the most important regulations regarding traffic signs. For this project iteration we will limit ourselves to the 43 traffic signs covered by the German Traffic Sign Recognition Benchmarks dataset (GTSRB) [6]. In Figure 2.3 you can get a glimpse at all the signs, that are covered by the GTSRB dataset.

Figure 2.3: Selected traffic signs covered by the GTSRB

2.1.2.1 Regulations on the Physical Appearance of Traffic Signs

Sign Appearance

Traffic signs are permanent unless temporarily installed. Posts, frames and sign backs are regulated to be of gray color. On the traffic signs itself, the sans serif font from the Linear-Antiqua family according to DIN 1451-2 is used. There are three different sizes for each traffic sign, depending on its geometric shape.

traffic sign shapes	size 1 (70 %)	size 2 (100 %)	size 3 (125 or 140 %)
circle (diameter)	420	600	750 (125 %)
triangle (side length)	630	900	1260 (140 %)
square (side length)	420	600	840 (140 %)
rectangle (W x H)	630 x 420	900 x 600	1260 x 840 (140 %)

Table 2.1: Overview of traffic sign sizes and shape types

As a rule, the size of a speed limit sign depends on the maximum permitted speed at the place of installation.

speed limit (km/h) on circular shape	speed limit (km/h) on other shapes	size
0 to 20	20 to 50	1
20 to 80	50 to 100	2
more than 80	more than 100	3

Table 2.2: Overview of speed limit signs and sizes

Placement

A maximum of three traffic signs, of which a maximum of two may be regulatory signs, may be attached to a single post. However, there are exceptions for signs for stationary traffic. Danger signs and signs with waiting obligation must be placed alone.

Traffic signs are in general placed at clearly visible locations on the right side of the road.

They may be placed on the left alone or above the road alone only if there can be no misunderstanding that they apply to all traffic in one direction and if it is ensured that they are clearly visible at a sufficient distance even in the dark.

Where necessary, especially at particularly dangerous road locations, traffic signs may be placed on both sides of the road, or on both sides of the road in the case of separated lanes.

Traffic signs may be curved in such a way that they can also be seen from the side if this appears to be required and if their appearance does not mislead other road users. However, this does not apply to signs regulating the right of way.

Several traffic signs or a traffic sign with at least one additional sign may be placed together on a white support plate. The supporting panel has a black border and a white contrasting stripe. Additional signs are each enclosed by an additional black border. Individual traffic signs may only be affixed to a carrier panel if improved perceptibility is to be achieved due to unfavorable ambient conditions.

The lower edge of traffic signs should generally be 2m above street level, 2.20m above cycle paths, 4.50m on gantries, 0.60m on islands and traffic dividers, unless otherwise stated for individual signs.

Traffic signs must not be placed within the roadway. As a rule, the lateral distance from it should be 0.50m within built-up areas and never less than 0.30m. For outside built-up areas it should be 1.50m.

Color

The colors used for traffic signs are regulated by the DIN 6171 „Aufsichtsfarben für Verkehrszeichen, Farben und Farbgrenzen bei Beleuchtung mit Tageslicht". The color charts „RAL-F 7 Reflexfarben" and „RAL-F 81 Farben im Straßenverkehr" show the colors of traffic signs in new condition. (https://www.ralfarbpalette.de/)

The color shades specified in the RAL-F 7 special color series are: RAL 2006, RAL 3019, RAL 3030, RAL 5016, RAL 6030, RAL 8026, RAL 9014, RAL 9019

The color shades specified in the RAL-F 81 special color series are: RAL 1023 (yellow), RAL 2009 (orange), RAL 3020 (red), RAL 4006 (purple), RAL 5017 (blue), RAL 6024 (green), RAL 7042 (gray A), RAL 7043 (gray B), RAL 9016 (white), RAL 9017 (black)

Other colors used for traffic signs are RAL 1003 (signal yellow) used for signs 306/307 (priority road), and RAL 5005 (signal blue) used for directional signs.

2.1.2.2 Conceptual Decomposition of Traffic Signs

It is in the nature of concepts to consist of further concepts, which can be described as a "part of" hierarchy. A concept on itself is therefore a partonomy which leads to the fact that the conceptual decomposition of traffic signs is ambiguous. The conceptual decomposition can therefore not be clearly determined, and depends much more on the observer.

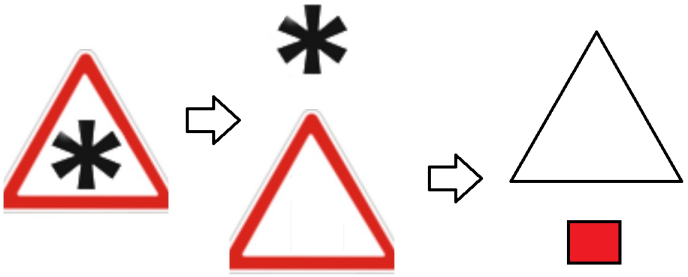


Figure 2.4: Example of a conceptual decomposition hierarchy

As an example, the traffic sign "Snow or ice" (shown in Figure 2.4) is a concept for itself, but it could be seen as the combination of the two concepts: "white equilateral triangle with red border" and "snowflake". The "white equilateral triangle with red border" could be divided in the geometric concept "triangle" and the color concept "red".

The conceptual hierarchy for traffic signs is shown in Figure 2.5. The surface of each traffic sign consists of either one of the following geometric forms: equilateral octagon, equilateral triangle, equiangular rhombus or circle. The octagon is always red, the rhombus is yellow with a white border, the triangles and some circles are white with a red border and there are also blue circles and white circles with four black diagonal lines. On the surface can appear symbols like the numbers 0,1,2,3,5,6,7,8 or the characters "S", "T", "O", "P" or punctuation marks like "!". On blue circles there

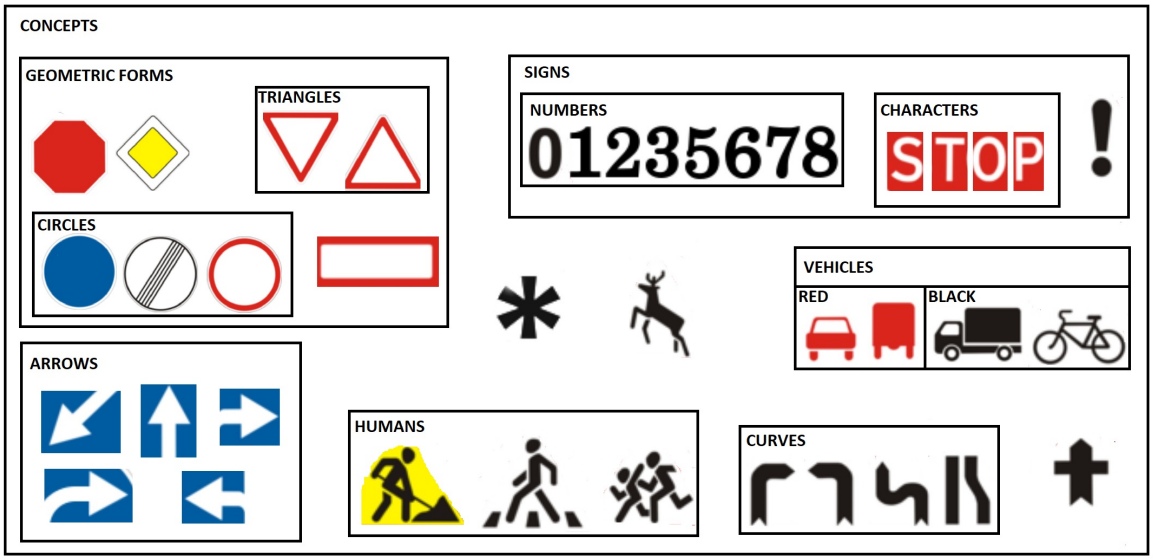


Figure 2.5: Conceptual Decomposition Model

are always arrows which can be pointing in any direction. On top of that, the surface can show some vehicles symbols like a truck or a car in red color and a truck or a bicycle in black color. Furthermore, there are three types of human symbols and four types of curves. Finally, the concept of a deer, a snowflake and a crossing symbol can be described as individual concepts.

2.1.3 Probabilistic Environmental World Model

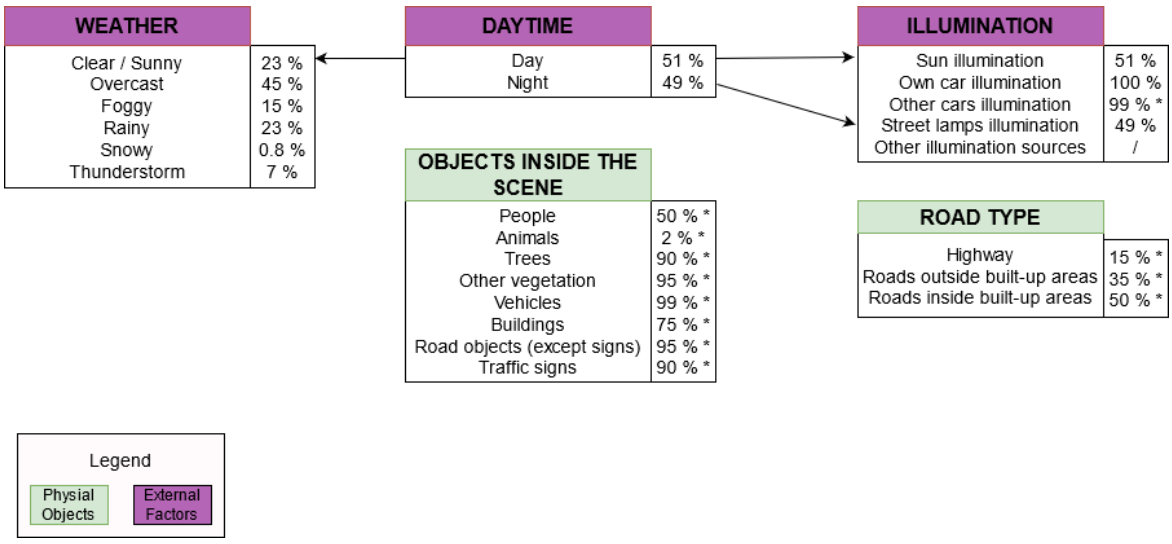


Figure 2.6: Probabilistic World Model (probabilities marked with a * were approximated) all probabilities are formed on a daily driving basis within Germany and the city of Frankfurt (source: [7], [8], [9], [10])

Our probabilistic environmental world model is based on the objects and illumination sources that are present in the image measurements taken by the camera sensor. One of the most important environmental aspects is the daytime, which highly influences the illumination of the scene and most of other environmental variables. Note that

the daytime influence on other environmental variables is not directly visualized inside Figure 2.6 but can be derived by correlation (this applies also for other external factors). Furthermore, we can say that the weather also plays an important role and also influences the illumination of the scene and can have a big impact on the physical appearance of traffic signs and other objects. Unlike the daytime, which more or less only influences the scene illumination of the image measurement, the weather can also have a huge impact on the quality of the image. With respect to these environmental variables, we derived different kinds of illumination sources which are present in the camera scene. It should be mentioned that there are many different illumination sources and we only covered the most important and frequent ones inside our model. Finally we divided our physical objects description into road types and objects that are present inside the scene. These represent most of the image measurement and have to be distinguished from the traffic sign objects we want to recognize.

2.1.4 Simulation for Bias-avoidance and Robustness Evaluation

Since traffic sign recognition is a very complex problem with many different aspects coming together, it is very hard to come up with a satisfactory solution. Once the problem reaches a certain scale inside our real world, it becomes very hard to validate the system robustness. This especially occurs in ML systems which often depend to the most part on the quality and quantity of the available data. In order to support our current system design for this but also for future iterations of this project, we need to implement a simulated environment. A simulation comes with different advantages, which can highly contribute to the system design. While it offers a automatic generation of unlimited amounts of data which would not be possible to collect in the real world, it also enables us to generate unseen or unusual situations that we can use to evaluate the system robustness. It also helps us in bias-avoidance, which often occurs when collecting data from the real world. By defining a contextual world model paired with a probabilistic world model, we get a rough idea on how to configure our simulation to make it as real as possible. Finally, a simulation paired with the underlying probability distributions allows us to make our system ready for all kinds of safety certifications, which could be introduced in the future.

2.2 System Design and Architecture Optimization

Based on our current domain knowledge, we derived our system design for this project iteration. In Figure 2.7 you can get an overview over our conceptual world model and the overall system architecture. Our system design consists of a multi-stage pipeline. In the first stage we are extracting images of the environment from the camera sensor. In the next stage we use an object detection algorithm for traffic signs with high recall to detect and locate possible objects in the image to generate traffic sign proposals. In the last step, we use a robust classification algorithm which resolves the object detection proposals and outputs the correct class in case of a traffic sign or background in case of a false positive detection proposal.

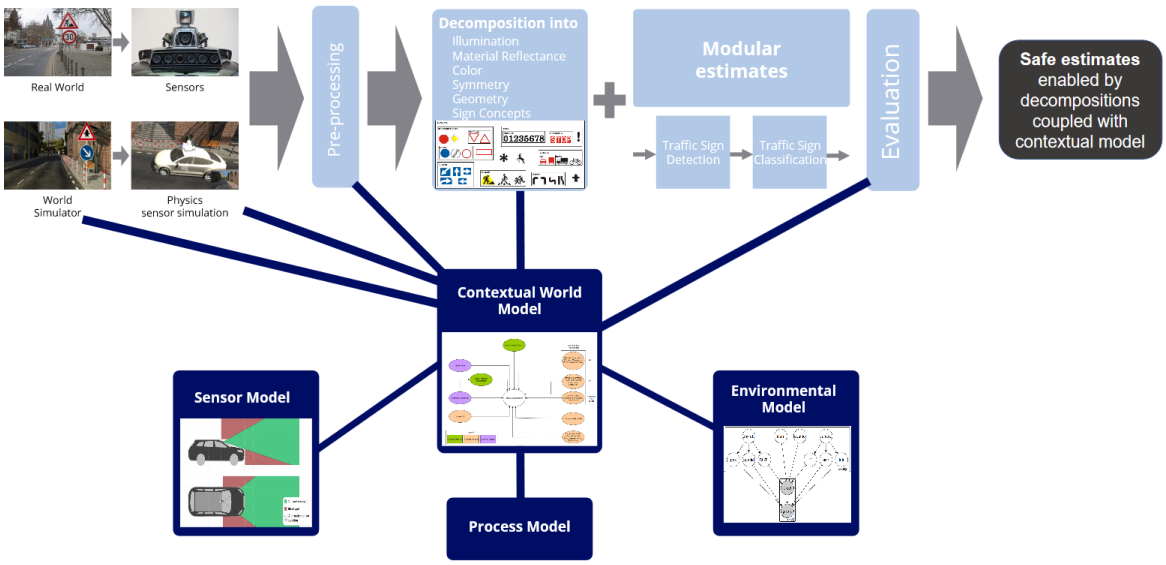


Figure 2.7: Visualisation of our contextual world model and its influence on our system design

The traffic signs with exception of the "stop sign", "give way", "priority road" and "no entry" could be divided into 4 big classes: blue circles, white circles with red borders, white triangles with red borders and white circles with 4 black diagonal lines. The ratio between the traffic sign size and the lines and borders is always the same. On top of that the color of the traffic signs is in most cases not present in the background objects like streets, buildings and trees.

Because of that the Viola and Jones algorithm with a little modification for the use on the important color images, which is a real time explainable algorithm should be a good choice for the object detection.

For the classification we decide to use the high power of convolutional neural networks. For explainability we add an additional concept whitening layer, which alter a given layer of a neural network to allow us to better understand the computation leading up to that layer.

2.3 Privacy and Transparency by Design

Since our system uses a camera sensor to take pictures of its environment, we have to consider how to protect the privacy of each individual who is affected by our system. Our approach to this problem would be to not store the camera recordings at all, if this does not turn out to be necessary in some later project iteration. At the moment we do not see any reasons to store the camera recordings with regards to traffic signs recognition, which is the reason why we tend to use the camera sensor only for real-time recognition without any video-storage component. One could argue that in case of an accident there should be some video recordings, which could be used to evaluate if there was some kind of system failure that caused the accident. A video recording would be very helpful in this case, since it could offer an end-to-end view on the

problem which occurred and could also be used as evidence. In such a case, we would tend to limit our system to only store the most recent few minutes of the car driving, which would be overwritten by new recordings once the car continues to drive on. In this case we would minimize the data that is being stored, which in fact would minimize the possibility of privacy violations. Furthermore, we would not deploy any further privacy protections like face blurring since we think that in case of an accident it is important for the authorities to identify, which individuals were the road users at the given moment. With regards to transparency, we want to deploy a new method for training deep learning models by using concept whitening layers. These enable us to train our models based on predefined handcrafted features, which we already described in section 2.1.2.2. Further details regarding the use of concept whitening will be covered in the implementation part of this document in section 3.3.

3 Implementation View

3.1 Traffic Sign Detection

To detect the regions of the traffic signs in an image we are using the Viola and Jones algorithm[11]. The algorithm is quite powerful, and its application has proven to be exceptionally notable in real-time face detection. This algorithm is painfully slow to train but can detect objects in real-time with impressive speed. Given an grayscale image, the algorithm looks at many smaller subregions and tries to find an object by looking for specific features in each subregion. The Viola Jones algorithm has the following 4 steps:

- Selecting Haar-like features
- Creating an integral image
- Running AdaBoost training
- Creating classifier cascades

3.1.1 Haar-like features

Haar-like features are digital image features used in object recognition. The red signs with a circle shape share some universal properties like the red ring on the edge of the traffic sign is darker than the pixels on the edge and the inside of the surface.

A simple way to find out which region is lighter or darker is to sum up the pixel values of both regions and compare them. The sum of pixel values in the darker region will be smaller than the sum of pixels in the lighter region. If one side is lighter than the other, it may be the edge of the surface or sometimes a middle small line may be shinier than the surrounding boxes, which can be interpreted as the border of the traffic sign. This can be accomplished using Haar-like features and with the help of them, we can interpret the different parts of a traffic sign category.

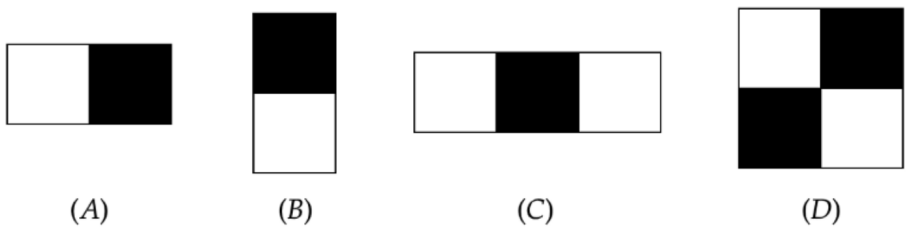


Figure 3.1: Haar-like feature types: a), b) edge features, c) line features, d) four-sided features, Figure 1 in [12]

There are 3 types of Haar-like features that Viola and Jones[11] identified in their research:

- Edge features
- Line-features
- Four-sided features

Edge features and Line features are useful for detecting edges and lines respectively. The four-sided features are used for finding diagonal features.

The value of the feature is calculated as a single number: the sum of pixel values in the black area minus the sum of pixel values in the white area. The value is zero for a plain surface in which all the pixels have the same value, and thus, provide no useful information.

To be useful, a Haar-like feature needs to give you a large number, meaning that the areas in the black and white rectangles are very different.

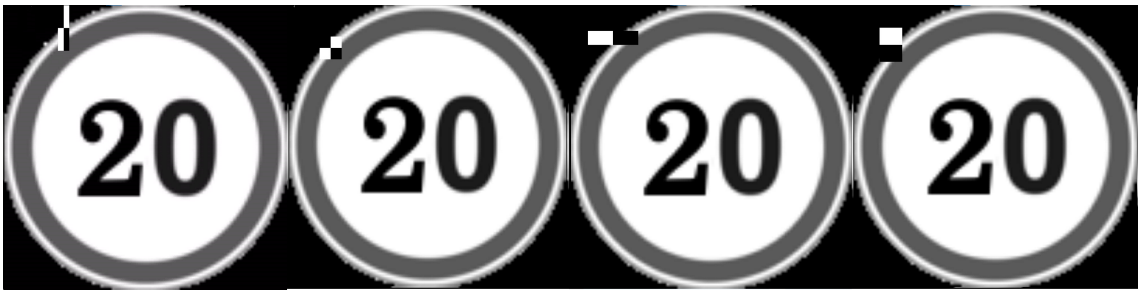


Figure 3.2: Haar-like features of an example traffic sign from our Viola-Jones[11] cascade classifier

3.1.2 Integral Image

To calculate a value for each feature, we need to perform computations on all the pixels inside that particular feature. In reality, these calculations can be very intensive since the number of pixels would be much greater when we are dealing with a large feature.

The integral image plays its part in allowing us to perform these intensive calculations quickly so we can understand whether a feature of several features fit the criteria.

An Integral Image is an intermediate representation of an image where the value for location (x, y) on the integral image equals the sum of the pixels above and to the left (inclusive) of the (x, y) location on the original image [11]. This intermediate representation is essential because it allows for fast calculation of rectangular region. Figure 3.3 shows that the sum of the red region D can be calculated in constant time instead of having to loop through all the pixels in that region. Since the process of extracting Haar-like features involves calculating the sum of dark/light rectangular regions, the introduction of Integral Images greatly cuts down the time needed to complete this task.

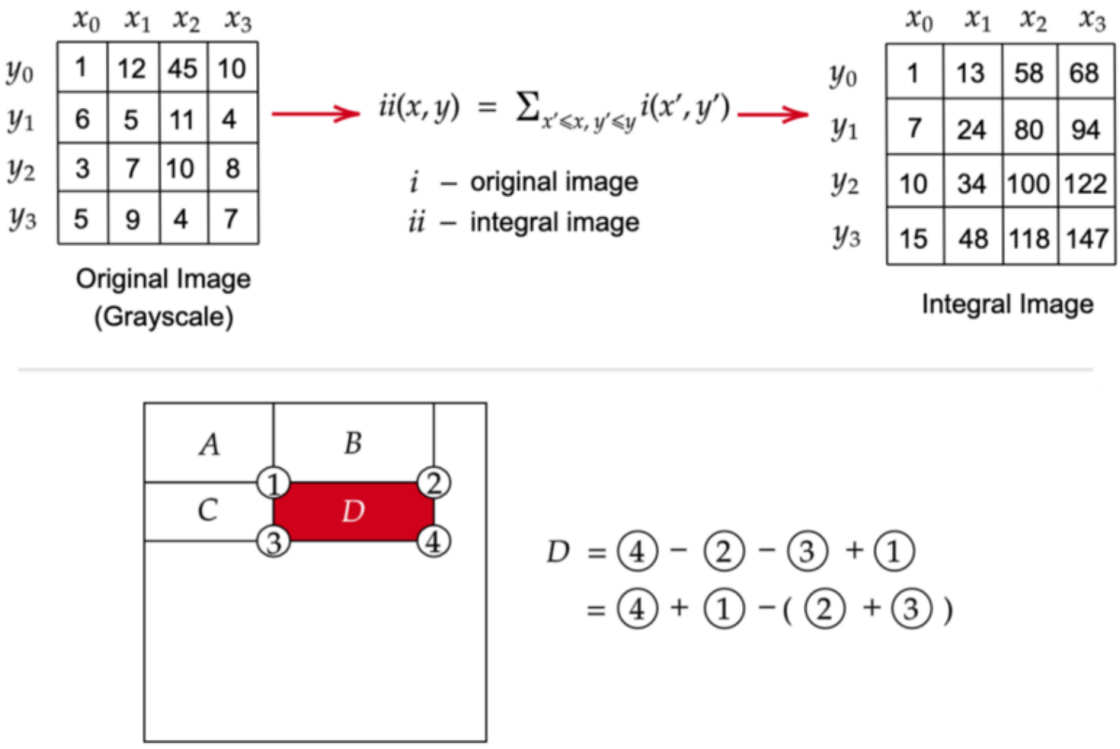


Figure 3.3: Conversion of original image to integral image (top) and how to calculate a rectangular region using an integral image (bottom) [11] cascade classifier

3.1.3 AdaBoost

The number of possible features in a 24×24 detector window is nearly 160,000, but not all of these features are important to identify a traffic sign. So the AdaBoost algorithm, shown in figure 3.4 is used to identify the best features from the 160,000 features.

Each Haar-like feature represents a weak learner. To decide the type and size of a feature that goes into the final classifier, AdaBoost checks the performance of all classifiers that are supplied to it.

The performance of a classifier is calculated by evaluating the feature on all subregions of all the images in the training dataset. The classifier will classify the subregions with a strong response as positive, which means they contain a traffic sign and subregions that do not provide a strong response will be classified as negatives. According to the classifier, these images do not contain a traffic sign.

The classifiers that performed well are given higher importance or weight. The final result is a strong classifier, also called a boosted classifier, that contains the best performing weak classifiers.

- Given example images $(x_1, y_1), \ldots, (x_n, y_n)$ where $y_i = 0, 1$ for negative and positive examples respectively.
- Initialize weights $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ for $y_i = 0, 1$ respectively, where m and l are the number of negatives and positives respectively.
- For $t = 1, \ldots, T$:

  1. Normalize the weights,

  $$w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$$

  so that w_t is a probability distribution.

  2. For each feature, j, train a classifier h_j which is restricted to using a single feature. The error is evaluated with respect to w_t, $\epsilon_j = \sum_i w_i \left| h_j(x_i) - y_i \right|$.
  3. Choose the classifier, h_t, with the lowest error ϵ_t.
  4. Update the weights:

  $$w_{t+1,i} = w_{t,i} \beta_t^{1-e_i}$$

  where $e_i = 0$ if example x_i is classified correctly, $e_i = 1$ otherwise, and $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$.

- The final strong classifier is:

$$h(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \text{otherwise} \end{cases}$$

where $\alpha_t = \log \frac{1}{\beta_t}$

Figure 3.4: The AdaBoost algorithm for classifier learning. Each round of boosting selects one feature from the potential features, Table 1 in [11]

3.1.4 Cascade Classifier

To speed up the prediction we set up a cascaded system in which we divide the process of identifying a traffic sign into multiple stages. In the first stage, we have a classifier which is made up of our best features. If the subregion contains the features of the classifier it will be given to the next classifier. Otherwise it will be predicted as a negative example, which means that it will be discarded. If all classifiers classify the subregion as positive it will be predicted as a traffic sign.

Every classifier on each stage will be trained with the goal that it achieves nearly a hit-rate of about 100% to classify a subregion, which contains a traffic sign as positive and a false-alarm rate of under 30%. The main idea is, that the stack of classifiers does not discard any traffic sign, but every classifier discard a negative example by different features.

3.1.5 Dataset

For the negative examples we are using the images from the German Traffic Sign Detection Benchmark - GTSDB, downloaded images from the internet and self generated

images from our simulation. For the positive examples we are using the images from our rebalanced dataset from the GTSRB[6]. The problem here is that the images do not containg clear bounding boxes, because it is a Recognition Benchmark, and it also does not contain enough images for training a cascade classifier.

3.1.5.1 Spatial Transformer Network

To get better bounding boxes we use a Spatial Transformer Network (STN) [13] that we train together with a simple convolutional neural network for image classification on our rebalanced dataset. After the training process we encapsulate the STN from the simple convolutional neural network and fed each image from our dataset into the STN. Some examples results images are shown below in Figure 3.5.

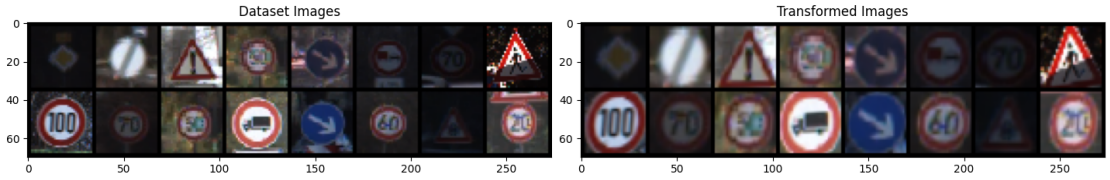


Figure 3.5: Example transformed images from the STN [13]

3.1.6 Training

We trained 4 different classifiers for the following 4 classes of traffic signs The first one

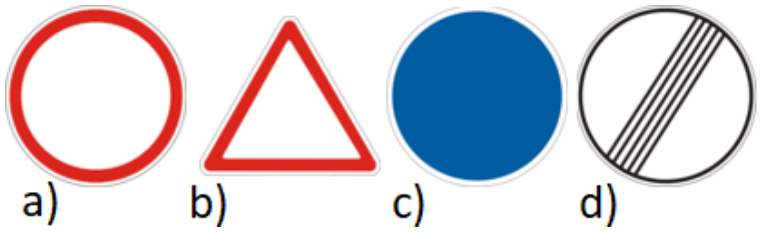


Figure 3.6: Traffic sign classes: a) red circles, b) red triangles, c) blue circles, d) white slashes

are all traffic signs with a circle surface and a red border. The second one are all traffic sings with a triangle surface and a red border. The third one are all traffic signs with a circle surface and a blue color. The last one are all traffic signs with white color and 4 black lines, which goes from the bottom left to the top right.

We trained all classifiers in the four cascades with a minimum 99% hit rate and a 30% false alarm rate with 1000 positive examples and 2000 negative examples as grayscale images.

On top of that we train for the red circle, red triangle and the blue circle class an additional cascade classifier with the same settings, but we only use the red color channel for the red circle and the red triangle class and the blue color channel for the blue circle class.

3.1.7 Validation

For the validation we use the test dataset of the GTSDB[14] which consists of 600 images with about 1000 bounding boxes. The results are shown in the following table:

class	color	recall
red circle	grayscale	79
	red	71
red triangle	grayscale	88
	red	81
blue circle	grayscale	48
	blue	64
white slashes	grayscale	83

3.2 Traffic Sign Classification

To classify the region proposals from the object detection algorithm, we used a convolutional neural network named ResNet18[15], which is shown in Figure 3.7. ResNet[15]

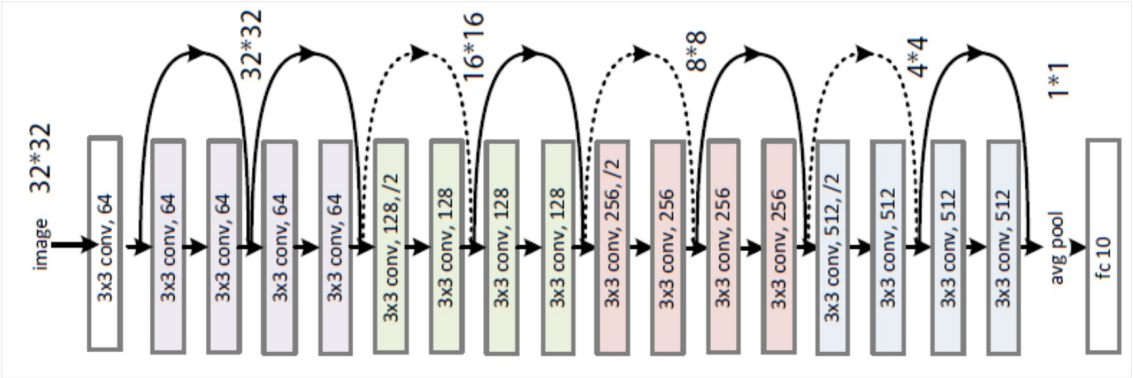


Figure 3.7: ResNet18 architecture [16]

is one of the most widely used neural networks for image classification and is available in many different versions. We used the version with 18 Layers because it is a little bit faster and the goal of our project is to have a real-time prediction. The visual appearance of German traffic signs is not really complex, so a smaller number of layers should not effect the total performance of the network. For training we add an additional batch normalization layer after each residual block, to get a faster convergence and to be robust against hyperparameters in the training process.

3.2.1 Dataset

As dataset for training and validation we use the German Traffic Sign Recognition Benchmark (GTSRB) [6]. It is a comprehensive, lifelike dataset of more than 50,000 traffic sign images that has been collected. It reflects the strong variations in visual

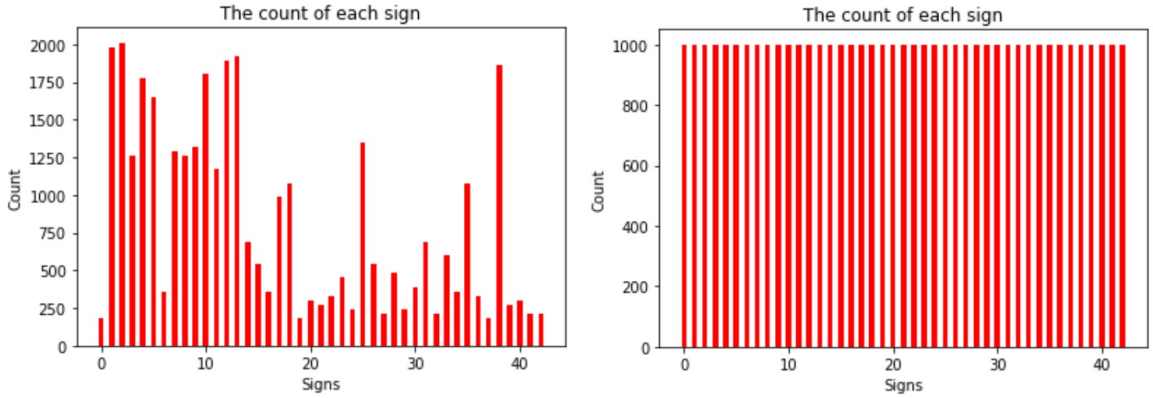
Figure 3.8: Different classes of GTSRB

appearance of signs due to distance, illumination, weather conditions, partial occlusions, and rotations. The images are complemented by several precomputed feature sets to allow for applying machine learning algorithms without background knowledge in image processing. The dataset comprises 43 classes, which are shown in Figure 3.8 with unbalanced class frequencies. The provided test set has more than 12.500 images.

Machine Learning in general requires independent, identically distributed datasets to ensure that the system learn each part of the dataset good and doesn't overfit some type of data. The unbalanced class frequencies in the dataset are a big problem that we have to solve before we can use the dataset to train our ResNet18[15] model. To avoid the unbalanced class frequencies we choose 1000 images of each class random uniform distributed, which has at least 1000 images. For classes with less than 1000 images we add for each image an additional rotated version with angle α until the limit of 1000 images is reached. The parameter α is defined as:

$$\alpha = \texttt{randInt(-20,20)},$$

where `randInt`(start, stop) generates a random uniform distributed number in [start, stop]. The result is shown in Figure 3.9.



(a) unbalanced dataset **(b)** preprocessed balanced dataset

Figure 3.9: Dataset class frequencies before and after preprocessing

3.2.2 Training

For training we add an additional class which is defined as the background class. It consists of 1000 random uniform distributed images that we choose from the GTSDB [14] and the self generated background images from our simulation which we used for the Viola and Jones[11] Cascade Classifier. We train ResNet18[15] for 10 epochs with a learning rate of 0.01, momentum of 0.9 and Stochastic Gradient Descent as optimizer. As loss function we use Cross Entropy Loss.

3.2.3 Validation

For the validation we measure the top1 and top5 precision on the training- and test dataset for the whole training process, which is shown in Figure 3.10. We reached the

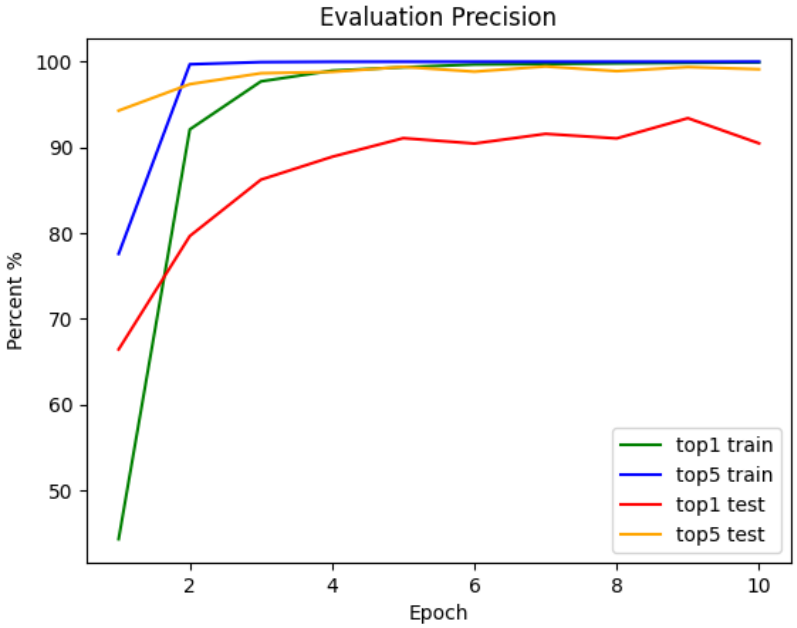


Figure 3.10: Validation of ResNet18[15] in training process

best parameters in the 9th epoch and saved the weights to use them in the following procedure:

train		test	
top1	top5	top1	top5
99.873	99.998	93.413	99.375

3.3 Concept Whitening

We use a method named Concept Whitening[17] which alter a given layer of a neural network to allow us to better understand the computation leading up to that layer,

rather than attempting to analyze a neural network posthoc like other methods. When a Concept Whitening[17] module is added to a CNN, the axes of the latent space are aligned with known concepts of interest. We use an additional concept dataset, to disentangle the latent space of a neural network, so that each channel of a convolution layer gets the highest activation, when the input image contains the concept, where the channel is aligned to.

3.3.1 Methodology

Suppose that Z is the latent space of a hidden layer, $x_1, x_2, ..., x_n$ are samples from our dataset and $y_1, y_2..., y_n$ are their labels. A DNN classifier

$$f : X \longrightarrow Y$$

can be divided into two parts, a feature extractor

$$\Phi : X \longrightarrow Z, \qquad \text{with parameters } \theta,$$

and a classifier

$$g : Z \longrightarrow Y, \qquad \text{parameterized by } \omega$$

Then

$$z = \Phi(x; \theta)$$

is the latent representation of the input x and

$$f(x) = g(\Phi(x; \theta; \omega)$$

is the predicted label. Suppose we are interested in k concepts called $c_1, c_2, ..., c_k$. We can then predefine k auxiliary datasets $X_{c_1}, X_{c_2}, ..., X_{c_n}$ such hat samples in X_{c_j} are the most representative samples of concept c_j. The goal is to learn Φ and g simultaneously, such that (a) the classifier $f(x)$ can predict the label accurately; (b) the j^{th} dimension z_j of the latent representation z aligns with concept c_j. In other words, samples in X_{c_j} should have larger values of z_j than other samples. Conversly, samples not in X_{c_j} should have smaller values of z_j.

3.3.2 Concept Separation Requirements

As illustrated in Figure 3.11c a latent space in which unit vectors can effectively represent different concepts should have small inter-concept similarity. In Figure 3.11b and c unit vectors are not valid for representing concepts. That is, samples of different concepts should be near orthogonal in the latent space. In addition, for better concept separation, the ratio between inter concept similarity and intra-concept similarity should be as small as possible. The Concept Whitening[17] module can make the latent space mean-centered and decorrelated. It can align predefined concepts in orthogonal directions.

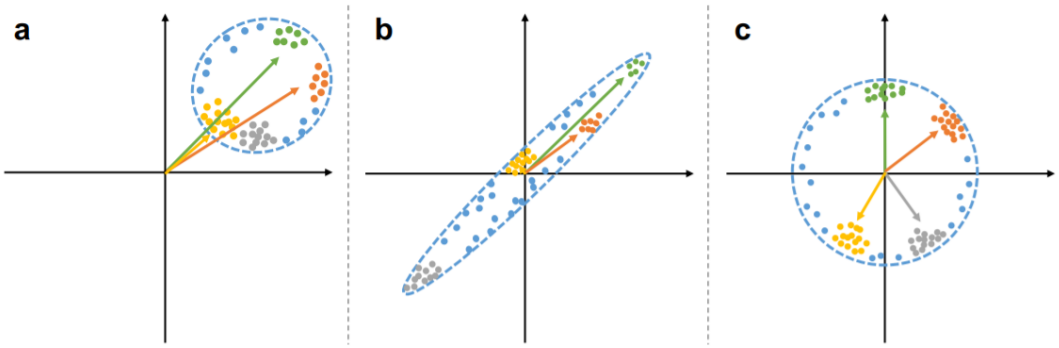


Figure 3.11: Possible data distributions in the latent space. **a** the data are not mean centered, **b** the data are standardized but not decorrelated, **c** the data are whitened. Figure 1 in [17]

3.3.3 Concept Whitening Module

The Concept Whitening[17] module consists of two parts, whitening and orthogonal transformation. The whitening transformation ψ decorrelates and standardizes the data by

$$\psi(\mathbf{Z}) = \mathbf{W}(\mathbf{Z} - \mu \mathbf{1}_{n\times 1}{}^T)$$

,where

$$\mu = \frac{1}{n}\sum_{i=1}^{n} \mathbf{z}_i$$

is the sample mean and $\mathbf{W}_{d\times d}$ is the whitening matrix that obeys

$$\mathbf{W}^T\mathbf{W} = \Sigma^{-1}.$$

Here,

$$\Sigma_{d\times d} = \frac{1}{n}(\mathbf{Z} - \mu \mathbf{1}^T)(\mathbf{Z} - \mu \mathbf{1})$$

is the covariance matrix. The other important property of the whitening matrix is that it is rotation free, so if $\mathbf{Q}$ is an orthogonal matrix, then

$$\mathbf{W}' = \mathbf{Q}^T\mathbf{W}$$

is also a valid whitening matrix. The Concept Whitening[17] module needs to rotate the samples in their latent space such that the data from concept c_j, namely $\mathbf{X}_{cj}$, are highly activated on the j^{th} axis. Mathematically we have to optimizing the following objective, by finding an orthogonal matrix $\mathbf{Q}_{d\times d}$ whose column $\mathbf{q}_j$ is the j^{th} axis.

$$\max_{\mathbf{q}_1,\mathbf{q}_2,\ldots,\mathbf{q}_k} \sum_{j=1}^{k} \frac{1}{n_j}\mathbf{q}_j^T \psi(\mathbf{Z}_{cj})\mathbf{1}_{n_j\times 1}, \qquad s.t.\ \mathbf{Q}^T\mathbf{Q} = \mathbf{I}_d$$

Algorithm 1. Forward Pass of CW Module

Input: mini-batch input $\mathbf{Z} \in \mathbb{R}^{d \times m}$
Optimization Variables: orthogonal matrix $\mathbf{Q} \in \mathbb{R}^{d \times d}$ (learned in Algorithm 2)
Output: whitened representation $\hat{\mathbf{Z}} \in \mathbb{R}^{d \times m}$
calculate batch mean: $\mu = \frac{1}{m}\mathbf{Z} \cdot \mathbf{1}$, and center the activation: $\mathbf{Z_C} = \mathbf{Z} - \mu \cdot \mathbf{1}^T$
calculate ZCA-whitening matrix $\mathbf{W}$
calculate the whitened representation: $\hat{\mathbf{Z}} = \mathbf{Q}^T\mathbf{W}\mathbf{Z_C}$.

As part of the training for any given layer of a neural network Concept Whitening[17] could constitute this whole procedure. The forward pass of the Concept whitening module is shown in Algorithm 1. The used ZCA-whitening is based on the IterNorm algorithm [18], which employs Newton's iterations to approximate ZCA whitening. The process will be explained in following. The whitening matrix in ZCA is

$$\mathbf{W} = \mathbf{D}\mathbf{\Lambda}^{-\frac{1}{2}}\mathbf{D}^T$$

where $\mathbf{A}_{d\times d}$ is the eigenvalue diagonal matrix and $\mathbf{D}_{d\times d}$ is the eigenvector matrix given by the eigenvalue decomposition of the covariance matrix

$$\mathbf{\Sigma} = \mathbf{D}\mathbf{\Lambda}\mathbf{D}^T$$

Like other normalization methods, for example batch normalization, a μ and $\mathbf{W}$ for each mini-batch of data will be created, and averaged to form the model used in testing. On top of the whitening step, Concept Whitening also need to learn an orthogonal matrix by solving an optimization problem. For this the objective will be optimized while strictly maintaining the matrix to be orthogonal by performing gradient descent with a curvilinear search on the Stiefel-manifold [19] and adjust it to deal with mini-batch data.

3.3.4 Alternating optimization on two steps

The model have to handle two types of data:

- data for calculating the main objective
- data representing the predefined concepts

The mini-batches of the main dataset and the auxiliary concept dataset are fed to the network, and both objectives are optimized in turns. The first objective, which is in our case the loss for the traffic sign classification:

$$\min_{\theta,\omega,W,\mu} \frac{1}{n}\sum_{i=1}^{n} l(g(\mathbf{Q}^T\psi(\Phi(x_i^{(cj)};\theta);\mathbf{W},\mu)$$

,where

- Φ is the layer before the Concept Whitening module parameterized by θ

- g is the layer after the Concept Whitening module parameterized by ω
- ψ is the whitening transformation parameterized by sample mean μ and whitening matrix $\mathbf{W}$
- $\mathbf{Q}$ is the orthogonal matrix, which forms in the combination $\mathbf{Q}^T\psi$ the Concept Whitening module
- l is any differentiable loss, in our case this is the cross-entropy loss

The second objective is the concept alignment loss:

$$\max_{\mathbf{q}_1,\mathbf{q}_2,\ldots,\mathbf{q}_k} \sum_{j=1}^{k} \frac{1}{n_j} \sum_{x_i^{(c_j)} \in X_{cj}} \mathbf{q}_j^T \psi(\Phi(x_i^{(c_j)};\theta);\mathbf{W},\mu), \qquad s.t.\ \mathbf{Q}^T\mathbf{Q} = \mathbf{I}_d$$

Directly solving for the optimal solution is intractable, because the optimization problem is a linear programming problem with quadratic constraints (LPQC) which is generally NP-hard. So it will optimize it by gradient methods on the Stiefel manifold. At each step t, in which the second objective is handled, the orthogonal matrix $\mathbf{Q}$ is updated by Cayley transform

$$\mathbf{Q}^{t+1} = \Big(I + \frac{\eta}{2}\mathbf{A}\Big)^{-1}\Big(I - \frac{\eta}{2}\mathbf{A}\Big)\mathbf{Q}^{(t)}$$

where

- $\mathbf{A} = \mathbf{G}(\mathbf{Q}^{(t)})^T - \mathbf{Q}^{(t)}\mathbf{G}^T$ is a skew-symmetric matrix
- $\mathbf{G}$ is the gradient of the loss function
- η is the learning rate

The Cayley transform has many solutions, because of the stationary points are reached when $\mathbf{A} = \mathbf{0}$, which has multiple solutions. Since the solutions are very likely to be saddle points, which can be avoided by SGD. Therefore the stochastic gradient is calculated by a mini-batch of samples to replace $\mathbf{G}$ at each step with applied momentum to accelerate and stabilize the gradient. The full optimization is provided by algorithm 2.

3.3.5 Concept Whitening on convolution outputs

In a CNNs the output of a layer is a tensor instead of a vector. In CNNs, a feature map contains the information of how activated a part of the image is by a single filter, which may be a detector for a specific concept. To reshape the feature map into a vector, where each element of the vector represents how much one part of the image is activated by the filter. A vector of length hw contains the activation information for one filter with size $h \times w$ around the whole feature map. The convolution Layer output $Z_{h\times w\times d\times n}$ is reshaped into a matrix $Z_{d\times(hwn)}$ by doing this procedure for each

Algorithm 2. Alternating Optimization Algorithm for Training

Input: main objective dataset $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^n$, concept datasets $\mathbf{X}_{c_1}, \mathbf{X}_{c_2}, ..., \mathbf{X}_{c_k}$
Optimization Variables: θ, ω, $\mathbf{W}$, μ, $\mathbf{Q}$, whose definitions are in Section 3.2
Parameters: β, η
for t = 1 to T **do**
    randomly sample a mini-batch $\{\mathbf{x}_i, y_i\}_{i=1}^m$ from $\mathcal{D}$
    do one step of SGD w.r.t. θ and ω on the loss $\frac{1}{m}\sum_{i=1}^m \ell(g(\mathbf{Q}^T\psi(\mathbf{\Phi}(\mathbf{x}_i;\theta);\mathbf{W},\mu);\omega), y_i)$
    update $\mathbf{W}$ and μ by exponential moving average
    if $t \bmod 20 = 0$ **then**
        sample mini-batches $\{\mathbf{x}_i^{(c_1)}\}_{i=1}^m, \{\mathbf{x}_i^{(c_2)}\}_{i=1}^m, ..., \{\mathbf{x}_i^{(c_k)}\}_{i=1}^m$ from $\mathbf{X}_{c_1}, \mathbf{X}_{c_2}, ..., \mathbf{X}_{c_k}$
        calculate $\mathbf{G} = \nabla_{\mathbf{Q}}$, with columns $\mathbf{g}_j = -\frac{1}{m}\sum_{i=1}^m \psi(\mathbf{\Phi}(\mathbf{x}_i^{(c_j)};\theta);\mathbf{W},\mu)$ when $1 \le j \le k$, else $\mathbf{g}_j = \mathbf{0}$
        calculate the exponential moving average of $\mathbf{G}$: $\mathbf{G}' = \beta\mathbf{G}' + (1-\beta)\mathbf{G}$
        obtain learning rate η by curvilinear search
        update $\mathbf{Q}$ by Cayley transform: $\mathbf{Q} \leftarrow (I + \frac{\eta}{2}(\mathbf{G}'\mathbf{Q}^T - \mathbf{Q}\mathbf{G}'^T))^{-1}(I - \frac{\eta}{2}(\mathbf{G}'\mathbf{Q}^T - \mathbf{Q}\mathbf{G}'^T))\mathbf{Q}$

filter, where d is the number of channels. After this Concept Whitening is performed on the reshaped matrix. After resizing the matrix to the older dimension, each channel represents weather a meaningful concept is detected at each location in the image for that layer. To apply the concept activation score, which have to be a scalar, on the feature map, there are multiple ways to do this:

- mean of all feature map values
- max of all feature maps values
- mean of all positive feature map values
- mean of down-sampled feature map obtained by max pooling

3.3.6 Concept Datasets

We defined concept training- and test datasets to apply Concept Whitening[17] to our ResNet18[15] model for the following concepts

- blue
- circle
- red
- triangle

The first one consists of manually downloaded free licensed images from `https://unsplash.com/`. The dataset contains 500 images for each concept. A few example images of each class are shown in Figure 3.12.

Figure 3.12: Example images from concepts of manually downloaded concept dataset

The second dataset is self created by the following formulars. The images for the blue concept are created by random chosen blue value pixels p_{blue}

$$p_{blue} = \Big(0, 0, \texttt{randInt}(0, 255)\Big).$$

The images for the red concept are created by random chosen red value pixels p_{red}

$$p_{red} = \Big(\texttt{randInt}(0, 255), 0, 0\Big).$$

The images for the circle and the red concept are created by the common variables background color $color_{bg}$, object color $color_{obj}$, thickness of the border in pixel $t \in \{-1, 2, 3, 4\}$, where -1 is a full filled object.

$$color_x = \Big(\texttt{randInt}(0, 255), \texttt{randInt}(0, 255), \texttt{randInt}(0, 255)\Big)$$
$$t = \texttt{randChoice}(\{-1, 2, 3, 4\})$$

The circle is defined by the center coordinates x and y and the radius r

$$x = \texttt{randInt}(0, width)$$
$$y = \texttt{randInt}(0, height)$$
$$r = \texttt{randInt}(1, r_{max})$$

where

$$r_{max} = \texttt{min}(x, y, width - x, height - y,)$$

The triangle is defined by the three corners $c_i = (x_i, y_i)$ $i \in 1, 2, 3$

$$c_i = \Big(\texttt{randInt}(0, width), \texttt{randInt}(0, height)\Big)$$

A few example images of each concept are shown in Figure 3.13.

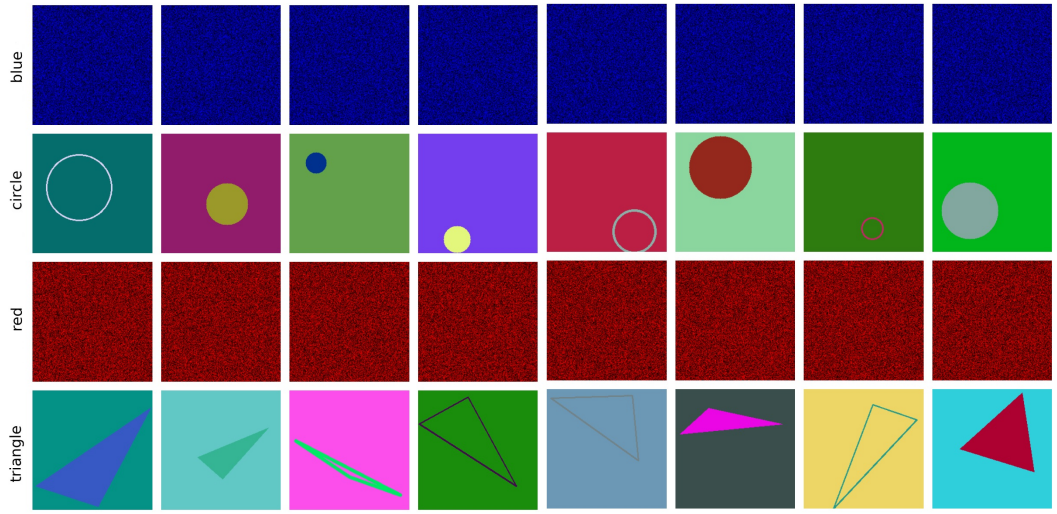


Figure 3.13: Example images from concepts of self generated images

3.3.7 Training

We replace only one Batch Normalization Layer after a residual block from our ResNet18[15] with the Concept Whitening[17] module. We do this procedure for the 5th residual block and trained with the same parameters like before, except learning rate 0.05 for 5 additional epochs.

3.3.8 Concept Whitening Validation

To measure the performance of the Concept Whitening[17] module, we measured the AUC for the testset of both dataset types. The result is shown below.

dataset	concept	1 epoch	5 epochs
selfmade	blue	0.894	0.932
	circle	0.671	0.670
	red	0.610	0.959
	triangle	0.537	0.344
downloaded	blue	0.9735	0.970
	circle	0.684	0.658
	red	0.845	0.923
	triangle	0.709	0.582

For the validation of the traffic sign classification we measure the top1 and top5 precision on the test dataset of GTSRB.

dataset	epochs	top1	top5
without	9	93.413	99.375
selfmade	10	90.792	99.097
	14	93.674	99.367
downloaded	10	91.696	99.305
	14	92.352	98.971

Figure 3.14: Top 10 activated images from test dataset for each concept

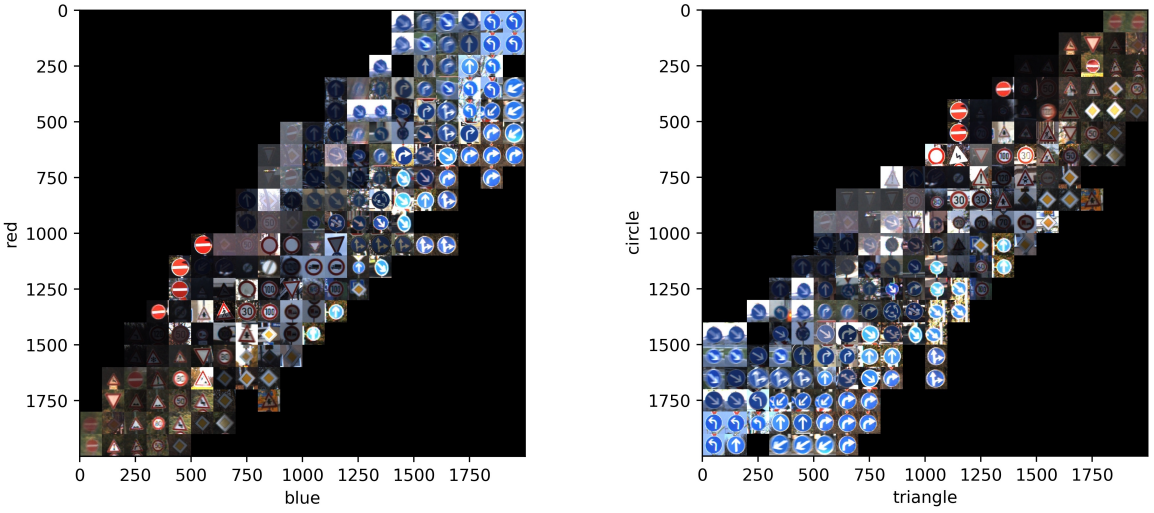


Figure 3.15: Random test images on the red/blue and the circle/triangle subspace

3.4 Simulation

To support our model decisions, we developed a realistic simulation which focuses on the task of traffic sign recognition in an outdoor environment. For this purpose we decided to use the Unity game engine which comes with a great amount of useful assets and physics controllers. Our simulation is based on 4 key-components: the environmental component, the car component, the traffic sign component and the traffic sign recognition system. The environmental component consists of almost all objects which are present in the simulation including the weather and lighting conditions. Furthermore, it contains the road infrastructure and all buildings and vegetation. The

car component is added in order to mount our system and to use it as a real driving simulation. It consists of a 3D car model, which we can configure and adapt depending on our needs. The traffic sign component is configured to regulate all the different parameters regarding traffic signs inside the simulation and can also be used for dataset generation. Lastly, we integrate our recognition system inside our simulation to detect and classify traffic signs like we would in the real world.

3.4.1 Environmental Component

The backbone of our simulation consists of the environmental component. It contains all contextual information for traffic sign recognition and plays a key role in building a strong simulation platform in which our system lives and operates. For this purpose, we choose to use the standard Unity asset "Windridge City" developed by NatureManufacture and Indago [20]. It was initially developed as an environment for the AirSim Simulation offered by Microsoft, which focuses on autonomous vehicle research [21]. Windridge City covers almost all possible environmental scenes which can occur in the context of traffic sign recognition. It includes all types of roads, ranging from urban streets of densely populated areas to rural area roads, highways and forest roads. It also comes with a set of different road textures and floor markings. The asset offers a beautiful scene view with realistic graphics and runs with very good performance in over 100 FPS on a standard PC. In Figure 3.16 you can view some screenshots of the Windridge City environment.

Figure 3.16: Four example scenes from the Windridge City asset [20]

The other important environmental component is the weather system and the external illumination of the scene. We constructed our weather system to have 5 possible options, which can be switched inside the UI during runtime. Our weather system supports the following options: sunny day, rainy day, sunrise/sunset, bright night, dark night. In Figure 3.17 you can get a glimpse at 5 different scenes from our simulation. For each of the available options we developed an individual scene lighting and used a different skybox, which suits best the given weather condition. Furthermore, we added

street lamp illumination around to city to simulate urban environment at night. For the rain particle system we used an asset developed by Digital Ruby [22], which offers a very good tradeoff between system performance and realistic graphics.

Figure 3.17: Example scenes from the simulation showing different weather conditions

3.4.2 Integration of the Car Component

Since our system requires a drivable car to be mounted on, we had to develop an individual car simulation component that supports our system requirements and fits our needs in the best possible way. For the 3D car model we used the free available asset offered by Final Form Studio [23], but could have also used any other 3D car model available on the internet. To make our car drivable we used the Unity built-in physics engine, which offers the wheel collider component. "The Wheel Collider is a special collider for grounded vehicles. It has built-in collision detection, wheel physics,

and a slip-based tire friction model. It can be used for objects other than wheels, but it is specifically designed for vehicles with wheels" [24]. Our simulation comes with 2 driving modes: autonomous driving and manual driving. The autonomous driving mode makes the car follow a predefined path that was handmade, while the manual mode gives the user an opportunity to drive the car himself. We wanted to automate the simulation as much as possible, but at the same time give the user an opportunity to further explore the simulation with custom experiences and create custom situations. The simulation also offers different camera modes which are centered around the car and can be manually switched during runtime. The user can move the camera freely to look around, like shown in Figure 3.18.

Figure 3.18: Screenshots from the simulation showing the different camera modes

Furthermore, we equipped the car with different lighting options including: brake lights, activated brake lights, low beam headlights, long beam headlights and turn signal lights, which all can be controlled via the UI except for the brake lights. To support our system design, we tried out different positions for the camera sensor and finally decided to place it slightly above the car roof, a few centimeters away from the windshield. We also concluded to set the field of view to 40 degrees since it offers a good front area coverage and does not stress out our system too much with too much information from the image measurement.

3.4.3 Integration of the Traffic Sign Component

The traffic sign component is arguably the most important component, since it directly influences the quality of our simulation and the image measurements for our task. We decided to create every single 3D traffic sign model manually to have full control over them and to make it easy to add new signs to the simulation. To create the traffic sign shapes, we used the 3D modeling software Blender and textured the models with traffic sign images found on the internet. Furthermore, we implemented a traffic sign spawning script that generates and distributes the traffic signs inside the scene based on some predefined parameters. Right before the simulation starts, the user can specify different sign distribution options inside the main menu like shown in Figure 3.19.

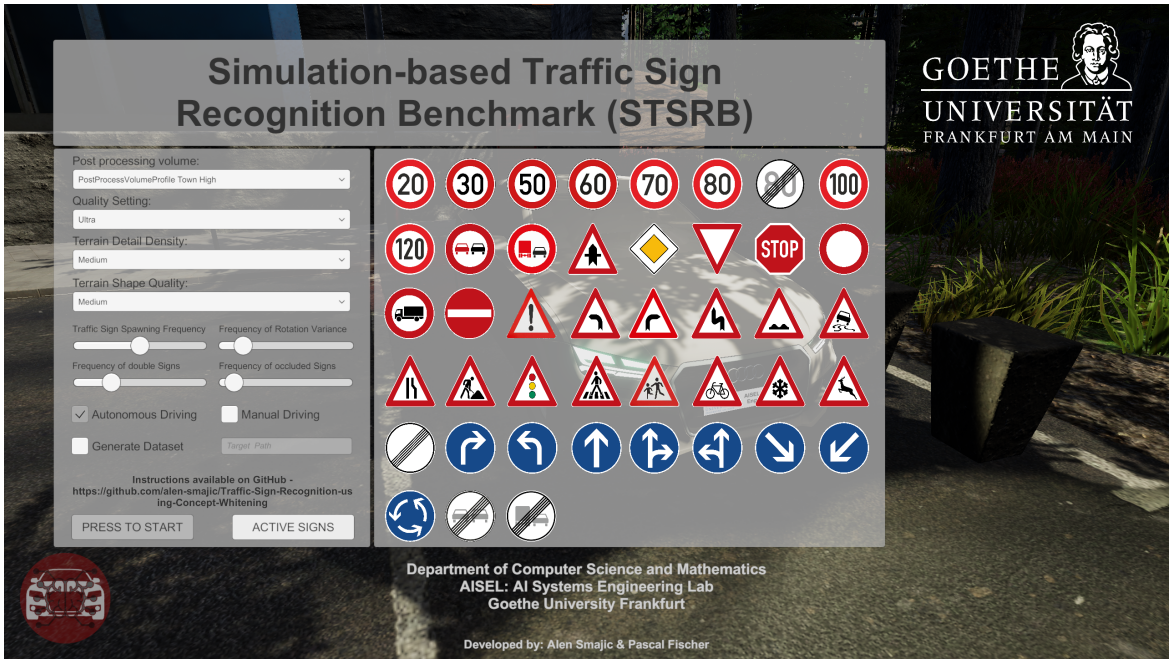


Figure 3.19: Main menu of the simulation

Traffic Sign Spawning Frequency

We manually distributed around 469 traffic sign poles on the map which are used as spawn points for the traffic signs. At the start of the simulation, all of these poles are deactivated and the user can specify the probability for each traffic sign pole to be active and to contain a traffic sign by changing the traffic sign spawning frequency. This probability directly influences the amount of visible traffic signs inside the simulation.

Frequency of double Signs

Since in the real world, there are many situations where a traffic sign pole contains more than one traffic sign, we implemented a parameter to specify the probability for every active traffic sign pole to contain two signs at the same time.

Frequency of Rotation Variance

To evaluate the robustness of traffic sign recognition models and to bring our simulation closer to reality, we added a parameter that controls the probability of a single sign being rotated around the z-axis. If the algorithm decides, that a sign should be rotated (based on the frequency of rotation variance parameter) it will rotate the sign randomly in either direction to a maximum of 30 degrees, which we determined as the strongest rotation degree where the sign is still identifiable.

Frequency of occluded Signs

To further expand the robustness check of our simulation, we also implemented a parameter to control the probability of an active sign being occluded by sticker objects. If our algorithm decided that a sign should be occluded (based on the frequency of occluded signs parameter) it will spawn between 1 and 5 stickers onto the visual surface of the traffic sign. Furthermore it will calculate a random rotation and sticker dimension and it will also apply some random sticker textures from our texture pool.

Active Signs

We implemented an option where the user can specify, which traffic signs should be spawned inside the simulation. This option is especially interesting in cases where we want to evaluate our model on some specific traffic sign classes.

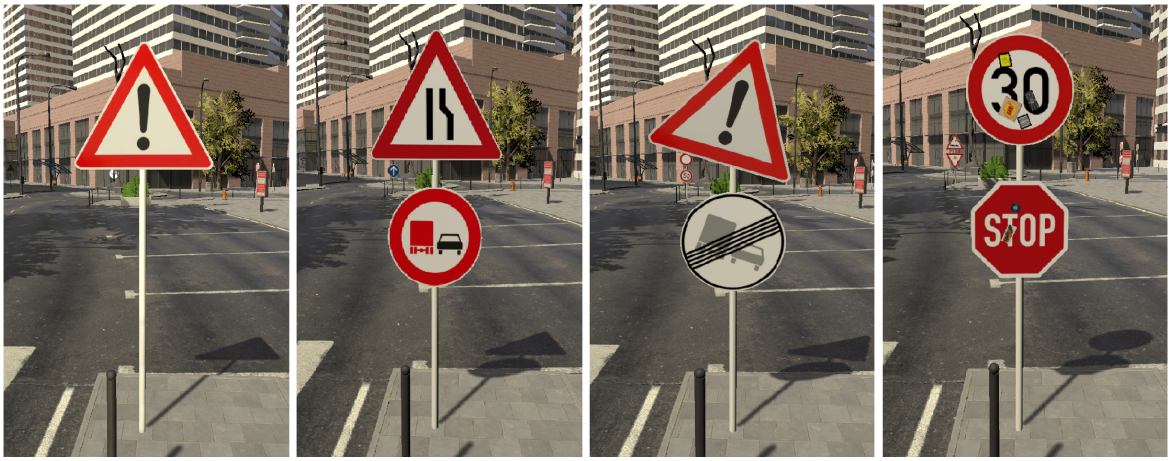


Figure 3.20: Example screenshots of randomly spawned traffic signs

Dataset Generation

The implementation of a dataset generation function turned out to be a very complex and challenging task. The reason for this was, that the camera component, which was used to take images of the environmental scene, had no information where the traffic signs are located inside the image and since the camera was moving along with

the car we would have many problems to deploy any localisation algorithms paired with our traffic sign spawn function. We solved this problem by applying a multistage approach. The first stage consist of generating the dataset images. Once the user starts the simulation, the algorithm stops the scene every few seconds (this frequency can be set by the user) to take two screenshots of the environmental scene. The first screenshot is taken from the immediate car environment and it represents what the camera sensor sees (sensor input). After this, the algorithm changes the scene for the second screenshot so that every traffic sign gets a pink texture (some color that is not present in the background objects at all). Furthermore, the algorithm adjusts the lighting conditions to be evenly distributed without any shadows to make a clear screenshot with clearly visible traffic signs, painted in pink. The algorithm stores both screenshots in separate folders. Once the first stage is finished, our algorithm has generated different environmental images including the corresponding ground truth images, which contain pink traffic signs. Now in the next stage we deploy an OpenCV function on the images containing the pink traffic signs, which thresholds all pixels from the image to be in a certain color range (in our case this is the pink color). Once this is done, we can separate foreground (traffic signs) and background (not traffic signs). Lastly we deploy the Contours function from OpenCV, which generates bounding boxes around the foreground objects. These bounding boxes can now be extracted in a custom format and used as the ground truth labels for training object detection algorithms. Figure 3.21 visualizes the process of extracting the ground truth labels.

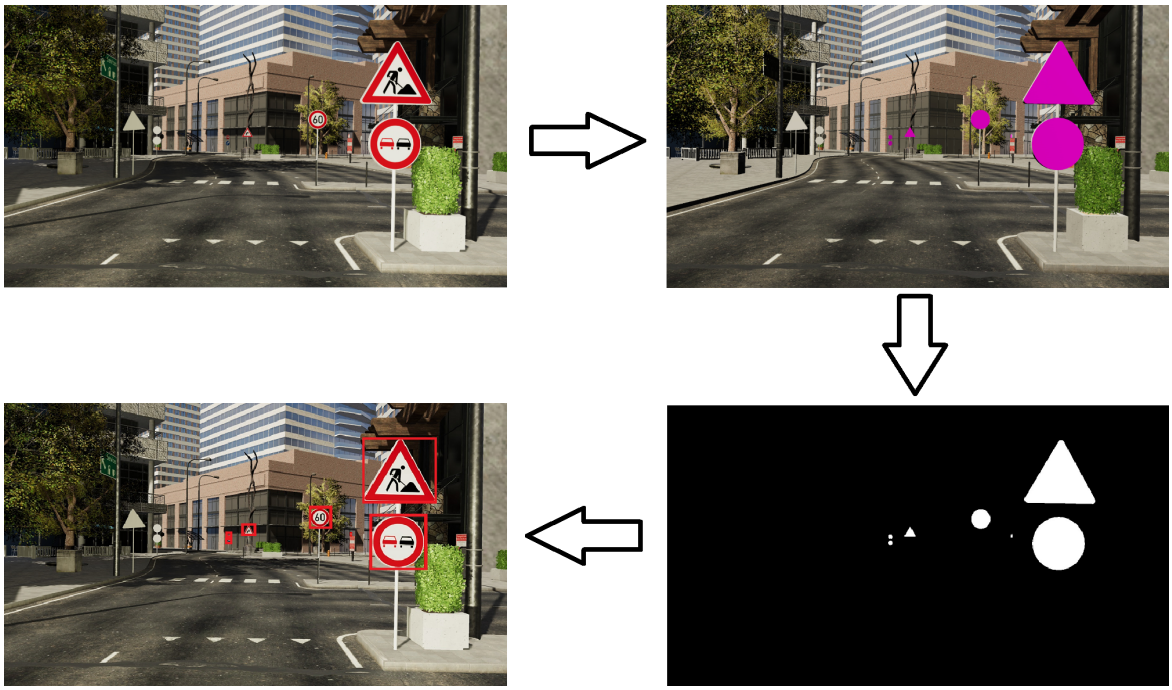


Figure 3.21: Label generation pipeline

Furthermore, we added an option inside the main menu where the user can specify the height and width dimension for the images, which are being generated for the dataset. The user can also specify the screenshot frequency in seconds, which controlls how often a screenshot is made.

This approach for the dataset generation function enables us to get the labels for the

task of traffic sign detection but does not give us any information on the respective sign classes. Future work on the simulation includes the further development of the label generation function, which supports also traffic sign classes. As a workaround, one can select only one traffic sign class inside the main menu, which will be spawned inside the simulation, and start the dataset generation process. With this approach, the simulation will generate traffic sign labels where each label corresponds to the selected traffic sign class, since it is the only one that is being spawned by the algorithm. If you repeat this process for each traffic sign you will get labels for each class.

3.4.4 Integration of the Traffic Sign Recognition System

The last component of our simulation is the traffic sign recognition system. Our goal was to deploy our model inside the simulation to measure its performance in a realistic car driving environment, where we can evaluate the recognition speed and the accuracy. Because of time constraints we only managed to integrate the first stage of our system, which is the traffic sign detection model using the Viola and Jones algorithm. To integrate the detection system, we used the OpenCV plus API [25], which is adapted from the OpenCVSharp library to be used inside the Unity environment. At runtime our camera sensor captures images from the environment, which are being rendered inside a render texture and fed into the traffic sign detection system. This algorithm produces bounding boxes which are displayed on the render texture and outputted to the user like shown in Figure 3.22. The detection of traffic signs inside our simulation runs at approximately 10 FPS on a standard PC, which is enough to prove the real-time performance of our system. Since the rendering of the simulation on its own requires some computational power, we are very confident that our system will run even faster once it is deployed on a real car.

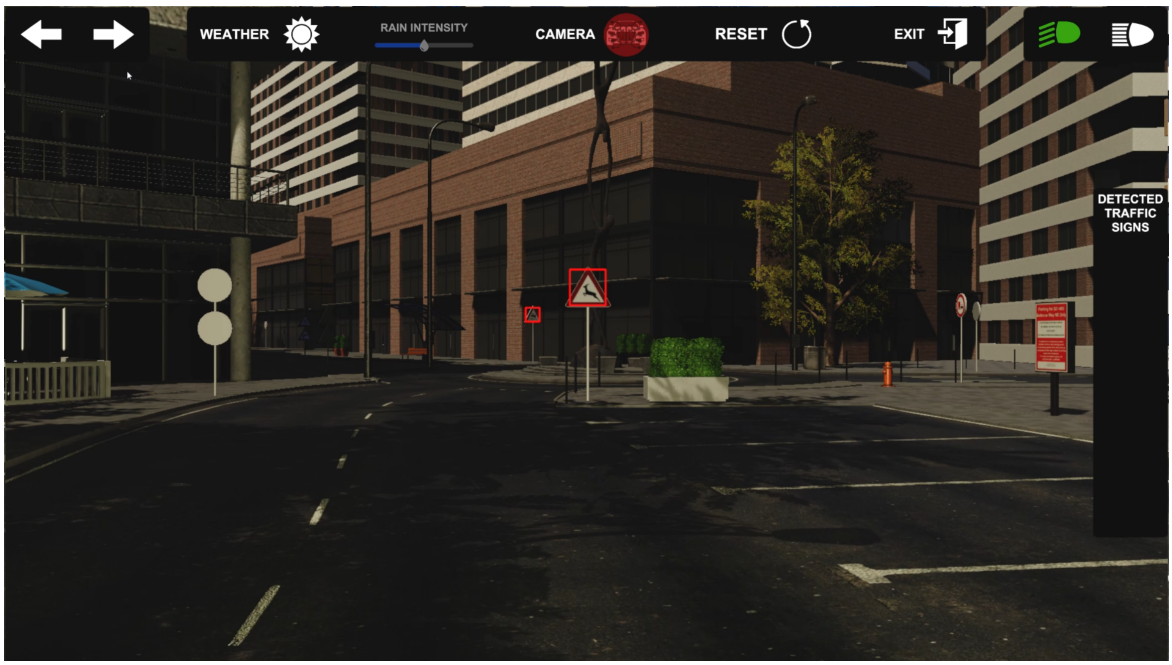


Figure 3.22: Screenshot inside the simulation showing the detection system

Our next step for future work includes the integration of the traffic sign classification

system inside our simulation. At the moment, our system recognises regions where traffic signs are located and draws a bounding box around them. In the next step we want our system to cut out the image regions within the bounding boxes to feed these into the classification model. This model will resolve the region proposal and output either the correct sign class in case of a traffic sign or background in case of a false positive detection. The final output of our system will be displayed on the right side of the UI, where the user can inspect the signs that are recognised at the given moment like shown in Figure 3.23.

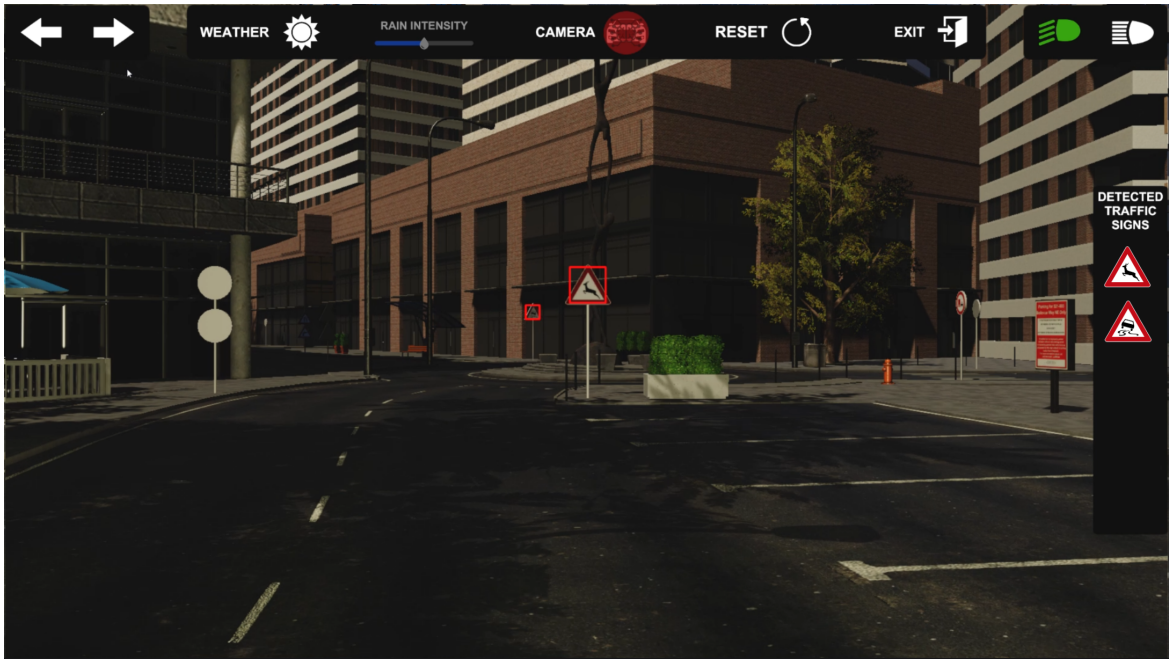


Figure 3.23: Edited screenshot showing the future version of our simulation

4 Conclusion and Future Work

In this project iteration we presented our first attempt at tackling the problem of traffic sign recognition using a systems engineering approach.

Bibliography

[1] Gene Lewis. "Object detection for autonomous vehicles," in: (2014).

[2] Eddie Forson. *Recognising Traffic Signs With 98 Percent Accuracy Using Deep Learning*. https://towardsdatascience.com/recognizing-traffic-signs-with-over-98-accuracy-using-deep-learning-86737aedc2ab. Accessed: 2021-03-30.

[3] V. Ramesh M. Pellkofer C. Bahlmann Y. Zhu and T. Koehler. "A system for traffic sign detection, tracking, and recognition using color, shape, and motion information". In: *IEEE Proceedings. Intelligent Vehicles Symposium* (2005).

[4] Felix. *Sensor Set Design Patterns for Autonomous Vehicles*. https://autonomous-driving.org/2019/01/25/positioning-sensors-for-autonomous-vehicles/. Accessed: 2021-03-31.

[5] Naturschutz und Reaktorsicherheit Die Bundesministerin für Umwelt. *Allgemeine Verwaltungsvorschrift zur Straßenverkehrs-Ordnung (VwV-StVO)*. http://www.verwaltu
im-internet.de/bsvwvbund$_2$6012001$_S$3236420014.*htm*. Accessed: 2021-03-31.

[6] Johannes Stallkamp, Marc Schlipsing, Jan Salmen and Christian Igel. "The German traffic sign recognition benchmark: a multi-class classification competition". In: *The 2011 international joint conference on neural networks*. IEEE. 2011, pp. 1453–1460.

[7] Laenderdaten.info. *Sonnenaufgang und Sonnenuntergang in Deutschland*. https://www.laenderdaten.info/Europa/Deutschland/sonnenuntergang.php. Accessed: 2021-04-02.

[8] Weatheronline. *Frankfurter Flughafen Klimarechner Jahresanalyse*. https://www.weatheronline

Accessed: 2021-04-02.

[9] Deutscher Wetterdienst. *Unwetterklimatologie: Gewitter*. https://www.dwd.de/DE/leistungen/
Accessed: 2021-04-02.

[10] Weather Spark. *Average Weather in Frankfurt am Main*. https://weatherspark.com/y/61302/A
Weather-in-Frankfurt-am-Main-Germany-Year-Round. Accessed: 2021-04-02.

[11] Paul Viola and Michael Jones. "Rapid object detection using a boosted cascade of simple features". In: *Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001*. Vol. 1. IEEE. 2001, pp. I–I.

[12] Socret Lee. *Understanding Face Detection with the Viola-Jones Object Detection Framework*. https://towardsdatascience.com/understanding-face-detection-with-the-viola-jones-object-detection-framework-c55cc2a9da14/. Accessed: 2021-03-29.

[13] Max Jaderberg, Karen Simonyan, Andrew Zisserman and Koray Kavukcuoglu. "Spatial transformer networks". In: *arXiv preprint arXiv:1506.02025* (2015).

[14] Sebastian Houben, Johannes Stallkamp, Jan Salmen, Marc Schlipsing and Christian Igel. "Detection of traffic signs in real-world images: The German Traffic Sign Detection Benchmark". In: *The 2013 international joint conference on neural networks (IJCNN)*. Ieee. 2013, pp. 1–8.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[16] *ResNet-18 implements Cifar-10 image classification Pytorch*. https://www.programmersought.c Accessed: 2021-03-27.

[17] Zhi Chen, Yijie Bei and Cynthia Rudin. "Concept whitening for interpretable image recognition". In: *Nature Machine Intelligence* 2.12 (2020), pp. 772–782.

[18] Lei Huang, Yi Zhou, Fan Zhu, Li Liu and Ling Shao. "Iterative normalization: Beyond standardization towards efficient whitening". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 4874–4883.

[19] Zaiwen Wen and Wotao Yin. "A feasible method for optimization with orthogonality constraints". In: *Mathematical Programming* 142.1 (2013), pp. 397–434.

[20] Nature Manufacture. *Windridge City*. https://naturemanufacture.com/windridge-city/. Accessed: 2021-04-01.

[21] Microsoft AI Research. *AirSim*. https://github.com/microsoft/AirSim. Accessed: 2021-04-01.

[22] Digital Ruby (Jeff Johnson). *Rain Maker - 2D and 3D Rain Particle System for Unity*. https://assetstore.unity.com/packages/vfx/particles/environment/rain-maker-2d-and-3d-rain-particle-system-for-unity-34938?aid=1011lGnLutm$_source = aff$. Accessed: 2021-04-01.

[23] Final Form Studio. *Sedan car - 01*. https://assetstore.unity.com/packages/3d/vehicles/land/sed car-01-190629. Accessed: 2021-04-02.

[24] Unity. *Wheel Collider*. https://docs.unity3d.com/Manual/class-WheelCollider.html. Accessed: 2021-04-02.

[25] Paper Plane Tools. *OpenCV plus Unity*. https://assetstore.unity.com/packages/tools/integratio plus-unity-85928. Accessed: 2021-04-02.