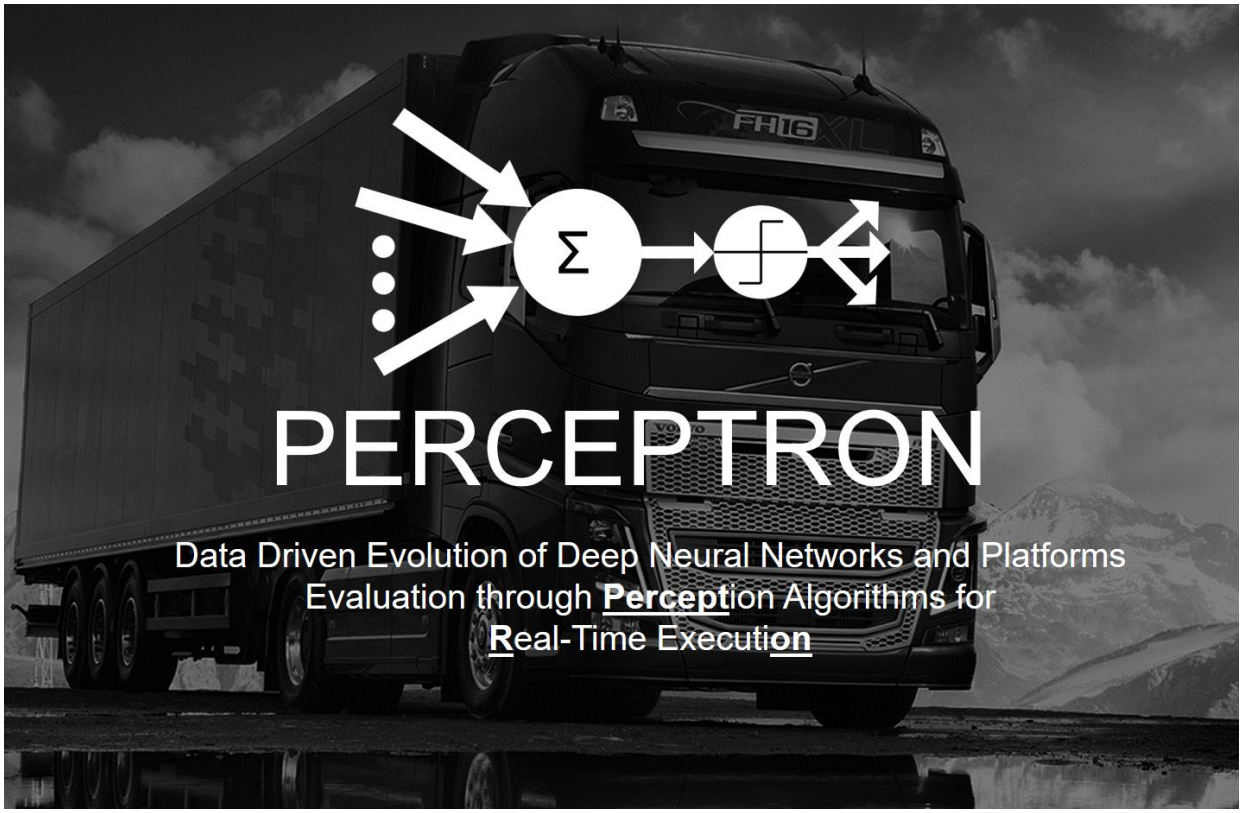
Perceptron

Public report



Project within Deep Learning for Embedded Systems

Authors: **Volvo GTT** - Saimir Baci, Carlos Gil Camacho, Selcuk Cavdar, Christian Ekholm, Jesper Ericsson, Henrik Kaijser. **Semcon** - Jens Henriksson, Laura Masaracchia. **Chalmers** - Måns Larsson.

Date: 2019-12-02

FFI Fordonsstrategisk Forskning och Innovation
VINNOVA Energimyndigheten TRAFIKVERKET
FKG VOLVO SCANIA VOLVO

Content

1. Summary

The global automotive industry is rapidly adapting Deep Learning as a key technology within several application areas, where autonomous driving is in the forefront. This poses a new threat to the Swedish automotive industry, since Deep Learning makes new products and services possible and thus provides a major competitive advantage. It is therefore of critical importance that the Swedish automotive industry builds Deep Learning competence immediately and starts adapting the technology.

The Perceptron project has addressed this need by developing a concept for data driven evolution of Deep Learning solutions, and built necessary infrastructure for logging data, training networks and for performing continuous deployment of the solution. This concept can be reused by the Swedish automotive industry, and the infrastructure itself can be used within Volvo projects.

A training platform is required to develop a Deep Learning solution. It is a problem for OEMs to know which training platform to rely on, since the field is new and changes fast. The Perceptron project have investigated different training platforms. Similarly, embedded inference platforms have been studied in the project. The knowledge and guidelines generated can have a significant impact on future product decisions and are intended for OEMs as well as suppliers.

Swedish OEMs have a short term-need for perception applications, since they provide fundamental building blocks for safety as well as autonomous driving applications. Although there has been some research done on perception in the car setting, the research volume related to trucks is very limited. As a result of the project, state-of-the art real-time perception applications for object, lane and free space detection in the truck setting, have been developed. The architecture of these solutions can be reused by the Swedish Automotive industry, and the solutions themselves will be constituents in future projects performed by each project partner.

At the end, the main goal fulfilled for the project was the increase of the knowledge in the Swedish industry for Deep Learning applications. For example, during the project it has been shown that the collection and processing of the data is one of the main time consuming activities of the complete chain. Finding techniques and solutions that help to reduce this time will be crucial for the development of the sector. The selection of the hardware for inference and training is another important factor to take into account for your solution. Despite nowadays, most of applications relies on GPUs for performing the different operations, other options that are more energy efficient, like FPGA or ASICs, will have an important role in the future. The importance of training the networks with data adapted to your use cases is also very important to achieve optimal results, e.g. data containing images recording from the same kind of vehicle where you want to run your network. In summary, we hope that these results and conclusions from the project, can be used for the industry in the future when developing Deep Learning Solutions for Autonomous Driving.

2. Sammanfattning på svenska

Maskininlärning och speciellt Deep Learning är relativt nya tekniker som innebär nya utmaningar för svensk fordonsindustri. I synnerhet inom området automatiserad körning är Deep Learning en teknologi där svensk industri inte har råd att hamna på efterkälken och bör därför satsa på att snabbt utöka kompetensen på området och utnyttja denna för att bygga nya tjänster och produkter.
Perceptron-projektet är ett led i detta initiativ genom att utveckla datadrivna metoder för att konstruera Deep Learning-lösningar. Detta innebär bland annat att bygga den infrastruktur som krävs för att logga data, träna neurala nätverk och sedan ladda ner dessa nätverk till fordonen där de sedan körs i sin målmiljö. Kunskapen från dessa aktiviteter kommer kunna tillämpas av svensk fordonsindustri och infrastrukturen kommer att vidareutvecklas och användas på Volvo.
Det finns nu ett starkt behov av applikationer som utnyttjar Deep Learning-teknologin för automatiserad körning och trafiksäkerhets-funktioner och det finns redan en del forskning på området. Denna har dock till största del fokuserat på fallet med personbilar medan det i fallet med tunga fordon knappt finns någon forskning alls. Därför har Percepton-projektet fokuserat på det senare och har utvecklat tre state-of-the-art-applikationer för perception med realtidsprestanda: objektdetektering, filmarkerings-detektering och detektering av körbar area. Arkitekturen för dessa applikationer kommer fungera som bas för framtida produkter inom Volvo.
Förutom de utvecklade applikationerna genomfördes undersökningar av hur man bäst utvecklar perceptions-applikationer och vad som är den största utmaningen och kräver den största arbetsinsatsen. För att träna ett neuralt nätverk krävs en tränings-plattform och det finns i nuläget ett antal sådana plattformar att välja mellan. Det finns även olika teknikval att göra när man skall köra det neurala nätverket i ett fordon, den så kallade inferensen. I Perceptron-projektet har båda dessa aspekter undersökts och rekommendationer har tagits fram. För inferens valdes GPU som bäst lämpliga hårdvarutyp och detta var även valet för träningen av de neurala nätverken där en dedikerad maskin från Nvidia (DGX Station) användes. Som träningsramverk användes både TensorFlow och PyTorch och det finns fördelar med båda ramverken vilket fick till följd att ingen klar vinnare kunde koras.
Den enskilda aktivitet som tog längst tid var skapandet av ett eget dataset. På förhand visste vi att det skulle vara en arbetskrävande aktivitet men det visade sig ändå att vi underskattade arbetet. Detta berodde delvis på införandet av GDPR-lagstiftningen under projektets gång vilket tvingade oss att genomgå en process för att få godkännande av Volvos legal-avdelning innan vi startade datainsamlingen och tvingade oss att anonymisera alla bilder innan Volvo kunde dela dem med övriga partners i projektet.

3. Background

There is a Machine Learning revolution on-going in the automotive industry, and the field as such has now passed the peak of inflated expectations according to the latest Gartner hype cycle **Error! Reference source not found.**. Deep Learning (DL) is arguably the most promising subfield of Machine Learning and has gained tremendous traction over the recent years in both academia and industry. The breakthroughs seen with DL have been enabled by theoretical advancements, increasing computational power and availability of big data sets.

The rapid advancement in DL is starting to affect the established automotive industry, through software companies like Google entering the automotive scene and with fast adapters like Tesla. This poses a major threat to the Swedish automotive industry, since DL makes new products and services possible and thus provides a competitive advantage. It is therefore of critical importance that the Swedish automotive industry builds DL competence immediately and starts adapting the technology.

There are many automotive application areas of DL including security, diagnostics, safety and autonomous driving. This project has used perception as the application area, i.e. the ability for the vehicle to perceive its surrounding environment. Perception applications typically provide DL fuelled building blocks in the current state of the art solutions for active safety and autonomous driving. Considering its strong position in active safety solutions and the global push for autonomous driving, there is thus a major short term need in the Swedish Automotive industry for perception solutions.

So far, much research has focused on the autonomous car in contrast to the autonomous truck. However, some analysts have predicted that the impact of autonomous technology will be more disruptive to the truck and semi-truck industry than the car industry. Despite similarities there are significant differences in technical, market and environmental challenges for the truck industry compared to the car industry. This project has partly aimed at addressing the gap in autonomous research volume by focusing on enabling technology for the autonomous truck.

4. Purpose, research questions and method

4.1. Purpose

The main purpose of this project is to increase the knowledge within the Swedish industry on DL solutions for autonomous vehicles. Since this is such a big area, we focused on covering the following needs within the industry:

- *Develop and validate a concept for data driven evolution of DL applications.*

  Data driven evolution is an efficient methodology for explicitly programmed embedded solutions, e.g. Simulink models or C programs. There is a need for tailoring this methodology for development of DL solutions. DL solutions are heavily dependent on training data that are collected by logging in vehicles. The amount of data required is generally big, since training DNNs typically requires many hours of driving data. This requires a lot of infrastructure that handle the compression and acquisition of the data. In the Perceptron project, there are thus needs for infrastructure components related to logging, transfer and deployment.

- *Find the optimal DNN training platform and embedded execution platform.*

  There are several training and inference platforms on the market, and new ones will likely appear in the coming years according to recent announcements by several large companies [2] [3]. The training platforms are normally PC-based but differ in scope and what kind of computational hardware they support and whether that hardware is local, or cloud based. There is a need to investigate these training platforms to be able to select the most suitable one.

  The process of executing a trained DNN on new input data is called inference. There are different kinds of inference hardware that can be used in an embedded system, mainly FPGAs and GPUs. There is a need to investigate these types to understand their pros and cons. Off-the-shelf devices containing this hardware need to be investigated to learn what inference devices to use going forward. Apart from the training and inference hardware, the software platform plays a major rule in both cases and has to be taken into account.

- *Develop state of the art perception DNNs adapted to Volvo Trucks.*

  There is an OEM need to build competence to develop DL solutions. Complex DNN architectures and training methods are important parts of the development process to master. Partly this is because the DNNs themselves are needed for upcoming active safety and autonomous driving solutions, partly due to the fact that OEMs need tailor-made DNNs since they adapt to the actual sensor types and the geometric position and orientation of the sensors during training. There are needs for all three main perception applications object detection, lane detection and free space detection. The DNN adaption to the specific sensor configuration means that a DNN for object detection trained with a particular camera type at a specific geometric position will perform worse with other camera types or different positions. There are some very deep neural networks that to some degree have learned to

compensate for different sensor characteristics and geometry, but one drawback is that inference with such networks requires higher computational capacity. In order to understand when it is necessary to tailor a DNN solution instead of buying a COTS (Commercial Off-The-Shelf) solution, there is a need for OEMs to learn more about how reliant DNNs are on the specific sensors and their geometric locations. On a similar note, real-time performance vs. accuracy properties needs to be investigated to find optimal balance.

4.2. Research Questions

The technologies covered by the Perceptron project represent several relevant and advanced research issues. A set of research issues (RQs) and some hypotheses are listed below.

RQ1. How do you tailor the Build-Measure-Learn model to handle data driven evolution of DL solutions?

Build-Measure-Learn is a well-established lean development model that has been applied by e.g. Chalmers Software Center for data driven evolution of software systems. **Error! Reference source not found.** describes a hypothetical concept that tailors this model to development of DL solutions. The outmost loop is identical to the original model, where the next iteration of the solution is built and then deployed to the real truck environment. Continuous deployment is facilitated by transferring the DNN through a mobile data network to the truck. Performance is measured in the truck and the data is analyzed to learn about the solution. This generates new ideas that are fed into the build phase of the next iteration. In the DL case, there are two additional inner loops required. The top inner loop is the training performance loop. During the training process the performance of the DNN is typically measured within the training environment. The performance data is analyzed to learn about the solution, and this gives ideas about what to change in the next step of the training process. Similarly, the bottom inner loop is the training data loop. When there is a need for more training data, it is fetched from the backend in the cloud and annotated. If there is not enough data available in the backend, vehicles are configured to start logging more training data that is eventually transferred to the cloud.

Since the volume of training data is big and the truck is typically driving and covering long distances, feasibility and cost poses problems for transfer via 4G/5G networks. Accordingly, this concept transfers data to the backend via WLAN at hotspots where the truck naturally stands still for some time, for example at transportation hubs.

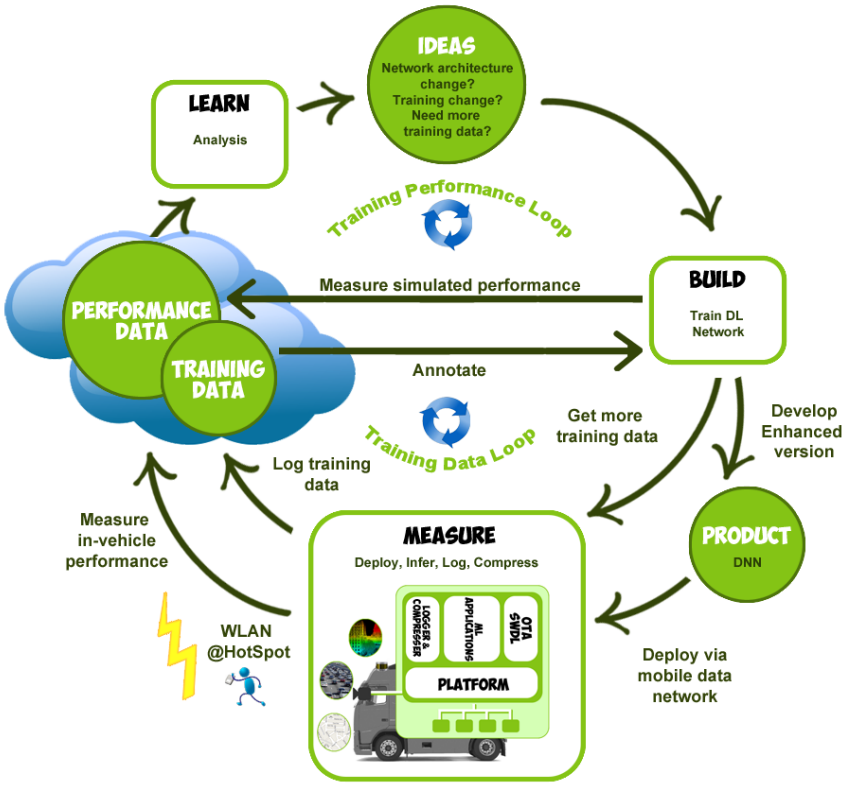


Figure 1: Hypothetical concept for data driven evolution of DL solutions.

RQ2. What methods can be used during data collection to reduce the amount of log data to transfer to the backend?

The simplest method is to apply some standard lossless compression algorithm in the vehicle and then decompress the data after it has been transferred to the backend. A more sophisticated approach would be to use statistics and to measure the new data for information content vs. the already logged data. New images or point clouds that are similar to already logged data will have low value and may be discarded. Using this approach, a good coverage of the input space can be reached with drastically reduced volume of data [4]. A similar approach is to only log feature rich images, for examples images that contain multiple vehicles.

RQ3. What kind of inference platform is suitable for which embedded inference situation?

Training and inference platforms are both key components in development of DL solutions. Because the inference platform is an in-vehicle mass produced component it is more critical for the OEM from a cost and integration perspective and will thus be the research focus of the project.

FPGAs and GPUs both have their proponents. It was the hypothesis in the Perceptron project, that there is no such thing as an optimal hardware but this will rather be dependent on problem type, performance requirements, cost etc. Guidelines are needed for hardware type selection. For

example, there are some early research indications that DNN of the type convolutional neural networks are more suitable for FPGAs than GPUs. Regarding the software platform of the inference hardware, there are differences depending on what kind of deep learning framework it supports that has to be taken into account for a particular inference situation.

RQ4. How do you improve DNN based perception applications beyond state of the art?

Current state-of-the-art DNNs are tailored for binary outputs, for instance "presence of a pedestrian", while the perception applications the project is interested in involving structured outputs, e.g. free space detection. Combining DNNs with Markov Random Fields is a promising research direction to model spatial and time correlations of the output. This line of approaches is currently the best performing methods for semantic segmentation tasks [5], but it is an open research issue if they can be applied to perception tasks under real-time performance constraints. In general, real-time performance is an area where there is much improvement to be made to current state of the art procedures, which often focus only on yielding the best accuracy while the computational constraints are disregarded. Perceptron has addressed this gap.

Applying multi-dimensional convolutions to take more sensor information into account is a promising approach. This can be used to fuse camera and LiDAR information in the DNN. The addition of LiDAR intensity and range map to camera input can plausibly improve state of the art classification since they are highly informative features.

RQ5. How important is it to train and infer using the same sensors and geometry?

Actual sensor types and geometric locations will appear in multiple variants in production vehicles. It is therefore useful to study the effects of using training data collected with different sensor configurations.

4.3. Method

Following the state-of-the-art in data driven evolution, training and inference platforms was the main method used to guide the project to the proposed goals. The new results published within the field were used to identify the technology, while adaptations and combinations of existing technology made it possible to fulfil the objectives.

The Build-Measure-Learn model was the main method that guided the development of the project. This is a model for data driven evolution of software systems, see Figure 2 below. The point of this pattern is to guide the next iteration of the system by using actual facts about the system acquired through measuring it. These measurements make it possible to learn properties of the system, yielding ideas for improving it. However, this is not the most common development cycle in the automotive industry. On the contrary, automotive companies often plan their development activities based on expert judgement of the current system rather than objective facts. The perceptron project tailored this model by adapting it to development of DL solutions.

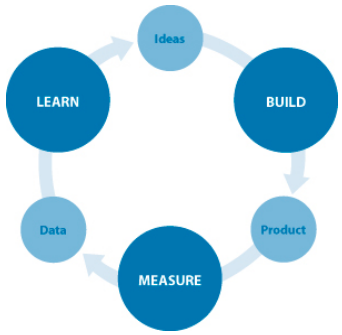


Figure 2: The Build-Measure-Learn model based on data driven evolution of software systems.

5. Objective

5.1. Contribution to Overarching FFI objectives

- *Increasing the Swedish capacity for research and innovation, thereby ensuring competitiveness and jobs in the field of vehicle industry.*
  The essence of the Perceptron project was to increase the Swedish capacity in DL for the perception area, which is highly significant for safety functions and autonomous driving. Considering the current domestic vs. international capacity, it is immensely important that the Swedish automotive industry catch up or jobs will by all likelihood be endangered.
- *Developing internationally interconnected and competitive research and innovation environments in Sweden.*

  The Perceptron project have created a new research and innovation environment among the partners. All partners have international connections. Both Semcon Sweden AB and CTH have produced world-class contributions to the area.
- *Promoting cross-industrial cooperation.*
  The concept for data driven evolution of DL solutions may easily be applied in other industrial domains. DL in general and new DL technology for perception applications in particular are relevant for other domains. Industrial cooperation has been promoted by dissemination of the Perceptron results at events that include companies from other industries.
- *Promoting cooperation between industry, universities and higher education institutions.*
  The Perceptron project was a cooperation between industry and university, as two partners are industrial and one is a university.
- *Promoting cooperation between different OEM.*
  Dissemination of Perceptron results at events where other OEMs attend has promoted cooperation, since DL and perception applications are highly important for all Swedish automotive OEMs.

5.2. Contribution to sub-programme research fields and roadmap subareas

The main sub-programme research fields addressed by Perceptron were "*Electrical architecture for embedded and connected systems"* and "*Technologies for green, safe and connected functions"*, which is elaborated further below. There is also a strong relation to the research field "*Adjacent areas with potential to strengthen Swedish automotive industry in a global perspective"*. The Perceptron project contribute to the roadmap subareas according to the following.

- *Architecture On-Board*: Hardware and software for on-board inference has been investigated. DNN inference requires purpose fitted embedded software blocks. It also requires high computational performance, and there needs to be architecture concepts for how to host the computational capacity, how sensor data is distributed to it through communication buses etc.

Hardware and software for data logging, transfer of data to backend and continuous deployment of DNNs are covered.

- *Electric Architecture Off-Board*: DL solutions require big amount of data which makes an off-board infrastructure for handling big amounts of data necessary. Data annotations are used by the training algorithms, which means that the off-board infrastructure must both store and provide access to such annotations. Perceptron has examined the full DL development chain and the interplay between on-board logging components and off-board data infrastructure. Regarding PMT, the Roadmap states that "focus is on exploiting the opportunities afforded by connected vehicles and decoupled SW engineering", which is exactly in line with the proposed concept for data driven evolution of DL solutions.
- *Technology for Green, Safe and Connected functions*: Perception applications can enable environmental friendly features like optimizing engine control based on traffic situation awareness. They are also important keystones for safety functions, since they necessitate a model of the world surrounding the vehicle. Connected functions are addressed both through transfer of log data and trained DNNs, but also in the sense that the DNN solutions run in a particular truck is conceptually affected by training and performance data collected from a fleet of trucks, i.e. each truck is part of an ecosystem of trucks and off-board infrastructure.
- *User Experience / HMI*: Although Perceptron did not directly address HMI, the DNNs delivered by the projects supply information that is valuable to display by the HMI. For example, imagine an augmented reality function that highlights a pedestrian in a camera view.
- *Verification & Validation*: The project was partly about development methodology for DL solutions, which is a new need in the automotive industry. Validation of the functions, including testing in real trucks, is part of this methodology although focus was not on full system validation. The roadmaps competence gap tells that there is need for more experienced engineers in verification & validation, and Perceptron builds V&V experience in the new DL domain including a state-of-the-art inference platform.

5.3. Measureable goals

The measureable goals of Perceptron were

- *G1. Develop a concept and infrastructure for data driven evolution of DL applications including supporting infrastructure.*

  The concept will be validated by applying it to develop 3 Perception DNNs, see *G3* below. The infrastructure shall support logging data at a rate of at least 25 MB/s, which will make it possible to log at least two cameras, two LiDAR sensors and a set of CAN signals. The logging system shall handle at least 3 hours of logging/day in one truck. On one hand, 3 hours provide a significant amount of data. On the other hand, assuming a lossless compression factor of 3 and a wireless transfer speed of 7 MB/s, it is feasible to transfer the logged data in roughly 3 hours. The truck standing still at a hotspot during three hours/day is a reasonable assumption. Furthermore, it shall be possible to perform continuous OTA deployment of a new DNN, i.e. the process to transmit and make the DNN

executing in the inference platform should require less than 10 minutes. For the final solution of Perceptron the transfer of data between the on-board hardware and the off-board hardware was simplified to using physical hard drives. This is due to a number of reasons, on one side the quantity of data logged was too high and the need to train with raw data as much as possible makes it impossible to use the current compression algorithms, this is one of the main reasons why most of the OEMs in the world choose to use physical drives. At the same time, the emergence of new legislations during the development of the project such as the GDPR, forced us to focus in other areas like anonymization. Overall, this solution allows to log from two cameras, two LIDARs and from the CAN network for more than five hours per day.

- *G2. Make a training and inference platform survey, and evaluate one training and one inference platform.*
  The evaluation shall take the following aspects into account: cost, usability, functionality, performance, limitations, automatisation and deployment. In this survey, we evaluate the currently most common hardware types to perform training and inference and at the same time we evaluate the most used frameworks for DL.

- *G3. Develop 3 DNNs based on state of the art methods that detect objects, free space and lanes, respectively.*

  Each DNN shall use Camera and/or LiDAR data as input. To safeguard real-time capabilities, the execution time of each network when run stand-alone on the inference hardware should be less than 100 ms. For accuracy, the trained DNNs shall be among the top 10 performers on public benchmarks such as the KITTI road detection benchmark [6] when compared under similar performance settings, i.e. less than 100 ms execution time and same sensor data. Due to the license issues with the KITTI dataset, other public benchmark datasets were used for the result of the project such as COCO [7].

6. Results and deliverables

The project was organised in eight Work Packages (WPs), see the figure below. The arrows indicate that there was a delivery from one WP to another, either of an explicit project deliverable or knowledge. The figure also indicates in which work packages each research issue and goal were addressed. WP2 develops the infrastructure and that was then used in WP3 to log training data. WP4-6 handled one perception application each using this training data, and the resulting DNNs were fed to WP8 that built the demonstrator truck. In parallel, the platform survey and evaluation were performed in WP7 having several dependencies to the other WPs. Finally, coordination and dissemination were the responsibility of WP1. Figure 3 describes the relation between the different work packages.

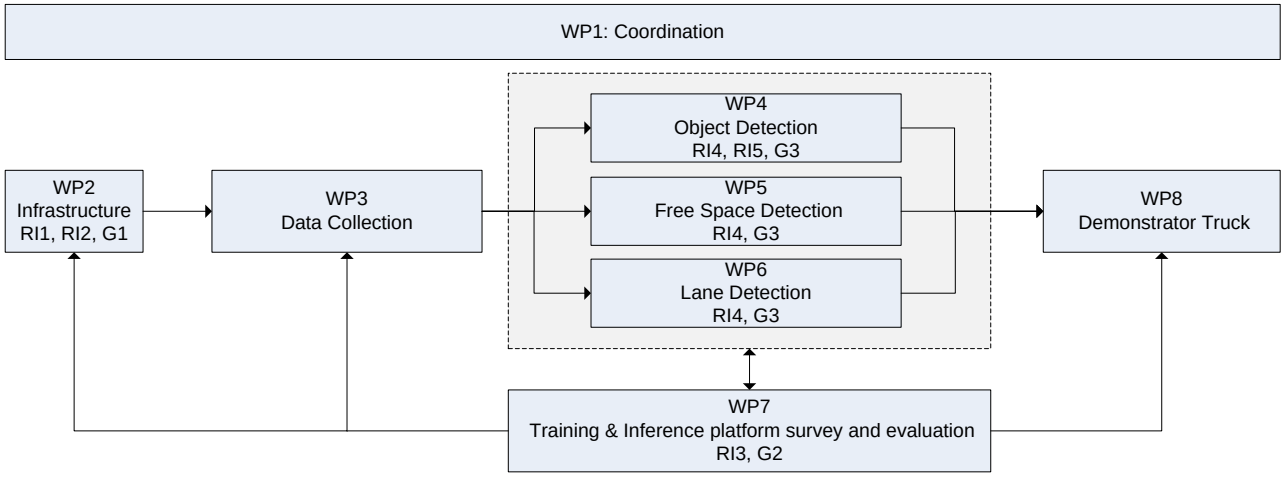


Figure 3: Project setup in terms of work packages and flow of deliverables.

A holistic perspective on OEM and supplier's Deep Learning needs were applied in the project. The interplay between the different topics covered were considered rather than treating each topic as a separate study. This was a suitable approach since data acquisition, DL application development and training and inference platforms are tightly connected.

Dissemination was a shared responsibility between the partners and was handled within each work package. Below, the result of the work packages are described along with their deliverables.

6.1. WP1 – Coordination

Perceptron was a cooperative project where Volvo, Semcon and CTH have been collaborating closely. Volvo was the main coordinator in charge of planning regular meetings and synchronize the communication between the partners. Apart from this, different tools were used to organize the deliveries from the different work packages. One example of these tools, were the SVN repositories that was used to coordinate the common files during the project. This coordination work was one

of the main factors that allowed us to facilitate the communication and finish the project successfully.

6.2. WP2 – Infrastructure

This work package contained the whole infrastructure developed for logging and deploying in the truck. The acquisition of data is one of the most important phases in the development of Neural Networks solutions. The DNNs that we used were trained with supervised learning methods in order to increase their performance. These algorithms learn to recognize patterns from data so to train a network requires a proper quantity of data and sufficient variation in the data. Acquiring and processing this data is normally a costly process that requires a significant time investment. Developing a good infrastructure for logging data can help to reduce the cost and the processing time making this one of the most critical parts.

At the same time, having an appropriate infrastructure for the deployment of the networks makes it possible to run the networks in the most optimized way. Having the suitable hardware and software for running the networks is equally important to succeed in the development of a data driven DL solution.

6.2.1. Results

In autonomous vehicles solutions, the choice and distribution of the sensors is always one of the critical steps. On one hand, a lack of sensors may lead to that the perception system cannot detect all the objects around the vehicle. On the other hand, too many sensors increase the cost of the solution and might produce redundant information. Also, the location of these sensors is crucial to capture as much as possible of the field of view around the vehicle.

In addition, when logging data for training the DNNs, it is important to have the sensors in similar positions of the sensors that will perform the inference process. Having the same point of view both for training and validation will increase the accuracy of the system.

For the Perceptron Project, these points were all considered, and several sensor configurations were studied. Due to the scope of the project and for simplicity, the sensors were placed in the front of the vehicle. More concretely, four sensors were used for logging data, two front cameras and two LIDARs [8] at the front sides. The cameras were placed close to each other to have the possibility of using them as a stereo camera if needed. Figure 4 shows the sensor location in the demonstration truck.

Figure 4: Sensor placement for the logging system in Perceptron.

This configuration allowed to capture the information in front of vehicles which can be used to train our Deep Neural Networks (DNNs) for Lane Detection [9], Object Detection [10] and Free Space Detection [11]. **Error! Reference source not found.** illustrates the field of view of the sensors placed on the truck.

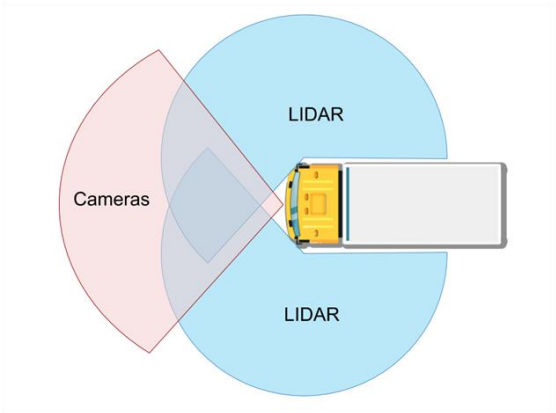


Figure 5: Sensors Field of View for Perceptron.

This sensor configuration was chosen due to its flexibility since is covers most of the possible scenarios:

- 2D cameras are very versatile sensors that works in the same way as our human vision, so they can interpret colours and read signals with the proper software. They are less expensive than LIDARS and they can be used under certain weather conditions, like rain or fog. For these reasons they are the most popular sensors in autonomous vehicles and some companies like Tesla states that with enough number of cameras they can provide a complete autonomous solution without the need of LIDARS [12].

- On the other hand, LIDARS use lasers to measure the distance between the vehicle and the surrounding environment. They are very accurate and give the information in 3D which allows to get the location of an object in a 3D world directly. Since they use time of light

technology, they are not as affected by shadows or bright lights as cameras. The problem is that they are an expensive technology and the accuracy can be affected by fog or rain.

The next table shows the properties of the sensors chosen for our demonstrator truck:

Sensor	*Description*	*Position*	*Usage*	*AOV (Hor x Ver)*	*EFL [mm]*
Cam 1	Left 2D camera	Behind the windshield, on the left side, 10 cm from the center, 2.3 m from the ground.	Capture images from ahead.	60° x 36,6°	5.8
Cam 2	Right 2D Camera	Behind the windshield, on the right side, 10 cm from the center, 2.3 m from the ground.	Capture images from ahead.	60° x 36,6°	5.8
LIDAR 1	Left corner LIDAR of 16 layers	Below the left-hand mirror, about 1.7 m above the ground.	Capture point cloud from ahead and left side.	225° x 30°	-----
LIDAR 2	Right corner LIDAR of 16 layers	Below the right-hand mirror, about 1.7 above the ground.	Capture point cloud from ahead and right side.	225° x 30°	-----

Table 1: Logging Sensors Specifications.

Once the sensor configuration was chosen, it was necessary to acquire a hardware for logging. This hardware contained all the drivers necessary to run the sensors and was in charge of collecting all the data. In our case, the logging platform chosen was the Drive PX 2 from Nvidia. This platform allowed us to connect up to 12 cameras using Gigabit Multimedia Serial Link (GMSL) interface. The platform also offers the possibility of connecting other sensors through Ethernet which is the case of the LIDARs on the truck. It contains two Tegras SoCs with two discrete GPUs. The GPUs contains 4GB of memory which offers reasonable capabilities of inference using DNNs.

An SSD was connected to the Drive PX 2 to store the logged data. This SSD can be easily transported to deploy the data in another station for processing. **Error! Reference source not found.** shows the diagram of the hardware connections.

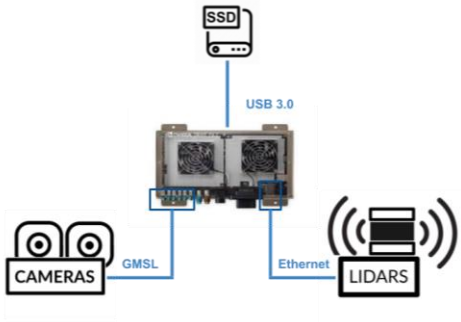


Figure 6: Basic Diagram of the hardware for the logging system.

After the hardware used for the data acquisition was chosen, the software was developed. In this case, the framework Robot Operating System (ROS) was used to manage the sensors and save the data. This framework contains a set of libraries and tools that is used in several industrial applications such as robotics and automotive.

The reasons for picking ROS as framework were several. First, ROS is open source which reduces the cost of the project and provides a big support from the community around it. Due to this, there are a huge quantity of packages available in ROS. These packages contain different libraries

(C++/Python) and drivers developed by the community such as drivers for GSMLs cameras or Ethernet LIDARs. At the same time, there are also packages that allows the recording of the data in different formats or to visualize it. In our case, we use the package Rosbag that provides a tool to save the data in a format called bag files. This reduces the time required for the creation of all these libraries and infrastructure. Figure 7 shows the integration of the ROS framework in our logging system.

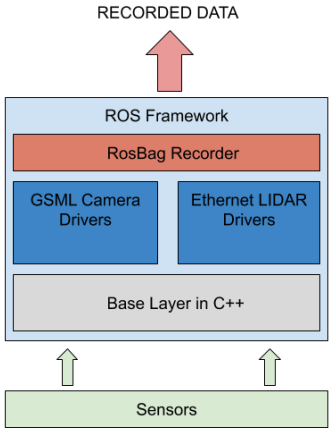


Figure 7: ROS Framework to log data.

Additionally, the way of working with ROS makes it possible to connect different processes in an easy way. The processes running in ROS are called nodes. Each node can establish communication with other nodes by using messages through topics. This way, a node can contain different subscribers or publisher of topics to send or receive information to other nodes. In our case, the drivers of the camera publish a topic containing images. At the same time, the driver of the LIDARS publishes a topic containing a point cloud. Then, the Rosbag tool get those topics and save them into a bag file in the SSD. This Rosbag file can be reproduced in another machine with the use of the Rosbag package and then the topics can be reproduced to stream the data. Figure 8 illustrates the diagrams for recording and reproducing the data.

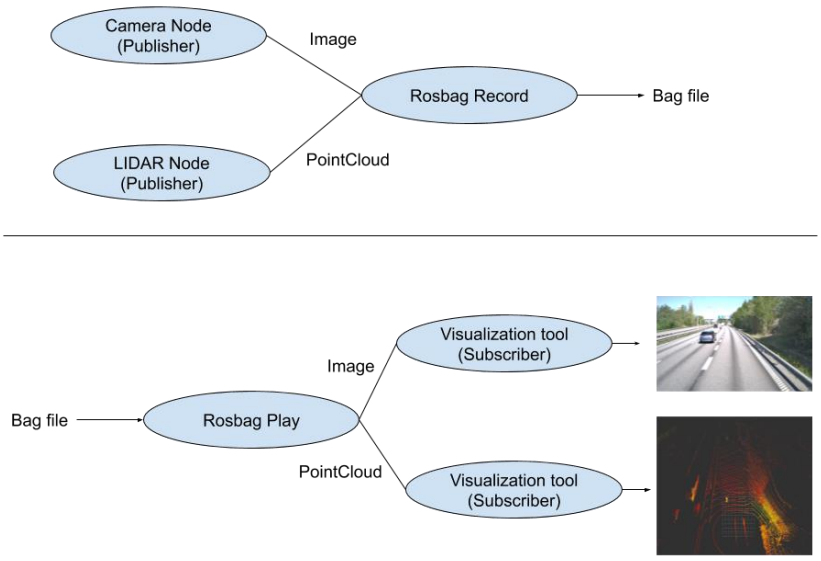


Figure 8: ROS nodes distribution. In the top part, the ROS nodes used to save the data are shown. The bottom part shows the nodes used for streaming the data from the bag file.

Once the data has been stored on the SSD, it can be deployed to a local machine in order to perform some post-processing steps. The steps to process the images are showed in Figure 9. The method described next is only done for the camera data since it was the only data annotated during the project due to lack of time.

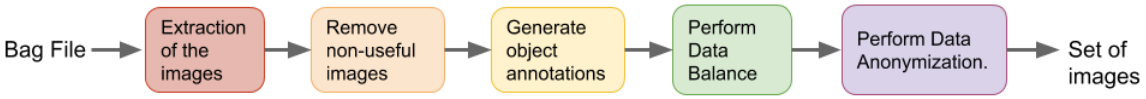


Figure 9: Pipeline for data curation.

- Extraction of the images: During this process a script extracts images from a bag file. The tool can identify the image topics present in the bag file and then extract the image in PNG format.
- Remove non-useful images: During this step images that cannot be used are discarded, e.g. images partially covered by the wiper of the truck or images that are over exposed.
- Generate object annotations: in this process a network is used to automatically annotate vehicles in the images. The generated annotations file can be used in the next step to balance the data set.
- Perform data balance: During the process the automatic annotations generated in the previous step are analysed automatically to pick those images that contain more interesting

objects in the scene. In this way, we ensure to have less frequent vehicles present in the images too. By this we can also create a good balance of big and small objects in the dataset.

- Perform data anonymization: the last processing step is the anonymization of the filtered images by the previous step. The number plates that could be recognized are blurred to follow the GDPR law. This process is done using networks that detect the plates and then blur them. Then a manual check is done to every image to ensure that no personal data is left in the dataset.

After these processing steps the data can be sent to be manually annotated so it can be used for training our DNNs. The manual annotations will ensure better quality of annotations for the object respect to the automatic generated objects from the previous step. Also annotations for lane marking, road edges and free space will be added during the manual annotation process.

Once the DNNs are trained they need to be deployed in the inference hardware on the truck in order to perform the detections while driving. The Drive PX2 was used due to its capacities for running networks in real time. This allows us to reuse the infrastructure done for logging partially for inference too. ROS was used to capture the data from the sensors in the same way as for the logging phase. The networks were running different frameworks that it was necessary to deploy in the hardware such as Tensorflow[14], Pytorch[15] and Darknet[16]. For Visualizing the output of the networks, ROS was also used since it supports different tools. In our case we use the RViz tool for visualization which allows us to represent the 2D and 3D data. The 2D representation contains images with the bounding boxes (BBs) around the object. For the 3D representation a list containing the objects in the 3D space can be displayed. Figure 10**Error! Reference source not found.** shows the complete process of inference, starting from data acquisition and finishing with the visualization.

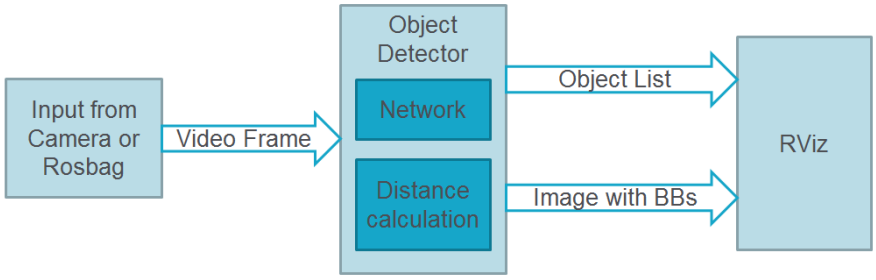


Figure 10: Object Detector Process. The visualization can be done with the images containing the bounding boxes (BBs) or represented in the 3D space with the objects from a list.

6.2.2. Explored research questions

RQ1. How do you tailor the Build-Measure-Learn model to handle data driven evolution of DL solutions?

The infrastructure developed during the project allows to apply the Build-Measure-Learn method. During the project we started with the data logging, then we analyzed that data and modified the logging infrastructure until we were satisfied with the data. Different kinds of compression format sand sensors were tested during the logging until we decided upon a final configuration. In the end, with respect to the requirements of the investigated networks, only 1 camera and 2 LIDARS were used for logging. This system allows us to log using raw format and full HD resolution for the camera and for the LIDARs a compressed package format was used that can be uncompressed again when deployed in a local machine. Despite that, the total quantity of data was very high, so the only feasible solution to transfer the data from the logging truck to the local machine was to use an external drive which then replaced the idea of using WLAN hotspots to transfer the data. When the 5G technology is more spread and developed this could be an interesting option to investigate but currently most of the OEMs use physical storage units to transfer data.

The concept of Build-Measure-Learn was applied to the complete process for our DL solution. In this after installing our sensors in the truck, we collected the data, then we deployed the networks trained with the data in the truck, we measured the performance of our network and finally we improve our solution based on what we learn from our measurements. Then the cycle was repeated to optimize the complete solution. The deployment of the DNNs was not done over the air as in the original plan due to the lack of time. Instead, the networks were deployed using hard drives like in the data collection process.

RQ2. What methods can be used during data collection to reduce the amount of log data to transfer to the backend?

During the project different techniques of compression where studied to reduce the quantity of data logged. In the end, ROS was selected for logging since the packages available for ROS makes it easy to capture and compress the data. For LIDAR, the point cloud was compressed into packages that can be uncompressed later on the local machine. Even though the camera data could be compressed, we decided to keep the raw format since the compression of the images adds some noise to the data that can affect the performance of the network. This decision made the quantity of data too large to use wireless transfer, so in the end, moving data through physical hard drives was a more adequate solution.

There was also plans to investigate methods of determining the “new” information from new logged data and only transfer the data that contained new valuable information but this turned out to be a much more complex problem than anticipated and would merit a research project of its own.

6.2.3. Deliverables and goals

The deliverables of this work packages consisted of the development of the infrastructure needed for a data driven DL solution This involves the concept, design and implementation of the hardware

architecture and the development of the software for logging, training and inference. All these components fulfil the goal *G1* of the project.

The implemented logging solution makes it possible to record from two cameras, three LIDARs and some CAN signals. But logging the sensor data from all these sensors produced too much data which would force us to apply some compression algorithm to make it feasible and this would probably affect network performance. Thus we decided to log most of the data on raw format and full resolution but with this image quality we could only log with one camera and two LIDARs without dropping an excessive quantity of frames.

As described above, the transmission of the data and the deployment of the networks was simplified to using physical hard drives due to the need of putting the focus in other fields. The introduction of GDPR, forced us to invest more time in getting legal acceptance for our data collection and also anonymizing the data we collected. In the end, the use of hard drives was the more efficient solution and the one that most of the OEMs currently use.

6.3. WP3 – Data Collection and Dataset

One of the main challenges in supervised learning [17] is the need of sufficient data to train the networks. Building a good dataset is an activity that requires a large investment of time and money. In the automotive area for example, the building of a dataset involves the need of a vehicle with the sensors required to collect the data. Additionally, after the data is acquired, the processing and annotation of this data requires a considerable amount of resources. To minimize this, there are several public datasets that can be used for training the networks. In this project, we have used some of them, like MS COCO [18], PASCAL VOC [19], KITTI [20] and CITYSCAPES [21]. These public datasets can serve as a good starting point to perform a more general training of a network, but it is often not enough for networks deployed in specific environments. For example, some vehicles or some kind of lane markers are only present in some countries. Also, these public datasets do not contain enough examples of different kinds of weather that are very prevalent in some countries, e.g. the snow in Sweden. It is also often the case that the public datasets have been created with sensor mounted on passenger cars. These sensor positions often differ from the sensor positions used for trucks, i.e. some truck sensors are placed higher to get a better view. The need to have a more specialized dataset of the Swedish highways with high sensor positions is the motivation of this work package.

6.3.1. Results

Environment selection when collecting data is an important factor since the performance of the neural network is highly dependent on it. Although there is a lot of public data available for training, to get a good performance of a neural network it should be trained on data that is similar to what it is expected to encounter in its operational design domain. This means that neural networks designed for specific conditions should be trained with data from these conditions.

The Perceptron project was focused on developing DNNs for highways in Sweden. Due to this, some of the national highways of Sweden were selected as the locations for the data collection. More specifically, the highways that pass through Göteborg were selected since it is the base for

the development of this project. These roads contain some scenarios that are only occurring in Sweden like 2+1 roads, and specific Swedish vehicles.

Figure 11 shows the stretch of highway selected for the data collection. More specifically, the roads are:

- E6, Göteborg - Svinesund: This is the main highway connecting Göteborg to Oslo along the coast. The data has been collected on 180 km of this road up to Svinesund.
- E45, Göteborg - Trollhättan: the road between these cities is about 78 km and it was sometimes selected as an alternative route to the previous section of the E6 to increase the variability of the data.
- E6, Göteborg - Malmö: this is the main highway going to the south from Göteborg. The data has been collected on 280 km of this road up to Malmö.
- E4, Göteborg - Linköping: this highway goes to Stockholm through Linköping and the data has been collected on 280 km.
- E20, Göteborg - Örebro: This is the alterative highway going to Stockholm. The data has been collected on 290 km of this road up to Örebro.

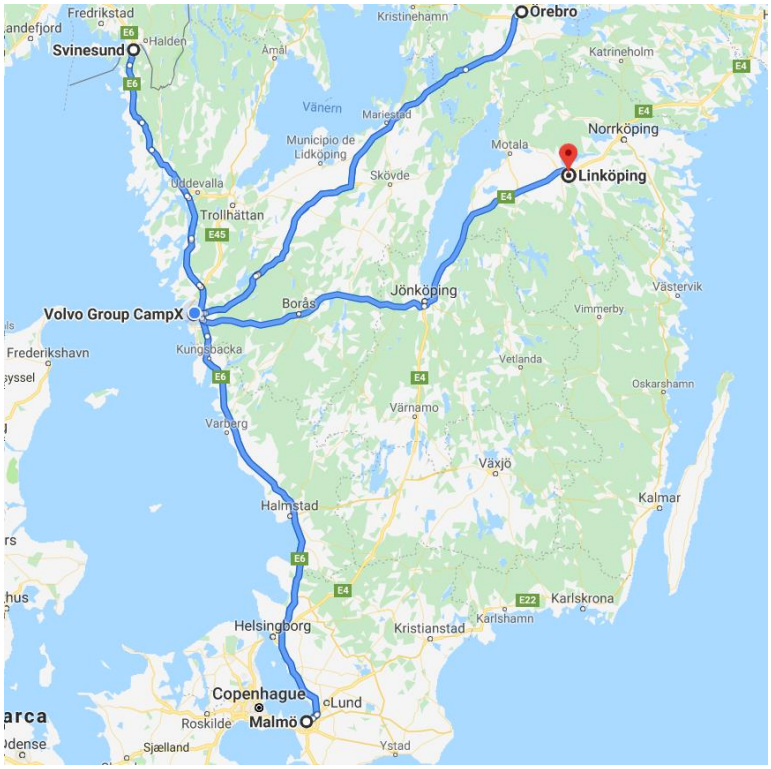


Figure 11: Map containing the highway sections for the data collection.

The different logging sessions have also been placed in time in such way to allow us to capture the different seasons of the year. Because of this, the dataset contains all kinds of weather conditions like, snow, rain and fog which can be used to make the networks more robust in handling these weather conditions. Figure 12 contains examples of images captured in different seasons.

Figure 12: Extracted images from the collected data from different seasons.

After the data was collected and processed, the data was sent to be annotated and the result of it was the Volvo Highway Dataset. This dataset is composed by 28778 images. During the project, 75% of the dataset was used for training and validation and 20% as a test dataset. The dataset contains the images in .PNG format and an annotation file for each image. The annotation file contains the annotations in the format GeoJSON [22], a special format to organize geometries in a JSON format. The annotations are separated into four categories: road objects, lane markers, road edges and free space.

The road object annotations mainly indicate the presence of vehicles in the road. The dataset contains 98592 annotations for road objects. These annotations are divided into 7 classes: car, van, truck, motorcycle, bus, other vehicle and unidentified road object like for example the presence of an animal in the scene. Both objects in the ego road and in the oncoming road are annotated.

In addition to the classes, each annotation contains the bounding polygon for each class so it can be used for semantic segmentation or object detection problems in addition to the classification. Figure 13 contains an example of images with a mask and with bounding boxes taken from the annotations. Figure 14 contains the class distribution for the road annotated objects.

Figure 13: Annotation for Object Detection. The left image contains the mask for object segmentation and the right image contains the bounding boxes for the objects and the object classification.

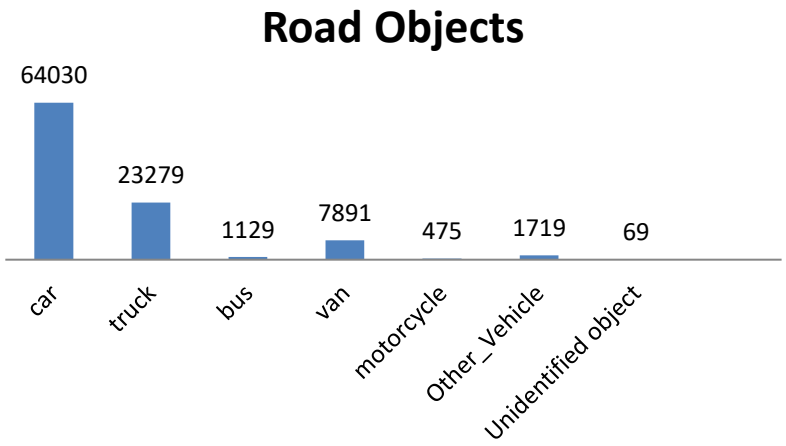


Figure 14: Class distribution for road objects.

The lane marking annotations are used to identify the presence of different lanes in the road. The dataset contains 99856 annotations for lane markers. The annotation for each lane contains several segments that are classified as either solid, dashed or implied. This way a lane can contain several segments with different classes, e.g. a lane changing from solid to dashed. The implied segment type is a very rare case which means that there should be a lane marker, but in practice no physical marker has been drawn on the road, e.g. an exit of the highway where the lane is cut. Figure 15 shows an example of lane marking annotations. Figure 16 contains the class distribution for the lane markings.

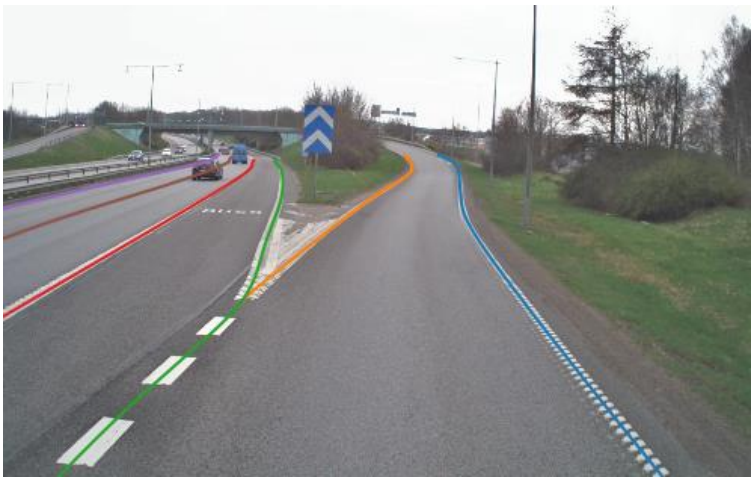


Figure 15: Image from lane marking annotations with some example or dashed and solid segment lanes.

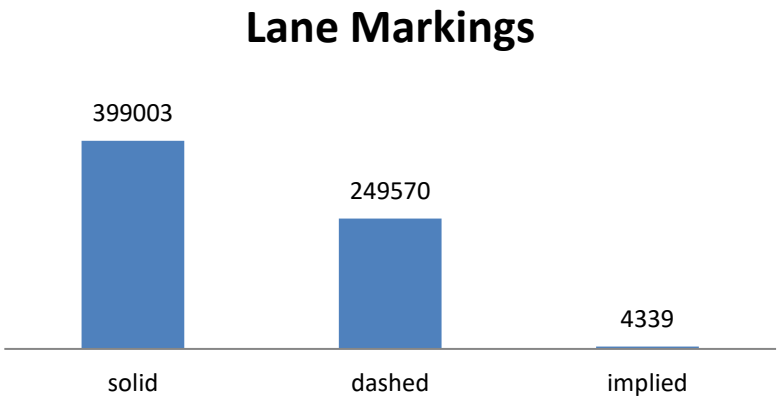


Figure 16: Class distribution for the segments of the different lane markings. A lane can contains several segments.

The road edges annotations are used to detect the limits of the road. The dataset contains 126997 annotation for road edges in the ego road. Again, the edges are formed by several segments. These segments are divided into three classes: guard rail, asphalt and Jersey barrier. There is an extra class of segment called connect, that was used to compute the free space, so it is not relevant for the road edge detection. This way a road edge can contain several segments with different classes. Figure 17 shows an example of the road edges annotations. Figure 18 shows the class distribution of the dataset for the segments that forms the road edges.

Figure 17: Annotation for road edges with asphalt segments.

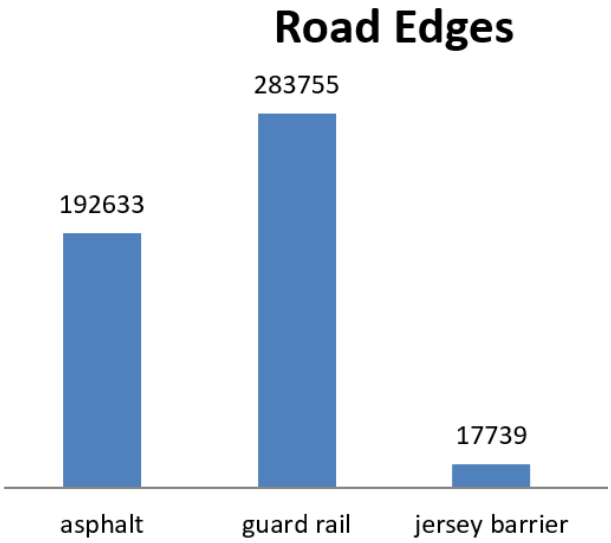


Figure 18: Class distribution for segments of the annotated road edges. An edge can contain several segments.

The free space annotations contain the part of the ego road that is drivable space. Each image of the dataset contains an annotation for the free space which is indicated by a polygon enclosing the drivable space. The number of polygons in the dataset is 36577. Figure 19 shows an example of a free space annotation.

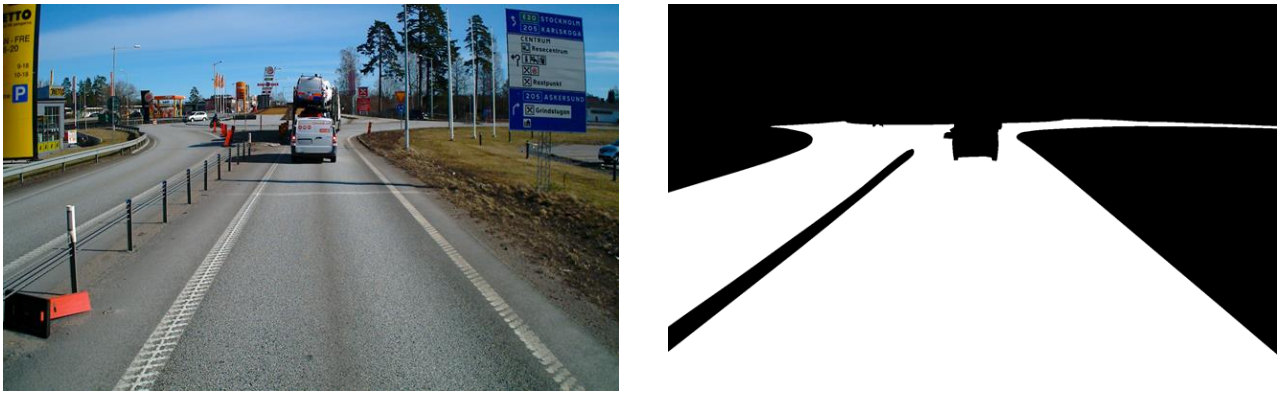


Figure 19: Free Space annotation example.

6.3.2. Explored research questions

For this work package we don't cover any of the research questions directly, but we cover some of them indirectly.

RQ1. How do you tailor the Build-Measure-Learn model to handle data driven evolution of DL solutions?

Nowadays, it is clear that the creation of a dataset is a very important step for a data driven DL solution. Part of this solution consist of logging data which then is annotated and used to retrain the network to have more accurate detections. In fact, the step of logging and annotating is one of the most resource consuming steps in the whole process but is also key to have a successful solution.

RQ4. How do you improve DNN based perception applications beyond state of the art?

Optimizing and adapting the dataset to a specific use case is also a way to surpass state-of-the-art methods. It has been proven during the development of the project that also changes in the dataset can help to increase the performance of your DL solution. The use of our dataset for training, which contains Swedish roads, some vehicles just presents here or weather conditions that are more common here, increase the accuracy of the networks when testing the algorithms in these roads. In this way, the networks learns how to identify specific features that are more common in this country and do not appears in a public dataset. A consistent dataset adapted to the needs of your use cases can be crucial.

6.3.3. Deliverables and goals

The deliverable of this project is a created dataset. The Volvo Highway Dataset contains 29000 labeled images. The annotations include road objects, lane markings, road edges and free space. These annotations are divided in 7 classes of vehicles, 3 classes of lanes markers and 4 classes of road edges. In addition, the dataset contains polygons for the objects that can be used for semantic segmentation and object localization. Although this deliverable is not related directly with any of the goals of the project, it contributed to reach all of them indirectly.

6.4. WP4 – Object Detection

Object Detection is one of the most important pillars of current Autonomous Driving solutions and consists of the detection and classification of the relevant objects around the ego vehicle. Not so long time ago, using Computer Vision methods built on handcrafted measures together with trainable shallow architectures for image or point cloud-based detection tasks, was common practice. As of today, it is clear that DL based approaches generalize considerably better than traditional Computer Vision methods. Figure 20 illustrates an example of the gap between hand coded CV and DL approaches for Object Detection from the Image-Net competition Object Detection task results [26].

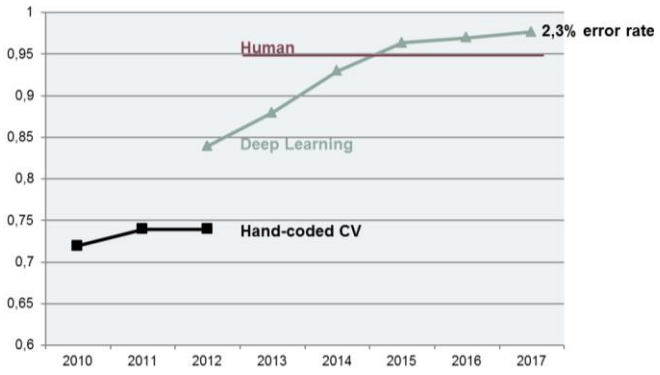


Figure 20: Deep Learning algorithms have outranked traditional computer vision (CV) for object detection for the last 6 years.

The common approach for Object Detection is using fully Convolutional Neural Networks either by using Images or Point Clouds which are generally built on base classification networks. The main focus in the area has been on increasing the accuracy while keeping computational complexity down. For this package, different techniques for object detection were investigated along the project.

6.4.1. Results

The goal of object detection algorithms is to determine and localize the presence of predefined object categories from raw or processed image sensory data. The first step in developing the algorithms was to identify the interesting object classes. Since the objective of the project was to develop a perception system for highways, we identified “Car, Van, Motorcycle, Bus, Truck” as the core classes. Then we added two extra classes, “Other Vehicles” and “Unidentified Road Object” to define objects out of the core classes.

Another important point was the placement of the sensors in the Vehicle. The data was collected using sensors positioned in different parts of the cab of the Vehicle, Figure 4. Having sensors placed on top of the cab allows us to detect objects that are further away, in contrast the sensors positioned on the bottom of the trucks allows for a better detection performance for close objects.

Sensors positioned on the sides allow the perception system to detect side objects, and sensors positioned on the trailer allow for creating a surround view of the environment. Controlling the dynamics of large vehicles, like trucks in our case, requires a long-range perception system in the range of 200 m. Long range perception would enable efficient trajectory planning algorithms.

Camera based Object Detection was the main focus during the project. Different state-of-the-art algorithms were identified and then adapted to our needs of training and inference. Among the architectures investigated in the project we can emphasize YOLOV3 [27], Mask-RCNN [28], Faster-RCNN [29] and SSD [30]. These architectures can be classified into two different categories:

- One stage detectors: YOLOV3 and SSD.
- Two stage detectors: Mask-RCNN, Faster-RCNN

The difference between these two categories is in the tradeoff between inference speed and detection accuracy. Traditionally, the detection happens in two stages, first, the model proposes groups of region of interests, ROIs, by a regional proposal network. Then, a classifier processes the region candidates. One stage-detectors skip the first step, running directly detection over a dense sampling of possible locations. This increases the speed of inference of these models at the expense of decreased accuracy.

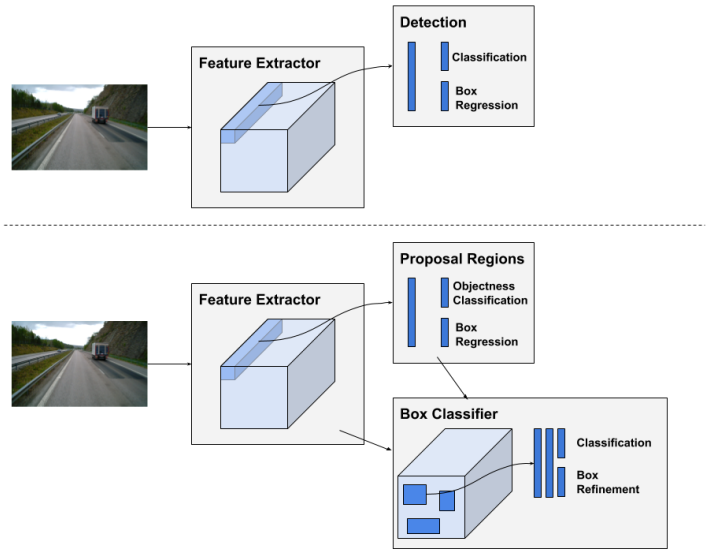


Figure 21: The image at the top shows a one-stage detector and the image below shows a two-stage detector. The one-stage detector performs the detection directly over sampling of locations. The two-stage detector uses a region proposal network that evaluates the objectness of region of interest. If the objectness is over a threshold then the selected region is cropped and evaluated by another network that determines the classification of the object and performs a refinement of the bounding box.

Also it is important to take the depth of the network into account. Deeper feature extractors are slower and have higher detection accuracy. In our experiments we have considered feature extractors based on the Resnet architecture [31]. We experimented with Resnets of different depth [50,100,152]. Increasing the depth of the DNN clearly improved the detection results, but heavily affected the speed of training and inference. The high performance Resnet152 feature extractor was used in a tool developed for automatic annotation of the collected data and automatic data anonymization.

Another key factor in our experiment was image resolution. Higher resolution images increase the memory usage and inference time substantially but has the potential for better accuracy of the networks.

For training the neural networks we have used several public datasets, like Pascal VOC, COCO, Cityscapes and Berkley BDD (Berkley Deep Drive). Since the datasets contain many classes that we are not interested in, a pre-processing step has been performed to filter data that did not contain the object classes that we were interested in. We have made use of data augmentation by applying different algorithms of augmentation such as flipping, Gaussian noise, cropping part of the image, injection of objects in the image, rotation, translation etc. The distribution of the training data is very important, i.e. that it is a balanced dataset. With data augmentation the balance can be

improved and in our experiments we can see that the robustness of the detection algorithms increases by 1-1.5%.

Table 2 shows a performance comparison between the main architectures investigated during the project using the public dataset COCO. The comparison shows the inference speed in our selected inference hardware, the Drive PX 2. It also shows the accuracy, using the mean Average Precision, mAP, a popular metric in measuring the accuracy of object detectors [32].

Object Detector Architecture	**Speed of execution (DrivePx2)**	**Accuracy mAP(COCO)**	**Memory Consumption**
YoloV2	20 Hz	20.4	1 GB
YoloV3	12 Hz	27.3	2.4 GB
Tiny YoloV2	30 Hz	15.2	0.5 GB
Faster RCNN(Resnet50)	4 Hz	37.3	3.5 GB
Mask RCNN(Resnet50)	3 Hz	38.1	4 GB
SSD (MobileNet)	10 Hz	32.4	1 GB

Table 2: Camera based object detection performance chart using COCO benchmark.

After experimenting with different kinds of object detectors, we decided to focus more on YOLOv3 as our main real time detector due to its better trade-off between speed and accuracy. We performed more experimentation with different kinds of architectures for YOLOv3 that are represented in Table 3. In addition to the classical YOLOv3 architecture, two more variations of these models were evaluated, YOLOv3-Tiny and YOLOv3-SPP. YOLOv3-Tiny is a smaller version of the regular model with only CNN layers, which is most optimized for constrained hardware environments. This version increases the fps by reducing accuracy of the detections. YOLOv3-SPP is a modification of the normal architecture that includes a spatial pyramid pooling layer.

The procedure followed to evaluate this method was:

1. Train the three Yolov3 versions (starting with weights from ImageNet) with a mix of public datasets: VOC, COCO and CS.
2. Train the three Yolov3 versions (starting with weights from ImageNet) with our own dataset.
3. Compare the performance of the resulting six networks.

The mAp for the networks trained with the public data was calculated using only 4 classes (car, truck bus and motorcycle) since the other classes were not present in the dataset. The results of these experiments are shown in Table 3. The results shows that the accuracy of the networks (tested on the test part of our own dataset) trained on our own dataset is almost double of the networks trained on the public datasets. This result is the expected, taking into account the differences between our own collected data and the public datasets, e.g., different location of camera, objects annotated further away, etc...¨

Arch	Classes	Accuracy mAP	
		Public Dataset (COCO + VOC + CS)	Volvo Dataset
Yolo V3	All	24	41.2
	Car	35.5	56.8
	Van	-	47.6
	Truck	25.4	70.4
	Motorcycle	16.6	29.1
	Bus	18.6	49.2
	Other Vehicle	-	35.3
	Unidentified Object	-	0
Tiny Yolo V3	All	16.83	27.3
	Car	21.7	32
	Van	-	31.5
	Truck	18.1	52.2
	Motorcycle	10.9	14.8
	Bus	16.6	44.1
	Other Vehicle	-	16.4
	Unidentified Object	-	0
Yolo V3 SPP	All	23.975	37.4
	Car	35.9	53
	Van	-	39.9
	Truck	26.4	66.6
	Motorcycle	12.8	28.5
	Bus	20.8	47.6
	Other Vehicle	0	26.1
	Unidentified Object	0	0

Table 3: Accuracy of YoloV3 models using the test data of the Volvo Dataset.

In general, it can be found that the model trained with the Volvo Dataset is able to detect vehicles that are further away which is expected since the Volvo Dataset contains more instances of vehicle annotations at a distance. This can be observed in the Figure 22.

Figure 22: Comparison of YoloV3 detector, trained with the Volvo Dataset and with public data. The left image shows the detector trained with the Volvo dataset. The right image shows the detector trained with public dataset. The second detector is not able to detect the two vehicles at the end of the ego road.

In addition to camera-based Object Detection some research was done in the field of Semantic Segmentation with LIDAR data. In this case, the input to the network is a set of point clouds captured by multiple LIDAR sensors and the output is those cloud points with an indication for each point if it belongs to the drivable free-space or other vehicles.

To segment free-space (i.e. drivable area) and obstacles (e.g. vehicles, pedestrians, etc.) in LIDAR point cloud data, we followed two independent strategies. In the very first strategy, we developed a pipeline shown in Figure 23 that consists of two different customized state-of-the-art deep neural networks [34], [35]. Given a set of 3D data points captured from multiple LIDAR sensors, the first network creates a top-view transformation of the data as illustrated in the top part in Figure 23. Such a top-view 2D representation is employed to detect drivable free-space, which is then back-projected to 3D space. The second network instead computes the front-view of the point cloud as depicted in the bottom row in Figure 23. Obstacles detected in the front-view are again back-projected to 3D space to be further fused with the free-space detected in the previous step.

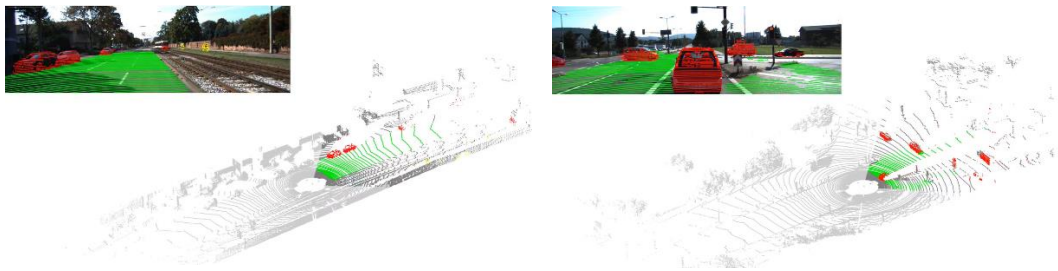


Figure 23: Two Sample Images from the KITTI dataset for a Qualitative Evaluation. Green and red segments represent detected free-space and obstacles respectively. Grey colour is used for the unlabelled points.

We have quantitatively evaluated the performance of both networks on a publicly available KITTI dataset. Figure 24 shows two sample images from the KITTI dataset for a qualitative evaluation. On the top-left corners, the original camera images with projected semantic segments are shown for the sake of clarity. Note that camera images are not processed in the pipeline. The bottom row in Figure 24 depicts detected free-space (shown in green) and vehicles (depicted in red) segments in 3D space.

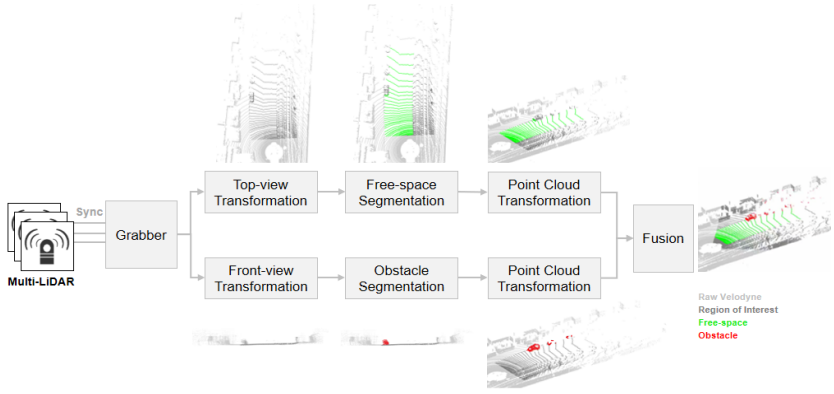


Figure 24: Visual Processing Pipeline for LiDAR-based Semantic Segmentation.

As the second strategy, we developed a new neural network that takes raw point cloud and returns free-space and vehicles at the same time. This new network is based-on the U-Net [36], but involves less layers and more skip connections. We further applied different regularization methods such as batch normalization, dropout and data augmentation to increase the network performance. Figure 25 below shows the basic structure of the network. The input of the network is the top-view image of the point cloud. Each channel of the 4D top-view image represents unique statistical information such as mean and max values of the height, number of pixels and reflectivity.

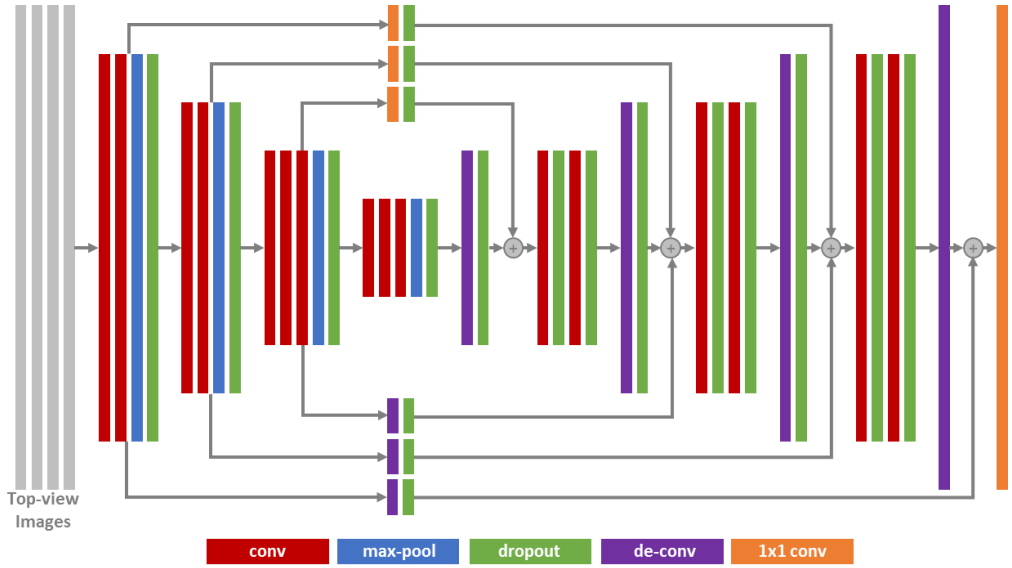


Figure 25: Network Architecture

6.4.2. Explored research questions

RQ4. How do you improve DNN based perception applications beyond state of the art?

In the Perceptron project, different methods to increase the accuracy over the state-of-the-art methods were applied. It was shown that by made modifications in the dataset, like dataset augmentation, the accurate of the network can be increased. By doing data augmentation, like transformation on images in the dataset, the quantity of images used in the dataset can be increased and this could help to boost the performance of the network. During the project, some techniques of optimization like, pruning and quantization were tested too on small networks, this applying of these techniques is still a subject of research on more complex networks but it will be a crucial step to increase the speed during the training and the inference phase.

RQ5. How important is it to train and infer using the same sensors and geometry?

During the development of the project the sensors were placed in a stationary position and they were not relocated due to lack time. Despite of this, it was found that the networks that were trained with our dataset had better performance than the ones trained with public datasets. These public

datasets often are recorded using normal passenger cars where the sensors are located at a lower distance from the ground. Therefore, it was demonstrated that having a training dataset with the same sensor configuration as the one intended for inference can boost the performance of the networks. The same reasoning can also be applied to the sensor type, e.g. using the same camera lens for the training dataset and during inference.

6.4.3. Deliverables and Goals

The deliverable of this package consists of the development of an Object Detector Network based on state-of-the-art methods. To do that, the state-of-the-art networks were identified and numerous experiments were carried out (described in previous section).

Different inference and training frameworks were also used. For Yolo based architectures we used the Darknet framework, which is an open source library for training and inference of deep neural networks. The Darknet platform is written in C and the CUDA programming language. For the final pipeline, a Docker container was created containing PytTorch, then a wrapper of PyTorch over Darknet was used to perform training and validation.

For the Mask-RCNN and Faster-RCNN architectures we used Caffe2 and the Detectron framework for both training and inference. Caffe2 is also open sourced and it is written in C++ and CUDA, while Detectron is written in Python. Later in the project, Detectron2 was released, based on PyTorch and the infrastructure was updated to use this platform.

For SSD based architectures we used the Tensorflow framework. Tensorflow is an open source framework developed by Google. Tensorflow is written in C++, CUDA and Python.

In the end, YoloV3 was selected as our real-time object detector. This network was deployed on the inference hardware in the truck. The execution time of the network was around 80 ms on the inference hardware which is in the top 10 DNNs object detector with the COCO benchmark [33].

6.5. WP 5 – Free Space Detection

Knowing what is surrounding a moving vehicle is a crucial part of enabling autonomous solutions. As described in Chapter 5.3, the autonomous agent is equipped with several cameras and LiDAR's to become aware of its surroundings. Throughout related research work, cameras have been the most common sensor to use for automated solutions. Cameras is a part of most autonomous solutions, as well as public datasets since it is a relatively cheap component, and has been part of research for many years.

For this project, Free Space (fs) is defined on a pixel-level, to enable a semantic segmentation network to classify each pixel of an input image. One strength of convolutions in neural networks (NN) is the spatial information of pixels neighbours. This fact is one key ingredient in these NN's, which are amplified even more in network setups with an *autoencoder* architecture. An *autoencoder* refers to a network with a distinct bottleneck in the centre of the network. Prior to the bottleneck, the input proceeds through an encoding where the spatial structure has to be learnt, which will be decoded after the bottleneck in a so-called decoder. As we shall soon see, three NN architectures of this type with different setups are analysed.

As most public research has been focusing on individual transportation, all found public available datasets consist of camera imagery taken from cars. This project focuses on trucks, thus gaining a perspective advantage. This project investigates how the performance changes when utilizing these public datasets with large variations, to a newly created dataset gathered from a truck. In addition, we compare how the performance change if the networks are first trained on public data, then fine-tuned on *private* (Private referring to the newly gathered dataset by Volvo) and vice versa.

6.5.1. Results

For the free space detection networks, we developed a three-stage training process. The process consisted of first training the model on a public generic merged dataset consisting of several large autonomous driving datasets, as well as some including detailed semantic segmentation tasks. In this task, the network is trained on a plethora of different classes, to gain generic feature extraction, rather than just detection free space. The argumentation for this initial step is taken from the idea from the success of transfer learning, where pre-training on a diverse task gives general features which in turn can be used as a head start for the specific task.
The second step consist of training only for free space. This is done either on the *public* autonomous driving dataset, or the *private* driving dataset. The third and last step consist of small fine-tuning for the desired dataset that is to be used. In Table 2 the different training results can be seen for the three chosen networks SegNet, ENet and SqueezeNet. All networks consist of autoencoder architectures with different width and depth. After the training the network were evaluated using the test data set from our *private* dataset and from the *public* dataset.

	Metrics			
	Train	**Pr. mIoU**	**Pu. mIoU**	**Est. FPS**
SegNet	Public	76.26 %	75.18 %	64.1
	Private	**90.00 %**	49.00 %	62.1
ENet	Public	78.87 %	**91.69 %**	250
	Private	89.69 %	52.99 %	243.9
SqueezeNet	Public	78.85 %	90.50 %	277.8
	Private	89.75 %	54.94 %	**285.7**

Table 4: Free Space mean Intersection of Union for the training setups for the three different architectures. The first column indicates the dataset used for the trained, the second column shows the evaluation results using the private test dataset. The third column shows the evaluation results using the public dataset. The last column shows the estimated frames per seconds for each network.

Most noteworthy for most of the training runs is the large difference in public and private mIoU, depending on which training set has been used in the second phase. This error is larger when trained on the private dataset, which easily is explained due to the fact that the public dataset consisting of large varieties in the data samples. This variety is less in the private dataset, as it has been gathered in Sweden over the course of one year. The public dataset has been gathered over several years, in several large cities around Europe and USA.
For all models, a reasonable fast FPS has been achieved. The estimated FPS have been computed using Pytorch as an average of one batch of samples, thus skewing the results slightly to the positive edge, compared to when running a single sample.

6.5.2. Explored research questions

This point illustrates how this work package explores the research questions previously mentioned:

RQ4. How do you improve DNN based perception applications beyond state of the art?
The work package has utilized state-of-the-art networks and build upon the training procedure of these. In particular, a three-step-phase of training were constructed with the goal of utilizing transfer learning from large public dataset, to show the power of using pre-training for the specific task at hand. In addition, we show that free space detection is possible for trucks as well as for cars.

RQ5. How important is it to train and infer using the same sensors and geometry?
As explained in RI4, the transfer learning process allows for a larger jump in geometry. We show that changes of camera resolutions and color saturation has some impact, but can be mitigated by longer training, as well as allowing for pre-training on a broader dataset.

6.5.3. Deliverables and Goals

This work package focused on creating and documenting neural networks for free space detection. The work package delivers several well-performing models, where the lightest model *SqueezeNet* provided the best results on the private Volvo data.
The models all operate on a very fast basis (above 20 Hz), thus work on real-time systems.

The networks are complemented with a descriptive report, explaining training process, results and different investigations. In particular, the report explains how robustness of all the networks has been tested. Especially how much disturbance each model can withhold without providing unusable results.

6.6. WP6 – Lane Detection

An initial step for any high-level visual analysis of observed road scenes from cameras mounted on vehicles is to find which regions of the input images are depicting the road.
Upon successful segmentation of road regions in the images, we need to figure out how many lanes does the road have and where each of these lanes located in the image coordinate system, i.e. lane detection.
Various machine vision approaches have been used to solve this problem.
Starting from hand-crafted features extracted directly from the input images, followed by robustness improving techniques against severe variations in lighting, shadowing, occlusion, input image quality, etc. which are often present in realistic evaluation scenarios, till the very recent methods adopting deep convolutional networks which have shown to be state-of-the-art performers generally in various visual analysis problems.
In the upcoming section, we open a short discussion of the common approaches to these problems and similar domains.

This is followed by a detailed description of the proposed method and finally the implementation details and experimental results are provided, where the proposed method is compared to a baseline approach. The proposed method is designed to provide a good combination of performance and inference time.

6.6.1. Results

The major outcome of WP6 was a complete pipeline of image based lane detection. A method that, given an input image, outputs a set of curves describing the position of lane markings in the image was developed and evaluated. The method developed consists of three steps: lane point detection, lane point clustering and lane curve estimation.

Lane point detection

Lane point detection is done using a CNN with similar structure to the Single Shot Detector networks used for object detection. The network structure is based on the Yolov3 network [27], which originally detects object on three scales (one for each detection head visualized in

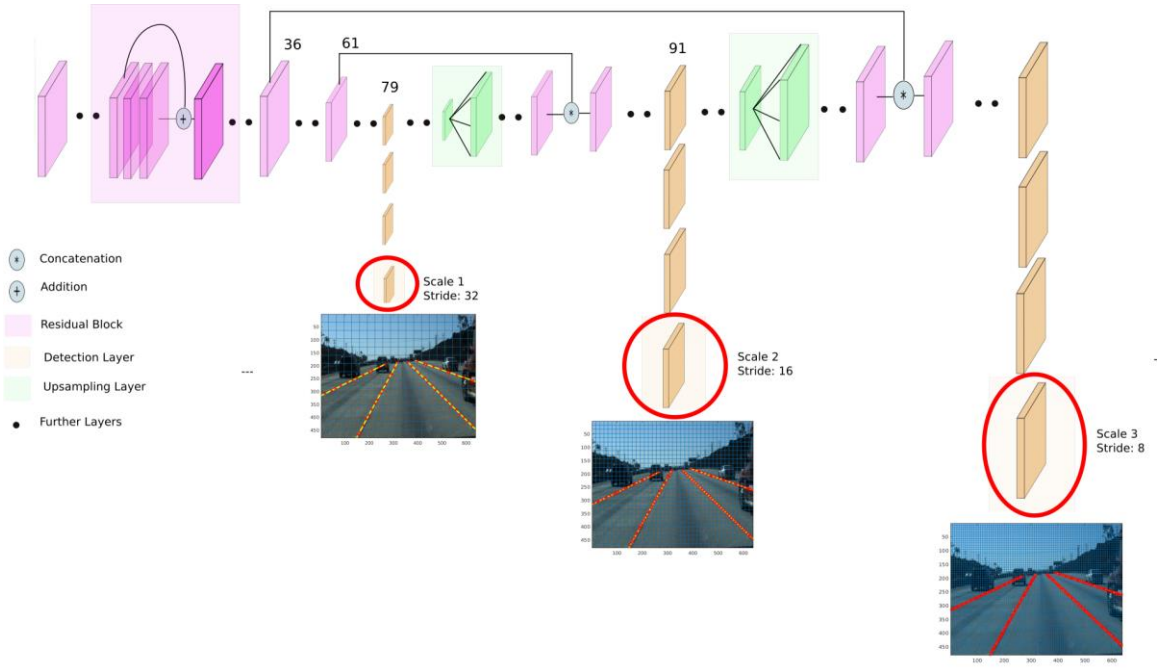


Figure 26, 32x32 pixels, 16x16 pixels, 8x8 pixels. This enables the option of sacrificing performance for faster inference by using a coarser scale.

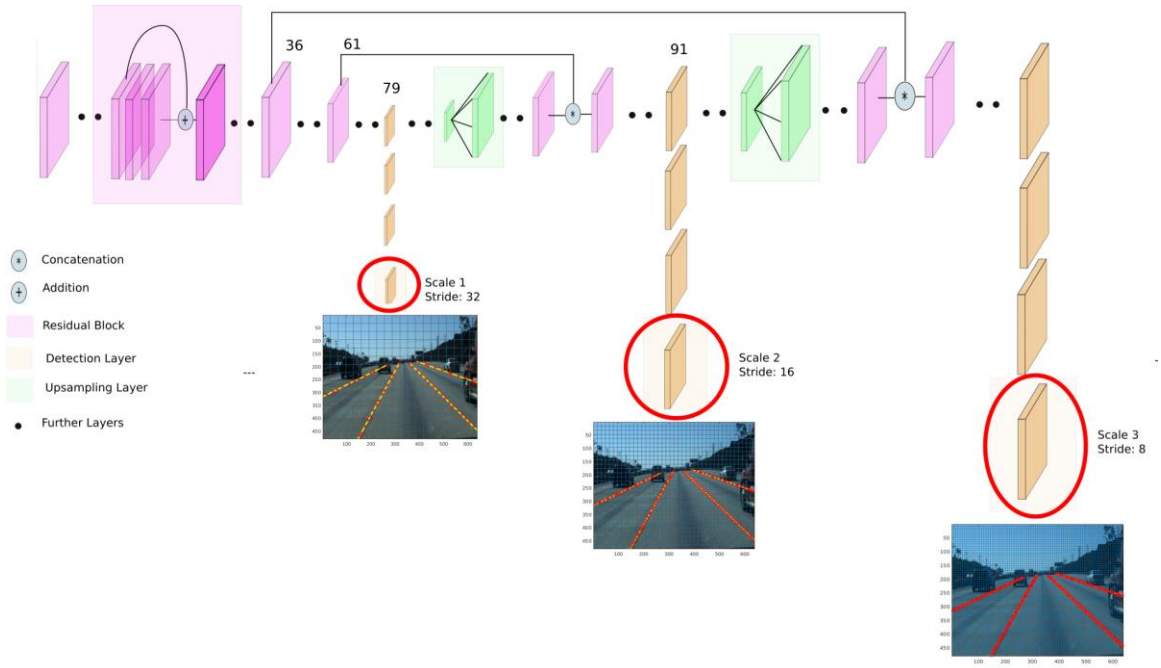


Figure 26: Network structure based on the Yolov3 network [8]. Red circles maps changes from the original, here the number of feature maps out are changed to 2+C where C is the number of classes.

To customize the network structure for lane point detection we change the number of output feature maps for the final layers of each of these detection heads. The number of features needed in our case is 2 + C where the last C layers are used to classify each box. A box is assigned a class from 1 to C if there is a lane inside it and 0 if there is no lane. For boxes with class 1 the first two feature maps are trained to predict the relative position, (dx, dy) of the closest point on the lane that is inside the box. See Figure 27 for a visual explanation.

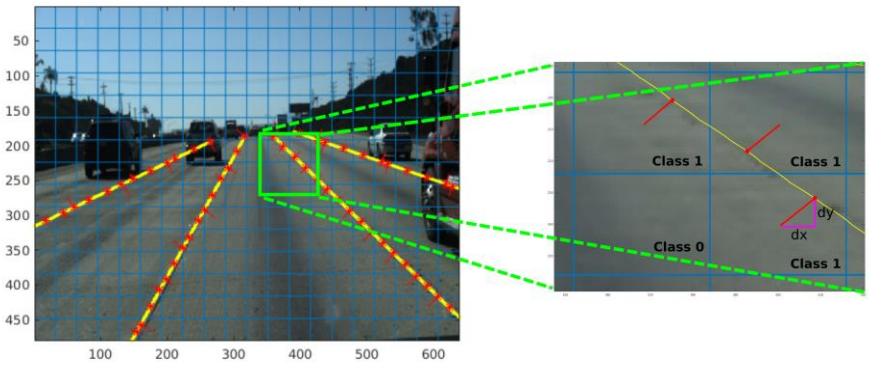


Figure 27: Data and network task visualization. The network outputs a feature map for each scale classifying each box as well as predicting dx and dy for boxes where a lane is present.

During training a cross-entropy loss is applied for the C output maps that predict the class of each box. In addition, a minimum squared error loss is applied for the two output maps predicting the displacement (dx, dy).

Lane point clustering

The lane points detected by the network are then clustered using a bottom up image scanning approach. The input to the algorithm is the image coordinates for all detected lane points, these are easily extracted from the output of the network. Given a set of detected lane points the cluster algorithm returns a set of indices dividing the lane points into groups depicting the same lane marking.

Lane curve estimation

As a final step, for each cluster of points that is large enough (the number of points in the cluster is larger than 5) the entire lane can be retrieved using cubic spline interpolation.

Experiments

The main results are for a network trained on the Perceptron dataset collected during the project. When applicable, we compare the results of our method with the naive approach of segmenting lane markings. In addition, we compare a network trained on the Perceptron dataset with a network trained on the TuSimple dataset.

For the segmentation approach we evaluate by simply calculating the intersection over union (IoU) for each class and then taking the mean across all images. For the lane detection network, we look at how well the lane points are classified. The measures reported are accuracy, precision, recall and IoU. In addition, we report the mean error of the predicted positions of the lane points. For result on the Perceptron Test set see Table 5.

	Point Precision			Point Recall			Point Accuracy		
class \scale	8x8	16x16	32x32	8x8	16x16	32x32	8x8	16x16	32x32
lane marking solid	0.866	0.884	0.895	0.858	0.884	0.895	0.995	0.992	0.988
lane marking dashed	0.835	0.877	0.900	0.802	0.851	0.876	0.995	0.993	0.991
lane marking implied	0.000	0.000	0.000	0.000	0.000	0.000	1.000	1.000	0.999

	Point Precision			Point Recall			Point Accuracy		
class \scale	8x8	16x16	32x32	8x8	16x16	32x32	8x8	16x16	32x32
lane marking solid	0.866	0.884	0.895	0.858	0.884	0.895	0.995	0.992	0.988
lane marking dashed	0.835	0.877	0.900	0.802	0.851	0.876	0.995	0.993	0.991
lane marking implied	0.000	0.000	0.000	0.000	0.000	0.000	1.000	1.000	0.999

Table 5: Results on the Perceptron Test set for the point detection at different output scales. Note that extremely few images contain the class"lane marking implied".

To evaluate how well the complete pipeline works we investigate the final lane predictions in the following way. For each lane in the annotation, create a binary map for the pixels that the lane touches. Dilate this map with 5 pixels both horizontally and vertically. The same thing is done for the predicted lane segments. Afterwards the annotated lane maps and the predicted lane maps are

matched. If there is a predicted lane map that overlaps enough with an annotated lane map (IoU > 0.5), it counts as a true positive. If an annotated lane map does not overlap with any predicted lane map, it counts as a false negative and finally if a predicted lane has no overlap, it counts as a false positive. In this way we can calculate precision and recall values for individual lane segments. For result on the Perceptron Test set see Table 6.

measure \scale	8x8	16x16	32x32
position error [pixels]	0.779	1.100	1.686
lane precision	0.554	0.572	0.567
lane recall	0.563	0.557	0.530

Table 6: Results on the Perceptron Test set for the lane detection at different output scales.

Summary

The idea of using a "Single Shot Detector"-like network for lane point detection seems promising. It provides good results without having to deal with large networks. However, the clustering method used at the moment has several disadvantages. For situations with horizontal lane markings in the image the clustering will fail. In addition, it has trouble with lanes that curve a lot in the image.

Developing a more principled clustering approach that could deal with these cases is left for future work.

Figure 28: Results on test images of the Perceptron dataset. Green lines mark solid lane marking while blue lines mark dashed lane marking. The last row is specifically chosen as a failure case.

Additional Research in the Work Package

Data collection and annotation have been important parts of this project, but at the start of the project, this data was not available. Further, it is infeasible to collect and annotate data under all possible conditions (for instance, day-night, summer-winter, rain etc…) and therefore we started looking at ways to use publicly available image which was not annotated, and apply machine learning methods that rely on self-supervision for training. This has turned out to be a very fruitful research direction and it has resulted in three different publications as listed below. Funding from the Perceptron project is acknowledged in all of the publications.

Benchmarking 6DOF Outdoor Visual Localization in Changing Conditions *by T. Sattler, F. Kahl, et. al.\ published at CVPR 2018.*

Visual localization enables autonomous vehicles to navigate in their surroundings and augmented reality applications to link virtual to real worlds. Practical visual localization approaches need to

be robust to a wide variety of viewing condition, including day-night changes, as well as weather and seasonal variations, while providing highly accurate 6 degree-of-freedom (6DOF) camera pose estimates. In this paper, we introduce the first benchmark datasets specifically designed for analysing the impact of such factors on visual localization. Using carefully created ground truth poses for query images taken under a wide variety of conditions, we evaluate the impact of various factors on 6DOF camera pose estimation accuracy through extensive experiments with state-of-the-art localization approaches. Based on our results, we draw conclusions about the difficulty of different conditions, showing that long-term localization is far from solved, and propose promising avenues for future work, including sequence-based localization approaches and the need for better local features. Our benchmark is available at visuallocalization.net.

A Cross-Season Correspondence Dataset for Robust Semantic Segmentation *by M. Larsson, F. Kahl, et. al. , published at CVPR 2019.*

In this paper, we present a method to utilize 2D-2D point matches between images taken during different image conditions to train a convolutional neural network for semantic segmentation. Enforcing label consistency across the matches makes the final segmentation algorithm robust to seasonal changes. We describe how these 2D-2D matches can be generated with little human interaction by geometrically matching points from 3D models built from images. Two cross-season correspondence datasets are created providing 2D-2D matches across seasonal changes as well as from day to night. The datasets are made publicly available to facilitate further research. We show that adding the correspondences as extra supervision during training improves the segmentation performance of the convolutional neural network, making it more robust to seasonal changes and weather conditions.

Fine-Grained Segmentation Networks: Self-Supervised Segmentation for Improved Long-Term Visual Localization *by M. Larsson F. Kahl, et. al. , published at ICCV 2019.*

Long-term visual localization is the problem of estimating the camera pose of a given query image in a scene whose appearance changes over time. It is an important problem in practice, for example, encountered in autonomous driving.

In order to gain robustness to such changes, long-term localization approaches often use semantic segmentations as an invariant scene representation, as the semantic meaning of each scene part should not be affected by seasonal and other changes. However, these representations are typically not very discriminative due to the limited number of available classes. In this paper, we propose a new neural network, the Fine-Grained Segmentation Network (FGSN) that can be used to provide image segmentations with a larger number of labels and can be trained in a self-supervised fashion. In addition, we show how FGSNs can be trained to output consistent labels across seasonal changes. We demonstrate through extensive experiments that integrating the fine-grained segmentations produced by our FGSNs into existing localization algorithms leads to substantial improvements in localization performance.

6.6.2. Explored Research Questions

RQ4. How do you improve DNN based perception applications beyond state of the art?
To improve state-of-the-art for lane detection two main areas need to be considered. Firstly, the CNN needs to be carefully designed to achieve good accuracy. Secondly, the method for predicting the actual lane marking instances need to be carefully designed for the specific situation, and for the type of CNN used.

6.6.3. Deliverables and Goals

For deliveries a trained DNN for lane detection based on camera data was delivered. The performance, training procedure are documented in the work package report. The approach allows for fast real-time inference and performs well. Unfortunately, no large enough public benchmark with the same task was available at time of development.

6.7. WP7 – Training and Inference Platform Survey and Evaluation

For this work package different aspects of the software and the hardware in the inference and training phases of deep learning were evaluated. For doing this evaluation, we have used information from different sources including our own experiences in the development of the project.

Traditionally most of the tasks make use of the CPU to perform the computations but even though they are very efficient at performing complex operations, they are not that efficient with algorithms that require a large amount of simple calculations, like deep learning applications. The appearance of GPUs was the turning point for DL applications since they are intended to work with matrice operations in parallel. Alternatively, other kind of hardware suitable for DL have been appearing during these years. One example is the Field Programmable Gate Arrays (FPGAs). FPGAs are reconfigurable integrated circuits that can be configured to become any specific circuit design. Another hardware that became popular during last years for DL solutions were the Application Specific Integrated Circuit (ASIC). In this case, they are circuits specially designed to perform specific computations. This is opposed to writing instructions for a general-purpose chip like in the case of a CPU or GPU. This makes ASICs much more optimized, but the cost of production is higher. In thisreport, we will talk frequently about the Tensor Processor Unit (TPU), which is a custom ASIC released by Google [37].

On the software side, we were evaluating different frameworks for DL applications. A framework provides us with a number of tools, libraries and an application programming interface (APIs) that makes it easier to build and deploy our applications in a standardized way. In the last years, the ones that have become most popular are TensorFlow [14] and PyTorch [15].

6.7.1. Results

Training Hardware

In terms of hardware for training, FPGAs are not suitable to work with a considerable amount of data and parameters due to the low on-chip memory (which is normally the case during training) so in this case the benefits of this platform cannot be exploited during this phase of the process.

GPUs are the classic option for training. They are relatively easy to use, and they are integrated in most of the AI frameworks. The GPUs from Nvidia also allow the use of the CUDA libraries which makes it possible to optimize the acceleration of your neural networks [38]. This library is quite mature and the documentation is very extensive and a lot of options for optimization are given to the developer.

ASICs on the contrary are more specialized hardware that requires more skill to be programmed but because of that, they are normally more efficient chips. However, this also depends on several factors as the level of optimization for a specific model.

Regarding the performance, there has been several studies performed to compare these platforms for the training phase. The study done in [39] by Harvard University performs an interesting benchmark using TPU and GPU for training of some of the most common architectures of CNNs, like RetinaNet [40], ResNet-50 [41] and MobileNet [42] among other deep learning networks. The general conclusion is that TPUs performs best for larger batch sizes in in Fully Connected (FC), CNNs and RNNs. Figure 29 shows the FLOPS percentage utilization for different networks using CPU, GPU and TPU. A bigger number of FLOPS utilization means a better optimization of the hardware for the network and thus better performance.

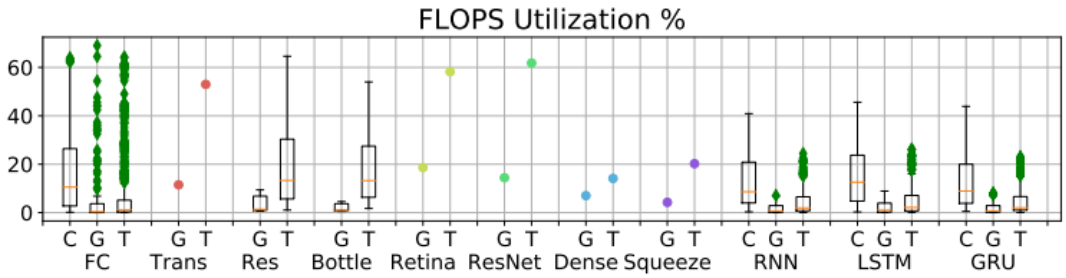


Figure 29: FLOPS Utilization % using CPU (C), GPU (G) and TPU (T) in several DNNs. The study is done in [39].

Another interesting benchmark is described in [64], where the author reaches the same conclusion as in the previous study using one of the most popular dataset CIFAR10 [49]. This evaluation makes use of a TPU in the cloud and GPU Nvidia K80 for the comparison. The test is done using the Keras version of Mobilenet and a Custom CNN consisting of 3 convolutional layers + 1 dense layer. Again, the experiments show that the TPU increases its performance when the batch size is large. However, the performance is better for the GPU when the batch size is reduced.

When it comes to the general cost, there are several studies comparing the cost between using regular NVidia GPUs and TPUs from Google and the result shows a slightly better cost-efficiency ratio for the TPUs. Still, the results depend on different considerations and the network model is also an important factor. In [47], a small convolutional network is trained using a TPUv2 and a Tesla V100 GPU and the results show that the most cost-efficient platform is the TPU in this case but for larger networks the difference would be smaller. Table 7 shows the comparison of the cost

of training in the previous study using regular machines and pre-emptible machines (virtual machines instances that are deleted after a few hours).

Device	Distributed	Mean training time (s)	Cost of Machine ($)	Cost Preemption ($)	Cost per Training ($)	Cost per Training Preemption ($)
TPU v2-8	Yes	190	4,57	1,37	0,2410	0,0734
GPU 1	No	769	1,87	0,78	0,3995	0,1668
GPU 8	Yes	208	14,09	5,98	0,8140	0,3454

Table 7: Cost comparison between TPU and Tesla V100 [47].

Training Software

Nowadays, the two most used frameworks for training and developing DL solutions are TensorFlow and PyTorch. PyTorch is the most balanced between simplicity and content. The modeling part is simple due to the use of a technique called reverse-mode auto differentiation, which is used to change the way a network behaves with a minor effort compared to other frameworks. In TensorFlow on the contrary, the graph that represents each model is not dynamic, so each time a model requires modification, this static graph has to be redone. This can be solved by using external packages but is not as natural as in PyTorch.

Anyway, different wrappers can be used on both TensorFlow and PyTorch to simplify the process of creating our DL model at the cost of decreasing the training speed due to the extra operations. In case of TensorFlow the most popular wrapper is Keras which is already integrated in the latest version of TensorFlow. In case of PyTorch, FastAI [48] is increasingly becoming the most used wrapper due to its simplicity for training and deployment.

Performance wise, most of the evaluations show that they are similar. The paper published by [50] evaluates the performance of TensorFlow and makes a comparison with two popular frameworks: Theano and CNTK. Theano is a DL framework created by the LISA Lab from the University of Montreal and CNTK is a DL framework that belongs to Microsoft. The experiment done during the study consists of evaluating the performance of the training process using MNIST, CIFAR-10 and a dataset from a Self-Driving Car Simulator as benchmarks for CNN models. They use several metrics to show the results but running time and memory utilization are the most relevant for our survey. In the case of MNIST and CIFAR-10 the result shows a lower GPU utilization for TensorFlow. For the Self-Driving Car dataset, the results were similar for all the frameworks. The running time on GPU was slightly better on Theano than TensorFlow for MNIST and CIFAR-10 and very similar for the Self-driving Car dataset with a small advantage for CNTK.

Another interesting article written for the site Synced, specialized at AI [51], where the author compares the performance during training of TensorFlow, PyTorch and MXNet. MXNet is another of the most used DL frameworks with GPU support, developed by Apache. The article performs a benchmark using VGG-16, ResNet-50 and Faster-RCNN using different test datasets. Figure 30 shows the results on training speed for the different models in the different frameworks. MXNet seems to be the faster on ResNet-50, TensorFlow on VGG-16 and PyTorch on Faster-RCNN. This is related to the different levels of optimization of these architectures on each framework.

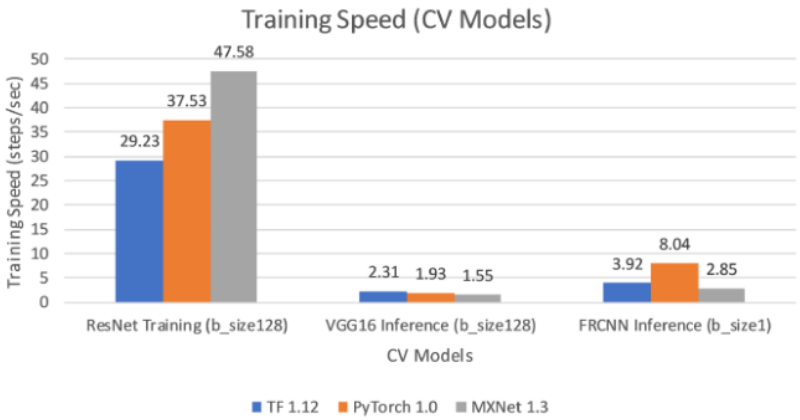


Figure 30: Training Speed Comparison of different computer vision DL models done by [51].

The article also analyses the GPU utilization and GPU memory utilization showing similar results for all the frameworks. The differences are always related with the optimization of the implementation for each framework, which can be achieved in other frameworks if the same implementation is performed. The article written by [52] also compares the performance of PyTorch, TensorFlow and Keras training with very popular CNN models like ResNet50, VGG16 and Inception. Keras in this case is just a wrapper of TensorFlow used to make the development of DL model more user-friendly [53]. The results in general shows that the difference in training speed between PyTorch and TensorFlow is not significant but Keras is slower than the other frameworks which is due to the infrastructure created to simplify the training process for the developer.

In terms of cost, all these frameworks are open source and therefore free to use. The engineering cost is also similar since the difficulty of programming is similar, but PyTorch has a slight edge in simplicity since it was designed to work with Python from the beginning. On the other hand, Tensorflow contains more ready implementations which also can reduce cost. Overall the investment needed for both of these frameworks is very similar.

Inference Hardware

It is in inference where FPGAs can have a big role nowadays. FPGAs are very flexible and can be reconfigured to work with different scenarios. This involves both easily modification of the algorithm used and reconfiguration of part of the chip while the remaining areas are still working [54], which make it ideal for fast prototyping. In addition, FPGA offers a high parallelization due to its editable logic hardware unit, which can be used to optimize the DNN and increase the inference speed [55]. But these benefits carry a cost, FPGAs are difficult to program and it can take some time to acquire the skills needed to work with this hardware. Reconfigurable computing requires hardware programming skills instead of the traditional high-level abstract programming languages most common on CPUs and GPUs. This is one of the main reasons that FPGAs are not used more in the industry.

Again, the performance is one of the main requirements to take into account when selecting an inference platform. An interesting study to analyze in this section is the one proposed in [56]. This paper analyzes the acceleration of RNNs in the inference process using FPGA, CPU, GPU and ASIC. The paper evaluates the performance using two modes, NoBatch, which means 1 input at a time, and Batch10 which means 10 inputs at a time. The result shows that without using any Batching, FGPA performs better than CPU and GPU but worse than ASIC. When using Batching the performance of the GPU is increased but still the device is underutilized in comparison with the other hardware. Figure 31 and Figure 32 show these results.

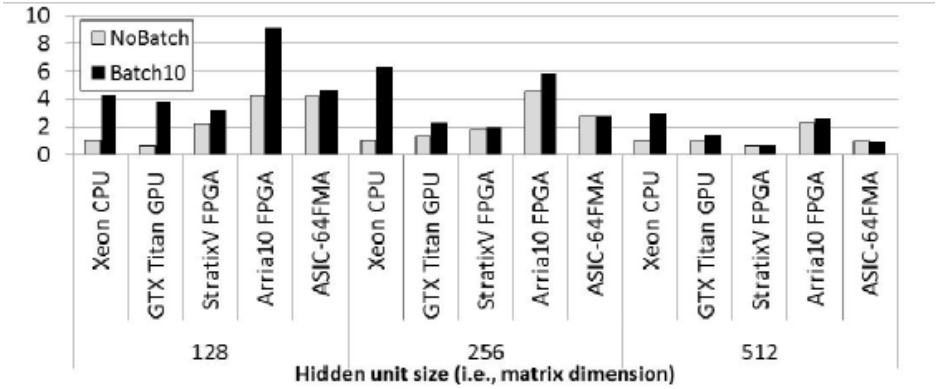


Figure 31: Performance of different accelerators. The y-axis in the diagram denotes the factor of performance increase compared to CPU performance with no batching [56].

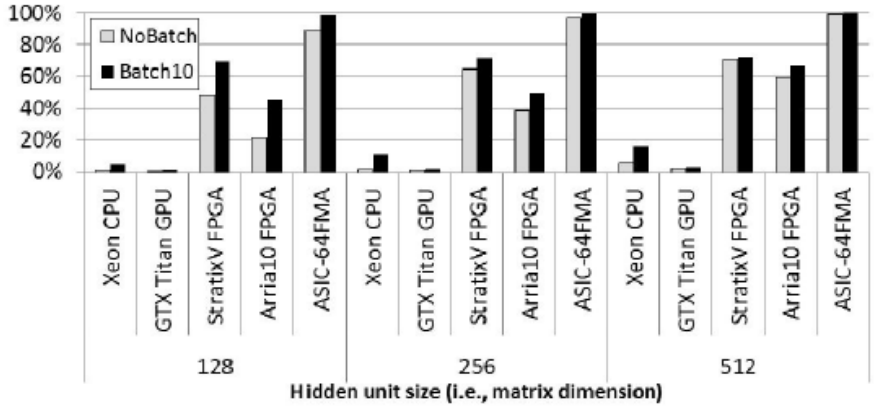


Figure 32: Percentage of utilization. E.g. 10% means that the system is underutilized, where the achieved GFLOPs is only 10% of the available peak GFLOPs. 100% means full utilization of the device [56].

Another interesting study is presented in [57], where a survey using different FPGAs for accelerating CNN inference is done. To do this, the author implemented different techniques of pruning and quantization on the models running on the FPGA, outperforming up to 10% of the state-of-the-art models on GPU. Similar conclusions are reached by the author in [58], this time testing on CNNs and GPUs. In summary, most of the studies conclude that FPGAs and ASICs have the potential to outperform GPUs in the acceleration of the inference of DNNs, since they can be designed to compute the specific operations required in DL. On the contrary, GPUs are designed for graphical computation and even thought this kind of computation works in a similar way to what is required for DL, they cannot match hardware designed specifically for it.

Regarding cost, FPGAs are an interesting option since the range of price is very wide and there exist options than can be cheaper than the others having the same performance. At the same time, they are normally very power efficient in comparison with GPUs. There exist several studies comparing the power efficiency of the FPGAs with other platforms like the one done Xilinx in [59]. In this study they compare a couple of FPGA models with some Nvidia GPUs and they find out that FPGAs can give up to 20% better efficiency than traditional GPUs. But of course, at the end this depends of the application to be done. Despite the mentioned benefits of FPGAs it is important to again emphasize the higher complexity of programming these platforms since this will increase the engineer cost as well.

Inference Software

Concerning the inference, the differences between different frameworks are similar as for the training process. TensorFlow is a little bit more scalable and offers more tools for deployment compared to PyTorch and it also supports a small version for mobile platforms and embedded systems. PyTorch is a newer framework and because of that it is lacking some of Tensorflow's features but most of these tools are added (or planned to be added) in different updates. The performance is also similar and the differences are again related more to the different implementations than to the platform itself.

6.7.2. Explored Research Questions

RQ3: What kind of inference platform is suitable for which embedded inference situation?

Different options for hardware and software were evaluated during the Perceptron Project. Regarding the hardware, each platform has pros and cons depending of the situation. For training GPUs and ASICs are more suitable, FPGAs can be an interesting option for fast prototyping for inference. For mass production, ASICs can be an interesting candidate since they can reach higher level of optimization than GPUs for DL applications. However, GPUs are currently the best option since they are flexible, easy to program and is supported by mature libraries like CUDA from Nvidia.

6.7.3. Deliverables and Goals

For this package a survey was done evaluating different options of hardware and frameworks for DL applications to fulfill *G2*. As mentioned in the goal, different aspects were taken into account like cost, usability, functionality, performance, limitations, automatisation and deployment. During the project, an inference platform and a training platform were also selected to perform the evaluation and development of a DL solution for the other package. As inference platform the Nvidia Drive PX2 was selected and for the training hardware the NVidia DGX was selected.

6.8. WP8 – Demonstration Truck

This work packages includes the demonstrator truck that was built in the project. This includes the hardware and the software that allows to run the networks while driving the truck in order to fulfill the goals of the project. The vehicle used as the demonstration truck is called “Golden Truck”. This truck is a Volvo FH model which have been modified to include the sensors and the hardware necessary to run the networks. The truck can be driven normally independently of the hardware installed since the actuators are not connected to the inference hardware. Figure 33 shows the exterior of the Golden Truck:

Figure 33: Lateral View of the Golden Truck used as a demonstrator truck.

6.8.1. Results

As mentioned before, the hardware selected for inference and logging was the Nvidia Drive Px2. Wire harnesses, connectors and switches were done and installed in the truck in order to power the Drive PX2 and the different components. Figure 34 shows the installation of the Nvidia Drive PX2 in the Golden Truck.

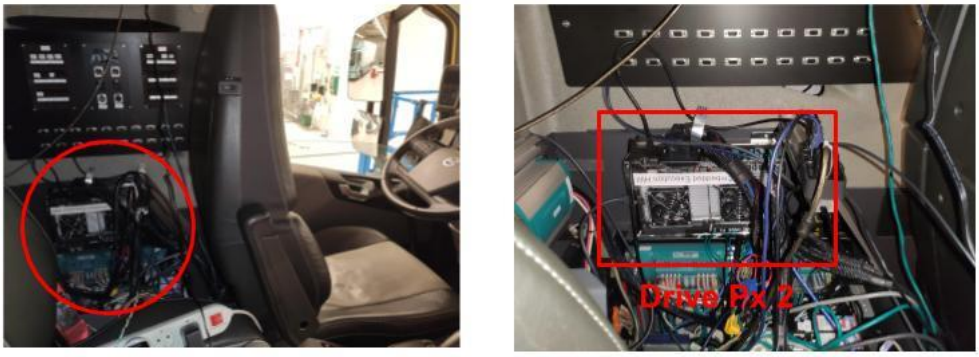


Figure 34: Inference hardware installation in the Golden Truck.

Several sensors were installed in the truck and connected to the Drive PX2. Figure 4 shows the sensor configuration used for the logging and the inference. The deployment of the networks on the hardware in the truck is done by using an external hard drive. The files necessary to update the SW can be transferred directly from the external drive into the internal memory of the Drive PX2. The demonstrator SW application was also done for this package in order to test the networks while driving. The demonstrator software application consists on several pieces of software that makes it possible to run the networks on the inference hardware. This includes, the software for the data acquisition, the software for doing inference with the networks, and the software for visualizing the output of the networks. Figure 35 shows a diagram of the software application components.

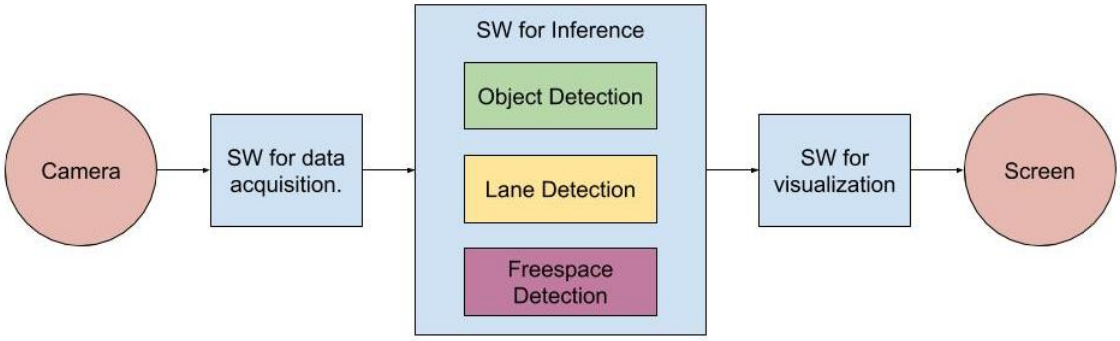


Figure 35: Diagram of the demonstrator software application components.

One of the RGB cameras in the front is used to capture the data to provide the input for our networks. The SW to do this uses as a base the drivers of Nvidia DriveWorks [60] and on top of that, a ROS [61] wrapper is used. This system transforms the packets received from the GMSL interface into a compatible format for the Nvidia Drive Software. Then the images in this format is converted into ROS Image format and published using ROS messages that can be read by the software that handles the networks. These drivers can be configured to acquire the data with different settings. Those settings include image compression, image resolution, image color mode and framerate. Figure 36 summarizes the pipeline for the data acquisition.

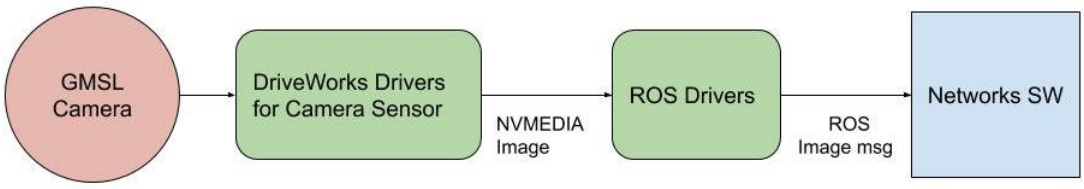


Figure 36: Diagram of the software used for the data acquisition.

Each network is run on a different ROS node that subscribes to the message published by the node of the camera. Once the message is received by the node of each network, the data is converted into a format that makes it possible to perform some pre-processing operations such as transformations

and normalization. After these operations, the result is input to each network that compute the different detections. Then, the output of the network is again processed to be visualized.

Table 8 shows the execution time needed to run each network node on the Nvidia Drive PX 2. In this case, the node of the lane detector is the slowest since the network generates a significant number of points that need to be post processed. This step requires a lot of computation and therefore increase the total execution time of the node.

Network Node	**Network Node Total Time (ms)**	**Inference time (ms)** **No pre/post-processing included.**
Free Space Detector	140	95
Object Detector	120	83
Lane Detector	240	55

Table 8: Execution time for the different network ROS nodes running on Nvidia Drive PX2. The first row shows the total time of the network, including pre-processing of the data, network inference and post-processing for visualization. The second row shows only the time that the network takes to perform inference.

Running the three networks required more computation power than can be handled by one Tegra. Due to this, both Tegras included in the Drive PX2 can be used to run all the networks in parallel. Since the data acquisition and the visualization are done in the Tegra A, the data needs to be transferred to/from the Tegra B. ROS allows the communication between different machines connected to the same network by configuring one of the machines as a master. In this way, the only requirement is that the master needs to start a node before any other machine connected to the network. Doing this, the different machines can publish messages that can be read in other machines. Since both Tegras are connected using a network bridge this configuration master/slave can easily be done. During the project we implemented this concepts to make the networks work in both Tegras at the same time. Figure 37 describes the message flow used for the transmission of the ROS messages between the different Tegras that was used in the project.

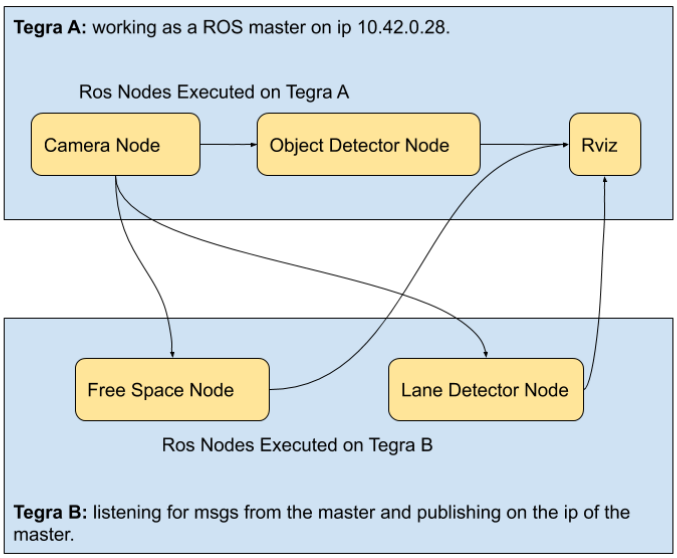


Figure 37: Message transmission flow from the different ROS nodes executed in parallel in both Tegras.

ROS is also used for visualization purposes. This allows us to use a number of tools available for these frameworks for visualization and data representation. RViz is one of the most used ROS tools for visualization due to the ability of visualizing both 2D and 3D data under the same reference system [62]. In our case the output of the ROS nodes of our networks are images. Since each network works with a different framerate, the easiest way to visualize the information is to use different display windows for each output image. Figure 38 illustrates the visualization pipeline using the ROS package RViz.

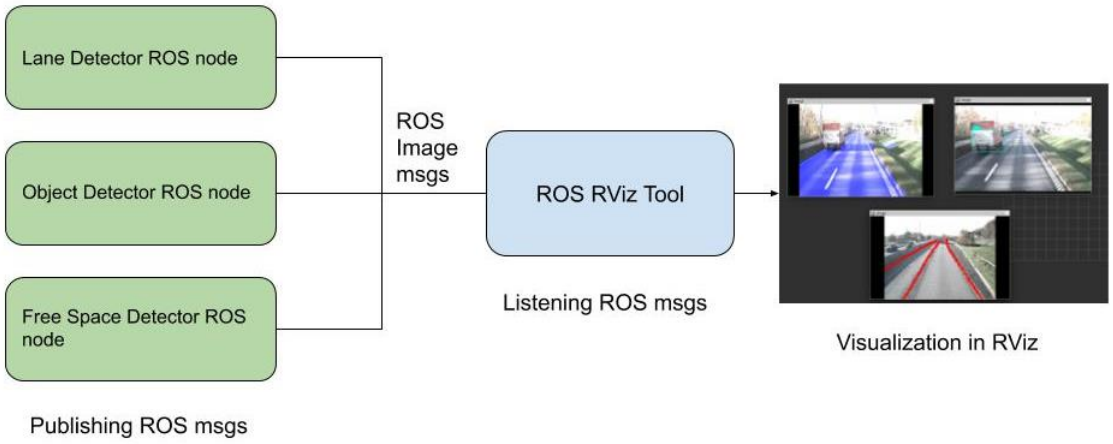


Figure 38: Visualization of the output of the networks.

6.8.2. Explored Research Questions

This package is not related directly with any of the researched question proposed in the application, but it involves indirectly all of the research questions since it involves all the steps of the Deep Learning chain. The truck has the software and hardware that is necessary for logging data that then is used to train the networks that will be deployed in the trucks again to perform the inference process in real time while driving.

6.8.3. Deliverables and Goals

The delivery of this work package contains the demonstration truck which includes the hardware and software needed for running a DL based perception system. This deliverable supports the goals fulfilled by the other package since it is demonstrating the complete DL solution.

7. Dissemination and publications

Dissemination

The dissemination of the Perceptron project has occurred through several mechanism:

- The results and development done during the project has helped to increase the knowledge for developing of DL solutions. This knowledge involves crucial tasks in a DL pipeline, like algorithm selection, data logging needs, selection of training platform and inference hardware, supplier evaluation, etc… This knowledge has been spread in the Volvo organization through presentations at several occasions.
- The deliverables and knowledge acquired during the project has been used in other advanced engineering projects. The data collected and annotated during the project has been used in AUTOFREIGHT [66]. The trained network will be used as a base for SHARPEN [67] and the knowledge acquired has been transferred into those projects and also the project PRELAT [65].
- The Perceptron project is part of a bigger cluster of projects focused at safely adapting DL technology within perception for autonomous driving. In this cluster, other FFI projects like SMILE I and SMILE II [68], AUTOFREIGHT, PRELAT, REALSIM4AD and SHARPEN are included. Volvo is a partner in all of them which have provided a lot of synergies and cooperation among the different industrial and academic partners.
- Indirectly the knowledge gained in the project and some deliverables will be passed into other product development projects and ultimately introduced on the market.

How are the project results planned to be used and disseminated?	Mark with X	Comment
Increase knowledge in the field	X	The project has been used to increase the knowledge in the area of development of DL solutions.
Be passed on to other advanced technological development projects	X	The results of the project will be used on another advance engineering projects, like SHARPEN.
Be passed on to product development projects		
Introduced on the market		
Used in investigations / regulatory / licensing / political decisions	X	The results of the project has been used to have a better understanding of the regulations related with the GDPR and the use of the data.

Table 9: How the project results are directly intended to be used or passed on.

Publications

The perception project has been disseminated along the duration of the project in numerous internal expositions and events. The project has been also disseminated through public conferences and publications. This is the list of events and papers:

- M Larsson, et al., “A Projected Gradient Descent Method for CRF Inference allowing End-To-End Training of Arbitrary Pairwise Potentials”, International Conference on Energy Minimization Methods in Computer Vision and Pattern Recognition (EMMCVPR), 2017 0.
- C Toft, et al. “Long-term 3D Localization and Pose from Semantic Labellings”, 3D Reconstruction meets Semantics in conjunction with International Conference on Computer Vision (ICCV), 2017 [24].
- M Larsson, et al., “Revisiting Deep Structured Models for Pixel-Level Labelling with Gradient-Based Inference”, 2018 [43].
- Deep Learning Deployment on Trucks in Auto AI conference in Berlin, 2018.
- Dissemination in Vehicle Electronics and Connected Services Conference, 2018.
- A Eriksson, et al., “Rotation Averaging and Strong Duality”, CVPR, 2018 [44].
- T Sattler, et al., “Benchmarking 6DOF Outdoor Visual Localization in Changing Conditions”, CVPR 2018 [45].
- C Toft, et al., “Semantic Consistency for Long-Term Visual Localization”, 2018 [46].
- M Larsson, et al. “A Cross-Season Correspondence Dataset for Robust Semantic Segmentation", IEEE Conference on Computer Vision and Pattern Recognition, 2019 [25].
- M. Larsson, F. Kahl. Et al, “Fine-Grained Segmentation Networks: Self-Supervised Segmentation for Improved Long-Term Visual Localization”, ICCV 2019 [63].

8. Conclusions and future research

In the Perceptron project a complete data driven DL solution was developed including all the processes, from the initial requirement setup up, to the deployment and testing of the neural networks. This was achieved using the Build-Measure-Learn methodology that allowed us to continue improving our system based on our measurements. The research on the state-of-the-art and existing solutions was also a key factor that helped us to identify the suitable technology and adapt it to our current solution.

Different kinds of technologies were developed in the Perceptron project. The complete infrastructure for logging data and inference of the network was built. The sensor set-up was defined based on the initial requirement and the hardware was acquired and integrated in a demonstrator truck together with the development software. The complete pipeline for data pre-processing and anonymization were performed which allowed us to annotate the data and build our own dataset. This dataset was used to train some of our developed networks which increased the accuracy considerably compared to the networks only trained with public datasets. The networks were then deployed in the truck to be tested thus closing the circle of our data driven DL solution.
In summary, this project proves that a DL-based perception solution will increase its performance by having dedicated data for each specific solution. And to have this data, the development of an infrastructure that handles continues deployment needs to be built. For example, having specific spots where the trucks can stop and send the data to a server in the cloud. The arrival of the 5G technology will be a key factor for the implementation of this feature.

Regarding future work, due to the lack of time there are some areas that we did not have time to explore. The creation of a wireless data transfer system would be an interested topic as it was laid out in the original application. This would require creating the infrastructure necessary to transfer the data logged in the truck to a backend server by using designated hotspots. The wireless transfer could also be used for the remote deployment of the network from the backend into the truck. In this way, the networks can be updated continuously after new training sessions. Also it would be interesting to explore the question of how to determine if the data from a logging session contains new valuable information or if the information is already covered by data logged in previous logging sessions.

Regarding the networks, there are some experiments that would be interesting to perform. For example, to train the networks with different sizes of our dataset and find a trade-off between size of a dataset and accuracy. This could help to understand how big a dataset should be to make a significant increase of the accuracy. Another interesting research topic would be to evaluate the performance of the networks using different kind of compressed data to understand what would be an acceptable rate of compression for training the networks.
The study of other sensor modalities (not only camera), would be one of the most prioritized topics of research and will be considered in future projects.

9. Participating parties and contact persons

The project participants and their roles are:

- *Volvo GTT:* Coordinator and principal contributor in the project providing needs related to commercial products, development of solutions related to the identified technologies and research on object detection.
- *Semcon Sweden AB*: Supplier and consultant provider to the automotive industry with a DL business unit. Semcon has contributed with own research on free space detection and DL knowledge to the consortium.
- *CTH:* Academic institution with expertise in Deep Learning and computer vision. CTH has performed own research on lane detection and contributed with DL knowledge and support to the consortium.

The contact persons of every institution are:

Organization	Name	Email
Volvo GTT	Carlos Gil Camacho	carlos.camacho@consultant.volvo.com
Semcon	Jens Henriksson	jens.henriksson@semcon.com
CTH	Fredrik Kahl	fredrik.kahl@chalmers.se

10. References

Articles & Papers

[1] Gartner hype cycle 2019: https://www.forbes.com/sites/louiscolumbus/2019/09/25/whats-new-in-gartners-hype-cycle-for-ai-2019/#49b68061547b

[2] Intel GO: https://newsroom.intel.com/news-releases/bmw-group-intel-mobileye-will-autonomous-test-vehicles-roads-second-half-2017

[3] AMD: http://www.forbes.com/sites/patrickmoorhead/2016/12/12/amd-enters-deep-learning-market-with-instinct-branded-accelerators-and-software-stacks/#6fb7b98b3fec

[4] Select relevant data: http://www.dis.uniroma1.it/~pretto/papers/pnp_ias2016.pdf

[5] Conditional Random Fields: M Larsson, F Kahl, S Zheng, A Arnab, P Torr, R Hartley, "Learning Arbitrary Potentials in CRFs with Gradient Descent", arXiv preprint, 2017.

[6] Benchmark: J. Fritsch et al, "A new performance measure and evaluation benchmark for road detection algorithms," in *International Conference on Intelligent Transportation Systems (ITSC)*, 2013.

[7] Microsoft Common Object in Context: https://arxiv.org/abs/1405.0312

[8] LIDAR: https://www.ngs.noaa.gov/RESEARCH/RSD/main/lidar/lidar.shtml

[9] Lane Detection: https://patentimages.storage.googleapis.com/cf/1e/f7/612a2a705aa228/US5351044.pdf

[10] Object Detection: https://papers.nips.cc/paper/5207-deep-neural-networks-for-object-detection.pdf

[11] Free Space Detection: https://ieeexplore.ieee.org/document/7313302

[12] Autopilot Review: https://www.autopilotreview.com/lidar-vs-cameras-self-driving-cars/

[13] NVIDIA: https://la.nvidia.com/object/drive-px-la.html

[14] Tensorflow: https://www.tensorflow.org/

[15] Pytorch: https://pytorch.org/

[16] Darknet: https://pjreddie.com/darknet/

[17] Artificial Intelligence: A Modern Approach: P.N. Stuart, J. Russell, Prentice Hall, 2010.

[18] COCO Dataset: http://cocodataset.org/#home

[19] PASCAL VOC Dataset: http://host.robots.ox.ac.uk/pascal/VOC/

[20] KITTY Dataset: http://www.cvlibs.net/datasets/kitti/

[21] CITYSCAPE Dataset: https://www.cityscapes-dataset.com/

[22] GeoJSON format: https://geojson.org/

[23] A Projected Gradient Descent Method for CRF Inference allowing End-To-End Training of Arbitrary Pairwise Potentials: https://arxiv.org/abs/1701.06805

[24] Long-term 3D Localization and Pose from Semantic Labellings: http://www1.maths.lth.se/matematiklth/vision/publdb/reports/pdf/toft-etal-iccv-2017.pdf

[25] A Cross-Season Correspondence Dataset for Robust Semantic Segmentation: https://zpascal.net/cvpr2019/Larsson_A_Cross-Season_Correspondence_Dataset_for_Robust_Semantic_Segmentation_CVPR_2019_paper.pdf

[26] Image-Net competition http://www.image-net.org/

[27] YoloV3: https://arxiv.org/abs/1804.02767

[28] Mask R-CNN: https://arxiv.org/abs/1703.06870

[29] Faster R-CNN: https://arxiv.org/abs/1506.01497

[30] SSD: Single Shot MultiBox Detector: https://arxiv.org/abs/1512.02325

[31] Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning https://arxiv.org/pdf/1602.07261.pdf
[32] mAP: https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173
[33] COCO benchmark real time object detection: https://paperswithcode.com/sota/real-time-object-detection-on-coco
[34] Fast LIDAR-based Road Detection Using Fully CNN: https://arxiv.org/abs/1703.03613
[35] SqueezeSeg: Convolutional neural nets with recurrent crf for real-time road-object segmentation from 3d lidar point cloud: https://arxiv.org/abs/1710.07368
[36] U-Net: https://arxiv.org/abs/1505.04597
[37] Google, "Cloud TPU" : https://cloud.google.com/tpu/
[38] NVIDIA CUDA: https://developer.nvidia.com/cuda-zone
[39] Benchmarking TPU, GPU, and CPU Platforms for Deep Learning: https://arxiv.org/abs/1907.10701
[40] RetinaNet, Focal Loss For Dense Object Detection: https://arxiv.org/abs/1708.02002
[41] ResNets: https://arxiv.org/abs/1512.03385
[42] MobileNets: https://arxiv.org/abs/1704.04861
[43] Revisiting Deep Structured Models for Pixel-Level Labelling with Gradient-Based Inference: https://research.chalmers.se/publication/509889
[44] Rotation Averaging and Strong Duality: http://openaccess.thecvf.com/content_cvpr_2018/html/Eriksson_Rotation_Averaging_and_CVPR_2018_paper.html
[45] Benchmarking 6DOF Outdoor Visual Localization in Changing Conditions: http://openaccess.thecvf.com/content_cvpr_2018/html/Sattler_Benchmarking_6DOF_Outdoor_CVPR_2018_paper.html
[46] Semantic Consistency for Long-Term Visual Localization: https://research.chalmers.se/en/publication/507453
[47] Cost Comparison between Google TPUv2 vs Nvidia Tesla V100: https://timdettmers.com/2018/10/17/tpus-vs-gpus-for-transformers-bert/
[48] FastAI: https://www.fast.ai/
[49] CIFAR-10 dataset: https://www.cs.toronto.edu/~kriz/cifar.html
[50] A detailed comparative study of open source deep learning frameworks: https://arxiv.org/abs/1903.00102
[51] Synced, "TensorFlow, PyTorch or MXNet? A comprehensive evaluation on NLP & CV tasks with Titan RTX": https://medium.com/syncedreview/tensorflow-pytorch-or-mxnet-a-comprehensive-evaluation-on-nlp-cv-tasks-with-titan-rtx-cdf816fc3935
[52] Deep Learning Frameworks Speed Comparison: https://wrosinski.github.io/deep-learning-frameworks/
[53] Keras: The Python Deep Learning Library: https://keras.io
[54] FPGA vs ASIC Comparison Summary: https://numato.com/blog/differences-between-fpga-and-asics/
[55] A Survey of FPGA Based Deep Learning Accelerators: Challenges and Opportunities: https://arxiv.org/pdf/1901.04988.pdf
[56] Accelerating Recurrent Neural Networks in Analytics Servers: Comparison of FPGA, CPU, GPU, and ASIC: https://jaewoong.org/pubs/fpl16-accelerating-rnn.pdf
[57] Accelerating CNN inference on FPGAs: A Survey: https://arxiv.org/pdf/1806.01683.pdf
[58] A Survey on CNN and RNN Implementations: https://www.semanticscholar.org/paper/A-Survey-on-CNN-and-RNN-Implementations-Hoffmann-Guzm%C3%A1n/4b986ab162c920855cbb2f12956ea01d3a590caa

[59] Xilinx All Programmable Devices: https://www.xilinx.com/support/documentation/white_papers/wp492-compute-intensive-sys.pdf
[60] Nvidia Drive Software: https://developer.nvidia.com/drive/drive-software
[61] Robot Operating System: https://www.ros.org/
[62] Robot Operating System – Rviz: http://wiki.ros.org/rviz
[63] Fine-Grained Segmentation Networks: Self-Supervised Segmentation for Improved Long-Term Visual Localization: https://arxiv.org/abs/1707.09092
[64] Comparison GPU and TPU training performance on Google Colaboratory: https://medium.com/datadriveninvestor/comparing-gpu-and-tpu-training-performance-on-google-colaboratory-c1e54e26993f

Projects

[65] PRELAT: https://www.vinnova.se/p//PRELAT/
[66] AUTOFREIGHT: https://www.vinnova.se/p/highly-automated-freight-transports/
[67] SHARPEN: https://www.vinnova.se/p/sharpen---skalbara-hogautomatiserade-fordon-med-robust-perception/
[68] SMILE: https://www.vinnova.se/p/smile-ii---sakerhetsanalys-och-verifieringvalidering-av-system-baserade-pa-maskininlarning/
[69] REALSIM4AD: http://realsim4ad.com/