CHALMERS
UNIVERSITY OF TECHNOLOGY

Detection of child-like objects inside vehicles using deep learning

A proof of concept study investigating if pruned CNNs deployed on mobile devices can detect child-like objects situated in vehicles.

Master's thesis in Electrical Engineering

HENRIK HALLBERG

VICTOR WESSBERG

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2018

Master's thesis 2018:EX012

Detection of child-like objects inside vehicles using deep learning

A proof of concept study investigating if pruned CNNs deployed on mobile devices can detect child-like objects situated in vehicles.

HENRIK HALLBERG

VICTOR WESSBERG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Signal Processing and Biomedical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2018

Detection of child-like objects inside vehicles using deep learning.

A proof of concept study investigating if pruned CNNs deployed on mobile devices can detect child-like objects situated in vehicles.

HENRIK HALLBERG
VICTOR WESSBERG

Supervisor: Jennifer Alvén, Department of Electrical Engineering
Examiner: Fredrik Kahl, Department of Electrical Engineering

Master's Thesis 2018:EX012
Department of Electrical Engineering
Division of Signal Processing and Biomedical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Various graphs and images from this report.

Typeset in LaTeX
Gothenburg, Sweden 2018

Detection of child-like objects inside vehicles using deep learning.

A proof of concept study investigating if pruned CNNs deployed on mobile devices can detect child-like objects situated in vehicles.

HENRIK HALLBERG
VICTOR WESSBERG

Department of Electrical Engineering
Chalmers University of Technology

Abstract

This master's thesis project comprises a proof of concept study with the aim to investigate if computer vision is a suitable tool for detecting child-like objects situated in vehicles. The developed system consists of a Raspberry Pi equipped with a camera sensor and the system is able to classify images in real time, using a convolutional neural network. Datasets were constructed, consisting of annotated images collected using both online sources, as well as with an experimental setup. The annotated images in the datasets show vehicle interiors with and without child-like objects present. These datasets were used to train, validate and test the convolutional neural networks in this project. Several convolutional neural networks were evaluated, showing that the well-known VGG16 architecture would yield the best results for this classification task. The convolutional neural network, based on the VGG16 architecture, was compressed with network pruning to accelerate the performance, while still maintaining the classification accuracy.

The results imply that a computer vision approach generate promising performance for the task of detecting child-like objects with high accuracy, which states that the proof of concept was successful. However, the system developed was not able to correctly classify all scenarios presented, and more data is needed to develop a system able to provide even more reliable detections, which would be a necessary step in order to commercialize the system.

Keywords: Convolutional Neural Networks, Deep Learning, Machine Learning, Network Pruning, Compression, Mobile Devices, Raspberry Pi, Embedded Systems, Image Classification.

Acknowledgements

We would like to thank our inspiring supervisor, Jennifer Alvén, who has contributed with valuable knowledge concerning the project. We would also like to thank all people working at Alten - Embedded Systems for their kind hospitality, support and coffee.

Henrik Hallberg & Victor Wessberg
Gothenburg, June 2018

Contents

1
Introduction

This chapter aims to provide the reader with background information on why this master thesis project was conducted. The chapter will also present a definition of the problem as well as, aim and limitations of the project. To give the reader an overview of the task, previous work conducted on the subject will also be presented.

1.1 Background

Sunny days is by many people often considered as a pleasant time with ice cream, ocean swims and long hot days. Unfortunately, this is not the case for everyone. For some, this time is related to tragic events caused by the dangerous rays from the sun. One group highly affected by these rays are small children incapable of protecting themselves. Leaving a child inside a vehicle, where heat rapidly builds up on a sunny day, will quickly lead to severe consequences or even death. In the United States, these sunny days have recently been highly associated with tragic accidents concerning children entrapped in vehicles. Occurrences of children killed by heat inside vehicles increase during the summer, but these accidents are reported even in the colder months. The rays from the sun can quickly make the passenger compartment of a vehicle a very dangerous place for children even when it is relatively cold outside [1]. In the U.S. alone, one child is killed every 10 days from being trapped inside a hot vehicle [2].

The website KidsAndCars.org has been collecting data regarding non-traffic incidents involving children and cars in the U.S. during the last 25 years, and as of today this database got the most comprehensive data of heat related child vehicle deaths. The data shows that the main reason behind these tragic events are parents forgetting that their child was in the vehicle all along. Other reasons are children accidentally locking themselves in and children being intentionally left in the vehicles when the parents leave to do quick tasks such as shopping, often without knowledge of the dangers they are exposing the children to.

The modern vehicles of today, often connected to cloud services, could prevent the cases of children being accidentally left inside vehicles by sending the parent a notice through the phone if a child is detected inside a parked vehicle. To provide such a notice, a sensor-system providing highly reliable detection is required. Time magazine released an article in 2014, stating that vehicle manufacturers did not think that technology yet existed to provide this detection in a way reliable enough

[3]. The vehicle manufacturers stated fear of misuse of the parents and malfunction that could lead to expensive lawsuits in situations where the system had failed and resulted in a lethal outcome [3].

Extensive literature studies show that there are still today no vehicle manufacturers offering a system to detect children alone in vehicles and thus preventing these kinds of accidents in a reliable way. However, such a system could be of great use to reduce child death and should be developed as quickly as possible. The tremendous increase in popularity for deep learning in recent years, both within academia as well as for commercial purposes, might yield a possibility to use such an approach when implementing child detection systems in future vehicles. The quickly evolving methods for classifying objects in images using deep learning might lead to a possibility to construct a computer vision system robust enough to cope with the task of detecting children inside vehicles. This project aims to investigate if such an approach would be feasible.

1.1.1 Statistics

The National Highway Traffic Safety Administration (NHTSA) has been working to prevent heat-related child deaths for several years. Statistical data of death occurrences in the U.S. have been gathered since 1998, yielding a overview of the current situation. The data shows that 742 children under the age of 14 have died between 1998 and 2017, yielding an average of 37 fatalities yearly. It also shows that the problem is not decreasing, rather the opposite. The latest reports show that 43 fatalities were reported during 2017 [4].

The increase of fatalities during the recent years have led to the U.S. Congress issuing a bill on the subject, suggesting that *"all new passenger motor vehicles to be equipped with a child safety alert system"* [5].

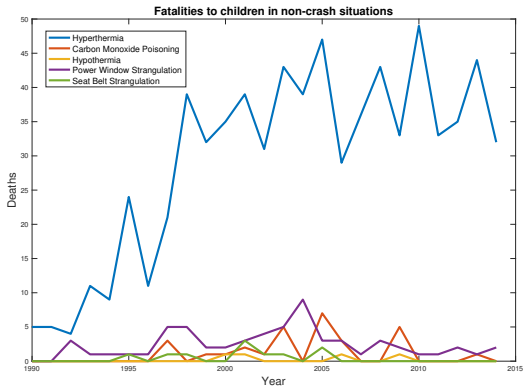

Figure 1.1: A graph showing child deaths from inside a vehicle in non-crash incidents over the years 1990 to 2014. Data summarized by Zonfrillo [6], but originally from NHTSA [7]. The blue line, representing hyperthermia, cover a majority of the cases.

1.1.2 Hyperthermia

Exposing the body to high temperatures can quickly lead to severe implications. The cellular processes of the body is directly affected by heat, and the body will try to reduce the response by cooling the body with sweat. This mechanism will distribute the blood flow away from the core of the body, putting large stress on the heart, which ultimately may fail to produce adequate cardiac output. This stress will eventually lead to cardiovascular collapse, multiorgan failure and ultimately death [8]. A child is more vulnerable to damage from heat than an adult, both because the body of a child heats up 3 to 5 times as quickly than the body of an adult, and also because a child has an immature and not yet fully developed thermoregulation system [8]. Studies show that the body of a child will reach lethal temperatures within 125 minutes on a hot summer day, and within 350 minutes during the winter months in Austin Texas, while situated in an enclosed environment like a vehicle [9]. This show that during a full day of work, the child could perish even during the coldest months.

1.1.3 Vehicle Heat Dynamics

On a sunny day the temperature inside an enclosed vehicle will rapidly increase, making the cabin dangerously hot. Research show that on a sunny day with ambient temperatures of 20°C the internal vehicle temperature will reach 40°C within the first half hour of being turned off, and then continue to increase even further. In a report from 2005, vehicles were placed in the sun for 60 minutes. The study showed that 80% of the total temperature increase occurred within the first half hour, meaning temperatures in the 40°C region. It was also shown that an ambient temperature of 20°C or 35°C did not significantly change the rate of temperature increase, which pose a big threat to unaware parents not knowing that the vehicle will heat up rapidly, even though it is not very hot outside [10, 11]. In the worst case, the internal vehicle temperature can reach child-lethal levels within 10 minutes, according to NHTSA [12].

1.2 Problem Description

There have been many approaches for trying to solve the problem of detecting forgotten children in vehicles, as stated in the previous section. Within the field of computer vision and more specifically within the image classification field, the use of machine learning is becoming more and more popular for all kinds of classification tasks, since it has shown great results in terms of classification speed and accuracy compared to other techniques [13].

The main problem addressed in this project is the development of a system aimed to detect child-like objects (CLOs) in vehicles using image classification with means of machine learning. The project is conducted as a proof of concept study and tries to answer the following research questions:

- Is it possible to detect CLOs using machine learning running on cheap hardware?
- What classification robustness is achievable?
- Is it possible to extract enough training data during the time frame for this project?
- What limitations does the implemented detector exhibit?
- Do the results of this thesis imply that a similar software system for detecting real children would be possible?

The thesis is conducted at Alten Sweden AB, in collaboration with with the Computer vision and Medical image analysis research group at Chalmers University of Technology. Alten provides with necessary hardware and facilities as well as knowledge considering development of embedded systems. The machine learning and computer vision expertise is mainly supplied by Chalmers.

1.2.1 Aim

The aim of this masters thesis is to develop an embedded system capable of detecting CLOs inside vehicles. The detection should be carried out using computer vision with a machine learning approach. Data will be collected, augmented and used for training a Convolutional Neural Network (CNN), that finally should be deployed on an embedded system placed inside a vehicle. The aim is to collect at least 500 images, with 100 positive samples where the CLO is visible, and 400 negative samples.

1.2.2 Limitations

- The automotive industry put high demands on keeping the component cost low. The developed system must therefore be implemented with low-cost components, leading to limitations on available computational power from the microcomputer in the embedded system and the resolution of run-time images.
- The computational power delivered from the relatively low-cost hardware used in the embedded system will put limitations on how complex the classification algorithm can be. The system must be able to analyze images from the passenger compartment of the vehicle, within a suitable time frame and with a high accuracy. Also, the performance of the provided computers for the development will put limitations on the complexity of the classification algorithm.
- Due to practical considerations, CLOs (i.e. dolls) will be used to mimic small children.
- Due to the relatively short time frame of a 30 credits master thesis project the experimental data collection will be limited in regards to vehicle variation and subject variation.
- The project will focus on heat stroke prevention only, not scenarios where the environment is dark and the vehicle will not heat up to dangerous temperatures.
- The run-time of a classification will have to be less than 1 minute.

1.3 Previous Work

With the increasing problem of heat-related child deaths, the development of various solutions for detecting children in vehicles have been initiated, on multiple fronts. In this section, a range of third-party solutions, currently on the market, is presented. Thereafter, a brief introduction to computer vision for image classification follows, to later lead into how CNNs today are used to solve a wide range of problems.

1.3.1 Child Detection Systems

Over the past 30 years there has been a significant increase in safety-related technology surrounding vehicles. However, this technology have mainly been aimed towards reducing crash-related incidents, like lane-following and collision avoidance warnings. As mentioned in Section 1.1.1, non-crash related incidents are increasing and forgotten children dying from hyperthermia cover a substantial part of these cases.

In 2015, NHTSA compared the functionality of a range of commercial products, many of which focused on detecting the weight of the child using a pressure sensor installed in the child restraint system, where system activation was achieved through smartphone notification, vehicle horn or an auditory signal from the device itself [14]. Many of these products rely on a key fob to detect when the driver walks away from the vehicle. Some, more complex solutions utilize a connection between the vehicle computer and a built in logic chip on the child protection system, to detect if the child is still strapped to the child car seat (CS) when the vehicle is locked.

Other solutions include passive infrared motion sensors, temperature sensors, microwave sensors, capacitive sensors or microwave sensors [15, 16]. These techniques have, to the best of our knowledge, not yet succeeded in providing proper reliability to detect forgotten children inside vehicles. Several of the developed systems rely on sensors installed in CSs, which limits the detection capabilities to infants only, since the system will only detect the child if it is placed in a CS [16].

The most recent and promising development within the field was presented at the Consumer Electronics Show 2018 by a company called Guardian Optical Technologies, who have come up with an integrated sensor capable of detecting motion down to one micrometer in scale, meaning that the sensor is capable of detecting the heart beat of a small child, even without a direct line of sight [17, 18]. The sensor make use of video image recognition (2D), depth-mapping (3D), and micro- to macro-motion detection to create this passenger-aware-sensor, which could also work as a detector of forgotten children inside a vehicle [19, 20].

Detection systems of adults using camera based sensors show promise and could be found in a various range of other applications today, some of them are pose estimation, face recognition, counting individuals in crowded spaces and various

safety systems like eye-tracking in modern vehicles [21, 22, 23, 24]. The application areas for small children within computer vision seem to be focused toward the medical imaging arena and not very much development have been made towards child-focused applications such as child detection in images.

1.3.2 Classification of Objects Using Computer Vision

Machine learning has become an important tool for several real world applications such as suggestions to web-users, understanding hand-writing and image classification. Focusing on the computer vision area, machine learning has recently shown great results, both considering the accuracy of which it manages to classify objects in images and also the computation time for the classification. These promising results can be explained by several factors, but the main breaktroughs are the increased availability of image data and computing power [25, 26].

The internet and web-based applications surround almost everything in modern society, and with them, the amount of content available is increasing. By 2017 there were over 95 million images being uploaded to Instagram, each day. The same trend can be seen with the amount of video data uploaded, with over 300 hours of video being uploaded to Youtube, each minute [27]. In terms of medical images there were over 60 billion medical images generated in the U.S. alone by 2015, with an expectancy to keep growing exponentially over the coming years [28]. Much of this data contains information that could be useful for several applications, if extracted properly, which has been the cradle for image classification during the last decades.

Image classification is the task of extracting information from image data by dividing images into different classes, based on what the image represents or shows. An example could be to differentiate a cat from a dog in an image, or even to determine the specific race of the creature. The localization of the objects in the images are not of high importance in image classification, but more associated to object detection and localization [29].

Image classification within the learning based arena is divided into two main categories, both with their own approach to the classification problem. These approaches are parametric and non-parametric classification, where the latter have seen a decrease in popularity due to the introduction of CNNs [30]. Parametric classification methods include support vector machines (SVM), random forest and neural networks. Non-parametric classification methods have some advantages over the parametric methods. Firstly they do not require any training and simply base their predictions directly on the data. This, in addition to being more robust against overfitting and also providing reasonable performance when dealing with large amounts of data made the non-parametric methods popular in the early 2000's [30]. One of the most popular non-parametric methods is the nearest-neighbor technique, where classification is based on similarities in image space, which makes this technique sensitive to intra-class variation [30]. Even until 2012 there were still some applications where non-parametric classification methods were shown to outperform parametric

classification methods [31].

With the technology advancing, especially Graphical Processing Units (GPUs), the advantages of non-parametric classification methods were diminished by the amount of data that could be used together with parametric classification methods. The first technique to take major leaps in classification performance within this branch was the SVM classifier, which was the leading approach by 2008 [30, 32, 33]. The fundamental idea behind the SVM was to find the hyperplane that manages to separate the different classes as well as possible [34].

For several computer vision tasks, CNNs have proven to outperform the other techniques available today [35]. The outstanding performance of CNNs in several image recognition and classification task has led to a continuous increase in their usage. Several research groups have created CNNs capable of outperforming human-level accuracy in image recognition, classification and detection tasks [36].

There are several CNNs that have obtained astonishing results in competitions such as the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), where some examples of the most popular and high-performing CNNs are AlexNet, VGGNet and ResNet [36]. The common factor among these high performing CNNs is the high complexity. The large amount of convolutional layers allows the CNNs to extract high level abstractions and features, and the fully connected layers can thereafter learn non-linear combinations of these features, allowing for the high performance in image classification tasks [36]. More on this in Chapter 2.

As of today, there are a wide range of more modern applications in which image classification is utilized, some of the more well known areas are face recognition tasks [37, 38, 39], satellite remote sensing [40, 41, 42] and medical image classification [43, 44, 45]. With the technology, knowledge and methods advancing, many more areas where image classification could be applied in a sufficient manner are in development. Some examples are autonomous vehicles, safety systems in subways and stores without cashiers.

1.3.3 CNNs for Living Object Detection

CNNs are, as mentioned earlier, very powerful tools to use when classifying objects in images. Image classification can be used in several applications, in a wide range of sectors. Alertness monitoring, lane tracking and collision avoidance are just a few examples of where object classification has been used in the automotive sector for some time [46]. There are several applications where classification of living objects can be of great use, for instance pedestrian detection [47].

CNNs can also be used to estimate human poses, [48] shows how a CNN is used on a human pose dataset with promising results. The network successfully estimates most poses, however the network experience some problems when uncommon poses are presented, or the subject limbs are concealed by clothes or other structures.

In another study from 2016, the different network architectures AlexNet, VGGNet, GoogLeNet and ResNet were used to classify animals captured by a low quality camera system. To further increase the performance of these classifiers a hierarchical classification cluster was used where the input initially is classified as either a bird or "other". After this, the "others" are classified as either small or big mammals and so on. With this approach it was shown that good results could be obtained even though images are of low quality and in gray scale. The results also showed that a deeper net increased the accuracy for this task up to a certain limit, after which the accuracy did not improve [49], These results suggests that a CNN with similar structure but more depth (number of hidden layers) does not automatically improve the accuracy of classification.

1.3.4 Image Classification Using Mobile Devices

One major drawback of these high complexity CNNs is the need of large amounts of computing power and memory [36]. In some applications these demands can be met, however there are a number of applications where the computing power and memory are limited, and the complexity of the previously mentioned CNNs are too large. One notable example where this applies is in mobile devices.

To solve this issue, there have been several methods proposed to decrease the complexity while still keeping the high performance. One interesting method recently released is called network slimming, or pruning. [50] suggests a solution able to decrease the memory-usage and amount of computations of the original CNN. This method can be applied to most popular CNNs and does not require any special hardware or software to be implemented. The method succeeded to decrease the model size of the VGGNet with a factor of 20 and also decreased the number of computation operations with a factor of 5, while still maintaining the performance of the original model. The suggested method automatically prunes the insignificant filters of the original model during the training phase, yielding a slimmer model, but with comparable performance [50].

1.4 Report Disposition

The project will be presented in four main chapters on top of the introduction. Firstly, theoretical background information is presented in Chapter 2, followed by a description of the methods used in the project in Chapter 3. The results of the project, i.e. design choices and performance metrics, are presented in Chapter 4 in form of graphs, images and conclusions. Finally, Chapter 5 presents a concluding discussion on the project results.

2
Theory

This chapter aims to provide the reader with the theoretical knowledge needed to understand the methodology and results of the project. The chapter is mainly presenting theory on CNNs, but other important theory concerning the development of CNNs is also presented. The theory is based on literature studies of well-established papers on the subject and knowledge retrieved by the authors of this report during their education.

2.1 Convolutional Neural Networks

A variant of CNNs were introduced already in the 70s [51], back then known as "Neocognitron". However, in 1989 a CNN named LeNet-5 [52] was introduced. This network was constructed to classify handwritten digits, and it laid the foundation regarding how many of the later network structures were to be developed. Over the last decade, the popularity for the CNNs in the computer vision and image analysis area has grown immensely. The key factor for this popularity growth is the proven performance by the CNNs in tasks such as image classification [53] and object detection [54]. The supporting factors which has allowed for the CNNs to achieve this remarkable performance in these fields are mainly the increased availability of annotated datasets and the increased power of GPUs, but also a consequence of new ideas, algorithms and network structures [55]. Below, a brief overview of CNN architectures and building-blocks will follow, as well as details on network training.

2.1.1 Architecture

CNNs are, just as artificial neural networks (ANNs), biologically inspired and the architecture originated from how the visual cortex of the brain is constructed. The architecture of the CNNs can be constructed in several different ways, however they generally consist of various numbers of convolution and pooling layers, combined into different modules, sometimes followed by one or more fully connected layer. It is generally assumed that architectures with filters of smaller sizes stacked on top of each other leads to better performance than a more shallow architecture with wider filters. However, this assumption is still under debate, and research shows that deeper networks are not always performing better than more shallow networks [56].

An example image of the basic CNN architecture can be seen in Figure 2.1. The

information flows in both directions, with one forward pass for inference followed by backpropagation, where the error is fed back to update the weights.

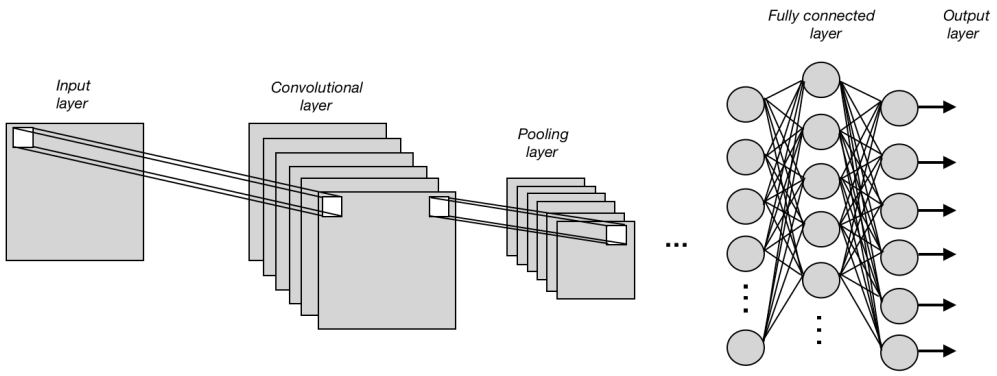

Figure 2.1: An image showing the basic structure of CNNs. The image includes an input layer which in this project will be represented by one image, a convolutional layer with the corresponding filter channels, a pooling layer with it's own filter channels and smaller spatial dimension, and lastly the fully connected layers with predicted probabilities in the output layer.

A CNN consists of large numbers, often millions, of trainable parameters. All these parameters allow for the networks to model in theory any function. The CNNs consist of one input layer, one output layer and one or several hidden layers. In image classification, the entire images are fed to the input layer and conditional probabilities over predetermined classes can generally be extracted from the output layer. The amount of classes in the output layer can be set to fit the problem at hand; if the network is constructed to classify digits, usually 10 outputs are used, but for other problems thousands of classes can be used. Information on the layers in a CNNs follows below.

Convolution Layers

The convolutional layers in CNNs are used to extract features, i.e. lines, edges, blobs etc. Depending on the depth and width of the CNN, it can learn to recognize more complex features, for example eyes, hands or faces. The earlier convolutional layers in the CNN are used to extract the more simple features, while the latter ones are used to extract the more complex features.

The neurons in the convolutional layers are arranged in multidimensional arrays where each neuron has a receptive field. These fields are connected to a subset of neurons in the previous layers via trainable weights [57]. The inputs to each convolutional layer is convolved with these weights, creating new feature maps which are then processed by a non-linear activation function. The non-linear activation function allows the CNNs to extract nonlinear features. Since all feature maps within the same layer is allowed to have different weights, several features can be extracted at each location [57]. The k:th feature map Y_k can be computed as

$$Y_k = f(W_k * x),$$

where x denotes the network input, W_k the convolutional filter, $f(\cdot)$ the non-linear activation function and where $*$ denotes the 2D convolution operation.

Pooling Layers

To reduce the spatial resolution of the feature maps, pooling layers can be applied after the convolutional layers. This operation is used to reduce the amount of parameters preventing undesirable effects such as overfitting and high computational complexity. In the early days of CNNs, it was common practise to use average pooling [58, 59, 60] to reduce the number of parameters. Average pooling computes the average value over a neighbourhood of values, and forwards it to the next layer. In more recent networks [61, 62, 63, 64, 65], max pooling has been used instead. An example of the principle of max pooling can be seen in Figure 2.2.

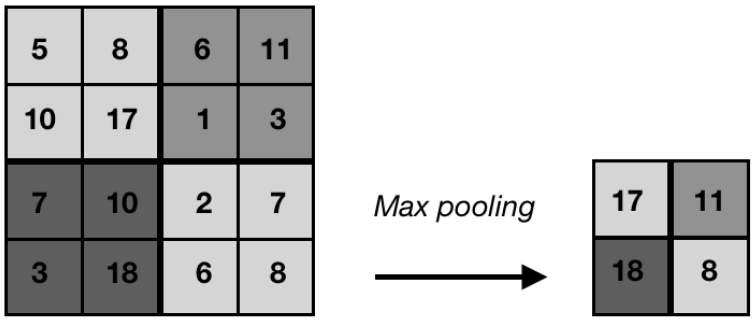

Figure 2.2: An image showing the principle of max pooling, where the max-value of each neighbourhood is saved, shrinking the spatial dimension by half.

Of course, Figure 2.2 only shows an example of the principle. Max pooling can be applied with different sizes of the filter, as well as with different stride lengths.

Activation Functions

Only using linear combinations of convolutional filters in a network to perform classification would require a linear relationship between the input and the output in order to achieve good results. This is usually not the case, instead the relationship is most often non-linear. To enable mapping of the non-linear relationships, activation functions are used. Some activation functions can also be seen as a tool that decides whether a neuron is activated or not, based on the value of the neuron at a specific time.

There are several non-linear activation functions available. Currently, the most prominent and most widely used function is the Rectified Linear Unit (ReLU) [66]. Other activation functions include sigmoids and tanh units [66], among others. The implementation of the ReLU in deep learning networks played a huge role for these networks' success. The ReLU enabled fully supervised training of state-of-the-art networks by reducing computational complexity, and also allowed for a more easy optimization, since gradients are allowed to flow only when the input to the ReLU

is positive [66]. The simplicity and effectiveness of the ReLU has made it the go-to activation function when developing deep neural networks. Looking at the computational cost of the ReLU compared to other activation functions, it is less costly. If x is the input to the ReLU, it simply outputs x, if x is positive. Otherwise, it outputs zero.

Since the ReLU returns zero for negative input values, sparse activation can be achieved, which is not the case for sigmoid or tanh functions. Sparse activation means that not all neurons must be used when processing an image, and leads to a lighter network. The ReLU has several beneficial properties, but it is not perfect in every way. Since the ReLU returns zero for negative input values, it can cause neurons to stop responding to error variations since the gradient will be zero. This is called the dying ReLU problem, and researchers have developed variations of the ReLU to mitigate this issue, one example is the leaky ReLU, which simply outputs a small value instead of zero, when the input is negative [67].

Dense Layers

Dense layers, often referred to as fully connected layers, are usually placed in the last part of the CNNs, just before the output layer. A CNN can be constructed without a dense layer, yielding a fully convolutional network. These networks are more suitable for tasks where the input images are of arbitrary size, compared to CNNs with dense layers where the input images has to be of the same size. The fully convolutional networks consists of learnable filters trough out the network, even the decision-making layer at the end consists of filters.

The standard practise when constructing a CNN is to use one or two dense layers, but more can be used depending on the problem at hand. The dense layers used in CNNs for image classification maps the multidimensional array of data to a 1D array. This array is then used to compute the probabilities for each class. For example, if the task at hand is to classify handwritten digits, and 10 classes are used, the output from the last dense layer will be a 1D array of size 10. To extract the conditional probabilities for each class, it is common practise to use a softmax operator. However, research has shown that applying a SVM instead of a softmax could further improve the classification performance [68, 69].

2.1.2 Before Training

To succeed when developing CNNs, handling the data in a structured way is crucial. Handling the data incorrectly can lead to bad performance or difficulties in interpreting the results. The amount of data needed to train a CNN to achieve high accuracy is large, for some applications thousands or even millions of labeled images are needed to prevent overfitting, i.e. when the model performs well on the training set, but fails to recognize not yet seen images. An overfitting model has learned to recognize e.g. noise or data-dependent features in the training examples, rather than general patterns [70].

Data Augmentation

To increase the dataset, augmentations can be applied to the raw data. There are several transformations available, such as mirroring, rotations, translations and scaling [71]. Even further augmentations are possible by changing the intensities in the RGB channels [72] .To create useful additional data by augmentation it is important to understand which augmentations that will provide realistic data. For example, if the developed system is intended to perform image classification inside a vehicle, an augmentation by large rotations will not generate any useful data, small rotations might be useful though. The same goes for translations. For some applications it could be necessary to perform augmentation during the training phase, meaning that an image is augmented live during the network training. This makes it possible to save a lot of disk space since the augmented images do not have to be stored locally.

Structuring the data

The data used to train and evaluate the models is often divided into three datasets, namely a training set, a validation set and a test set. Several different structures of division is suggested by literature, however, one possible rule of thumb is to use approximately 70% of the data during training, 20% of the data for validation and 10% of the data for final performance tests. It is of great importance that these datasets does not overlap i.e. that the same image does not occur in more than one of the datasets, since this can lead to biased results.

Transfer Learning

As previously mentioned, the amount of available annotated data will greatly impact the result of the network training. A method to increase the amount of available data was presented earlier, utilizing data augmentation. However, in some cases the amount of data can still be insufficient, even when aggressive data augmentation is used.

A widely used method to overcome this issue is transfer learning. Transfer learning is a branch within machine learning, where knowledge from another task could be used to improve the results of the models for the target task [73]. The method is

based on the fact that features learned in tasks similar to the target task can be of use, e.g. if a CNN has learned to recognize apples, this knowledge could also be used to recognize pears, which is a completely different task, but with similarities in image space to the task of detecting apples. [73].

The early layers of the CNNs consists of feature extractors of relatively simple nature, extracting generic features such as blobs, edges and corners. If the available data for the task at hand is limited, it could be seen as a waste to use the task specific data when training a CNN to learn such features since they are present in most other images. A better idea would be to transfer this knowledge from a CNN trained on some other dataset, and use the limited data while training the final layers, which are intended to extract highly abstract and task-specific features [74]. Utilizing transfer learning when developing CNNs has proven to significantly reduce the amount of additional task-specific data needed to achieve high accuracy results in image classification tasks, reduced training times by several orders of magnitude and even eliminating the need for optimizing hyperparameters [74]. Models that have been trained using transfer learning are often refered to as pre-trained models in literature and code-implementations.

2.1.3 During Training

To tune the trainable parameters of the CNNs, optimization algorithms are used. The most common algorithm for this purpose is backpropagation [58]. This algorithm calculates a gradient of an objective function, which is used to determine how to update the learnable parameters in order to minimize the objective function. When training CNNs, there are several important aspects to take into consideration. Two of the most important ones are to avoid overfitting and choosing hyperparameters in a suitable way. These aspects are briefly explained below.

Overfitting

When a network is performing well, only for the images it has been trained on and fails to recognize images never seen before, it is probably overfitting. An example of this behaviour is if the model is containing too many learnable parameters in comparison to the amount of training data. If this is the case, the model can correctly classify all training data by simply memorizing the dataset. CNNs containing large amounts of parameters are keen to overfit, even when relatively large amounts of annotated data is available [75].

To solve the overfitting issue, there are several approaches suggested by literature. These methods include Dropout [76], regularizing the norm of the weights, weight decay [77], and Maxout [78]. A widely used regularizer for CNNs is Dropout. This method attempts to prevent co-adaption of neuron activities, which occurs when two or more hidden units incorrectly rely too much on each other [75].

The Loss Function

The loss function is used to guide the training of CNNs. The aim of this function is to measure how the predictions differ from the ground truth. For image classification, there are two widely used loss functions, presented below.

Mean Squared Error (MSE): MSE, also referred to as the $L2$ Loss is the most common loss function in machine learning. The MSE calculates the loss by taking the average of the squared differences between the ground truth and the prediction. The MSE is calculated as follows

$$MSE := \frac{1}{n}\sum_{t=1}^{n}(y_t - p_t)^2,$$

where y is the ground truth, p is the predicted values and n is the number of samples. The differences between the predictions and the ground truths are squared and summarized for each class and thereafter the average error is calculated.

Cross entropy loss: The cross entropy loss, also referred to as log loss, is also a widely used function when calculating the loss in image classifiers. This function is used when the output from the classifier is a probability between 0 and 1. The cross entropy loss increases in a logarithmic fashion as the distance between the prediction and ground truth increases. The log loss therefore put high penalties on predictions that are confident and wrong. The cross entropy loss is calculated as follows

$$CEL := \sum_{t=1}^{n} -(y_t log(p_t) + (1 - y_t) log(1 - p_t)),$$

where log is the natural logarithm, n is the samples, p is the prediction probability and y is the ground truth (zero or one).

Optimizers

To update the trainable parameters in CNNs effectively, optimization algorithms are used. The most common optimization algorithm is gradient descent, which can be implemented in various ways [79]. When updating the trainable parameters in the CNNs, gradient descent is used to minimize the loss function. This is done by updating the trainable parameters in the opposite direction of the loss function's gradient with respect to the trainable parameters [79]. Some examples of commonly used gradient descent variants are *Stochastic Gradient Descent (SGD), Batch gradient descent* and *Mini-batch gradient descent* [79].

There are several optimization algorithms constructed to further increase the performance of the gradient descent algorithms. These algorithms often aims to decrease the convergence time during training and increase the robustness by avoiding bad local optima [79]. A widely used optimization algorithm is *SGD with Momentum*[80], but there are also many other optimizers available, such as *Adagrad* and *RMSprop* [79].

SGD is keen to oscillate around local optima, not progressing to the good optima in sufficient time. Momentum is a method that accelerate the SGD in the direction towards the global optima. This is done by adding a fraction of the update vector from the last step to the update vector of the current step. The fraction used is usually set to around 0.9. Using momentum can be seen as pushing a ball down a hill, where the ball will gain momentum as it travels down the hill. The momentum method allows for faster convergence and less oscillation [80].

Hyperparameters

When training a CNN, there are some parameters that must be set manually. These parameters are often referred to as hyperparameters, and can greatly affect the success of the training. Below follows a brief description of hyperparameters that are not directly connected to the network design but rather design choices concerning training.

Learning rate: The learning rates determines how much to update the weights in the optimization algorithm in every iteration. There are several options for this parameter, e.g. a fixed learning rate, a gradually decreasing learning rate, an adaptive learning rate or a momentum based learning rate. Every method has its own strengths and drawbacks, and the successfulness of a method is often determined by the chosen optimizer.

There is a trade-off between convergence speed and accuracy when choosing the learning rate. A large learning rate leads to faster convergence, but the optima might not be as accurate as when using a smaller learning rate. However, it is of great importance not to choose a learning rate too big or too small, since this might cause the optimization not to converge correctly. These cases are illustrated in Figure 2.3.

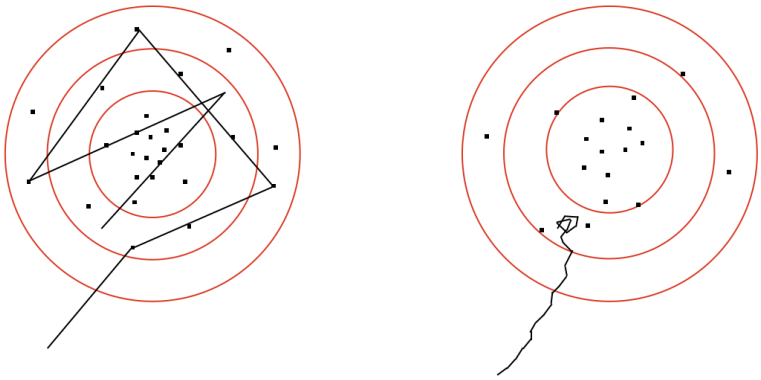

Figure 2.3: An illustration of the effects when the learning rate is chosen too big (left) or too small (right). In the left image, the function struggles to end up at the centre, since the steps are too large. To the right, the function has gotten stuck in a bad optima and would require some larger steps to continue towards the centre.

Figure 2.3 tries to illustrate the optimization problem, where the dots are the dat-

apoints of the chosen dataset. These points determine where the optima for the function will be situated, in other words, where the concentration of dots are as large as possible. In this example this would be the middle of both images. The goal during the training phase is for the optimization algorithm to end up as close to the middle as possible, where the sum of the distances to all datapoints is minimized, hence reaching a good optima.

As seen to the left in Figure 2.3, with a learning rate too large, the function jumps around the good optima, failing to converge. With a learning rate too small, the function can get stuck in a bad optima, as seen to the right in Figure 2.3.

Mini-batch size: When training CNNs it is common to split the training data in mini-batches. Splitting the data will allow for more frequent updates of the weights, which leads to a more robust convergence by avoiding bad optima. Using mini-batches will also allow a more efficient implementation of the training, since all training data does not have to be saved in memory during each iteration [81].

The mini-batch size, often referred to as batch size, is usually chosen between 1 and a few hundred and it is desirable to make sure that each batch represent the entire dataset in terms of class variations [81]. Theoretically, this hyperparameter should only affect the training time, and not the performance of the network [81]. Hence, since the batch can not completely represent the entire training dataset the gradient calculated in the SGD will only be approximative to the true value. The larger the batch size, the better the approximative gradient becomes.

Number of epochs: The number of epochs, i.e. the number of training iterations, is a hyperparameter that can be chosen in a quite simple manner. Since the number of epochs will determine how many times the model will see all the training data, choosing it too high might lead to overfitting [82]. Because of this, one simple way to determine this hyperparameter is to inspect the training accuracy and the validation accuracy. If the training accuracy is improving, while the validation accuracy has stopped improving the training should be stopped, this is sometimes referred to as early stopping [82].

If the validation accuracy keeps improving, the model can be further trained, and the number of epochs is simply limited by the amount of time available for training.

2.1.4 After Training

When the network is trained, it is common to measure the performance using a set of test-images, usually extracted from the original dataset before the training is started. There are several metrics that can be used to evaluate the performance of a classifier. The most used metric for image classification performance is the accuracy of classification, which states the overall effectiveness of the classifier [82].

The accuracy is calculated as

$$\text{accuracy} = \frac{\#\text{Correctly classified samples}}{\#\text{Samples}},$$

where #Correctly classified samples refers to the number of correctly classified images and #Samples refers to the total number of classified images.

This metric is often used in literature to show the performance of a model, however there are several cases where this metric does not state the actual performance in a sufficient manner. An example of this can be observed in large-scale object classification. If there is an abundance of samples which the model is classifying accurately, and substantially fewer samples which the network is not classifying accurately, only measuring the total accuracy can be misleading. In this case, the total accuracy would be high, even if the classifier is not performing well on all classes.

To state the performance in a more sufficient manner, other metrics could be used. Some well known methods are average accuracy per class, precision and recall [82]. Average accuracy per class is simply calculated as ordinary accuracy, but done for every class and averaged over the number of classes present. The precision and recall for binary classification is calculated as follows

$$\text{precision} = \frac{\#\text{True positives}}{\#\text{True positives} + \#\text{False positives}},$$

$$\text{recall} = \frac{\#\text{True positives}}{\#\text{True positives} + \#\text{False negatives}},$$

where #True positives refers to the number of correctly labeled examples in the positive class, #False positives refers to the number of examples incorrectly labeled as positive and #False negatives refers to the number of examples incorrectly labeled as negative.

The precision shows the fraction of the examples which are truly positive among all examples labeled as positive. The recall shows the fraction of examples classified as positive among the number of examples that actually should be classified as positive, indicating how effective the classifier is, when identifying positive labels. In multiclass classification, precision and recall is often calculated for every class [82].

2.1.5 Well Established Network Structures

Several research groups and companies have created network structures with prominent performance in image classification tasks. To measure the performance of the networks, most groups tend to use the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [83]. This competition has been around since 2010 and is now seen as the standard benchmark for large-scale object recognition.

In ILSVRC, the networks are supposed to differentiate thousands of images which are divided into 1000 classes, and the performance of the networks are evaluated by

inspecting *top-1* and *top-5* errors. These errors are computed by feeding the networks with a set of test images to classify. To put it simple, when a network is fed an image, it will return the five classes which has returned the highest probabilities. The *top-1* error shows the percentage of wrongly classified images when only looking at the class with the highest probability, and the *top-5* error shows the percentage of wrongly classified images when looking at the five most probable classes.

Below is a short overview of the network structures which has yielded the most promising performance in this competition.

VGG16 & VGG19

The VGG16 and VGG19 CNN models were developed to investigate the effect of depth of CNNs regarding accuracy on large-scale image recognition. The research group utilized very small (3×3) convolutional filters and found that increasing the depth to 16 or 19 layers yielded significant accuracy improvements. Both the VGG16 and VGG19 models achieved state-of-the-art accuracy results in the ILSVRC 2014 challenge [84].

Ever since these findings, the VGG models have grown popular by researchers and enthusiasts and is as of today two of the most well-known CNNs, with over 4900 hits on Google Scholar. However, literature suggests that the VGG models are clearly over-sized. By inspecting the accuracy per parameter researchers has found that these models are not utilizing their full potential, compared to e.g. ResNet or GoogleNet [85], see Figure 2.4.

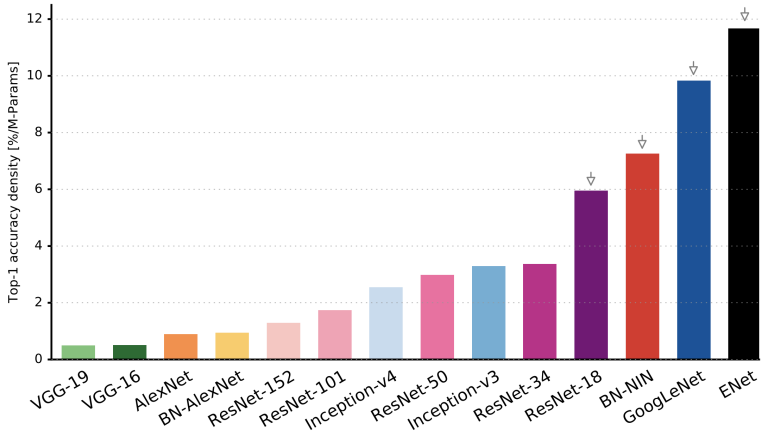

Figure 2.4: Accuracy per amount of learnable weights vs. network architecture. Information density (accuracy per parameters) is an efficiency metric that highlights the network's capacity of utilizing its parametric space [85].

The fact that the models are over-sized, and that relatively large amount of weights are not utilized in a effective manner, suggests that network compression could be applied with successful results. For more information on network compression, see Section 2.1.7.

MobileNet

In several real world applications, such as autonomous vehicles or robots, image recognition is required. The computation resources available in these applications are often limited, and deep network structures can easily lead to a latency which is way to high for the problem at hand [86]. The development team behind MobileNet aimed to create a network structure which performed well on large-scale image recognition tasks, while still maintaining low latency [86].

DenseNet

With technology such as powerful GPUs advancing in a rapid pace, such has the depth of the CNNs. The general impression has over the last decade been that a deeper model with more parameters yields better accuracy results on image recognition tasks. However, as stated earlier, this is not always the case [85]. The research group behind the DenseNet CNN model showed that they were able to achieve state-of-the-art performance while utilizing less parameters than their predecessors [87].

With the introduction of more and more parameters new problems seems to arise, one example is that the information about the input or gradient vanishes while traveling trough the deep networks [87]. The researchers behind the DenseNet model have adressed this problem by providing shorter connections between early and late (shallow and deep) layers of the network, which enables feature-reuse, leading to a more efficient parameter utilization [87].

Deep Residual Learning - ResNets

When training CNNs with deeper structures, they require more computational power. The research group that developed the ResNet models aimed to decrease the computational complexity of deep networks, while still maintaining their accuracy on image recognition tasks [88].

The group showed that it was easier to optimize the training of referenced functions than optimizing the training of un-referenced functions. The achieved optimization allowed them to construct networks with depth up to 152 layers - eight times deeper than the VGG networks, while still maintaining lower computational complexity [88]. This model managed to win the ILSVRC 2015, with a 3.57% top-5 error.

2.1.6 Development Frameworks

Development frameworks are often used to simplify and streamline the development of machine learning algorithms. The frameworks can often reduce the required time for training CNNs, since the training algorithms supplied by the frameworks are optimized. There are several machine learning frameworks available, many of which are specified for certain tasks. This section aims to briefly introduce the frameworks used for this master thesis project. Figure 2.5 shows the amount of Google-searches for some of the most popular frameworks, indicating their respective level of popularity.

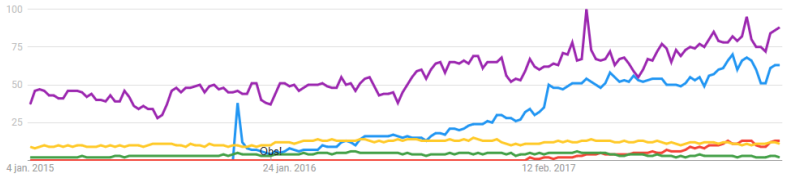

Figure 2.5: Data from Google showing the number of searches for each category: TensorFlow, Caffe, PyTorch,Theano, Keras. The y-axis is scaled to the maximum value of all the presented data, thus representing a percentage [89].

TensorFlow

Google's machine learning framework TensorFlow is one of the most prominent machine learning frameworks today [90]. It allows for high performance numerical computations on a wide range of platforms, from mobile devices to clusters of servers. The framework is supported by Python and GPUs can be used to reduce training times for heavy tasks [91].

Keras

Keras was developed with user-friendliness, modularity, extensibility and Python compatability in mind. The framework minimize the user actions needed for common use cases and allows for CPU and GPU support, mathematical optimization and several useful functions for machine learning development [95].

Keras is the most high-level option of the frameworks in Figure 2.5, and it must run on top of another framework. The choices available for back-end frameworks are TensorFlow, Theano and CNTK [95]. Using a framework with such high-level implementation has several advantages, such as compact code, fast implementation and easy extensibility. However, low-level adjustments as e.g. changing network structures might be more tricky than with other frameworks.

2.1.7 Network Compression

As previously mentioned, CNNs have shown impressive performance in several classification task, especially image classification. However, the CNNs achieving these results usually require powerful hardware, and deploying them on e.g. mobile devices or embedded systems with limited resources might be problematic [96]. The resources that are limited on mobile devices and embedded systems are usually storage space, computing power and battery capacity [96], all of which are required by complex CNNs. As an example, the VGG16 model computes 15.5 million floating point operations to classify one single image [96].

To reduce the resources required to successfully deploy CNNs on mobile devices, several compression approaches have been suggested by literature. The methods are generally grouped into three categories, namely approximation, quantization and pruning [97]. Pruning is a well-studied approach, proven to effectively compress and

accelerate CNNs [96], two aspects highly desired when when deploying CNNs on mobile devices. Pruning have recently shown great success, where CNNs have been compressed without any loss in accuracy [98].

Network Pruning

Pruning can be performed on filter-level or weight-level, both methods yielding different advantages and drawbacks. The commonality between all pruning methods is that less significant parts of the network are removed, to increase the efficiency in terms of speed and memory footprint [97]. Pruning is generally done in three steps, 1: Determining what to remove, 2: Remove chosen weights or filters, 3: Fine-tune model to recover the performance.

To determine which filters or weights to remove, several ranking methods are proposed by literature. Two ranking methods widely used with well-documented results are presented below.

Magnitude-based ranking: Ranking weights or filters based on their magnitude is a relatively easy way to determine what to remove. Literature suggests that weights with small values does not impact the classification performance noticeably [97, 98]. Furthermore, it has been shown that removing parameters based on their values, i.e. removing the smaller ones, yields better results than removing larger or random parameters [99]. However, different suggestions at which threshold this applies can be seen from different research groups.

The magnitude-based ranking can be done both on weight-level and filter-level. For the weight-level, a threshold can be used, and weights with values lower than the threshold is simply removed [98]. This approach leads to a sparse model, with connections removed in several parts of the network. Literature suggests networks with pruned weights instead of whole filters can be harder to accelerate because of the lack of efficient sparse libraries [99].

Pruning entire filters using magnitude-based ranking is similar to pruning certain weights. However, instead of comparing the values of the weights, the absolute sum of all weights in each filter is calculated [99]. These sums are thereafter used to determine which filters to remove. Pruning according to magnitude-based ranking has been successfully applied to convolutional layers, yielding models with smaller memory requirements and less computational complexity than the original model [97, 99].

Entropy-based ranking: The discernment ability of each convolutional filter is closely related to its activation channel [100]. High activation suggests that the filter is important in the classification process, and low values suggests that the filter is not very important [100]. A simple and naive approach to choose which filters to prune would be to calculate the average percentage of zeros (APoZ) in the activation channels, pruning the filters yielding the largest values. However, this approach would miss filters yielding low activation, e.g. 0.001, that could in fact be pruned successfully [100]. Furthermore, this approach would also fail to remove

filters always yielding the same values, which suggests that the filters discernment ability is bad [100].

A better option would be to determine which filters to remove using an entropy-based metric. Entropy is a commonly used metric to measure uncertainty in information theory [100]. Using this metric, filters are pruned according to the entropy value in each activation channel. A low entropy value of the activation channel suggests that the filter corresponding to that channel contains less information, and can successfully be pruned [100].

Strategies Concerning Pruning

There are several different approaches to pruning, both in the aspect of how the parameters of the network are ranked, but also in the aspect of how the parameters are actually removed. One common approach is iterative pruning, where parameters are removed gradually. By conducting iterative pruning, fine-tuning can be performed during the pruning. The fine-tuning allows for the network to regain accuracy between iterations, which might lead to a higher performing model compared to a model where all pruning was done at once [99]. Iterative pruning can be implemented in such a way that parameters are removed globally across the entire network and then fine-tuned, however, literature suggests that removing parameters in a single filter, and then performing fine-tuning might allow for more pruning, while still maintaining the original performance [99].

Determining the number of pruning iterations depends on the desired result. If the goal is to prune the network to a certain size, the number of iterations simply depends on how many parameters that are removed every iteration. If the goal is to prune the network as much as possible, while keeping the same performance as an unpruned network, one common strategy is to continue iterating the pruning and fine-tuning until it is not possible to regain the validation accuracy during fine-tuning.

Iterative pruning allows for a robust way to remove parameters from deep networks, however, the training can easily become very time-consuming, requiring large amounts of training epochs for all fine-tuning [99]. For very deep network structures, the required time might be too long to be feasible, and another pruning approach might be more suitable. An approach suggested by literature is to prune once and then retrain the network. This approach might require less time, however, the chance of recovering the original validation accuracy is lower, compared to the iterative strategy [99].

2.2 Embedded Systems

Electrical technology is today one of the most important building blocks of modern society, empowering everything from construction of houses, to surgeries and making sure that airplanes don't collide on busy airports. It is estimated that 98% of all

microprocessors being manufactured are used as components in embedded systems [101]. Having a complete device or machine, an embedded system often works as a sub-system with a specific task such as the Anti Block System (ABS) of a modern vehicle or the climate control of a house. The embedded system is controlled by sensors, the user and/or the parent-system [101]. By 2006 the embedded systems in a new car contributed to over 25% of the total cost, making it a significant concern to car manufacturers to keep this cost low. The development of these systems are often done by different original equipment manufacturers (OEMs), which put a high demand for interoperability between the different embedded systems of a car [102].

2.2.1 Raspberry Pi

The Raspberry Pi 3B microcomputer is supplied with a 1.2 GHz Broadcom BCM2837 64bit CPU, 1GB RAM and a Broadcom VideoCore IV GPU. Together with a Camera Serial Interface (CSI) for the Raspberry Pi camera-module and a Micro SD-card slot for storing an operating system and data makes the Raspberry Pi one of the most popular micro computers among computer scientists [103].

Raspberry Pi supplies a camera module fully compatible with the Raspberry Pi 3, capable of delivering 3280 $\times$ 2464 pixels for still images with good color fidelity and low-light performance [104]. Other camera options include the Arducam OV5647 capable of still image capture of 2592 $\times$ 1944, which would be a cheaper option [105]. There is also an option to use a USB camera, a choice that would be even cheaper, though this would have an effect on the performance of the system since the proprietary modules (CSI) connect directly to the GPU, making it possible for the CPU to perform other tasks. A USB camera would connect directly to the CPU [106].

The need for CNNs running on portable devices, powered by batteries and low-performance components have made the Raspberry Pi 3 a popular hardware choice when conducting experiments. It has been successfully used in a binary classification task of detecting occupied parking spaces [107] and it has been used to power the vision system of an autonomous model car in real time [108, 109]. The CNNs used for these tasks have been down-scaled versions of AlexNet (5 Layers) [107], DeepPicar (9 layers) [108] as well as GoogLeNet (27 layers), AlexNet (11 layers) and VGG_CNN_F (13 layers) [109]. Furthermore, there have been studies made where this hardware is capable of detecting 5 signs per second in images captured by the Raspberry Pi camera module with a resolution of 1920 x 1080 pixels [110], which indicates that the Raspberry Pi is capable of delivering high performance in image classification tasks, despite the small device footprint.

3
Methods

This chapter aims to present the methodology of the project and it will be structured in a chronological order according to the workflow of the project. The project consists of six main steps, namely:

- A preparation step, where initial choices regarding hardware and software were made.
- A data collection step, during which training data was collected and data augmentation was performed.
- A network development step, during which interesting network architectures were implemented for the task at hand.
- A network evaluation step, where the different network structures were compared.
- A network pruning step, which aimed to slim the developed networks to comply with the limitations determined by the chosen hardware.
- Lastly, a step where a full system test was made as a proof of concept, to verify the achieved results in an online environment.

3.1 Preparations

Before commencing on the actual task of this project, some initial choices had to be made. These choices included which microcomputer and camera to use, which machine learning framework to utilize and the size of the captured images. Poor choices could potentially lead to inefficient implementation and ultimately worse results.

3.1.1 Hardware

The choice of hardware or embedded system to use was crucial, since it would limit which cameras that were available. The camera was the only sensor of the proposed system and there were a large number of viable choices, but for this thesis it was considered of high importance that the chosen hardware was well documented and had proven capable of similar tasks by other research groups. Because of previously mentioned reasons, the Raspberry Pi was chosen.

Since the Raspberry Pi has a proprietary camera module that connects through

CSI, which is not utilizing the CPU, the choice of camera was quite easy. The Raspberry Pi Camera module had previously proven to work well in image classification tasks and online reviews made it clear that this module would be sufficient for our task.

To make attachment of a GoPro-mount possible and to protect the circuits of the system, a protective case was also bought. All components and their respective cost is presented below in Table 3.1.

Component	**Cost [SEK]**
Raspberry Pi 3B [111]	*350*
Camera Module [112]	*200*
Case [113]	*100*
Total:	**650**

Table 3.1: A table showing hardware component costs. Prices from April 2018.

To simplify the process of installing different machine learning software on the Raspberry Pi, an operating system (OS) called Raspbian was installed. This OS supplied a graphical user interface. To control the Raspberry Pi during the experiments, a Secure Shell (SSH) connection was made between the microcomputer and a personal laptop. All images captured using the Raspberry was stored locally.

The computers used to develop, train and evaluate all network structures was provided by the company (Alten) at which the thesis was conducted. The computers supplied were two portable working stations (PWS) from Lenovo. A Linux distribution OS called Ubuntu was installed on one of the PWS, while the other one had Windows 10 OS installed.

3.1.2 Software

In order to guide the direction of the literature studies, decisions were taken early on regarding programming environment and frameworks to use. It was decided that Python would be the programming language for this project, both because it is widely used and that the group members wanted to increase their python programming knowledge.

It was decided that Tensorflow would provide the machine learning framework. The main reason for this choice was the wide range of available articles where Tensorflow had been used, which in combination with the successful use of Tensorflow in various applications made it a feasible choice.

Once the programming language and framework was chosen the group started to look into different implementation approaches available in Tensorflow. Using only Tensorflow could potentially lead to a time-consuming implementation and evaluation phase. Therefore, a more high-level framework was used on-top of Tensorflow, called

Keras, which would later prove to make implementations and changes throughout the project much easier.

3.2 Data Collection

Machine learning requires large amounts of labelled data. Thus, data management was a critical process in the early stages of this project. Data management covers everything from collecting data to downloading, annotating, and in the end, structuring the data in a desired way.

There are several extensive databases available that provide datasets to researchers for free, like the Imagenet [114], CIFAR 10/100 [115] or Kaggle.com [116] databases, just to mention a few. Although these services can provide large quantities of image data for free, there are still many cases where more task-specific image data is required.

In this thesis the goal is to provide a proof of concept, which means that the data used have to resemble the real scenario, i.e. a child in a vehicle. The Imagenet provides about 1000 images of children and infants, but very few, if any, where children or infants are situated in a vehicle or even a CS. Due to this, it became a requirement to collect appropriate data using other sources, first by collecting our own images and later complementing these with images found using Google's image search.

3.2.1 Our Data

To collect our own data, a real scenario had to be mimicked in which the system could be operable. A doll and a CS was bought and a vehicle was rented. The aim of this task was to collect as much data as possible, since it is well-known that machine learning methods benefit from large amounts of data.

The first step in this process was to decide on the camera position inside the vehicle. A desirable solution would be to cover all seats in one image, which would be possible with a camera placed in the middle of the vehicle in combination with for instance, a fisheye lens mounted on the camera. This approach was first considered, but later neglected since the group was doubtful regarding problems connected to fisheye distortion. It was decided that only focusing on the backseats of the vehicle would be sufficient for this proof of concept study, which meant that different mounting positions for the camera could be chosen. As a final solution the camera was placed in a case and a GoPro-mounting mechanism was used to attach the whole hardware piece (Raspberry Pi and Camera Module) to the rear-view mirror.

From the chosen position the group started to collect training data. The images containing the doll were considered "positive samples" while images without the doll were named "negative samples". The positive samples were captured in a structured way with different placement for the arms and legs of the doll, and later repeated

for all three back-seats of the vehicle. The negative samples would require various items to be placed in the seats. Towels, bags, humans and clothes were used. These items were placed in a random fashion to create as much intra-class variation as possible for the negative samples.

In terms of software, a simple photo-script was implemented that would utilize the camera module and save an image of size 224x224 to the on-board SD-card. The user had to chose whether a positive or negative sample was taken and the photo-script would set up the folder structure accordingly.

Factors that were important during this step was to make sure that the image captured the same perspective of the vehicle during different sessions, that the surrounding light conditions would not differ too much and that the correct settings were used by the photo-script in terms of image resolution and positive/negative samples. The doll used when collecting the training data can be seen in Figure 3.1. The clothes on this doll will be referred to as the "old clothes" in several experiments.

Figure 3.1: An example image of the positive samples from the data collection phase.

Figure 3.2: An example image of the negative samples from the data collection phase.

3.2.2 Web Data

To introduce more variation to the dataset it was desirable to collect data from different sources, especially in this case when the own data is relatively similar between samples. By introducing new angles, subjects and illumination conditions the network could be forced to perform classification in a new way, making the model more robust.

During the data collection phase some time was spent online to collect images. The Google search engine was mainly used and then the images were downloaded from their respective site. In this process it is essential to avoid copyright protected images, which put limitations on how many images that could be collected.

3.2.3 Data Augmentation

Since CNNs in general require large amounts of data, different augmentation techniques were used in order to reach sufficient data amounts. Some of these augmentations were performed prior to the training, like skin color changes and color channel flip, but the majority of the augmentations were made during run-time, with Keras built-in functions.

Skin Color Changes

To ensure that the final system was capable of detecting children of different skin colors, the images captured in Section 3.2.1 were edited in a photo editing software to make the skin of the doll darker, see Figure 3.3.

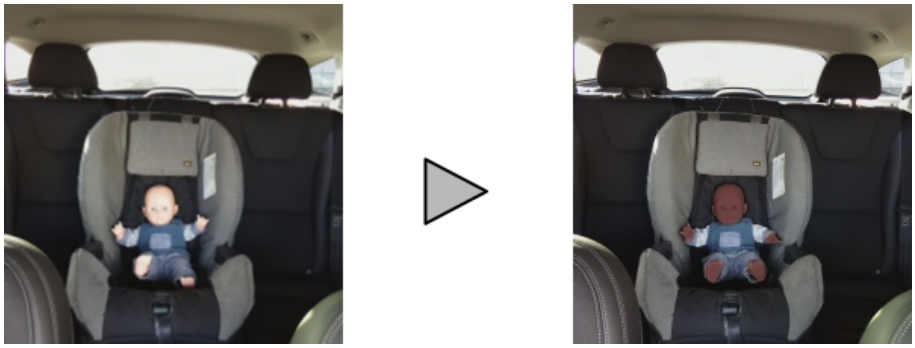

Figure 3.3: Changing the skin color of the CLO in our own images. The skin was made darker, while keeping the rest of the image unchanged.

Color Channel Flip

As a measure to further enlargen the dataset, and to make the system less dependent of certain colors, for instance the colors of the clothes, the group decided to include images with flipped color channels. These images were called RGB-flipped images. The actual pixel values for each channel were kept, but the ordering of the channels in the image representation was flipped. The original image, RGB (Red Green Blue) was changed into RBG (Red Blue Green) and the training data was thus doubled.

Additional combinations of color-channel flips were tested but later neglected since the child would become green or blue for these cases, representing unrealistic scenarios. The RBG-version would make the child appear red, see Figure 3.4. The group decided this was a sufficiently realistic color of a child, hence these images were included in the training data. This RGB-flip kept most of the vehicle interior colors unchanged but altered the colors of the clothes, as seen below. This was a desirable property, since the final model should be able to detect children wearing different clothes.

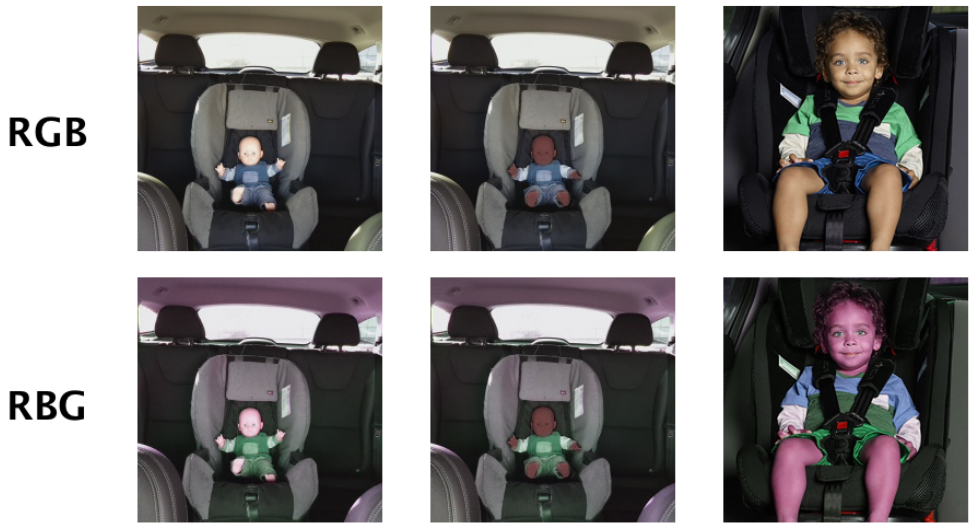

Figure 3.4: Three sample images showing the effect of the RGB-flip on different skin colors. As seen in the images, both the CLO and the child become more red as the color channels are flipped. Also note that the blue and green details in all images have changed places.

Online Augmentations

Keras provide various techniques to perform image augmentation online, meaning that the augmentation is performed as each image is processed by the network. Therefore, several augmentations can be performed without having to store each individual image locally on the computer. The function used was the `ImageDataGenerator`, which is a part of the Keras image pre-processing package [117]. Initially, the data was pre-processed by re-scaling, meaning all intensity values would be scaled by 1/255 to make each number smaller, in range 0-1 instead of 0-255, making it less demanding for our model to process.

The augmentations used were horizontal flip, zoom and shear. As stated earlier, it was of great importance to keep all augmentations realistic, not to introduce strange scenarios that would trick the model.

- Horizontal Flip: Each image can be flipped horizontally since the CLO might be situated at all places in the vehicle.
- Zoom: Each image can be rescaled in order to allow the network to learn recognizing CLOs of different sizes.
- Shear: By applying small amounts of shear, the shape of the CLO could be altered within realistic limits.

These augmentations were applied to all datasets used during the project.

3.2.4 Datasets Overview

During the project, three different datasets have been used. Each dataset served a different purpose and they are all presented in Table 3.2.

Dataset 1 (DS1) was used during the early parts of the project and consisted of images collected according to the procedure in Section 3.2.1. This dataset was mainly used to ensure that the tested CNNs were capable of image classification on an easy test case. The images were all captured in the same vehicle, with similar colors and illumination throughout all scenes. In short, DS1 introduce quite few intra-class variations. For DS1, five different color channel flips were used, i.e. RGB to RBG, BGR, BRG, GBR and GRB

The images obtained online, presented in Section 3.2.2 were added to DS1 resulting in DS2. This dataset share many attributes with DS1, but contain images that should be harder to classify, since the web-based images are highly uncorrelated to our own data. The web data contain images of real children, with varying skin color. Other variations included different clothes, small variations in illumination and different vehicles. This dataset was used to evaluate the CNN performance on harder test cases. For DS2, five different color channel setups were used, i.e. RGB, BGR, BRG, GBR and GRB

To be able to detect children of different skin colors, skin color augmentations were introduced to our own data, see Section 3.2.3. The new images of CLOs with darker skin color were shuffled with the original ones, making it a 50-50 ratio between bright and dark skinned CLOs, introducing more intra-class variations. This led to DS3, which was used for the largest part of the project, since it was considered to cover the most important test-cases. This dataset was used to evaluate the baseline model and pruned models. For DS3, 2 different color channel setups were used, i.e. RGB and RBG. All datasets are summarized below, in Table 3.2.

Data	**DS1**	**DS2**	**DS3**
Training	*3125 (700,2425)*	*3450 (895,2555)*	*1375 (358,1017)*
Validation	*900 (200,700)*	*975 (255,720)*	*400 (102,298)*
Test	*450 (100,350)*	*490 (125,365)*	*196 (50,146)*
Total:	*4475 (1000,3475)*	*4915 (1275,3640)*	*1971 (510,1461)*

Table 3.2: The different datasets used througout the project. The separation ratios were kept around: Training 70%, Validation 20%, Test 10%. Datasets are presented as: #Total (#class 0, #class 1).

Each time new data was added to the datasets it was evenly divided across the training-, validation-, and test-sets. All datasets and their compositions are summarized in the following list:

- DS1: All images from Section 3.2.1 times 5 color channel flips.
- DS2: All images from Sections 3.2.1 & 3.2.2 times 5 color channel flips.
- DS3: All images from Sections 3.2.1 & 3.2.2 times 2 color channel flips, RGB and RBG. In this dataset the skin color changes were introduced, as well as all images from Section 3.2.3.

3.3 Network Development

Constructing a network able to achieve high accuracy classification results from scratch is a tedious process, which is currently being researched by numerous research groups and companies. Spending the majority of the project-time re-inventing the wheel did not seem like a good idea, and instead the focus was put on investigating available architectures with proven performance in image classification tasks. These architectures could be evaluated and thereafter modified to fit the task of this project.

Based on the information retrieved during the literature study, it was clear that developing a network able to achieve high accuracy results with limited data would be greatly simplified by utilizing transfer learning. The literature clearly stated that training networks consisting of large amounts of parameters would be very keen to overfit with limited data. By using the transfer learning approach, the project-specific data could be used to train only the last layers of the altered networks, and other data could be used to train the early layers, preventing the overfitting issue as much as possible.

3.3.1 Determining Architectures

To determine which architectures that could perform well on the task at hand, some different aspects were taken into consideration. The first aspect investigated was how well the architectures had performed on the ILSVRC. If the architectures were able to achieve high accuracy results on large-scale object classification tasks, they should be able to perform well on the task at hand as well. The second aspect was how well the documentation of the architectures was written. Access to reliable information on how the networks were developed could be of great use when investigating the performance. The third aspect taken into consideration was if the architecture could be easily implemented in the chosen framework or not, since this could save several working hours. Based on the previously mentioned factors, the following networks were chosen to be further evaluated: VGG16, VGG19, DenseNet121, ResNet18 and MobileNet.

The networks chosen for further evaluation are all relatively well known and high performing in image classification tasks. The MobileNet network stands out from the rest since it was not developed with the goal of performing as well as possible on large scale object recognition tasks. Instead, it was developed to perform classification as efficiently as possible, enabling deployment on mobile devices. This fact raised interest to include it in the evaluation, since the task of this project might suit a network of such nature.

The other networks were all developed with the aim of performing as well as possible on large scale object recognition tasks. The classification performance of these networks are state-of-the-art, with remarkable results in the ILSVRC. For information on how the networks are evaluated in the ILSVRC, see Section 2.1.5.

3.3.2 Modifying the Architectures

Since the chosen networks were mainly developed for large-scale object recognition tasks, modifications had to be made so that they would suit the task of the project. The network structures chosen for further evaluation returns 1000 probabilities at the final layer. Since the task of this project was to differentiate an empty vehicle from a vehicle in which a CLO is present, the last layer had to be modified to return one single output, stating if a child is present in the image or not.

To accomplish this, the final dense layers from each network were removed and replaced with dense layers more suitable for the problem. The replacement layers developed for this task were kept the same for all network architectures, to allow for fair comparisons. The replacement layer was developed by utilizing the `Sequental`-function in Keras. This function allows the user to create parts of, or even full models, by stacking layers on top of each other. The replacement layer created consisted of two dense layers, the first one with 256 nodes, and the second one with one single node. This allowed for extraction of one single output from the last layer.

The activation function for the first dense layer was chosen to be the ReLU function, based on the information stated in Section 2.1.1. To extract a probability from the last layer, the sigmoid activation function was used. This function is a binary version of the softmax function and thus returns a probability of the output belonging to class 1, i.e. an empty vehicle in this project. When initializing the weights of the replacement layers, bottleneck extraction was used. More information on this may be found in Section 3.3.3.

3.3.3 Training

As previously mentioned, the networks chosen contains large amounts of trainable parameters, requiring large amounts of training data to reach high classification performance. Since the data collected during this project was limited, the network parameters had to be trained using a combination of the collected data from Section 3.2.1, Section 3.2.2 as well as other publicly available datasets, like ImageNet. Even if the public datasets available do not represent the child classification problem at all, it can still be successfully used to pre-train the networks. This approach is common when training networks when the training data is limited, and is referred to as transfer learning, see Section 2.1.2 for more information.

The training of the networks was performed in four steps, see Figure 3.5. Explanations of every step can be seen below.

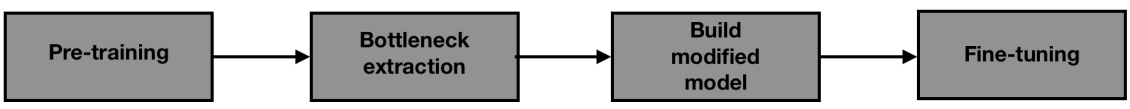

Figure 3.5: A flowchart showing the major steps performed during the network training.

Pre-training

To train the early layers of each network, pre-training was used. The pre-training consisted of training the networks using the ImageNet dataset, containing millions of images divided into 1000 classes. Remember, the early layers of the networks are supposed to extract simple features, present in most images. Training the networks to recognize these simple features using pre-training means that the limited dataset collected for the project can be used to train the layers located later in the network, which are used to extract more abstract and problem specific features.

Training deep networks, such as the ones used in this project, with millions of images requires large amounts of computing power and time. During this project, both time and available computing power were limited, hence training several deep networks from scratch would not be feasible. Luckily, the chosen networks had been trained on the ImageNet dataset by others, and the trained weights were publicly available. Using Keras, the pre-trained ImageNet weights were easily imported to the networks.

Extracting Bottleneck Features & Building Modified Model

As mentioned in Section 3.3.2, the final layers in the CNN were modified in order to output 1 prediction probability, for this binary classification case. A common approach when initializing the parameters of CNNs is to set them with mean 0 and standard deviation 1. This approach can cause problems when utilizing transfer learning, since the gradients might become too large. Instead, to initialize the weights of the modified top layer, bottleneck extraction was used. The bottleneck extraction was performed by feeding the task-specific training images to the pre-trained network architectures with a modified top layer one time. By feeding the training images trough the network one time, the weights in the modified top layer could be updated to better fit the pre-trained network. The updated weights were thereafter stored, and could then be used when initializing the final layers of the modified model.

Hyperparameters and Fine-tuning

The choice of hyperparameters can have a large impact on the training of the CNNs. Since the pre-training was conducted by downloading weights trained by others, the hyperparameters used during that training could not be altered. The choice of hyperparameters for the fine-tuning were mainly based on the theory from Section 2.1.3 and observations made during training. The hyperparameters were changed several times during the fine-tuning stage of the networks, which allowed for investigation on how each hyperparameter affected the outcome. However, the hyperparameters finally used during the fine-tuning of all network architectures were set as follows:

```
Learning rate:0.0001, Optimizer:SGD, Momentum:0.9, Batch size:25
```

The amount of epochs were changed when the dataset used during fine-tuning

changed. When tuning the network architectures on DS1, 20 epochs were used, which allowed for full convergence. Furthermore, when tuning the networks on DS2, 60 epochs were used to allow full convergence. Later during the project DS3 was used, with varying amounts of training epochs. The hyperparameters used were not changed between the different architectures, simply because this could allow for easy comparison.

When the network architectures had been pre-trained and the top layer had been replaced to fit the task at hand, the networks had to be fine-tuned. The fine-tuning consisted of training the networks with the data collected during the data collection phase of the project.

Locking layers: As previously mentioned, the early layers of the CNNs consist of feature extractors intended to extract simple features present in most images. Since the networks are already pre-trained at this point, it can be beneficial to lock these layers [118]. Locking the layers means that the weights in these layers will not be updated during training. By locking layers, the limited data can be used to update only the final layers of the networks. Remember, the final layers are intended to extract more task-specific features from the image. This approach will decrease the risk of overfitting the network, since the amount of trainable parameters will be lower with locked layers. When fine-tuning the networks, the first 15 layers were locked, leaving only the three final convolutional layers and the dense layers unlocked. At a later stage in the project, more layers were left unlocked, more information on this can be seen in Section 3.5.3.

Updating the weights: To update the weights of the networks, the `fit-function` supplied by Keras was used. This function updates the weights, trying to minimize the training loss. To calculate the loss, the binary cross-entropy was used. The training and validation accuracy as well as training and validation loss was continuously saved to a file, this data could thereafter be visualized using `MATLAB`, which allowed for investigation of the training results.

Output from Model

In this thesis project, binary classification is performed, indicating if the CLO is present in the scene or not. The last layer in the chosen framework will output a number between 0 and 1, corresponding to the predicted probability of the input belonging to class 1. The group decided on using the classes as presented in Table 3.3.

Class	**0**	**1**
	CLO	*Empty*

Table 3.3: A table showing the class structure used in this project. In this case "empty" refers to a scene without a CLO present.

3.4 Network Evaluation

To evaluate the performance of the networks, several factors had to be investigated. Firstly, the data saved during the fine-tuning of the network was evaluated. To evaluate this data, it was visualized using `MATLAB`. Important factors here were the training and validation accuracy, the training and validation loss and also whether the models did converge during training or not.

To further evaluate the performance of the models, the test-set was used, information on how these images were collected is presented in Section 3.2. As stated in Section 2.1.4, only investigating the accuracy of classification is in many cases not enough to state how well a model is performing. This applies to the task at hand, since it is of great importance that the models correctly classifies images in which CLOs are present. Therefore, precision and recall were calculated for every class, for information on how precision and recall are defined and calculated, see Section 2.1.4. These metrics would provide a informative measure of the performance of each model.

To calculate and visualize these metrics, a python script was developed, in which each model classified all test-images in the datasets. These classifications were thereafter used to calculate all important metrics. The precision, recall and accuracy were saved in a excel-sheet, which could thereafter be used for evaluation of the performance.

On top of the evaluation measures presented above, the prediction values were visualized on a line-chart using `MATLAB`. This line-chart provided a visualization on how robust each model was while classifying the test-images. Remember, the prediction probability returned from the models after classifying an image is a number between 0 and 1, stating with what probability the image belongs to class 1. Hence, if the prediction probability values are close to 0.5, the model is not classifying the images robustly.

The purpose of the network evaluation was to create a baseline model, representing the architecture that performed the best.

3.5 Network Pruning

To reduce the memory and computing power requirements needed to classify images, pruning was used. The pruning was aimed to compress and accelerate the networks by removing redundant filters. The pruning was done in three main phases, presented in Sections 3.5.2 - 3.5.4. However, all pruning phases utilized the same basic pruning operations, and these operations are presented below.

3.5.1 Pruning Operations

The pruning algorithm implemented for this project utilizes a form of iterative pruning. Redundant filters are ranked and removed in an iterative manner, with fine-tuning of the model between the iterations to regain lost performance. A visualization of the algorithm can be seen in Figure 3.6.

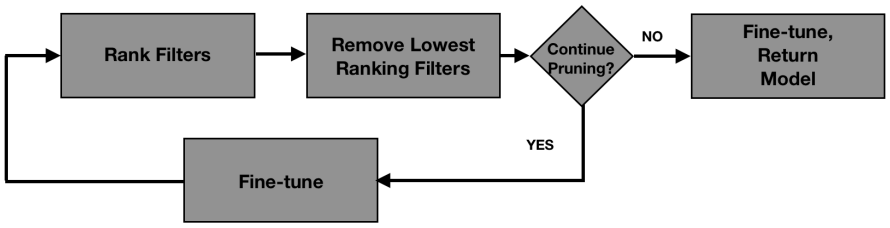

Figure 3.6: A flowchart showing how the pruning was performed, starting from the top left corner. Fine tuning refers to the same fine tuning operation performed in Section 3.3.3.

The algorithm visualized in Figure 3.6 was implemented in Python, utilizing functions provided by the Keras API. The implemented algorithm allows the user to select the amount of filters to remove in each layer by supplying a percentage. For example, if the user selects to remove 1 % of filters on every iteration, the algorithm will rank all filters in each layer independently, and thereafter remove the 1 % lowest ranking filters in each layer. Since CNNs are often constructed with more filters in the later layers, more filters will be removed in the later layers.

Ranking the Filters

The ranking method used in the pruning algorithm is magnitude based ranking, presented in Section 2.1.7. The filters in each layer are ranked without respect to other layers. As stated in Section 2.1.7, this ranking method is quite simple, only ranking the filters based on the absolute sum of the weights in each filter, which allows for a fast and efficient ranking.

Removing the Lowest Ranking Filters

When the ranking of the filters is completed, the algorithm removes the lowest ranking filters in each layer according to the supplied percentage by the user, meaning the lowest absolute sum. The algorithm keeps track of the indexes of the removed filters, since the corresponding input feature maps has to be removed from the subsequent layer for the network to function properly. For example, if the number of outputs from layer N is 256, the number of inputs in layer $N + 1$ must be 256 as well.

Determining the Number of Iterations

The number of pruning iterations are obviously determined by how much pruning the user wants to achieve, and how much pruning the user wants to perform in

every iteration. Optimally, the algorithm should remove only one filter in each iteration, this approach would maximize the amount of pruning achievable while still keeping original performance. However, this approach would be very time-consuming. Hence, there will be a trade-off between the time-consumption and the amount of pruning achievable. The smallest amount of filters removed in every iteration during this project was set to 1 %, or at least one filter if the amount of filters were below 100. The choice of removing 1 % of present filters in each iteration instead of removing one filter in each iteration was based on time-constraints.

3.5.2 Phase 1 - Determining Amount of Pruning

To evaluate at which point a degradation in classification performance would occur, the baseline model was pruned in steps of 5 %. Setting the incrementation at 5 % would allow for a relatively fast way of finding the degradation point.

All models were trained with equally many training epochs, to enable comparison. The way each model was trained is presented in Table 3.4.

Model	**Training Scheme [Epochs]**
0%	*60 = 60*
5%	*30+30 = 60*
10%	*20+20+20 = 60*
15%	*15+15+15+15 = 60*
20%	*12+12+12+12+12 = 60*
25%	*10+10+10+10+10+10 = 60*
30%	*9+9+9+9+9+9+9 = 63*

Table 3.4: A table showing how the training was performed during Phase 1. During this phase, all models were trained for approximately 60 epochs to ensure proper comparison. 5% pruning was added for each iteration, except the first one that was necessary for bottleneck extraction. For example, the 15%-model was pruned 0+5+5+5=15%.

The pruned models were thereafter evaluated in a similar manner as seen in Section 3.4. If the pruned model performed significantly worse compared to the baseline model, it is considered to be pruned too much.

3.5.3 Phase 2 - Fixed Training Scheme

A conclusion drawn from the results presented in Section 4.2.1 was that more training epochs between each pruning iteration were needed to ensure full convergence. Based on the earlier mentioned results, it was also decided that the final fine-tuning of the model needed more training epochs as well. This led to a new training scheme, presented in Figure 3.7.

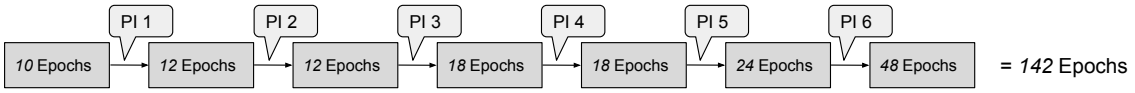

Figure 3.7: A table showing how the training was performed during phase 2. PI - Pruning iteration. This fixed training scheme was used for training each model from this point onward. In total 142 training epochs and 6 pruning iterations, where the actions showed in Figure 3.6 could be performed.

The decision to choose this particular training scheme was based on discussions with the project supervisor as well as trial and error. Some different versions were tested that would later prove provide insufficient time for convergence during the last training iterations.

With the introduction of a new training scheme for all models, different pruning levels had to be achieved in a slightly different way than previously. By pruning 1% at each pruning iteration a total 5%-model would be achieved. To clarify, as seen in Figure 3.7, 6 pruning iterations are performed for each model (the first 10 epochs are used for the bottleneck extraction). To achieve a 5%-pruned model, 1% of the available filters are removed at each PI. For the 10%-model, 2% of the available filters are removed at each PI. This setup means that it is not possible to achieve the exact pruning-amount that each model name refers to. For further explanation, see Table 3.5

Model	**PPI**	**TPF**	**Real Percentage**
0%	*0%*	*0*	*0%*
5%	*1%*	*198*	*4,9%*
10%	*2%*	*450*	*10,6%*
15%	*3%*	*666*	*15,7%*
20%	*4%*	*881*	*20,8%*
25%	*5%*	*1088*	*25,7%*
30%	*6%*	*1275*	*30,1%*

Table 3.5: A table showing amount of pruning per iteration, amount of pruned filters and how much each model is actually pruned. PPI - Pruning Per Iteration, TPF - Total amount of Pruned Filters. The total number of filters in the baseline model is 4224, hence #Pruned Filters/4224 = Real Percentage. This table is based on results from Chapter 4.1.

The original model names were kept to ensure readability.

By evaluating the results from the first pruning phase, which indicated that removing filters might have a regularizing effect on the models, led to discussions regarding unlocking more filters. The baseline model was since the beginning of the project kept locked for the first 15 layers to reduce the risk of overfitting. Together with the project supervisor it was decided that each of the available models should be trained with 10 and 15 layers locked respectively. To clarify, after completing

this phase there were 14 models up for comparison, two of these were the baseline with 10 and 15 layers locked respectively. From phase 2 onward, these models are denoted as "10-Locked models" and "15-Locked models".

3.5.4 Phase 3 - Comparison

The results of the second pruning phase, presented in Section 4.2.2 showed that 6 models performed well on the images in the test-set. These models were all chosen to be further evaluated and compared using a full system test, i.e. a test where the models would be deployed on the embedded system. A small studio where the full system test could be performed was built in the office. This studio would allow for controlled conditions, e.g. good lightning. The studio can be seen in Figure 3.8.

Figure 3.8: An image showing the studio in which the model comparison was made. The models were run on the Raspberry Pi at the centre of the image. The screens were used for reading results from the terminal. The CLO was placed in the seat placed on the floor.

The first step of the full system test was to decide which scenes that would be interesting to test. To help create these scenarios, new items and doll clothes were obtained. This meant that it was possible to construct scenes with, for the models, unknown items. The different scenes are presented in Table 3.6.

Scene	Positive	Scene	Negative
1	Doll, no clothes, CS	21	Empty CS
2	Doll, no clothes, new seat	22	CS, old clothes
3	Doll, old clothes, CS	23	Empty CS, down left
4	Doll, old clothes, new seat	24	Empty CS, top right
5	Doll, new clothes, CS	25	Empty, new seat
6	Doll, new clothes, new seat	26	New seat, old clothes
7	Doll, body covered, head visible, CS	27	Empty CS, rotated 45
8	Doll, body visible, head covered, CS	28	CLO, old clothes, CS
9	Doll, half covered from head to feet, CS	29	CLO, new clothes, CS
10	Doll, objects, CS	30	Bag, Henrik, CS
11	Doll, toy, CS	31	Bag, Victor, CS
12	Doll, standing in CS	32	Plastic bag, CS
13	Doll, no clothes, white covered CS	33	Skin-colored object, CS
14	Doll, old clothes, white covered CS	34	Victor head, CS
15	Doll, new clothes, white covered CS	35	Henrik head, CS
16	Doll, no clothes, CS, CS-rotate 45	36	Umbrella, CS
17	Doll, no clothes. CS, Doll-rotate 180	37	Victor, new seat
18	Doll, no clothes, CS, Low light	38	Henrik, new seat
19	Doll, no clothes, CS-move top right	39	Towel, new seat
20	Victor, new seat, holding Doll	40	Towel, CS

Table 3.6: A table with descriptions on scenes 1-40. CS - Child Seat, CLO - Child Like Object. Doll refers to the same doll used in Section 3.2.1. Images of all scenes can be found in Appendix A.1.

The tests were performed in the following way: one of the scenes above was prepared and the full system, visible in Figure 3.8, captured the scene in one image. This image was then fed as input to all tested models sequentially; when one model finished, the next model started. With this setup it was ensured that all models were given the exact same input, hence their respective outputs could be compared.

Score

Throughout testing, the group realized that picking one model as a clear winner in this phase would be impossible, and that some scoring metric had to be used to evaluate the performance of each model over all 40 scenes. It was argued that a score should favor the models capable of making very robust predictions, and penalize the models that were not as robust.

A score was introduced, which was calculated in the following way. Remember, the positive samples correspond to class 0, meaning an optimal class 0-predicted probability would be 0. For class 1 the optimal predicted probability would be 1. This meant that for the first 20 scenes the absolute value between the predicted probability and 0 was calculated, and for the last 20 scenes the absolute value between the predicted probability and 1 was calculated. These distances represent how far from the optimum each prediction was, hence these distances could be sum-

marized over all 40 scenes, for each model individually. The score was calculated as

$$\text{score} = \sum_{n=1}^{20} 0 + \text{predicted probability(n)} + \sum_{n=21}^{40} 1 - \text{predicted probability(n)},$$

where predicted probability is the output from the model, ranging from zero to one and n is the scene number, taken from Table 3.6. This function is similar to the loss-function, but implemented in a simplified fashion for use during the online-tests, where the original loss-function was not available in the developed environment.

Execution Times

As a way to measure system load for each model, two different times were measured. Time for loading each model individually, as well as classification time of one image. Over all 40 scenes the average times were calculated. The actual loading and classification times are not that important for the application in this project, but were mainly used as a measure of the load put on the Raspberry Pi. A shorter time would indicate that less powerful and cheaper hardware could manage the same classification task.

3.6 System Test - Proof of Concept

With each pruned model thoroughly compared and evaluated, one model was considered the best and was used in the final system, as a proof of concept of this project.

With the promising results shown during the earlier stages, it was desirable to make the proof of concept study a bit more demanding for the system, to discover a range of areas where the developed system would fail the classification, and also to discover scenarios where the so far presented performance actually could be achieved. This was done by performing tests in a new, unknown vehicle and by introducing a new doll, with dark skin and new clothes.

The scenes used for this final test are presented in Table 3.7.

Scene	Description	Scene	Description
41	Doll 1, CS, Middle	51	CS, New Clothes
42	Doll 2, CS, Middle	52	CS, Object
43	Doll 1, CS, Left	53	Doll 1, CS, Covered Body
44	Doll 2 + Objects	54	Doll 1, CS, Covered Head
45	Doll 1, CS + Object	55	Doll 1 + CS + Object
46	Doll 1 + 2, CS + Object	56	Doll 1 + Empty + Object
47	Empty	57	Human
48	Old Clothes	58	Doll 1 + Human
49	CS, Empty	59	Objects
50	CS, Old Clothes	60	Doll 1 + Objects

Table 3.7: A table showing descriptions for each scene used during the proof of concept stage. Images of all scenes are found in Appendix A.3.

The study was conducted in daylight using the same photo script that had been used in the studio experiments, although this time with one model only. The scenes were presented and the prediction probabilities saved in a text-file. The new vehicle introduced at this stage was different from the vehicle used during data-collection in two main aspects. Firstly, the interior was brighter in color and had a pattern, in addition the light-conditions were somewhat different. The new CLO had darker skin color, a somewhat different facial expression and clothes of other colors.

The hardware was mounted in the vehicle in a similar fashion to when the data was collected and a laptop was used through SSH to control the system. As a final note, it should be said that the system was quite sensitive to changes in perspective, which led to many problems and strange results until it was mounted correctly.

4
Results

In this chapter, the results are presented in a chronological fashion, with an ordering determined by the methodology of the project. The final conclusion of the project is based on several important results achieved along the way. An overview of the content of this chapter can be seen in Figure 4.1, this overview aims to help the reader understand how and why the results are structured in the way they are.

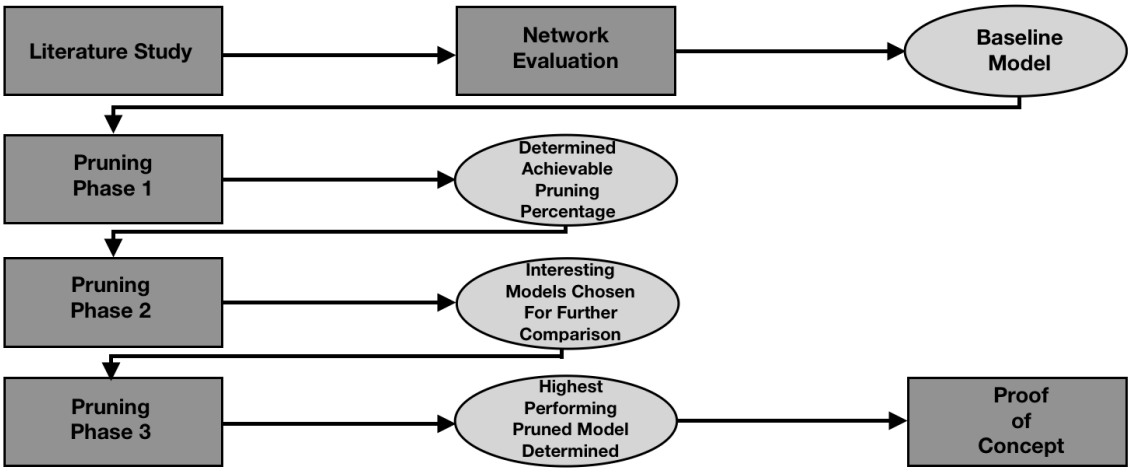

Figure 4.1: An image showing an illustrative flowchart of the results presented in this chapter.

As seen in Figure 4.1, the project consists of several phases which all lead to important results. The final conclusion of the project is based on the last phase, i.e. the proof of concept phase.

4.1 Network Evaluation

Based on the literature study, five network architectures were chosen to be further evaluated, VGG16, VGG19, MobileNet, DenseNet121 and ResNet18. For information on why these network architectures were chosen, see Section 3.3.1.

Figure 4.2 shows the accuracy and loss, evaluated on both the training and validation data in DS1. All network architectures show high performance, with fast convergence.

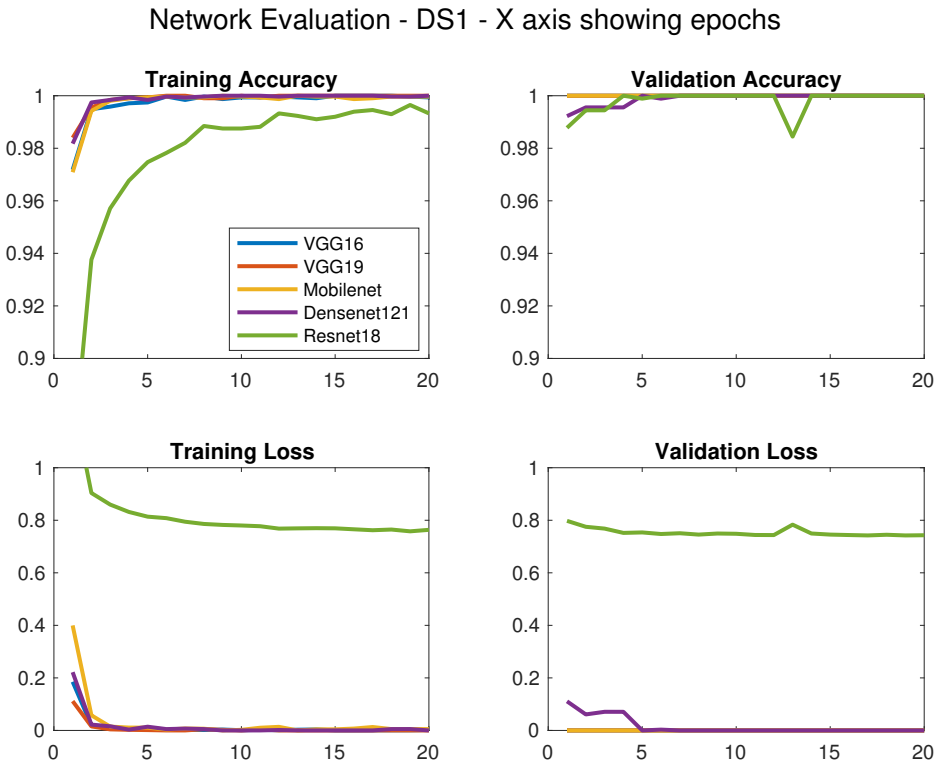

Figure 4.2: Graphs showing training and validation losses and accuracies of the different models evaluated on DS1

During the training phase each model shows rapid increase in training and validation accuracy, the exception being Resnet18 that seem to require more training than the other models. Table 4.1 clearly shows that all network architectures are able to classify the test images in DS1, with all models reaching top-scores and zero miss-classifications. This result is not that surprising since the images in DS1 are very similar and do not include much intra-class variations, others than those manually introduced when collecting the data.

Model	Dataset	Param.	Pr. 0	Pr 1	Re. 0	Re. 1	#Wrong
VGG16	DS1	13M	*1,00*	*1,00*	*1,00*	*1,00*	*0*
VGG19	DS1	26M	*1,00*	*1,00*	*1,00*	*1,00*	*0*
Mobilenet	DS1	16M	*1,00*	*1,00*	*1,00*	*1,00*	*0*
Densenet121	DS1	19M	*1,00*	*1,00*	*1,00*	*1,00*	*0*
Resnet18	DS1	11M	*1,00*	*1,00*	*1,00*	*1,00*	*0*
VGG16	DS2	13M	*1,00*	*1,00*	*1,00*	*1,00*	*0*
VGG19	DS2	26M	*0,97*	*1,00*	*1,00*	*0,99*	*4*
Mobilenet	DS2	16M	*0,96*	*1,00*	*1,00*	*0,99*	*5*
Densenet121	DS2	19M	*1,00*	*1,00*	*1,00*	*1,00*	*0*
Resnet18	DS2	11M	*0,89*	*0,99*	*0,97*	*0,96*	*19*

Table 4.1: A table showing the results from the model tests during the evaluation phase. The table includes number of trainable parameters (Param.), precision for class 0 & 1 (Pr. 0 & Pr. 1), recall for class 0 & 1 (Re. 0 & Re. 1) and the amount of incorrectly classified images (#Wrong).

To further evaluate the models, a dataset with more intra-class variations was used, referred to as DS2. Information on how more variations were introduced can be found in Section 3.2. Furthermore, literature studies conducted prior to the network evaluation suggested that 20 training epochs might not be enough for all models to fully converge on DS2, since the dataset poses a more complex classification task. For this reason, the second evaluation used 60 training epochs on DS2, yielding the results found in Figure 4.3.

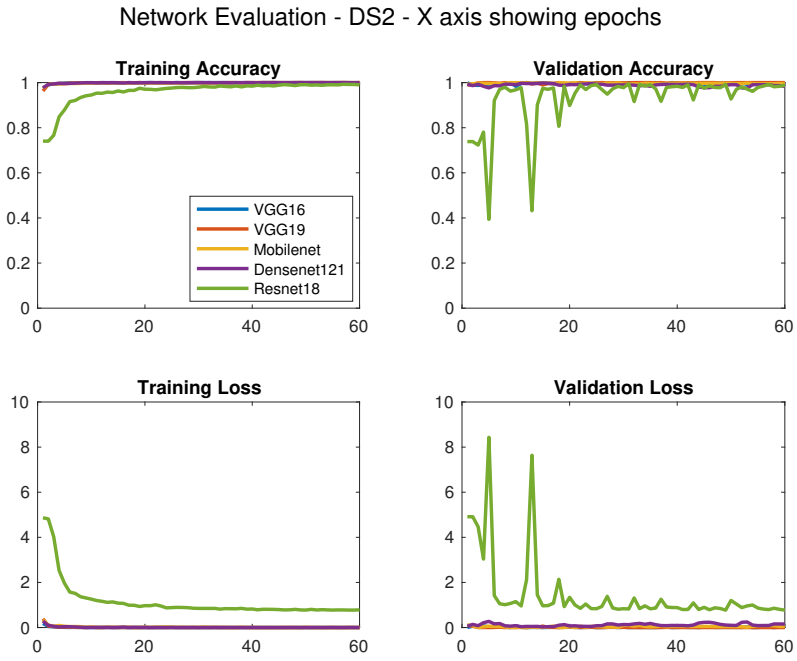

Figure 4.3: Graphs showing training and validation losses and accuracies of the different models evaluated on DS2

The results from the evaluation on DS2 can be seen in Figure 4.3 and Table 4.1. It is evident that four of the network architectures achieve high training and validation accuracies even here. As seen in Table 4.1, the higher amount of intra-class variations seems to affect some of the architectures. VGG16 and Densenet121 seem to handle DS2 very well, while the other architectures show worse performance for the test images. Figure 4.3 might indicate that the implementation and training of Resnet18 was not performed in a proper manner, resulting in relatively high losses and unstable validation metrics, which would also explain the high amount of miss-classified test-images.

The overall results from the evaluation phase led to a decision regarding which network architecture to investigate further. ResNet18, VGG19 and MobileNet were excluded based on their test results presented above. Both Densenet121 and VGG16 showed promising results during the evaluation, however, it was argued that VGG16 was considered the most interesting choice, simply based on the fact that VGG16 contains a smaller amount of parameters than DenseNet121. The smaller amount of parameters suggests that it will be more efficient when classifying images on mobile devices, both when it comes to the memory aspects as well as computing power aspects. To conclude, VGG16 was chosen as the baseline architecture for the remainder of the project.

4.1.1 Baseline Model - VGG16

The baseline model was further evaluated using DS3, which is the dataset that was used for the remainder of the project. This datset introduces even more variations, for more information, see Section 3.2. Table 4.2 shows the performance of the baseline model on DS3.

Model	**Dataset**	**Param.**	**Pr. 0**	**Pr 1**	**Re. 0**	**Re. 1**	**#Wrong**
Baseline	DS3	13M	*0,98*	*1,00*	*1,00*	*0,99*	*1*

Table 4.2: A table showing the results of the baseline model evaluated on DS3. The table includes number of trainable parameters (Param.), precision for class 0 & 1 (Pr. 0 & Pr. 1), recall for class 0 & 1 (Re. 0 & Re. 1) and the amount of incorrectly classified images (#Wrong).

These test results act as a reference from now on. Since the next challenge is to prune the baseline model while still keeping classification performance the mentioned results will be of great importance.

To further visualize the performance and robustness of the baseline model, all predictions of the test data from DS3 were plotted on a line-chart. Remember, the model performs binary classification where each prediction is a number ranging from 0 to 1. The predicted value is the certainty of the image belonging to class one. Simply speaking, a prediction of 0.5 would imply that the model does not have a clue about how to classify the image. The prediction robustness of the baseline model is presented in Figure 4.4.

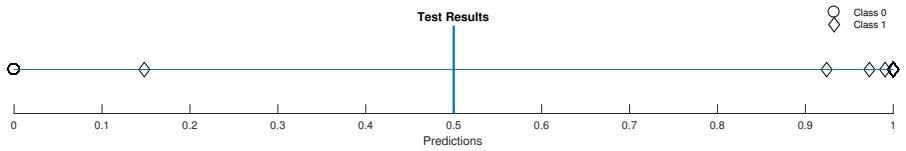

Figure 4.4: A line chart visualizing the baseline prediction probabilities on the test images from DS3 (196 images) Many prediction probabilities are stacked on top of each other, hence all can not be seen.

Figure 4.4 clearly shows that the baseline model is robust when classifying the test images in DS3. All correct classifications are of high certainty, since the predictions are either close to zero or close to one. However, the incorrectly classified image is classified with a quite high certainty as well, but for class zero instead of one. The incorrectly classified image is shown in Figure 4.5, and it is hard to tell why the model fails here, since the model is able to correctly classify the color channel flipped counterpart, which is shown in Figure 4.6.

Figure 4.5: An image showing an incorrectly classified image by the baseline model. This image is the RBG-version.

Figure 4.6: An image showing a correctly classified image by the baseline model. This image is the RGB-version.

4.2 Network Pruning

Pruning can have several effects on the performance of CNNs and it is hard to prematurely determine exactly how it will affect the task at hand. As previously mentioned, the pruning in this project was divided into three main phases, the results of these phases are presented below. For more information on how each phase was conducted, see Section 3.5. From this point on, models will be referred to as a percentage, representing the amount of filters that have been removed from the baseline model.

4.2.1 Phase 1 - Determining Amount of Pruning

The baseline model was pruned in steps of 5%. This pruning incrementation allowed for investigation on how the amount of pruning affected the performance. This phase resulted in seven different models, i.e. one baseline model and 6 pruned models, ranging from 5% to 30 % pruned filters. The baseline model will hereafter be referred to as 0%. The training and validation results of each model is presented in Figure 4.7.

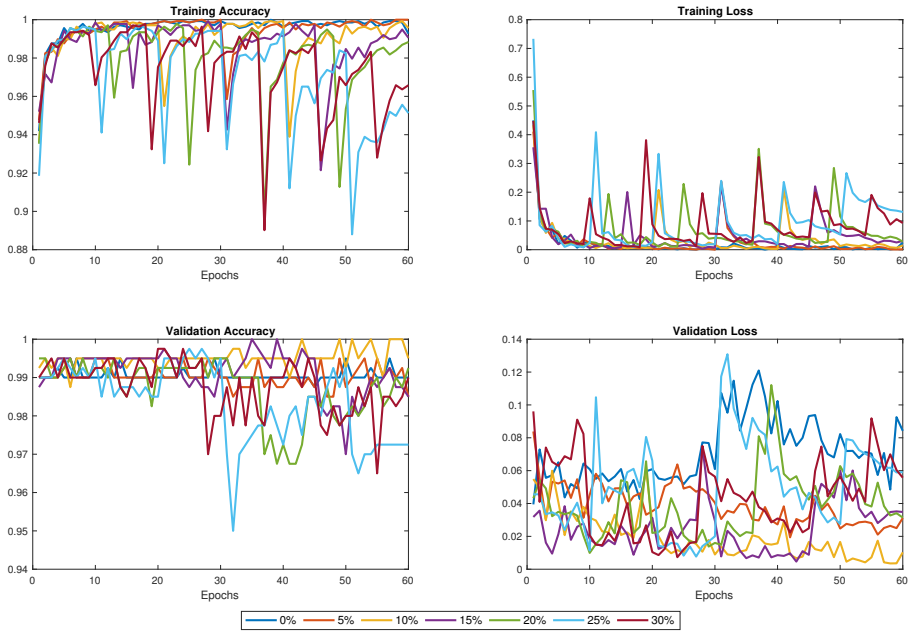

Figure 4.7: Graphs showing training and validation losses and accuracies of the pruned baseline model evaluated on DS3. The baseline model was pruned by removing different amount of layers to evaluate when an accuracy degradation would occur.

The results confirm the conclusions from Figure 2.4, i.e. that the VGG16 model contain a large amount of redundant parameters. The models pruned up to 20% show promising accuracies and manages to keep the loss relatively low during training. The validation metrics look more messy, but does still exhibit high accuracy and low loss. Furthermore, a clear decrease of the training accuracy is visible around 25 - 30% pruning, which could indicate that this threshold might be the pruning limit which guarantees a preserved accuracy.

Figure 4.8 clearly shows how the pruning affects the ability to differentiate the classes, with the predictions moving closer to 0.5 when the pruning increases. This is completely logical, since the removal of more filters lead to less features, which might ultimately lead to worse predictions.

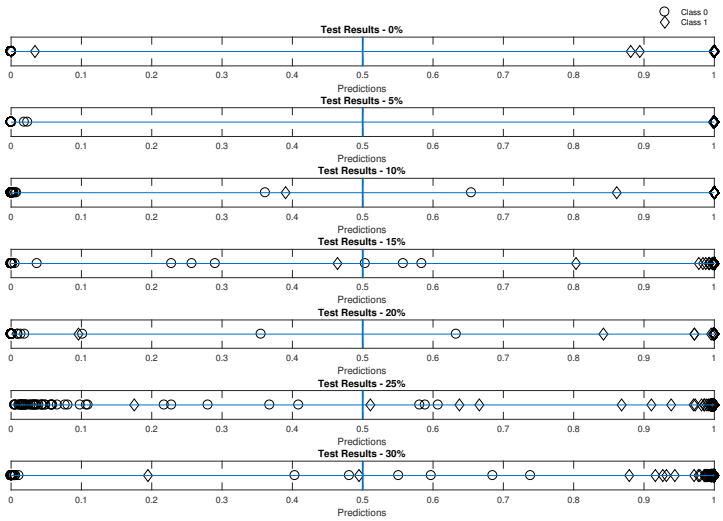

Figure 4.8: Line charts visualizing the prediction probabilities from the pruned baseline model evaluated on the test data in DS3.

Surprisingly, the 30% model seem to perform better than the 25%-model, both when considering training and validation metrics, as well as test predictions. Especially when investigating the test-predictions in Figure 4.8 it becomes clear that the 30%-model is able to more robustly separate the two classes.

As seen in Table 4.3, the file size needed to store the model decreases as the the amount of pruning increase, but is not linearly correlated to the amount of parameters in each individual model. For instance, the first pruned model has lost 5% of the initial parameters, but have been reduced 7.77% in file size.

Pruned model	**0 %**	**5 %**	**10 %**	**15 %**	**20 %**	**25 %**	**30 %**
File Size [Mb]	*135,39*	*124,86*	*115,11*	*106,15*	*97,88*	*90,62*	*83,94*
Reduction [%]	*0*	*-7.77*	*-14.98*	*-21.60*	*-27.70*	*-33.07*	*-38.00*
Accuracy	*0.99*	*1*	*0.99*	*0.98*	*0.99*	*0.98*	*0.97*
Precision 0	*0.96*	*1*	*0.98*	*0.98*	*0.98*	*0.98*	*0.96*
Precision 1	*1.00*	*1*	*0.99*	*0.98*	*0.99*	*0.98*	*0.97*
Recall 0	*1.00*	*1*	*0.98*	*0.94*	*0.98*	*0.94*	*0.92*
Recall 1	*0.98*	*1*	*0.99*	*0.99*	*0.99*	*0.99*	*0.99*
# Wrong	*2*	*0*	*2*	*4*	*2*	*4*	*6*

Table 4.3: A table showing total file sizes, file size reductions and test metrics for each pruned model. The last row indicates amount of incorrectly classified images in the test-set of DS3.

To conclude, the first phase of pruning indicates that there is a significant decrease in classification performance when more than 20% of the convolutional filters have been removed from the baseline model. Figure 4.7 indicates that all pruned models have yet failed to converge, suggesting that more training epochs could increase the

classification performance. It can also be clearly seen in Figure 4.7 that the training and validation accuracy decreases more when the pruning increases, which ultimately suggests that more training epochs could be needed to achieve full convergence at higher pruning levels.

4.2.2 Phase 2 - Fixed Training Scheme

As presented in Section 4.2.1, removing more than 20% of the convolutional filters from the baseline model leads to a significant decrease in classification performance. As previously mentioned, the reduction of classification performance by the models pruned more than 20% could well be due to the fact that those models have failed to converge during training, suggesting that more training epochs could lead to improved results. The new alternative training method developed, presented in Section 3.5.3 led to the results seen in Figure 4.9 and Figure 4.10. Note that there are 14 models presented in this section due to each model having 10 and 15 layers locked respectively, more information on this can be found in Section 3.5.3.

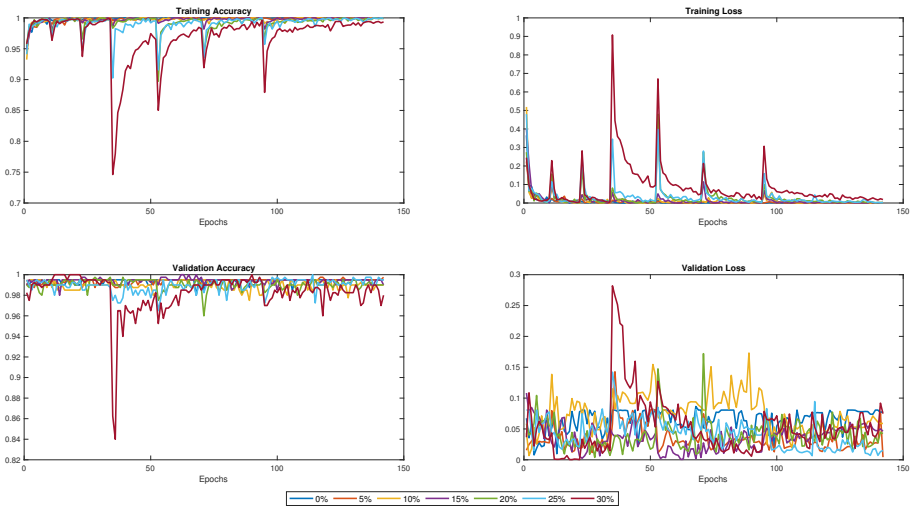

Figure 4.9: Graphs showing training and validation losses and accuracies of the pruned baseline models with 10 layers locked evaluated on DS3.

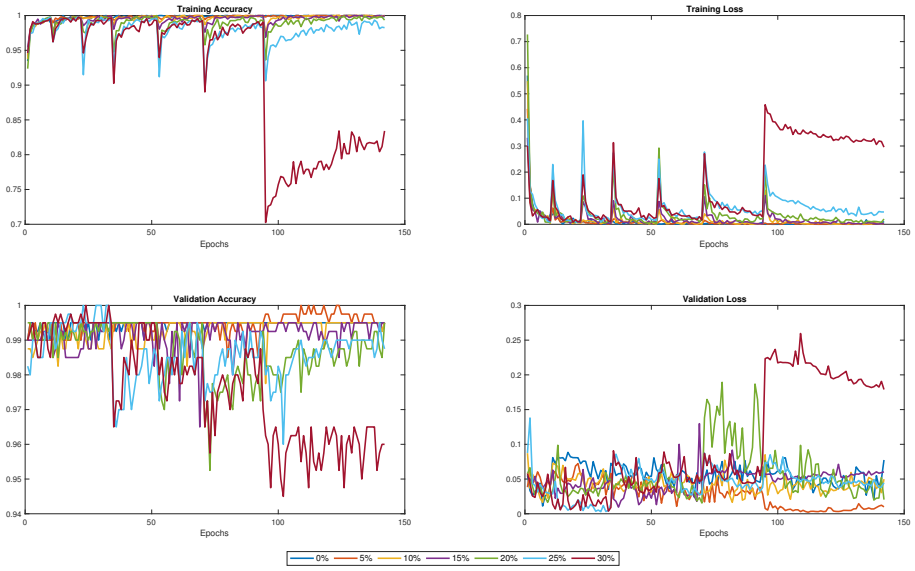

Figure 4.10: Graphs showing training and validation losses and accuracies of the pruned baseline models with 15 layers locked evaluated on DS3.

Figure 4.9 and Figure 4.10 clearly shows that the training loss for most models converge towards zero in the end of each pruning iteration. The only model not converging close to zero is the 30% pruned model with 15 locked layers. This is not surprising since this model have the highest amount of removed convolutional filters and also most locked layers - which means that this model has the lowest amount of trainable parameters.

The models with only 10 convolutional layers locked show better training and validation performance compared to the models with 15 locked convolutional layers. Remember, the models with only 10 layers locked contains more trainable parameters than the models with 15 layers locked. Hence, this result is not very surprising, since the models with more parameters can extract more features.

The robustness of all models is visualized in Figure 4.11 and Figure 4.12. These line-charts clearly shows that the models with only 10 layers locked classifies the test-images in DS3 with higher robustness than the models with 15 layers locked. The models with only 10 layers locked manages to classify the test images well even when they are pruned above 20%.

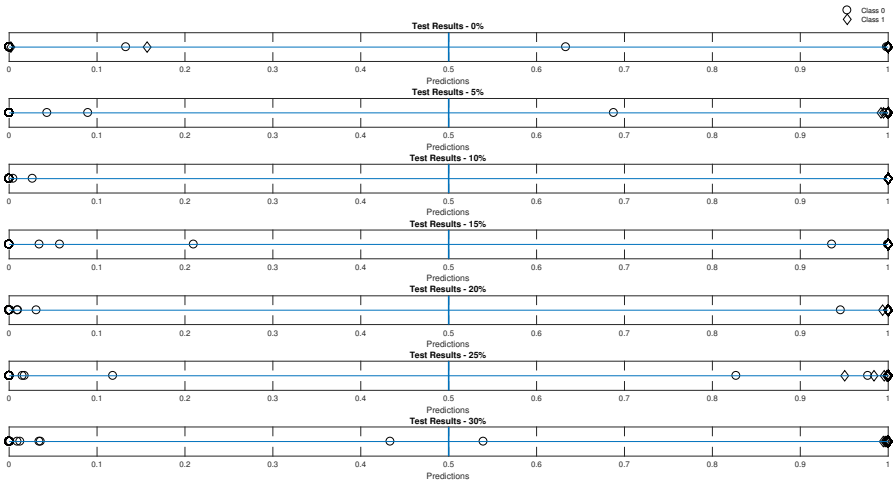

Figure 4.11: Line charts visualizing prediction probabilities from the pruned baseline models with ten locked layers evaluated on the test data in DS3.

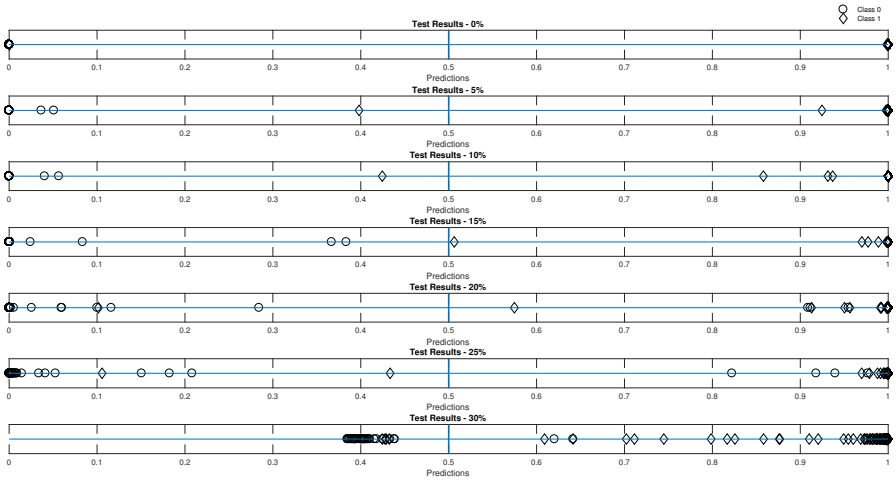

Figure 4.12: Line charts visualizing prediction probabilities from the pruned baseline models with 15 locked layers evaluated on the test data in DS3.

Figure 4.12, visualizing the robustness of the pruned models with 15 layers locked, show a degradation of classification performance when around 10 % or more convolutional filters are removed. The 0-class predictions in particular seem to draw closer to 0.5, indicating that these models are struggling to classify the CLO. This trend is less noticeable with the models with 10 layers locked, as seen in Figure 4.11.

When comparing the file sizes of the models with 10 respectively 15 locked layers it can be clearly seen that the models with 10 locked layers leads to significantly larger file sizes. Table 4.4 and Table 4.5 shows that a model with 10 layers locked, with 10 % pruning, yields a model of almost the same size as a model with 15 layers

locked and 0 % pruning.

Pruned model	**0 %**	**5 %**	**10 %**	**15 %**	**20 %**	**25 %**	**30 %**
File Size [Mb]	*158,44*	*144,84*	*130,76*	*118,26*	*107,14*	*96,57*	*87,38*
Reduction [%]	*0*	*-8,58*	*-17,47*	*-25,36*	*-32,38*	*-39,05*	*-44,85*
Accuracy	*0,97*	*0,99*	*1*	*0,98*	*0,99*	*0,99*	*0,98*
Precision 0	*0,96*	*1*	*1*	*1*	*1*	*1*	*1*
Precision 1	*0,98*	*0,99*	*1*	*0,98*	*0,99*	*0,99*	*0,98*
Recall 0	*0,94*	*0,96*	*1*	*0,94*	*0,98*	*0,96*	*0,94*
Recall 1	*0,99*	*1*	*1*	*1*	*1*	*1*	*1*
# Wrong	*5*	*2*	*0*	*3*	*1*	*2*	*3*

Table 4.4: A table showing total file sizes, file size reductions and test metrics for each pruned model with ten locked layers. Last row indicates amount of incorrectly classified images in the test images from DS3.

Pruned model	**0 %**	**5 %**	**10 %**	**15 %**	**20 %**	**25 %**	**30 %**
File Size [Mb]	*135,34*	*124,13*	*112,45*	*102,05*	*92,77*	*83,94*	*76,22*
Reduction [%]	*0*	*-8,28*	*-16,91*	*-24,60*	*-31,45*	*-37,98*	*-43,68*
Accuracy	*0,99*	*0,99*	*0,99*	*1*	*0,98*	*0,97*	*0,97*
Precision 0	*0,96*	*0,98*	*0,98*	*1*	*0,98*	*0,96*	*0,92*
Precision 1	*1*	*1*	*1*	*1*	*0,98*	*0,97*	*0,99*
Recall 0	*1*	*1*	*1*	*1*	*0,94*	*0,92*	*0,96*
Recall 1	*0,99*	*0,99*	*0,99*	*1*	*0,99*	*0,99*	*0,97*
# Wrong	*2*	*1*	*1*	*0*	*4*	*6*	*6*

Table 4.5: A table showing total file sizes, file size reductions and test metrics for each pruned model with 15 locked layers. Last row indicates amount of incorrectly classified images in the test images from DS3.

Table 4.4 and Table 4.5 shows how the regularizing effect of pruning has affected the test results, since models with larger file sizes perform worse than models with smaller file sizes. Remember, since the training have been conducted with a relatively small dataset, the models with large amounts of parameters are likely to be keen to overfit.

Overall, the models with 10 locked filters perform better than the models with 15 locked layers on the test images in DS3. However, it must be taken into consideration that these models are more costly when considering their memory footprint. With all these facts in mind, there are five pruned models standing out, which will be further evaluated. The models that chosen were: `10-Locked 10%`, `10-Locked 20%`, `15-Locked 5%`, `15-Locked 10%` and `15-Locked 15%`.

The choice to further evaluate these five models was based on the models classification performance and file size. All models chosen for further evaluation had a maximum of one incorrectly classified image from the DS3 test data.

4.2.3 Phase 3 - Comparison

All models chosen to be further evaluated after the second pruning phase had proven to perform well on the test data in DS3, as seen in Section 4.2.2. The high performance of all these models was also noted in the studio environment. In general, each model can classify most scenes presented with high reliability. Some scenes for which all models returned correct classifications with mostly high prediction certainty can be seen in Figure 4.13. The prediction probabilities of these images can be seen in Table 4.6.

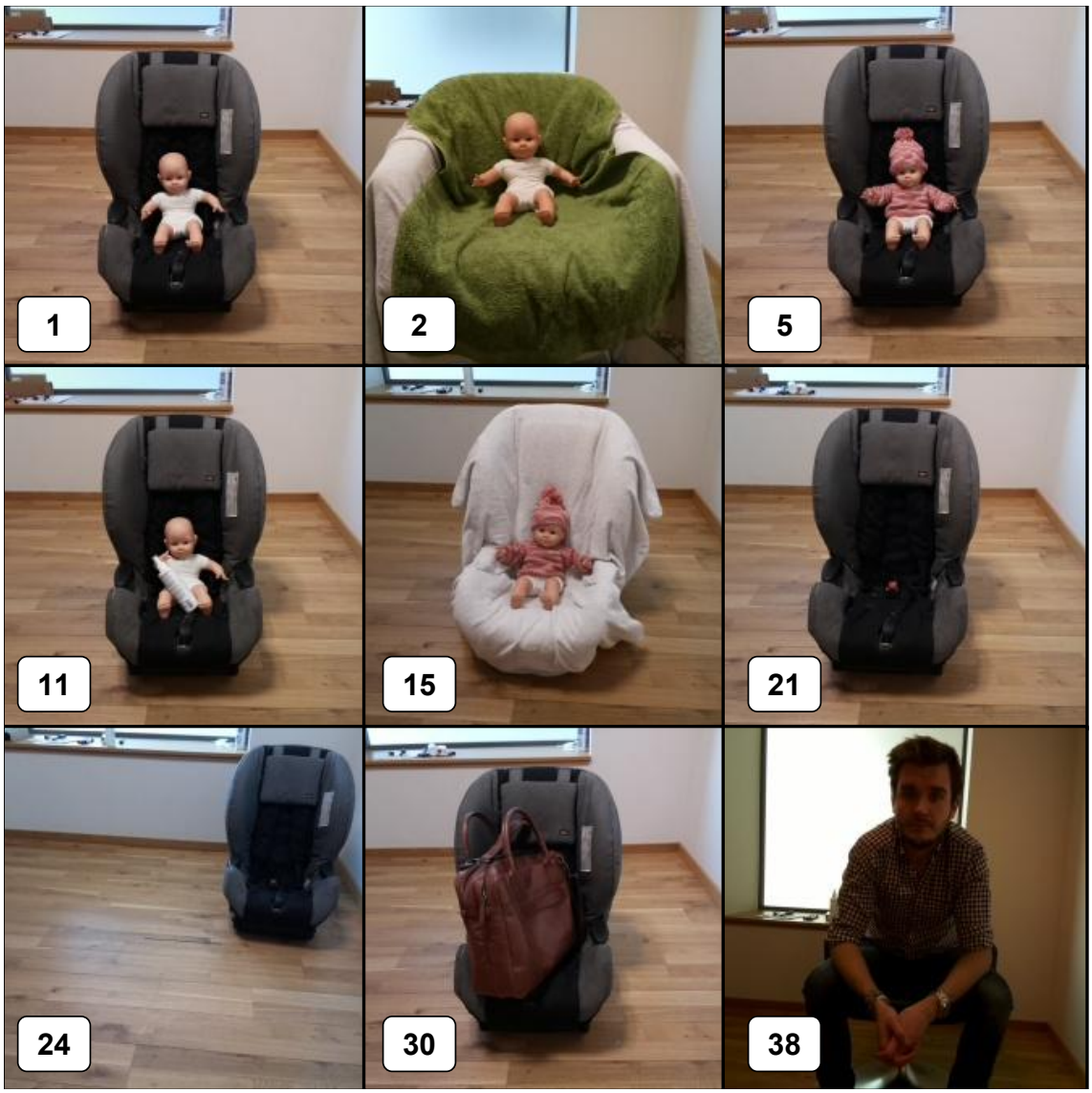

Figure 4.13: Images showing some of the scenes that all models classified correctly. The prediction probabilities are presented in Table 4.6.

Scene:	Baseline	10L-10%	10L-20%	15L-5%	15L-10%	15L-15%
1	*2.38e-14*	*1.87e-2*	*6.30e-3*	*8.81e-7*	*3.50e-3*	*0.19*
2	*1.69e-8*	*1.28e-2*	*3.60e-3*	*6.07e-5*	*1.72e-2*	*3.19e-2*
5	*5.75e-15*	*5.06e-4*	*9.10e-3*	*4.52e-7*	*3.60e-3*	*0.16*
11	*1.09e-13*	*7.00e-3*	*1.37e-2*	*2.10e-6*	*4.60e-3*	*0.19*
15	*1.08e-13*	*9.50e-5*	*0.49*	*4.78e-10*	*1.20e-3*	*2.70e-3*
21	*0.91*	*1.00*	*1.00*	*0.98*	*1.00*	*1.00*
24	*1.00*	*1.00*	*1.00*	*1.00*	*1.00*	*1.00*
30	*1.00*	*1.00*	*1.00*	*0.98*	*0.97*	*1.00*
38	*1.00*	*1.00*	*1.00*	*1.00*	*0.91*	*1.00*

Table 4.6: A table showing the prediction probabilities for the scenes presented in Figure 4.13.

The results presented above shows that every model perform well when classifying images gathered in a completely new environment. One noticeable difference in performance between the models can be seen in scenes 1, 11 and 15, where a decrease in robustness is present when moving from the baseline to the pruned models. All models are capable of classifying the CLO in most cases; where the doll is not concealed too much, rotated too much, moved or in other ways manipulated to not look like the training data. In addition, they show great adaptivity to changes of chairs and clothes. It should be noted that the previously mentioned scenes are only a fraction of the cases where the developed models perform well. For a full summary of all results, see Section A.1.

Furthermore, when the models are exposed to more difficult scenes, differences between model performance are highlighted. Figure 4.14 shows three images where some models fails to return correct classifications, as seen in Table 4.7.

Figure 4.14: Images showing scenes seven to nine. The prediction probabilities are presented in Table 4.7.

Scene:	Baseline	10L-10%	10L-20%	15L-5%	15L-10%	15L-15%
7	*4.05e-9*	*8.02e-2*	*0.64*	*1.00e-3*	*1.50e-3*	*0.92*
8	*8.87e-13*	*5.50e-3*	*2.74e-2*	*2.41e-7*	*3.70e-3*	*0.33*
9	*1.06e-8*	*0.36*	*0.32*	*1.95e-2*	*3.30e-3*	*0.65*

Table 4.7: A table showing the prediction probabilities for the scenes presented in Figure 4.14.

Scenes 7 - 9, presented in Figure 4.14 depict the doll being concealed in different ways. As seen in Table 4.7, the 10L-20% and the 15L-15% models fails to classify at least one of these images. Furthermore, Figure 4.15 shows three scenes where all models fails to return reliable classifications, as seen in Table 4.8.

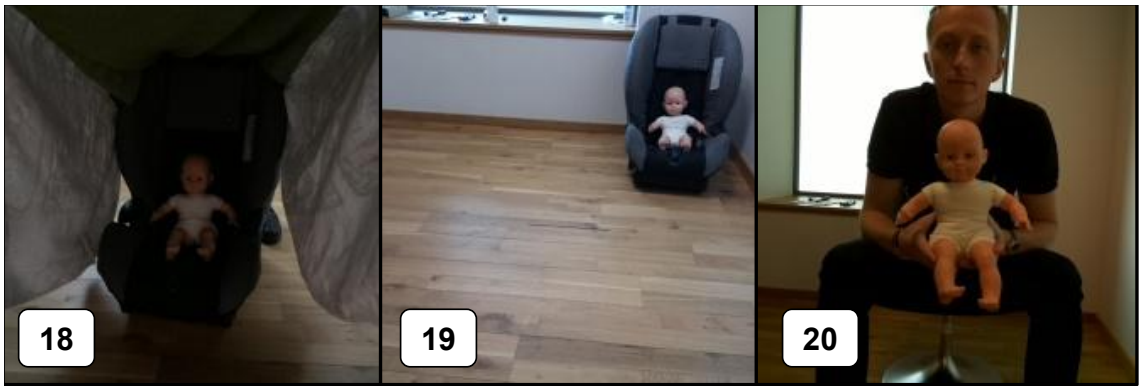

Figure 4.15: Images showing scenes 18-20. The prediction probabilities are presented in Table 4.8.

Scene:	Baseline	10L-10%	10L-20%	15L-5%	15L-10%	15L-15%
18	*0.64*	*1.00*	*0.66*	*9.10e-3*	*0.99*	*0.99*
19	*1.00*	*1.00*	*1.00*	*1.00*	*1.00*	*1.00*
20	*1.00*	*1.00*	*0.99*	*0.99*	*0.14*	*0.99*

Table 4.8: A table showing the prediction probabilities for the scenes presented in Figure 4.15.

The results presented in Figure 4.15 and Table 4.8 were expected, since these scenes intended to show the weakness of the models caused by the limited training data. For example, the models have not been trained on images were the light-conditions are bad or images where the CLO is situated far away. Also, Table 4.8 show how most models fail to classify scene 20. It was later discovered that the human subjects were only visible on the negative training data, meaning that the models predicted as they had been trained, although incorrectly in this case. The models seem to be biased towards a class 1 prediction, i.e. an empty vehicle for scenes with the adult human subjects.

Figure 4.16 and Table 4.9 shows three other scenes which are also worth investigating

further. These results indicate that the models with the most trainable parameters, i.e. the baseline and 15L-5% model, have been overfitted. These models failed to classify scenes 22, 27 and 28 (15L-5% barely making a correct classification), pointing towards the conclusion that these models are biased towards the old doll clothes present in scene 22 and 28. This set of clothes have only been present in the positive training samples.

Figure 4.16: Images showing scenes 22, 27 and 28. The prediction probabilities are presented in Table 4.9.

Scene:	Baseline	10L-10%	10L-20%	15L-5%	15L-10%	15L-15%
22	*3.02e-5*	*1.00*	*0.73*	*2.78e-2*	*0.85*	*0.99*
27	*1.12e-4*	*1.00*	*1.00*	*3.16e-2*	*1.00*	*0.98*
28	*2.94e-4*	*1.00*	*1.00*	*0.63*	*0.99*	*1.00*

Table 4.9: A table showing prediction probabilities for the scenes presented in Figure 4.16.

When introducing human subjects in scenes 34 and 35 there is a high misclassification-rate, see Figure 4.17 and Table 4.10. This is obviously not desirable, and one reasonable explanation for this could be the simple fact that real humans share many similar features with CLOs, in comparison to the other test items presented. The respective prediction probabilities reflect this issue, with many prediction probabilities close to 0.5, which indicate that the models are uncertain when adult humans are present in the image.

Figure 4.17: Images showing scenes 34 and 35. Predictions probabilities are presented in Table 4.10.

Scenes:	Baseline	10L-10%	10L-20%	15L-5%	15L-10%	15L-15%
34	*4.03e-6*	*0.47*	*0.58*	*9.14e-2*	*0.20*	*0.43*
35	*1.50e-3*	*0.77*	*0.95*	*0.72*	*0.46*	*0.94*

Table 4.10: A table showing the prediction probabilities for the scenes presented in Figure 4.17.

Score

The score system was intended to simplify the process of determining which model returned the most robust classifications. The score-results can be seen in Table 4.11.

Model:	Baseline	10L-10%	10L-20%	15L-5%	15L-10%	15L-15%
Score	*10.18*	*6.24*	*9.33*	*7.78*	*6.90*	*10.07*

Table 4.11: A table showing the scores for all models in phase 3. See Section 3.5.4 for further explanation of score calculations (less is better).

The score once again show what have been seen throughout phase 3, i.e. that all models perform relatively similar. However, according to the score 15% pruning is achievable while still preserving the classification performance, and in these cases, actually surpassing the baseline model in performance. Furthermore, the 10L-10% model achieves the best score, by a small margin.

Execution Times

Table 4.12 shows that increased pruning lead to a decreased time consumption while classifying images and loading models.

Model:	Baseline	10L-10%	10L-20%	15L-5%	15L-10%	15L-15%
Avg $\mathbf{t}_{load}[s]$	*19.27*	*20.47*	*19.14*	*18.76*	*18.27*	*17.67*
Avg $\mathbf{t}_{class.}[s]$	*4.14*	*3.64*	*3.08*	*4.16*	*3.68*	*3.45*

Table 4.12: A table showing the average times calculated over all 40 test images during phase 3. Avg t_{load} - Average model load time, Avg $t_{class.}$ - Average classification time.

Loading models seem to be a demanding task for the Raspberry Pi and is not directly correlated to the model file size. For instance, the 10L-10% model has a smaller file size than the baseline, but still require more loading time. For file sizes, see Tables 4.4 & 4.5.

To conclude, there are many scenarios where the models perform well and some demanding ones where different models fail. Occlusion, light-conditions, distances and human-CLO-combinations are some of the scenarios where the models struggle. All models show good robustness against new clothes, objects and chairs but become less certain when only humans are present in the images. The best performing model, 10L-10%, was chosen based on these results. In addition, the 10L-10% model have reduced classification time by 12% compared to the baseline model.

4.3 System Test - Proof of Concept

The 10L-10% model achieved promising performance in the studio and was used in the final proof of concept, which aims to introduce more demanding challenges, presented in Section 3.6. Twenty different scenes were tested and the average classification time over all images were 3.48s, compared to 4.14s for the baseline model.

Initially, some easy scenes were presented, to ensure that the system was functional. These easy scenes are shown in Figure 4.18, with prediction probabilities in Table 4.13.

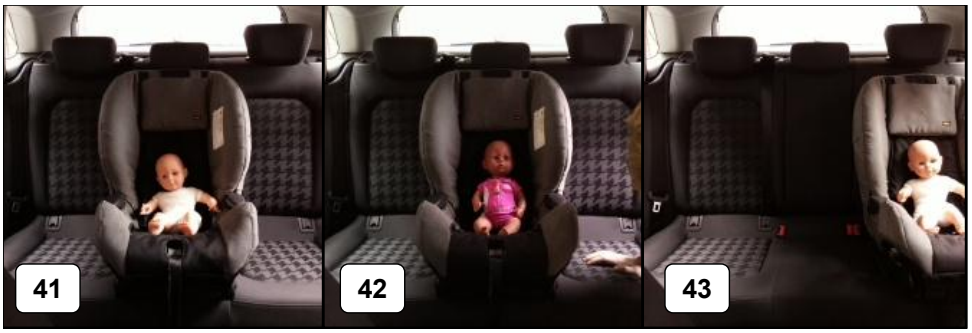

Figure 4.18: Images showing scenes 41, 42 and 43 with their respective prediction probability in Table 4.13.

Scene	41	42	43
Prediction	*2.71e-12*	*2.49e-10*	*3.43e-4*

Table 4.13: A table showing the predictions probabilities for the scenes presented in Figure 4.18.

These results show that the developed system is capable of detecting the CLOs in the easiest conditions, even though the vehicle and one of the CLOs are new. Scene 43 show that moving the CLO to another seat will also result in a robust prediction probability. Moving on to more challenging test scenes, Figure 4.19 and Table 4.14 show the results achieved when placing various objects in the CS.

Figure 4.19: Images showing scenes 45, 46 and 52 with their respective prediction probability in Table 4.14.

Scene	45	46	52
Prediction	*1.20e-7*	*1.41e-6*	*1.00*

Table 4.14: A table showing prediction probabilities for scenes presented in Figure 4.19.

An equally promising performance is seen in Figure 4.19 as well. Robust prediction probabilities in each case, even when two CLOs are placed together. These scenes also show that the bag does not trick the system. Furthermore, occlusion was one of the biggest problems in Phase 3, and various levels of occlusion was also tested inside the vehicle, see Figure 4.20 and Table 4.15.

Figure 4.20: Images showing scenes 44, 53 and 54 with their respective prediction probabilities in Table 4.15.

Scene	44	53	54
Prediction	*1.26e-4*	*4.59e-5*	*0.95*

Table 4.15: A table showing prediction probabilities for the scenes presented in Figure 4.20.

In the above figure it is shown how the system is capable of making correct predictions even when the CLO is partly covered, however this is not the case in every scene. When the head is covered, the model struggle to find the CLO, which point towards the conclusion that the model in these cases focus more on the head than the body of the CLO. These findings differs from the results found in Section 4.2.3, where the 10L-10% model managed to classify all levels of occlusion.

To ensure that the model did not focus on finding the CS instead of the CLO, scenes were included where this could be ruled out, see Figure 4.21 and Table 4.16.

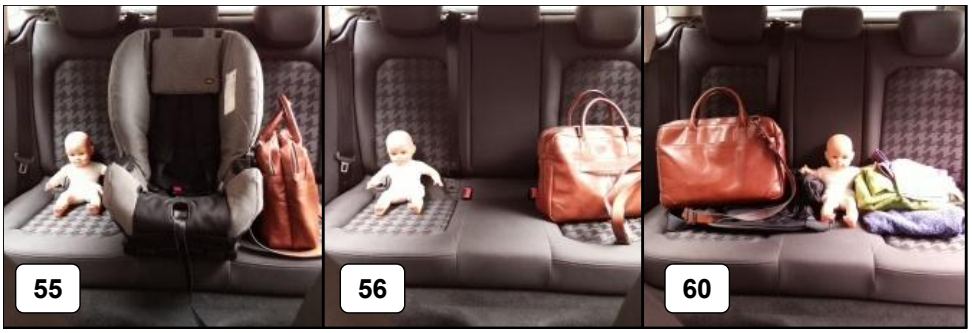

Figure 4.21: Images showing scenes 55, 56 and 60 with their respective prediction probabilities in Table 4.16.

Scene	55	56	60
Prediction	*0.21*	*1.02e-2*	*2.89e-3*

Table 4.16: A table showing prediction probabilities for the scenes presented in Figure 4.21.

Compared to earlier results, Table 4.16 show correct predictions in all cases, but with less certainty. Scene 55 and 56 are similar, except for the CS, and the predictions show that the robustness of the prediction increase as the CS is removed. Both scene 55 and 60 present some occlusion of the limbs of the CLO, with relatively certain predictions.

Next, various ways of tricking the system were tested, with mixed results as seen in Figure 4.22 and Table 4.17.

Figure 4.22: Images showing scenes 48, 49 and 50 with their respective prediction probabilities in Table 4.17.

Scene	48	49	50
Prediction	*1.00*	*0.85*	*1.70e-2*

Table 4.17: A table showing prediction probabilities for the scenes presented in Figure 4.22.

Here, a problem in the developed model becomes apparent. Moving from scene 48 to 50, at first the model is very robust in it's prediction, but when the CS is introduced there is a noticeable drop in classification certainty. Lastly, when both the clothes and CS are present the model is tricked into thinking the CLO is visible, with relatively high certainty. Other challenging scenarios include the ones where humans are visible, as seen in Figure 4.23 and Table 4.18.

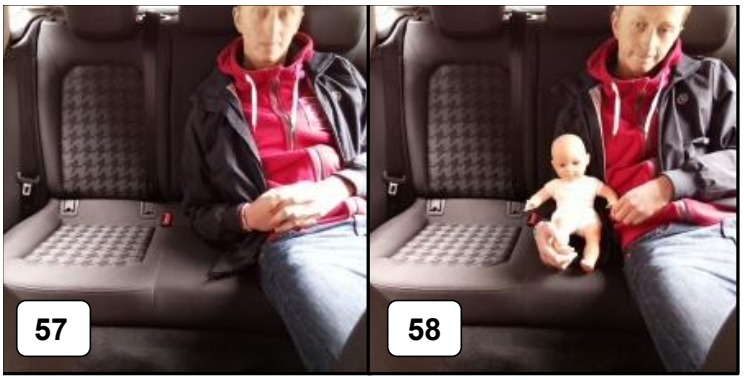

Figure 4.23: Images showing scenes 57 and 58 with their respective prediction probabilities in Table 4.18.

Scene	57	58
Prediction	*0.61*	*4.25e-6*

Table 4.18: A table showing prediction probabilities for the scenes presented in Figure 4.23.

In scene 57 it becomes clear that the model still has a hard time with human subjects. In contrast to this, the model makes very robust predictions when the CLO is introduced in the scene.

5
Concluding Discussion

With a computer vision approach for classification there are always extreme cases where the system will fail and provide unreliable predictions. The image space is often highly varying; there are even cases where the human eye have problems detecting changes between different scenarios. Knowing this, the overall result of this project is promising and in line, or even above, the initial expectations. There are several cases where the system always manages to make correct classifications and there are some cases where the system is plain wrong or somewhat stochastic in it's predictions.

These results have been achieved by a model based on the VGG16 architecture which has undergone successful pruning of 10%. The pruning made it possible to reduce the running time by 12% while running on a Raspberry Pi.

Starting with the positive results, the developed system seem to be very adaptive to items never seen before, for instance new chairs, new clothes, new dolls and even putting a hat on the CLO. What is striking is the performance shown when placing the CLO in different chairs, totally unfamiliar to the model. Since the model is capable of classifying these images without any notable problems it seems like the system have learned not to put too much focus on the CS itself, but rather focus on the CLO, which of course is desirable and essential in order to prevent false positives. In the sense of scene adaptation, the system shows high performance, and does not show any signs of classification problems, which is beyond what was expected. When covering the CLO, relatively high performance is achieved as well, with some miss-classifications depending on the amount and placement of occlusion.

The offline tests showed that the model managed to classify all test-images and achieve zero miss-classifications, which means it outperformed the baseline model as seen in Table 4.4. This points towards a successful implementation of the pruning and that the models have experienced some regularizing effects. Similar effects are seen when looking at the 15L-15% model in Table 4.5, which raises questions regarding what would happen if more layers were unlocked, more on this further on.

Focusing on the scenes that confuses the model, there are several areas that can be deeper discussed. Some of the most interesting results were found when adult humans were placed in the scene, and it was clear that the model had difficulties classifying these instances. Scenes 34 and 35 in Figure 4.17 reflect this. Initially these findings were thought to be connected to the fact that the human face share simi-

larities with the CLO. However, this statement contradicts with the results found in scene 20, where the face is clearly visible, but the prediction is clearly "empty". Why the system became uncertain on scenes 34 and 35, but not on scene 20 is interesting, and hard to answer. One explanation could be that the heads of scene 34 and 35 are placed in the CS, which in combination with the similar face structure tricked the system. This behaviour can also be found in Figure 4.22, where the clothes by themselves seem easy to classify, but when they are placed in the CS the system is deluded. One solution to solve this might be to use several different CSs when collecting training data, reducing the possibility that the model connects the CS to the CLO in any way.

In terms of concealment, the results found in the studio and vehicle environments were not entirely coherent. In the studio, the 10L-10% model was quite certain of scene 7 and 8 and less certain when predicting scene 9, but still correct in its classifications. This goes in contradiction to the proof of concept studies where the model miss-classified when the head of the CLO was covered. This is probably due to the limbs being more distinguishable in the studio images, or that the way the towel covered the head in scene 8, made it look like a head for the model.

In addition to these issues, the system experienced problems when dealing with low light conditions, objects far away from the camera and changes in perspective. None of these findings are very surprising, since the setup used for collecting data was static and changes in light condition, distance or perspective were never made. Hence, over 90% of the training data depictured these same conditions, which ultimately means that the model is dependent on them being similar when using the system.

It is of high importance to continue working on solving these classification problems to increase the safety provided by the final system. By missing a CLO in an image it is obvious that the system has failed from a safety stand point. It is crucial to remember that minimizing the false positives is also of high importance, since a safety system constantly delivering false positives to the user will probably be switched off and not used.

When looking for future possibilities, there are several aspects of this project that could be considered. Firstly, if hardware of higher performance was available for network development it would be possible to unlock more layers. When more layers were unlocked in pruning phase 2, Section 4.2.2, a notable increase in overall performance was seen. Both in classification robustness and better test results at higher pruning levels. The reason for this is probably that with less locked layers the final models become more specialized to this specific training data, which with relatively little intra-class variations have several common features over the entire set, that could be picked up by the models. By being able to unlock more layers it is possible that pruning levels could be pushed further than what have been done here, which would mean that even cheaper hardware than the Raspberry Pi could be used to run the final system.

As stated throughout the project, a computer vision solution to a problem is seldom fail-proof and there are often ways in which such a system can be fooled. For a safety solution of this kind, where failing to classify a child could have catastrophic consequences, we think it would be necessary to rely on more than one image taken of the passenger cabin. Maybe several images could be taken, some seconds or minutes apart to account for cases where the child might have moved or become better visible to the camera. An even safer solution would be to utilize several different sensors, like infrared cameras to detect a child covered under a blanket or a microphone to pick up screams. This would trigger a wide range of other discussions concerning the cost of adding more sensors, integrity and machine learning problems of infrared images, which will not be covered here.

One of the easiest way to achieve better results in a future system is to increase the amount of training data, focusing on making it more diverse. Maybe focusing on different occlusion scenarios and child positions would be a good place to start. A more complex solution could be to develop a system that captures training data from the car owners over time, as they are utilizing the vehicle in their daily life. As an example, an app could be developed where the owner provides inputs, telling the system who is in the vehicle as he or she locks the doors and goes shopping. While shopping, the vehicle would then be able to collect training data from the passenger cabin where various scenarios could unfold. With this approach large amounts of good training data, for that specific family, could be easily collected and in time be used to train an even more accurate detection system.

To conclude the discussion, the research questions presented in Section 1.2 will be answered, starting with:
"Is it possible to detect CLOs using machine learning running on cheap hardware?" - Well, yes. In this project a 350 SEK Raspberry Pi was used, and we do not see any obstacle as for why an even cheaper system would not work. As long as sufficient memory and computing capability is provided.

What classification robustness is achievable? - The robustness of the performed classification is very high in the easiest test cases. In some cases the system becomes unsure in it's predictions, entirely depending on scenario.

Is it possible to extract enough training data during the time frame for this project? - It is possible to extract enough training data for a proof of concept study, a commercialized system would require much more data.

What limitations does the implemented detector exhibit? - The implemented solution have problems with varying light conditions, perspective and distances from the camera, as well as occlusion of the CLO and some cases involving human subjects.

Does the results of this thesis imply that a similar software system for detecting real children would be possible? - It has not been possible to try the developed solution on a real child, but considering the adaptation shown by the model while classifying new dolls, with new clothes for instance, a qualified guess is that a child could be detected as well.

Bibliography

[1] Heatstroke Deaths of Children in Vehicles. 2017. Monthly Statistics. [Online] Available at: http://www.noheatstroke.org/monthly_stats.htm.. [Accessed 12 February 2018].

[2] Safe Kids Worldwide. 2014. Heatstroke Prevention: It's time to take action and put an end to hot car deaths. [Online]. Available at: https://tinyurl.com/ya57yvuf. [Accessed 12 February 2018].

[3] TIME. 2014. "Who's at Fault When a Child Dies in a Hot Car?". [Online]. Available at: http://time.com/hot-car-death/. [Accessed 12 February 2018].

[4] National Highway Traffic Safety Administration. 2015. "Traffic Safety Facts". [Online]. Available at: https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812120. [Accessed 16 January 2018].

[5] Congress.gov. 2017. "H.R.2801 - HOT CARS Act of 2017". [Online]. Available at: https://www.congress.gov/bill/115th-congress/house-bill/2801/text. [Accessed 17 January 2018].

[6] Zonfrillo et al. 2017. "Unintentional non-traffic injury and fatal events: Threats to children in and around vehicles". In *Traffic Injury Prevention*. 1. pp. 1-5. [Online]. Available at: http://www.tandfonline.com/doi/ref/10.1080/15389588.2017.1369053?scroll=top. [Accessed 16 January 2018].

[7] National Highway Traffic Safety Administration. 2017. "Child Safety". [Online]. Available at: https://www.nhtsa.gov/road-safety/child-safety#topic-heatstroke. [Accessed 16 January 2018].

[8] K. Thelma. 2012. "Hyperthermia and Children Left in Cars". In *Journal of emergency nursing*. 38. 3. pp. 287-288. [Online]. Available at: http://www.sciencedirect.com/science/article/pii/S0099176712000128?_rdoc=1&_fmt=high&_origin=gateway&_docanchor=&md5=b8429449ccfc9c30159a5f9aeaa92ffb&ccp=y. [Accessed 16 January 2018].

[9] Grundstein et al. 2015. "Evaluating infant core temperature response in a hot car using a heat balance model". In *Forensic Science, Medicine, and Pathology*. 11. pp. 13-19. [Online]. Available at: https://link.springer.com/content/pdf/10.1007%2Fs12024-014-9619-7.pdf. [Accessed 17 January 2018].

[10] Aappublications. 2005. "Heat Stress From Enclosed Vehicles: Moderate Ambient Temperatures Cause Significant Temperature Rise in Enclosed Vehicles". [Online]. Available at: http://pediatrics.aappublications.org/content/116/1/e109. [Accessed 17 January 2018].

[11] Horak et al. 2017. "Cabin air temperature of parked vehicles in summer conditions: life-threatening environment for children and pets calculated by

a dynamic model". In *Theoretical and Applied Climatology*. 130. pp. 107-118. [Online]. Available at: https://link.springer.com/article/10.1007%2Fs00704-016-1861-3. [Accessed 24 January 2018].

[12] NHTSA. 2017. "NHTSA helps raise awareness of child heatstroke in cars". [Online]. Available at: https://www.nhtsa.gov/press-releases/nhtsa-helps-raise-awareness-child-heatstroke-cars.. [Accessed 24 January 2018].

[13] J. Stallkampa, M. Schlipsinga, J. Salmena, C. Igelb. 2012. "Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition". [Online]. Available at: https://www-sciencedirect-com.proxy.lib.chalmers.se/science/article/pii/S0893608012000457?_rdoc=1&_fmt=high&_origin=gateway&_docanchor=&md5=b8429449ccfc9c30159a5f9aeaa92ffb&ccp=y. [Accessed 30 January 2018].

[14] NHTSA. 2015. "Functional Assessment of Unattended Child Reminder Systems". [Online]. Available at: https://one.nhtsa.gov/DOT/NHTSA/NVS/812187_UnattendedChildReminderSystems.pdf. [Accessed 13 February 2018].

[15] SAE MOBILUS. 2016. "Detector of Babies and Animals Forgotten in a Vehicle with Remote Notification System". [Online]. Available at: https://saemobilus.sae.org/content/2016-36-0188. [Accessed 17 January 2018].

[16] Abdullah et al. 2017. "Development of comprehensive unattended child warning and feedback system in vehicle". In *MATEC Web of Conferences*. 90. pp. 2-9. [Online]. Available at: https://www.matec-conferences.org/articles/matecconf/pdf/2017/04/matecconf_aigev2017_01008.pdf. [Accessed 17 January 2018].

[17] Guardian Optical Technologies. 2018. "One super smart sensor. An entirely new in-car experience". [Online]. Available at: https://www.guardian-optech.com/. [Accessed 13 February 2018].

[18] Fox News. 2018. "CES 2018: New sensor technology could prevent 'hot car' infant deaths". [Online]. Available at: https://tinyurl.com/ycn49dgm. [Accessed 13 February 2018].

[19] Forbes. 2018. "New Sensor Technology Can Detect Your Heartbeat In The Car". [Online]. Available at: https://www.forbes.com/sites/jenniferhicks/2018/01/05/new-sensor-technology-can-detect-your-heartbeat-in-the-car/#8f93cb446efa. [Accessed 13 February 2018].

[20] Business Wire. 2017. "Guardian Optical Technologies Raises $5.1 M to Accelerate Development of Breakthrough Sensor Technology for "Passenger-Aware" Cars". [Online]. Available at: https://www.businesswire.com/news/home/20171204005593/en/Guardian-Optical-Technologies-Raises-5.1-Accelerate-Development. [Accessed 13 February 2018].

[21] Marsico et al. 2017. "Human Recognition in Unconstrained Environments: Using Computer Vision, Pattern Recognition and Machine Learning Methods for Biometrics". 1st ed. London, United Kingdom: Academic Press.

[22] Newell et al. 2018. "Stacked Hourglass Networks for Human Pose Estimation". In *European Conference on Computer Vision*. pp. 483-499. [Online]. Avail-

able at: https://link-springer-com.proxy.lib.chalmers.se/chapter/10.1007%2F978-3-319-46484-8_29. [Accessed 30 January 2018].

[23] B S. Deepak, S. Shiv, R B. Venkatesh. 2017. "Switching Convolutional Neural Network for Crowd Counting" arXiv preprint: 1708.00199v2. [Online]. Available at: https://arxiv.org/pdf/1708.00199.pdf. [Accessed 13 February 2018].

[24] T. Santini, W. Fuhl, E. Kasneci. 2017. "Robust pupil detection for real-time pervasive eye tracking" arXiv preprint: 1712.08900v1. [Online]. Available at: https://arxiv.org/pdf/1712.08900.pdf. [Accessed 13 February 2018].

[25] Shin et al. 2016. "Deep Convolutional Neural Networks for Computer-Aided Detection: CNN Architectures, Dataset Characteristics and Transfer Learning". In *IEEE Transactions on Medical Imaging 35* 5. pp. 1285-1298. [Online]. Available at: http://ieeexplore.ieee.org/document/7404017/authors?ctx=authors. [Accessed 24 January 2018].

[26] K. Pasupa, W. Sunhem. 2016. "A comparison between shallow and deep architecture classifiers on small dataset". In *2016 8th International Conference on Information Technology and Electrical Engineering*. 1. pp. 1-6. [Online]. Available at: http://ieeexplore.ieee.org/document/7863293/. [Accessed 24 January 2018].

[27] Brandwatch. 2017. "Marketing: 105 Amazing Social Media Statistics and Facts". [Online]. Available at: https://www.brandwatch.com/blog/96-amazing-social-media-statistics-and-facts-for-2016/. [Accessed 1 March 2018].

[28] IBM. 2018. "Watson Health Imaging (video)". [Online]. Available at: https://www.ibm.com/watson/health/imaging/. [Accessed 1 March 2018].

[29] B. Cyganek. 2013. "Object Detection and Recognition" In *Digital Images: Theory and Practice. 1st ed. U.S.: John Wiley Sons.*

[30] O. Boiman O, E. Shechtman, M. Irani. 2008. "In defense of Nearest-Neighbor based image classification". In *IEEE Conference on Computer Vision and Pattern Recognition. Anchorage, AK, USA.*

[31] R J. Ramteke, K. Monali. 2012. "Automatic Medical Image Classification and Abnormality Detection Using K-Nearest Neighbour". In *International Journal of Advanced Computer Research.* 2. [Online]. Available at: https://pdfs.semanticscholar.org/9e23/2d7bb393f51a544561d82a76df418d47eedd.pdf. [Accessed 1 March 2018].

[32] M. Varma, D. Ray. 2007. "Learning The Discriminative Power-Invariance Trade-Off". In *IEEE 11th International Conference on Computer Vision. Rio de Janeiro, Brazil.*

[33] A. Bosch, A. Zisserman, X. Munoz. 2007. "Representing shape with a spatial pyramid kernel". In *CIVR '07 Proceedings of the 6th ACM international conference on Image and video retrieval. Amsterdam.* pp. 401-408.

[34] O. Chapelle, P. Haffner, V N. Vapnik. 1999. "Support vector machines for histogram-based image classification". In *IEEE Transactions on Neural Networks.* 10. pp. 1055 - 1064. [Online]. Available at: http://ieeexplore.ieee.org/abstract/document/788646/. [Accessed 1 March 2018].

[35] Chatfield et al. 2014. "Return of the Devil in the Details: Delving Deep into Convolutional Nets", In *Proceedings of the British Machine Vision Conference.*

1. [Online]. Available at: https://arxiv.org/abs/1405.3531. [Accessed 23 January 2018].

[36] A. Ardakan, C. Condo, W J. Gross. 2017. "Multi-Mode Inference Engine for Convolutional Neural Networks". arXiv preprint: 1712.03994. [Online]. Available at: https://arxiv.org/pdf/1712.03994.pdf. [Accessed 31 January 2018].

[37] Y. Gao, J. Ma, A L. Yuille. 2018. "Semi-Supervised Sparse Representation Based Classification for Face Recognition With Insufficient Labeled Samples". In *IEEE Transactions on Image Processing*. 26. pp. 2545 - 2560. [Online]. Available at: http://ieeexplore.ieee.org/abstract/document/7865903/ [Accessed 2 March 2018].

[38] A. Bhattacharyya, R. Saini, P P. Roy, D P. Dogra, S. Kar. 2017. "Recognizing Gender from Human Facial Regions using Genetic Algorithm" arXiv preprint: 1712.01661. [Online]. Available at: https://arxiv.org/abs/1712.01661. [Accessed 13 February 2018].

[39] S. Kumar, H. Kaur. 2012. "Face recognition techniques: Classification and comparisons". In *International Journal of Information Technology and Knowledge Management*. 5. pp. 361-363. [Online]. Available at: http://www.csjournals.com/IJITKM/PDF%205-2/26_Sanjeev_Kumar.pdf. [Accessed 2 March 2018].

[40] S. Abburu, S B. Golla. 2015. "Satellite Image Classification Methods and Techniques: A Review". In *International Journal of Computer Applications*. 119. [Online]. Available at: https://pdfs.semanticscholar.org/6961/3390ca76bf103791ef251e1568deb5fe91dd.pdf. [Accessed 2 March 2018].

[41] A. Romero, C. Gatta, G. Camps-Valls. 2016. "Unsupervised Deep Feature Extraction for Remote Sensing Image Classification". In *IEEE Transactions on geoscience and remote sensing*. 54. pp. 1349 - 1362. [Online]. Available at: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7293195 [Accessed 2 March 2018].

[42] F. Melgani, L. Bruzzone. 2004. "Classification of Hyperspectral Remote Sensing Images With Support Vector Machines". In *IEEE Transactions on geoscience and remote sensing*. 42. pp. 1778 - 1790. [Online]. Available at: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1323134. [Accessed 2 March 2018].

[43] M R. Zare, A. Mueen, W C. Seng. 2013. "Automatic Medical X-ray Image Classification using Annotation". In *Journal of Digital Imaging*. 27, Issue 1. pp. 77–89. [Online]. Available at: https://link.springer.com/article/10.1007%2Fs10278-013-9637-0. [Accessed 21 March 2018].

[44] R. Zhanga, J. Shen, F. Wei, X. Li, A K. Sangaiah. 2017. "Medical image classification based on multi-scale non-negative sparse coding". In *Artificial Intelligence in Medicine*. 83. pp. 44-51. [Online]. Available at: https://www.sciencedirect.com/science/article/pii/S0933365716306029?_rdoc=1&_fmt=high&_origin=gateway&_docanchor=&md5=b8429449ccfc9c30159a5f9aeaa92ffb. [Accessed 21 March 2018].

[45] Y. Song, W. Cai, H. Huang, Y. Zhou, D D. Feng, Y. Wang, M J. Fulham, M. Chen. 2015. "Large Margin Local Estimate With Applications to Medical Image Classification". In *IEEE Transactions on Medical Imaging*. 34, Issue: 6. pp.

1362 - 1377. [Online]. Available at: http://ieeexplore.ieee.org/document/7014242/. [Accessed 21 March 2018].

[46] E. Dias, M. Mirmehdi, T. Perrett. 2017. "Cost-based Feature Transfer for Vehicle Occupant Classification". arXiv preprint: 1512.07080. [Online]. Available at: https://arxiv.org/pdf/1512.07080.pdf. [Accessed 31 January 2018].

[47] J. Mao, T. Xiao. 2017. "What Can Help Pedestrian Detection?". arXiv preprint: 1705.02757. [Online]. Available at: https://arxiv.org/pdf/1705.02757.pdf. [Accessed 8 May 2018].

[48] Chalmers - Studentarbeten. 2017. "An evaluation of human pose estimation using a deep convolutional neural network". [Online]. Available at: https://tinyurl.com/yb5bvtxw. [Accessed 31 January 2018].

[49] Gomez et al. 2016. "Animal Identification in Low Quality Camera-Trap Images Using Very Deep Convolutional Neural Networks and Confidence Thresholds. International Symposium on Visual Computing". In *ISVC 2016: Advances in Visual Computing*. pp. 747-756. [Online]. Available at: https://link.springer.com/chapter/10.1007/978-3-319-50835-1_67. [Accessed 30 January 2018].

[50] G. Huang, J. Li, Z. Liu, Z. Shen, S. Yan, C. Zhang. 2017. "Learning Efficient Convolutional Networks through Network Slimming" arXiv preprint: 1708.06519. [Online]. Available at: https://arxiv.org/pdf/1708.06519.pdf. [Accessed 31 January 2018].

[51] K. Fukushima, "Neural network model for a mechanism of pattern recognition unnaffected by shift in position- neocognition," In *Electron. Commun*. Japan. 62. 10. pp. 1097-1105.

[52] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. 1989. "Backpropagation applied to handwritten zip code recognition." In *Neural Comput*. 1. 4. pp.541–551.

[53] A. Krizhevsky, I. Sutskever, G E. Hinton. 2012. "Imagenet classification with deep convolutional neural networks," In *Advances in Neural Information Processing Systems*. pp. 1097-1105.

[54] R. Girshick, J. Donahue, T. Darell, J. Malik. 2014. "Rich feature hierarchies for accurate object detection and semantic segmentation," In *IEEE Conference on Computer Vision and Pattern Recognition*. pp. 580-587.

[55] D. Anguelov, D. Erhan, Y. Jia, W. Liu, S. Reed, A. Rabinovich, V. Vanhoucke, P. Sermanet, C. Szegedy, 2014. "Going deeper with convolutions". arXiv preprint:1409.4842. [Online]. Available at: https://arxiv.org/pdf/1409.4842.pdf [Accessed 1 April 2018].

[56] Z. Alom, T. Josue, N. Rahman, W. Mitchell, C. Yakopcic, T. M. Taha. "Deep Versus Wide Convolutional Neural Networks for Object Recognition on Neuromorphic System". arXiv preprint: 1802.02608. [Online]. Available at: https://arxiv.org/pdf/1802.02608.pdf. [Accessed 2 April 2019].

[57] Y. LeCun, Y. Bengio, G. Hinton. 2015. "Deep learning" in *Nature*. 521. 7553. pp. 436–444.

[58] Y. LeCun. 1989. "Generalization and network design strategies". In *R. Pfeifer, Z.Schreter, F. Fogelman, L. Steels (Eds.) Connections in perspective* pp. 143–155.

[59] Y. LeCun, B. Boser,J. S. Denker, D. Henderson,R. E, Howard, W. Hubbard, L. D. Jackel. 1989. "Handwritten digit recognition with a back-propagation network". In *D. S. Touretzky (Ed.), Advances in neural information processing systems*. 2. pp. 396–404.

[60] Y. LeCun, B. Boser,J. S. Denker, D. Henderson,R. E, Howard, W. Hubbard, L. D. Jackel. 1989. "Backpropagation applied to handwritten zip code recognition". In *Neural Computation*. 1. 4. pp. 541–551.

[61] D C. Ciresan, U. Meier, J. Masci, L M. Gambardella, J. Schmidhuber. 2011. "Flexible, high performance convolutional neural networks for image classification". In *Proceedings of the International Joint Conference on Artificial Intelligence*. 1. pp. 1237–1242.

[62] A. Krizhevsky, I. Sutskever, G E. Hinton. 2012. "ImageNet classification with deep convolutional neural networks". In *F. Pereira, C. J. C. Burges, L. Bottou, K. Q. Weinberger (Eds.), Advances in neural information processing systems*. 25. pp. 1097–1105.

[63] K. Simonyan, A. Zisserman 2014. "Very deep convolutional networks for large-scale image recognition". arXiv preprint: 1409.1556. [Online]. Available at: `https://arxiv.org/pdf/1409.1556.pdf`. [Accessed 2 April 2018].

[64] M. D. Zeiler, R. Fergus. 2014. "Visualizing and understanding convolutional networks". In *Proceedings of the European Conference on Computer Vision*. pp. 818–833.

[65] B. Xu, N. Wang, T. Chen, M. Li. 2015. "Empirical evaluation of rectified activations in convolutional network". arXiv preprint: 1505.00853v2. [Online]. Available at: `https://arxiv.org/pdf/1505.00853v2.pdf`. [Accessed at 2 April 2018].

[66] P. Ramachandran, B. Zoph, Q. V. Le. 2017. "Searching for activation functions". arXiv preprint: 1710.05941. [Online]. Available at: `https://arxiv.org/pdf/1710.05941.pdf`. [Accessed 3 April 2018].

[67] A. Agarap. 2018. "Deep Learning using Rectified Linear Units (ReLU)". arXiv preprint: 1803.08375. [Online]. Available at: `https://arxiv.org/pdf/1803.08375.pdf`. [Accessed 9 May 2018].

[68] Y. LeCun, L. Bottou,Y. Bengio, P. Haffner. 1998. "Gradient-based learning applied to document recognition". In *Proceedings of the IEEE*. 86. 11. pp. 2278–2324.

[69] Y. Tang. 2013. "Deep learning using linear support vector machines". arXiv preprint: 1306.0239. [Online]. Available at: `https://arxiv.org/pdf/1306.0239.pdf`. [Accessed 2 April 2018].

[70] D. Barkana, A A. Mallouh, Z. Qawaqneh. 2017. "Deep Convolutional Neural Network for Age Estimation based on VGG-Face Model". arXiv preprint arXiv:1709.01664. [Online]. Available at: `https://arxiv.org/abs/1709.01664`. [Accessed 31 January 2018].

[71] S. Dieleman. 2014. "My solution for the Galaxy Zoo challenge". [Online]. Available at: `http://benanne.github.io/2014/04/05/galaxy-zoo.html`. [Accessed 23 January 2018].

[72] A K. Krizhevsky. 2018. "ImageNet classification with deep convolutional neural networks". In Communications of the ACM. 60. 6. pp. 84-90. [Online]. Available

at: https://dl.acm.org/citation.cfm?id=3065386. [Accessed 23 January 2018].

[73] T. Semwal, G. Mathur, P. Yenigalla, S. Nair. 2018. "A Practitioners' Guide to Transfer Learning for Text Classification using Convolutional Neural Networks". arXiv preprint: 1801.06480. [Online]. Available at: https://arxiv.org/pdf/1801.06480.pdf. [Accessed 16 April 2018].

[74] D. George, H. Shen, E A. Huerta. 2017. "Deep Transfer Learning: A new deep learning glitch classification method for advanced LIGO". arXiv preprint: 1706.07446. [Online]. Available at: https://arxiv.org/pdf/1706.07446.pdf. [Accessed 16 April 2018].

[75] M. Cogswell, F. Ahmed, R. Girshick, L. Zitnick, D. Batra. 2016. "Reducing overfitting in deep networks by decorrelating representations". arXiv preprint: 1511.06068. [Online]. Available at: https://arxiv.org/pdf/1511.06068.pdf. [Accessed 6 April 2018].

[76] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov. 2014. "Dropout: A simple way to prevent neural networks from overfitting". In *The Journal of Machine Learning Research*. 15. 1. pp. 1929–1958.

[77] T A. Nikolayevich. 1943. "On the stability of inverse problems". In *Dokl. Akad. Nauk SSSR*. pp. 195–198.

[78] I J. Goodfellow, D. Warde-Farley, M. M. Courville, A. Yoshua, B. Yoshua. 2013. "Maxout networks". In *Proceedings of the International Conference on Learning Representations*.

[79] S. Ruder. 2017. "An overview of gradient descent optimization algorithms". arXiv preprint: 1609.04747. [Online]. Available at: https://arxiv.org/pdf/1609.04747.pdf. [Accessed 9 April 2018].

[80] N. Qian. 1999. "On the momentum term in gradient descent learning algorithms". In *Neural Networks : The Official Journal of the International Neural Network Society 12(1)*. pp. 145–151. [Online]. Available at: http://doi.org/10.1016/S0893-6080(98)00116-6 [Accessed 9 April 2018].

[81] Y. Bengio. 2012. "Practical Recommendations for Gradient-Based Training of Deep Architectures". arXiv preprint: 1206.5533. [Online]. Available at: https://arxiv.org/pdf/1206.5533.pdf. [Accessed 7 April 2018].

[82] M. Sokolova, G. Lapalme. 2008. "A systematic analysis of performance measures for classification tasks". [Online]. Available at: http://rali.iro.umontreal.ca/rali/sites/default/files/publis/SokolovaLapalme-JIPM09.pdf. [Acessed 6 April 2018].

[83] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, L. Fei-Fei. 2015. "ImageNet Large Scale Visual Recognition Challenge". arXiv preprint: 1409.0575. [Online]. Available at: https://arxiv.org/pdf/1409.0575.pdf. [Accessed 2 April 2018].

[84] K. Simonyan, A Zisserman, 2015. "Very deep convolutional networks for large-scale image recognition". arXiv preprint: 1409.1556. [Online]. Available at: https://arxiv.org/abs/1409.1556v6. [Accessed 19 March 2018].

[85] A. Canziani, A. Paszke, E. Culurciello, 2016. "An Analysis of Deep Neural Network Models for Practical Applications". arXiv preprint: 1605.07678. [Online]. Available at: https://arxiv.org/abs/1605.07678. [Accessed 19 March 2018].

[86] Howard et.al, 2017. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications". arXiv preprint:1704.04861. [Online]. Available at: https://arxiv.org/abs/1704.04861. [Accessed 19 March 2018].

[87] G. Huang, Z. Liu, L. Van der Maaten, K Q. Weinberger. 2018. "Densely Connected Convolutional Networks". arXiv preprint: 1608.06993. [Online]. Available at: https://arxiv.org/abs/1608.06993. [Accessed 19 March 2018].

[88] K. He, X. Zhang, S. Ren, J. Sun. 2015. "Deep Residual Learning for Image Recognition". arXiv preprint: 1512.03385. [Online]. Available at: https://arxiv.org/pdf/1512.03385.pdf. [Accessed 3 April 2018].

[89] Google Trends. 2018. "Tensorflow,Pytorch,Caffe,Theano,Keras". [Online]. Available at: https://trends.google.se/trends/explore?date=2014-12-29%202018-01-29&q=tensorflow,Pytorch,%2Fg%2F11g6ym8nbt,Theano,Keras. [Accessed 29 January 2018].

[90] A S. Saxena. 2016. "Convolutional neural networks: an illustration in TensorFlow". In *XRDS: Crossroads, The ACM Magazine for Students - Cultures of Computing*. 22. 4. pp. 56-58. [Online]. Available at: https://dl.acm.org/citation.cfm?id=2951024. [Accessed 23 January 2018].

[91] Google research. 2016. "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems". arXiv preprint: 1603.04467. [Online]. Available at: https://arxiv.org/pdf/1603.04467.pdf. [Accessed 3 April 2018].

[92] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, T. Darrell. 2014. "Caffe: Convolutional Architecturefor Fast Feature Embedding". arXiv preprint: 1408.5093. [Online]. Available at: https://arxiv.org/pdf/1408.5093.pdf. [Accessed 3 April 2018].

[93] Y. Serdar. 2017. "Facebook brings GPU-powered machine learning to Python". In *InfoWorld*. [Online]. Available at: https://www.infoworld.com/article/3159120/artificial-intelligence/facebook-brings-gpu-powered-machine-learning-to-python.html. [Accessed 3 April 2018].

[94] The Theano Development Team. 2016. "Theano: A Python framework for fast computation of mathematical expressions". arXiv preprint: 1605.02688. [Online]. Available at: https://arxiv.org/pdf/1605.02688.pdf. [Accessed 3 April 2018].

[95] Chollet et al. 2015. "Keras". [Online]. Available at: https://github.com/keras-team/keras. [Accessed 3 April 2018].

[96] J H. Luo, J. Wu. 2017. "An Entropy-based Pruning Method for CNN Compression". arXiv preprint: 1706.05791 [Online]. Available at: https://arxiv.org/pdf/1706.05791.pdf. [Accessed 9 April 2018].

[97] Z. Wang, C. Zhu, Z. Xia, Q. Guo, Y. Liu. 2017. "Towards thinner convolutional neural networks trough gradually gobal pruning". arXiv preprint: 1703.09916v1. [Online]. Available at: https://arxiv.org/pdf/1703.09916v1.pdf. [Accessed 9 April 2018].

[98] S. Han, H. Mao, W. Dally. 2015. "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding". arXiv preprint: 1510.00149. [Online]. Available at: https://arxiv.org/pdf/1510.00149.pdf. [Accessed 10 April 2018].

[99] H. Li, A. Kadav, I. Durdanovic, H. Samet, H. P. Graf. 2016. "Pruning filters for efficient convnets". arXiv preprint: 1608.08710. [Online]. Available at: https://arxiv.org/pdf/1608.08710.pdf. [Accessed 10 April 2018].

[100] J. Luo, J. Wu. 2017. "An Entropy-based Pruning Method for CNN Compression". arXiv preprint: 1706.05791. [Online]. Available at: https://arxiv.org/pdf/1706.05791.pdf. [Accessed 10 April 2018].

[101] J. Wang. 2017. "Real-Time Embedded Systems". John Wiley Sons.

[102] J. Yu, B. Wilamowski. 2011. "Recent advances in in-vehicle embedded systems". In *Annual Conference on IEEE Industrial Electronics Society*. [Online]. Available at: http://ieeexplore.ieee.org.proxy.lib.chalmers.se/document/6120072/. [Accessed 30 January 2018].

[103] Raspberry Pi. 2016. "Raspberry Pi 3 Model B". [Online]. Available at: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/. [Accessed 29 January 2018].

[104] Raspberry Pi. 2016. "Camera Module V2". [Online]. Available at: https://www.raspberrypi.org/products/camera-module-v2/. [Accessed 24 January 2018].

[105] Arducam. 2015. "New Low Cost OV5647 Mini Camera Module for Raspberry Pi Now Available". [Online]. Available at: http://www.arducam.com/lowcost-raspberry-pi-mini-camera-module/. [Accessed 24 January 2018].

[106] Intorobotics. 2015. "The Raspberry Pi camera guide". [Online]. Available at: https://www.intorobotics.com/raspberry-pi-camera-guide/.. [Accessed 24 January 2018].

[107] G. Amato, F. Carrara, F. Falchi, C. Gennaro, C. Vario. 2016. "Car parking occupancy detection using smart camera networks and Deep Learning". In *IEEE Symposium on Computers and Communication. Messina, Italy*. [Online]. Available at: http://ieeexplore.ieee.org/abstract/document/7543901/. [Accessed 24 January 2018].

[108] M G. Bechtel, E. McEllhiney, H. Yun. 2017. "DeepPicar: A Low-cost Deep Neural Network-based Autonomous Car". arXiv preprint: 1712.08644v1. [Online]. Available at: https://arxiv.org/pdf/1712.08644.pdf. [Accessed 31 January 2018].

[109] D. Pena, A. Forembski, X. Xu, D. Moloney. 2017. "Benchmarking of CNNs for Low-Cost, Low-Power Robotics Applications". [Online]. Available at: http://juxi.net/workshop/deep-learning-rss-2017/papers/Pena.pdf. [Accessed 31 January 2018].

[110] K A. Petrova. 2017. "Computer vision system for robotic complex based on Arduino and Raspberry Pi". In *IEEE II International Conference on Control in Technical Systems*. pp. 25-27. [Online]. Available at: http://ieeexplore.ieee.org/document/8109570/. [Accessed 24 January 2018].

[111] http://se.farnell.com/raspberrypi-boards#mk-pi-3-model-b. [Accessed 10 April 2018]

[112] http://se.farnell.com/raspberry-pi/rpi-8mp-camera-board/raspberry-pi-camera-board-v2/dp/2510728. [Accessed 10 April 2018]
[113] https://www.kjell.com/se/sortiment/dator-natverk/enkortsdator/raspberry-pi/chassi-for-raspberry-pi-2-3-model-b-och-b-svart-p87281. [Accessed 10 April 2018]
[114] http://www.image-net.org/. [Accessed 11 May 2018].
[115] https://www.cs.toronto.edu/~kriz/cifar.html. [Accessed 11 May 2018].
[116] https://www.kaggle.com/. [Accessed 11 May 2018].
[117] Keras. 2016. "ImageDataGenerator". [Online]. Available at: https://keras.io/preprocessing/image/. [Accessed 22 March 2018].
[118] S. Hinterstoisser, V. Lepetit, P. Wohlhart. 2017. "On Pre-Trained Image Features and Synthetic Images for Deep Learning". arXiv preprint: 1710.10710. [Online]. Available at: https://arxiv.org/pdf/1710.10710.pdf. [Accessed 11 May 2018].

A
Appendix

A.1 Appendix 1

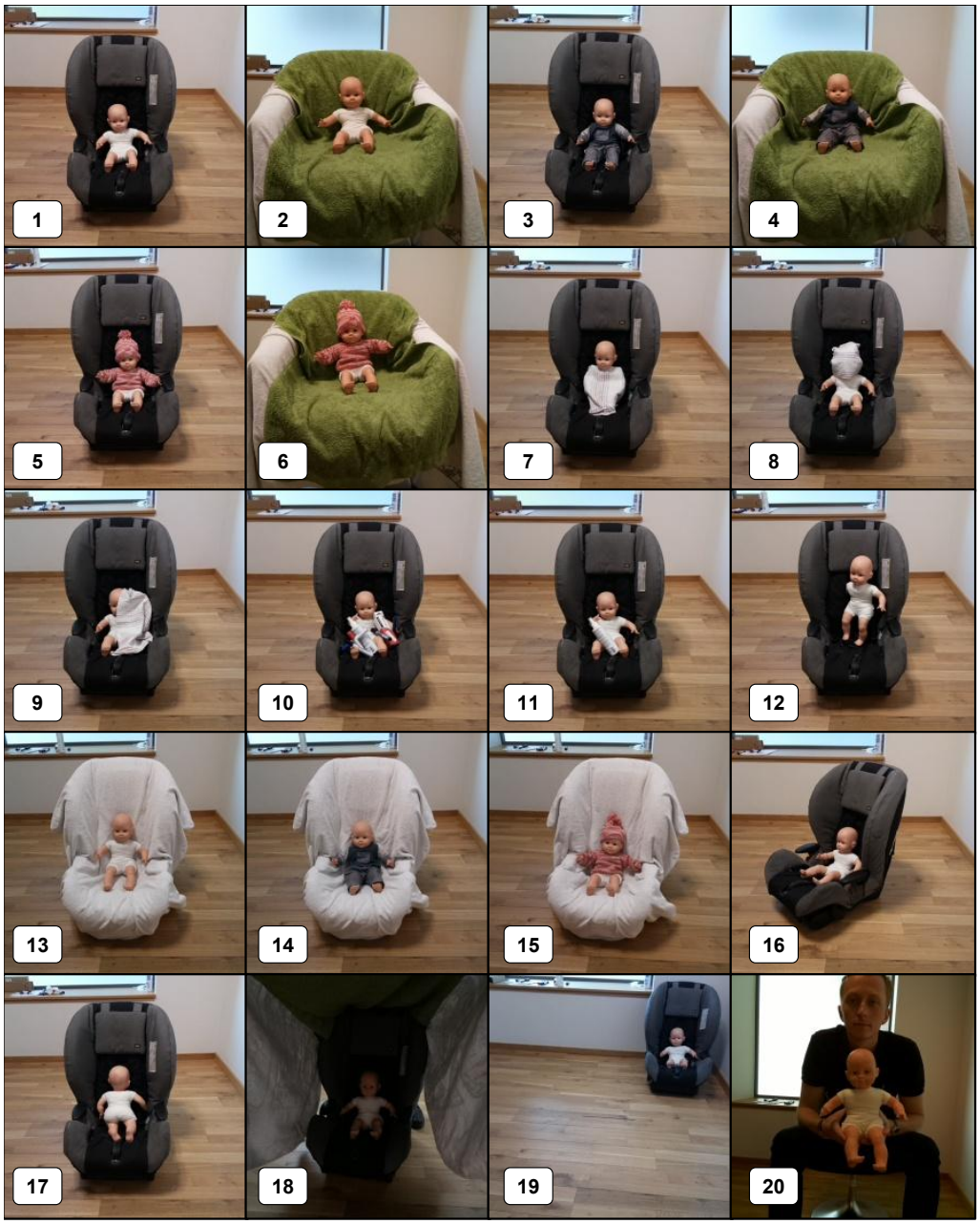

Figure A.1: Pruning - Phase 3 - All class 0-scenes used during phase 3 of pruning. The same scenes are presented in Table 3.6.

Scene:	Baseline	10L-10%	10L-20%	15L-5%	15L-10%	15L-15%
1	*2.38e-14*	*1.87e-2*	*6.30e-3*	*8.81e-7*	*3.50e-3*	*0.19*
2	*1.69e-8*	*1.28e-2*	*3.60e-3*	*6.07e-5*	*1.72e-2*	*3.19e-2*
3	*2.99e-13*	*9.71e-5*	*0.13*	*1.58e-7*	*2.10e-3*	*0.86*
4	*5.78e-8*	*7.42e-5*	*3.00e-3*	*1.05e-6*	*8.20e-3*	*1.43e-2*
5	*5.75e-15*	*5.06e-4*	*9.10e-3*	*4.52e-7*	*3.60e-3*	*0.16*
6	*1.26e-5*	*8.08e-4*	*1.40e-3*	*5.50e-3*	*2.63e-2*	*4.10e-3*
7	*4.05e-9*	*8.02e-2*	*0.64*	*1.00e-3*	*1.50e-3*	*0.92*
8	*8.87e-13*	*5.50e-3*	*2.74e-2*	*2.41e-7*	*3.70e-3*	*0.33*
9	*1.06e-8*	*0.36*	*0.32*	*1.95e-2*	*3.30e-3*	*0.65*
10	*1.99e-16*	*5.44e-4*	*3.69e-2*	*2.66e-8*	*3.60e-3*	*0.14*
11	*1.09e-13*	*7.00e-3*	*1.37e-2*	*2.10e-6*	*4.60e-3*	*0.19*
12	*7.50e-9*	*4.91e-4*	*0.15*	*6.96e-5*	*6.20e-3*	*0.53*
13	*2.26e-11*	*2.80e-3*	*0.33*	*8.42e-8*	*2.00e-3*	*8.80e-3*
14	*5.53e-11*	*4.69e-4*	*0.35*	*2.58e-7*	*3.01e-4*	*4.70e-3*
15	*1.08e-13*	*9.50e-5*	*0.49*	*4.78e-10*	*1.20e-3*	*2.70e-3*
16	*6.39e-14*	*3.36e-4*	*3.70e-3*	*5.39e-8*	*8.67e-4*	*1.38e-2*
17	*1.71e-8*	*0.12*	*0.61*	*3.90e-4*	*2.58e-2*	*0.70*
18	*0.64*	*1.00*	*0.66*	*9.10e-3*	*0.99*	*0.99*
19	*1.00*	*1.00*	*1.00*	*1.00*	*1.00*	*1.00*
20	*1.00*	*1.00*	*0.99*	*0.99*	*0.14*	*0.99*

Table A.1: Pruning - Phase 3 - The prediction of each model for all scenes presented in Figure A.1.

A.2 Appendix 2

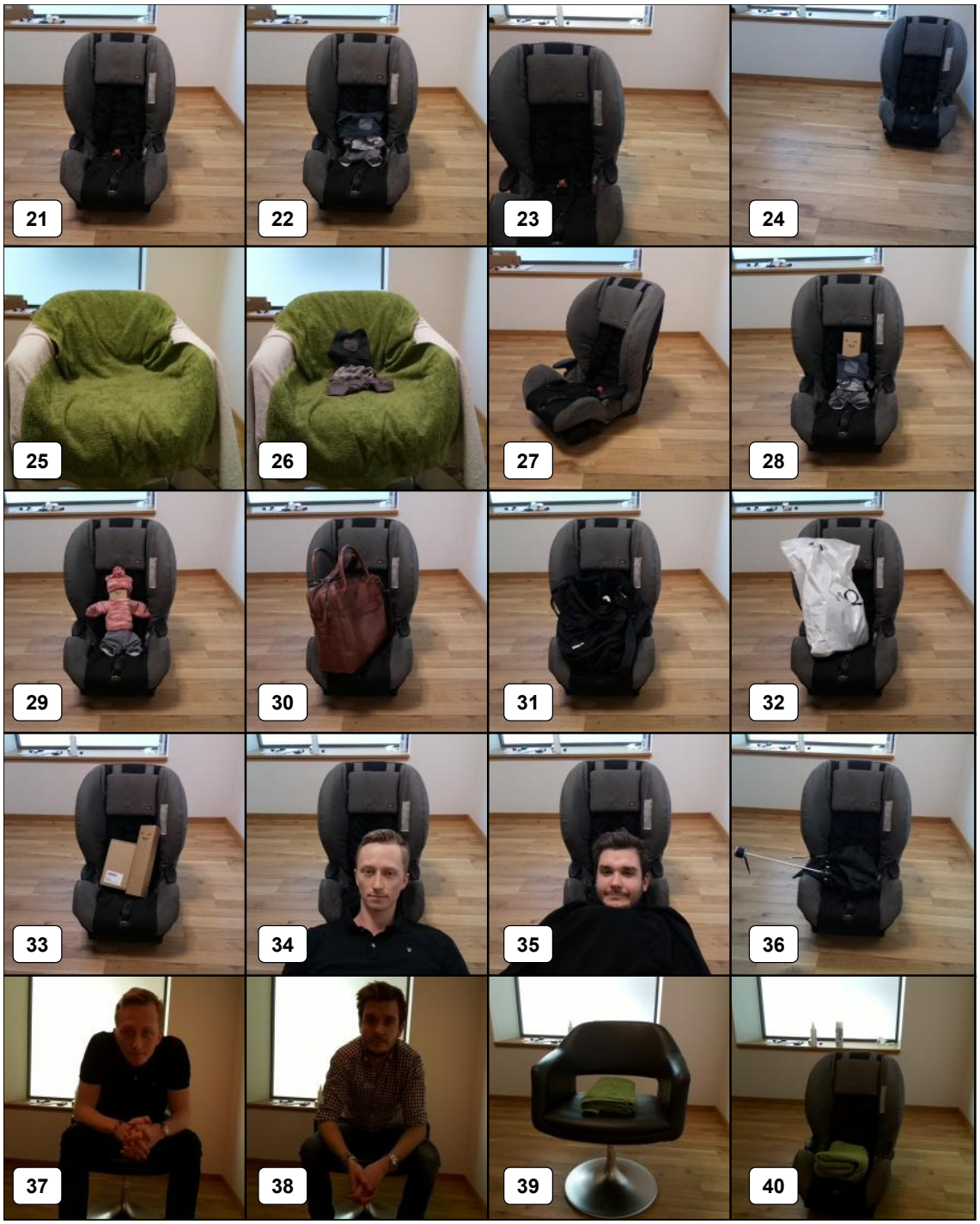

Figure A.2: Pruning - Phase 3 - All class 1-scenes used during phase 3 of pruning. The same scenes are presented in Table 3.6.

Scene:	Baseline	10L-10%	10L-20%	15L-5%	15L-10%	15L-15%
21	*0.91*	*1.00*	*1.00*	*0.98*	*1.00*	*1.00*
22	*3.02e-5*	*1.00*	*0.73*	*2.78e-2*	*0.85*	*0.99*
23	*1.00*	*1.00*	*1.00*	*0.86*	*1.00*	*1.00*
24	*1.00*	*1.00*	*1.00*	*1.00*	*1.00*	*1.00*
25	*1.00*	*1.00*	*0.23*	*0.99*	*0.99*	*0.86*
26	*1.4e-3*	*0.14*	*2.00e-3*	*6.00e-3*	*2.25e-2*	*2.04e-2*
27	*1.12e-4*	*1.00*	*1.00*	*3.16e-2*	*1.00*	*0.98*
28	*2.94e-4*	*1.00*	*1.00*	*0.63*	*0.99*	*1.00*
29	*5.41e-11*	*2.46e-4*	*4.74e-2*	*7.95e-4*	*1.30e-3*	*0.46*
30	*1.00*	*1.00*	*1.00*	*0.98*	*0.97*	*1.00*
31	*1.00*	*1.00*	*1.00*	*1.00*	*1.00*	*1.00*
32	*0.95*	*1.00*	*0.90*	*1.00*	*0.95*	*1.00*
33	*0.60*	*1.00*	*1.00*	*1.00*	*1.00*	*1.00*
34	*4.03e-6*	*0.47*	*0.58*	*9.14e-2*	*0.20*	*0.43*
35	*1.50e-3*	*0.77*	*0.95*	*0.72*	*0.46*	*0.94*
36	*1.00*	*1.00*	*1.00*	*1.00*	*0.99*	*1.00*
37	*1.00*	*1.00*	*1.00*	*0.94*	*3.18e-2*	*1.00*
38	*1.00*	*1.00*	*1.00*	*1.00*	*0.91*	*1.00*
39	*1.00*	*1.00*	*1.00*	*1.00*	*1.00*	*1.00*
40	*1.00*	*1.00*	*1.00*	*1.00*	*1.00*	*1.00*

Table A.2: Pruning - Phase 3 - The prediction of each model for all scenes presented in Figure A.1.

A.3 Appendix 3

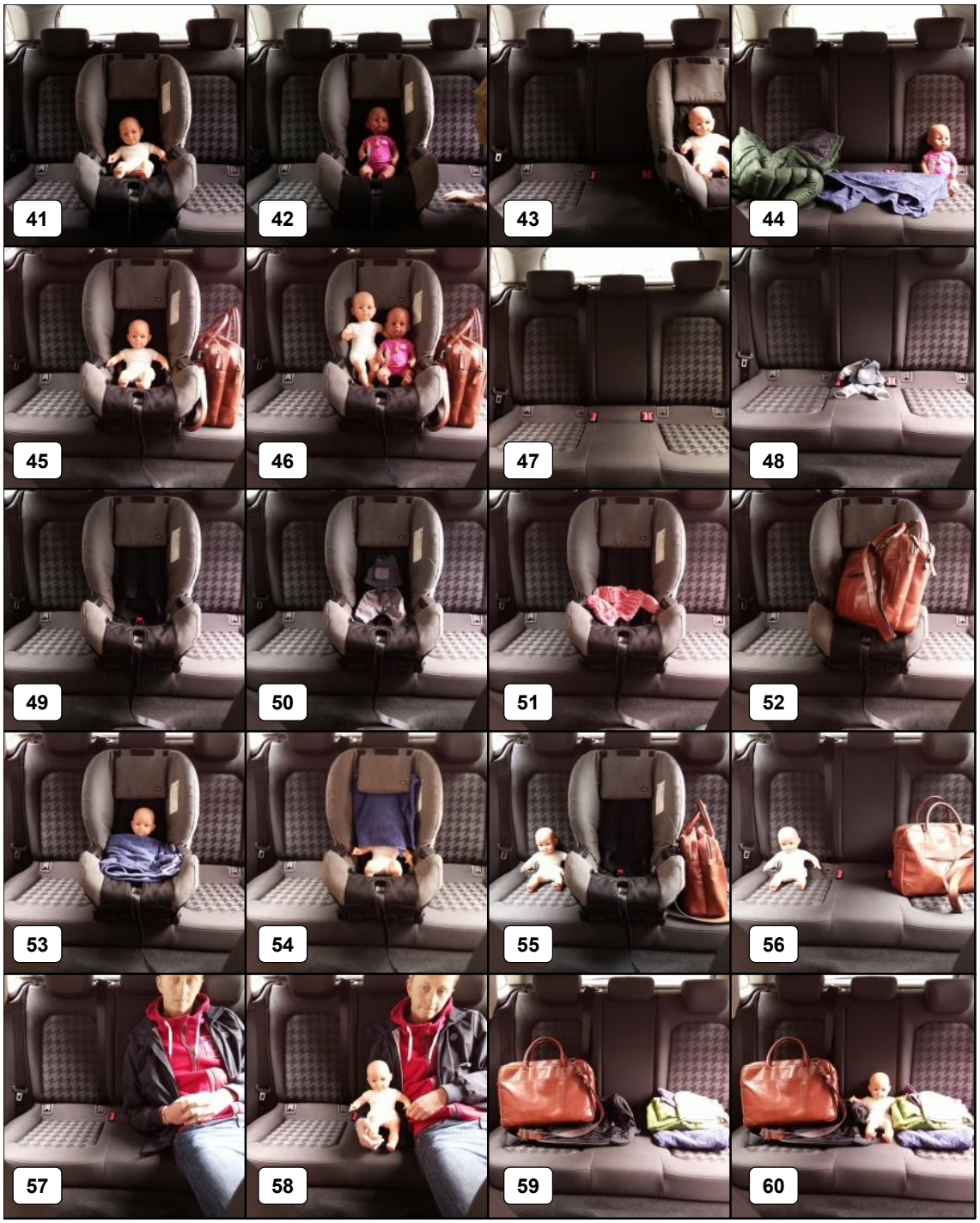

Figure A.3: Results from proof of concept. Images were taken using a new vehicle, never present in the training data.

Scene	Prediction	Scene	Prediction
41	*2.71e-12*	51	*0.94*
42	*2.49e-10*	52	*1.00*
43	*3.44e-4*	53	*4.59e-5*
44	*1.26e-4*	54	*0.95*
45	*1.20e-7*	55	*0.21*
46	*1.41e-6*	56	*1.02e-2*
47	*0.70*	57	*0.61*
48	*1.00*	58	*4.25e-6*
49	*0.85*	59	*0.79*
50	*1.70e-2*	60	*2.89e-3*

Table A.3: Predictions for scenes presented in Figure A.3