Driver Monitoring System for Outside World Context

Adrian-Răzvan Iordache

coordinated by

Associate Professor Bogdan Alexe

Faculty of Mathematics and Computer Science

University of Bucharest

Adrian-Răzvan Iordache: *Driver Monitoring System for Outside World*

MSc thesis, June 2022

Contact: *adrian.razvan.iordache@gmail.com*

This MSc thesis was written in LaTeX, with special thanks to Bogdan Alexe and Andrei Manolache for providing this template.

Abstract

Computer vision represents an interdisciplinary scientific sub-field of artificial intelligence with the sole purpose of computers/machines extracting and generating information from visual input, and from that gaining a high-level perception of the world. The fast growth of hardware capabilities and the academic discoveries of the research community allowed deep learning models to establish dominance over the already existent computer vision techniques, improving the state-of-the-art performance on various tasks and reducing the need for human-level interaction with the visual input.

Based on those improvements, a variety of tasks from the computer vision domain, such as face recognition, object detection, semantic segmentation, and pose detection, found their need in various fields like robotics, medicine, military, and automotive industry.

This thesis aims to bring a solution that improves the domain of driver monitoring and assistance for video telematics on embedded devices. We propose a solution based on a single camera visual input for Safe Distance Warning systems and Lane Departure Warning systems based on multiple stacked deep neural networks for various tasks and a computer vision algorithm for detection interpolation using accumulative features masks.

In the Introduction (Chapter 1), we present a broader perspective of the problem, the context in which we are solving it, and a more detailed structure of the thesis.

In Chapter 2, we describe the concepts, techniques, optimization methods, building blocks, and specialized deep learning architectures used during this thesis's research and development stages.

Finally, in Chapter 3, we present our in-depth solution, the problems encountered during development, and the ideas that helped solve them. Chapter 4 presents real-world results of the system and statistical measures quantified from a large number of vehicles.

The Chapter 5 ends this thesis based on a summarized description of our contribution to the domain driver monitoring systems on embedded devices and presents possible future directions for improving our solution.

There's always an answer to everything. — Louis Zamperini

Acknowledgements

The research for this thesis was done during the last two years at idrive's AI Advanced Development department for research and engineering.

I am very thankful to my supervisor Professor Dr. Bogdan Alexe for all the guidance and the provided supervision during this period and also for the great amount of knowledge shared in his courses, Advanced Machine Learning and Computer Vision.

I want to extend my deepest gratitude to the Chief Technology Officer of idrive's AI, Dr. Vlad Văduva for long hours of sharing ideas, discussions, and explanations, for his personal and professional counseling, and nonetheless for his inexhaustible calm and patience.

He will always be a great friend and an outstanding mentor.

Also, from idrive AI, I want to acknowledge the help and contribution of the Solution Architect and Senior Embedded Engineer, Justinian-Robert Popa, for low-level integration advice and innovative late-night discussions.

I would also like to thank professors Radu Ionescu, Marius Popescu, Marina Cidota, and laboratory assistant Antonio Barbălau for the excellent courses taught during those last two years.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Problem statement

One of many domains where artificial intelligence continues to expand is the video telematics industry. The fast growth of portable machine learning hardware accelerators allows the deployment of deep learning solutions on embedded devices for various tasks.

As expected, using an embedded device instead of a local GPU or a Cloud Provider for production deployment drastically increases the complexity of the problem. Many bottlenecks, like the inference speed of the model on the device, numerous unsupported tensor layers by the hardware accelerator, cross-compiled libraries, or undocumented APIs become daily problems to confront in trying to develop a solution for a specific task on an embedded device.

This enabled the research community to create a sub-domain of specialized low latency architectures for Computer Vision. Those can be compiled for specific devices with custom OS systems like Linux, Android, and iOS, for inference purposes only.

Our thesis research and development project focuses on generating a solution for monitoring the interactions between the driver and the surrounding outside context. We are interested in quantifying the risk of dangerous driver behavior concerning the following:

- Keeping a safe distance, concerning our speed, between our vehicle and the in front vehicles, motorcycles, and pedestrians
- Speeding near crosswalks or crowded areas
- Crossing the continuous lane

The deployment stage of our solution is done on an embedded device, a Qualcomm Snapdragon processor with a machine learning hardware accelerator incorporated.

Our devices analyze simultaneously the input from two cameras.

On the inside camera, we have features like facial recognition, detecting distracting driver behavior, cell phone detection, and seat belt detection.

The outside camera has the features presented above for monitoring the driver's behavior in the outside environment.

Other machine learning based features like accident prediction, detecting hard-breaking, hard-cornering, and hard-accelerating patterns based on accelerometer input are available also on our device.

The device, besides the two cameras and the accelerometer, has GPS, Bluetooth, 4G, a Wi-FI module, and an SD card for data storage and can continuously record the driver while the vehicle is started.

With this device instead of needing to watch the full route traveled by the driver, we use the AI features for recommending segments of interest from the continuous recordings, while still maintaining those recordings if ever needed by the fleet manager.

The context behind our device is necessary because the number of features running concurrently impacts the inference speed of our solution reducing from 5-25 FPS to 2-12 FPS depending on the architecture used.

With those details presented, we move forward by describing the contribution of this thesis to the domain of video telematics.

1.2 Contribution of the thesis

This thesis aims to improve the Advanced Driver Assistance Systems (ADAS) for monovision embedded devices.

We propose a solution based only on visual input for estimating the distances between our vehicle and the surrounding objects, and also a warning system for the driver when the vehicle it's crossing the white continuous lane.

Using for development a Snapdragon processor with a machine learning hardware accelerator integrated, our main focus during this thesis is the trade-off between model complexity/accuracy and inference time.

Another frequently encountered problem during the research stage is the lack of support for certain tensor operations to be accelerated on the device.

Those types of problems are regularly experienced when developing Computer Vision systems for embedded devices.

Our solution is based on Deep Learning architectures and Computer Vision methods.

The solution represents a stacked multi-network approach for various tasks followed by a feature mask interpolation algorithm, as a fail-safe mechanism in case of losing detected objects for short intervals of time in dynamic contexts and certain cases.

We summarize our ideas in the following lines:

1. A **base network** used for object detection over different classes
2. Three **stacked neural networks** used over the base network for solving various tasks such as:
    - Estimating the distance between our vehicle and the detected objects
    - Detecting the objects in front of the vehicle, invariant to the camera position on the windshield
3. A computer vision interpolation algorithm for predicting bounding boxes used for lane detection

During this thesis, we also present the encountered problems for each approach used and the thought process for solving those problems, quantified in different indicators and evaluation metrics.

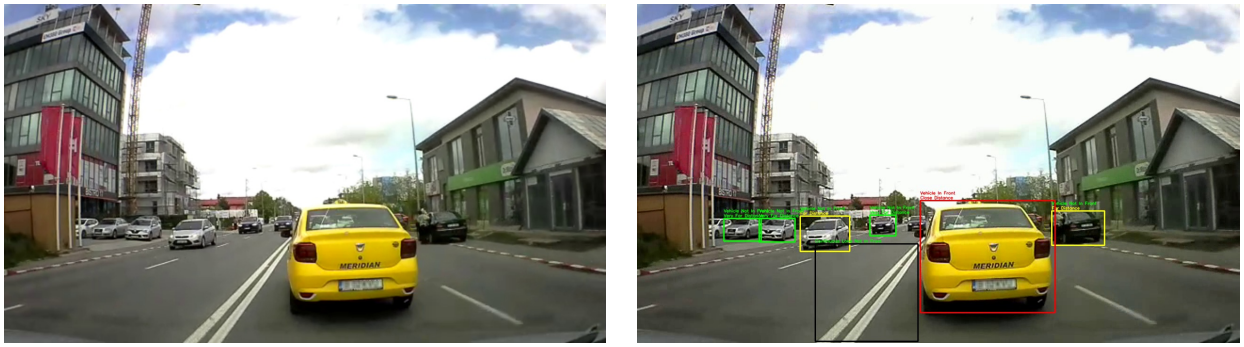
After presenting the training and optimization process with the respective validation metrics, our final results are presented in a real-world environment.

As a preview of our results, on the next page, we present four different scenarios and how our system responded.

In the following images, we have:

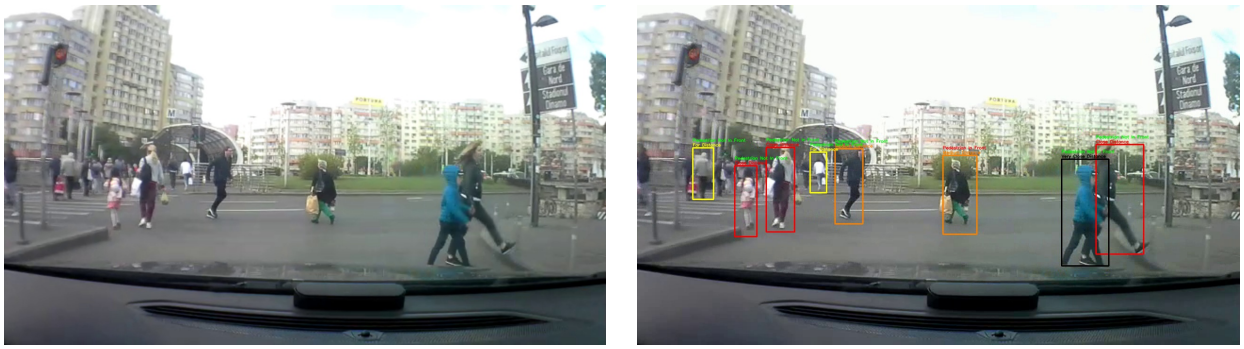
- Location of the objects based on the bounding box
- The predicted object class above the bounding box
- *In front* / *Not in front* label placed above the predicted object class
- The estimated distance presented based on the color of the bounding box, from very close to very far objects in: Black, Red, Orange, Yellow, Green.

  The distance is not estimated for the continuous lanes.



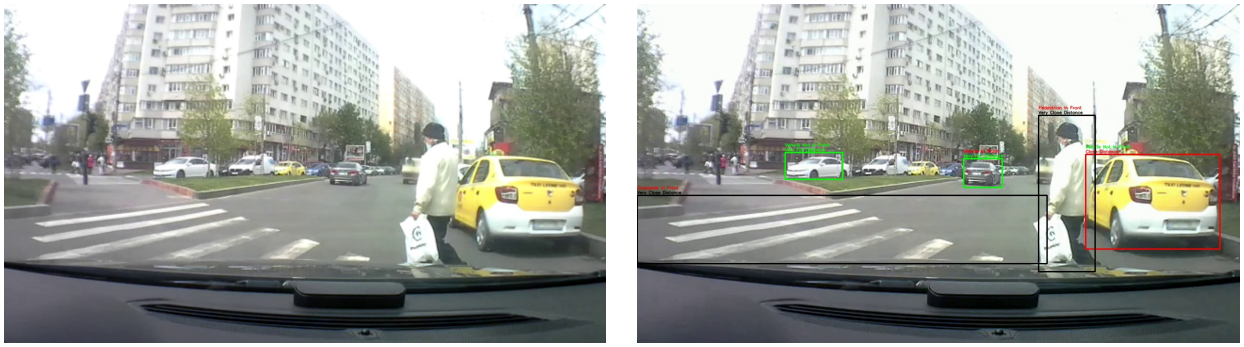
(a) (b)

Figure 1.1: Multiple vehicles scenario. *(a) Raw input image (b) Predicted objects in image. Both SDW and LDW systems work simultaneously, with all objects detected in front or not in front, and the estimated distances marked based on the color of the bounding box.*



(a) (b)

Figure 1.2: Crowded area scenario. *(a) Raw input image (b) Predicted objects in image. Multiple pedestrians are detected with the estimated distance and the in-front labels above. Used for detecting when the driver speeds in crowded areas.*



(a) (b)

Figure 1.3: Crosswalk with pedestrians crossing scenario. *(a) Raw input image (b) Predicted objects in image. Detected crosswalk for speed-limiting in such areas and detected in front pedestrian when crossing.*

(a) (b)

Figure 1.4: Vehicle crossing the continuous lane scenario. *(a) Raw input image (b) Predicted objects in image. Again, both SDW and LDW systems work simultaneously, maintaining a possible warning based on the distance/speed relation with the vehicle in front, and warning the driver for crossing the lane.*

1.3 Organization of the thesis

The structure of the next chapters is organized as follows:

In Chapter 2, we introduce various concepts used in our solution.

Here we discuss in-depth details about:

- **Active Learning**, approach used for selective sampling annotation
- **Adversarial Validation**, as a learning methods for determining if the two datasets came from different distributions
- **Weight Quantization**, necessary for running deep learning models on embedded devices
- **Neural Architecture Search**, used for efficient model generation

Also, Chapter 2 contains a detailed description of the deep learning models available in our context.

Chapter 3 presents our solution. We describe the objectives of the final system, the possible approaches for each task of the system, the limitations and problems encountered, the thought process and intuition for solving those problems, experiments, and validation metrics used for sustaining our beliefs.

Our system provides a solution for the following problems:

1. **Estimating distances** between camera and various objects based on a single frame
2. In-depth analysis between the **end-to-end approach** and **multi-network method** for our system
3. **Detecting the objects in front** of our vehicle invariant to the camera position on the windshield
4. Algorithm for **detection interpolation** between consecutive frames in dynamic contexts

5. An **active learning approach for selective data annotation** and training object detection pipelines

6. A **multi-stage method for hyper-parameter search** and validation in high dimensional spaces

7. Using **adversarial validation for evaluating data augmentation** techniques of tabular data

In Chapter 4, we test our system in a real-world environment, presenting images after the inference stage and statistical measures plotted over a large number of vehicles using the system.

Finally, in Chapter 5, we conclude our thesis by presenting a summarized description of our contribution and possible improvements to our system.

2 Fundamental Notions

2.1 Active Learning

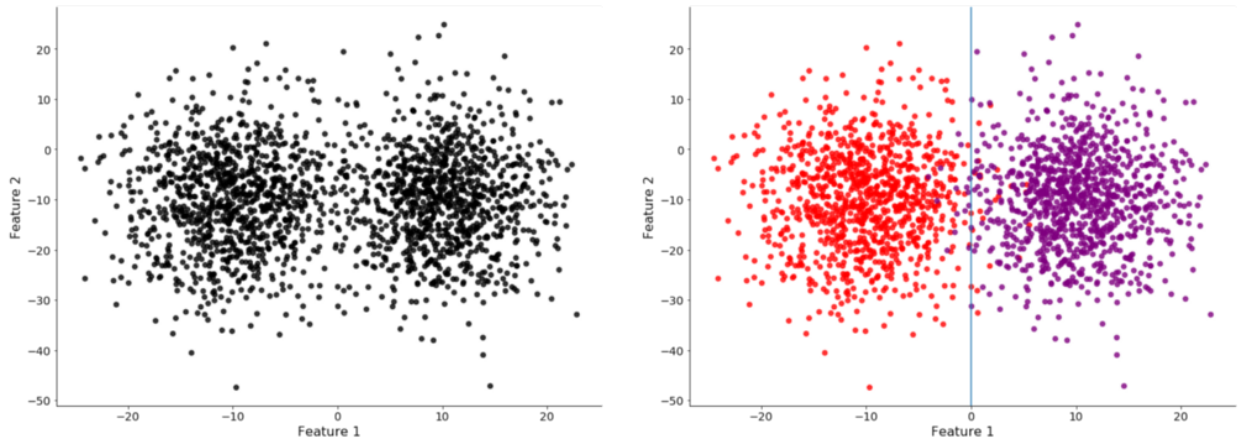
One of the main struggles for supervised learning approaches is the process of data collection and annotation. For machine learning and especially for deep learning having a large and diverse dataset it's a well-known requirement for achieving generalization over a certain problem.

Assuming a standard context, where we have a considerable amount of unlabelled data and the manual annotation process is expensive, the presented requirement becomes difficult to tackle. In those scenarios, we consider **Active Learning** [1], [2], [3] as a possible solution.

Active Learning represents a strategy for prioritising the data samples in the labeling process, which usually results in significant improvements.

Let's consider the following examples. In Figure 2.1 (a), we have two clusters of unlabelled data points for a two-dimensional input problem.

Assuming a complete annotation of the dataset, we obtain the optimal model for separating the two clusters, represented in Figure 2.1 (b).

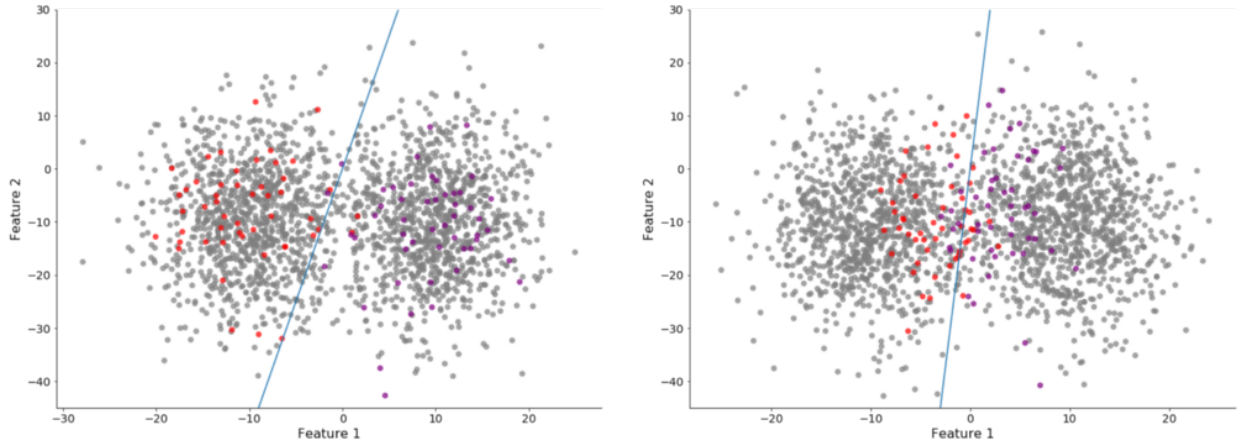


(a) (b)

Figure 2.1: Unlabelled data and optimal model. *(a) The initial unlabelled data clusters. (b) The model with the smallest classification error, obtained based on the completely annotated dataset.*

Taking into consideration the fact that the process of annotation can be expensive, we select for labeling a random subset of data points, obtaining the binary classification model represented in Figure 2.2 (a).

Instead of a random approach, we consider a selective selection process for data annotation based on active learning, We obtain the following separation boundary, Figure 2.2 (b).



(a) (b)

Figure 2.2: Random sampling compared to active learning. *(a) Classification model generated by training on random samples. (b) Classification model generated by training on selected samples from an active learning pipeline.*

As we can observe, active learning results in a better classification model, closer to the optimal one presented in Figure 2.1 (b).

Very simple architecture for active learning can be presented as:

1. Manually annotate a small random subset of the unlabelled data
2. Train a model m on the annotated dataset
3. Using the model m predict each point in the remaining data samples
4. Based on model prediction establish a prioritisation score for future labeling
5. Add the data points for manual annotation based on the priority score
6. Repeat the previous steps

The process presented above can be seen in the Figure 2.3:

As possible **prioritisation scores**, we define the following:

1. **Least Confidence Prediction**: Samples with lower prediction confidence are annotated first.

$$S_{LC} = \underset{x}{\text{argmax}}(1 - P(\hat{y} \mid x)), \text{ where } \hat{y} = \underset{y}{\text{argmax}} P(y \mid x) \tag{2.1}$$

2. **Margin Sampling**: Samples, where the difference between the first predicted classes is small, are annotated first.

$$S_{MS} = \underset{x}{\operatorname{argmin}} \left(P\left(\hat{y}_{\max} \mid x\right) - P\left(\hat{y}_{\max -1} \mid x\right) \right) \tag{2.2}$$

3. **Entropy Reduction**: Samples with higher entropy between predictions are annotated first.

$$S_E = \underset{x}{\operatorname{argmax}} \left(-\sum_i P\left(\hat{y}_i \mid x\right) \log P\left(\hat{y}_i \mid x\right) \right) \tag{2.3}$$

Those presented above represent regular concepts for defining an active learning pipeline to improve the probability of obtaining optimal models when dealing with high amounts of unlabelled data.

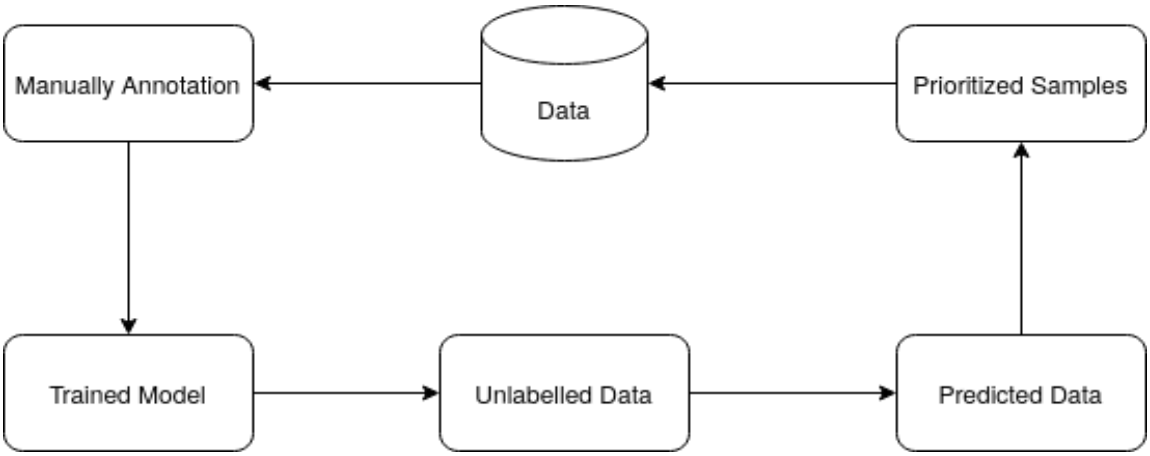


Figure 2.3: Standard active leaning pipeline.

2.2 Adversarial Validation

Most of the time, in supervised learning problems, we start from the assumption that our dataset is independent and identically distributed (i.i.d).

For a brief explanation behind the i.i.d. concept, we can think about a succession of multiple coin tosses, assuming that we have a fair coin.

Every coin toss in the succession is a stochastic process, that is unrelated to the other coin tosses. Based on this, we consider a coin toss to be **independent** random variable.

Besides that, every coin toss maintains the same probability distribution of the outcome, with a probability of 50% for landing tails and 50% for landing heads. For this reason, we consider each coin toss in the succession to be **identically distributed**.

Based on those presented above, we can conclude that a succession of multiple coin tosses is a random

variable independent and identically distributed.

Going back to the initial statement, in supervised learning, we often start from the assumption that our dataset is independent and identically distributed, but for various reasons, this might not be the case.

We present a couple of examples when the i.i.d assumption can be violated:

1. When datasets are composed of samples labeled by different annotators
   (e.g. in medical problems, when some samples are annotated by doctors and others by medical school students)

2. When our problem contains a temporal component
   (e.g. sequences of frames from video segments)

3. When data is artificially generated or augmented

Any of those three scenarios represent a problem when generating training and validation splits that can result in overfitting. The main reason for that is based on the requirement of having the training set and the validation set sampled from the same distribution.

Assuming that the training set is not well sampled, we risk our model learning the noise in the data, instead of the pattern from the problem we want to solve. Similarly, when having a validation or testing set with samples from the outside of the problem distribution, we risk optimizing our models for representing noise and not generalizing in a real-world environment.

Based on those problems, we need to define a similarity measure between the distributions of training samples and validation samples.

A possible solution for those problems it's called **Adversarial Validation** [4].

Instead of having statistical indicators or mathematical equations for defining a similarity score for those distributions, we induce a learning algorithm for determining the similarity of a sample between those two distributions. Based on this approach, we study the degree of separability between two datasets based on a learned estimator.

The process of implementation is straightforward.

Assuming a dataset:

$$D = \left\{ (x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m) \;\middle|\; \begin{array}{l} y_i \in \mathbb{R}, i = \overline{1, m} \\ x_i \in \mathbb{R}^n, i = \overline{1, m}, x_i = (x_{i1}, x_{i2}, \ldots x_{in}) \end{array} \right\} \tag{2.4}$$

From a label stratified split of the dataset D, we generate a training dataset D_T and a validation dataset

D_V. For each sample in D_T, we replace $y_i, i \in 1 \dots m_T$ with 0 and for each sample in D_V, we replace $y_i, i \in 1 \dots m_V$ with 1.

Based on this, we obtain:

$$D_T = \{(x_1, 0), (x_2, 0), \dots (x_{m_T}, 0) \mid x_i \in \mathbb{R}^n, i = \overline{1, m_T}\} \tag{2.5}$$

$$D_V = \{(x_1, 1), (x_2, 1), \dots (x_{m_V}, 1) \mid x_i \in \mathbb{R}^n, i = \overline{1, m_V}\} \tag{2.6}$$

We combine and shuffle D_T and D_V. Using another stratified split based on the new label, we generate D_T^* and D_V^*. We consider M a binary classification model trained on D_T^* and validated on D_V^*. Based on M, we define the similarity between the distributions of two datasets.

To keep things simple, we define the following formula:

$$S(D_T, D_V) = \begin{cases} 1, & \text{if } ROC\_AUC(Y^*_{D_V}, \hat{Y}_{D_V})) \leq threshold \\ 0, & \text{otherwise} \end{cases} \tag{2.7}$$

where $ROC\_AUC(Y^*_{D_V}, \hat{Y}_{D_V}))$ represents the area under receiver operating characteristic curve over $Y^*_{D_V} = \{y_i^* \in D_V^*, i = \overline{1, m_{D_V^*}}\}$ and $\hat{Y}_{D_V} = \{M(x_i^*), x_i^* \in D_V^*, i = \overline{1, m_{D_V^*}}, x_i = (x_{i1}, x_{i2}, \dots x_{in})\}$.

Informally, we describe the process of adversarial validation, by training a learning algorithm to predict the initial source of a sample from two possible datasets.

If the learning algorithm can accurately predict which samples came from which particular dataset then we can be confident that the distributions of those two datasets are different.

If our model has the AUC close to random chance (50%), we consider that those two datasets came from the same distribution, because the learning algorithm can't find a pattern to differentiate them.

2.3 Quantization for Neural Networks

With growing the complexity of neural networks, the number of parameters scales significantly, decreasing the inference speed and increasing the need for memory space, and also the power consumption.

A neural network is mostly represented by weights, biases, and activations, those are usually represented in memory using 32-bit float values. This type of decimal precision helps the model to represent patterns in data, achieving high accuracy but also increases the need for memory storage.

Quantization [5] represents a procedure that reduces the precision of weights, biases, and activations. This process decreases memory usage and increases the inference speed at small costs in network accuracy. Usually, this procedure is done between 32-bit floats to 16-bit floats, and even from 32-bit floats to 8-bit integers.

Besides the obvious reduction in space, the network latency is improved because the operations that are performed on integer types require fewer computations on most devices.

Another benefit is represented by the improved efficiency in power consumption based on decreasing the memory access, and also the reduced number of computations.

Despite all that, reducing the precision can produce a loss in accuracy, depending on the amount of precision lost. In most cases, quantization results in minimal losses, and it proves to be a necessary step when deploying a deep learning model on an embedded device.

In practice, there are two main methods for neural network quantization:

- Post-training quantization
- Quantization-aware training

Post-training quantization [5] represents a method that is completely done after the training stage. Assuming a regular model trained on floating-point precision, after training, we freeze the model and quantize weights, generating a model ready for deployment.

This is the easiest method for quantization, but it can produce higher accuracy losses.

A better approach for quantization is **Quantization-aware training** [5]. Using this method, we accumulate the quantization-related errors with neural network loss and based on the optimizer, during training the weights are adjusted minimizing the overall error.

This approach usually results in a much lower accuracy loss compared to the post-training quantization method. For a better understanding of the quantization trade-offs, TensorFlow [6], provides a benchmark between accuracy, latency, and memory for various image classification models in Table 2.1.

Table 2.1: Benchmark of model quantization in TensorFlow [6]

Model	Top-1 Accuracy (Original)	Top-1 Accuracy (Post Training Quantized)	Top-1 Accuracy (Quantization Aware Training)	Latency (Original) (ms)	Latency (Post Training Quantized) (ms)	Latency (Quantization Aware Training) (ms)	Size (Original) (MB)	Size (Optimized) (MB)
Mobilenet-v1-1-224	0.709	0.657	0.70	124	112	64	16.9	4.3
Mobilenet-v2-1-224	0.719	0.637	0.709	89	98	54	14	3.6
Inception_v3	0.78	0.772	0.775	1130	845	543	95.7	23.9
Resnet_v2_101	0.770	0.768	N/A	3973	2868	N/A	178.3	44.9

We present a descriptive representation of quantization-aware training in Figure 2.4, provided from [7].

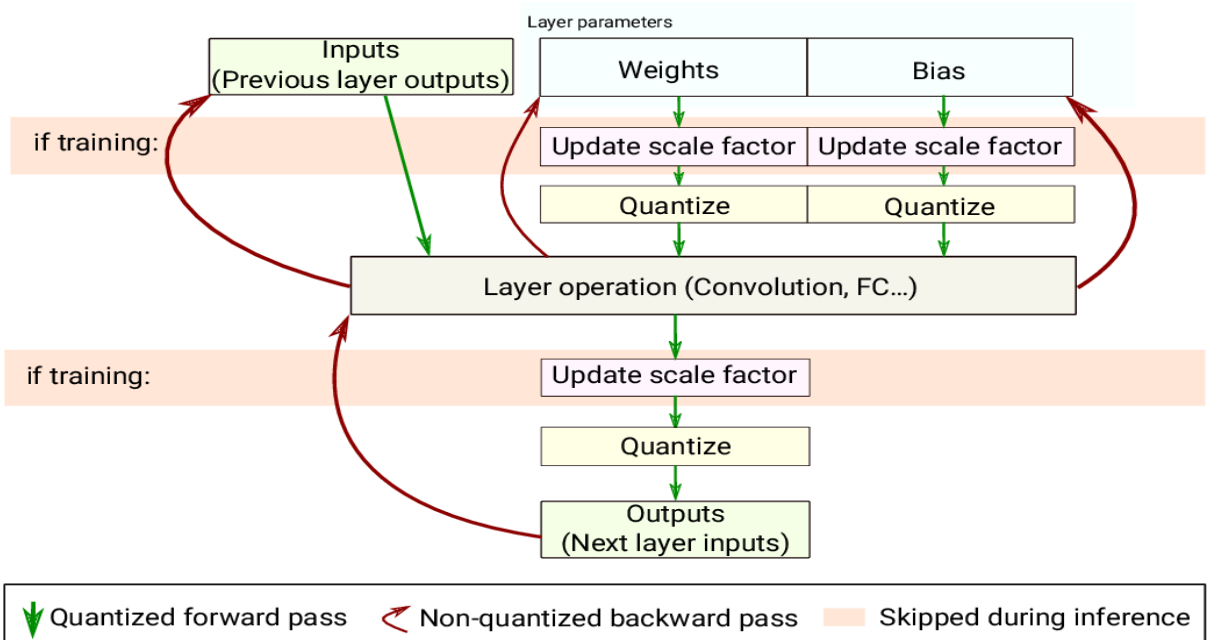


Figure 2.4: Quantization aware-training [7]

At the end of this section, we present the available quantization techniques provided by TensorFlow in Table 2.2.

Table 2.2: Quantization techniques in TensorFlow [6]

Technique	Data requirements	Size reduction	Accuracy	Supported hardware
Post-training float16 quantization	No data	Up to 50%	Insignificant accuracy loss	CPU, GPU
Post-training dynamic range quantization	No data	Up to 75%	Smallest accuracy loss	CPU, GPU (Android)
Post-training integer quantization	Unlabelled representative sample	Up to 75%	Small accuracy loss	CPU, GPU (Android), EdgeTPU, Hexagon DSP
Quantization-aware training	Labelled training data	Up to 75%	Smallest accuracy loss	CPU, GPU (Android), EdgeTPU, Hexagon DSP

2.4 MobileNet Family

In the year 2017, a new sub-domain of deep learning research for computer vision began based on the introduction of MobileNet V1 [8].

This architecture was presented as capable of running in environments with low hardware capabilities, at high inference speeds, and with a minimum loss of accuracy, compared to already existing models.

Those types of models will later represent the stepping stone for deploying deep learning solutions on

embedded systems.

The main concept behind this architecture that allowed this discovery can be summarized as ”replacing a full convolutional operator with a factorized version that splits convolution into two separate layers” [8]. Those types of layers are named by the authors Depthwise Separable Convolutions.

Two years later, at the beginning of 2019, the second member of this family was added, MobileNet V2 [9]. Introducing concepts such as Inverted Residuals or Linear Bottlenecks, MobileNet V2 improved the state-of-the-art performance of mobile models on various tasks and benchmarks.

A more in-depth description and analysis of efficiency for Single Shot Detector, MobileNet V1, and MobileNet V2, and the corresponding layers for each architecture, can be found in previous work. [10]

The last iteration of mobile architectures was MobileNet V3 [11]. The improvements provided by MobileNet V3 are represented in the following.

The main difference between MobileNet V3 and its predecessors it’s based on the fact that instead of using human hand-crafted layers and architectures, it uses two algorithms specialized for optimizing the neural network architectures.

During the paper [11], the authors describe using both Neural Architecture Search [12] and NetAdapt [13] for optimizing the architecture based on the trade-off between accuracy and latency.

Neural Architecture Search (NAS) represents a sub-field of AutoML, which aims to automate all processes in machine learning and deep learning. Using NAS in 2016, the authors achieved state-of-the-art performances for language modeling and image recognition tasks using a reinforcement learning approach. We describe the summarized algorithm of Neural Architecture Search based on the Figure 2.5, presented in the original paper [12].

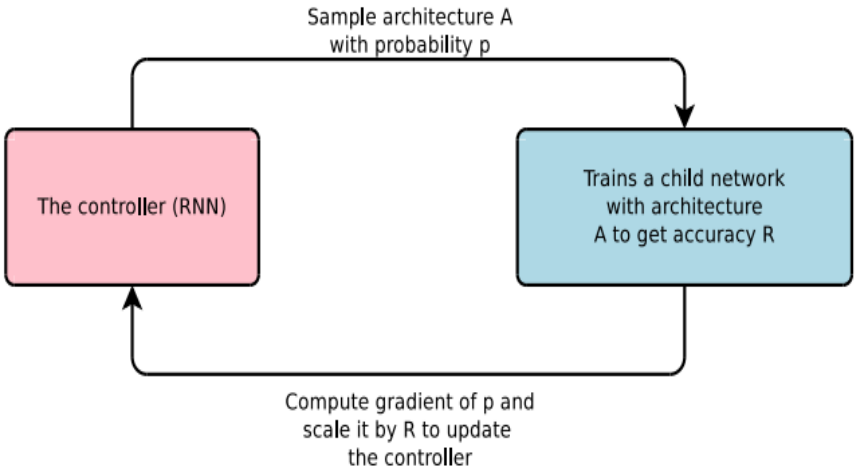


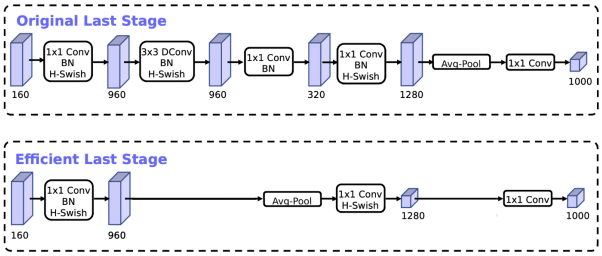
Figure 2.5: Neural Architecture Search overview. [12]

Based on Neural Architecture Search, we discover optimized architectures for specific purposes. Using reinforcement learning, NAS, evaluates a space of possible neural network architectures based on search strategy, selecting the models that maximize an objective function.

Going back to MobileNet V3, as previously specified, the authors of the [11], use NAS for designing efficient sub-modules that can be stacked together.

Based on the result from Neural Architecture Search, the authors efficiently redefine the number of filters in each convolutional layer using NetAdapt.

Other improvements brought by MobileNet V3 are manually redesigning certain layers for efficiency, Figure 2.6, (a) and the addition of Squeeze-and-Excite modules to the inverted residuals blocks from MobileNet V2, Figure 2.6 (b).



(a)

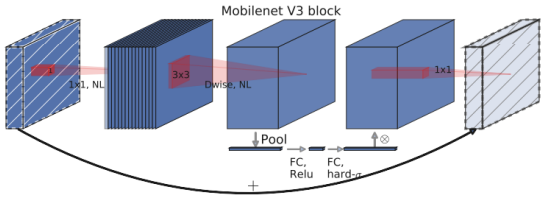


Figure 4. MobileNetV2 + Squeeze-and-Excite [20]. In contrast with [20] we apply the squeeze and excite in the residual layer. We use different nonlinearity depending on the layer, see section 5.2 for details.

(b)

Figure 2.6: MobileNet V3 layer improvements. *(a) Efficient redesign of the last stage of the architecture. (b) MobileNet V3 block, added Squeeze-and-Excite module to MobileNet V2 inverted residuals layers. Both illustrations are provided from [11].*

The last improvement from MobileNet V3 is redefining the swish activation function introduced by [14], using neural architecture search, which proved to increase the accuracy but also being computationally expensive.

The authors redefine the formula of the swish function, Equation 2.8, into the hard-swish function, Equation 2.9 as follows:

$$\text{swish}\ (x) = x \cdot \sigma(x) \tag{2.8}$$

$$\text{h} - \text{swish}(x) = x\frac{\text{ReLU6}(x+3)}{6} \tag{2.9}$$

With all the improvements presented, we move forward by presenting the overall structure of the architecture. The paper [11] uses two definitions of the model: MobileNet V3 Large, Figure 2.7 (a), and MobileNet V3 Small, Figure 2.7 (b).

Input	Operator	exp size	#out	SE	NL	s
$224^2 \times 3$	conv2d	-	16	-	HS	2
$112^2 \times 16$	bneck, 3x3	16	16	-	RE	1
$112^2 \times 16$	bneck, 3x3	64	24	-	RE	2
$56^2 \times 24$	bneck, 3x3	72	24	-	RE	1
$56^2 \times 24$	bneck, 5x5	72	40	✓	RE	2
$28^2 \times 40$	bneck, 5x5	120	40	✓	RE	1
$28^2 \times 40$	bneck, 5x5	120	40	✓	RE	1
$28^2 \times 40$	bneck, 3x3	240	80	-	HS	2
$14^2 \times 80$	bneck, 3x3	200	80	-	HS	1
$14^2 \times 80$	bneck, 3x3	184	80	-	HS	1
$14^2 \times 80$	bneck, 3x3	184	80	-	HS	1
$14^2 \times 80$	bneck, 3x3	480	112	✓	HS	1
$14^2 \times 112$	bneck, 3x3	672	112	✓	HS	1
$14^2 \times 112$	bneck, 5x5	672	160	✓	HS	2
$7^2 \times 160$	bneck, 5x5	960	160	✓	HS	1
$7^2 \times 160$	bneck, 5x5	960	160	✓	HS	1
$7^2 \times 160$	conv2d, 1x1	-	960	-	HS	1
$7^2 \times 960$	pool, 7x7	-	-	-	-	1
$1^2 \times 960$	conv2d 1x1, NBN	-	1280	-	HS	1
$1^2 \times 1280$	conv2d 1x1, NBN	-	k	-	-	1

Table 1. Specification for MobileNetV3-Large. SE denotes whether there is a Squeeze-And-Excite in that block. NL denotes the type of nonlinearity used. Here, HS denotes h-swish and RE denotes ReLU. NBN denotes no batch normalization. s denotes stride.

(a)

Input	Operator	exp size	#out	SE	NL	s
$224^2 \times 3$	conv2d, 3x3	-	16	-	HS	2
$112^2 \times 16$	bneck, 3x3	16	16	✓	RE	2
$56^2 \times 16$	bneck, 3x3	72	24	-	RE	2
$28^2 \times 24$	bneck, 3x3	88	24	-	RE	1
$28^2 \times 24$	bneck, 5x5	96	40	✓	HS	2
$14^2 \times 40$	bneck, 5x5	240	40	✓	HS	1
$14^2 \times 40$	bneck, 5x5	240	40	✓	HS	1
$14^2 \times 40$	bneck, 5x5	120	48	✓	HS	1
$14^2 \times 48$	bneck, 5x5	144	48	✓	HS	1
$14^2 \times 48$	bneck, 5x5	288	96	✓	HS	2
$7^2 \times 96$	bneck, 5x5	576	96	✓	HS	1
$7^2 \times 96$	bneck, 5x5	576	96	✓	HS	1
$7^2 \times 96$	conv2d, 1x1	-	576	✓	HS	1
$7^2 \times 576$	pool, 7x7	-	-	-	-	1
$1^2 \times 576$	conv2d 1x1, NBN	-	1024	-	HS	1
$1^2 \times 1024$	conv2d 1x1, NBN	-	k	-	-	1

Table 2. Specification for MobileNetV3-Small. See table 1 for notation.

(b)

Figure 2.7: MobileNet V3 architecture. *(a) MobileNet V3 Large. (b) MobileNet V3 Small. Both tables are provided from [11].*

At the end of the summarized presentation of the MobileNet architectures, we present the final results obtained by the authors of [11] for classification, Figure 2.8 (b), and object detection, Figure 2.8 (b), tasks.

Network	Top-1	MAdds	Params	P-1	P-2	P-3
V3-Large 1.0	**75.2**	**219**	5.4M	**51**	**61**	**44**
V3-Large 0.75	73.3	155	4.0M	39	46	40
MnasNet-A1	75.2	315	3.9M	71	86	61
Proxyless[5]	74.6	320	4.0M	72	84	60
V2 1.0	72.0	300	3.4M	64	76	56
V3-Small 1.0	**67.4**	**56**	2.5M	**15.8**	**19.4**	**14.4**
V3-Small 0.75	65.4	44	2.0M	12.8	15.6	11.7
Mnas-small [43]	64.9	65.1	1.9M	20.3	24.2	17.2
V2 0.35	60.8	59.2	1.6M	16.6	19.6	13.9

Table 3. Floating point performance on the Pixel family of phones (P-n denotes a Pixel-n phone). All latencies are in ms and are measured using a single large core with a batch size of one. Top-1 accuracy is on ImageNet.

(a)

Backbone	mAP	Latency (ms)	Params (M)	MAdd (B)
V1	22.2	270	5.1	1.3
V2	22.1	200	4.3	0.80
Mnasnet	23.0	215	4.88	0.84
V3	22.0	173	4.97	0.62
V3†	22.0	150	3.22	0.51
V2 0.35	13.7	54.8	0.93	0.16
V2 0.5	16.6	80.4	1.54	0.27
Mnasnet 0.35	15.6	58.2	1.02	0.18
Mnasnet 0.5	18.5	86.2	1.68	0.29
V3-Small	16.0	67.2	2.49	0.21
V3-Small†	16.1	56	1.77	0.16

Table 6. Object detection results of SSDLite with different backbones on COCO test set. †: Channels in the blocks between $C4$ and $C5$ are reduced by a factor of 2.

(b)

Figure 2.8: MobileNet V3 results. *(a) MobileNet V3 results for the classification task. (b) MobileNet V3 results for the object detection class. Both tables are provided from [11].*

3 Architecture of the solution proposed

In this chapter, we introduce our solution for the problem of monitoring the interactions between the driver and the outside world. Also, during this chapter, we present the thought process and the encountered problems during the research and validation stages. The solution, as announced in the beginning, is based on deep learning and computer vision methods.

The main purpose of this system is simplified into two requirements:

1. Estimating the distances from the vehicle to certain objects (e.g. pedestrians, other vehicles)

2. Detecting when the vehicle is crossing a continuous white line on the road

Those types of systems are frequently encountered in the automotive world for Advanced Driver-Assistance System (ADAS), as Safe Distance Warning (SDW) and Lane Departure Warning (LDW).

The main difference in our context is that we only use those systems for driver monitoring and scoring, not for assisted driving.

Based on those two, we start presenting our solution in the next sections.

3.1 Learning to estimate distances from a single frame

The problem of depth estimation is usually solved based on disparity maps. The concept of disparity maps represents a computer vision approach, available in a stereo vision context preferably with no hardware limitations.

In our situation, we propose a solution that does not require those two conditions to be met, allowing higher errors in the process of distance estimation, but still maintaining the notion of continuity for our problem.

Taking into consideration the hardware limitations of the embedded device and the need for a system that runs at a higher frame rate, we decide to take advantage of our machine learning hardware accelerator and solve this problem based on deep neural networks. This way our model based on an input

image predicts the class of the detected object, its coordinates, the confidence of the detection, and also the estimated distance between our vehicle and the respective object.

The standard approach in the case of image classification would be to replace the final fully connected layer, adding an extra linear head for regression and optimizing based on a weighted average between multiple loss functions.

In our current case, the problem of object detection, this approach would be significantly different and increases the complexity of the model. Taking as an example the architecture of VGG16 [15] with a Single Shot Detector (SSD) [16], for simplicity, we use only the base layers from VGG16, excluding the auxiliary extra feature layers presented in [16].

We already know that the SSD uses the feature maps of the ”backbone” network at different sizes with various priors for predicting the class object and its location.

The default architecture can be seen in Figure 3.1.

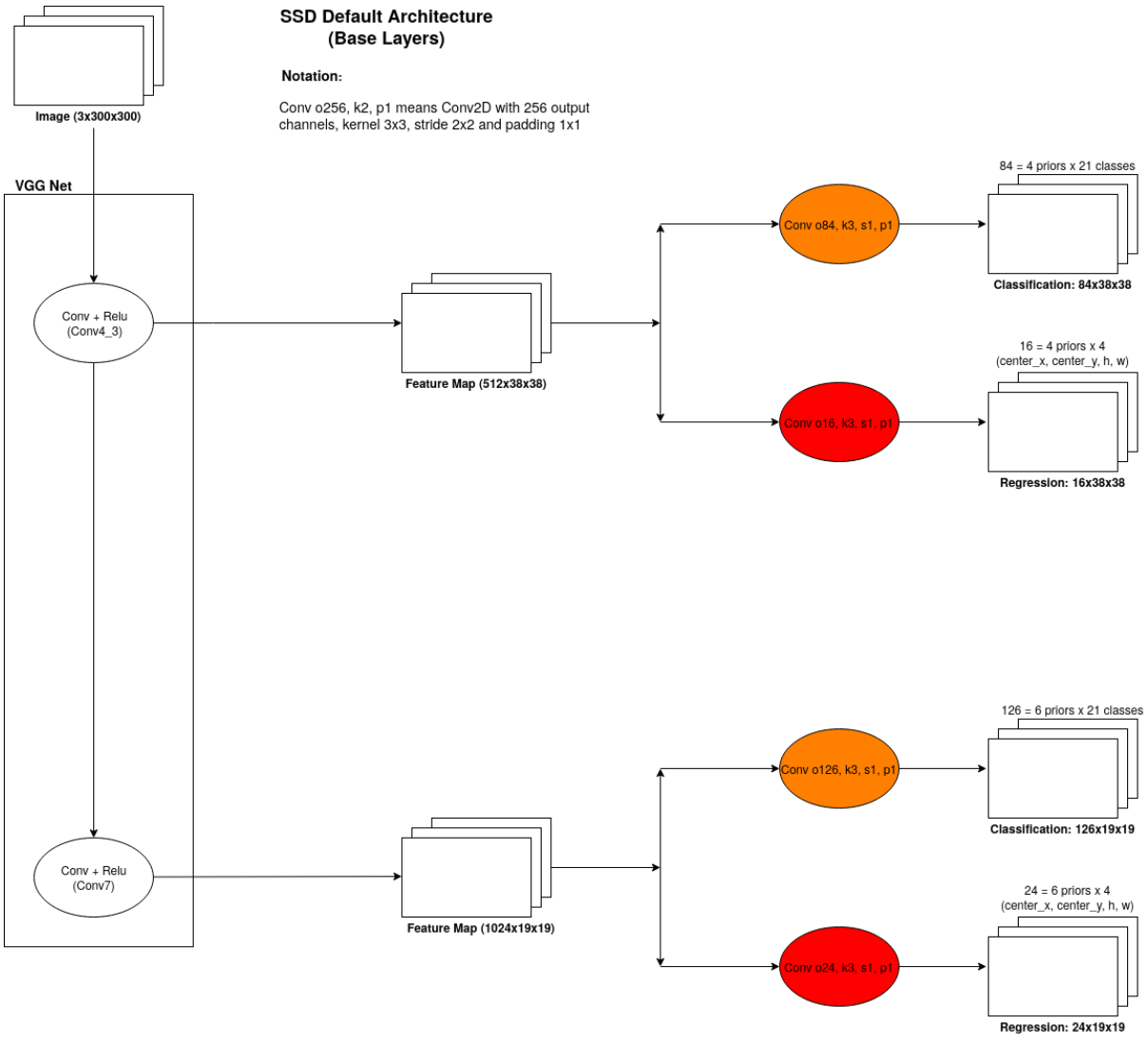


Figure 3.1: Single Shot Detector default architecture (base layers)

Based on the default architecture, being able to extract the depth of a possible object would be necessary on the same extracted feature maps to attach convolutional layers at each size level for predicting the estimated distance.

This custom approach for extracting the estimated depth of an object is presented below in Figure 3.2.

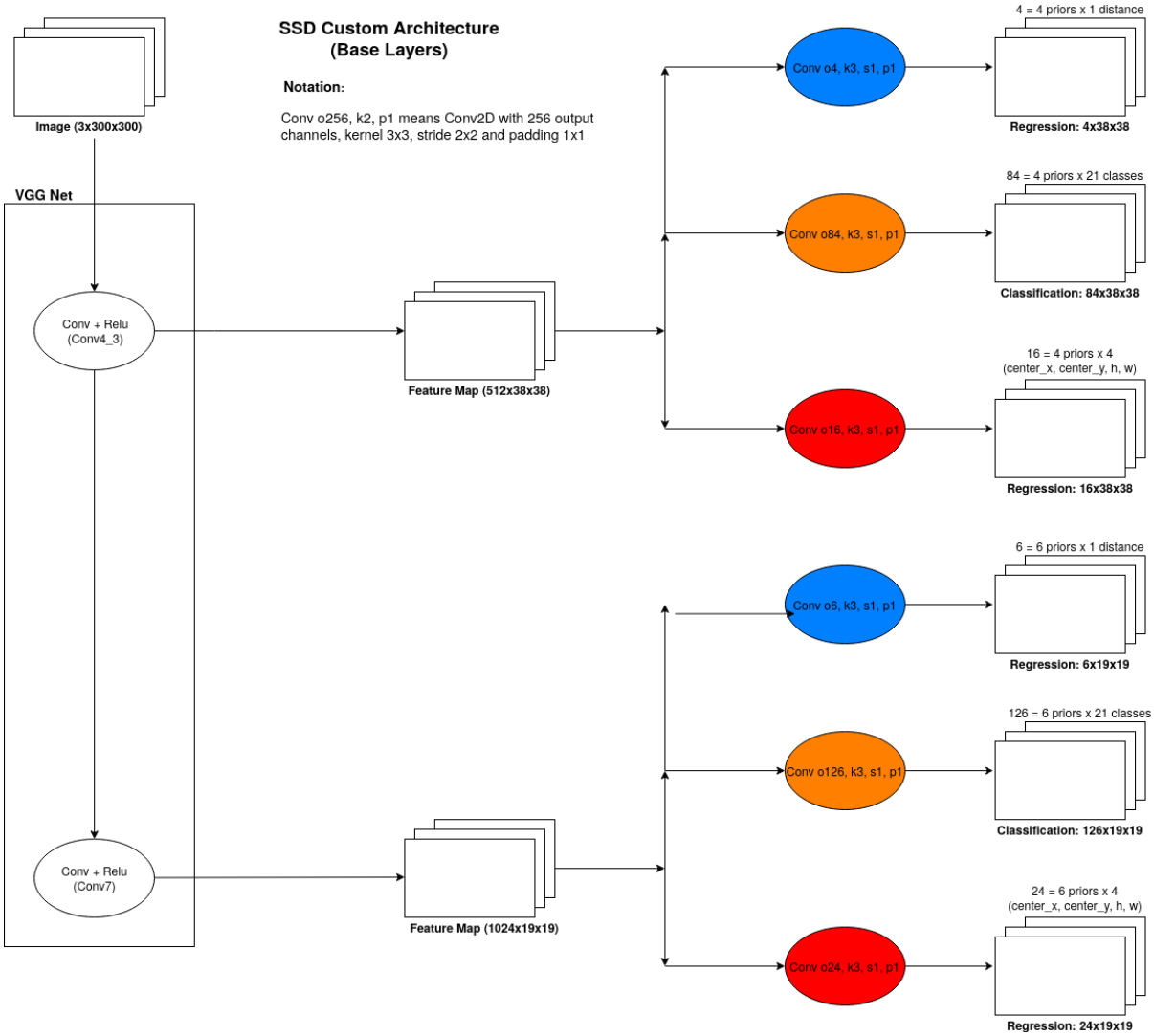


Figure 3.2: Single Shot Detector custom architecture for depth estimation (base layers)

Unfortunately, experimenting with both types of architectures we discovered, that the custom approach reduced the inference speed considerably, making it impossible to run in real-life conditions.

This represents a major bottleneck for the rest of the pipeline, we decide to shift the perspective of the problem to be able to make use of the standard models that output only coordinates, classes and scores.

The solution to this problem came with the idea of mapping the continuous space in 5 discrete intervals of depth, resulting in each initial category C having $5 \times C$ classes.

Those 5 discrete classes of distance will be labeled as:

Label	Distance Intervals
Very Close Distance	between 0-2 meters
Close Distance	between 2-5 meters
Medium Distance	between 5-10 meters
Far Distance	between 10-15 meters
Very Far Distance	above 15 meters

Particularly, in our case, this was necessary for 5 out of the 7 categories: average size vehicles, large size vehicles, pedestrians, motorcycles, and crosswalks, without taking into consideration the distance for continuous white lane or dashed white lane. This results in a classification problem with 27 classes.

As an example of our approach, in Figure 3.3, we present those 5 categories of distance in the scenario of detecting an average size vehicle.

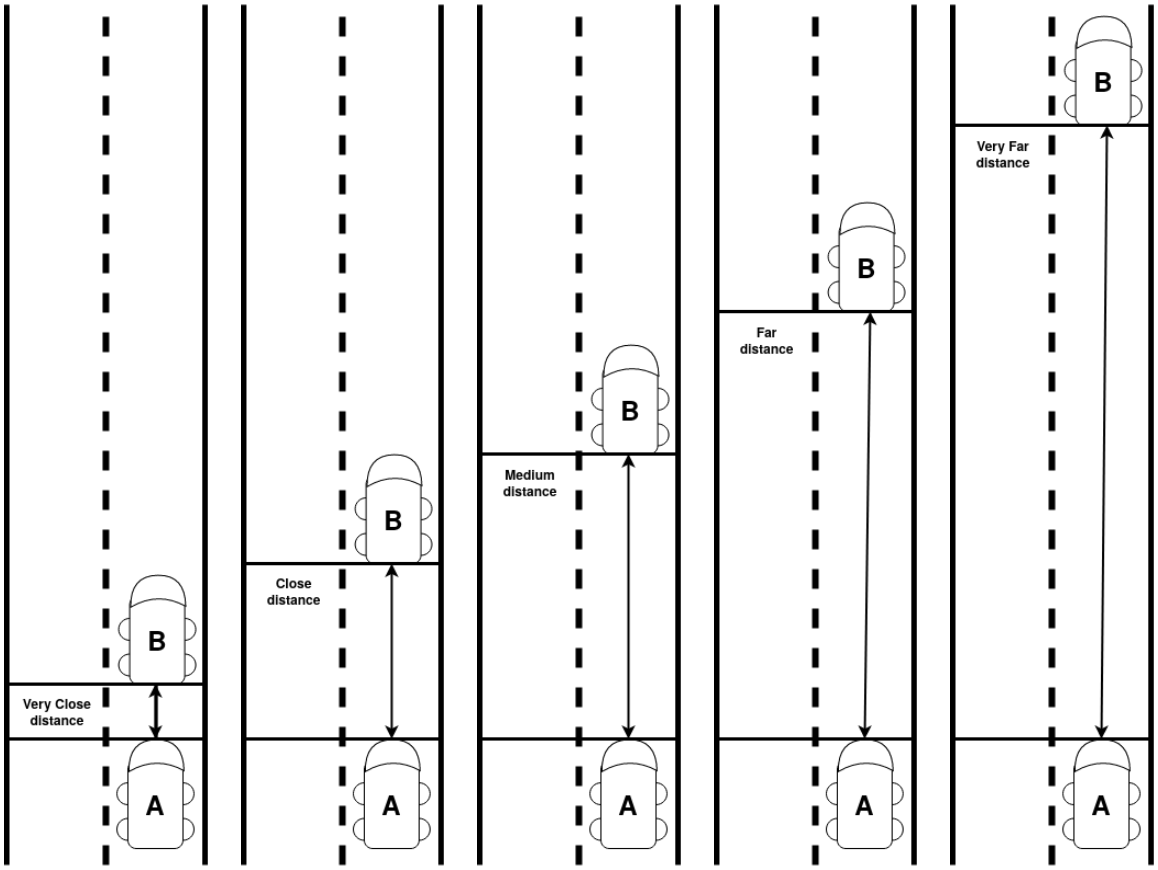


Figure 3.3: Mapping the continuous space of distances in discrete categories. *(a) Very close distance. (b) Close distance. (c) Medium distance. (d) Far distance. (e) Very Far distance.*

Establishing an annotation scheme for distance estimation in a controlled environment, with every image being annotated and verified by 4 annotators, a proof of concept model was generated.

After approximately 40,000 collected and annotated images from the real world the test results are meeting the industry standards.

The label distribution for the 40,000 dataset can be seen in the image below (Figure 3.4).

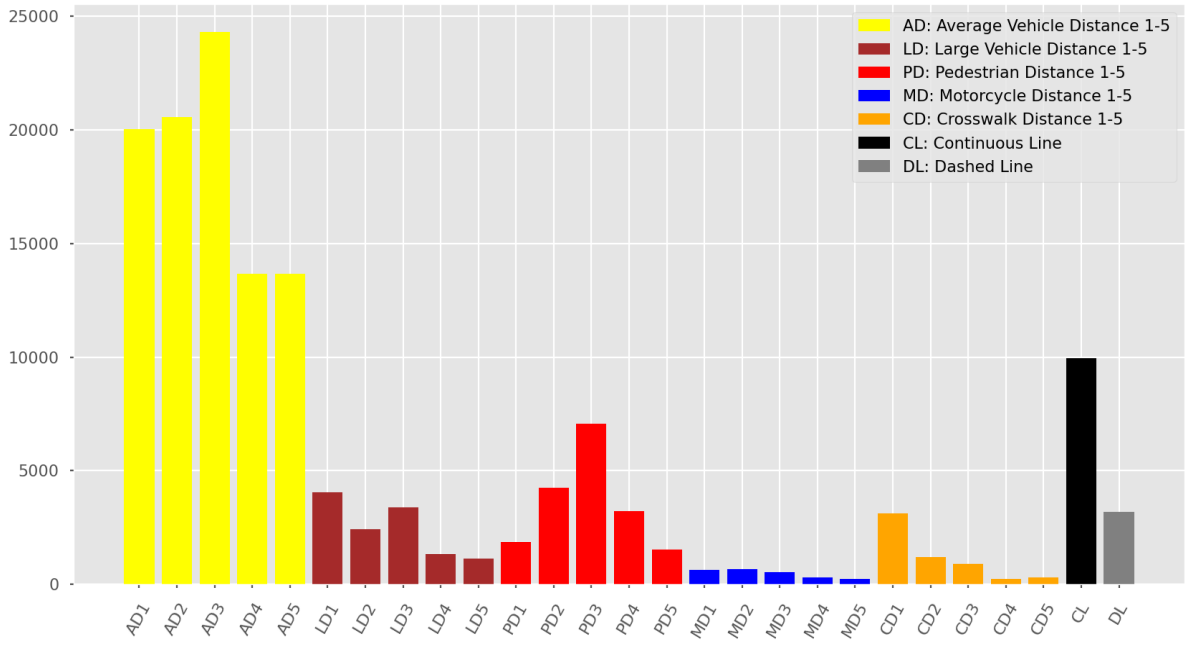


Figure 3.4: Object classes distribution over all images in dataset

As we can observe, we are dealing with a highly imbalanced problem. This situation is caused by the fact that in traffic, we are more likely to see vehicles, opposite to seeing pedestrians, motorcycles, or crosswalks. Another factor that caused this context to be imbalanced was generated by the idea of using multiple discrete intervals for each predicted category.

In the next section, we present our analysis for the current methodology over a label stratified test dataset and a possible solution that diminishes the problem of highly imbalanced classes.

3.2 End-to-End Network vs Multi-Network approach

At the end of the previous section, we noticed the problem of highly imbalanced object classes in the dataset. As we previously said, this problem is caused by two major factors: the nature of the problem, the fact that the driver in traffic interacts mostly with vehicles, lanes, and pedestrians and the context in which we approach the problem.

We chose to approach this problem by using discrete intervals of depth, splitting 5 out of 7 object categories into 5 distinct categories. This method is practical in the field and could be considered a viable solution, but from other perspectives, this solution generated some possible problems.

There are two main problems based on this approach, both of them caused by the split in depth intervals.

First problem is represented by the fact that objects further away from the camera are less present in the collected dataset. This represents an issue because using deep learning methods, our pipeline is data-driven, so having fewer samples from a distance class for an object could lead to the failure of recognition for that object. More than that, by splitting an object category into other five distinct categories, we are more likely to fail to recognize the whole category class, especially smaller classes like pedestrians or motorcycles.

The second problem was generated by using each distance category of an object as an individual class for a classification task. This it's not necessarily a problem because the decision between using a regression approach or a classification approach it's dependent on the evaluation process. Usually, training a network on a continuous problem with discrete bins using classification methods leads to a better separation between classes because we are not trying to optimize for smaller errors, we want only the accurate label. Training in the same context, using instead a regression approach can be more unstable and difficult to optimize, but it retains the notion of the neighborhood between classes, not allowing for high label estimation errors.

In our case, we lose the continuous notion of the problem, our network does not know that some classes are closer to each other than others and this could lead to higher distance estimation error. As we said in the beginning, for us it's more important retaining the neighborhood between classes than to predict the exact discrete depth interval.

Based on those two problems described above, we present a possible improvement for our method.

The main idea comes from the intuition that for **estimating the distance** between the camera and the detected object the most important features are **the size of the bounding box** (as bigger boxes tend to be closer to the camera) and the **the category of the detected object**, instead of the pixel level features that the convolutional neural network sees.

We start from the reasonable assumption that for measuring the distance between our vehicle and other objects we can use only the bounding box coordinates, and a learning algorithm can learn meaningful patterns like the size of the detection, and from that the distance.

We can quickly observe that our assumption fails in the context of detecting objects from different categories of various sizes. A possible problem it's that a large vehicle like a truck has a larger bounding box at 5 meters than a pedestrian has at 2 meters.

We solve this problem by adding the category of the detected object as the input in the learning algorithm, this way our estimator knows that there are not just boxes from the same distribution, those boxes have particular shapes and sizes based on the category predicted.

To better express our intuition, we present in Figure 3.5 two perspectives, one perspective is the ***in front of the car perspective*** and the other one is ***from above perspective***. This way, we can better understand how the size of the bounding box is used for predicting the distance.

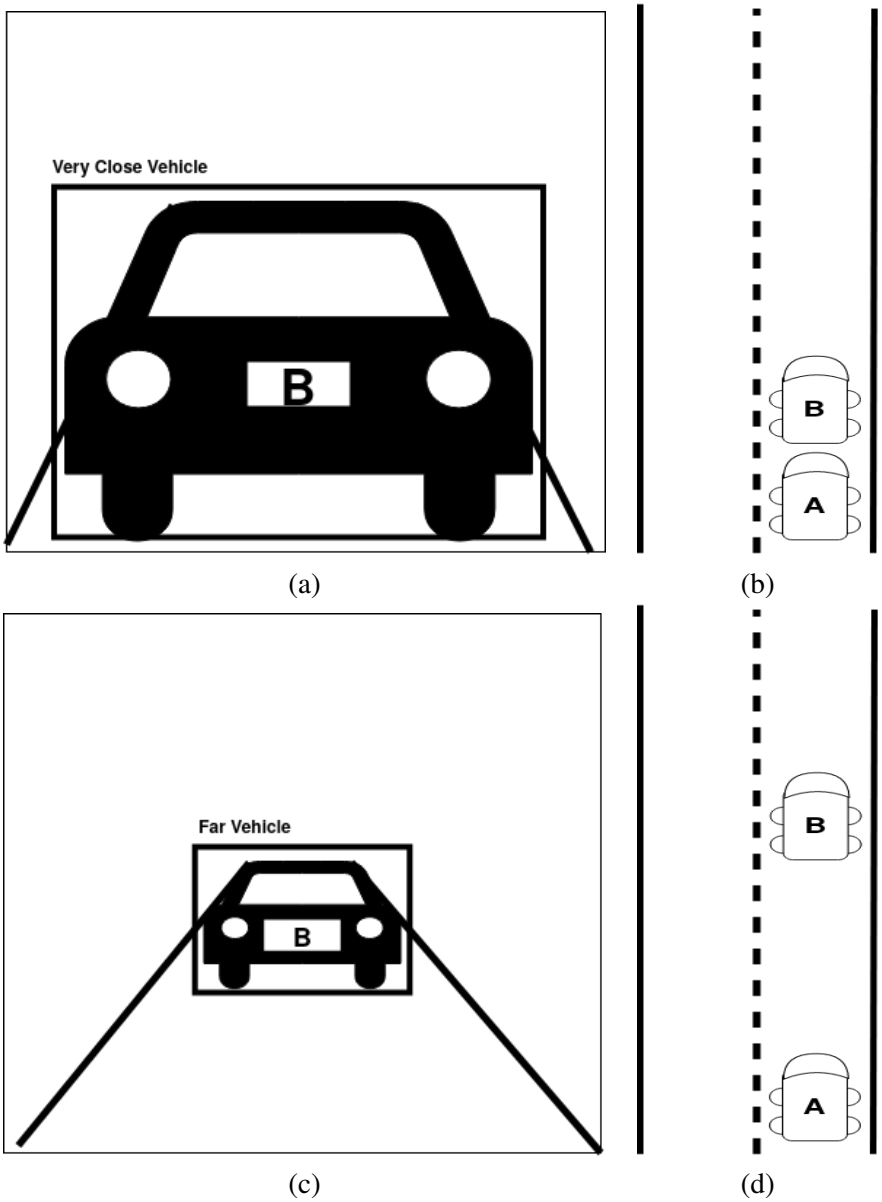


Figure 3.5: How bounding box sizes affects the distance. *(a) Very close vehicle viewed from in front perspective. (b) The same vehicle from (a) viewed from above. (c) Far vehicle viewed from in front perspective. (d) The same vehicle from (c) is viewed from above. We see how for a specific detected category (average size vehicle), we can train an estimator based on various bounding boxes to estimate the distance. To use only one estimator to predict the distance for all object categories, we need to use as input feature the predicted category, besides the bounding box.*

Based on this idea, we shift the model from an ”End to End Network” to a ”Multi-Network” approach, where the CNN only detects the objects in the image and based on a second tabular neural network that takes as inputs the outputs of the previous network, it predicts the estimated distance from the camera to the object.

This concept allows us to remove the task of depth estimation from the convolutional neural network, narrowing down the number of classes from 27 to only 7: Average Vehicles, Large Vehicles, Pedestrians, Motorcycles, Crosswalks, Continuous Lanes, and Dashed Lanes. Based on this the CNN focuses only on detecting those 7 objects, increasing the number of samples for each class, leading to probably better detections, especially for the classes less frequent in the dataset.

On the side of estimating distances, the multi-network approach allows us to use the second network as a regression head maintaining the continuous relation between the distance classes, thresholding bins for the discrete intervals, and optimizing for the minimum the error between the estimated distance class and the annotated one, instead of predicting the exact annotated distance. This can help to reduce the risk of learning from possible wrong annotations.

The illustrations for both of those concepts can be seen in the Figure 3.6:

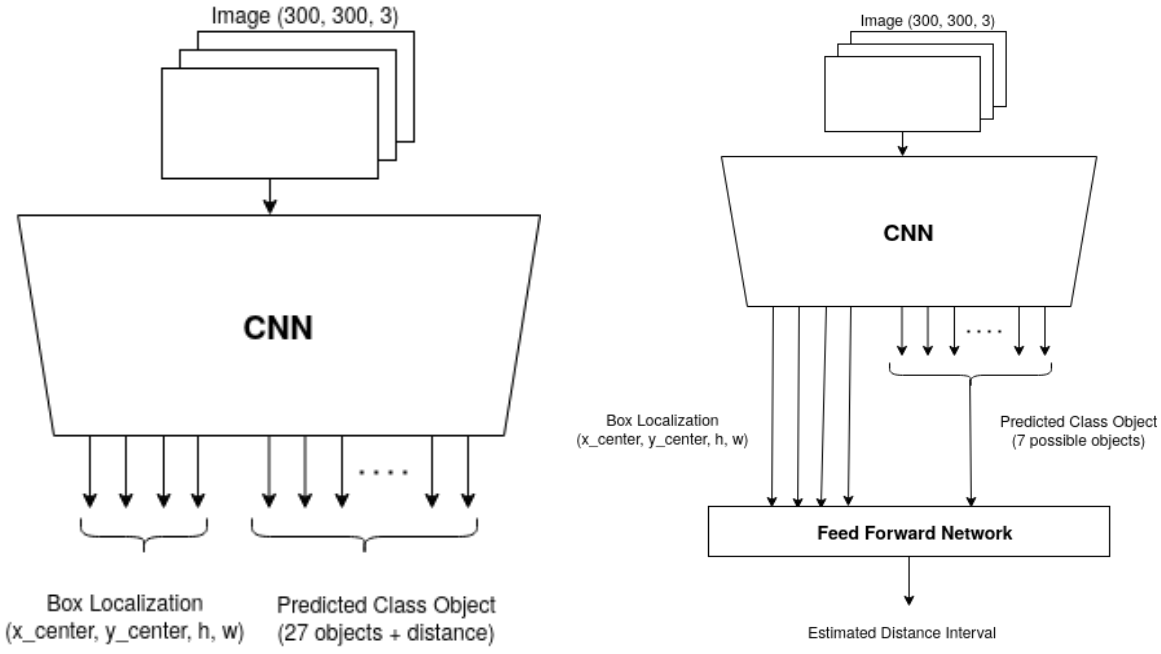


Figure 3.6: End To End Network approach vs Multi-Network approach. *(a) End to end architecture, 5 of 7 classes have attached the estimated distance (ex. very close **pedestrian**) resulting in 25 classes of detection + continuous lane and dashed lane, resulting in 27 detection classes. (b) Multi-Network approach, the CNN predicts only the initial 7 classes, (ex. **pedestrian**), the distance is predicted by a stacked neural network using the bounding box and the detection class as inputs.*

To be able to adopt this idea in the production environment, we quantify our intuition in statistical metrics extracted from a test dataset stratified by the object classes and the depth intervals, analyzing both the detection results and the distance estimation error.

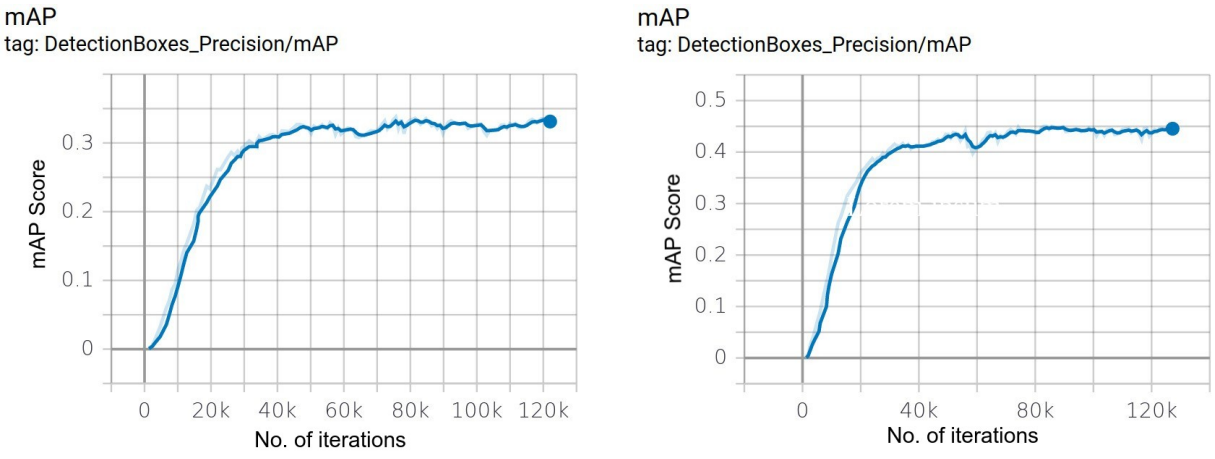
Object Detection Analysis: Using the same splits for training and validation, we trained a Single Shot Detector with a backbone a MobileNetV2 for approximately 120,000 iterations, both models were already pre-trained on the COCO dataset [17]. Comparing the general mAP averaged over 10 IoU thresholds of $.50 : .05 : .95$, mAP@.50IOU, and mAP@.75IOU for the best checkpoint selected by the general mAP we obtained the following results on the validation set displayed in Table 3.1.

Table 3.1: Validation Results over different IOU Thresholds

Method	Iteration Step	mAP	mAP@.50IOU	mAP@.75IOU
Multi-Network	83,880	**45.0%**	**69.72%**	**46.33%**
End to End Network	77,250	33.4%	51.08%	34.2%

As we can observe the validation results were much better for the Multi-Network architecture, those results were expected because the object detector is only detecting the object class, instead of at the End to End architecture where every mislabelled distance category is considered as a misclassified prediction.

The validation evolution for both methods is displayed in Figure 3.7.



(a) (b)

Figure 3.7: Compared validation mAP between End To End and Multi-Network. *(a) Validation mAP for End To End architecture. (b) Validation mAP for Multi-Network architecture.*

Based on this reason, on the test set, we don't use the mean average precision as an evaluation metric, remaining for us to define custom metrics and indicators for evaluating both methods.

Using those two object detectors we analyzed over the test dataset three possible architectures:

- End To End Network
- Multi-Network with classification head
- Multi-Network with regression head

For the error analysis stage, we use as evaluation indicators the following:

- **TP**: Number of predicted boxes that overlap with the annotated boxes for a 0.5IOU threshold
- **FP**: Number of predicted boxes that were not annotated in the image
- **FN**: Number of annotated boxes that were not predicted by the model
- **Missed Category (MC)**: Number of predictions where the object class was predicted wrong
- **Missed Distances (MD)**: Number of wrong predicted distances from the annotated one
- **Relative Mistakes (RM)**: Number of wrong predicted distances by maximum one interval

And based on the extracted indicators we compute the following evaluation metrics:

1. **Precision**: $Precision = \frac{TP}{TP+FP}$
2. **Recall**: $Recall = \frac{TP}{TP+FN}$
3. **Percentage of Missed Category (PMC)**: $PMC = \frac{MC}{TP}$
4. **Percentage of Missed Distances (PMD)**: $PMD = \frac{MD}{TP}$
5. **Percentage of Relative Mistakes (PRM)**: $PRM = \frac{RM}{TP}$
6. **MAE Distance Loss**:

$$MAE_{\text{distance}} = \frac{1}{TP} \sum_{\substack{t=0 \\ \text{lou}\left(\hat{b}_t, b_t\right) > 0.5}}^{\#\text{samples}} \left| \hat{d}_t - d_t \right|$$

   where $\hat{b}_t, b_t$ represent the predicted bounding box and ground-truth bounding box and $\hat{d}_t, d_t$ represent the predicted distance and ground-truth distance

7. **MSE Distance Loss**:

$$MSE_{\text{distance}} = \frac{1}{TP} \sum_{\substack{t=0 \\ \text{lou}\left(\hat{b}_t, b_t\right) > 0.5}}^{\#\text{samples}} \left(\hat{d}_t - d_t \right)^2$$

   where $\hat{b}_t, b_t$ represent the predicted bounding box and ground-truth bounding box and $\hat{d}_t, d_t$ represent the predicted distance and ground-truth distance

With those indicators and metrics described, we go further and evaluate our approaches. The test dataset was composed of 4,133 images stratified by the object classes. The number of annotated objects in the dataset is 14,682. For the test evaluation, we choose to display only the metrics. Based on the test set we obtained the following results:

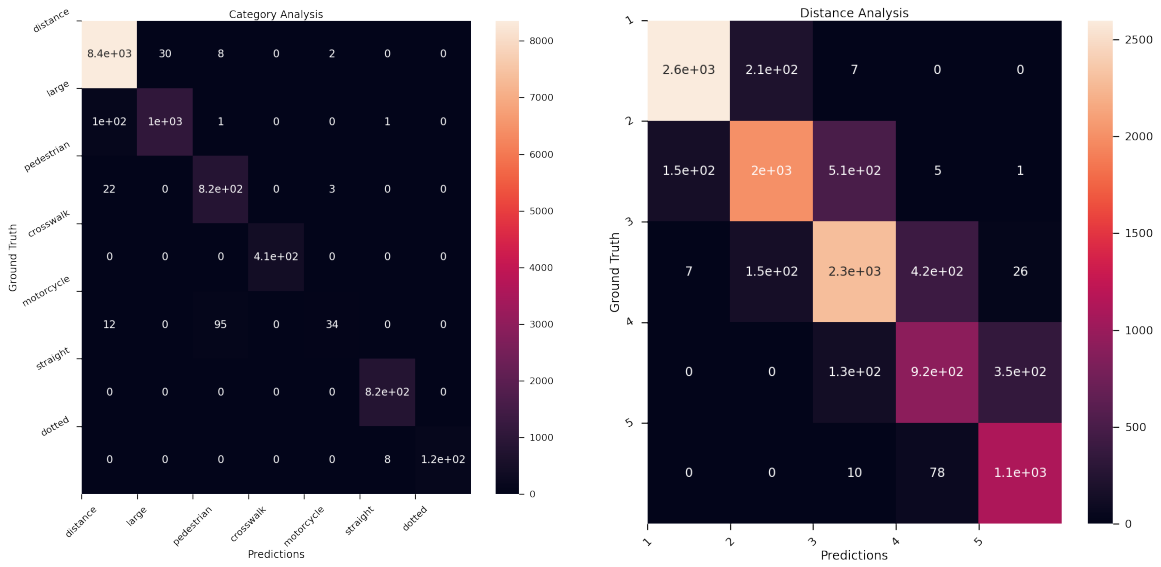
Table 3.2: Evaluation metrics on the test dataset

Method	Precision	Recall	PMC	PMD	PRM	MAE	MSE
End to End Network	**92.7%**	81%	2.4%	17.1%	97.2%	**17.5**	**18.5**
Multi-Network (Classification Head)	90.4%	**85.2%**	2.3%	24.1%	95%	25.3	27.9
Multi-Network (Regression Head)	90.4%	**85.2%**	2.3%	21.9%	**97.4%**	22.4	23.6

As expected, using only the coordinates of the bounding box and its predicted class for estimating the distance interval generates slightly worst results than using the whole context of the detection from the End To End architecture.

On the bright side of the situation, based on the 4.2% increase in recall score, our intuition was right. By reducing the number of classes for the object detector, the CNN focuses only on detecting the objects and with more samples for each class can better recognize objects in more diverse contexts, leading to an improved probability of generalization for our problem.

To better understand the prediction distribution and where our models predictions are wrong for each approach, we present the confusion matrices for predicted objects and predicted distances. Those illustrations can be seen in the following figures.

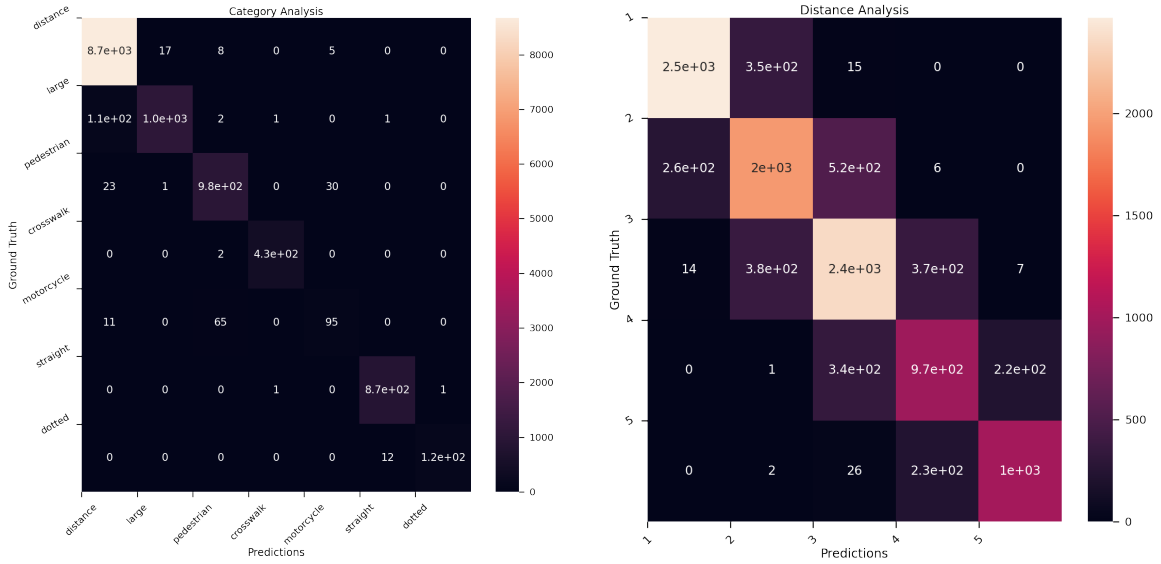


(a) (b)

Figure 3.8: Error analysis based on confusion matrix for End To End approach. *(a) Confusion matrix for predicted categories (e.g. large vehicles (large), pedestrians). (b) Confusion matrix for predicted distance intervals.*

Those experiments reiterate the idea that End to End Deep Learning methods are better when you have enough data for each class to sustain the complexity of the model.

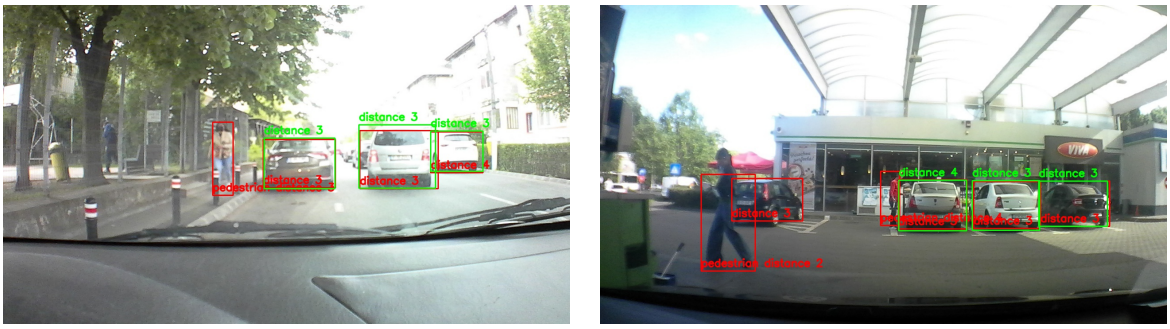
In our context, even with the increase in the error of the distance estimation model, we prefer using the Multi-Network approach with a regression head. Based on hyper-parameter tunning, we can improve the distance estimation error while still maintaining a better object detector for recognizing objects.



(a) (b)

Figure 3.9: Error analysis based on confusion matrix for Multi-Network approach (Regression Head). *(a) Confusion matrix for predicted categories (e.g. large vehicles (large), pedestrians). (b) Confusion matrix for predicted distance intervals.*

For a better visual representation of both architectures, we present a subset of images with the objects and distances predicted by both methods. In the following images, the red detections are provided by the Multi-Network architecture and the green detections are provided by the End to End architecture.



(a) (b)

Figure 3.10: End-to-End vs Multi-network, visual comparison. *Red: Multi-Network architecture, Green: End-to-End architecture. We use the images (a) and (b) to present the improvements made by switching from the End-to-End approach to Stacked Multi-Networks, especially for detecting smaller objects like pedestrians.*

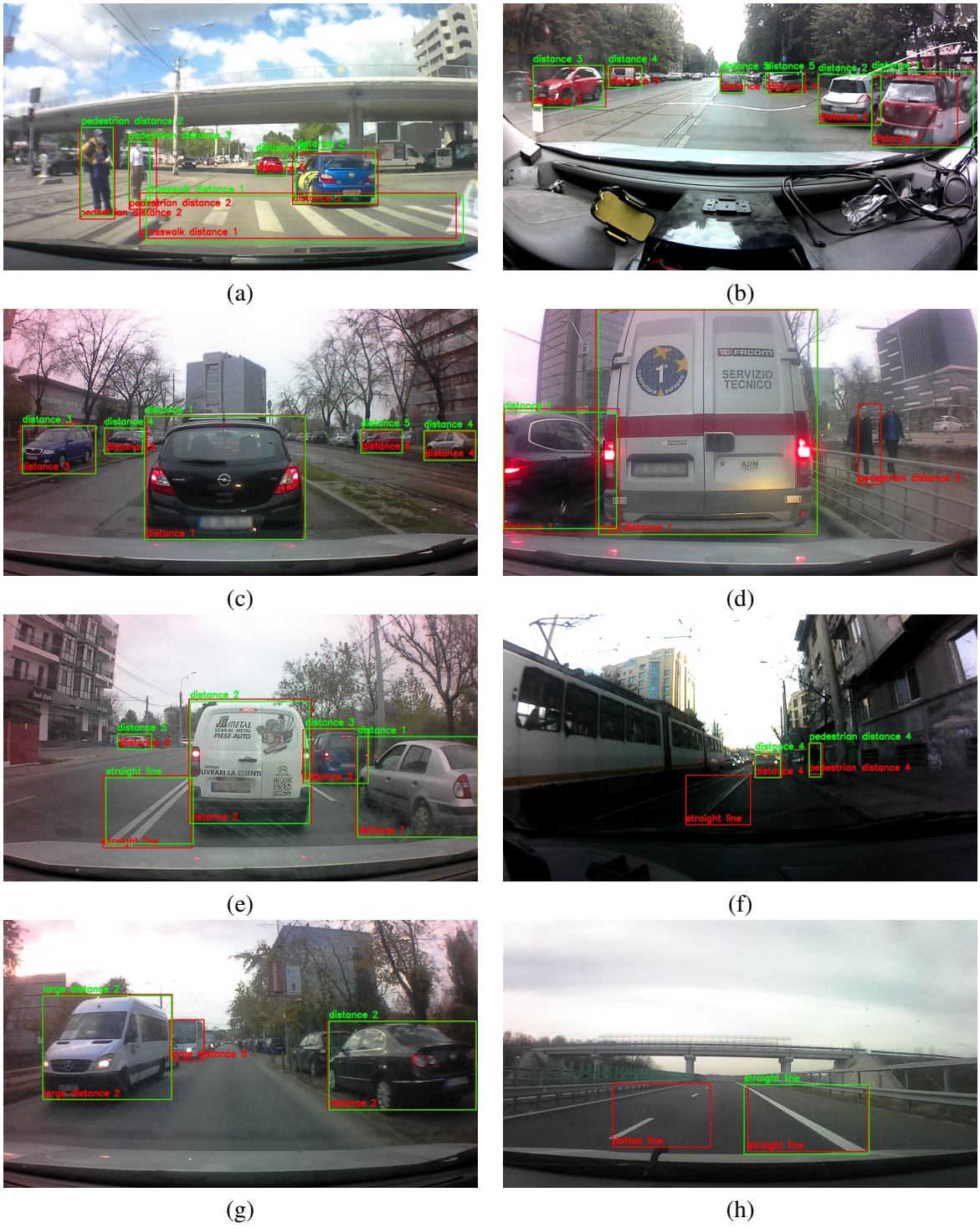


Figure 3.11: End-to-End vs Multi-network, visual comparison. *Red: Multi-Network architecture, Green: End-to-End architecture. As we can observe from the previous images, the Multi-Network approach improves significantly the detections proposed by the system.*

Choosing the Multi-Network approach instead of an End to End architecture, we move forward to the next problem, predicting the objects in front of the vehicle invariant to the camera position inside the cabin.

3.3 Camera horizontal calibration

At the current stage, we have a complete pipeline for detecting objects and estimating the distances between them and the camera.

The next problem we need to solve is detecting the objects that are in front of the vehicle, very important here to underline is **in front of the vehicle** and not necessarily in front of the camera.

In an ideal context, the camera would be always placed/integrated on the middle of the windshield, and after that based on some geometric conditions, we could establish which objects are in front and which of them are not in front.

Figure 3.12 presents this type of environment, detections only from the Safe Distance Warning.

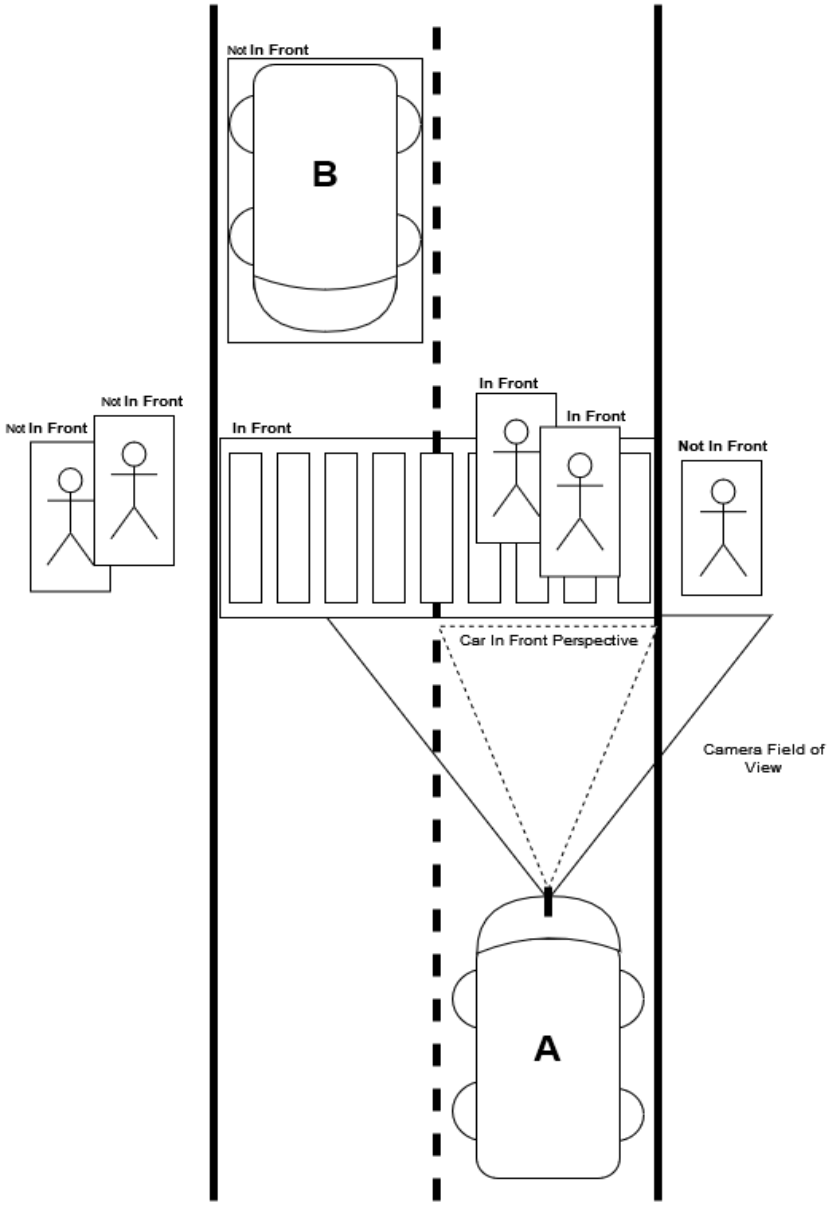


Figure 3.12: Camera field of view, center positioning.

In different circumstances, those two perspectives might differ because the field of view of the camera it's not always the same as the field of view of the vehicle.

Those two can be highly influenced by the positioning of the camera on the windshield.

For a better understanding, we present in the following Figures 3.13 (a) and 3.13 (b) the cases when based camera positioning the perspective from the vehicle it's different from the field of view of the camera.

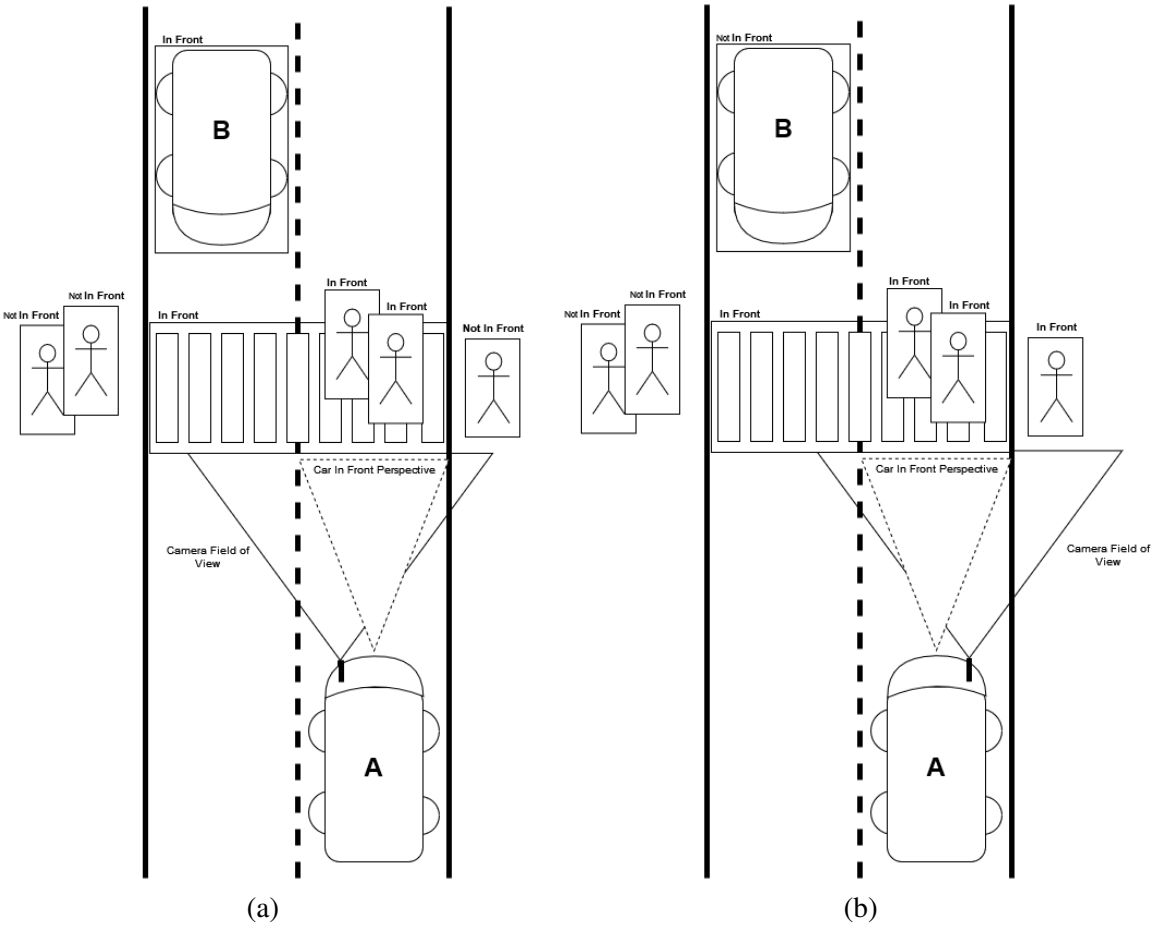


Figure 3.13: Camera field of view based on different positions. *(a) Camera field of view, left positioning. (b) Camera field of view, right positioning.*

Taking into consideration the fact that for various reasons the camera might not be placed on the middle of the windshield this problem becomes very difficult to tackle.

From the previous experiences with stacked networks, we decide to approach this problem based on Machine Learning. For this problem, we use as possible input features the outputs of the previous networks. This way we keep the complexity of the model as small as possible, not affecting the inference time.

As potential features for this neural network, we can use the detection coordinates, the aspect ratio

and the area of the bounding box, the class predicted by the object detector, and the estimated distance. Those features by themselves won't be sufficient to solve the problem because we have no indication about the position of the camera in the cabin.

The solution here came from the problem itself, considering the positioning of the camera on the windshield as a varying factor that can affect the detection of the in-front objects, we decide to use as an input feature, the camera deviation from the middle of the windshield.

Further on, we name the deviation of the camera placement from the center of the cabin where is installed the **calibration offset**.

As we said in the beginning, we are trying to develop a driver monitoring system based on two features, Safe Distance Warning (SDW) and Lane Departure Warning (LDW), using only Deep Learning methods and Computer Vision.

The context presented above described the problem of horizontal calibration only for the SDW system.

For the LDW system, the requirement of detecting the in-front lanes remained but some adjustments are necessary to be made.

In Figures 3.12, 3.13 (a), and 3.13 (b), we describe how the positioning of the camera can affect the detection of in-front objects using a perspective from above, a 2D perspective.

For a more representative perspective of the problem of camera calibration in the LDW context, we use a 3D perspective of the road. To be able to differentiate between those two examples we use Figure 3.14, presented below:

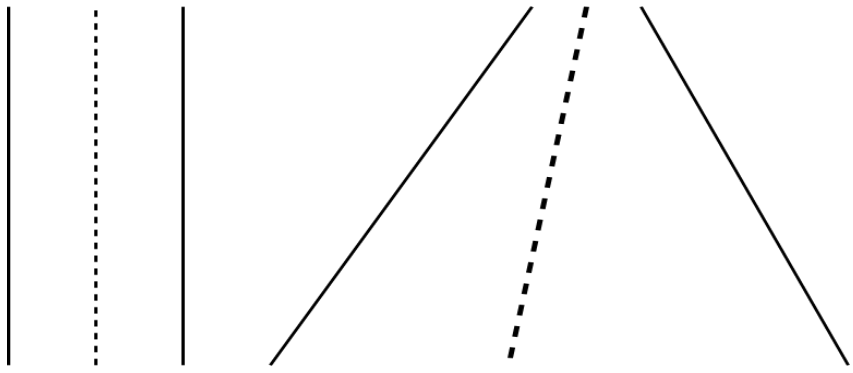


Figure 3.14: 2D perspective vs 3D perspective of the road.

Based on the second image in Figure 3.14, we'll try to reproduce a 3D perspective for the camera horizontal calibration in the context Lane Departure Warning system.

After presenting the new perspective, we can describe the context of triggering the system of lane departure warning and how can it be influenced by the positioning of the camera on the windshield.

In Figure 3.16, we present the cases when the system should trigger a warning for passing the lane.

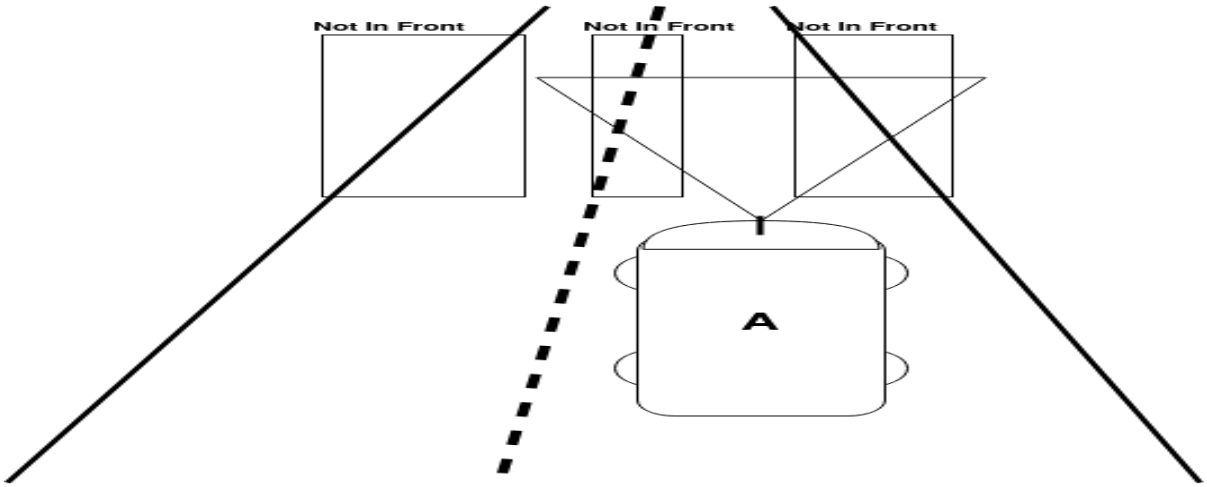


Figure 3.15: LDW correct non-trigger case.

Based on the diagram from the Figure 3.15, we observe why the neural network that is used for predicting in-front objects for the Safe Distance Warning system can't be the same as the one predicting if a lane is in front or not.

Assuming that the right lane detected in the image would be an object class from the SDW system like a vehicle or pedestrian, that object should be considered as in front because at high speeds a collision would be possible, but in the context of Lane Departure Warning system, this type of detection box should describe an object as being not in front.

Based on this argument, we conclude that the problem of camera calibration should be treated differently between the contexts of the Safe Distance Warning system and the Lane Departure Warning system.

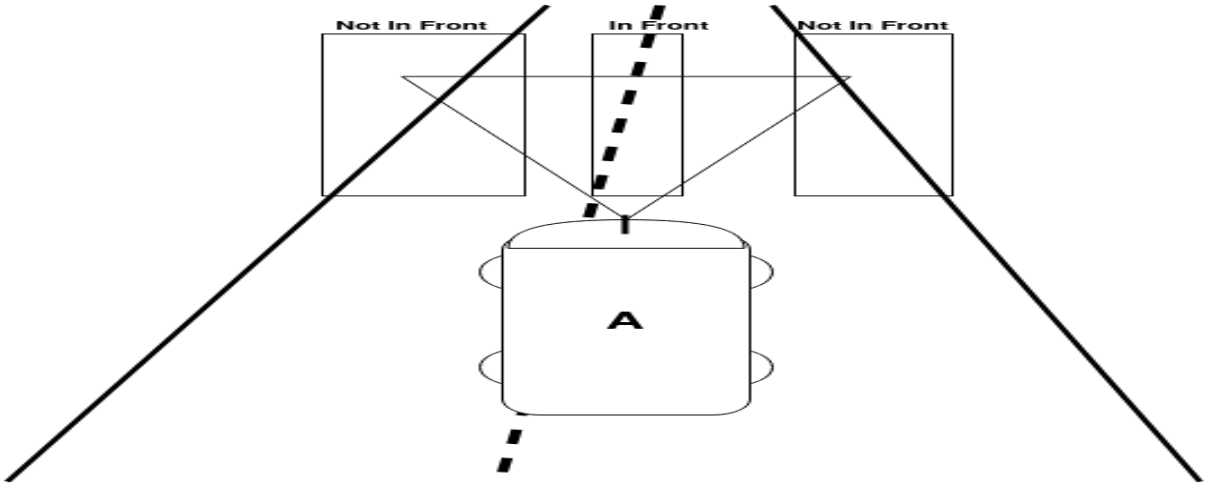


Figure 3.16: LDW correct trigger case.

Before wrapping up this section, we present why the problem of camera horizontal calibration is also present in the LDW context and how the positioning of the camera can affect detection in front lanes. We present this scenario based on the Figure 3.17.

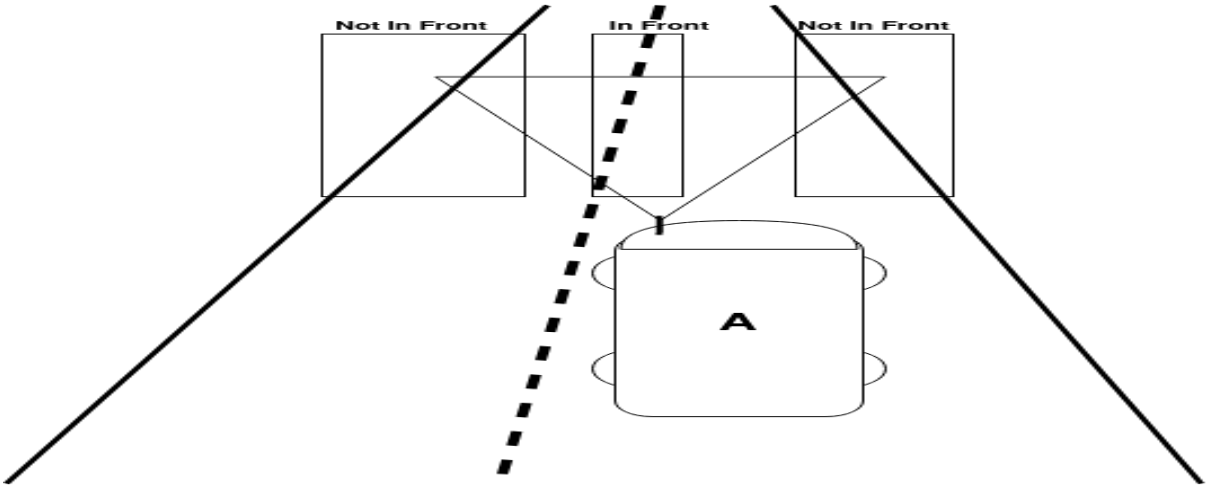


Figure 3.17: False positive trigger example from the LDW system.

Based on the Figure 3.17, we observe how detecting when the vehicle is crossing the lane becomes difficult in relation to the position of the camera on the windshield.

With all those in mind, we present the current diagram of our solution for a driver monitoring system in the outside world context. This diagram is represented in Figure 3.18.

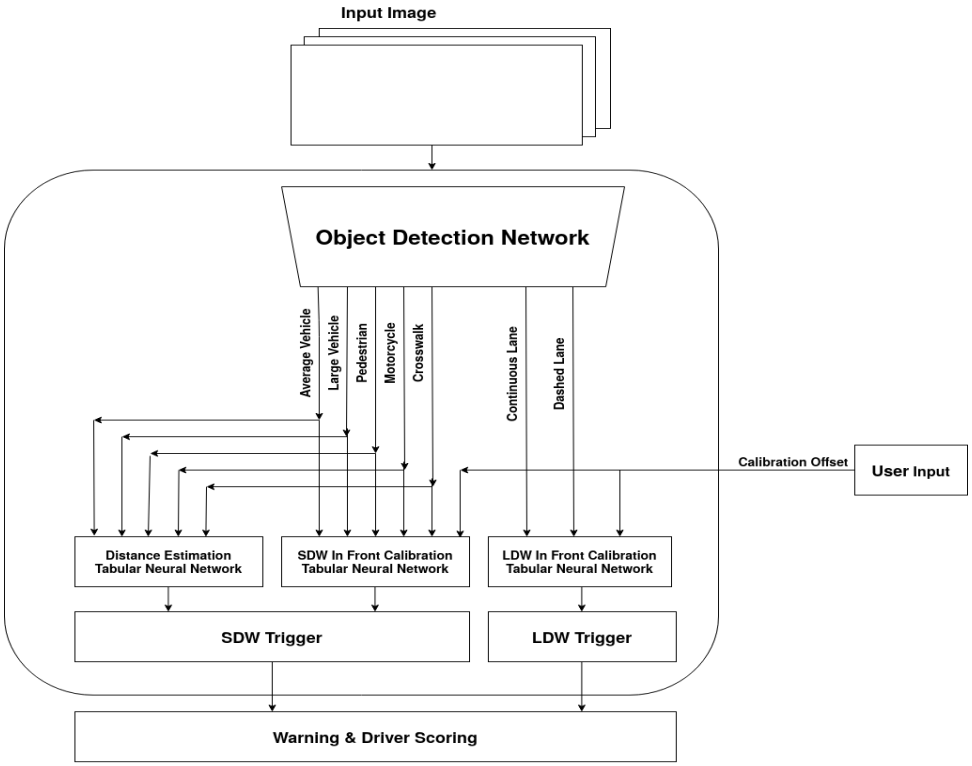


Figure 3.18: Driver Monitoring System incomplete solution.

3.4 Lanes interpolation based on features masks

The last piece of our driver monitoring system is the interpolation algorithm for object detection predictions. Taking into account the fact that the object detection approach does not persevere the identity of the object, we risk losing detections when using the convolutional neural network frame-by-frame.

In most situations, this does not represent a problem because increasing the dataset size with images from various contexts and optimizing thresholds should reduce significantly the frequency of this problem. For the Lane Departure Warning system, this scenario of losing detections could generate problems. At high speeds, the outside context becomes very dynamic, and failing to detect lanes in those moments could lead to our device losing dangerous events.

We present this type of scenario in Figure 3.19:

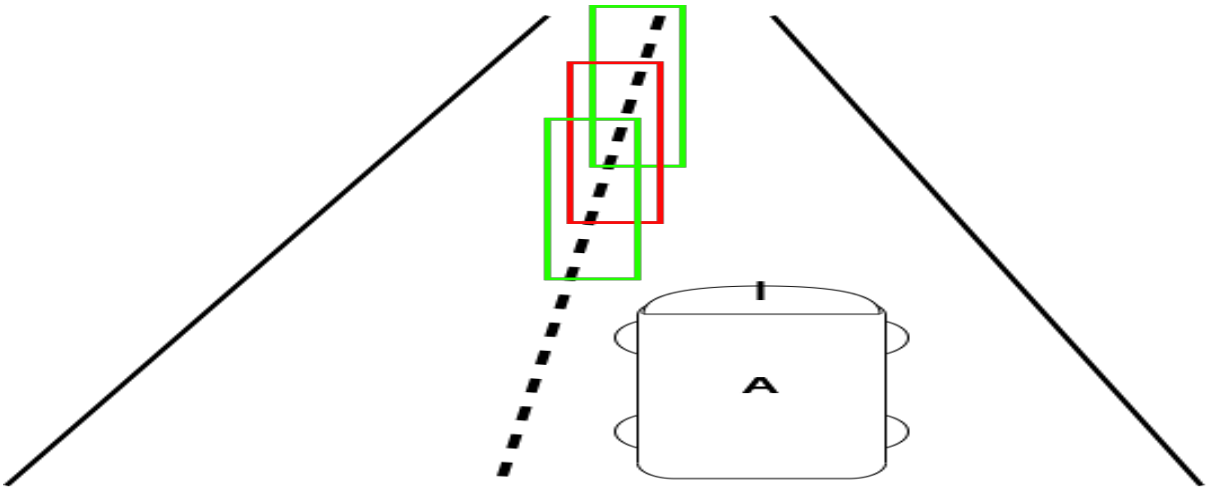


Figure 3.19: Lane Departure Warning missed detection scenario.

As a fail-safe mechanism, we design an interpolation algorithm for detections based on the previous bounding box coordinates in case of the convolutional neural network fails to detect the object. The idea for this algorithm is based on our assumption that the object of interest, the white continuous lane, and its coordinates should not differ ”much” in consecutive frames at various speeds.

For a better understanding, we change the perspective from the Figure 3.19, spreading our detections on a timeline axis:

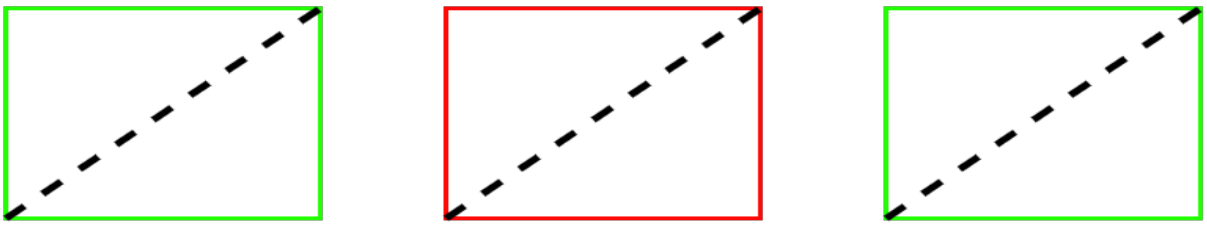


Figure 3.20: Our assumption about consecutive detections from Figure 3.19

Based on our intuition, we examine computer vision techniques for generating a feature mask based on accumulated bounding boxes from consecutive frames. We infer using that feature mask at the last known coordinates in the current frame, where the lane it's not detected.

Considering the following example: We have k-consecutive frames where the continuous lane is detected, we denote this as timestamps from $T_0, T_1, \ldots T_k$ each of them representing a lane detection from a distinct frame. At the frame T_{k+1} the lane is not detected. This could lead to two possible outcomes, either the convolutional neural network failed to detect the lane for various reasons (e.g. a too strict threshold value) and the detection could return in the following frames, or the lane it's no longer present in the image or it's covered by another object. Those scenarios can be seen in Figures 3.21 and 3.22.

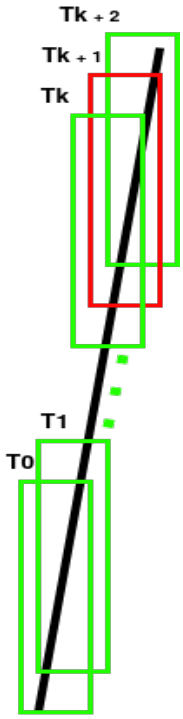


Figure 3.21: Missed lane detection by CNN.

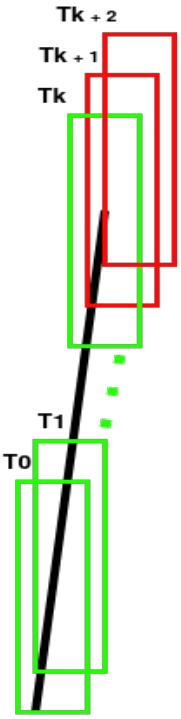


Figure 3.22: Lane vanishing from the image.

The first case in the images presented above, Figure 3.21, represents a false negative example, and based on the interpolation algorithm we reduce those fail-points from our system. We want to use the knowledge predicted by the convolution neural network in previous detections from $T_0, T_1, \ldots T_k$ frames to infer the probability of failed detection in the frame T_{k+1}.

For our algorithm, we introduce a combining function for generating a feature mask over the previous detections in the last K consecutive frames, where $FeatureMask = Combine(T_0, T_1, \ldots, T_k)$. For a better representation of the problem, we don't combine the images in the raw format, we apply computer vision morphological transforms before the combine function, so the formula above becomes

$FeatureMask = Combine(Transform(T_0), Transform(T_1), \ldots, Transform(T_k))$.

We define a similarity function between the generated feature mask and T_{k+1}, where T_{k+1} is the region from the frame $k+1$ at the last know detection coordinates, those from frame k.

We have $S = similarityFunction(FeatureMask, T_{k+1})$ and based on S we decide T_{k+1} should be interpolated and considered as regular detection for the LDW system.

In the second case, Figure 3.22, we underline the need for a strong stopping condition from the algorithm for avoiding possible false positives.

We introduce an algorithm based on two steps: **Accumulation Step** and **Inference Step**

3.4.1 Accumulation Step

At this stage, we have a frame where at least one lane is detected. Based on that detection and the current state of the feature mask we can update or initialize the mask.

A significant detail we need to underline is that by using object detection we won't have only one lane detected in the image. Based on the threshold value used for the continuous lane class, we can have multiple lanes detected, some of them being false positives that can pollute the feature mask.

For that reason, before initializing the feature mask we need to use only the detection with the highest score and before adding to the feature mask we need to ensure an IOU coefficient above a certain threshold between the feature mask at the previous coordinates and the current detection coordinates.

After adding a new detection, we update the bounding box coordinates of the feature mask.

Below, in the Algorithm 1, we present a pseudo-code for a general approach to the accumulation step.

Algorithm 1: Accumulation Step

```
def AccumulationStep(current_detection, threshold):
    if FeatureMask.active == False:
        /* intiatize the Feature Mask                                  */
        FeatureMask = Transform(current_detection)
        FeatureMask.active = True
    else:
        /* add the current detection to the Feature Mask based on IOU  */
        if IOU(FeatureMask.box, current_detection.box) > threshold:
            FeatureMask = Combine(FeatureMask, Transform(currrent_detection))
            /* update last coordinates                                 */
            FeatureMask.box = current_detection.box
        else:
            /* deactivate the mask                                     */
            FeatureMask.empty()
            FeatureMask.active = False
    return FeatureMask
```

3.4.2 Feature Mask Inference Step

At the inference stage, we check if the mask is active and if we have accumulated a minimum number of consecutive frames.

Based on the coordinates of the feature mask, and the last known coordinates from the convolutional neural network, we select the region of interest from the current frame and we check the similarity between the feature mask and that region.

If the similarity condition between those two is satisfied, we consider the region of interest from the current frame as a regular detection from the convolutional neural network.

The algorithm 2, presents the pseudo-code for the inference step.

Algorithm 2: Inference Step

```
def InferenceStep(FeatureMask, current_frame):
    if FeatureMask.active == False:
        return False
    assumed_detection = current_frame [FeatureMask.box]
    S = similarityFunction(FeatureMask, Transform(assumed_detection))
    if similarityConndition(S) == True:
        /* assumed detection is considered as a regular detection */
        return True
    else:
        /* deactivate the mask */
        FeatureMask.empty()
        FeatureMask.active = False
        return False
```

3.4.3 Interpolation Pipeline for Lane Departure Warning

Taking into consideration the fact that we expect a left lane and a right lane in each frame we consider two feature masks.

As said before, for initialization we need to select the lane detected with the highest score from the convolutional network. Now, we add the condition to find the highest score left detection and the highest score right detection.

We define those conditions for selecting the left and right lanes from the CNN detections with the auxiliary function defined below, algorithm 3:

Algorithm 3: getCurrentDetection

```
def getCurrentDetections(current_frame):
    /* Selecting the left and right detections with the highest scores       */
    left_score, right_score = 0, 0
    left_detection, right_detections = None, None
    for detection in detections:
        if detection.x_center ≤ image_width/2 and detection.score > left_score:
            left_detection = detection
            left_score = detection.score
        if detection.x_center > image_width/2 and detection.score > right_score:
            right_detection = detection
            right_score = detection.score
    return left_detection, right_detection
```

Based on the functions for selecting the current left and right lane, accumulation, and inference, we can define a generic pseudo-code for the interpolation pipeline, the algorithm 4:

Algorithm 4: Interpolation Algorithm

```
def InterpolationAlgorithm(current_frame):
    left_detection, right_detection = get CurrentDetections(current_frame)
    if left_detection is not None:
        left_mask = AccumulationStep(left_detection, threshold)
    else:
        interpolate_left = InferenceStep(left_mask,current_frame)
    if right_detection is not None:
        right_mask = AccumulationStep(right_detection, threshold)
    else:
        interpolate_right = InferenceStep(right_mask,current_frame)
```

3.4.4 Testing and Results

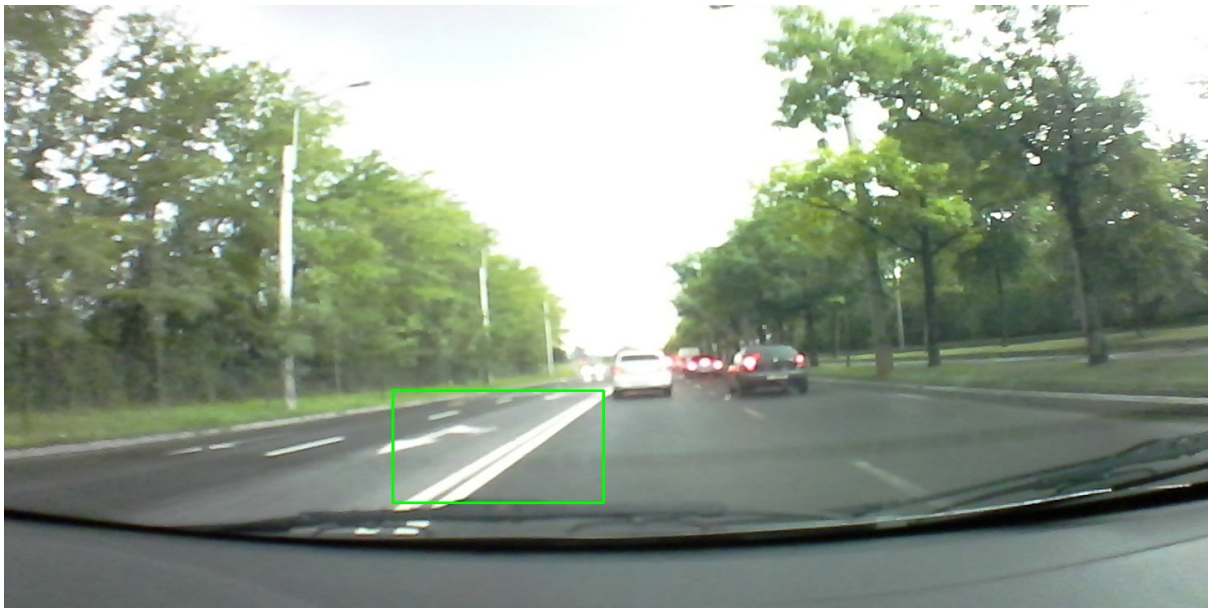
As a proof of concept, we generated 10 test cases in various scenarios for validating our approach over different $Transform$, $Combine$, $Similarity$ and $similarityCondition$ functions.

We present an example collected from the real environment for our algorithm using as:

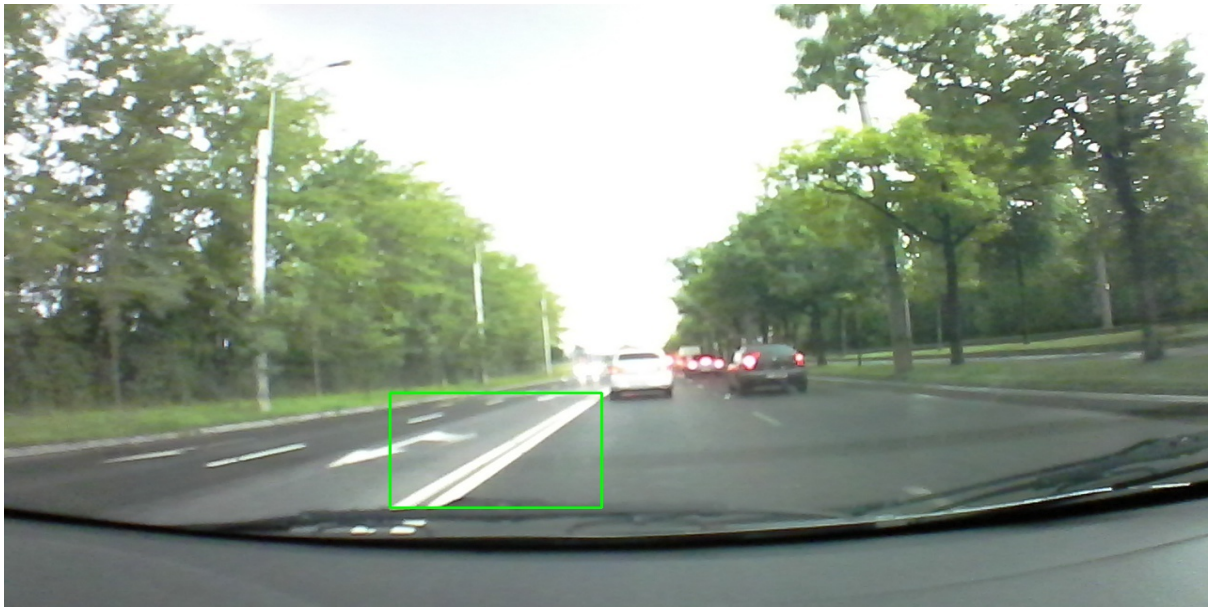
- $Transform(CurrentDetection)$: Median Blur, Erosion (for noise reduction)
- $Combine(FeatureMask, CurrentDetection)$: Weighted Blending with 70% importance for current frame
- $Similarity(FeatureMask, AssumedDetection)$: Adaptive Thresholding, Dilation, resulting S = the percentage of white pixels after BitwiseAnd
- $similarityCondition(S)$: S between lower limit and upper limit of general white percentage from the FeatureMask

With those details presented, we display as an example a couple of frames from a test scenario from the proof of concept stage, with a 50% of CNN missing the lane detected.

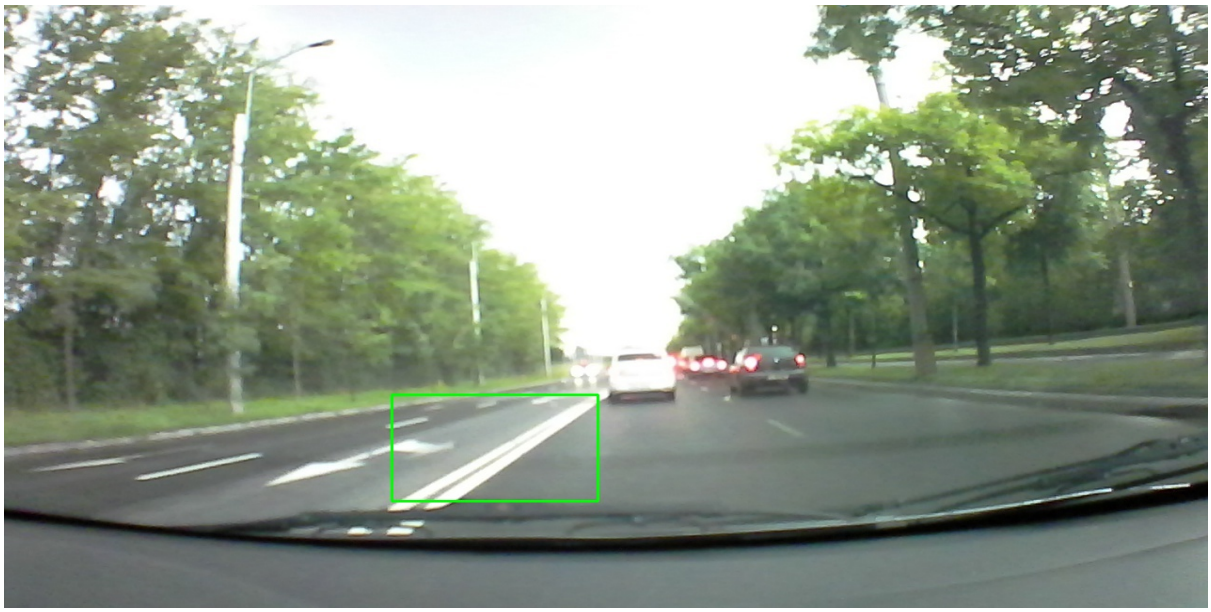
In the following example, the green boxes represent those from the CNN and those with red represent the interpolated detections.



CNN detected the lane in T_0



CNN detected the lane in T_1



CNN detected the lane in T_2

In the T_3 the CNN fails to detect the lane and the process of interpolation begins

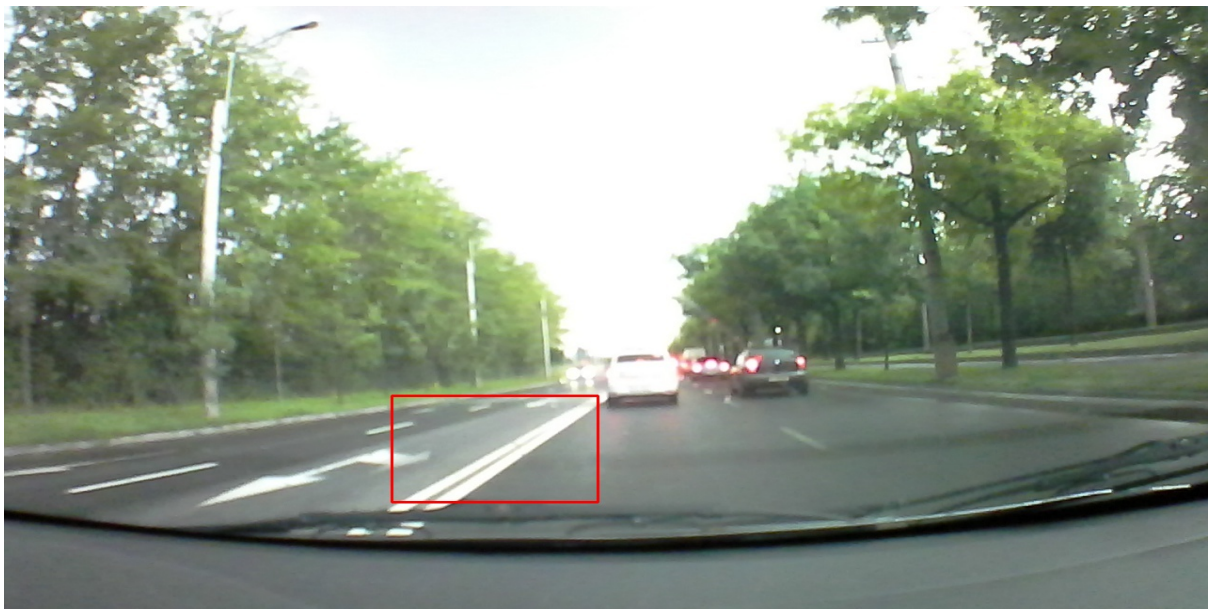
Feature Mask | Assumed ROI | Transformed Mask | Transformed ROI | Bitwise And Mask

The percentage of white pixels from the resulted bitwise and mask is 11.8%. (1)

The lower bound computed from the Transformed Feature Mask is 6.8%. (2)

The upper bound computed from the Transformed Feature Mask is 20.5%. (3)

From (1), (2), (3) $\Longrightarrow$ we consider the assumed ROI as a regular detection



The Interpolation Algorithm detected the lane in T_3

In the T_4 the CNN fails to detect the lane and the process of interpolation begins

Feature Mask | Assumed ROI | Transformed Mask | Transformed ROI | Bitwise And Mask

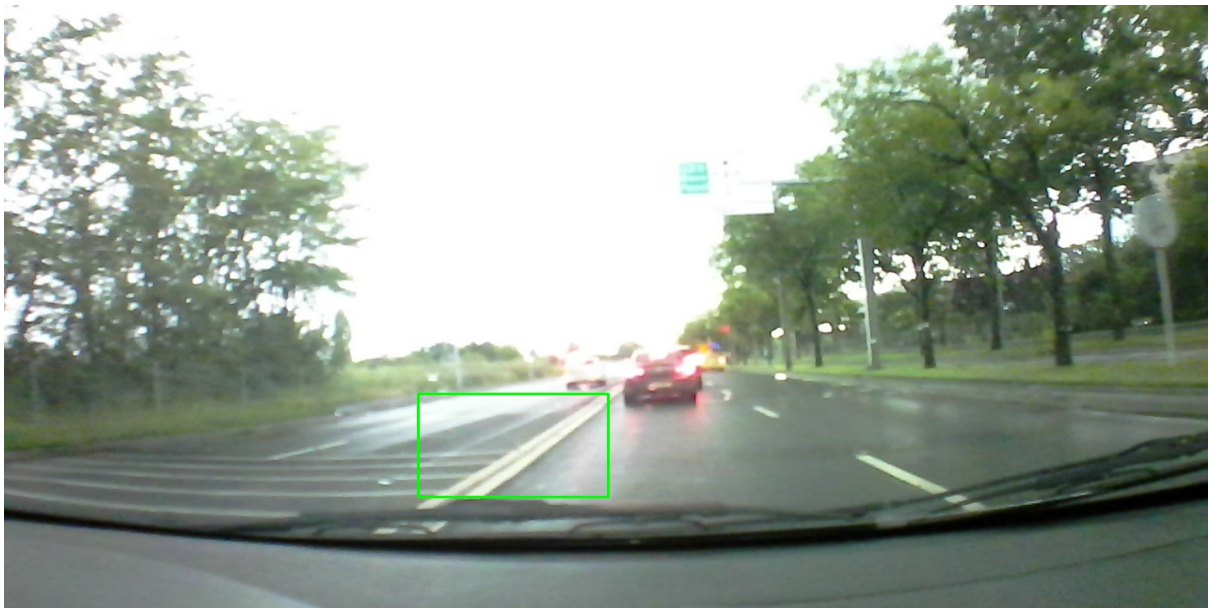
The percentage of white pixels from the resulted bitwise and mask is 11.4%. (1)

The lower bound computed from the Transformed Feature Mask is 6.7%. (2)

The upper bound computed from the Transformed Feature Mask is 20%. (3)

From (1), (2), (3) $\Longrightarrow$ we consider the assumed ROI as a regular detection

The Interpolation Algorithm detected the lane in T_4



CNN detected the lane in T_5

In the T_6 the CNN fails to detect the lane and the process of interpolation begins

Feature Mask | Assumed ROI | Transformed Mask | Transformed ROI | Bitwise And Mask

The percentage of white pixels from the resulted bitwise and mask is 0.9%. (1)

The lower bound computed from the Transformed Feature Mask is 6.8%. (2)

The upper bound computed from the Transformed Feature Mask is 20.4%. (3)

From (1), (2), (3) $\Longrightarrow$ **we don't consider the assumed ROI as detection and deactivate the mask**

The CNN and the Interpolation Algorithm correctly stopped detecting the lane in T_6

With this example presented, we display the final diagram of our solution, Figure 3.23.

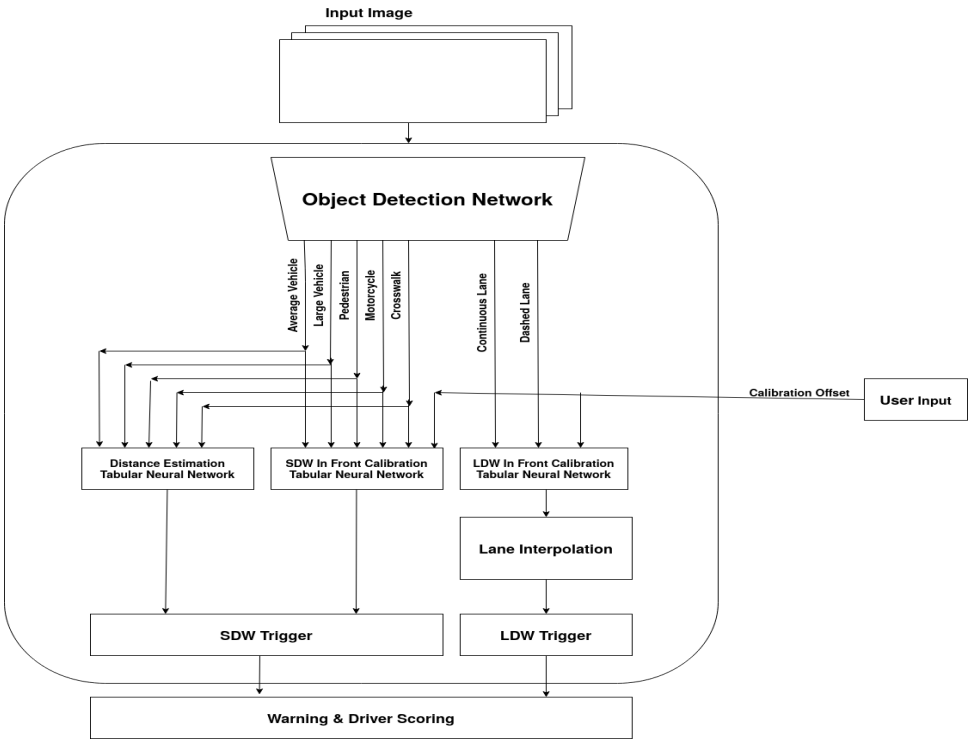


Figure 3.23: Final solution for our Driver Monitoring System.

3.5 Advanced Active Learning for Driver Assistance Systems

To generate our dataset we collected video samples from our devices based mainly on 6-8 Uber drivers and other clients from around the world.

In the first stages of annotations, we discovered that many video segments were redundant, like staying behind a vehicle at a red light, driving on highways, etc.

Training on redundant frames in dataset generated models that could not generalize in various contexts. This represented a major problem because even if our dataset is mainly collected from Bucharest, we have expectations for our system to generalize in foreign regions like Tanzania, Australia, and South Africa.

The initial solution came from training our models on video segments from our clients located in exotic places, resulting in major improvements. But from various technical factors and the extremely high amount of data from our clients, we generated a robust active learning pipeline for selective searching and data annotation.

Presenting this iterative process, we have the following steps:

1. Standard data collection for a small number of samples
2. Manual Annotation, based on multiple annotators
3. Training a Production Pipeline Network, almost identical to one in the field, as close as possible to real-life conditions
    - One Shot Detector
    - Single Shot Detector with MobileNetV3 for embedded devices, optimized for latency
    - Multiple Networks approach
4. Training a Proposal Network for predicting annotations
    - Two Shot Detector
    - FasterRCNN with Resnet-101 architecture optimized for mAP, no inference time constraints
    - Later to be replaced by EfficientDet-D7 [18]
    - Trained on single or multiple RTX2080Ti
5. Saving images from the Production Network with High and Low confidence over predictions
6. Error analysis over a subset of saved results from Production Pipeline Network.
7. Training Specialized One Shot Detectors only for certain contexts, issues, or classes
   (e.g. trained for a certain subset of data samples and output classes)

8. Deploying the Specialized One Shot Detectors for Selective Search
9. Sending to Annotation Proposal Network for automated labeling
    - Selective searched data
    - Frames containing predictions with low confidence
    - Random subset of frames containing predictions with high confidence
10. Manually verification of predicted labels

For a better representation of the process, we display the Figure 3.24:

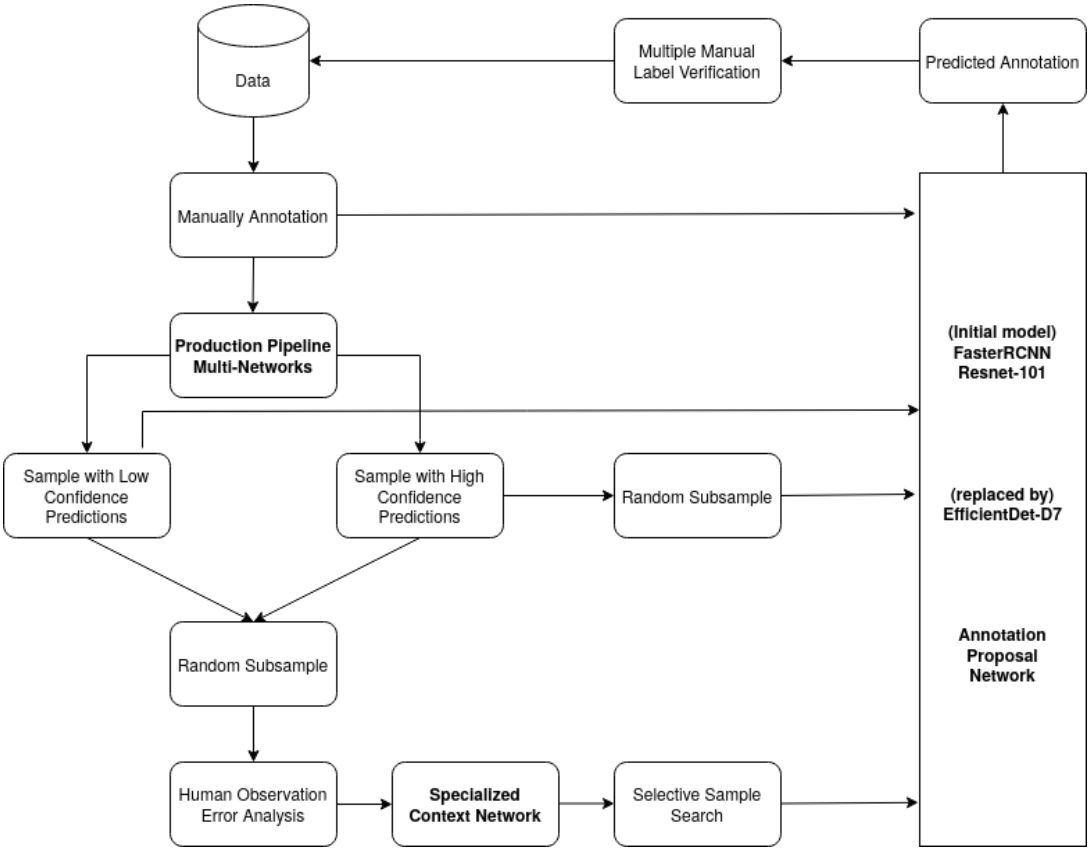


Figure 3.24: Active learning for Driver Assistance Systems.

Based on this design:

- we decrease the need for human intervention in the labeling process
- the overall labeling time is reduced by replacing manual annotation with only verification
- we add more diverse data points for model training

With those details about the dataset collection and annotation pipeline presented, we move forward in the next section presenting the models trained for the object detection stage in our system.

3.6 Detecting outside world objects

Moving towards the training stages, the principal component of the monitoring system is the object detection network. As specified before, this part of the system has the most constraints.

The main restrictions came from the hardware accelerator. The most important limitation is the one based on the inference speed of the system. Dealing with the outside context at high speeds everything tends to be very dynamic.

Taking into consideration the fact that the entire inference process is done on an embedded device instead of a regular GPU, we are required to use low latency deep learning models, like Single Shot Detector with MobileNet models, instead of using more effective models like EfficientDet [18].

Another major limitation comes from Qualcomm API, which allows us the interaction with the hardware accelerator. The main problem here was that a large number of layers used in various state-of-the-art architectures are introduced in the API with a significant delay or not introduced at all.

The layers that are not supported by the API cause the following part of the architecture to fall on the CPU, reducing the inference time so the practical use in the real world would be impossible. With equal chances, an unsupported layer could lead to compilation failure of the entire model.

For a better representation of the problem, we use the Figure 3.25 provided from Google Documentation for ”TensorFlow models on the Edge TPU”:

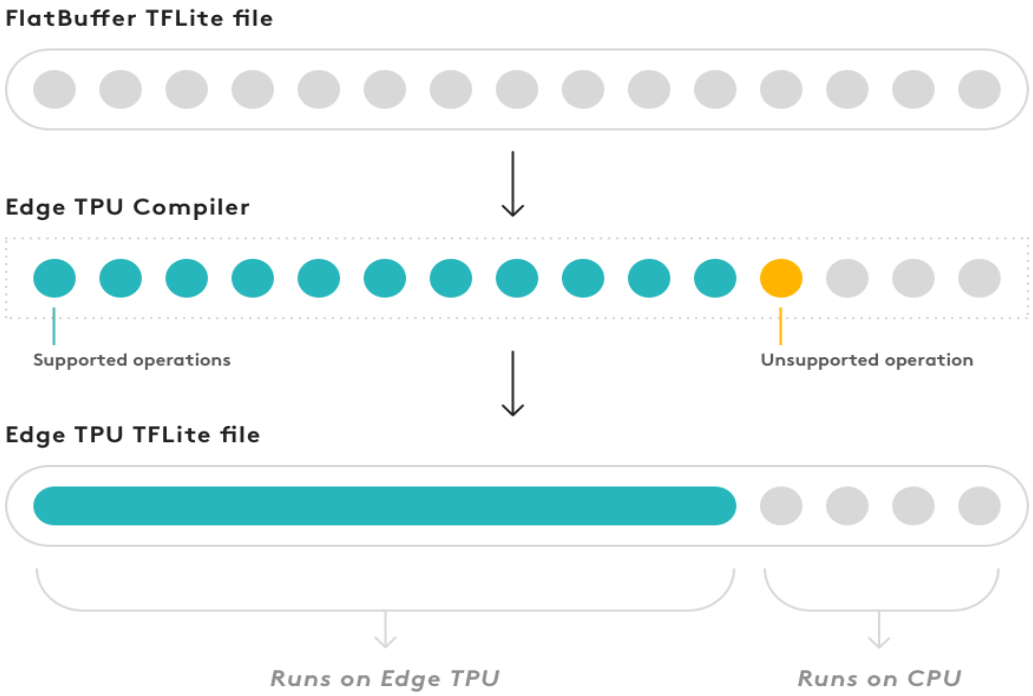


Figure 3.25: Unsupported layers on Edge TPU.

Even if during our project we don't use as a development device a Google Coral with Edge TPU, the behavior of the device, compared to our Qualcomm Snapdragon, tends to be similar. Based on this, we use Figure 3.25 for describing our problem with unsupported layers.

Based on those main limitations, we are forced to limit our possibilities when referring to deep neural networks and instead focus on finding ideas for improving our system.

Before presenting our experiments, another step is necessary for using any architecture on our device. That step is the quantization of model weights from floating-point to integer. As presented in the previous section, the quantization process results in latency reduction and model compression. This requirement is regular for most embedded devices when using deep learning architectures for inference.

In our case, we have two possible options:

- Quantization-Aware Training
- Post-Training Quantization

During this section, we present experiments with both quantization methods.

For training the object detection models we used the pipeline provided from TensorFlow, versions ranging between 1.11 and 1.15, using especially the Object Detection API from tensorflow/research repository.

As underlined previously, in section 3.2, we use the multi-network approach for our system. Resulting in a classification and localization problem over 7 classes: Average Vehicle, Large Vehicle, Pedestrians, Motorcycle, Crosswalks, Continuous Lane, and Dashed Lane. Those classes are presented in our dataset accordingly to the following distribution, Figure 3.26.

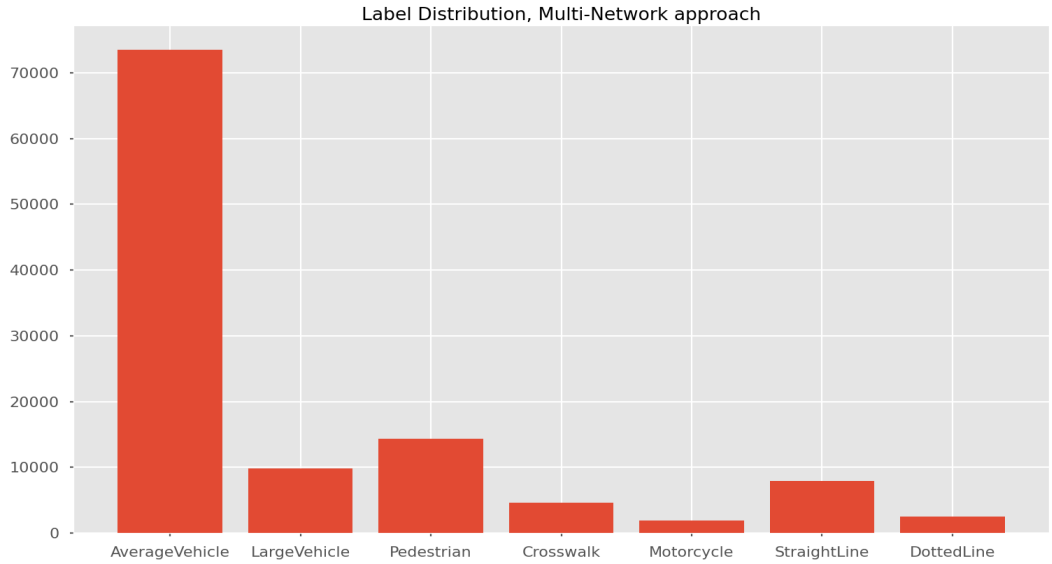


Figure 3.26: Label distribution, multi-network approach.

We experimented using the following architectures, but not all of them were available for compiling on the device for inference.

We mark with (✓) the ones that were successfully compiled for our device and with (✗) the rest.

- Single Shot Detector - MobileNet V1 (✓)
- Single Shot Detector - MobileNet V1 - Quantization Aware Training (✓)
- Single Shot Detector - MobileNet V1 - Feature Proposal Network (✗)
- Single Shot Detector - Resnet50 - Feature Proposal Network (✗)
- Single Shot Detector - MobileNet V2 (✓)
- Single Shot Detector - MobileNet V2 - Quantization Aware Training (✓)
- YOLOv3 (You Only Look Once) (✗)
- Single Shot Detector - MobileNet V3 Small (✓)
- Single Shot Detector - MobileNet V3 Large (✓)

Based on the models that were successfully compiled, we provide the following results in tabel 3.3:

Table 3.3: Results for successfully compiled models

Architecture	**Aware-Quantization**	**COCO mAP***	**Our Dataset mAP***	**Latency****
SSD MobileNet V1	False	21%	35.8%	5.8 FPS
SSD MobileNet V1	True	18%	37.2%	5.8 FPS
SSD MobileNet V2	False	22%	37.2%	8.2 FPS
SSD MobileNet V2	True	22%	**44.4%**	8.2 FPS
SSD MobileNet V3 Small	False	15.4%	27.2%	25 FPS
SSD MobileNet V3 Large	False	22.6%	32.5%	9.6 FPS

(*) The AP is averaged over 10 Intersection over Union (IoU) thresholds of $.50 : .05 : .95$.
(**) Latency is the time to perform one inference, in our case, we average that time over 30 frames.

All models above were trained on 100,000 iterations with a batch size of 24 images, representing approximately 65 epochs based on a training dataset of 37,000 images. As we can observe, the quantized MobileNet V2 achieved the best results, surpassing the results of MobileNet V3.

In our case, the difference in performance it's significant and the reason for that might be the fact that usually, MobileNet V3 needs between 3-4 times as many iterations for training to reach convergence comparatively to MobileNet V2, as observed from previous experiments.

Another plus of MobileNet V2 it's that the architecture is already quantized during training and we don't need post-quantization which might decrease the real-life performance of the model.

Based on those results, we select as the backbone for the system a Single Shot Detector based on a MobileNet V2 quantized during training and we can move forward with training the other stacked neural networks.

3.7 Training multiple neural network heads

In the final section of our solution, we present the training stage for multiple tabular neural networks used based on a stacking approach for different tasks. In stacking methods, we use the outputs of a first layer architecture as inputs for the second layer of estimators.

During this section, we train models for the following task:

1. Distance estimation between our vehicle and other objects
2. Detecting which objects are in front of the vehicle and which are not
3. Detecting when our vehicle is crossing a detected lane

As a training pipeline, we provide a general scheme organized in multiple stages used for all models:

- Stage 1: Bayesian Optimization, used in the original large hyper-parameter space
- Stage 2: Statistical error analysis for hyper-parameter space reduction
- Stage 3: We apply a grid search approach based on the reduced hyper-parameter space
- Stage 4: Error analysis over the results from Stage 3
- Stage 5: Hyperparameter manual search based on Stage 2 and 4 (Optional)

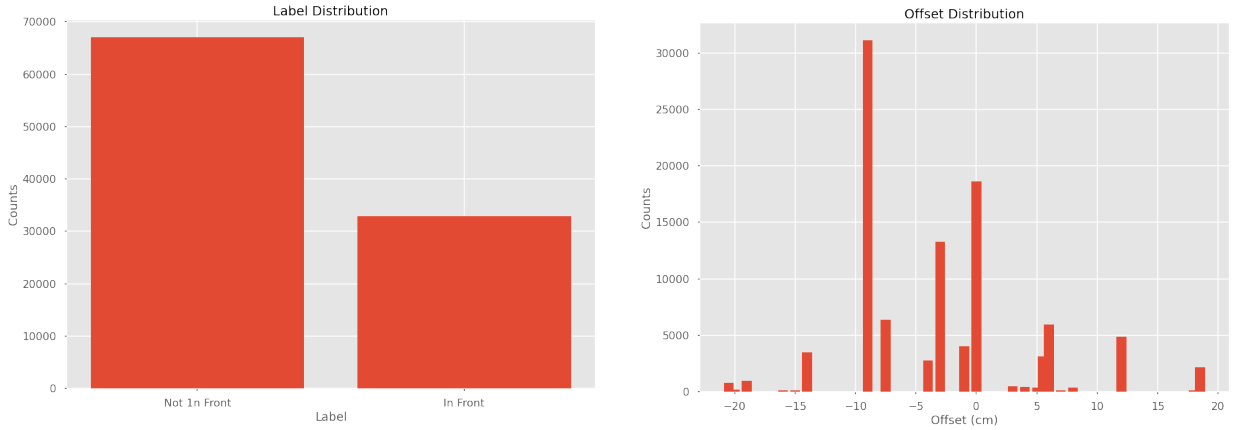
3.7.1 Camera Calibration Network

This neural network is used for predicting which objects are in front of our vehicle. This network predicts the in-front label only for Safe Distance Warning classes (Average size vehicles, Large size vehicles, Pedestrians, Motorcycles, and Crosswalks).

As we observe this is a binary classification problem, with possible inputs: bounding box coordinates, the aspect ratio of the bounding box, the area of the bounding box, the center, and the calibration offset provided by the user.

Those possible features are tested during the optimization stages we presented above as possible hyper-parameters.

Before training, we present the label distribution in Figure 3.27 (a) and the calibration offset distribution in Figure 3.27 (b):



(a) (b)

Figure 3.27: Label and offset distribution for camera calibration network. *(a) Label distribution for the calibration problem. (b) Offset distribution, the deviation of the camera from the center of the windshield, negative means towards the driver, positive means towards the passenger, assuming the steering wheel on the left side. Measured in centimeters.*

The first stage of hyper-parameter search is represented by **Bayesian optimization**. Considering a higher hyper-parameter space, a grid search approach would increase significantly the experimenting time. We use Bayesian optimization because it provides a good balance between exploration and exploitation in a reasonable amount of time.

We define as an evaluation metric a weighted average between precision and recall, with a higher weight on precision based on the fact that in real-environment we are more likely to see objects that are not in front and we want to reduce the risk of false positives.

We start the Bayesian optimization based on 512 randomly selected experiments in the hyper-parameter space and after those, we train for another 1024 bayesian selected experiments. Based on all 1536 experiments, we reduce the hyper-parameter space for the next optimization stage.

To reduce the hyper-parameter space, during the error analysis stage, we are not interested in one particular experiment and its results. We group hyper-parameters in discrete bins and we extract statistics like min, max, mean, standard deviation, and percentiles from that group concerning the evaluation metric, and we compare them with those from other groups.

As a simple example, assuming we have 3 preprocessing strategies for our input data: normalization, standardization, and using the raw inputs. We are interested if there is a difference between groups using the mean of those groups, not only based on a couple of selected experiments.

During this stage, we encounter the problem of models getting stuck to a local minimum. Based on the imbalanced dataset and the need of avoiding false positives, in our search, multiple models end up predicting all vehicles as being not in front. This is not necessarily a problem because, at this stage,

Bayesian optimization provides a guided search avoiding in the end those points of minima.

To better describe this problem, we present the experimental results over a double stratified test dataset based on the label and offset achieved by Bayesian Optimization in Figure 3.28.

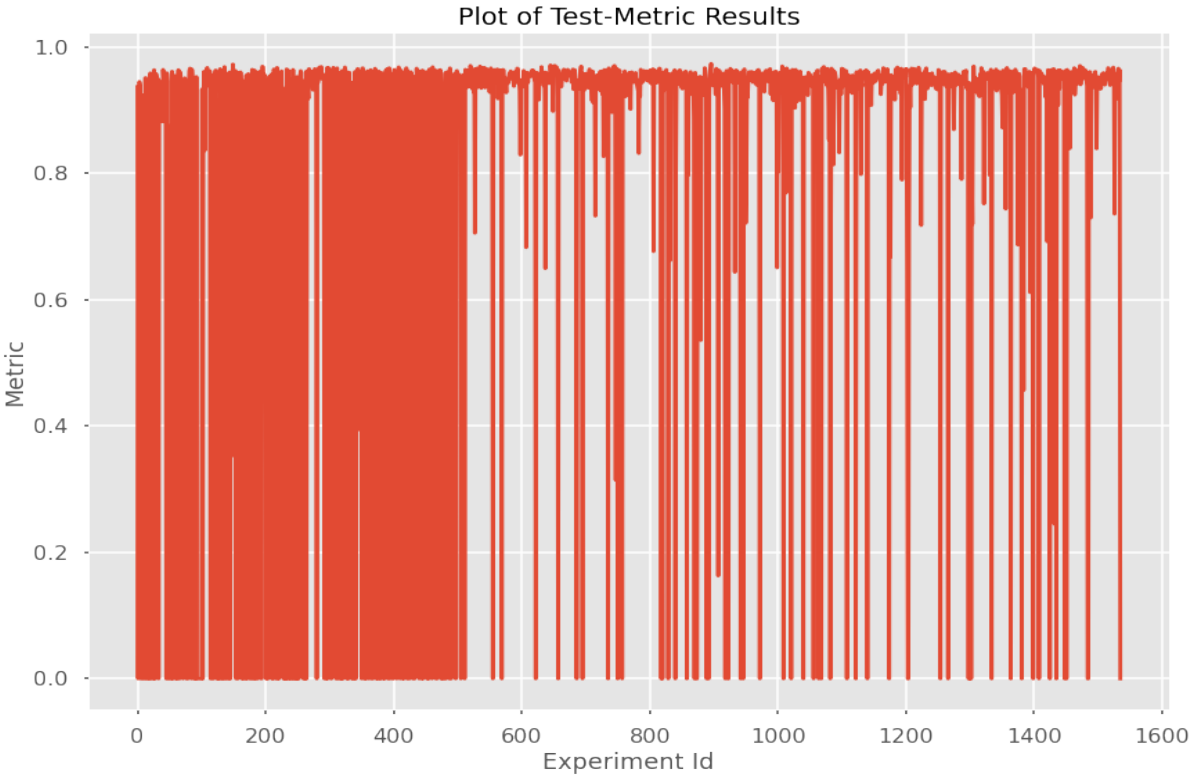
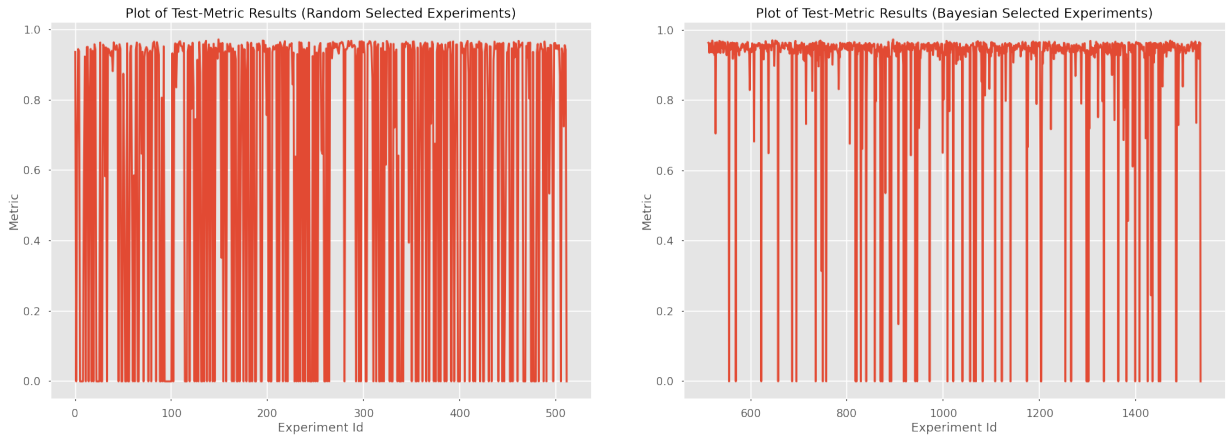


Figure 3.28: Experiments generated by Bayesian optimization

To take a better look at this situation, we divide those experiments into the initial random experiments, Figure 3.29 (a), and the selected experiments by Bayesian optimization, Figure 3.29 (b).



(a) (b)

Figure 3.29: Random selected experiments vs. Bayesian selected experiments. *(a) The first 512 random experiments generated during exploration stage. (b) The following 1024 bayesian selected experiments were generated during the exploitation stage.*

Based on the Figure 3.29, we observe how Bayesian optimization is better suited for this problem instead of a random selection search or a grid search.

Using the obtained results from the Bayesian optimization stage, we reduce the space of hyper-parameters by keeping as constants hyper-parameters statistically selected from the error analysis stage. In this obtained reduced space, we induce a grid search approach for a better level of granularity over the remained hyper-parameters.

After this final layer of optimization, we have a model ready for the production stage. After the first optimization stage, we obtain a model with 97.54% on the validation set and 97.35% over the testing stage. With the second stage done, we obtain improvements only on the validation stage 97.65% and the metric on the testing set stays the same at 97.35%. As we mentioned before, both results and all experiments were done using a double stratified split by the labels and the camera offset feature.

Based on those results, we move forward with training the neural network used for distance estimation.

3.7.2 Distance Estimation Network

We approach the problem of distance estimation based on a regression neural network with a single continuous output. We map, based on optimized thresholds, that continuous output in distance intervals, representing 5 categories of distance.

We present the label distribution below, Figure 3.30:

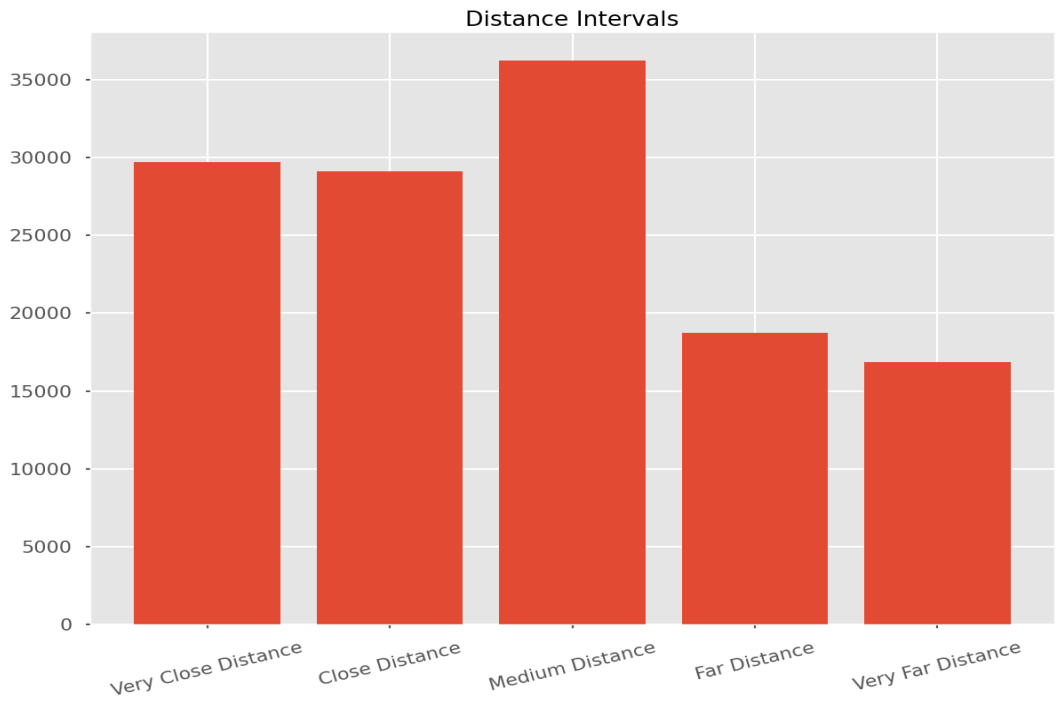
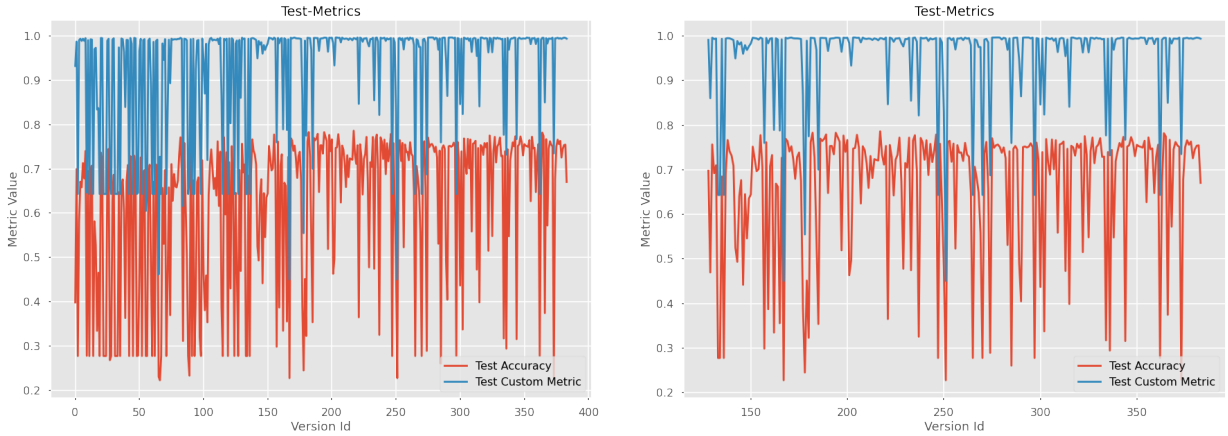


Figure 3.30: Distribution of distance intervals

The process of hyper-parameter optimization is similar to the one from the previous network. The main difference is the optimized metric. We choose to observe as metrics the accuracy of predicting exactly the annotated distance and a custom version of accuracy allowing for relative mistakes that were possible during the annotation process, we consider $+/-1$ differences from the annotated label as correct predictions.

We observe the same problem from the Camera Calibration Network, the problem of local minima points. Some of our estimators tend to predict only the predominant distance because this leads to a minimal penalty. This does not represent a problem because Bayesian optimization guides the search between those types of models.

We present the search for the optimal model based on Bayesian optimization, Figure 3.31, monitoring both the accuracy and the custom accuracy that allows for relative mistakes.



(a) (b)

Figure 3.31: Bayesian search for the distance estimation network. *(a) The complete stage of Bayesian optimization, including the randomly selected experiments from the start. (b) Bayesian selected experiments.*

After Bayesian optimization stage and grid search, we obtain a model with the following results:

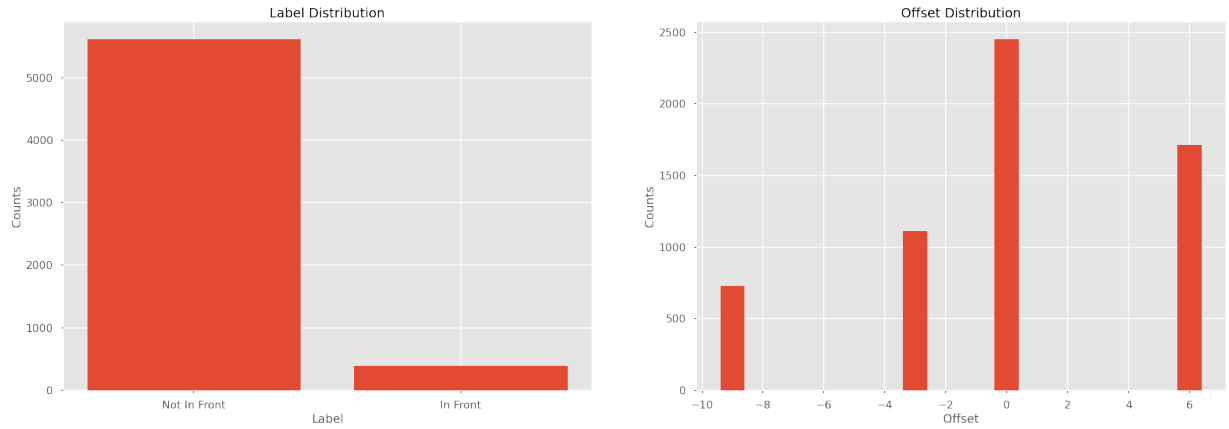
- Validation custom accuracy: 99.58%
- Test custom accuracy: 99.51%
- Validation accuracy: 80.75%
- Test accuracy: 80.47%

3.7.3 Lane Calibration Network

As the Camera Calibration Network, this architecture is a binary classification model and the optimization process remains the same as previously presented.

The main difference here is the fact that we are dealing with a highly imbalanced problem.

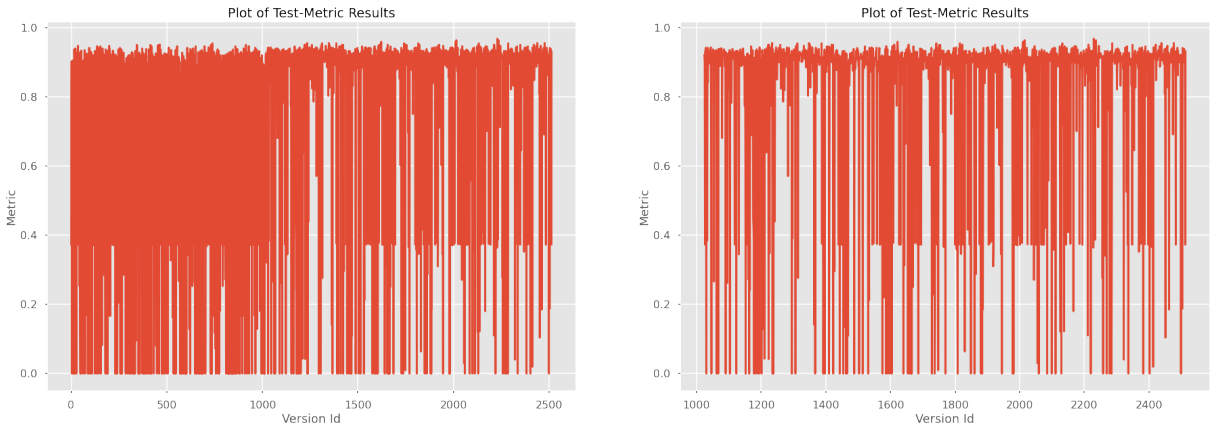
In Figure 3.32, we present the label distribution (a) and the offset distribution (b).



(a) (b)

Figure 3.32: Label and offset distribution for lane calibration. *(a) Label distribution, as underlined, highly imbalanced. (b) Calibration offset distribution.*

As we can expect this generates a problem also for Bayesian optimization to avoid points of local minima, Figure 3.33.



(a) (b)

Figure 3.33: Bayesian search for the lane calibration network. *(a) The complete stage of Bayesian optimization, including the randomly selected experiments from the start. (b) Bayesian selected experiments.*

The main reason why this problem is so imbalanced it's because crossing the white continuous lane it's illegal, so we don't expect this to happen all the time. And based on this fact we can't collect data in real-environment, especially without endangering the other drivers or pedestrians on the road.

In this scenario, we propose two methods for detection of bounding boxes augmentation:

- **Uniform movement of the box based on a threshold**
  1. Normalize bounding box coordinates between (0-1)
  2. Generate a random uniform variable representing the noise based on a given threshold $(0 < threshold < 1)$
  3. Move the bounding box based on the noise variable in one of the 8 possible directions (left, right, up, down, diagonals)
- **Normal sample augmentation**
  1. Given a dataset of samples (bounding boxes), normalized between (0-1)
  2. Compute the mean and the standard deviation from each coordinate
  3. Compute the mean of the standard deviations
  4. Compute a random uniform variable representing the noise, based on the mean standard deviation computed and a given value ≥ 0.
  5. Move on the mean sample computed (mean_x_min, mean_y_min, mean_x_max, mean_y_max) based on the noise variable in one of the 8 possible directions

We are interested in shifting bounding boxes around based on some rules for augmenting our dataset. As we can observe to prevent introducing unwanted noise to our dataset, we decide not to change the area or the aspect ratio of the bounding box.

In the following, we present a couple of examples with both methods, Figure 3.34, and Figure 3.35.

(a) (b)

Figure 3.34: Uniform movement augmented sample. *(a) Augmented sample with uniform noise threshold 0.01. (b) Augmented sample with uniform noise threshold 0.1.*

We also have samples that are not well augmented, for any augmentation method and any threshold values, Figure 3.36.

(a) (b)

Figure 3.35: Normal sample augmentation. *(a) Augmented sample based on the mean coordinates, moved by 2 standard deviations. (b) Augmented sample based on the mean coordinates, moved by 5 standard deviations.*

(a) (b)

Figure 3.36: Possible wrong augmentations. *(a) Augmented sample with uniform noise threshold 0.05. (b) Augmented sample based on the mean coordinates, moved by 2 standard deviations.*

The problem with these augmentations it's that we don't know when are done write and we generate quality artificial samples and when we add more noise that can confuse the classification model.

As a solution to this problem, we implement **Adversarial Validation** for checking the distributions between real-annotated samples and artificially generated ones. Generating a dataset composed by:

- Annotated samples (1)
- Samples generated with uniform random noise for positioning (2)
- Samples generated based on the movement of a mean generated sample (3)

We apply the adversarial validation training LGBM classifier optimizing for the ROC AUC score in between all 3 possible combinations. The results are the following:

- Validation ROC AUC score between (1) and (2): 65.7%
- Validation ROC AUC score between (1) and (3): 98.6%
- Validation ROC AUC score between (2) and (3): 98.2%

We present those results using minimal thresholds for augmentation, for uniform movement of the box we use a 0.01 threshold (as in Figure 3.34 (a)), and for normal sample augmentation, we use the number of standard deviations from the mean box selected only 0.1.

Based on adversarial validation, we find that augmentation techniques that for us might look natural, for a learning algorithm are easy to distinguish between the real annotated examples.

Our final model after the optimization stages presented above is trained on the following distribution, Figure 3.37, and it achieves 94.45% on the validation set and 96.46% on the testing set.

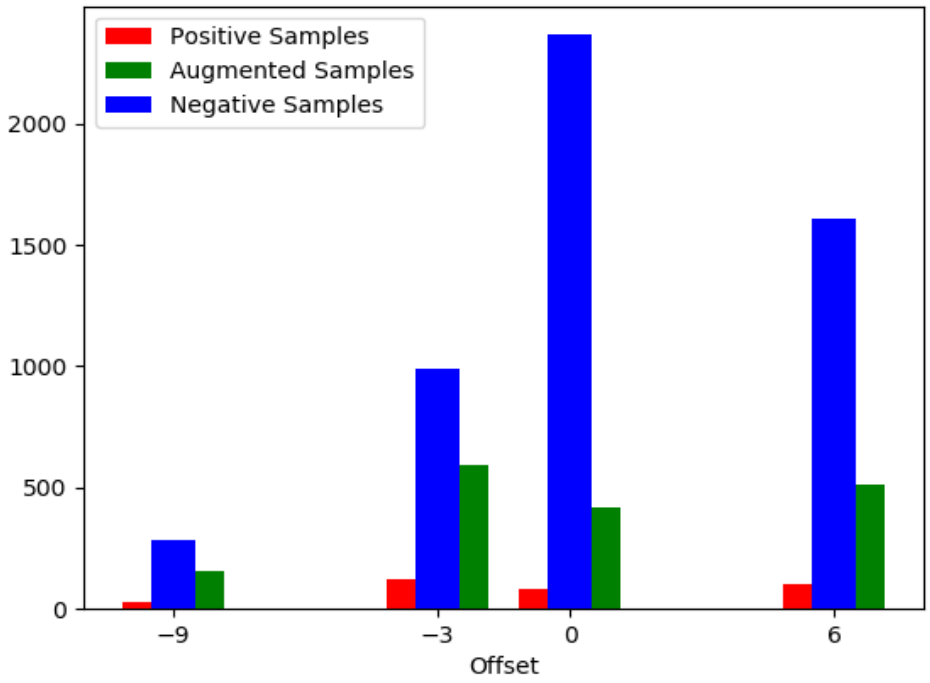


Figure 3.37: Label and offset distribution after augmentation

With those details presented, we conclude the Chapter 3 and the presentation of our solution. In the next chapter 4, we present our results.

4 Real-world application results

To quantify our results, we daily evaluate thousands of video segments proposed by the system from most of our clients, annotate them manually, we extract statistics based on the vehicle, client company, or even all video segments accumulated in a specific period.

To examine the behavior of Safe Distance Warning over a 1-2 days period, we present the Figure 4.1:

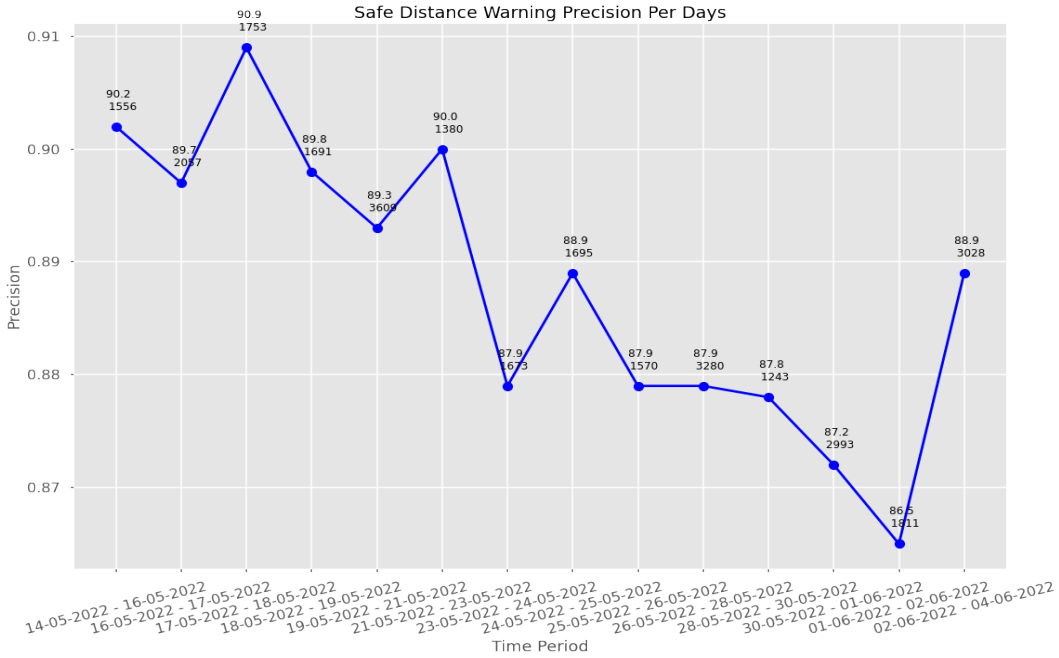


Figure 4.1: Safe Distance Warning daily precision.

In Figure 4.1, each point corresponds to a period and we plot for each the precision and the number of annotated video segments used for computing this precision.

As we said, we daily annotate true and false video segments from our clients. We are interested to see when our detections are true, representing in SDW context detecting one or more objects in front of our vehicle, when we are above a certain speed threshold selected based on the estimated distance, or when our detections are false. We distinct 3 categories for false detections:

- **Detection errors**: when the object detection network predicts false objects (e.g.: clouds)
- **Calibration errors, on the left side**: the calibration network detects an object as being in front of the vehicle, but is on the left side of the vehicle
- **Calibration errors, on the right side**: the calibration network detects an object as being in front of the vehicle, but is on the right side of the vehicle

Based on those, we present the prediction distribution over the entire period of analysis, Figure 4.2 and over just an entire month, Figure 4.3.

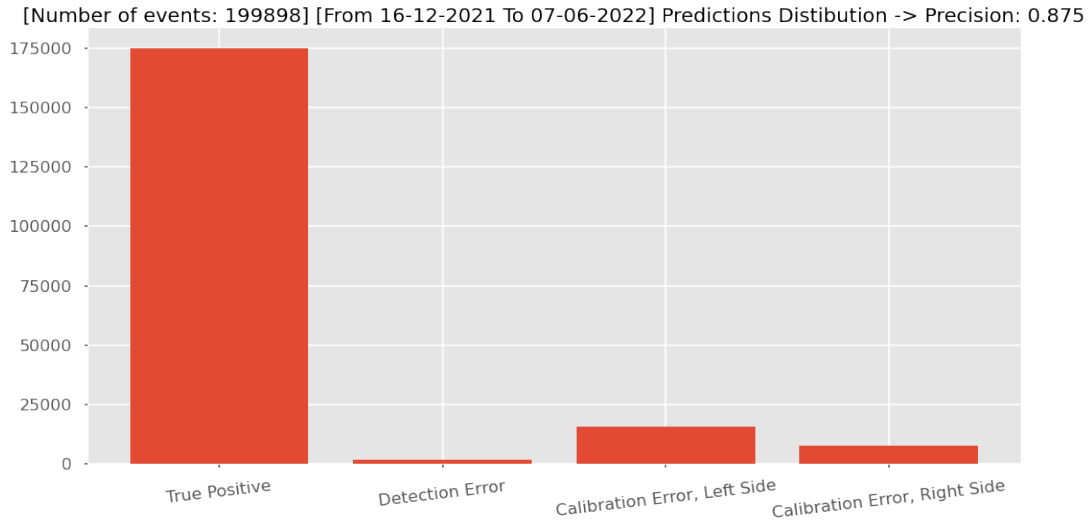


Figure 4.2: Prediction distribution, over the entire analysis period.

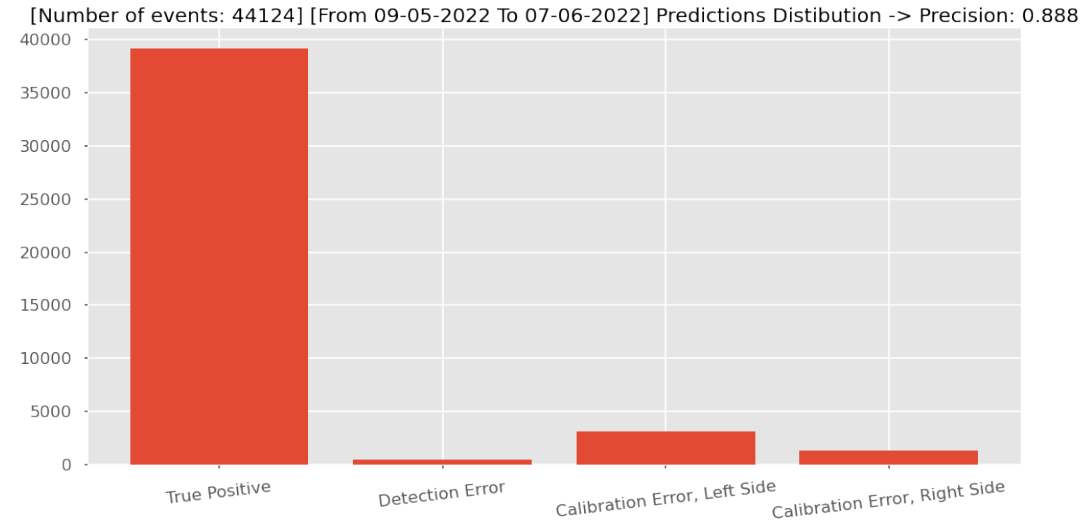


Figure 4.3: Prediction distribution, over one month.

As we can observe, the main problem for our system comes from the calibration network. This it's also the hardest problem to tackle because we rely on the input from users, which it's not always accurate.

To better represent this problem, we present the histograms of true positive and false positives predictions from the calibration network, Figure 4.4. Those histograms are composed of all video segments collected and annotated in the analysis period, 175,010 true positive events, and 15,601 false-positive events caused by calibration problems.

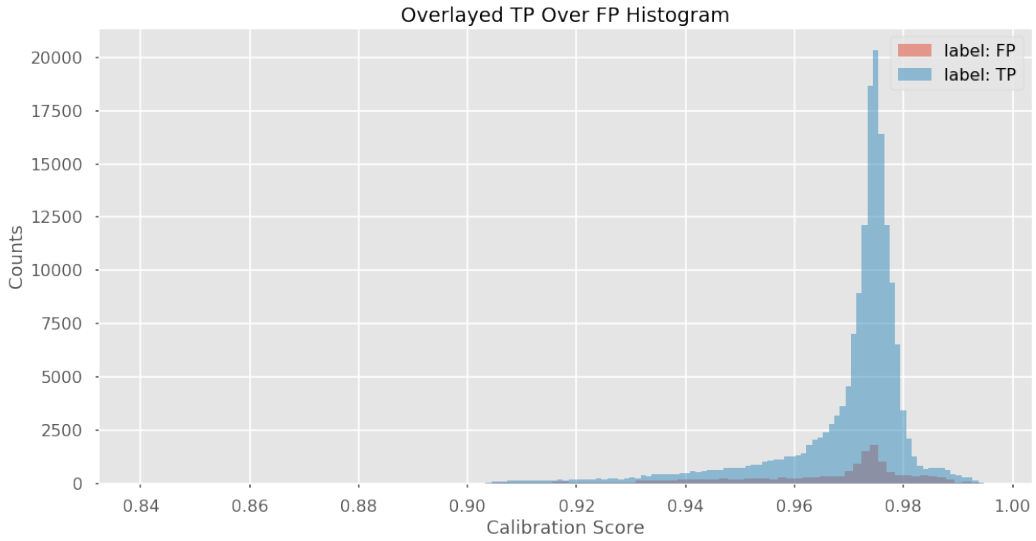
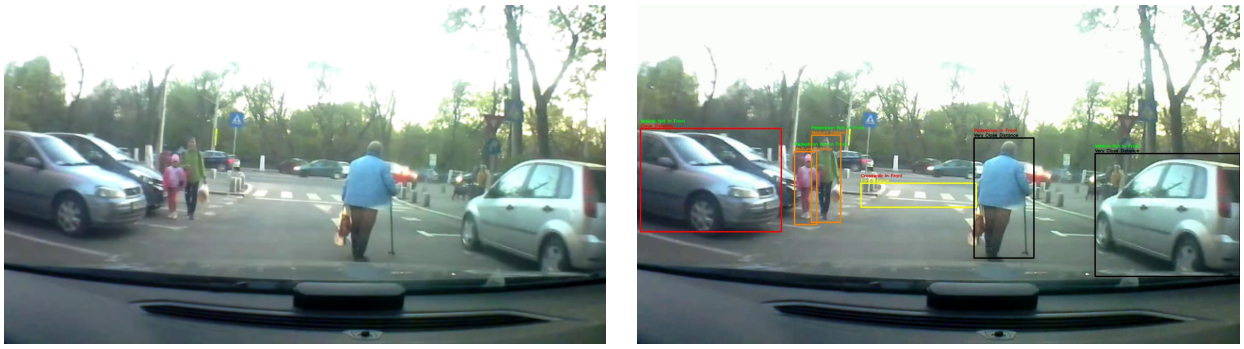


Figure 4.4: True positives and false positives calibration probabilities.

At the end of this thesis, we present a couple of pairs of ground-truth images and the correspondent outputs.



(Left) (Right)

Figure 4.5: Raw images and predicted results. *(Left) Raw input image. (Right) Predicted objects in image.*

(Left) (Right)

Raw images and predicted results. *(Left) Raw input image. (Right) Predicted objects in image.*

5 Conclusion and future work

We propose our system for Safe Distance Warning (SDW) and Lane Departure Warning (LDW) features, an architecture for low-latency embedded devices with a single camera that can be positioned anywhere on the windshield.

Based on a machine learning hardware accelerator, we present a stacked multi-network approach for solving multiple tasks:

1. We use as the backbone for our system an object detection model for embedded devices
2. Using a neural network, we estimate the distance between our vehicle and the detected objects
3. Based on two stacked neural networks, we predict which objects are in front of our vehicle

During this thesis, we also design a computer vision interpolation algorithm as a fail-system for our base network, we present an active learning pipeline for collecting and annotating data with less human intervention and in the end, we describe how using adversarial validation can improve the quality of augmented tabular data.

To further improve our system, we consider as possible directions the following:

- **On the current device:**
  - A system for estimating the calibration offset, removing the need for accurate user input
  - Using sensor fusion between the features on the inside camera/accelerometer and our system
  - An adaptive system for speed thresholding based on the outside environment
- **On a future device:**
  - Using disparity maps in stereo vision context for better distance estimation
  - Replacing object detection with tracking or using a fusion of results between them
  - Replacing the hardware accelerator with an EdgeTPU for improving the quality of detections

References

[1] Yichen Xie, Masayoshi Tomizuka, and Wei Zhan. Towards general and efficient active learning. *CoRR*, abs/2112.07963, 2021.

[2] Ksenia Konyushkova, Raphael Sznitman, and Pascal Fua. Learning active learning from real and synthetic data. *CoRR*, abs/1703.03365, 2017.

[3] Yazhou Yang and Marco Loog. Active learning using uncertainty information. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 2646–2651, 2016.

[4] Jing Pan, Vincent Pham, Mohan Dorairaj, Huigang Chen, and Jeong-Yoon Lee. Adversarial validation approach to concept drift problem in automated machine learning systems. *CoRR*, abs/2004.03045, 2020.

[5] Markus Nagel, Marios Fournarakis, Rana Ali Amjad, Yelysei Bondarenko, Mart van Baalen, and Tijmen Blankevoort. A white paper on neural network quantization. *CoRR*, abs/2106.08295, 2021.

[6] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[7] Pierre-Emmanuel Novac, Ghouthi Boukli Hacene, Alain Pegatoquet, Benoît Miramond, and Vincent Gripon. Quantization and deployment of deep neural networks on microcontrollers. *CoRR*, abs/2105.13331, 2021.

[8] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.

[9] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *CoRR*, abs/1801.04381, 2018.

[10] Adrian Iordache. Sistem de estimare a distanței între mașini, 06 2017.

[11] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3. *CoRR*, abs/1905.02244, 2019.

[12] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. *CoRR*, abs/1611.01578, 2016.

[13] Tien-Ju Yang, Andrew G. Howard, Bo Chen, Xiao Zhang, Alec Go, Vivienne Sze, and Hartwig Adam. Netadapt: Platform-aware neural network adaptation for mobile applications. *CoRR*, abs/1804.03230, 2018.

[14] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions. *CoRR*, abs/1710.05941, 2017.

[15] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[16] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: single shot multibox detector. *CoRR*, abs/1512.02325, 2015.

[17] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014.

[18] Mingxing Tan, Ruoming Pang, and Quoc V. Le. Efficientdet: Scalable and efficient object detection. *CoRR*, abs/1911.09070, 2019.