

VRIJE UNIVERSITEIT AMSTERDAM

MASTER THESIS

Vehicle Damage Detection using Deep Convolutional Neural Networks

Author:
R.E. van Ruitenbeek

Supervisor:
Prof. Dr. S. Bhulai

Second reader:
Dr. M. Hoogendoorn

External Supervisor:
M.Sc. N.L. de Bruin

Faculty of Science
Master Business Analytics

July 24, 2019

Vehicle Damage Detection using Deep Convolutional Neural Networks

Author:
R.E. van Ruitenbeek

Supervisor:
Prof. Dr. S. Bhulai

External Supervisor:
M.Sc. N.L. de Bruin

*Submitted in fulfilment of the requirements
for the degree Master Business Analytics*

Vrije Universiteit Amsterdam
Faculty of Science
Master Business Analytics
De Boelelaan 1085
1081 HV Amsterdam

Pon Holdings BV
Pon Datalab
Rondebeltweg 31
1329 BN Almere

July 24, 2019

VRIJE UNIVERSITEIT AMSTERDAM
Faculty of Science
Master Business Analytics

Abstract

This paper investigates the applicability of deep learning to detect vehicle damages. 5,000 images, with more than 10,000 objects are used to draw a comparison between different deep learning models. A total of 13 damage classes are incorporated in this research, showing a strong performance difference between the classes. Using different transfer learning approaches, we optimise the damage detection performance. A comparison between human performance and the deep learning approach is conducted, showing that the deep learning model achieves comparable performance. Additionally, an evaluation is conducted at the light street of Pon Logistics, showing several limitations of the model under strong light conditions.

Keywords: Computer Vision, Image Recognition, Object Detection, Deep Learning, Vehicle Damage Detection

Acknowledgements

This thesis has been written to fulfil the requirements for the master Business Analytics at Vrije Universiteit Amsterdam. This thesis has been conducted from February 2019 till July 2019 for the amount of 36 EC.

I would like to thank my university supervisor, Prof. Dr. S. Bhulai, for the support and guidance throughout this research. Furthermore, I would like to thank M.Sc. N.L. de Bruin for fulfilling the role of external supervisor at Pon Datalab. Lastly, I would like to thank both Pon Holdings BV and Pon Logistics for the opportunity to conduct this research and the support and time provided by a variety of employees to make this possible.

Contents

Abstract	iii
Acknowledgements	v
1 Introduction	1
2 Problem Statement	3
2.1 Company Background and Context	3
2.2 Objective	4
2.3 Potential Value	5
3 Literature	7
3.1 Damage Detection	7
3.2 Neural Networks	10
3.2.1 Activation Functions	11
3.2.2 Forward Propagation	11
3.2.3 Loss Function	12
3.2.4 Backward Propagation	13
3.2.5 ANN for Image Processing	15
3.3 Convolutional Neural Network (CNN)	15
3.3.1 Layers	15
3.3.2 Classification Models	17
3.4 Object Detection	20
3.4.1 Sliding Window	20
3.4.2 Region-based Convolutional Neural Network (R-CNN)	20
3.4.3 Single Shot multi-box Detector (SSD)	21
3.4.4 You Only Look Once (YOLO)	24
3.5 Evaluation	26
3.6 Training Neural Networks	28
3.6.1 Regularisation	28
3.6.2 Augmentation	29
3.6.3 Transfer Learning	30
3.6.4 Hyperparameter Tuning	31
4 Data	33
4.1 Damage Dossiers	33
4.2 Damage Web	34
4.3 Light Street	35
4.4 Pon Logistics Master Data	37
4.5 Preprocessing	38
4.5.1 Train and Validation Split	39
4.5.2 Annotation Process	39

5	Methodology	41
5.1	Research Design	41
5.1.1	Image Preprocessing	42
5.1.2	Models	43
5.1.3	Implementation	44
5.2	Hyperparameter Optimisation	46
5.2.1	General Hyperparameters	46
5.2.2	Augmentation	46
5.2.3	Transfer Learning	47
5.2.4	Anchor Boxes	47
5.3	Evaluation	48
6	Results	49
6.1	Initial Parameter Tuning	49
6.1.1	Input Preprocessing	50
6.1.2	Hyperparameter Optimisation	51
6.2	Model Comparison Damage Web	53
6.3	Model Comparison Damage Dossiers	56
6.4	Model Comparison Combined	59
6.5	Employee Performance	59
6.6	Light Street Performance	62
6.7	Inference Speed	64
7	Conclusion and Discussion	65
7.1	Conclusion	65
7.2	Discussion	66
	Bibliography	69
A	Activation Functions	73
B	Model Architectures	74
C	Web Image Extraction	77
D	Excerpts from the Damage Datasets	78
D.1	Damage Dossiers	78
D.2	Damage Web	79
E	Annotations	80
E.1	Annotation Classes	80
E.2	Bounding Box Dimensions	81
F	Excerpts from the Augmented Data	82
G	Anchor Priors	83
H	Excerpts from the Predictions	84
H.1	Damage Web	84
H.2	Damage Dossiers	85

List of Abbreviations

- Adam** Adaptive Moment Estimation. 14
- AlexNet** Alex Network. 17, 18
- ANN** Artificial Neural Network. vi, 7, 11, 15
- API** Application Protocol Interface. 44
- AUC** Area Under the Curve. 27
- AWS** Amazon Web Services. 44, 64
- BCE** Binary Cross-Entropy. 12, 13
- BP** Backward Propagation. 13, 14, 28, 31, 48
- BS** Batch Size. 41, 56
- CAD** Computer-Aided Design. 9
- CAE** Convolutional AutoEncoders. 7
- CCE** Categorical Cross-Entropy. 12, 13
- CNN** Convolutional Neural Network. vi, 7–9, 15–17, 21, 31, 42, 44, 46, 67
- COCO** Common Objects in Context. 15, 54
- CPU** Central Processing Unit. 44, 64
- DNN** Deep Neural Network. 11
- DSSD** Deconvolutional Single Shot Detector. 23, 24
- ECR** Elastic Container Register. 44
- FCL** Fully Connected Layer. 11, 15
- FP** Forward Propagation. 11, 28
- FPS** Frames Per Second. 19, 21, 25, 36, 45
- FSSD** Feature Fusion Single Shot multi-box Detector. 23
- GPU** Graphical Processing Unit. 12, 44, 64
- HD** High Definition. 37
- ILSVRC** ImageNet Large Scale Visual Recognition. 17, 20

IoU Intersection over Union. 27, 48, 51

IP Internet Protocol. 37

LR Learning Rate. 41, 56

LRN Local Response Normalisation. 9

mAP Mean Absolute Precision. 9, 20, 21, 24, 25, 27, 44, 48–56, 58, 59, 65, 67, 68

MP Mega Pixel. 37

MSE Mean Squared Error. 12

PASCAL VOC 2012 Pattern Analysis, Statistical modelling and Computational Learning Visual Object Classification. 8, 15, 20, 21, 25, 38, 54

PoE Power over Ethernet. 36, 37

PPA Pyramid and Patching Augmentation. 9

R-CNN Region-based Convolutional Neural Network. vi, 20, 21, 24, 25, 27, 44

ReLU Rectified Linear Unit. 11

ResNet Residual Network. 18, 19

RFB-SSD Receptive Field Block Single Shot multi-box Detector. 24

RMSprop Root Mean Square Propagation. 14, 46

S3 Simple Storage Solution. 45

SGD Stochastic Gradient Descent. 14, 31

SQL Structured Query Language. 37

SSD Single Shot multi-box Detector. vi, 21–25, 27, 42–44, 46–48, 53

SVM Support Vector Machine. 21, 30

TanH Hyperbolic Tangent. 11

VGG Visual Geometry Group. 18, 19, 23

YOLO You Only Look Once. vi, 9, 24, 25, 27, 41–44, 47, 48

Chapter 1

Introduction

During the last decade, deep learning models gained popularity due to the increased computational power at large scale data centres. This opened up opportunities for the area of image classification, to classify images based on the content, and later object detection to locate different objects within the image. The application of image classification and object detection increased in the last few years to a wide variety of domains. A lot of research has been conducted in the field of cancer detection (Ragab, Sharkas, Marshall, & Ren, 2019) and Brinker et al. (2018) and self-driving cars.

So far, limited research has been conducted on the automatic detection of vehicle damages. Patil, Kulkarni, Sriraman, and Karande (2017) and De Deijn (2018) used deep learning to classify vehicle damages based on a limited dataset from the internet. Li, Shen, and Dong (2018) went beyond damage classification and added damage localisation by the use of object detection. Although Li et al. (2018) used damage detection, they used only two classes: damaged or undamaged. We extend previous research in two ways. Firstly, we use a significantly larger dataset, by extending the vehicle damages from the internet with internal data from Pon Logistics. Secondly, we apply damage detection with 13 different damage classes. With the second contribution, we overcome the intra-class interference of Patil et al. (2017) and De Deijn (2018) and extend the research of Li et al. (2018).

In this research, the following research question is answered: How accurately can deep learning detect vehicle damages and how can these models be used to improve the logistics process?

This research shows that deep learning is able to detect vehicle damages accurately. More precisely, it shows that deep learning can locate and classify vehicle damages on a detailed level. Furthermore, this research indicates a large performance difference between deep learning models when applied to vehicle damage detection. It is shown that the deep learning approach meets human performance on 2D images. Furthermore, the damage detection proved its applicability to reduce the intra-class interference.

This research starts with a detailed problem statement in Chapter 2, followed by a literature study in Chapter 3. The datasets are presented in Chapter 4 and the selected methods and research design is presented in Chapter 5. Lastly, the results are presented in Chapter 6, followed by a conclusion and recommendation for further research in Chapter 7.

Chapter 2

Problem Statement

This section describes the host company, Pon Logistics, and the context of the problem in Section 2.1. The objective of this research, together with the research question, is formulated in Section 2.2. The potential value for Pon Logistics is summarised in Section 2.3.

2.1 Company Background and Context

Pon Logistics assists a broad range of companies in automating and improving their logistics process. One of their key businesses is the vehicle terminal in Leusden (The Netherlands), responsible for handling over 120,000 vehicles a year. Vehicles are being imported from Volkswagen factories across Europe and transported to Leusden by truck or train. On arrival, all vehicles receive a detailed quality control and are stored at the vehicle terminal, having a total capacity of 7,500 vehicles. A broad range of services¹ is applied to the vehicles, before being transported to the end customer. The logistics process is summarised in Figure 2.1.

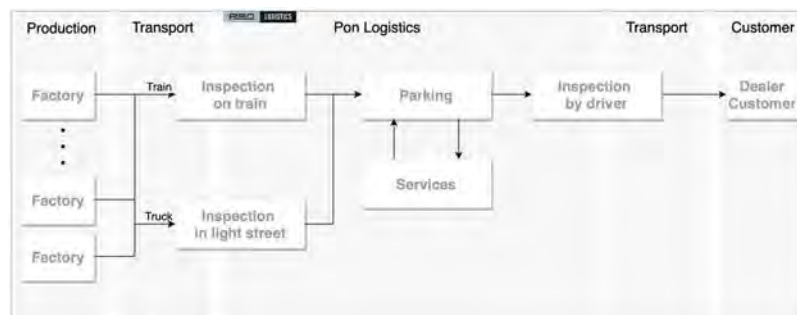


FIGURE 2.1: Flow chart of the Pon Logistics vehicle terminal.

Pon Logistics has a strong focus on innovation, communicated by the Dutch slogan: "Voorsprong door slimme logistieke oplossingen" ("Advantage by smart logistics solutions"). Its focus is placed on a fast process while maintaining high-quality standards on the products. Part of the process is the inbound quality control in the Light Street. At the quality control, each vehicle is driven from the incoming truck and placed in the Light Street. A damage check is performed by Pon employees, taking up a few minutes to complete. Furthermore, the vehicle will be formally registered. When logged in to the system, the vehicle is officially handed over from the transporting company to Pon Logistics. This step is crucial in the process, since any damage that is not detected will be at the expense of Pon Logistics.

¹Switching the tires to Continental, changing the interior such as leather or seat heating, etc.

However, the current process is time-consuming. At peak moments, the queue for the Light Street can extend to tens of vehicles, making the waiting time long. As the truck driver has to drive approximately eight vehicles from the truck, a total of eight visits to the Light Street is required. This process takes up a large proportion of the unloading time for the truck driver. Additionally, long waiting queues could potentially result in more employee pressure and therefore less accurate damage detection. Currently, approximately 2 percent of the vehicles are detected with damage, when received. Approximately 1.5 percent of the vehicles are detected with damage later in the process, which is either undetected damage at the receiving process, or damage created during the logistics process after the Light Street. Currently, not all vehicles are routed through the Light Street as it is a rather time-consuming task. Therefore, applying automated vehicle damage detection could speed up this process and make it beneficial to screen all incoming vehicles.

2.2 Objective

This research focuses on identifying to which extend vehicle damages can be detected by the development of a deep learning model. As limited previous research is available in this field of study, no clear performance target can be presented upfront. Therefore, the main focus will be to identify the applicability of deep learning on vehicle damage detection and to identify its boundaries and limitations. Improving the logistics process is not limited to an increased detection rate, or a speed improvement, but also focuses on the improvement of working conditions for the employees. Some areas are hard to inspect by a human, where especially the roof and the rims are a bottleneck in the process.

Based on the above objectives, the following research question is formulated:

Research question

How accurately can deep learning detect vehicle damages and how can these models be used to improve the logistics process?

It is expected that the model will be working in parallel with the employees, rather than replacing them. Therefore, it is desired that the model constructs the predictions for the damage detection, within the time window where the vehicle is present in the Light Street. As each vehicle is only a few minutes in the Light Street, it is desired that the analysis will be performed within a minute to give employees enough time to interpret the output. The faster the model creates the predictions, the more potential there is to speed up the process in the future. Having a fast detection speed might eventually reduce the average vehicle time in the Light Street. Although the detection speed is of relevance, Pon Logistics is mostly concerned with identifying the ability of an automatic system, making the accuracy the most important evaluation measure.

The model will have to deal with varying conditions. Firstly, damage images of Pon Logistics are collected outside and are therefore subject to reflection and weather conditions. Secondly, the vehicles in the Light Street are transported by truck and therefore exposed to weather types such as rain and ice, covering the vehicle surface. Lastly, a protection cover is sometimes placed partly over the vehicles, making the detection of damage less evident. As cover damage is a signal of vehicle damage, the detection of cover damage should be included in this research.

An initial setup for automatic vehicle inspection will be installed at the Light Street, to evaluate the trained models in the correct environment. This gives the

ability to compare the performance between the damage images and the Light Street video stream.

2.3 Potential Value

When the developed model proves to be able to recognise vehicle damages accurately enough, the software can be used to: improve the working conditions of the employees, reduce the number of missed damages, and to speed up the process. The software can be used for difficult areas such as the roof, tire rim, and spoiler. This makes the damage detection for employees less intensive, as bending (rim and spoiler) and climbing (roof) can be limited. The quality of the detection process can be improved when the system runs as a decision support system, showing damage locations through an interface. Lastly, the overall inspection duration can potentially be reduced when moving to more automatic screening, as the screening process can be performed in a matter of seconds.

Speed improvement opens up opportunities to screen all inbound and outbound vehicles, due to its scalability. Passing all inbound vehicles through the Light Street, can improve the detection rate and therefore directly reduces the expenses for Pon Logistics. Adding additional outbound vehicle inspection could improve the quality delivered towards customers. Lastly, external transportation companies benefit from the speedup in the process, as waiting lines can be reduced. Reducing the waiting lines makes the truck unloading significantly faster.

Automatic systems are largely scalable, being robust to loss of knowledge by retiring employees. In the future, the knowledge can be applied in other companies of Pon Holdings BV, such as the rental branch. A joined model can be constructed, combining the knowledge and training data of all business units.

Chapter 3

Literature

In this chapter, a literature review is conducted on damage detection in Section 3.1. The development and technical details of Artificial Neural Networks, as well as its extension to image processing with Convolutional Neural Networks, is described in Section 3.2 and 3.3, respectively. A variety of evaluation measures is presented in Section 3.5, where the chapter ends with an approach for training ANNs in Section 3.6.

3.1 Damage Detection

To our best knowledge, only Patil et al. (2017), Li et al. (2018) and De Deijn (2018) used a deep learning approach to identify vehicle damages from 2D images. That is, Patil et al. (2017) evaluated the ability of Convolutional Neural Networks (CNNs) (see Section 3.3) in classifying vehicle damages. They used three different approaches to classify the damage into seven damage categories¹ and one undamaged category. They used a total of 1,200 images containing vehicle damage and 1,271 images without damage. They trained a CNN from scratch, resulting in a classification accuracy of 72.46 percent. Secondly, Patil et al. (2017) used Convolutional AutoEncoders (CAE) which made use of unsupervised learning methods and requires, therefore, less training data. A slightly higher classification accuracy (73.43 percent) was obtained.

The third applied method, of Patil et al. (2017), made use of transfer learning (see Section 3.6.3) to fine-tune different pre-trained models with the set of vehicle damages. This method increased the classification accuracy to 88.24 percent. Furthermore, they showed that pre-trained models, trained on a broad object range, outperform pre-trained models trained specifically for vehicle classification. This broad object range is of relevance as vehicle damages detection might require different shapes compared with recognising vehicles itself (Patil et al., 2017). Lastly, they used an ensemble method which combines all trained models by use of a linear combination. A final classification accuracy was achieved of 89.5 percent. They describe that small damages are often misclassified, since the proportion of damage is relatively low, compared to the undamaged proportion of the vehicle. Furthermore, they explained that the classification task is non-trivial due to the large inter-class similarity of damages.

Although De Deijn (2018) claimed to be the first in analysing 2D vehicle damages, they conducted a research comparable with Patil et al. (2017). De Deijn (2018) added the use of a cascade of three models, to perform damage classification. The first classifier recognises if the image contains a vehicle. The second classifier indicates

¹Categories: Bumper dent, Door dent, Glass shatter, Head-lamp broken, Tail-lamp broken, Scratch, Smash.

if a damage is present on the images, whereas the third model classifies the type, location and size of the damage. For their research, 29,000 images without a vehicle, 16,185 images with undamaged vehicles, and 1,007 images with damaged vehicles have been used. To increase the dataset size, they used image augmentation which increases the dataset size by constructing artificial images from the original images. They solely used horizontal flipping as image augmentation to avoid overfitting.

De Deijn (2018) used pre-trained CNNs for all three classifiers and applied transfer learning in a similar way as Patil et al. (2017). De Deijn (2018) emphasised the effect and importance of the parameter tuning for all three models and focused on achieving high accuracy while working with limited data. They achieved an impressive 99.04 percent test accuracy on the classification of vehicles. Damaged vehicles were excluded from the training and testing, as those images were mainly deviating from the other images. When applied to the damaged vehicles, their system still achieved 89.1 percent test accuracy. To compare, Chen, Zhu, Papandreou, Schroff, and Hartwig (2018) from *Google Research* achieved 95.5 percent accuracy on the car category of PASCAL VOC 2012 (Everingham, Van Gool, Williams, Winn, & Zisserman, n.d.). However, it should be mentioned that the samples of De Deijn (2018) might have been easier, compared with PASCAL VOC 2012. The majority of images from De Deijn (2018) contains a single large vehicle and no other classes.

The damage detection, of De Deijn (2018), suffers from a large class imbalance of 94.2 percent undamaged vehicle images. Predicting all images as undamaged will already achieve a 94.2 percent classification accuracy. Despite this, they achieved 99 percent precision and 89.6 percent recall, indicating that the system is able to recognise the damaged vehicles as such. Classification of damage classes², location³ and size⁴ achieved an accuracy of respectively 75.1 percent, 68.7 percent and 54.2 percent (De Deijn, 2018). They explained that inter-class similarity complicates the classification process, especially for location and size. This statement is in line with the statement of Patil et al. (2017) that, location prediction can be complicated due to boundaries between two locations. Furthermore, De Deijn (2018) gave rise to the idea that the model might not predict the location of the damage, but rather the view on the vehicle.

De Deijn (2018) showed that the damage size classification suffers from large interference between the categories small and medium. They explain that this might be biased by the manual labelling process. Furthermore, the difference in zoom level to the vehicle might complicate this classification process. Having close-up pictures might complicate the true size estimation, compared with a full vehicle view. Furthermore, having only one label for each image increases the class interference, for images where two types of damage are present. Vehicles with multiple damage classes are assigned by De Deijn (2018) to the dominant damage class in the image. They showed that a dent and scratch get largely misclassified, which might occur due to both classes being present.

Li et al. (2018) conducted similar research, as compared to Patil et al. (2017) and De Deijn (2018). However, Li et al. (2018) went beyond image classification and added damage localisation by use of bounding boxes. They used this technique to develop a system pipeline, able to identify similar damages to cope with insurance fraud. For this research, they used manually annotated images from the internet (1790) and extended this with 98 images captured in parking lots. Their dataset contained mainly three types of damage: scratch, dent, and crack. Although the dataset

²Classes: Dent, Glass, Hail, and Scratch.

³Location: Front, Rear, Side, and Top.

⁴Size: Large, Medium, Small.

contained multiple classes, they focused on detecting damage itself and did not apply multi-class detection. Li et al. (2018) explain that localising vehicle damages is a challenging task as, unlike normal object detection, each damage can be of different shape. They used the one-stage model: You Only Look Once (Section 3.4.4), to detect if a damage is present and where the damage is located. An advantage of object detection over image classification of Patil et al. (2017) and De Deijn (2018) is that the explained intra-class interference is reduced. Object detection enables to label each damage separately, making sure that the different damages are localised and classified independently.

Although Li et al. (2018) used slightly more images, compared to Patil et al. (2017) and De Deijn (2018), they applied transfer learning as well. To handle the various light conditions, they applied Local Response Normalisation (LRN) layers which reduce the effect of light reflections being misclassified as damage. They showed that adding these layers increases the precision from 32.75 percent to 37.96 percent and the recall from 57.58 percent to 81.75 percent. Their example detections showed that damages can be localised correctly, however, predicting an accurate bounding box can be complicated. This falls in line with the statement of Patil et al. (2017) that detecting damage is non-trivial due to the different shapes of the damages.

Jayawardena (2013) applied a completely different methodology to detect vehicle damages, compared with previously mentioned research. They proposed to use 3D Computer-Aided Design (CAD) models of undamaged cars and used their 3D pose estimation algorithm to align the damaged vehicle with the ground truth CAD model. They researched the effect of reflection detection and isolation before applying the CAD models as they discovered a large inter-object reflection on the vehicle surfaces. They proposed a method, using a multi-view on the vehicle, to distinguish damage edges from reflection edges. Although the method succeeds in recognising damage correctly, they showed that a large proportion of reflections gets misclassified as damage. Although no performance accuracy is given, Jayawardena (2013) visually showed the potential of their proposed 3D CAD models. Due to the lack of performance measurement, this method cannot be compared to the 2D methods applied by Patil et al. (2017), Li et al. (2018), and De Deijn (2018). Jayawardena (2013) explained the importance of an accurate estimation of the 3D CAD models for the final detection accuracy. The proposed and used laser scanner costs around USD 5,000 and outperforms the cheaper versions (around USD 100) by far. They stress the importance of an accurate model, especially for minor damages.

Going beyond vehicle damage detection, extensive research on damage detection can be found in general. Cha, Chen, and Büyüköztürk (2017) proposed a computer vision architecture to detect cracks in concrete structures, by use of edge detection filters and Kalman filtering of frequencies. A second research was conducted using CNNs to detect cracks under various light conditions, making it more robust than the Kalman filtering approach.

Shihavuddin et al. (2019) used a CNN to detect wind turbine blade damages from high-resolution drone images. They showed that the automatic system almost achieves human performance accuracy while reducing downtime and increasing detection speed. Different base network architectures with transfer learning were applied, which showed that a strong performance difference exists between the backbones. Furthermore, they focused strongly on data augmentation and applied a Pyramid and Patching Augmentation (PPA) to dramatically increase the Mean Absolute Precision (mAP) (see Section 3.5) from 25.9 percent to 70.5 percent. They applied this augmentation to cope with the small damage objects in images of high resolution.

3.2 Neural Networks

Neural Networks have grown in popularity over the last few decades. This is partly driven due to the growth in computational power, needed for large scale training. Neural networks are built around the central idea of a Perceptron, which has been introduced by Rosenblatt (1958). A Perceptron can be seen as the simplest available neural network, which is able to linearly classify data points in a similar way as the Least Squares methods (Aggarwal, 2018, p. 5). In contrast to the Least Squares methods, a Perceptron guarantees to find a perfect separation of the data points if the data is linearly separable. Figure 3.1 visualises a Perceptron, having X_i with $i \in \{1, 2, \dots, n\}$ input signals, a bias B and an activation function. Each input signal (x_i or B) contributes with weight w_i to the output variable.

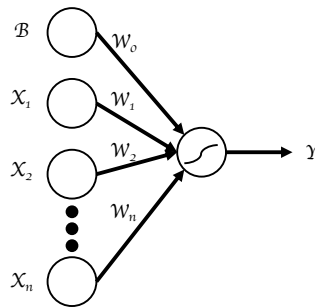


FIGURE 3.1: Perceptron with bias.

Calculation of the target value for the perceptron is performed in two stages: calculating the weighted sum of the input signals by Equation (3.2) and transforming the weighted sum by activation function ϕ to output \hat{y} (Equation (3.1)). The Perceptron makes use of the Heaviside step function, defined in Equation (3.3), mapping the weighted sum into a binary output. Rosenblatt (1958) explained that the Perceptron, with use of the Heaviside function, mirrors the neurons in the brain. The input signal activates the neuron when the signal is strong enough ($z \geq 0$) and sets the output equal to one. The evaluation of \hat{y} can be simplified to matrix notation, resulting in Equation (3.4), with $x_0 = 1$ and $B = w_0$.

$$\hat{y} = \phi(z) \quad (3.1)$$

$$z = \sum_{i=1}^n x_i w_i + b \quad (3.2)$$

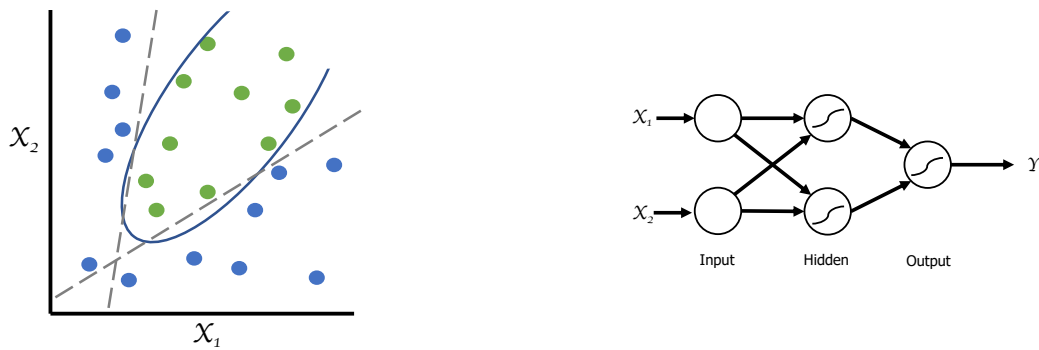
$$\phi(z) = \begin{cases} 1 & z \geq 0 \\ 0 & z < 0 \end{cases} \quad (3.3)$$

$$\hat{y} = \phi(\mathbf{x} \mathbf{w}) \quad (3.4)$$

Although the perceptron algorithm guarantees a linear separable decision boundary, there is no ability to model non-linear data. Minsky and Papert (1969) showed, by use of the classical XOR-problem, that the single-layer perceptron fails. Stacking multiple Perceptrons on top of each other will give the ability to model non-linear decision boundaries, such as the boundary given in Figure 3.2a. Modelling this boundary, a slightly more complex model is required, containing an input layer

(the input signal), a hidden layer, and an output layer combining the output of the hidden layer. Figure 3.2b represents the required architecture to model the non-linear decision boundary of Figure 3.2a.

The model presented in Figure 3.2 can be extended arbitrarily large by varying the number of hidden layers and the number of nodes in each layer. A layer where all nodes of layer l are connected with all nodes of layer $l + 1$ is called: Fully Connected Layer (FCL).



(A) Non-linear decision boundary.

(B) Two-layer perceptron model without bias for readability.

FIGURE 3.2: Modelling a non-linear decision boundary.

3.2.1 Activation Functions

Where the perceptron uses the Heaviside step function of Equation (3.3), multi-layer networks can have different activation functions. The choice of the activation function strongly depends on the nature of the output. For example, a binary prediction requires the Sign function or the Heaviside step function and a probability output requires mostly the Sigmoid function (Aggarwal, 2018, p. 11). Using non-linear activation functions, the network layer is able to learn non-linear interaction between the input variables and output variables. See Appendix A for five widely used activation functions. In the early developed ANNs, the Sign, Sigmoid, and Hyperbolic Tangent (TanH) functions were most frequently used (Aggarwal, 2018, p. 12). The Rectified Linear Unit (ReLU) function has largely overruled the early adopted functions due to its ease of training in deep neural networks (Pattanayak, 2017, p. 106). Pattanayak (2017) explains that ReLU is not only fast in training, due to the absence of exponents, but also has an advantage as both Sigmoid and TanH suffer from a vanishing gradient. A Deep Neural Network (DNN), that has multiple hidden layers, benefit largely from the speedup of ReLU over Sigmoid and TanH.

3.2.2 Forward Propagation

Forward Propagation (FP), frequently referenced to as Inference, calculates the network output based on a given input signal. The calculation is executed according to the Perceptron principles explained in Equation (3.3) and Equation (3.4). Figure 3.3 shows a generalised network architecture with multiple input nodes, multiple hidden layers, and multiple output nodes. The notation can be generalised by taking L as the number of hidden layers, M_l as the number of nodes in layer l , and $a_m^{(l)}$

as node m of layer l . The weights between layer $l-1$ and layer l are denoted by \mathbf{W}_l . Furthermore, the input vector \mathbf{x} is defined as $\mathbf{a}^{(0)}$ and the output vector \mathbf{y} as $\mathbf{a}^{(L+1)}$.

Using the defined notation, forward propagation for layer l is generalised from the Perceptron calculation to the matrix notation of Equation (3.5) and Equation (3.6). Followingly, $\mathbf{z}^{(l)}$ represents the weighted sum between all input signals of layer l ($\mathbf{a}^{(l-1)}$) and the corresponding weights (\mathbf{W}^l). Lastly, $\mathbf{z}^{(l)}$ gets mapped into a new domain by passing it through the activation function ϕ^l . The matrix formulation is of special interest for large networks, as matrix computations can efficiently be computed with GPUs.

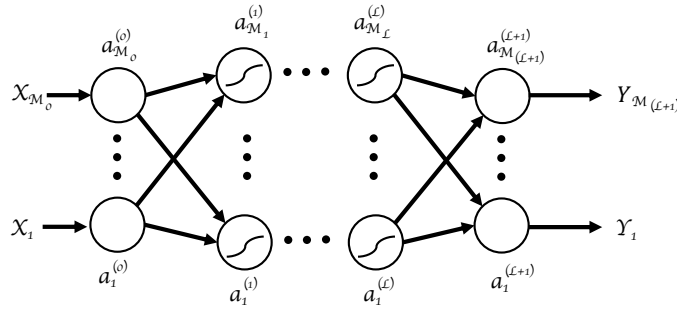


FIGURE 3.3: Multi-layer network architecture, without bias nodes for readability.

$$\mathbf{z}^{(l)} = \mathbf{w}^{(l)} \mathbf{a}^{(l-1)} \quad (3.5)$$

$$\mathbf{a}^{(l)} = \phi^{(l)}(\mathbf{z}^{(l)}) \quad (3.6)$$

3.2.3 Loss Function

Similar to regression models, the loss function defines the deviation between the target value (y) and the predicted value (\hat{y}). The defined network in Figure 3.3 makes use of multiple output neurons, to predict multiple output values. A deviation between the predicted value and the target value is, therefore, a vector of shape $(1, K)$, with $K = M_{L+1}$ (the number of targets to predict). Different loss functions can be used, depending on the nature of the target values. Regression tasks commonly use the Mean Squared Error (MSE), or a related measure, for the loss value. The MSE loss function is defined in Equation (3.7). The MSE takes for each data sample i the squared deviation between the target values and predicted values, where $j \in \{1, 2, \dots, K\}$ (Duffner, 2009, p. 64).

$$E_{MSE} = \frac{1}{N} \sum_{i=1}^N \frac{1}{K} \sum_{j=1}^K (\hat{y}_{ij} - y_{ij})^2 \quad (3.7)$$

For classification tasks, Binary Cross-Entropy (BCE) is most frequently used which gives a higher loss value if the incorrect class receives a high confidence prediction. The BCE is shown in Equation (3.8), where $y_i = 1$, if example i corresponds to class 1. K is omitted, as the number of output nodes is one. The BCE can be generalised to multiple classes (C) by Equation (3.9), called Categorical Cross-Entropy (CCE) (Godoy, 2018). The number of classes should be equivalent to the number of output

nodes and therefore: $C = K$. y_{ij} is 1, if sample i corresponds to class j . $\hat{y}_{ij} \in [0, 1]$ is the predicted probability of sample i belonging to class j .

$$E_{BCE} = - \sum_{i=1}^N (y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)) \quad (3.8)$$

$$E_{CCE} = - \sum_{i=1}^N \sum_{j=1}^C y_{ij} \log(\hat{y}_{ij}) \quad (3.9)$$

3.2.4 Backward Propagation

Backward Propagation (BP), was introduced by Rumelhart, Hinton, and Williams (1986). It minimises the loss function with an iterative approach by updating the weights of the network. Since BP is an iterative approach, that moves towards a lower loss value, it potentially reaches a local minimum and does not guarantee a global minimum. Using the notation from Section 3.2.2, BP is formulated for the general network of Figure 3.3. The BP algorithm calculates the derivative of the error and moves gradually in the opposite direction to minimise the loss.

The BP algorithm starts by calculating the derivative of the error and updating the weights from layer $L + 1$ towards layer 1. We derived the following formulations of the BP algorithm from (Bishop, 2006, pp. 241-249). The computation will firstly be introduced per node and is then generalised towards matrix notation. Though we are only interested in the derivatives of the weights, the derivatives of the error with respect to a^l and z^l are required in order to propagate through the network. Let w_{ij} be defined as the weight from node i in layer l to node j in layer $l + 1$. Furthermore, we define z_j^l as z of node j in layer l and a_j^l similarly. By taking the derivative of E with respect to w , the derivative of Equation (3.10) arises. By substituting Equation (3.5), the formulation is further reduced to the right-hand side.

$$\frac{\partial E}{\partial w_{ij}^l} = \frac{\partial E}{\partial z_j^l} \frac{\partial z_j^l}{\partial w_{ij}^l} = \frac{\partial E}{\partial z_j^l} a_j^l \quad (3.10)$$

The derivative of the error, with respect to the output layer, is shown in Equation (3.11). This derivative simply reduces towards the deviation between the target and prediction.

$$\frac{\partial E}{\partial z_j^{L+1}} = \frac{\partial \frac{1}{2} \sum_{i=1}^N (y_i - \hat{y}_i)^2}{\partial y_j} = y_j - \hat{y}_j \quad (3.11)$$

Before defining the derivative of E with respect to z^l , the derivatives of z_j^{l+1} with respect to a_i^l and the derivative of a_i^l with respect to z_i^l need to be defined. Equation (3.12) is obtained by substituting Equation (3.5). Equation (3.13) is obtained by the substitution of Equation (3.6).

$$\frac{\partial z_j^{l+1}}{\partial a_i^l} = \frac{\partial \sum_i^N w_{ij}^{l+1} a_i^l}{\partial a_i^l} = w_{ij}^{l+1} \quad (3.12)$$

$$\frac{\partial a_i^l}{\partial z_i^l} = \frac{\partial \phi^l(z_i^l)}{\partial z_i^l} \quad (3.13)$$

The derivative of the output error, with respect to node i in the hidden layer l , is defined recursively by Equation (3.14). The previously defined derivatives of Equation (3.12) and Equation (3.13) are used to simplify the equation.

$$\frac{\partial E}{\partial z_i^l} = \sum_j \frac{\partial E}{\partial z_j^{l+1}} \frac{\partial z_j^{l+1}}{\partial a_i^l} \frac{\partial a_i^l}{\partial z_i^l} = \sum_j \frac{\partial E}{\partial z_j^{l+1}} w_{ij}^{l+1} \frac{\partial \phi^l(z_i^l)}{\partial z_i^l} \quad (3.14)$$

The above defined BP can be transformed to matrix notation by reformulating: $\frac{\partial \mathbf{E}}{\partial \mathbf{z}^{(L+1)}} = \Delta \mathbf{z}^{(L+1)}$, $\frac{\partial \mathbf{E}}{\partial \mathbf{z}^{(l)}} = \Delta \mathbf{z}^{(l)}$, $\frac{\partial \mathbf{E}}{\partial \mathbf{w}^{(l+1)}} = \Delta \mathbf{w}^{(l+1)}$, and $\frac{\partial \phi^l(z_i^l)}{\partial z_i^l} = \phi'(\mathbf{z}^{(l)})$. By substituting the reformulated equation into Equation (3.10), (3.11), and (3.14), the matrix notation of Equation (3.15)-(3.17) is defined.

$$\Delta \mathbf{z}^{(L+1)} = \hat{\mathbf{y}} - \mathbf{y} = \mathbf{a}^{(L+1)} - \mathbf{y} \quad (3.15)$$

$$\Delta \mathbf{z}^{(l)} = \mathbf{w}^{(l+1)T} \Delta \mathbf{z}^{(l+1)} \cdot \phi^{(l)'}(\mathbf{z}^{(l)}) \quad (3.16)$$

$$\Delta \mathbf{w}^{(l+1)} = \Delta \mathbf{z}^{(l+1)} \mathbf{a}^{(l)T} \quad (3.17)$$

The error is used to optimise the weights throughout the network, based on a set of rules. A variety of optimisation algorithms exists, of which the three most commonly used are Stochastic Gradient Descent (SGD), Adaptive Moment Estimation (Adam), and Root Mean Square Propagation (RMSprop). The optimisation algorithm updates the weights of the network, based on the found values in the backward step.

Building on the notation defined for BP, SGD, as proposed by Rosenblatt (1958), is formulated by Equation (3.18) and Equation (3.19). η is the learning rate and β defines the momentum which accelerates SGD in the right direction, by dampening the oscillations. For each batch of images, $\Delta \mathbf{W}$ is averaged with the previous step by β .

$$\mathbf{v}_{\Delta W_t} = \beta \mathbf{v}_{\Delta W_{t-1}} + (1 - \beta) \Delta \mathbf{W} \quad (3.18)$$

$$\mathbf{W}_{t+1} = \mathbf{W}_t - \eta \mathbf{v}_{\Delta W_t} \quad (3.19)$$

Despite the momentum of SGD to damp oscillations, the learning is often not stable enough. That is, the optimisation is not performed in a straight line towards the minimum. RMSprop tries to overcome these shortcomings by the notion of $\mathbf{S}_{\Delta W_t}$ (Equation (3.20)), which is the exponential average of squares of gradients (Ruder, 2018). Another advantage of RMSprop is the automatic decrease of the gradient as soon as the steps are decreasing, indicating that the algorithm is approaching a (local) minimum. This reduces the need for a manual learning rate reduction. The RMSprop algorithm is derived from the formulation of Ruder (2018) as shown in Equation (3.20) and Equation (3.21) respectively.

$$\mathbf{S}_{\Delta W_t} = \beta \mathbf{S}_{\Delta W_{t-1}} + (1 - \beta) \Delta \mathbf{W}_t^2 \quad (3.20)$$

$$\mathbf{W}_{t+1} = \mathbf{W}_t - \eta \frac{\Delta \mathbf{W}_t}{\sqrt{\mathbf{S}_{\Delta W_t} + \epsilon}} \quad (3.21)$$

Adam combines the benefits of SGD and RMSprop. It takes the benefits of accelerating in the direction of the local minimum (SGD), while reducing the search towards oscillations (RMSprop). The formulation is derived from Ruder (2018) and shown in

Equation (3.22). $\mathbf{v}_{\Delta W_t}$ and $\mathbf{S}_{\Delta W_t}$ are as defined in Equation (3.18) and Equation (3.20), with substitution of β by β_1 and β_2 respectively.

$$\mathbf{W}_{t+1} = \mathbf{W}_t - \eta \frac{\mathbf{v}_{\Delta W_t} \mathbf{W}_t}{\sqrt{\mathbf{S}_{\Delta W_t} + \epsilon}} \quad (3.22)$$

3.2.5 ANN for Image Processing

Applying ANNs to image analysis requires the mapping between the input image and the network input layer of size $1 \times N$. Figure 3.4 shows the most straightforward approach, where each pixel represents one input neuron. The input layer dimension equals $1 \times WH$ where W and H represent the number of image pixels in the horizontal and vertical direction respectively. For colour images, the $W \times H$ dimension extends to $W \times H \times 3$, making the input layer dimension equal to $1 \times 3WH$.

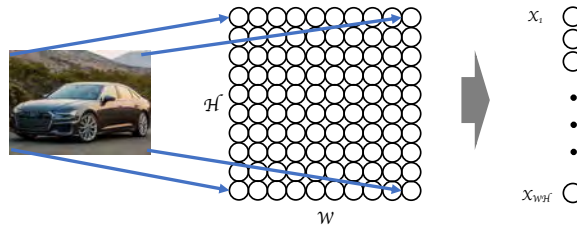


FIGURE 3.4: General approach of an ANN for image processing.

3.3 Convolutional Neural Network (CNN)

Applying ANNs to the field of image analysis resulted in a decreasing improvement on both the PASCAL VOC 2012 and Common Objects in Context (COCO) (Lin et al., 2014) challenges. These challenges received a dramatic improvement when LeNet was introduced by LeCun et al. (1989), being the foundation of the modern CNNs. This network was mainly developed to serve banks by identifying handwritten digits from checks (Aggarwal, 2018, p. 316). CNNs improved both PASCAL VOC 2012's and COCO's performance dramatically. One of the key benefits of CNNs is the ability to share weights across layers, making it possible to learn complex structures with limited parameters. Applying one FCL for a 100×100 image size and a hidden layer of 1024 nodes already requires more than ten million parameters ($100 \times 100 \times 1024 = 10,240,000$). Furthermore, an ANN has trouble identifying general features as spatial image information is not incorporated in the network.

3.3.1 Layers

Similar to ANNs, CNNs are built of multiple layers. The main difference is the spatial structure, preserved in the layers of CNNs. That is, ANNs mainly use FCL, where each neuron of layer $l - 1$ is connected with each neuron of layer l , whereas CNNs use a variety of layer types. Each layer has its own function and some CNNs still use the FCL in the final layers to map the feature into an output vector. The input layer is a matrix of pixel intensities, having the dimension: $width \times height \times colour$, of which the colour dimension is 1 (black and white) or 3 (colour/RGB image).

Convolution

The most important part of the CNN is the convolution layer, hence the term ‘Convolutional Neural Network.’ The central idea of convolution arises from the fact that each pixel shares information with its surrounding pixels. Therefore, a submatrix is taken from layer l and multiplied with a filter to transfer surrounding pixel information to layer $l + 1$ (Gulli & Pal, 2017, pp. 74–75). Convolutions can be calculated by applying a filter to the input layer through the sliding window approach (3.4.1), applying the same filter to multiple areas of the image. Reusing the same filter for multiple areas reduces the number of parameters significantly. Convolutions use multiple hyperparameters to define how each filter is applied to the image and what the output size will be. The used parameters are summarised below, where Figure 3.5 shows the defined parameters with a zero padding of 1. Based on the below parameters, the output dimension can be calculated by Equation (3.23) (Deshpande, 2019). Figure 3.6a displays a 3×3 filter applied on a 5×5 matrix with strides of 1.

- Filter size (F): The height and width of the filter in number of pixels, where the filter is mostly of squared size.
- Number of filters (N_F): The number of applied filters.
- Stride (S): The number of pixels between the centre of each applied filter. A stride of 1 indicates that the filter is moved one pixel, each time the filter is applied. A stride of 2 shifts the filter two pixels between each application.
- Padding (P): Padding increases the matrix size, where a zero padding of i adds i rows and columns on all sides of the matrix. Padding is used on the input matrix to preserve the same output size after the convolution is applied.

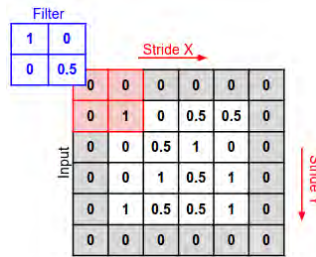


FIGURE 3.5: A 2×2 filter applied on a matrix of size 4×4 with zero padding of 1 (Perera, 2018).

$$(W_{l+1}; H_{l+1}; D_{l+1}) = \left(\frac{W_l - F + 2P}{S} + 1; \frac{H_l - F + 2P}{S} + 1; N_F \right) \quad (3.23)$$

Pooling

A Pooling layer sub-samples the volume spatially, by applying an operation on sub-matrices. The main application of pooling is the dimensional reduction, which decreases the required memory and computational cost of the network. The most commonly used operation is max pooling, which calculates the maximum over the input matrix. Other types are average pooling and L2-normalisation pooling (Deshpande, 2019). A visual representation of 2×2 pooling is shown in Figure 3.6b. Although pooling has been a key feature in CNNs, Deshpande (2019) explained that several researchers argue to remove pooling from the network, as the performance improvement is relatively low compared with the added computational complexity. The output dimension ($W_{l+1} \times H_{l+1} \times D_{l+1}$) after applying the pooling is determined by the input dimension $W_l \times H_l \times D_l$, the filter size F , and the stride size S . The calculation is displayed in Equation (3.24).

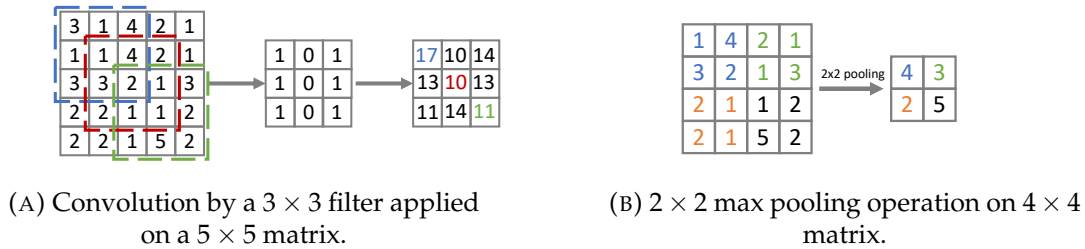


FIGURE 3.6: CNN layers: 3×3 convolution (left) and 2×2 max pooling (right).

$$(W_{l+1}; H_{l+1}; D_{l+1}) = \left(\frac{W_l - F}{S} + 1; \frac{H_l - F}{S} + 1; D_l \right) \quad (3.24)$$

3.3.2 Classification Models

A broad range of classification algorithms has been developed over the past ten years, in which convolutional networks started playing an important role from 2013 onward. A typical CNN consists of a convolutional base, which extracts features from the image and a classifier which transforms the detected features into a prediction. Some of the most important networks will be discussed briefly, since classification networks mainly serve as a basis for object detector algorithms. All below performance measures are reported for the ImageNet Large Scale Visual Recognition (ILSVRC) challenge, commonly referenced to as ImageNet. This competition requires the classification model to classify 50,000 images in 1,000 classes (Krizhevsky, Sutskever, & Hinton, 2012). All below-mentioned error rates are defined as the classification error for the ILSVRC challenge.

Krizhevsky et al. (2012) published the first deep learning CNN, reducing the classification error from 26.2 percent to 15.3 percent. The network, called **AlexNet**, used only five convolutional layers and max-pooling layers, with an addition of three fully connected layers. They applied three different filter sizes: 11×11 , 5×5 , and 3×3 .

In 2014, the error rate dropped below ten percent for the first time, creating a significant improvement over AlexNet. The first place was received by **GoogleNet**, also referenced to as Inception V1. They used 22 layers with small convolutions to reduce the size of the network from 60 million parameters (AlexNet) to 4 million (Szegedy et al., 2015). They introduced the Inception module, which combines features from multiple filter sizes. Different filters are applied on the input layers and all results are concatenated to form one output. The Inception module is visualised in Figure 3.7a.

VGG-16, proposed by Simonyan and Zisserman (2015) achieved the second place with an error rate of only seven percent. They showed that reducing the size of convolutional filters to 3×3 , alongside an increase in the number of layers, significantly reduces the number of parameters while maintaining the same feature extraction. They showed that a 7×7 filter results in 49 parameters, while 3 consecutive layers of 3×3 filters accumulate to 27 parameters. This approach reduces the number of parameters significantly while covering the same area as the 7×7 filter. Depending on the configuration, they used 11-19 filter layers, with three fully connected layers on top. Despite its effective algorithm, between 133 and 144 million parameters are present, mainly due to the three fully connected layers taking up around 85 percent of the total number of parameters.

He, Zhang, Ren, and Sun (2016) evaluated the effect of an increasing network depth on the classification error. They discovered that an increased network depth, increases the error rate, due to the complexity of both training and optimising the network. He et al. (2016) introduced Residual Learning to connect convolutional output values with the input values, by use of an identity mapping. They named the network: **ResNet**. Using Residual Learning, convolutional layers make smaller changes to the input, transferring more information towards the deeper layers. The residual learning is imposed by the use of Residual blocks, where He et al. (2016) used two consecutive layers in each Residual Block. The used Residual Block is shown in Figure 3.7b. An impressive error rate of only 3.57 percent has been achieved, with their configuration of 152 convolutional layers (ResNet-152).

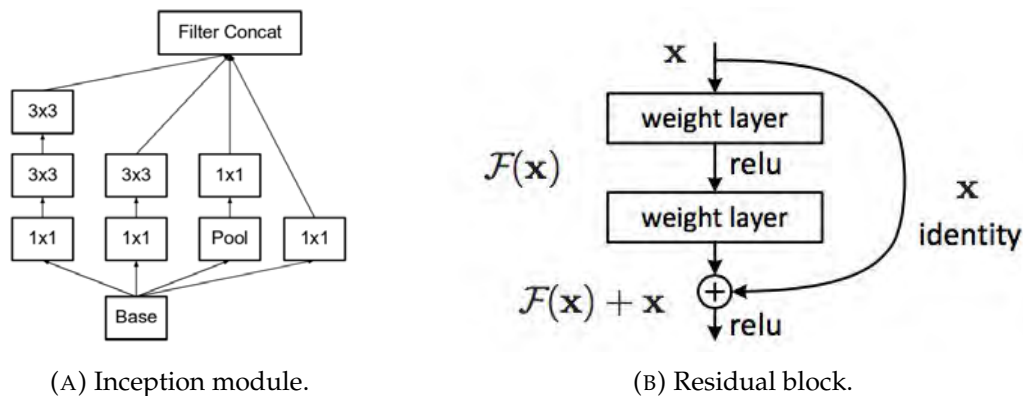


FIGURE 3.7: Advanced layers: Inception module (left) (Szegedy, Vanhoucke, Ioffe, Shlens, & Wojna, 2016) and Residual block (right) (He, Zhang, Ren, & Sun, 2016).

Redmon and Farhadi (2017) proposed a novel network of only 19 convolutional layers and 5 max-pooling layers, called: **Darknet-19**. Their network achieved comparable performance with ResNet-50, with an error of 8.8 percent instead of the 7.8 percent of ResNet-50. Although a comparable performance is achieved, they managed to reduce the required number of calculations by 27 percent (Redmon & Farhadi, 2017). This reduction is mainly achieved by replacing the fully connected layers with a 1×1 convolution. Darknet-19 achieves a speed of 200 Frames Per Second (FPS), compared with 90FPS for ResNet.

Based on Darknet-19 and the residual blocks of ResNet, Redmon and Farhadi (2018) designed **Darknet-53**. A deeper network of 53 convolutional layers, in which the residual blocks of He et al. (2016) are used to ensure the trainability of the increased network depth. They used 3×3 and 1×1 convolutions within the residual blocks, similar to ResNet. This network is much more powerful than Darknet-19, while being more efficient than ResNet-101 (Redmon & Farhadi, 2018). The error rate of Darknet-53 is comparable to ResNet-101, while doubling the FPS (Redmon & Farhadi, 2018).

The extensive growth of networks increases the need for computational power in both training and inference. Using deep learning applications on mobile devices is therefore not always possible. Although the training can be processed upfront in the cloud, the inference is sometimes required to be locally available. A large model is both memory intensive and slow in inference. Therefore, Google developed **MobileNet** v1 and v2, taking up only 16 MB, compared with 256 MB for VGG-16. Besides the memory reduction, the inference time is improved significantly, making it a well-suited model for applications on mobile devices (Sandler, Howard, Zhu, Zhmoginov, & Chen, 2018).

The above networks are summarised in Table 3.1, comparing the classification error and the model size in the number of parameters.

TABLE 3.1: Classification comparison in model performance and model size, based on: Szegedy et al. (2015), Siddharth (2017), Redmon and Farhadi (2017, 2018).

Model	Year	Place	Error	Parameters (million)
AlexNet	2012	1st	15.3%	60
GoogLeNet	2014	1st	6.6%	4
VGG-16	2014	2nd	7.3%	138
ResNet-50	2015	-	7.1%	26
ResNet-152	2015	1st	3.6%	60
Darknet-19	2017	-	8.8%	<i>unk</i>
Darknet-53	2018	-	6.2%	<i>unk</i>
MobileNet v1	2017	-	12.9%	4
MobileNet v2	2018	-	8.9%	3

3.4 Object Detection

Whereas classification assigns one label to each image, object detection can assign labels on an object level. Object detection models predict, in addition to the class, the location of an object. Therefore, each image can have multiple objects. This approach extends the problem from classification to classification and regression, increasing the number of output values from C (classes) to $h(C + 4)$, where h is the number of objects and 4 coordinates are used for each object location.

3.4.1 Sliding Window

Perhaps the most convenient and straight forward approach to move from image classification to object detection, is by the use of the *sliding window approach*. The sliding window approach consists of three steps: the cropping stage, the detection stage, and the merge stage. Figure 3.8 visualises the sequential steps. A more accurate bounding box can be generated by reducing the size of the crops, or making overlapping crops of the original image. The downside of this approach is its high inference time, as each cropped image needs to be passed through the classifier. Furthermore, Gandhi (2018) explained that large differences in object size complicate the detection process as multiple window sizes should be used.



FIGURE 3.8: Sliding window approach.

3.4.2 Region-based Convolutional Neural Network (R-CNN)

R-CNN, proposed by Girshick, Donahue, Darrell, and Malik (2014), outperformed previously developed models by far on the PASCAL VOC 2012, by increasing the mAP with more than 30 percent. They compared their proposed model with OverFeat (sliding window detector), proving R-CNN to outperform the sliding window detection in both speed and accuracy on the ILSVRC dataset (Girshick et al., 2014). They explained that the efficiency of R-CNN is established by reducing the large number of regions of the sliding window approach, to only 2,000 region proposals by use of selective search (Gandhi, 2018). The algorithm steps are summarised below.

1. Generate many candidate regions.
2. Use a greedy algorithm to merge similar regions recursively into large regions.
3. Warp each region to a squared input image.

Each proposed region is passed through a CNN to produce a 4096-dimensional feature vector, used to predict the bounding box coordinates with regression and the object class with the use of a Support Vector Machine (SVM). Although the network uses the same feature vector for the bounding box regression and object classification, 2,000 regions need to be evaluated by the CNN. Additionally, no learning is applied to the region proposal which might result in poor region proposals. Although the model is an improvement over the sliding window approach, there is no uniform model. This is, the region proposal, the CNN, the bounding box regression, and the SVM for classification are operating in isolation. The model architecture is shown in Appendix B.

To address the independent models in R-CNN, Girshick (2015) aggregates the independent feature vectors of R-CNN into a single CNN. This innovative approach removes the separate regression and SVM and predicts both tasks with the same CNN. This new approach is named: Fast R-CNN (see Appendix B). The single forward pass constructs a feature map from which regions are extracted. Each of the regions is then passed to a bounding box regressor and object classifier.

Although fast R-CNN resulted in a major speed improvement, selective search for the region proposals still results in a high computational burden. To address this, Ren, He, Girshick, and Sun (2017) proposed Faster R-CNN, which removes the selective search and incorporates the region proposal in a second CNN. This new approach has the major advantage over the previous versions, in that it is able to optimise the proposals, whereas it was static in previous versions. As a result, Faster R-CNN is almost 11 times faster compared with Fast R-CNN (Ren et al., 2017). Therefore, the model is able to run up to 5 FPS on a Titan-X GPU, making it applicable for real-time object detection.

He, Gkioxari, Dollar, and Girshick (2017) extended Fast R-CNN with an object mask prediction and called their approach: Mask R-CNN. Whereas bounding boxes are forced to predict a box region, mask enables to predict the contours of the object accurately. Using this, Mask R-CNN is able to construct a more accurate location estimation. The mask is learned in parallel with the bounding box regression and the object classification. By this, the mask adds only a small burden to the computational speed (He et al., 2017).

3.4.3 Single Shot multi-box Detector (SSD)

Single Shot multi-box Detector (SSD) combines feature extraction, class prediction, and localisation in one forward calculation, making the model applicable for real-time image processing while outperforming Faster R-CNN (Liu et al., 2015). SSD achieves a mAP of 76.9 for input size 300×300 and a mAP of 76.9 percent for input size 512×512 on the PASCAL VOC 2012 challenge (Liu et al., 2015). SSD maintains the high accuracy due to its uniform model architecture, making it possible to use the complete image for predicting each bounding box, contrary to R-CNN which uses regions to predict each bounding box. The model architecture is visualised in Appendix B.

SSD uses a trained image classification model, without the classification layers, as base network. On top of the base network, convolutional feature layers are added, decreasing in dimensionality further in the network. Each of the feature layers predicts the class and location of the objects for multiple cells. A novel feature map is introduced, where the image is separated into a grid with cells, where each cell predicts a fixed set of objects. The number of objects per cell is the number of default

anchors used. For each default anchor, the model predicts the confidence score per object class and the location of the object, relative to the cell centre.

The location is predicted by the height and width offset between the detected object and default anchor. An input image, with its corresponding 8×8 and 4×4 feature map, is shown in Figure 3.9. Different default anchors are drawn on the feature maps for illustration. A network with C classes, k default anchors, and a $m \times n$ feature map, results in $(C + 4)kmn$ predictions. Breaking the equation down $m \times n$ cells with k default anchors, results in a total of $k \times m \times n$ default anchors to predict. For each default anchor, C class predictions are made, as well as the $\Delta(x, y)$ (anchor offset) and the $\Delta(w, h)$ (weight and height offset).

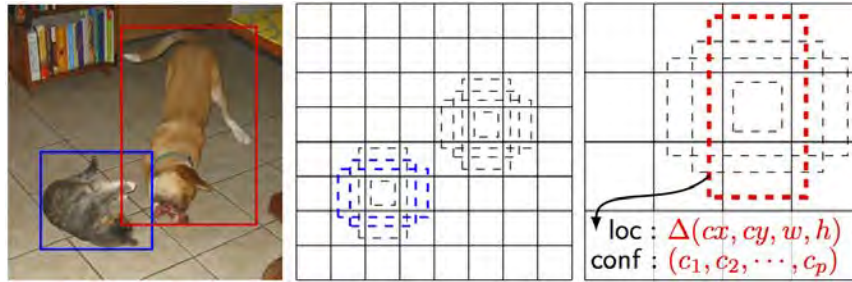


FIGURE 3.9: SSD feature maps. Original image (left), 8×8 feature map (centre), and 4×4 feature map (right) (Liu et al., 2015).

Training the network is performed with a single loss function, combining the confidence loss and location loss. Liu et al. (2015) derived the loss function from the Multi-Box objective and extended this to handle multiple object classes. To define the loss function, we use the parameters listed below.

i = default box,

j = ground truth box.

C = # classes.

$$x_{ij}^p = \begin{cases} 1, & \text{if the } i\text{-th default box matches the } j\text{-th ground truth box of class } p. \\ 0, & \text{else.} \end{cases}$$

$$\hat{c}_i^p = \frac{e^{c_i^p}}{\sum_p e^{c_i^p}}$$

Softmax loss over multiple class confidence.

More informally, the confidence of the actual class p divided by the sum of all confidences.

N = Number of images in the epoch.

l = Predicted box.

g = Ground truth box.

Before the loss can be calculated, each predicted bounding box needs to be matched to a ground truth box. Liu et al. (2015) uses the Jaccard overlap (Equation (3.25)) between the ground truth box and the predicted box and sets the threshold to 0.5, indicating that an overlap of at least 0.5 is a positive match. A negative match assigns the truth to be background (no object).

$$\text{Jaccard Overlap}(i, j) = \frac{\text{area}(i \cap j)}{\min(\text{area}(i), \text{area}(j))} \quad (3.25)$$

The total loss, defined in Equation (3.26), consists of the weighted sum over the confidence loss (L_{conf}) and the localisation loss (L_{loc}), where α is set to one by cross-validation of Liu et al. (2015). As the majority of predicted boxes (i) are considered to be negative, hard negative mining is applied where the negative predictions are sub-sampled to a ratio of 3 : 1 (negative : positive). The below formulations are derived from Liu et al. (2015).

$$L(x, c, l, g) = \frac{1}{N} (L_{conf}(x, c) + \alpha L_{loc}(x, l, g)) \quad (3.26)$$

Using the Jaccard overlap to distinguish the positive and negative boxes and by using the softmax for class confidence, the confidence loss of Equation (3.27) is formulated.

$$L_{conf}(x, c) = - \sum_{i \in pos} x_{ij}^p \log(\hat{c}_i^p) - \sum_{i \in neg} \log(\hat{c}_i^0) \quad (3.27)$$

The localisation loss makes use of the smooth L1 loss, which compares the predicted box ($l = (x_c, y_c, x_w, y_w)$) with the ground-truth box ($g = (x_c, y_c, x_w, y_w)$). The Smooth L1 loss makes the regression less sensitive to outliers (Liu et al., 2015). The location loss is formulated in Equation (3.28). To account for the difference in bounding box sizes, the predicted box size is divided by the width and height of the actual width and height of the ground truth box. This approach transforms the actual deviation into a percentage deviation and makes each bounding box contribute equally to the loss function, independently of its size.

$$L_{loc}(x, l, g) = \sum_{i \in pos} \sum_{m \in \{c_x, x_y, w_y, y\}} x_{ij}^k L_1(l_i^m, \hat{g}_i^m) \quad (3.28)$$

$$L_1(l_i^m, \hat{g}_i^m) = |x_c - \hat{x}_c| + |y_c - \hat{y}_c| + |x_w - \hat{x}_w| + |y_w - \hat{y}_w| \quad (3.29)$$

Deconvolutional Single Shot Detector (DSSD)

To address the lack of detection capabilities of small objects in SSD, Fu, Liu, Ranga, Tyagi, and Berg (2017) developed Deconvolutional Single Shot Detector (DSSD). They made two contributions. Firstly they introduced the use of ResNet-101 as the base network, instead of VGG-16 proposed by the original authors. Secondly, they introduced a deconvolutional module which increases the resolution of the feature maps. The latter significantly increased the performance for the small object by providing more context to the classifier. The increased performance comes at a relatively small speed reduction (Fu et al., 2017). The model architecture is displayed in Appendix B.

Feature Fusion Single Shot multi-box Detector (FSSD)

Similar to DSSD, Feature Fusion Single Shot multi-box Detector (FSSD) extends the conventional SSD to address the lack of performance on small objects. Li and Zhou (2017) applied the Feature Fusion module on the SSD layers, to transfer contextual

information between layers. The transfer of contextual information is of special interest for smaller objects, as small objects rely more on the surrounding context (Li & Zhou, 2017). They explain that feature maps, responsible for small object detection, are not large enough to incorporate contextual information and therefore benefit largely from the transfer between layers. Furthermore, Li and Zhou (2017) explain that the use of contextual information improves the accuracy, by reducing the detection of multi-parts of an object such as a dog and the head of a dog. They achieve a significantly higher mAP, compared with the conventional SSD: 82.7 on 512×512 compared with 81.2 on 300×300 . Their model architecture is displayed in Appendix B.

Receptive Field Block Single Shot multi-box Detector (RFB-SSD)

Liu, Huang, and Wang (2018) explained that the performance gain of DSSD is mostly obtained from the use of ResNet-101, making it relatively slow in inference. To create a powerful and yet efficient network, Liu et al. (2018) constructed a Receptive Field block, inspired by the Receptive Field of the human visual cortex. Using the Receptive Field block, features of multiple convolution filters are combined and therefore also referred to as multi-branch convolution layer (Liu et al., 2018). Liu et al. (2018) showed that a more light-weighted network such as MobileNet can be used while maintaining a comparable detection performance. The receptive field block is shown in Figure 3.10, the model architecture is displayed in Appendix B.

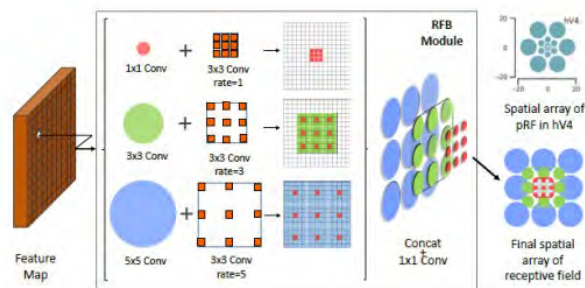


FIGURE 3.10: RFB block with three convolution branches (Liu, Huang, & Wang, 2018).

3.4.4 You Only Look Once (YOLO)

You Only Look Once (YOLO), proposed by Redmon, Divvala, Girshick, and Farhadi (2016) outperformed R-CNN in terms of detection speed, while maintaining a comparable accuracy performance. They achieved this by combining the bounding box regression and class estimation in the same network as the feature extraction. The power of this method is the unified architecture, making it possible to optimise the model end-to-end (Redmon et al., 2016). Besides the fact that YOLO runs in 45 frames per second, and is, therefore, able to run in real-time. It makes fewer mistakes in the background as it uses the full image for the class prediction, contrary to fast R-CNN which only incorporates the local surrounding for the class prediction. Additionally, it generalises better to other domains compared with previous methods (Redmon et al., 2016). Redmon et al. (2016) explained the downside of YOLO

being less accurate compared to systems not running in real-time and having troubles predicting accurate bounding boxes, as well as predicting small objects.

Techniques, similar to SSD, are used to optimise the architecture. An $S \times S$ grid (Figure 3.11) is used with B default bounding boxes per grid cell. Each default bounding box makes a prediction for the x, y coordinate of the object, and the box width and height (Redmon et al., 2016). Furthermore, each grid cell has C conditional class probabilities, making a vector of $S \times S \times (5B + c)$ predicted values in the output (Redmon et al., 2016). The network uses a relatively small input size of 416×416 images and uses a sequence of 24 convolutional layers, followed by two fully connected layers, which map the predictions to the output tensor. When the default anchors represent the object sizes in the dataset, the need for bounding box regression training is reduced. The full network architecture is visualised in Appendix B.



FIGURE 3.11: YOLO default anchors on grid size 11×8 with $B = 2$.

Redmon and Farhadi (2017) presented version two of YOLO, approximately a year after the first introduction. The improved version outperforms Faster R-CNN, having 78.6 mAP on 40 FPS instead of 76.4 mAP on 5 FPS for PASCAL VOC 2012. A variety of optimisation techniques were implemented: introducing the Darknet-19 backbone, applying batch normalisation, removing all fully connected layers and adding more default anchors. They added k -means clustering to define the optimal default anchor sizes. To overcome the prediction error for small objects in version one, they added a second feature map from an earlier layer and added this to the final prediction.

To further improve the performance on small objects, Redmon and Farhadi (2018) introduced YOLO v3, pushing the performance from 44.0 AP_{50} (version two) to 61.1 AP_{50} on the COCO challenge. To achieve this, they changed the Darknet-19 backbone to Darknet-53. Although the performance increased significantly, the model dropped in inference speed. Furthermore, they changed the softmax layer, for class prediction, to independent logistic classifiers which improve the training process for overlapping objects (Redmon & Farhadi, 2018). The last big change to the network is a third prediction scale and adding upsampling to further improve the performance on small objects, while decreasing the performance on medium and large size objects (Redmon & Farhadi, 2018).

YOLO uses a single loss function to update the weights across the full network. The final loss function, developed by Redmon and Farhadi (2018), consists of classification loss, localisation loss, and confidence loss. The total loss function is the sum of the independent loss functions, having weight α_{coord} , α_{conf} , and α_{noobj} . $\mathbb{1}_i^{obj} = 1$ if an object appears in grid cell i and $\mathbb{1}_{ij}^{obj} = 1$ if default bounding box j in cell i is

responsible for detecting the object. $\mathbb{1}_{ij}^{noobj} = 1 - \mathbb{1}_{ij}^{obj}$. C defines the confidence score, where $p(c)$ defines the classification loss.

The classification loss is displayed in Equation (3.30) and calculates for each grid cell $i \in \{S\}^2$ the squared sum of differences between the actual class and the predicted class probability. The classification loss is only taken into account if $\mathbb{1}_{ij}^{obj} = 1$. This is, the classification loss is only taken into account if default bounding box j in cell i is responsible for detecting the object (Redmon & Farhadi, 2018). The localisation loss, displayed in Equation (3.31), takes for each grid cell and each bounding box, the squared difference in box centre and box size. The box width and height is reduced by the square root to make it more robust to outliers. When no object is present, the loss is taken over the confidence score C and no localisation or classification loss is used (Equation (3.32)).

$$loss_{classification} = \sum_{i=0}^{S^2} \mathbb{1}_i^{obj} \sum_{c \in C} (p_i(c) - \hat{p}_i(c))^2 \quad (3.30)$$

$$loss_{localisation} = \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{obj} \left((x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 + (\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 + (c_i - \hat{c}_i)^2 \right) \quad (3.31)$$

$$loss_{noobj} = \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{noobj} (c_i - \hat{c}_i)^2 \quad (3.32)$$

3.5 Evaluation

One of the most widely known evaluation measures for classification problems is *accuracy*. The accuracy for C classes can be measured by Equation (3.33), where the corresponding variables are shown in the confusion matrix (3.34). The confusion matrix shows, with p_{ij} , the number of objects or images of class i which are predicted as class j . Therefore, the number of correctly predicted objects, or images, of class c is p_{cc}

$$Accuracy = \frac{\sum_{i=1}^C p_{ii}}{\sum_{i=1}^C \sum_{j=1}^C p_{ij}} \quad (3.33)$$

$$\begin{bmatrix} p_{11} & p_{12} & \dots & p_{1C} \\ p_{21} & p_{22} & \dots & p_{2C} \\ \vdots & \vdots & \ddots & \vdots \\ p_{C1} & p_{C2} & \dots & p_{CC} \end{bmatrix} \quad (3.34)$$

Due to the simplicity of both calculation and interpretation, the accuracy measure is widely applied for many machine learning evaluations. However, Michelucci (2018) explained the downside when imbalanced class distributions are present, having ambiguous accuracy values for different distributions. Class imbalanced arises if n_i (the number of samples in class i) for $i \in [1, 2, \dots, C]$ strongly deviates. The drawback can most easily be explained with two classes. If $n_1 = 90$ and $n_2 = 10$, the trained

model can achieve a 90 percent accuracy by predicting the first class for all 100 instances. An accuracy of 90 percent will not be accurate, whereas a class distribution of $n_1 = n_2 = 50$ with accuracy 90 percent will be a good accuracy.

To overcome class unbalance influence, both Precision and Recall are introduced. $Precision_c$ defines the fraction that the model predicted class c correctly out of all predicted class c . $Recall_c$ defines how often images or objects of class c are classified correctly as class c . The Precision and Recall multi-class calculation, inspired by Sokolova and Lapalme (2009), is given in Equation (3.35) and Equation (3.36) respectively.

$$Precision_c = \frac{p_{cc}}{\sum_i^C p_{ic}} \quad (3.35)$$

$$Recall_c = \frac{p_{cc}}{\sum_j^C p_{cj}} \quad (3.36)$$

Classification problems, such as the Caltech 256 dataset (Griffin, Holub, & Perona, 2007), requires a forced choice to predict one of the classes for each instance. Object detectors like R-CNN, YOLO, and SSD not only require evaluation of the classification, but also require evaluation of the bounding box estimation. They require to estimate the class and location for all objects in the given instance/image (Everingham, Van Gool, Williams, Winn, & Zisserman, 2009). A common approach to evaluate the generated bounding boxes is by use of the Intersection over Union (IoU), which compares the predicted bounding box with the ground truth bounding box. Figure 3.12 displays a partly matched bounding box, where Equation (3.37) can be used to calculate the IoU with B_p the bounding box of the prediction and B_{gt} the bounding box of the ground truth.

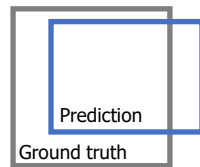


FIGURE 3.12: IoU visualisation.

$$IoU = \frac{area(B_p \cap B_{gt})}{area(B_p \cup B_{gt})} \quad (3.37)$$

The Mean Absolute Precision (mAP) combines the Precision, Recall, and IoU into one unified measure. Prior to the use of the mAP, the Area Under the Curve (AUC) has been widely used, to measure the model quality in a performance/recall trade-off graph (Everingham et al., 2009). Everingham et al. (2009) explained that the two methods are generally in agreement. However, the mAP is considered under different IoU values, resulting in the measure: mAP_α with α the IoU threshold as shown in Equation (3.37). $Precision_c(Recall_{ci})$ is defined as the precision at recall value i for class c . The interpolated maximum of the precision for class c , to the right, is then defined by: $\max(Precision_c(Recall_{cj} : j \in [i, i + 0.1, \dots, 1]))$. The final mAP is defined in Equation (3.38). When averaging the mAP over multiple IoU thresholds, more accurate localisation tends to get rewarded.

$$mAP_\alpha = \frac{1}{C} \sum_{c=1}^C \frac{1}{11} \sum_{i \in [0, 0.1, \dots, 1]} \max(\text{Precision}_c(\text{Recall}_{c_j} : j \in [i, i + 0.1, \dots, 1])) \quad (3.38)$$

3.6 Training Neural Networks

The goal for Neural network training is to minimise the loss function by use of an iterative approach. The Forward Propagation (FP) and Backward Propagation (BP) is used to optimise the weights. The total dataset with N images is split into B equal batches of N_B images. Each batch is passed through the network by use of FP and the output is stored for each individual image. Secondly, BP is used to adapt the weights, based on the error over the batch of images. One forward and backward pass is called an **iteration**. Within each iteration, the BP updates the weights based on the optimisation algorithm, defined in Section 3.2.4. The intensity of the weight update is depending on the Learning rate (α). The batch size, as well as the learning rate, are largely influencing the training process. The effect of these two parameters is explained in Section 5.2. The model is optimised during several epochs, where one **epoch** processes N_B batches in sequence. A single epoch makes the complete dataset pass through the FP and BP once.

3.6.1 Regularisation

As previously explained in Section 3.2, increasing the model complexity enables the network to learn more complex structures. Without regularisation, the network could potentially learn the training data correctly, without being able to generalise to unseen examples (overfitting). Table 3.2 shows four possible performance options after training. The model in the top left fits the data well and generalises good to new data, the model in the top right has trouble fitting the data and does not generalise. The model in the bottom left fits the data well but does not generalise to unseen data (overfitting). The model in the bottom right does not fit the train data (high bias) and is, therefore, also unable to accurately predict the unseen data.

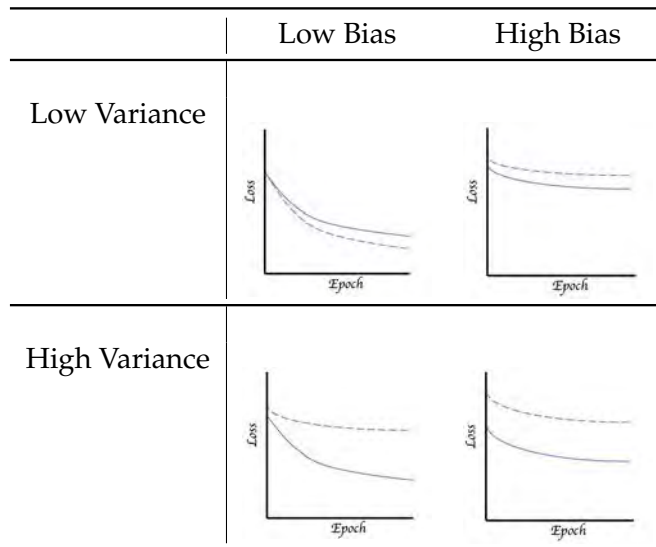
Overfitting can be reduced using Regularisation. A common approach for regularisation is dropout, which removes each node and corresponding connections with probability p . The central idea behind dropout is to reduce the dependency on each node and therefore, force the network to focus on more general features (Witten, Eibe, Hall, & Pal, 2017, p. 434).

Two other approaches are L1 regularisation and L2 regularisation. Both methods perform a form of feature selection by penalising large weights in the network. L1 regularisation, also called Lasso, performs feature selection by assigning a zero weight to irrelevant features. L2 regularisation, also called Ridge, makes irrelevant weights smaller but does not make them zero in a way Lasso does. This makes L2 regularisation not a feature selector. The regularisation can be used to extend the original loss function. Equation (3.39) shows L1 regularisation and Equation (3.40) shows L2 regularisation.

$$L = L + \alpha \sum_i |w_i| \quad (3.39)$$

$$L = L + \alpha \sum_i w_i^2 \quad (3.40)$$

TABLE 3.2: Bias variance trade-off with dotted line the validation loss and solid line the training loss.



To reduce overfitting in particular, early stopping can be used. This approach cuts-off training as soon as the validation loss does not improve for a fixed set of epochs, or the validation loss starts to increase (Witten et al., 2017, p. 261). This approach is of special interest when the number of samples is low, making overfitting a potential danger. By early stopping, the training is terminated before the overfitting starts.

3.6.2 Augmentation

Image augmentation constructs different images, based on the images in the original dataset. Using image augmentation, a more diverse dataset arises and is, therefore, especially important for small datasets. Use of augmentation reduces overfitting and makes the model more robust to unseen images. Some frequently used augmentation functions are listed below:

- **Cropping and Padding:** A random percentage is used to either crop ($\alpha < 1$) or pad ($\alpha > 1$) the image. Padding increases the image dimension by adding pixels to the side of the image with the pixel mean.
- **Horizontal flipping:** An image is flipped horizontally with probability $p = 0.5$.
- **Contrast Normalisation:** Changes the contrast in an image.
- **Multiply:** Multiply is used to create a darker ($\alpha < 1$) or brighter ($\alpha > 1$) image.
- **Rotate:** Random rotation is used to make the model more robust to different camera angles.
- **Gaussian blur:** Gaussian blur distorts the image slightly, to make the model more robust against quality loss in images. With this, a filter is of size f is applied to the image where the filter is normally distributed in the width and height of the filter.

3.6.3 Transfer Learning

Each deep-learning model tries to extract features, starting from low-level features at the first layers, till high level features at the last layers. As deep-learning models require an extensive number of training images to learn the full range of features, it cannot be applied when the number of available training instances is low. Using transfer-learning, a trained model on a large number of images for task A is reused for a different task (B), which has a low number of training instances. The central idea behind transfer learning is that low-level features are relatively similar across networks, trained for different tasks. Low-level features are typically stripes, where the last layers combine the lower level features into a prediction (Martin, 2019).

Transfer learning can be applied in multiple ways. Pu, Apel, Szmigiel, and Chen (2019) removed the last X layers from the networks and trained a Support Vector Machine (SVM) on the output vector, to transform the features into the desired class predictions. This approach requires a pre-trained model, where the original task is relatively similar to the new objective.

Another approach takes the first layers (base network) from the trained model on task A , and initialises another model for task B with the trained weights. Using this technique, the second model is already able to extract low-level features without any training. Fine-tuning is then used to train the last layers, to transform the low-level features into a prediction. Since the number of classes most frequently deviate between the two tasks, the number of weights in the last layers deviates. Therefore, the weights of the last layers cannot be transferred and are mostly initialised at random. The transfer process is visualised in Figure 3.13. During the fine-tuning, the base model can be frozen (no learning is applied on the weights) and only the extra layers, as well as the location and confidence, are trained. Another approach is to train the complete network.

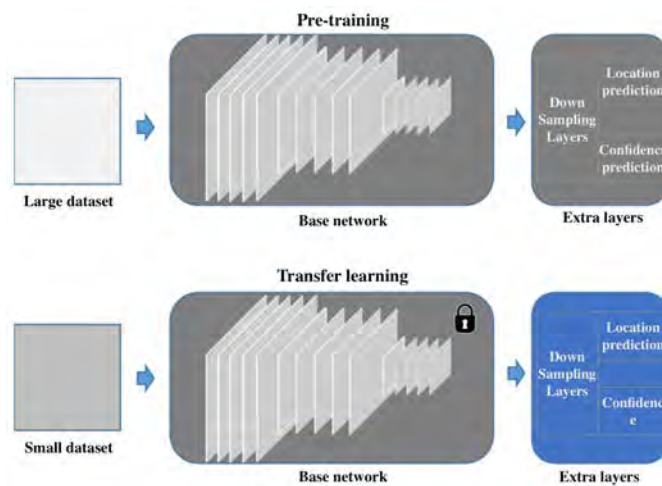


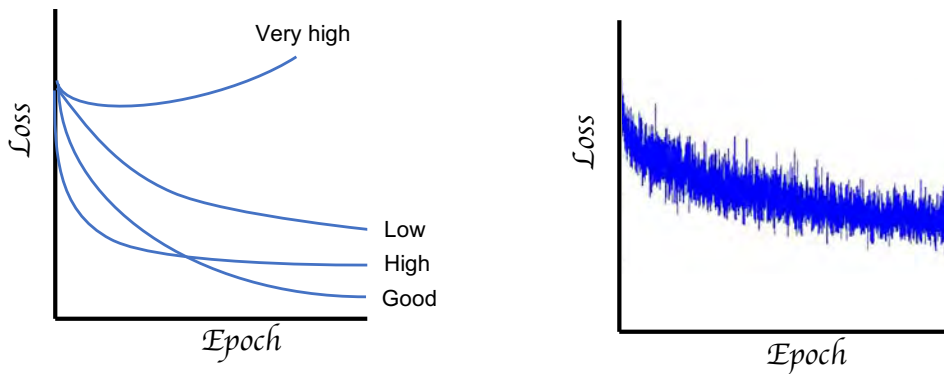
FIGURE 3.13: Transfer learning, based on (Pu, Apel, Szmigiel, & Chen, 2019).

3.6.4 Hyperparameter Tuning

Apart from the weight parameters within the network, a large variety of hyperparameters is present. The weight parameters itself are updated using the BP algorithm, specified in Section 3.2.4. Hyperparameters are set upfront and require optimisation by an iterative search process. Three commonly tuned parameters during CNN training are listed below. Although many more hyperparameters exist, we will not dive into all of them. A few other examples are augmentation types and intensity, regularisation intensity, and learning rate reduction.

- **Learning rate:** To which extent should the weights be updated after each iteration.
- **Batch size:** The number of images provided in each iteration.
- **Optimiser parameters:** The optimisation algorithm used during BP, requires the specification of different parameters (e.g. momentum in SGD and regularisation intensity for Adam.).

Figure 3.14a shows the effect of different learning rates on the loss value. A high learning rate ensures fast convergence, however, the learning potentially ends up in a local minimum. A low learning rate gives too few learning abilities and the model might under-fit or training takes too long. A low batch size, Figure 3.14b, gives fluctuating loss values. This unstable loss value could reduce the training ability of the network. On the other hand, a high batch size results in a more smooth training process but reduces the training speed.



(A) Learning rate influence on loss per epoch, (B) Batch size influence on loss per iteration, based on (Karpathy, 2017).

FIGURE 3.14: The effect of the hyperparameters on the training process: learning rate (left) and batch size (right).

Chapter 4

Data

This section describes the data used throughout this research. Three separate image data sources are used, further referenced to as *Logistics Dossiers*, *Damage Web*, and *Light Street video stream*. The first dataset is provided by Pon Logistics (Section 4.1), whereas the *Damage Web* dataset is collected from the internet (Section 4.2), and the Light Street video stream is captured using a camera setup at the Light Street of Pon Logistics (Section 4.3). Section 4.4 describes the master data of all incoming vehicles, used as ground truth for the damages reported in the Light Street.

4.1 Damage Dossiers

Historical vehicle damages have been made available by Pon Logistics. This data consists of 2,499 dossiers, where each dossier contains multiple¹ images of the damaged vehicle and/or the damaged vehicle during the repair process. The dataset contains a wide range of viewpoints and zoom levels. Furthermore, undamaged sides of the vehicle are included as well. A total of 19,907 images is present, where the irrelevant images such as vehicles during repair, vehicles without damage, images without vehicles, or poor quality images² are omitted. The relevant images are manually selected from the dossiers, yielding a total of 3,513 images, being only 17.65 percent of the provided dataset. An excerpt from the dataset is included in Appendix D.1.

Image dimensions fluctuate in the pixel range of $w \in [640, 4800]$ and $h \in [680, 4100]$, having an average aspect ratio of approximately 2.0 : 1.0³. The dimension diversity is visualised in Figure 4.1a. Images are manually annotated according to Section 4.5.2, yielding a total of 7,900 objects, where the object class frequency is visualised in Figure 4.1b. A large class imbalance is present, which is further discussed in Section 4.5.2.

¹Average: 8, min: 0, and max: 89.

²Too blurred images, taken against the light, or when the damage is not visible for annotation.

³Aspect ratio: (weight : height).

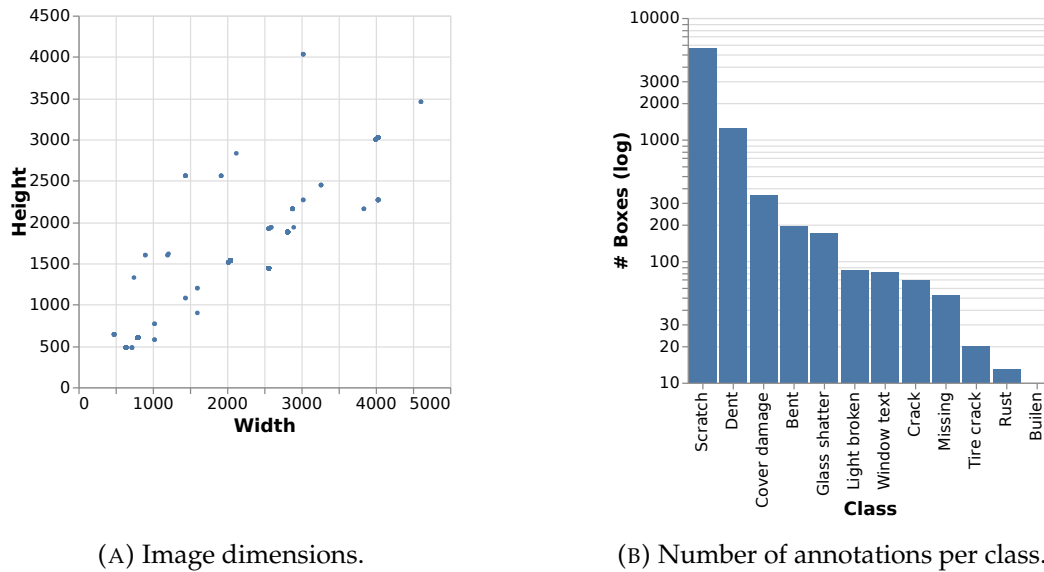


FIGURE 4.1: Data statistics of the Damage Dossiers dataset.

4.2 Damage Web

To enrich the data provided by Pon Logistics, a dataset is constructed by use of web scraping on Google image search. Besides the enlargement of the total image dataset, this external data is used to compare the performance of the model trained on internal data with a model trained on external data. The external data contains images of higher resolution and vehicles are on average less clean, making the model potentially more robust. Lastly, although Li et al. (2018) and De Deijn (2018) used a slightly different approach, it enables some performance comparison with previous research.

By use of web-scraping in Python, approximately 2,500 images have been extracted from Google image search. The used search words are listed in Appendix C, where the maximum number of images (limit) is set to prevent from downloading too many irrelevant images. The limit differs per query based on manually inspecting the threshold of relevance. Any irrelevant⁴ images are manually removed during the annotation process. Duplicate removal and removal of irrelevant images yields a dataset of 1,338 images. Compared with the *Damage Dossiers*, stronger fluctuations are present in the image dimensions: $w \in [150, 9216]$ and $h \in [130, 4800]$, having an average aspect ratio of approximately: 2.2 : 1.0. An excerpt from the dataset is included in Appendix D.2. Images are manually annotated according to Section 4.5.2, yielding a total of 3,392 objects. The image dimensions are visualised in Figure 4.2a, where Figure 4.2b shows the object class frequency.

⁴Irrelevant images are either of bad quality, containing unclear damages, or no vehicle.

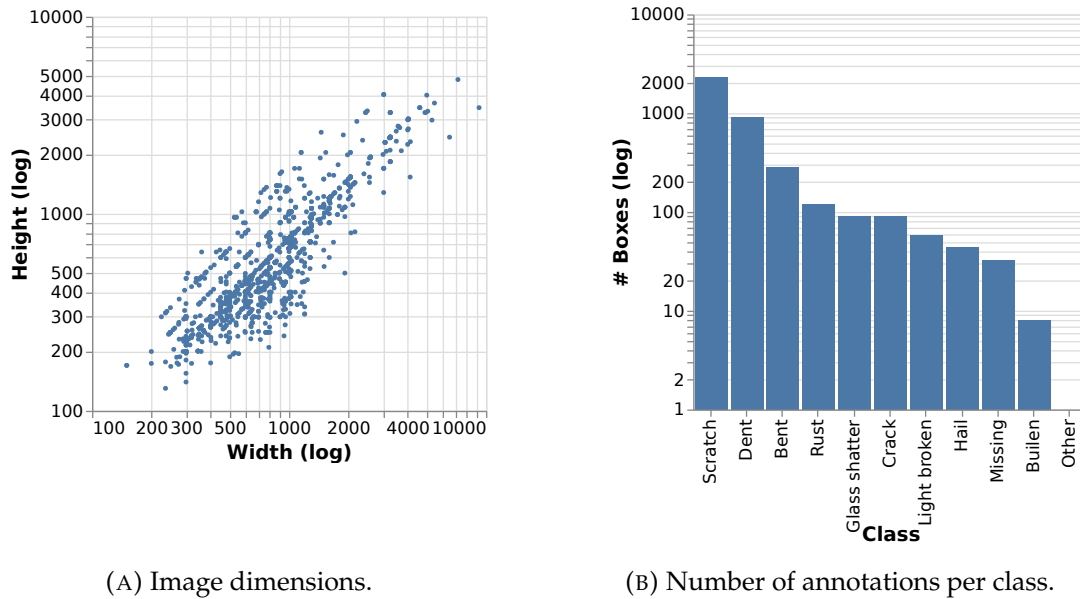


FIGURE 4.2: Data statistics of the Damage Web dataset.

4.3 Light Street

Each vehicle, which arrives by truck, is checked on damage in the Light Street. No vehicle images are captured in the Light Street and therefore a camera system is put in place to capture the incoming vehicles. For testing purpose, it has been decided to start with four cameras to prove business value. Any expansions in the number of cameras can be made, if the initial approach shows its ability to add value to the business process. Increasing the number of cameras could potentially increase the number of areas to inspect.

As explained in Section 2.2, some areas are more frequently damaged and some areas are more difficult to inspect by a human. As only four cameras are placed in the testing phase, a selection is made on the areas to cover with the cameras. Priority has been given to capture the vehicle from each side and to include the rim, tire, and roof. Each camera should be placed as close as possible to the vehicle to capture a less distorted image. Furthermore, each vehicle is of different size in height, width and length and the cameras should therefore not be placed too close to the vehicle.

Based on all requirements, it seemed best to install the cameras according to the setup of Figure 4.3. The side cameras (camera one and camera three) are placed high enough to capture the roof for each passenger car. Capturing a minivan roof is not ensured in all cases (e.g., Volkswagen Crafter and Volkswagen Transporter), as placing the camera high enough for minivans endangers the quality of capturing the lower parts of the vehicle. Therefore, it has been decided to place these cameras at a height of 265 cm from the floor. The front camera (camera four) and rear camera (camera two) are placed at a lower height of 200 cm as the roof is already covered by the side cameras.

One of the alternative camera locations would be to point each camera at the corner of the vehicle, to ensure that all areas are accurately visible. However, the downside is that rims are less visible under a strong side angle. As rim damage is relatively difficult to inspect, it has been decided that priority is put on inspecting the left and right side of the vehicle. Furthermore, most vehicle damages are at

the side, front, rear, bonnet, and roof. Contrary to vehicles on the road, only a small proportion has damage at the vehicle corners. This is due to the majority of damages being transport-related, making the described areas most vulnerable. Therefore, the setup of Figure 4.3 has been selected as implemented setup.

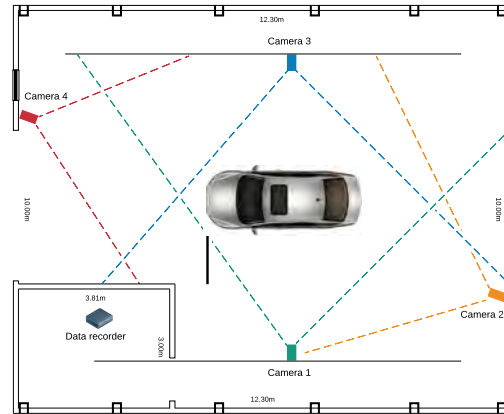


FIGURE 4.3: Camera setup.

In order to remove the need for manual interference, it has been decided to capture a video stream instead of pictures. This ensures that present employees do not need to press the capture button, when vehicles arrive. Furthermore, it ensures that the system can run autonomously, making it possible to implement the system at the train unloading area where speed is crucial. Furthermore, using a video stream ensures more frames and gives the flexibility to select only the frames where no employees are blocking the view on the vehicle. To reduce the generated data flow from the cameras, the FPS has been reduced to the lowest supported configuration: 2 FPS. The used camera system is selected based on the following requirements:

1. *Image quality*: The large distance between the camera and vehicle, as well as the relatively small damages, requires a high image resolution.
2. *Light adjustable*: The Light Street contains a large illumination and therefore, the cameras should be manually configurable to adjust for light conditions.
3. *Cloud integration*: A (near) real-time connection should be possible to enable cloud-based damage detection.
4. *Internal storage*: No data should be lost if internet connection (temporary) fails.
5. *Power over Ethernet support*: No sockets are available at the camera locations and hence the camera should be powered by Power over Ethernet (PoE).
6. *Wide angle*: The side cameras are relatively close to the vehicle and should therefore have at least a 70 degree angle to capture the largest passenger vehicles in full.

After ranking different providers, both Reolink and Swann seem to meet the above requirements. Swann seems to have the best cloud integration, however, shipment to the Netherlands was not well supported and therefore Reolink has been selected.

5MP Super HD PoE IP cameras⁵ are used for the research as Ultra HD was not yet available. The cameras ensure a stable resolution of 2560x1920 at 5MP, well above average resolution of the provided damage images. Figure 4.4 shows a capture of all four video streams, according to the setup of Figure 4.3.

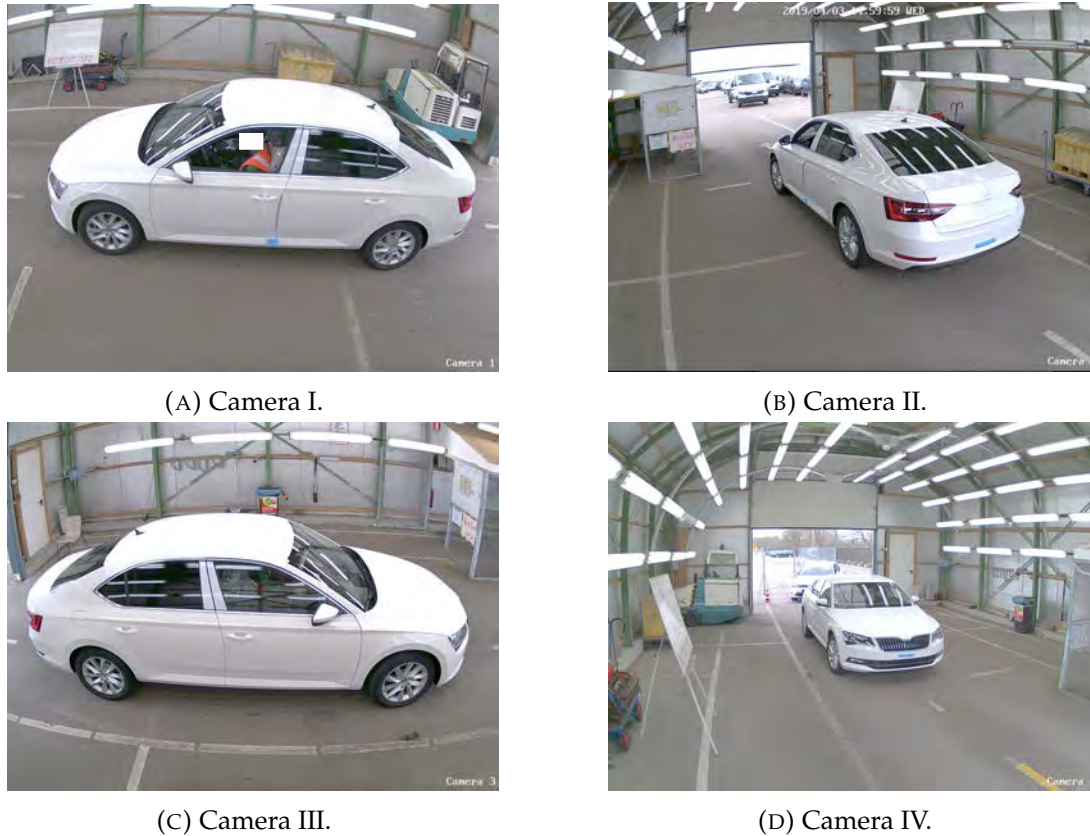


FIGURE 4.4: Light Street camera view.

4.4 Pon Logistics Master Data

Each vehicle, entering the Light Street, is formally checked in by a scan event. The scan event, together with general vehicle information, is stored in a Structured Query Language (SQL) database. Each damaged vehicle is registered in a second database table with information regarding the damage. These two datasets are used to search effectively in the captured Light Street video stream to compare the model performance with the output of the employees in the Light Street. A vehicle not being present in the second table indicates that the vehicle was not damaged.

The vehicle will appear in the damage table if the damage is detected later in the process. This can either indicate that the vehicle is damaged later in the process, or the damage was present but not detected in the Light Street. The below features are a subset of the obtained join between the required tables. Only vehicles which have detected damage somewhere in the logistics process are included. Furthermore, we excluded all vehicles which are not transported by truck and therefore not routed through the Light Street. The following fields are present in the final table:

⁵Product code: 5MP RLK8-410B4.

- *Description*: Unique id of vehicle.
- *Dstamp*: Scan event datetime.
- *To Loc id*: Destination of the vehicle after the scan event is performed.
- *Brand*: The brand of the vehicle.
- *Model*: The model of the vehicle.
- *id*: Unique id of vehicle.
- *Required repair*: Some damages are within the margins and a 'N' will indicate no repair required. 'Y' will be repair required.

Figure 4.5 shows the number of registered damages per week from 2018-12-31 till 2019-07-08. The damages are grouped into four categories, depending on the location of detection: Customer, Inbound, Logistics process (between inbound and outbound), and Other. The diversity of vehicles is limited to the Volkswagen group, incorporating: Volkswagen, Seat, Skoda, Audi, and Porsche. The fraction of damages, caused by transportation is approximately fifty percent.

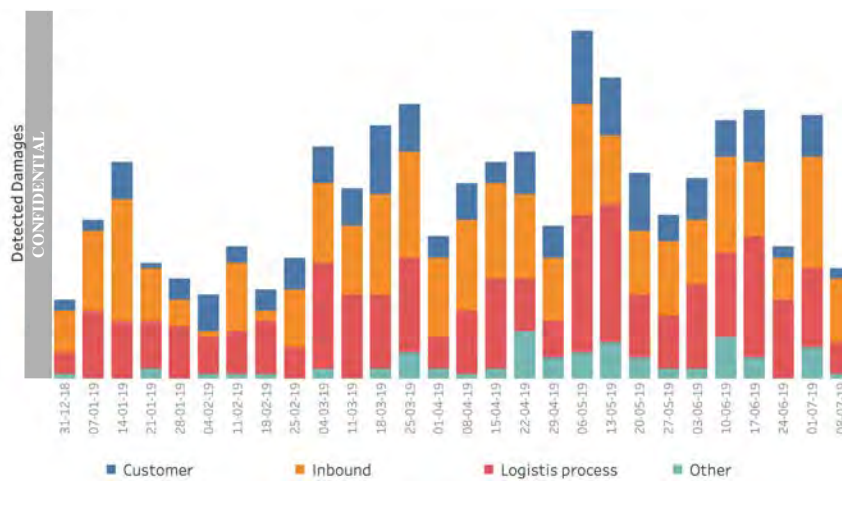


FIGURE 4.5: Number of detected damages per detected location.

4.5 Preprocessing

To make the data suitable for deep-learning models, several preprocessing steps are applied in sequence. Figure 4.6 shows the five successive steps. Restructuring transforms all image extensions to *.jpg* (for convenience), renames images, and structuring the files. Duplicates are removed by comparing the average image hash, inspired by the approach of Zhang, Fu, and Qiu (2013). Contrary to Zhang et al. (2013), only exact duplicates should be removed as most images are relatively similar but contain a different damage or vehicle. Therefore an average hash distance of zero is used, to only exclude exact duplicates. The duplicate removal is followed by a manual cleaning process, where irrelevant images are removed. The fourth step involves annotating the damages, where polygons are used to avoid excluding deep learning models based on the annotation type. Lastly, the annotations are formatted into PASCAL VOC 2012 format and divided into a train and validation set.

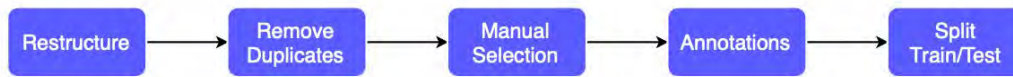


FIGURE 4.6: preprocessing steps.

4.5.1 Train and Validation Split

A reliable estimate of the model performance is required to evaluate the applicability of the model, before implemented at Pon Logistics. To create a reliable estimation, the data is divided into a train set (used for training) and a validation set (used for validation). The train fraction is set to 80 percent, slightly lower than the 90 percent proposed by Flach (2012, p. 10). This value is set slightly lower as no cross-validation is used due to the computationally expensive algorithms. Furthermore, the damages are diverse and noisy and therefore a larger validation set will create a more reliable estimation of the performance.

The *Damage Web* dataset is split randomly between the train and validation set with probability 80 percent of ending up in the train set. Applying the same random image split to the *Damage Dossiers* might leak information from the train set into the validation set, as each dossier can contain multiple images. Therefore, the *Damage Dossiers* are split randomly on a dossier level with probability 80 percent of ending up in the train set. An assigned dossier to either the train or the validation set will assign all images of the corresponding dossier to the train or validation set.

4.5.2 Annotation Process

Manual annotating all images has been done using the tooling: Supervisely⁶. Each image has been annotated using polygons to support object detection, instead of annotating the complete image for classification. This approach has been chosen as De Deijn (2018) explained that classification between damaged and undamaged can be complicated due to small object sizes, as the majority of the vehicle is mostly undamaged. Besides this, of the requirements from the Light Street employees is to have an estimation of the damage location, as a generic classification will require the employees to search for the damage. The need for detection might not have been necessary for Patil et al. (2017) and De Deijn (2018), as the damage sizes are relatively large and therefore easier to detect when the model predicts damage. For Pon Logistics, small damages are present and therefore classification will not be suited. Both the previous arguments, as well as the promising results from Li et al. (2018), to apply damage localisation, made us select object detection as the used technique for this research.

Each dataset is manually annotated in a detailed manner, to include all small damages which are of major interest for Pon Logistics. Pon Logistics uses multiple classes to define the type of damage. Builen, Dent, Hail, and Scratch are inspired by the internal damage evaluation system. From interviews with employees of the Light Street, it turned out that cover damage frequently turned out to be vehicle damage. Therefore, having cover damage should alarm the employees as much as

⁶See: <https://supervise.ly>

damage itself. To capture this, the class: Cover damage, has been added. Overlapping damage types are frequently occurring, therefore a dent can contain a scratch and so on. Both Patil et al. (2017) and De Deijn (2018) showed that classification suffers from large class interference which can be resolved as multiple damages and damage types can be annotated in the same image. Therefore, a scratch can be annotated within a dent, potentially improving the learning process as less class interference should be present.

A total of thirteen damage types is used, to compare the model performance between different types. This level of detail is selected as it gives the ability to compare model performance across different damage types. A detailed class explanation is presented in Appendix E.1. The following object classes are used:

- Bent
- Crack
- Light broken
- Tire Crack
- Builen
- Dent
- Missing
- Window text
- Cover damage
- Glass shatter
- Rust
- Scratch
- Hail
- Scratch

Bounding box dimensions differ within and across classes. Especially Scratch, Dent, and Cover damage have a diverse range of bounding box sizes. Crack, Tire crack, Builen, Rust, and Window text are relatively small. Hail damage is mostly large, as it is frequently affecting the complete vehicle. Therefore, object size and class are related. Appendix E.2 shows scatter plots of the bounding box dimensions for all thirteen image classes. Bounding boxes are normalised by the dimension of the image for unbiased comparison.



FIGURE 4.7: Example annotations.

Chapter 5

Methodology

In this subsection, we first give a detailed description of the research design. Secondly, we focus on the used models and configurations. Finally, we discuss the implemented techniques.

5.1 Research Design

Our research consists of five consecutive steps, which are summarised below. We firstly optimise a subset of hyperparameters (1), regarding the input images, augmentation, and transfer learning, respectively. The input preprocessing approach is outlined in Section 5.1.1, whereas the augmentation and transfer learning are outlined in Section 5.2.2 and Section 5.2.3, respectively.

The initially optimised parameters are used to tune the different deep learning models on the *Damage Web* dataset (2). We firstly optimise the Batch Size (BS) and Learning Rate (LR), according to Section 5.2.1. Based on the optimised LR and BS, we optimise the default anchors in combination with the pre-trained weights, during transfer learning. Lastly, the best parameter settings are projected across all implemented deep learning models and backbones. The best models and configurations of (2) are used to train a model on the *Damage Dossiers* dataset (3). A comparison is drawn in (4) to evaluate the performance of the model trained on the *Damage Web* dataset, the *Damage Dossiers* dataset, and the combined dataset. Lastly, the models are evaluated on the data from the Light Street and compared with human performance (5). The following research steps are conducted:

1. **Initial Parameter Tuning**
 - (a) **Input Preprocessing:** Input normalisation and Image resize method.
 - (b) **Initial hyper Parameter Optimisation:** Transfer learning and augmentation.
2. **Damage Web performance optimisation**
 - (a) **Hyperparameter tuning:** Tuning hyperparameters based on YOLOv3 with Darknet-53.
 - (b) **Model and backbone tuning:** Comparing different models and backbones.
3. **Damage Dossiers performance optimisation**
4. **Cross-comparison:** Between models and datasets.
5. **Performance Evaluation:** Comparing human performance with the best performing model and evaluating the performance in the Light Street.

Due to the relatively low number of bounding boxes per object class, we only exclude classes having fewer than 10 bounding boxes. Although this value is relatively low, we do not want to exclude too many classes upfront as it might turn out that some classes are easier to detect and therefore require less bounding boxes. Furthermore, some classes are having shared information such as Hail and Dent, which potentially decrease the required number of bounding boxes. Lastly, it gives the opportunity to estimate which data should be collected in the future, based on performance instead of the number of objects.

Hyperparameter optimisation is performed on a selective number of parameters. Cross-parameter interference requires to optimise parameters in a high-dimensional grid, instead of isolation. However, due to the computational complexity of training CNNs, testing a multi-dimensional hyperspace of parameters increases computation time drastically. To illustrate, three hyperparameters with five discrete options each, resulting in a total of $5^3 = 125$ configurations. Therefore, grid search on all parameters at once is not desired. Each parameter space is carefully considered, to reduce the dimension per parameter. Furthermore, some parameters are having small inference with other parameters and are, therefore, optimised in isolation.

Hyperparameters are firstly optimised on YOLO V3 with the default backbone: Darknet-53. This choice has been made as optimising all hyperparameters for each model results in too many configurations. Despite potential performance decrease compared with an hyperparameter optimisation per model, it is expected that the loss in performance is relatively low. This assumption is made as the different SSD models, as well as YOLO v3 are relatively similar.

5.1.1 Image Preprocessing

Section 4.5 explained the relevance of input normalisation and outlined several approaches. The used pre-trained models make use of normalisation with respect to the dataset pixel means in each channel (RGB). As the base network will not be re-trained, it seems most suited to implement the same normalisation technique. However, the normalisation technique can strongly influence the model performance. As the pre-trained weights used data normalisation during training, we restrict to different normalisation techniques and omit standardisation.

Let $x_{l,ijk} \in [0, 255]$ define the original pixel intensity in row i , column j , and channel $k \in \{RGB\}$ for image l . Define $\tilde{x}_{l,ijk}$, as the normalised pixel intensity, in a similar way. Then the mean pixel intensity of image l and channel k is given by Equation (5.1). Lastly, the channel means for a dataset with N instances is defined by Equation (5.2). Using this, we evaluate the following three normalisation techniques:

$$x_{l..k} = \frac{1}{W_{img}H_{img}} \sum_{i=1}^{W_{img}} \sum_{j=1}^{H_{img}} x_{l,ijk} \quad (5.1)$$

$$x_{...k} = \frac{1}{N} \sum_{l=1}^N x_{l..k} \quad (5.2)$$

- **Pixel centring per dataset channel:** Scale each image channel (RGB) to have zero mean across the entire dataset: $\tilde{x}_{l,ijk} = \frac{x_{l,ijk}}{x_{...k}} \forall l, i, j, k$.
- **Pixel centring per channel:** Scale each image channel (RGB) to have zero image mean: $\tilde{x}_{l,ijk} = \frac{x_{l..k}}{x_{...k}} \forall l, i, j, k$.

- **Pixel normalisation:** Scale each pixel to the range $[0 : 1]$: $\tilde{x}_{ijk} = \frac{x_{ijk}}{255} \forall i, j, k$.

We compare two image resize methods: resizing while preserving the aspect ratio and resizing while ignoring the aspect ratio. When preserving the aspect ratio, the image is first resized to have the largest dimension fit the target size. Secondly, it is placed randomly in the target canvas and the surrounding area is filled with the pixel mean of the dataset. Placing the image randomly on the target canvas makes the model more robust than centring it. When the resize is performed, while ignoring the aspect ratio, the image gets resized and stretched to fit the target canvas. The two approaches are visualised in Figure 5.1.

These two approaches are evaluated as preserving aspect ratio keeps the image more closely to its original, while making a part of the image irrelevantly filled with the dataset mean. Contrary, ignoring the aspect ratio moves further away from the original, but makes use of the full image size and contains, therefore, more information. Atwood (2007) explained that the inter cubic resize method preserves more information compared with Inter linear and Nearest Neighbour, when downsizing the images. As Section 4 showed that most images are larger than the input size of the network, inter cubic will be used.



FIGURE 5.1: Resize an image in landscape (top) or portrait (bottom).

5.1.2 Models

We evaluate the damage detection performance under different algorithms. Sections 3.4.2-3.4.4 point out that each model has its benefits and drawbacks. Although it is explained that one-stage models largely outperform two-stage models, we incorporate R-CNN in the comparison to serve as a benchmark for the two-stage models. The default configuration of SSD, as well as the extension: RFB-SSD and FSSD are incorporated in the comparison. we excluded the DSSD configuration, as Liu et al. (2018) explained that the performance improvement of DSSD is largely due to the ResNet-101 backbone, as well as the fact that RFB-SSD and FSSD are already outperforming SSD for small objects. Therefore, SSD is included to serve as a benchmark for the one-stage models. The extensions: RFB-SSD and FSSD are included for its performance improvement over SSD, with respect to small objects. Lastly, YOLO v3 is added to the evaluation as it compares with FSSD and RFB-SSD in terms of performance and is especially applicable for small objects. Its predecessors are not included as Redmon and Farhadi (2018) explained that the third version outperformed its predecessors by far.

We compare all one stage models across four base models: Darknet-53, ResNet-50, VGG-16 and MobileNet v2. The first three are selected due to the proven performance on classification tasks, as displayed in Table 3.1. MobileNet v2 is added to the comparison as it gives a fair estimation of the performance when implemented on a mobile device. R-CNN is only implemented on ResNet-50 as it serves as a benchmark and is expected to perform worse, compared with the one-stage methods in terms of the mAP. Furthermore, R-CNN is relatively slow and therefore not expected to be implemented in the Light Street, but only evaluated for research purposes.

5.1.3 Implementation

We used the Python programming language for the implementation of the various deep learning models. Initially, Keras (Chollet et al., 2015) has been used in combination with Tensorflow (Abadi et al., 2015) to implement the deep learning models. Keras is a high-level API, suitable for rapid prototyping due to its user-friendly syntax. Tensorflow is a highly optimised library for efficient matrix multiplications, required for large scale neural networks and therefore used by Keras for the computations.

Since Tensorflow used significant memory during preliminary model training, large GPU instances were required to provide the required GPU memory. Besides this, Keras trained relatively slow. These two drawbacks made us eventually select PyTorch (Paszke et al., 2017) as the final deep learning library. Rosinski (2017) showed that PyTorch is on average 20 percent faster compared with Keras for CNN models, delivering a significant improvement during parameter tuning.

The large number of parameters in the backbone of SSD, YOLO, and R-CNN requires the use of GPU enabled machines, which performs matrix multiplication much faster than CPUs. Fortunately, Pon Datalab provided access to their Amazon Web Services (AWS) environment, which is a cloud-based service, offering a variety of GPU enabled machines, as well as storage and service management. Although the GPU reduces training time from days to a few hours, the CPU is still used to move images from storage to the GPU and to preprocess the images (resize and augmentation). To load a batch of 16 images into the GPU, all 16 images have to be loaded from disk and preprocessed, taking up to 50 percent of the total training time. Fortunately, PyTorch can be used to run preprocessing and training in parallel, ensuring that the GPU does not have to wait for the preprocessing. This approach improved the GPU utilisation from 25 percent to 80 percent, reducing the training time significantly.

In addition to the parallel data training and loading, we decreased the training time significantly by keeping all images into the RAM of the instance. This approach removes the data loading of all images from disk during the preprocessing. The data is only read from disk at initialisation. This approach is possible due to the high RAM (64GB) present in the used instances and the relatively small dataset size. This improvement increased the GPU utilisation further to over 90 percent.

Cloud Architecture

To ensure a stable model, which can be used in training and production, we created the cloud architecture of Figure 5.2. The deep learning model, written in Python, is uploaded by use of a Docker container to Elastic Container Register (ECR). The

dataset is stored in Simple Storage Solution (S3), as well as the pre-trained weights for each model. Before each training, the following steps are taken:

1. **Create configuration file:** containing the model to train, hyperparameters, dataset location, and the pre-trained weights location in S3.
2. **Submit the training job:** Start Jupyter notebook and submit the job with the configuration file location to AWS Sagemaker.

The program will log all intermediate results to a Tensorboard file on S3 and store the trained weights every 5 epochs on S3. The benefits of the above-defined architecture are that different models and hyperparameters can easily be evaluated by modifying the configuration file. This removes the need for any adaptations in the program when different models, datasets, or hyperparameters need to be evaluated. This benefit extends beyond this research, by enabling the Pon Datalab to reuse the developed program for other object detection tasks in the future.

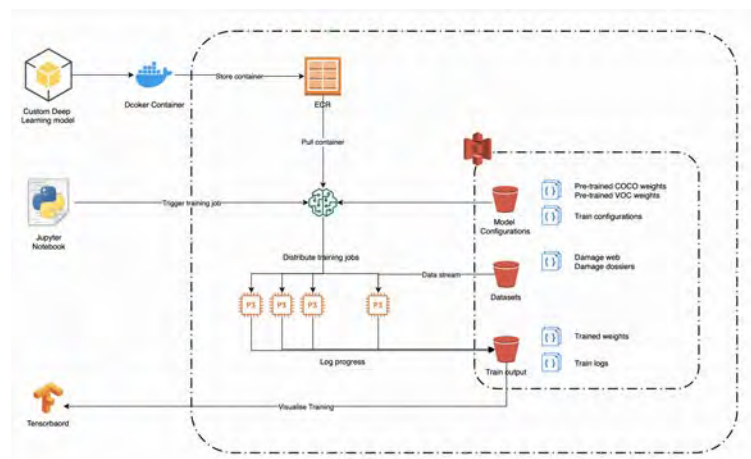


FIGURE 5.2: Implemented cloud architecture.

Light Street

Damage detection in the Light Street requires several preprocessing steps, to be able to transform the video stream into input images for the damage detection algorithm. Figure 5.3 summarises the preprocessing steps schematically. The 2 FPS video stream is sub-sampled by a factor 10 to reduce the number of frames to process. Vehicle and person detection are performed on the frames, where frames with a vehicle count ≥ 1 are kept. Each frame is cropped to the vehicle size, to remove irrelevant surroundings from the image. This approach reduces false positives and additionally, increases the size of the car in the input image. Persons are detected and removed from the image, for privacy matters, by filling the bounding box with white pixels. Lastly, the preprocessed image is passed through the damage detection algorithm to create the final predictions.

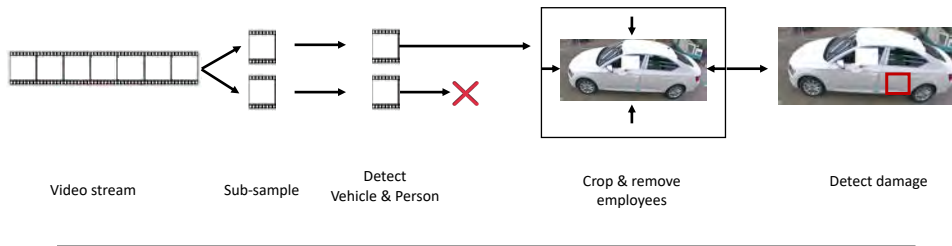


FIGURE 5.3: preprocessing steps on the Light Street video stream.

5.2 Hyperparameter Optimisation

The preprocessing of image data is more topic-related than dataset related. Therefore, initial parameter tuning on input preprocessing is performed for the *Damages Web* dataset and the found optimal settings are used for the *Damage Web* and *Damage Dossiers* respectively. The same approach is followed for a subset of hyperparameters, to reduce the required number of evaluations.

5.2.1 General Hyperparameters

One of the most important hyperparameters during the training of CNNs is the initialisation of the Learning rate and Batch size, as explained in Section 3.6. As the LR and BS are closely related, we use a small grid-search to optimise the parameters in the 2D state space: $\{(LR, BS) : LR \in [1^{-3}, 5^{-3}, 1^{-4}, 5^{-4}, 1^{-5}] \text{ and } BS \in [16, 32, 64]\}$.

We use the Adam optimiser, instead of SGD or RMSprop, since it is explained in Section 3.2.4 that Adam combines the benefits of SGD and RMSprop in a single optimiser. Furthermore, we use an adaptive learning rate schedule which reduces the learning rate by a factor of 0.5 if the evaluation loss did not improve for three consecutive epochs. This approach reduces overfitting as it reduces the learning rate when the evaluation loss is either stable or increasing. At the first few epochs, a higher learning rate is required than during the last few epochs as it is approaching the (local) minimum. We set the early stopping to 10 epochs, to ensure that the learning stops when the evaluation loss did not improve for 10 epochs. This number is relatively high but has been selected to see how the model behaves when it starts to overfit. Lastly, we use for the L1 regularisation: $\alpha = 0.0001$ following the approach of Liu et al. (2015).

The threshold for the Jaccard overlap in YOLO v3 and all SSD models, is set to 0.5 as proposed by Liu et al. (2015) and Redmon and Farhadi (2018). The non-max suppression threshold is set to 0.5, being in line with the papers of Liu et al. (2015) and Redmon and Farhadi (2018). Lastly, we set the maximum number of detections to 100 to restrict the time of the NMS algorithm.

5.2.2 Augmentation

Using augmentation increases the variety of images in the dataset and is especially of use for small datasets. Although our dataset is significantly larger compared with Patil et al. (2017), Li et al. (2018) and De Deijn (2018), image augmentation will still be used. The following augmentations, explained in Section 3.6.2, are evaluated on the resized images to enlarge the dataset. The first two augmentation methods are included by default, as recommended by Liu et al. (2015) for SSD models. Multiply, rotate and Gaussian blur are evaluated in addition to the default augmentation.

The frequently used vertical flipping is omitted as it is expected to complicate the learning process and does not represent the real situation accurately.

- **Cropping and Padding**
- **Horizontal flipping:** $p = 0.5$.
- **Contrast Normalisation:** Normalised the contrast in an image.
- **Multiply:** To make the model more robust to different light conditions.
- **Rotate:** Random rotations are used to make the model more robust to different camera angles. Furthermore, rotation is expected to make the model more robust against different scratches. The fraction horizontal scratches is considerably larger than slightly rotated scratches.
- **Gaussian blur:** Small Gaussian blur is added to distort the image slightly and make it more robust against quality loss in images.

5.2.3 Transfer Learning

As described in section 3.1, Patil et al. (2017) explained that transfer learning for damage detection works better when the initial model is trained on a broad class of objects. Therefore, pre-trained models on PASCAL VOC 2012 and COCO 2014 are used with twenty and eighty object classes respectively. Both datasets are incorporated in this research as Pont-Tuset and Gool (n.d.) explained that COCO 2014 has in general smaller object where 80 percent of the objects are below 5 percent of the image size, compared with 50 percent for PASCAL VOC 2012. As the object sizes are relatively small for both damage datasets, it is expected that COCO 2014 pre-trained will result in the best performance. Besides this, the diversity of the objects is larger, making it potentially better for damage detection, as explained by Patil et al. (2017). However, to test this assumption, both PASCAL VOC 2012 and COCO 2014 are used for the pre-trained weights.

It is explained in Section 3.6.3, that the number of trainable layers is a trade-off between overfitting and flexibility of the model. To avoid overfitting, we initially train only the extra layers, confidence layers, and location layers. The base network, as well as any normalisation and pyramid layers, are initially frozen. Based on the first results, we will include more layers in the training if overfitting remains within limits while improving evaluation performance.

5.2.4 Anchor Boxes

The different SSD configurations, as well as YOLO v3 require the construction of default anchors. Liu et al. (2015) provide a detailed calculation for the construction of the default anchors, which is independent of the dataset. Therefore, the proposed default anchors from the initial authors of SSD (Liu et al., 2015), FSSD (Li & Zhou, 2017), and RFB-SSD (Liu et al., 2018) will be used for each model. The used default anchors are displayed in Appendix G.

Redmon and Farhadi (2017) showed that YOLO v3 largely benefits from default anchors, which are set based on the used dataset. They propose to use K -means clustering, to construct 9 default anchors for each dataset. Therefore, we normalise each bounding box according to Equation (5.3) and Equation (5.4) to compensate for the difference in image size. $\tilde{H}_{bb} \in [0, 1]$ and $\tilde{W}_{bb} \in [0, 1]$ represent the normalised

box wide, whereas the original bounding box is of size $H_{bb} \times W_{bb}$ and the image of size: $H_{image} \times W_{image}$. We applied K-means clustering with $k = 9$, as proposed by the authors, on each dataset, and displayed the exact cluster centres in Appendix G. The clusters are visually represented in Figure 5.4a and Figure 5.4b for the *Damage Web* and *Damage Dossiers* respectively.

$$\tilde{H}_{bb} = \frac{H_{bb}}{H_{image}} \quad (5.3)$$

$$\tilde{W}_{bb} = \frac{W_{bb}}{W_{image}} \quad (5.4)$$

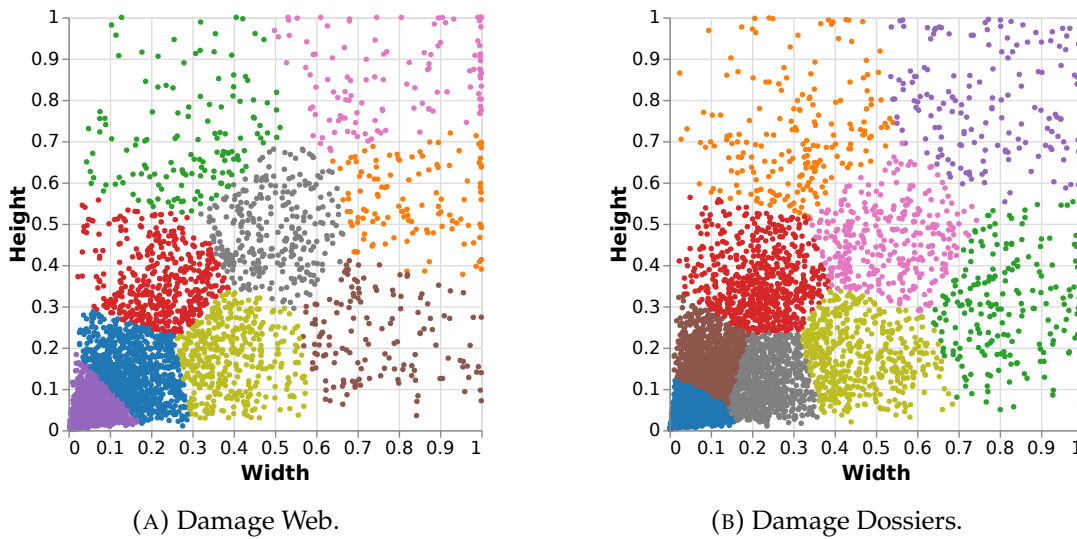


FIGURE 5.4: Bounding box clustering for anchor dimension estimation.

5.3 Evaluation

Although SSD and YOLO typically use a different loss function during the BP, we applied the loss function of SSD to all models, enabling a fair comparison between the different models. Besides the loss function, a confusion matrix is used to compare the model performance with the employees of Pon Logistics. This measure is added as it shows if damages get misclassified and which classes are frequently interfering with each other. The third and last method used to compare model performance is the mAP (See Section 3.5). The mAP will be reported for an IoU threshold of 0.5, instead of multiple thresholds as suggested by the COCO 2017 challenge. The mAP of PASCAL VOC2007 is used instead of the mAP of COCO 2017 as the second tends to reward models that are better at precise localisation, which is of lower relevance for Pon Logistics.

Chapter 6

Results

This section describes the results of the experiments, defined in Section 5. The results of the initial parameter tuning are described in Section 6.1 and are used as starting values for all other experiments. Section 6.2 compares the performance between different deep learning models and model configurations for the *Damage Web* dataset. The same analysis is performed for the *Damage Dossiers* in Section 6.3. The performance influence of extending internal data with external data, from the internet, is evaluated in Section 6.4. The model performance on the *Light Street* data is evaluated in Section 6.6 and the inference speed per model is shown in Section 6.7.

6.1 Initial Parameter Tuning

All evaluations for the initial parameter tuning were performed with the *Damage Web* dataset. Parameter tuning for the input preprocessing stage is presented in Section 6.1.1. The hyperparameters, from Section 5.2, are presented in Section 6.1.2.

Initially, the default augmentation methods, as explained in Section 5.2.2, were slightly tuned. The cropping and padding augmentation was evaluated in the following domain: $\{(\alpha_{crop}, \alpha_{pad}) : \alpha_{crop} \in \{0.1, 0.3, 0.5, 0.7\} \text{ and } \alpha_{pad} \in \{1.1, 1.3, 1.5, 1.7\}\}$. The performance comparison for the cropping and padding augmentation is shown in Table 6.1. Cropping seems to result in the best mAP at $\alpha_{crop} = 0.3$, where more or fewer cropping results in a lower mAP independently of α_{pad} . An increased α_{pad} seems to improve the mAP, independently of the α_{crop} . The optimal parameter setting in the performed grid search is $(\alpha_{crop} = 0.3, \alpha_{pad} = 1.7)$.

TABLE 6.1: Effect of α_{crop} and α_{pad} on the mAP for YOLO v3 with Darknet-53, Adam optimiser, a learning rate of $1e-3$, normalised input with 0-1 scaling, and image resize while ignoring the aspect ratio.

Pad \ Crop	Crop			
	0.1	0.3	0.5	0.7
1.1	0.170	0.183	0.177	0.153
1.3	0.181	0.198	0.182	0.164
1.5	0.216	0.243	0.220	0.197
1.7	0.239	0.251	0.244	0.239

6.1.1 Input Preprocessing

Figure 6.1a compares three image normalisation techniques from Section 5.1.1: 0-1 scaling, dataset mean subtraction, and image mean subtraction. The normalisation by image mean (pixel centring per channel) seems to perform less, compared with 0-1 scaling (pixel normalisation) and dataset mean normalisation (pixel centring per dataset channel). The best performance is achieved when the pixel mean of the complete dataset is subtracted per image channel. Furthermore, Figure 6.1 indicates that preserving the aspect ratio does not improve the mAP.

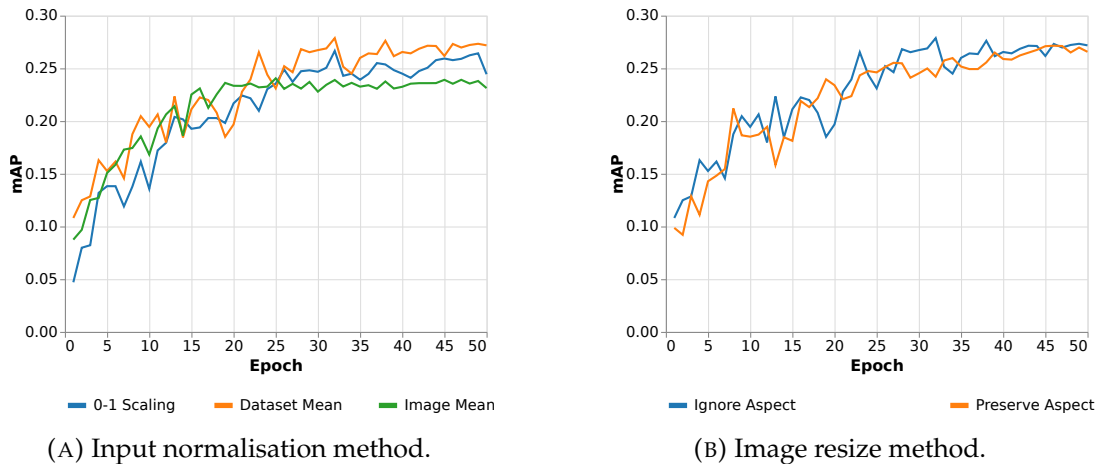


FIGURE 6.1: Image normalisation method (left) and image resize method (right).

Despite the absence of a clear performance difference between ignoring the aspect ratio and preserving the aspect ratio, a performance difference is present between the different damage classes. Figure 6.2 indicates an increased performance for the class Missing, when preserving the aspect ratio. Contrary, ignoring the aspect ratio seems to improve the mAP on the class Hail and the class Scratch. As the detection of scratches is of more relevance for Pon Logistics, we ignore the aspect ratio in the further extension of this research.

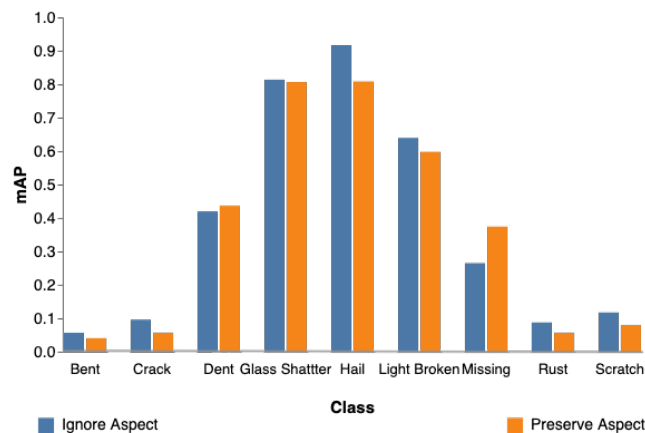


FIGURE 6.2: mAP comparison for resize without perseverance of the aspect ratio (blue) and with perseverance of the aspect ratio (orange).

6.1.2 Hyperparameter Optimisation

Firstly, different augmentation techniques are compared in addition to the previously optimised parameters: $\alpha_{crop} = 0.3$ and $\alpha_{pad} = 1.7$. These parameters, in combination with normalisation by the dataset mean and resize while ignoring the aspect ratio, will be used throughout this section. Secondly, different transfer learning strategies are evaluated.

Augmentation

A preliminary evaluation showed large diversity in mAP per damage class (Figure 6.2), where the classes Bent, Crack, Rust, and Scratch were lacking behind in performance. Although the performance on all four classes is relatively low, the number of objects of class Scratch is relatively large. Therefore, all images containing at least one scratch were isolated and a model has been trained on this dataset. Table 6.2 shows the performance of seven different evaluations, where evaluation 1 serves as a benchmark. For a larger image size (evaluation 2), the model tends to locate the objects more precisely. However, the mAP does not increase to a large extent. As the mAP takes objects into account with an IoU of at least 0.5, it gives rise to the idea that an increased image size improves box locations, which already had an IoU of 0.5. Evaluation 3 and 4 result in an increased performance when more rotation is added. Lastly, evaluation 5 and 6 show that Gaussian blur and multiplying the pixels to change light conditions, increased the performance further. The best mAP was achieved for evaluation 6, where the loss values are comparable with the benchmark evaluation, but the mAP increased by a factor 2.4.

TABLE 6.2: Effect of augmentation on scratch detection for IoU= 0.50, with the model: YOLO v3 and backbone: Darknet-53 on 50 epochs. The following hyperparameters are used: $\alpha_{crop} = 0.3$, $\alpha_{pad} = 1.7$, horizontal flipping($p = 0.5$), resize while ignoring the aspect ratio, $LR = 1e^{-3}$, and $BS = 32$.

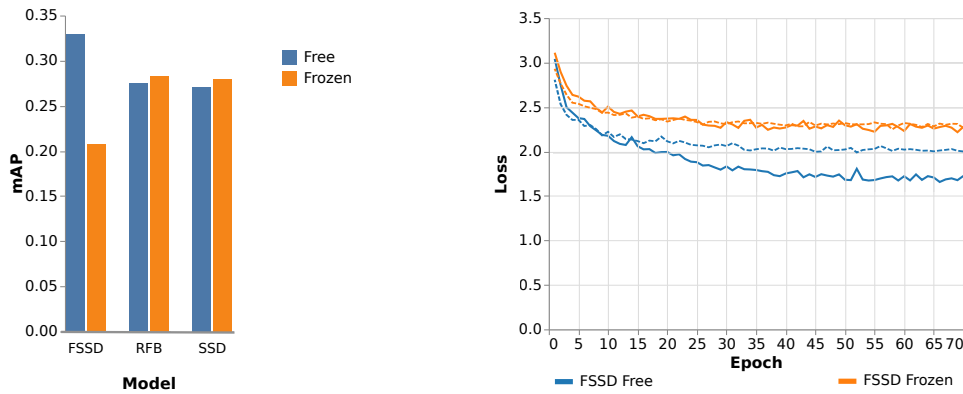
Evaluation	1	2	3	4	5	6	7
Scratch subset 416×416	✓		✓	✓	✓	✓	
Scratch subset 680×680 .		✓					✓
Rotation range			10	15		30	15
Gaussian blur & multiply					✓	✓	✓
mAP	0.067	0.076	0.132	0.152	0.142	0.161	0.115
Total loss	4.04	3.62	4.11	4.08	4.16	4.10	3.77
Confidence loss	2.33	2.24	2.27	2.32	2.32	2.27	2.16
Location loss	1.71	1.38	1.84	1.76	1.84	1.83	1.61

Transfer Learning

The learning performance is highly affected by the fraction of locked weights during transfer learning. A trade-off exists between the flexibility of the model and overfitting. Figure 6.3a shows the mAP after 70 epochs for two different levels of frozen layers. The category *frozen* freezes all layers and only trains the extra layers, the location layers and the confidence layers. The category *free* unfreezes more layers where

only the base network is frozen and all other layers¹ are trainable. FSSD benefits largely from the additional trainable weights, increasing the mAP from 0.21 to 0.33.

Figure 6.3b shows the location loss for FSSD, with the solid line as the training loss and the dashed line as the validation loss. With the frozen layers, the model is not able to learn the data to a large extent, as the training and validation loss is approximately equal (high bias). The added trainable layers increased the flexibility of the model, resulting in a lower loss value for both training and evaluation.



(A) Influence on the mAP per model. (B) Influence on the confidence loss for FSSD.

FIGURE 6.3: Influence of more trainable weights, using the same settings as for Table 6.2 with the parameters of evaluation 6.

As this preliminary research showed promising results for FSSD, we made a more detailed comparison for the effect of the number of trainable layers on the loss value and the mAP, respectively. We decided to perform the detailed analysis on FSSD solely, as RFB and SSD did not increase in performance when more layers are trainable. The results are summarised in Table 6.3, showing more overfitting when the number of trainable layers is increased. Contrary, the mAP increased when unfreezing more layers. The mAP decreased significantly when the base network is added to the trainable layers. The model tends to overfit when the base network is trainable, resulting in a large difference between the training loss and the evaluation loss. The last evaluation shows the performance when the base network is frozen for the first 50 epochs and made trainable for the last 20 epochs. Using this, the base network is used during the fine-tuning process with a relatively low learning rate. This resulted in the highest mAP while having a comparable evaluation loss with evaluation 1.

¹FSSD: normalisation, extra, transformation, pyramid, location, and confidence. RFB: normalisation, extras, location, and confidence. SSD: normalisation, extras, location, and confidence.

TABLE 6.3: Effect of the number of trainable layers on the loss and mAP respectively. Trained on 70 epochs.

Evaluation	1	2	3	4	5	6
Base					✓	> epoch 50
Normalisation				✓	✓	✓
Transformation			✓	✓	✓	✓
Pyramids			✓	✓	✓	✓
Extras		✓	✓	✓	✓	✓
Location	✓	✓	✓	✓	✓	✓
Confidence	✓	✓	✓	✓	✓	✓
mAP	0.139	0.211	0.252	0.331	0.221	0.341
Training loss (total)	4.644	3.818	3.627	3.427	2.073	2.807
Validation loss (total)	4.764	4.281	4.122	3.988	4.315	3.883

6.2 Model Comparison Damage Web

This section shows the batch size (BS) and learning rate (LR) in a small grid search for the *Damage Web* dataset and compares the influence of different pre-trained weights, in combination with different default anchors. Lastly, it draws a performance comparison between different deep learning models, configured with varying backbones (base models). In the comparison, the input was scaled by the dataset mean, the image was resized while ignoring the aspect ratio, and the augmentation was set to the best evaluation of Section 6.1.2: $\alpha_{crop} = 0.3$, $\alpha_{pad} = 1.7$, horizontal flipping ($p = 0.5$), rotation of 30 degrees, Gaussian blur, and multiply. All layers, except the base network, were trainable for the SSD, FSSD, and RFB-SSD models.

Batch Size and Learning Rate

The batch size (BS) and learning rate (LR) were optimised using a small grid in the dimensions $BS \in \{16, 32, 64\}$ and $LR \in \{1e^{-3}, 5e^{-3}, 1e^{-4}, 5e^{-4}, 1e^{-5}\}$. The results are shown in Table 6.4. A batch size of 32 seems to outperform the other batch sizes on all evaluated learning rates. Using a higher batch size (64) seems to prefer a higher learning rate. Overall, batch size 32 with learning rate $1e^{-4}$ resulted in the highest mAP on the evaluated grid.

TABLE 6.4: Batch size and learning rate optimisation.

BS \ LR	$1e^{-3}$	$5e^{-3}$	$1e^{-4}$	$5e^{-4}$	$1e^{-5}$
	16	0.286	0.289	0.291	0.287
32	0.303	0.313	0.333	0.292	0.288
64	0.234	0.216	0.207	0.196	0.179

Anchor Size and Pre-trained Weights

Figure 6.4a illustrates the mAP performance for pre-trained weights: COCO and PASCAL VOC 2012, for different anchor sizes: COCO, PASCAL VOC 2012, and K-means clustering as suggested in the paper of Redmon and Farhadi (2017). The clustered anchors, illustrated in Figure 5.4a are used, where all anchor sizes are presented in Appendix G. Figure 6.4b shows the lowest loss in the localisation of the bounding boxes for the COCO pre-trained weights with corresponding anchors. The anchors constructed with K-means clustering result in a higher loss and lower mAP, compared with the anchors used during training of the pre-trained weights.

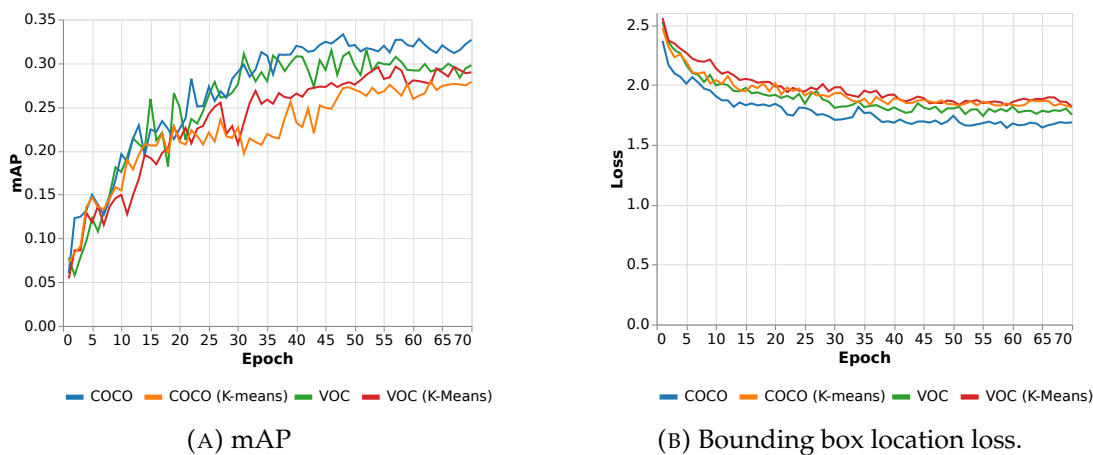


FIGURE 6.4: Influence of pre-trained weights and anchors on the mAP (left) and location loss (right).

Model and Backbone Comparison

Table 6.5 compares the mAP for all five models with different backbones (base models). The trainable layers were set to the settings as discussed in **Transfer learning**, with only the base network frozen.

TABLE 6.5: Cross comparison on the mAP between different models with varying backbone configurations For each model, the backbone with the highest mAP is indicated in bold.

Model	Darknet-53	ResNet-50	VGG-16	MobileNet v2
FSSD	0.330	0.254	0.278	0.273
RFB	0.275	0.272	0.298	0.207
SSD	0.271	0.257	0.234	0.223
YOLO v3	0.333	0.297	0.257	0.253
R-CNN	-	0.241	-	-

Both YOLO v3 with Darknet-53 and FSSD with Darknet-53 resulted in the highest mAP. To boost the performance further, the base network was added to the trainable layers and the networks were trained for an additional 30 epochs. This approach was used as Table 6.3 showed a promising improvement in both the loss and mAP for this strategy. The performance of FSSD improved from 0.330 to 0.361 and the performance of YOLO v3 improved from 0.333 to 0.413. YOLO v3 benefits the most with

an improvement of 25 percent, compared with 9 percent for FSSD. A breakdown per object class is visualised in Figure 6.5. The class Missing seems to benefit largely from the additional training, as well as the smaller objects: Bent, Rust, and Scratch. YOLO v3 seems to outperform FSSD in all classes, except the class: Light broken.

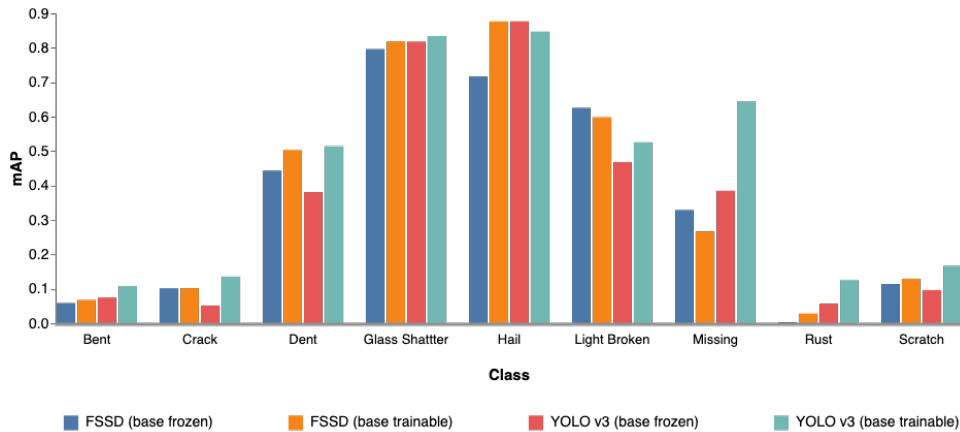
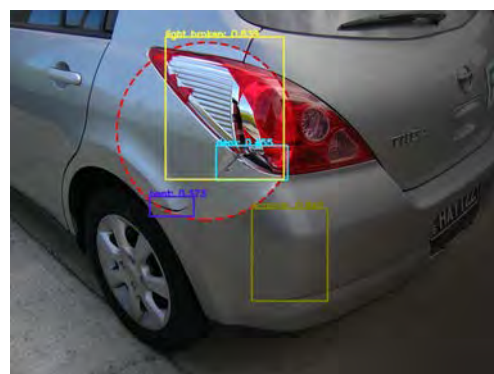


FIGURE 6.5: mAP comparison per class for FSSD and YOLO v3 before unfreezing the base network and after unfreezing the base network.

Figure 6.6a shows the ability of the model to detect small scratches on the vehicle accurately. The benefit of damage localisation instead of classification is shown in Figure 6.6b. The model accurately predicts different damages on the same vehicle, providing a more detailed result on both the location and the class. Figure 6.7 illustrates the robustness of the model against light fluctuations and reflections and makes this comparison before and after the repair. Figure 6.8 shows that the model is able to detect small damages, but is having trouble to accurately construct a bounding box. Furthermore, it fails to predict one large scratch, while being able to detect the others. Although the model seems to be robust against water on the surface, the water might complicate the construction of the bounding boxes. Additional excerpts are provided in Appendix H.



(A) Scratches.



(B) Bending (dark blue), dent (light blue), light broken (yellow), and scratch (ochre).

FIGURE 6.6: Two examples with detected damage for YOLO v3 with Darknet-53.

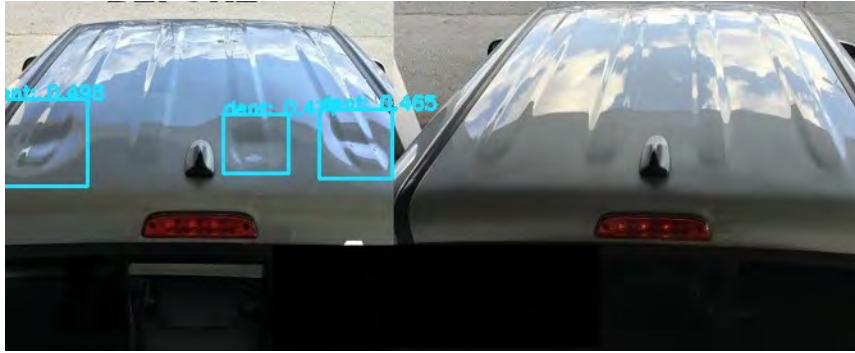


FIGURE 6.7: Detected damages on a reflecting surface before the repair (left) and after the repair (right) for YOLO v3 with Darknet-53.

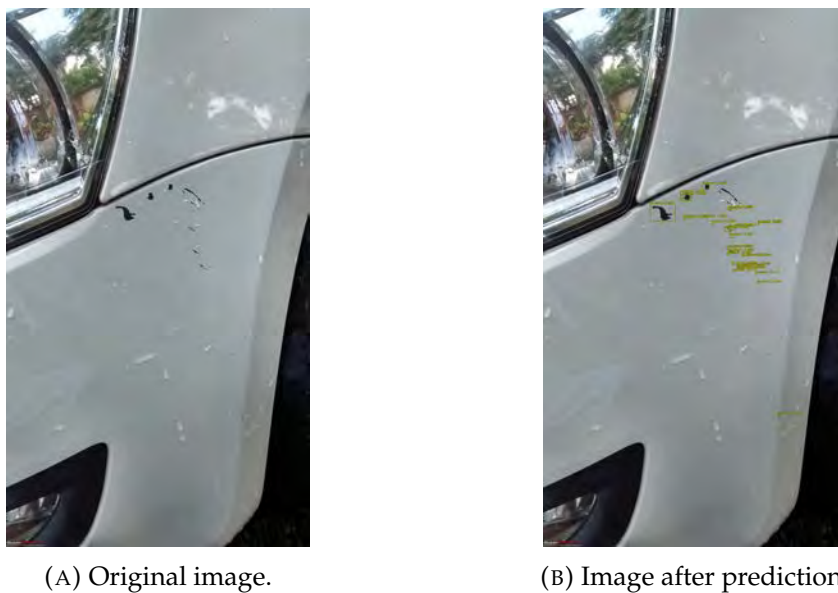


FIGURE 6.8: Damaged vehicle before the prediction (original image) (left) and image after prediction of the bounding boxes with YOLO v3 and Darknet-53 .

6.3 Model Comparison Damage Dossiers

The best performing model on the *Damage Web* dataset was used to set a benchmark on the *Damage Dossiers*. YOLO v3 with Darknet-53 backbone was used, as well as FSSD with Darknet-53. We firstly optimised the LR and BS in a similar way as done for the *Damage Web* dataset. Table 6.6 shows the performance difference on the mAP for varying parameter values. The optimal learning rate, on the evaluated grid, is $5e^{-3}$ and the optimal batch size is 32. The LR is slightly higher compared with the LR for the *Damage Web* dataset.

TABLE 6.6: Batch size and learning rate optimisation for 70 Epochs.

BS \ LR	$1e^{-3}$	$5e^{-3}$	$1e^{-4}$	$5e^{-4}$	$1e^{-5}$
16	0.236	0.257	0.287	0.299	0.301
32	0.305	0.349	0.315	0.292	0.263
64	0.313	0.321	0.300	0.240	0.221

We set the batch size to 32 and the learning rate to $5e^{-3}$. The train and evaluation loss development for FSSD is visualised in Figure 6.9. The model clearly started overfitting after epoch 70, where we unfroze the base network. The confidence loss for the training data started to decrease, while the confidence loss for the evaluation data increased. After 140 epochs the model started to overfit extremely. YOLO v3 showed similar patterns, indicating that this dataset overfits more easily, compared with the *Damage Web* dataset.

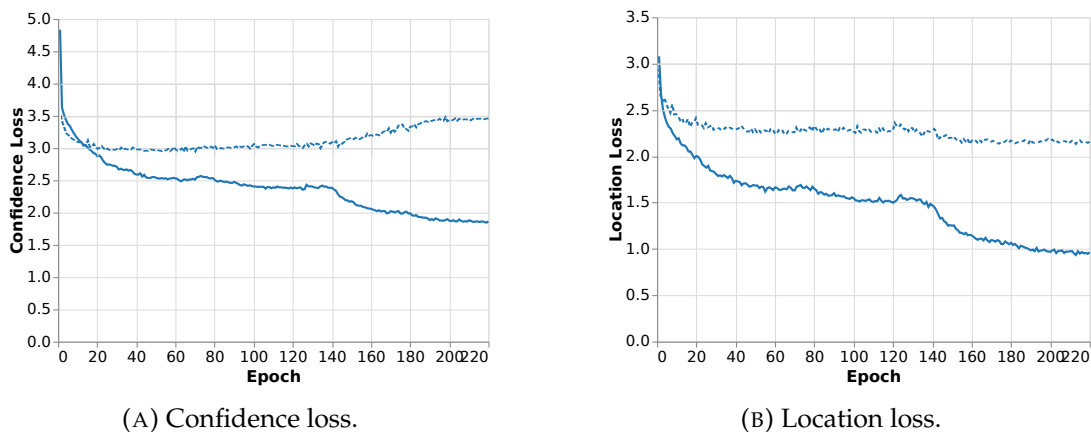


FIGURE 6.9: Loss per epoch for FSSD with the confidence loss (left) and localisation loss (right).

We resumed training from epoch 70 and added heavier data augmentation with padding on 2.0 and cropping on 0.1, as the damage dossiers contain more close-up images. Furthermore, we froze the base network to avoid overfitting. This approach improved the confidence loss for FSSD to 2.781 and the localisation loss to 1.942. The same methodology was followed for YOLO v3, where the model results per object class are shown in Table 6.7. In contrast with the results for the *Damage Web* dataset, FSSD seems to outperform YOLO v3 on almost all classes. The model seems to be robust against water and reflections, as displayed in Figure 6.10a and Figure 6.10b respectively. Although, the model can handle reflections, it seems to be less robust than the model which is trained on the *Damage Web* dataset. Figure 6.11a and Figure 6.11b show that the model is having trouble to detect some of the scratches under different conditions. This might indicate that the model requires more training samples to perform consistently under different light conditions and angles. Contrary, the reflection in Figure 6.11a might cause the poor detection quality, as the reflection is on the same location as the scratch.

TABLE 6.7: mAP comparison between YOLO v3 and FSSD per object class.

Class	YOLO v3	FSSD
Bent	0.232	0.246
Cover Damage	0.142	0.176
Crack	0.257	0.467
Dent	0.272	0.323
Glass Shattter	0.654	0.713
Light Broken	0.098	0.202
Missing	0.276	0.365
Rust	0.143	0.156
Scratch	0.176	0.186
Tire Crack	0.0	0.411
Window Text	0.615	0.600
mAP	0.266	0.349



(A) Scratch.



(B) Dent (blue) and glass shatter (yellow).

FIGURE 6.10: Two examples of detected damage for FSSD with Darknet-53.



(A) Scratches, partly missed.



(B) Scratches detected.

FIGURE 6.11: Two examples of detected damage for FSSD with Darknet-53.

6.4 Model Comparison Combined

A cross-comparison is made on how well each trained model for dataset y is performing on the evaluation set of dataset x . Table 6.8 shows the used training data on the y -axis and the evaluation dataset on the x -axis. To construct an unbiased comparison, we compared all models on the same classes. Therefore, we removed the class Hail from the *Damage Web* dataset and removed the classes Builen, Cover damage, Tire crack, and Window text from the *Damage Dossiers*. It can be seen that increased performance on the *Damage Dossiers* is achieved when the *Damage Web* data is added to the training instances. Contrary, the performance on the *Damage Web* data decreased when the *Damage Dossiers* are added to the training instances.

TABLE 6.8: mAP performance for the best model, trained on the dataset of axis x .

	Damage Web	Damage Dossiers	All data
Damage Web	0.348	0.159	0.213
Damage Dossiers	0.170	0.258	0.247
All data	0.293	0.304	0.301

6.5 Employee Performance

We compared the performance of the model with the performance of two employees from Pon Logistics. For this, we used 100 images from the *Damage Dossiers* dataset. Since each dossier contains a detailed description of the damage location and class, the ground truth is known. For the *Damage Web* dataset, no exact ground truth is known and therefore this dataset was excluded from the comparison. Two employees, working at the Pon Logistics Light Street, have annotated the total of 100 images manually. The model used approximately 1.5 minutes to process all images, where the employees used approximately 2 hours and 15 minutes.

A confusion matrix is shown in Table 6.9. This matrix is constructed by ignoring the different ways of annotating the same damage. For this, we count each damage class only once in each image and assign True if the model/employee predicts the damage class correctly. Which is, having a bounding box overlap of at least 30 percent with the correct class. We assign False otherwise. The confusion matrix shows that both the employee and FSSD Darknet-53 made a relatively low number of errors between the damage classes. The last column indicates the predicted damages which are, in fact, no damage. The number of missed damages are presented in the last row. Compared with the employee, the model is especially better in detecting the class Bent, as well as the class Cover damage. Contrary, the employee seems to be better at detecting the class Dent and Scratch.

The random sample of 100 images seems to be slightly easier compared with the general images in the *Damage Dossiers* dataset. The mAP of the model was 0.382. Figure 6.12 compares the performance of the employees (left) with the performance of FSSD with Darknet-53 (right). The first row shows that the employees assigned the label Scratch, instead of Scratch and Dent. The second row shows that the model did not detect the scratch at the rim and incorrectly detected a scratch on the tire. The third row illustrates that the employees missed the right broken tail light. Furthermore, it indicates that the model classifies the damages in more detail. The fourth row shows that the model detects the cover damage, which is not annotated by the

employees. Lastly, the glass shatter in row five is detected in more detail by the model.

TABLE 6.9: Confusion matrix with the prediction on the y-axis and the ground truth on the x-axis. The predictions are made with threshold 0.25. (Bent, No Damage) indicates that the model predicted class Bent but there was no Bent present. Contrary, (No Damage, Bent) indicates that the model did not predict Bent but there was in fact a damage of class Bent present.

	Employee								FSSD Darknet-53									
	Bent	Cover Damage	Crack	Dent	Glass Shatter	Light Broken	Missing	Scratch	No Damage	Bent	Cover Damage	Crack	Dent	Glass Shatter	Light Broken	Missing	Scratch	No Damage
Bent	3									8								
Cover Damage		5									8							
Crack			2									2						
Dent				35				1	5				27				3	3
Glass Shatter					8									7				
Light Broken						2									3			
Missing							2									2		
Scratch								47	4				3				42	5
No Damage	6	4	1	2	2	1		3	-	1	1	1	7	3			6	-



(A) Scratch.



(B) Scratch and Dent.



(C) Scratch.



(D) No Damage.



(E) Scratch and Dent.



(F) Scratch, Dent, Bent, and Light broken.



(G) Scratch.



(H) Scratch and Cover damage.



(I) Glass shatter and Dent.



(J) Glass shatter and Dent.

FIGURE 6.12: Example in performance difference between the employees (left) and FSSD Darknet-53 with confidence threshold 0.25 (right).

6.6 Light Street Performance

We evaluated the performance in the Light Street with the model trained on all data (*Damage Web* and *Damage Dossiers*), as specified in Section 6.4. The performance evaluation was done by use of the master data as ground truth, as the low-resolution images make it impossible to detect the damage with the human eye. To evaluate the performance in the Light Street, we took images from 50 vehicles with detected damage. We excluded images when the detection location is not the Light Street. Any damage detected after the Light Street does not guarantee that the damage was already present at the Light Street. This made us decide to exclude all images which did not have a detected damage in the Light Street. Furthermore, we added 50 vehicles without a detected damage anywhere in the supply chain. To ensure this, we sampled 50 undamaged vehicles from the vehicles which have been transported to a dealer or customer. By this, we are almost certain that all used negative examples are indeed negative. Table 6.10 shows the confusion matrix, constructed in a similar way as done in Section 6.5.

The model has a relatively low precision with a moderate recall. The model incorrectly finds many damages, ending up in the last column of the confusion matrix. The confusion between the different damage classes is relatively low. With this, the model seems to have trouble identifying the difference between background and a damage. When a damage is present, the model seems to accurately predict the correct damage class.

TABLE 6.10: Confusion matrix with the prediction on the y-axis and the ground truth on the x-axis. The predictions are made with threshold 0.2. (Bent, No Damage) indicates that the model predicted class Bent but there was no Bent present. Contrary, (No Damage, Bent) indicates that the model did not predict Bent but there was in fact a damage of class Bent present. The images where no damage was present and no damage is predicted are excluded from the matrix.

	Bent	Cover Damage	Crack	Dent	Glass Shatter	Light Broken	Missing	Scratch	No Damage
Bent	2								1
Cover Damage		9							23
Crack									
Dent				14				1	43
Glass Shatter					1				18
Light Broken									3
Missing									
Scratch								2	8
No Damage	4	6	1	1	8	8	-		

We noticed that the environmental variables in the Light Street are largely influencing the detection performance. Figure 6.13a illustrates that the strong light influence, in combination with the low resolution is creating multiple false positives. The same issue arises in Figure 6.13b, where the roof got incorrectly classified as dent. Figure 6.14 shows a vehicle at the Light Street before removal of the cover (left) and after

removal of the cover (right). The cover damage is not visible from the image and the model was not able to detect the cover damage. With the cover partly removed, Figure 6.14b shows that the model accurately detected the partly removed cover and detected the right dent. The left dent was not detected by the model. Figure 6.15a shows that, despite the reflection, the model is able to detect a bonnet dent correctly. Figure 6.15b shows that the model is able to detect the dent at the right front door correctly.



(A) Incorrectly detected glass shatter and dent.



(B) Incorrectly detected dent.

FIGURE 6.13: Incorrectly predicted damages.



(A) Cover damage on the bonnet, not detected by the model.



(B) Detected cover damage and detected dent on the bonnet. The left bonnet dent is not detected by the model.

FIGURE 6.14: Vehicle with Cover damage and two dents at the bonnet. Vehicle before removal of the cover (left) and vehicle after partly removal of the cover (right).



(A) Correctly detected dent on the bonnet.



(B) Correctly detected dent on the right front door.

FIGURE 6.15: Correctly detected dents.

6.7 Inference Speed

Inference speed is measured for a single forward pass through the network, without required image preprocessing. The inference time in milliseconds is reported for academic purposes on a V100 Tesla GPU (Table 6.11). The inference time on a CPU is measured to resemble the most likely implementation at Pon Logistics. The CPU inference time is shown in Table 6.12.

TABLE 6.11: Inference time on an AWS p3.2xlarge instance, configured with one TeslaV100 GPU (16GB Memory), eight Intel Xeon E5-2686 v4 CPUs with a total of 61GB RAM. Each reported time in milliseconds per image for a forward pass through the network without augmentation, preprocessing and loading from disk. All inference times are reported in milliseconds.

Backbone	Input dimension	Darknet-53	ResNet-50	VGG-16	Mobilenet v2
YOLO v3	416×416	4.97	5.03	2.03	4.49
SSD	300×300	4.21	3.43	1.63	4.05
FSSD	300×300	4.01	3.85	1.98	3.87
RFB-SSD	300×300	6.11	5.75	4.00	4.89
Faster R-CNN	800×800	-	203	-	-

TABLE 6.12: Inference time on a 2.8GHz Intel Core i7 with 16GB RAM. Each reported time in milliseconds per image for a forward pass through the network.

Backbone	Input dimension	Darknet-53	ResNet-50	VGG-16	Mobilenet v2
YOLO V3	416×416	694	656	803	568
SSD	300×300	388	358	421	342
FSSD	300×300	615	437	700	387
RFB-SSD	300×300	447	497	613	324
Faster R-CNN	800×800	-	6,420	-	-

Chapter 7

Conclusion and Discussion

This section firstly concludes on the results and answers the research question in Section 7.1. Secondly, some limitations of this research are outlined in Section 7.2, as well as topics for further research are presented.

7.1 Conclusion

We will conclude on the results, presented in Section 6, to answer our research question: *How accurately can deep learning detect vehicle damages and how can these models be used to improve the logistics process?*. We will first look into the performance of the different models and datasets, to conclude if deep learning is able to locate damages. Secondly, we conclude how the performance of deep learning compares with human performance on the *Damage Dossiers* dataset. Lastly, we conclude on the performance in the Light Street and close this section by providing possibilities to optimising the business process by use of automatic damage detection.

Table 6.5 showed that FSSD with Darknet-53 and YOLO v3 with Darknet-53 yields the best mAP for the *Damage Web* dataset. Using different transfer learning techniques, a major improvement has been achieved, resulting in YOLO v3 with Darknet-53 as the best performing model. Therefore, we can conclude that, among the evaluated models, YOLO v3 with Darknet is best suited for damage detection on the *Damage Web* dataset. Furthermore, the performance of FSSD shows that damage detection benefits from the preserved contextual information in the FSSD model. This can be concluded as the performance of FSSD is significantly higher, compared with the performance of SSD. The *Damage Dossiers* dataset is affected by overfitting, indicating that too few data was present. The best model for the *Damage Dossiers* is FSSD with Darknet-53. The mAP on the *Damage Dossiers* dataset is lower, compared with the mAP on the *Damage Web* dataset. This gives rise to the idea that the damages at Pon Logistics are relatively difficult when compared with the *Damage Web* dataset. We showed, that the mAP on the *Damage Dossiers* dataset improved when the *Damage Web* dataset is added to the training data. Therefore, adding external data improves the performance on the internal data of Pon Logistics.

Comparing the performance of the employees with the performance of FSSD Darknet-53, showed that the model made 33 errors on an object level and the employee 29 for a subset of 100 images from the *Damage Dossiers* dataset. The employee is better at detecting the class Dent and more accurately assigns a damage to the correct class. FSSD Darknet-53 makes fewer *false negatives* on the class Bent and Cover damage. The employee makes fewer false negatives on the class Dent. With the fact that the model has a relatively large *true positive* fraction in the confusion matrix, we can conclude that deep learning is able to detect vehicle damages from images. Furthermore, the model annotates the damages in more detail compared with the employees. This is, the model makes a better distinction between different classes

and constructs smaller bounding boxes. FSSD with Darknet-53 seems to lack performance in the Light Street. The model got largely influenced by the strong light reflections, resulting in many *false positives*. Based on the confusion matrix, the model seems to have a relatively low precision with a moderate recall. The model seems to have trouble identifying the background (no damage) from a damage, resulting in the large proportion of *false positives*. Contrary, when a damage is present on the vehicle, and the model predicts a damage, the model seems to accurately predict the the correct damage class. Therefore, the model is able to distinguish the different damage classes within the light street.

In short, we can conclude that deep learning is able to detect vehicle damages. A relatively high performance is achieved for both the *Damage Web* dataset and the *Damage Dossiers* dataset. We showed that the model is able to detect small damages and distinguish accurately the different damage classes. Furthermore, we evaluated different deep learning models in combination with the different backbones and showed that the performance is strongly depending on the dataset. This is, FSSD with Darknet-53 works best for the *Damage Dossiers* where YOLO v3 with Darknet-53 works best for the *Damage Web* dataset. Comparing the performance of the employees with the performance of the deep learning model showed that the model achieves comparable results. The performance in the Light Street is relatively low, compared with the performance on the *Damage Web* dataset and the *Damage Dossiers* dataset. Despite this, we showed that deep learning can be used to locate and classify damages in detail.

Using the automatic vehicle damage detection within the process of Pon Logistics is possible. However, automatic detection without the inference of employees is not yet possible. Some adaptations are required in order to improve the performance and reliability of the model. We recommend to implement more cameras and to place each camera closer to the vehicle. Using this, the resolution is increased and this could potentially improve the detection of scratches. Furthermore, we recommend Pon Logistics to evaluate the ability to place the cameras outside the Light Street. The *Damage Dossiers* were mainly captured outside, having a more natural light and therefore fewer reflections, resulting in a higher performance. Alternatively, replacing the current lights with a more natural light might reduce the reflection.

7.2 Discussion

In this section, we discuss several advantages and limitations of this research. Furthermore, we recommend further studies to focus on the limitations of this research, by providing potential resolutions.

Limited Data

We used significantly more data than previous research Patil et al. (2017) and Li et al. (2018) and De Deijn (2018). Besides this, the dataset was still limited in size and diversity. We noticed that the model is not completely stable against different light conditions, camera angles, and zoom levels. Therefore, using a larger training set might increase the stability of the model. Furthermore, the images from Pon Logistics contained multiple images of the same damage, making the total number of distinct damages lower. As we evaluated our research on multiple datasets, we expect that the outcomes on different training techniques and augmentation methods

can be projected to a situation with more training images. The images from Pon Logistics contain more reflections, compared with images from the web. This might be the reason for the lower mAP. Therefore, we suggest exploring the possibilities of reflection removal before applying CNNs. Lastly, the data from Pon Logistics is limited to the Volkswagen concern, therefore, no claims can be made on the performance for other vehicle brands.

Annotations

Our approach showed that the damage detection results in a relatively low interference between classes. Therefore, using damage detection instead of classification for the complete image improves the performance. Although the detailed annotation process results in less class interference, it is explained that the localisation of each damage is non-trivial. Scratches can be separated into multiple scratches or labelled as one large scratch. Due to the manual annotation process, the reported performance is largely influenced by the annotated ground truth boxes. As damage location is more ambiguous than for objects of the COCO or PASCAL VOC 2012 challenges, further research could use multiple persons to cross-validate or average the annotations. Lastly, further research could focus on the impact on the mAP when different annotation granularities are used.

Default Anchors

Despite our approach to use k-means clustering, as suggested by Redmon and Farhadi (2018), to construct the default anchors, we did not realise a performance improvement over the COCO default anchors. This might be due to the low number of training samples, but we were not able to test this assumption. Therefore, we suggest that if more data becomes available, this assumption gets reevaluated.

Model Comparison

We used multiple deep learning models and compared these models for a variety of base networks. This comparison showed strong performance differences between the different models, indicating that damage detection especially benefits from models which are focused on small objects and contextual information. Although the performance differences were large between some models, we evaluated each model and backbone configuration only once. We performed each evaluation only once to reduce the overall training time. The random initialisation of weight within the network requires to perform multiple comparisons for each configuration. Although we found large performance differences, further research could try to evaluate each configuration multiple times to statistically test the performance difference.

To limit the training time, we optimised the hyperparameters on YOLO v3 with Darknet-53 and used these parameters for all other models. This approach relies on the assumption that the (sub)optimal hyperparameters of YOLO v3 with Darknet-53 are representative for the other models. As we did not validate this claim, further research could focus on identifying if this claim is justified by optimising the hyperparameters for each model.

Image Input Size

We evaluated the performance of the model under two different image input sizes for Scratch detection. This evaluation did not show a major improvement and, therefore, we did not construct a performance comparison for all damage categories. Further research could incorporate the effect of the image input size on the overall mAP performance.

Evaluation Measure

We used multiple evaluation measures to construct a well-founded performance comparison. We used the same loss function for each model to ensure that the loss values can be compared between the models. Despite this, a large performance difference exists between the different damage classes in terms of the mAP. We noticed that this is partly due to the level of difficulty and partly due to the non-triviality of the damages. Scratches seem to be difficult to annotate, while damages such as Glass Shatter or Hail are relatively easy to assign. Therefore, we expect that the model is performing significantly lower on Scratches than on Hail and Glass Shatter. By this, comparing the different types of damage on the same mAP scale might be an unfair performance evaluation. We recommend exploring other methods to either overcome the non-trivial annotations for some classes, or to incorporate this unfair comparison within the performance measure during further research.

Employee Performance

We used a subset of 100 images from the evaluation set of the *Damage Dossiers* to compare the performance of FSSD Darknet-53 with the performance of employees within Pon Logistics. The small dataset made it hard to compare the performance in detail. The confusion matrix was relatively sparse and contained only a few samples for some classes. We made use of this small dataset, to limit the effort and time required from the employees. Further research could try to make a more solid comparison by increasing the number of images, annotated by employees with domain knowledge. Furthermore, increasing the number of employees creates a more reliable estimation.

Light Street Performance

Although we used a relatively high-resolution camera in the Light Street, the distance between the camera and vehicle made the resolution of the captured vehicle low. In most cases, we were not able to detect the damage itself from the captured video stream, due to a combination of the small damage sizes and the low resolution. Due to the low number of cameras, we had to place the cameras relatively far away from the vehicle to capture it in full. This resulted in a low resolution as the vehicle is only located on a small part of the image. Further research might use higher camera resolutions, or use more cameras to make each camera capture a smaller proportion of the car. Using this, the overall resolution is increased. Due to the low number of detected damages in the Light Street, we were not able to train a model specific for this environment. Fine-tuning the model on the Light Street data might improve its performance, therefore, collecting more training data is required before further improvements can be made. Using this, we expect that the model will be more robust against the light reflections.

Bibliography

- Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., ... Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Retrieved May 28, 2019, from <http://tensorflow.org/>
- Aggarwal, C. (2018, September 13). *Neural networks and deep learning*. Springer-Verlag GmbH. ISBN: 3319944622. Retrieved from https://www.ebook.de/de/product/33161337/charu_c_aggarwal_neural_networks_and_deep_learning.html
- Atwood, J. (2007). Better image resizing. Retrieved March 6, 2019, from <https://blog.codinghorror.com/better-image-resizing/>
- Bishop, C. (2006, August 17). *Pattern recognition and machine learning*. Springer-Verlag New York Inc. ISBN: 0387310738. Retrieved from https://www.ebook.de/de/product/5324937/christopher_m_bishop_pattern_recognition_and_machine_learning.html
- Brinker, T. J., Hekler, A., Utikal, J. S., Grabe, N., Schadendorf, D., Klode, J., ... von Kalle, C. (2018). Skin cancer classification using convolutional neural networks: Systematic review. *Journal of Medical Internet Research*, 20(10), e11936. doi:10.2196/11936
- Cha, Y., Chen, J., & Büyüköztürk, O. (2017). Output-only computer vision based damage detection using phase-based optical flow and unscented kalman filters. *Engineering Structures*, 132, 300–313. doi:10.1016/j.engstruct.2016.11.038
- Chen, L., Zhu, Y., Papandreou, G., Schroff, F., & Hartwig, A. (2018). Encoder-decoder with atrous separable convolution for semantic image segmentation. In *The european conference on computer vision (eccv)* (Vol. 11211, pp. 833–851). doi:10.1007/978-3-030-01234-2_49
- Chollet, F. et al. (2015). Keras (Version 2.2.4). Retrieved May 28, 2019, from <https://keras.io>
- De Deijn, J. (2018). *Automatic car damage recognition using convolutional neural networks* (Master's thesis). Retrieved from <https://www.semanticscholar.org/paper/Automatic-Car-Damage-Recognition-using-Neural-Deijn/48f258246f7248a81458ddb14d9214e33759d7d7>
- Deshpande, A. (2019). A beginner's guide to understanding convolutional neural networks part 2. Retrieved February 12, 2019, from <https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/>
- Duffner, S. (2009). *Face image analysis with convolutional neural networks*. Munich: GRIN Publishing. ISBN: 3640397169.
- Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., & Zisserman, A. (n.d.). The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results.
- Everingham, M., Van Gool, L., Williams, C., Winn, J., & Zisserman, A. (2009). The pascal visual object classes (VOC) challenge. *International Journal of Computer Vision*, 88(2), 303–338. doi:10.1007/s11263-009-0275-4
- Flach, P. (2012). *Machine learning*. Cambridge: Cambridge University Press. ISBN: 1107422221.

- Fu, C.-Y., Liu, W., Ranga, A., Tyagi, A., & Berg, A. (2017). Dssd : Deconvolutional single shot detector. *CoRR*, *abs/1701.06659*. arXiv: 1701.06659. Retrieved from <https://www.semanticscholar.org/paper/DSSD-%3A-Deconvolutional-Single-Shot-Detector-Fu-Liu/cc3bd45efe9a6db3c4dc06c83ab2d63605696fc4>
- Gandhi, R. (2018). R-cnn, fast r-cnn, faster r-cnn, yolo - object detection algorithms. Retrieved June 11, 2019, from <https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e>
- Girshick, R. (2015). Fast r-CNN. In *2015 IEEE international conference on computer vision (ICCV)* (pp. 1440–1448). doi:10.1109/iccv.2015.169
- Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *2014 IEEE conference on computer vision and pattern recognition* (pp. 580–587). doi:10.1109/cvpr.2014.81
- Godoy, D. (2018). Understanding binary cross-entropy / log loss: A visual explanation. Retrieved June 20, 2019, from <https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a>
- Griffin, G., Holub, A., & Perona, P. (2007). *Caltech-256 object category dataset* (tech. rep. No. 7694). Retrieved from <https://core.ac.uk/download/pdf/4875878.pdf>
- Gulli, A. & Pal, S. (2017, April 28). *Deep learning with keras*. Packt Publishing. ISBN: 1787128423. Retrieved from https://www.ebook.de/de/product/29085781/antonio_gulli_sujit_pal_deep_learning_with_keras.html
- He, K., Gkioxari, G., Dollar, P., & Girshick, R. (2017). Mask R-CNN. *CoRR*, *abs/1703.06870*. arXiv: 1703.06870. Retrieved from <http://arxiv.org/abs/1703.06870>
- He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *2016 IEEE conference on computer vision and pattern recognition (CVPR)*. doi:10.1109/cvpr.2016.90
- Jayawardena, S. (2013). *Image based automatic vehicle damage detection* (Doctoral dissertation, College of Engineering and Computer Science (CECS)). Retrieved from <https://openresearch-repository.anu.edu.au/handle/1885/11072>
- Karpathy, A. (2017). Convolutional neural networks for visual recognition. Retrieved February 18, 2019, from <http://cs231n.github.io/neural-networks-3/>
- Kathuria, A. (2018). What's new in yolo v3? Retrieved July 3, 2019, from <https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b>
- Krizhevsky, A., Sutskever, I., & Hinton, G. (2012). ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, *60*(6), 84–90. doi:10.1145/3065386
- LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation*, *1*(4), 541–551. doi:10.1162/neco.1989.1.4.541
- Li, P., Shen, B., & Dong, W. (2018). An anti-fraud system for car insurance claim based on visual evidence. *CoRR*. Retrieved from <http://arxiv.org/abs/1804.11207>
- Li, Z. & Zhou, F. (2017). Fssd: Feature fusion single shot multibox detector. Retrieved from https://www.researchgate.net/publication/321511662_FSSD_Feature_Fusion_Single_Shot_Multibox_Detector
- Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., ... Zitnick, C. L. (2014). Microsoft COCO: Common objects in context. In *Computer vision – ECCV 2014* (pp. 740–755). doi:10.1007/978-3-319-10602-1_48
- Liu, S., Huang, D., & Wang, Y. (2018). Receptive field block net for accurate and fast object detection, 404–419. doi:10.1007/978-3-030-01252-6_24
- Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., & Berg, A. (2015). Ssd: Single shot multibox detector. *CoRR*, *abs/1512.02325*. doi:10.1007/978-3-319-46448-0_2. eprint: 1512.02325

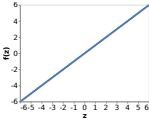
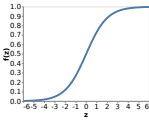
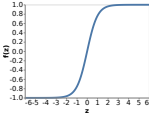
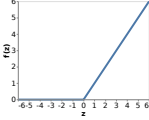
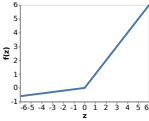
- Martin, S. (2019). What is transfer learning? [Blog Post]. Retrieved from <https://blogs.nvidia.com/blog/2019/02/07/what-is-transfer-learning/>
- Michelucci, U. (2018). *Applied deep learning*. Apress. ISBN: 9781484237908.
- Minsky, M. & Papert, S. (1969). *Perceptrons: An introduction to computational geometry*. Cambridge, USA: MIT Press. ISBN: 9780262130431.
- Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., ... Lerer, A. (2017). Automatic differentiation in pytorch. Retrieved from <https://openreview.net/pdf?id=BJJsrnfCZ>
- Patil, K., Kulkarni, M., Sriraman, A., & Karande, S. (2017). Deep learning based car damage classification, 50–54. doi:10.1109/icmla.2017.0-179
- Pattanayak, S. (2017). *Pro deep learning with tensorflow*. New York: Apress. ISBN: 1484230957. Retrieved from https://www.ebook.de/de/product/29894588/santanu_pattanayak_pro_deep_learning_with_tensorflow.html
- Perera, A. (2018). What is padding in convolutional neural network's(cnn's) padding (multi-class image classification step by step guide part 4). Retrieved March 6, 2019, from <https://medium.com/@ayeshmanthaperera/what-is-padding-in-cnns-71b21fb0dd7>
- Pont-Tuset, J. & Gool, L. V. (n.d.). Boosting object proposals: From pascal to coco, 1546–1554. doi:10.1109/ICCV.2015.181
- Pu, Y., Apel, D. B., Szmigiel, A., & Chen, J. (2019). Image recognition of coal and coal gangue using a convolutional neural network and transfer learning. *Energies*, 12(9), 1–11. doi:10.3390/en12091735
- Ragab, D. A., Sharkas, M., Marshall, S., & Ren, J. (2019). Breast cancer detection using deep convolutional neural networks and support vector machines. *PeerJ*, 7, e6201. doi:10.7717/peerj.6201
- R-CNN, Fast R-CNN, and Faster R-CNN Basics. (2019). Retrieved May 20, 2019, from https://nl.mathworks.com/help/vision/ug/faster-r-cnn-basics.html#mw_a9cdd2b3-b910-4d3d-90db-b485b415fd9b
- Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection, 779–788. doi:10.1109/cvpr.2016.91
- Redmon, J. & Farhadi, A. (2017). YOLO9000: Better, faster, stronger, 6517–6525. doi:10.1109/cvpr.2017.690
- Redmon, J. & Farhadi, A. (2018). Yolov3: An incremental improvement. *CoRR*, abs/1804.02767. eprint: 1804.02767. Retrieved from <http://arxiv.org/abs/1804.02767>
- Ren, S., He, K., Girshick, R., & Sun, J. (2017). Faster r-CNN: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6), 1137–1149. doi:10.1109/tpami.2016.2577031
- Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6), 386–408. doi:10.1037/h0042519
- Rosinski, W. (2017). Deep learning frameworks speed comparison. Retrieved June 28, 2019, from <https://wrosinski.github.io/deep-learning-frameworks/>
- Ruder, S. (2018). An overview of gradient descent optimization algorithms. Retrieved June 18, 2019, from <http://ruder.io/optimizing-gradient-descent/index.html#rmsprop>
- Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Parallel distributed processing: Explorations in the microstructure of cognition. In D. E. Rumelhart, J. L. McClelland, & C. PDP Research Group (Eds.), (Chap. Learning Internal Representations by Error Propagation, Vol. 1, pp. 318–362). Cambridge, MA, USA: MIT Press. ISBN: 0-262-68053-X. Retrieved from <http://dl.acm.org/citation.cfm?id=104279.104293>

- Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L. (2018). MobileNetV2: Inverted residuals and linear bottlenecks, 4510–4520. doi:10.1109/cvpr.2018.00474
- Shihavuddin, A., Chen, X., Fedorov, V., Christensen, A., Riis, N., Branner, K., ... Paulsen, R. (2019). Wind turbine surface damage detection by deep learning aided drone inspection analysis. *Energies*, 12(4), 1–15. doi:10.3390/en12040676
- Siddharth, D. (2017). Ccn architectures: Lenet, alexnet, vgg, googlenet, resnet and more... Retrieved March 20, 2019, from <https://medium.com/@sidereal/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5>
- Simonyan, K. & Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556. Retrieved from <https://www.semanticscholar.org/paper/Very-Deep-Convolutional-Networks-for-Large-Scale-Simonyan-Zisserman/061356704ec86334dbbc073985375fe13cd39088>
- Sokolova, M. & Lapalme, G. (2009). A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, 45(4), 427–437. doi:10.1016/j.ipm.2009.03.002
- Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... Rabinovich, A. (2015). Going deeper with convolutions. doi:10.1109/cvpr.2015.7298594
- Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the inception architecture for computer vision. doi:10.1109/cvpr.2016.308
- Tsang, S. (2018). Review: Dssd - deconvolutional single shot detector (object detection). Retrieved April 5, 2019, from <https://towardsdatascience.com/review-dssd-deconvolutional-single-shot-detector-object-detection-d4821a2bbeb5>
- Witten, I., Eibe, F., Hall, M., & Pal, C. (2017). *Data mining: Practical machine learning tools and techniques* (4th ed.). Cambridge, USA: Morgan Kaufmann Publishers. ISBN: 0128042915. Retrieved from https://www.ebook.de/de/product/26440029/ian_witten_eibe_frank_mark_a_hall_christopher_j_pal_data_mining.html
- Zhang, Q., Fu, H., & Qiu, G. (2013). Tree partition voting min-hash for partial duplicate image discovery, 1–6. doi:10.1109/icme.2013.6607460
- Zhuang, S., Wang, P., Jiang, B., Wang, G., & Wang, C. (2019). A single shot framework with multi-scale feature fusion for geospatial object detection. *Remote Sensing*, 11(5), 1–20. doi:10.3390/rs11050594

Appendix A

Activation Functions

TABLE A.1: Activation functions.

Name	Plot	Function	Derivative
Identity		$\phi(z) = I(z) = z$	$\phi'(z) = 1$
Sigmoid		$\phi(z) = \sigma(z) = \frac{1}{1+e^{-z}}$	$\phi'(z) = \sigma(z)(1 - \sigma(z))$
TanH		$\phi(z) = \text{TanH}(z) = \frac{e^{2z}-1}{e^{2z}+1}$	$\phi'(z) = 1 - \text{TanH}^2(z)$
ReLU		$\phi(z) = \text{ReLU}(z) = \max(0, z)$	$\phi'(z) = \begin{cases} 1 & z > 0 \\ 0 & z \leq 0 \end{cases}$
L-ReLU		$\phi(z, \alpha) = \text{LReLU}(z, \alpha) = \begin{cases} \alpha z & z > 0 \\ z & z \leq 0 \end{cases}$	$\phi'(z, \alpha) = \begin{cases} -\alpha & z < 0 \\ 1 & z \geq 0 \end{cases}$

Appendix B

Model Architectures

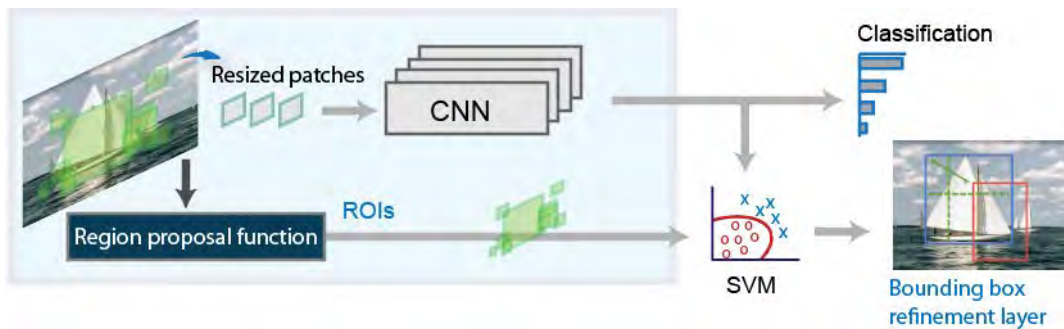


FIGURE B.1: R-CNN (“R-CNN, Fast R-CNN, and Faster R-CNN Basics”, 2019).

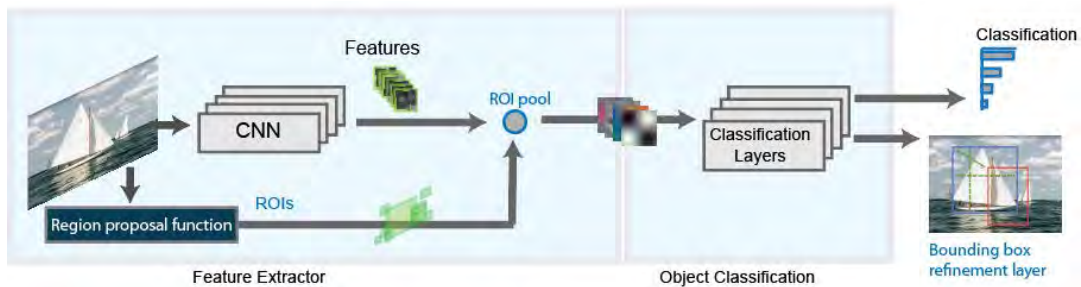


FIGURE B.2: Fast R-CNN (“R-CNN, Fast R-CNN, and Faster R-CNN Basics”, 2019).

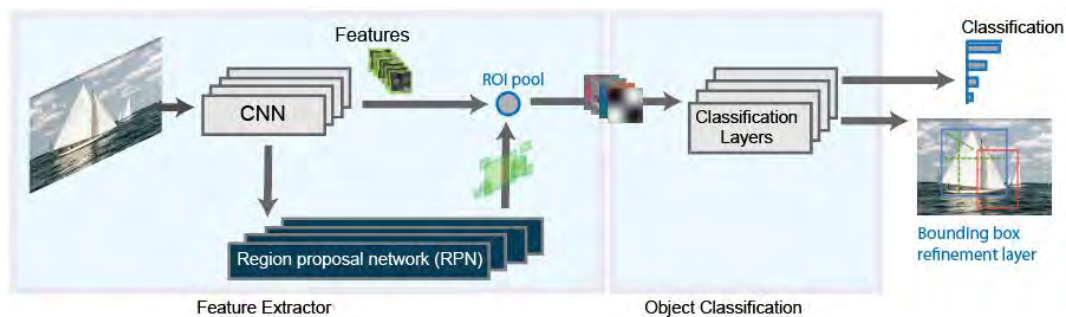


FIGURE B.3: Faster R-CNN (“R-CNN, Fast R-CNN, and Faster R-CNN Basics”, 2019).

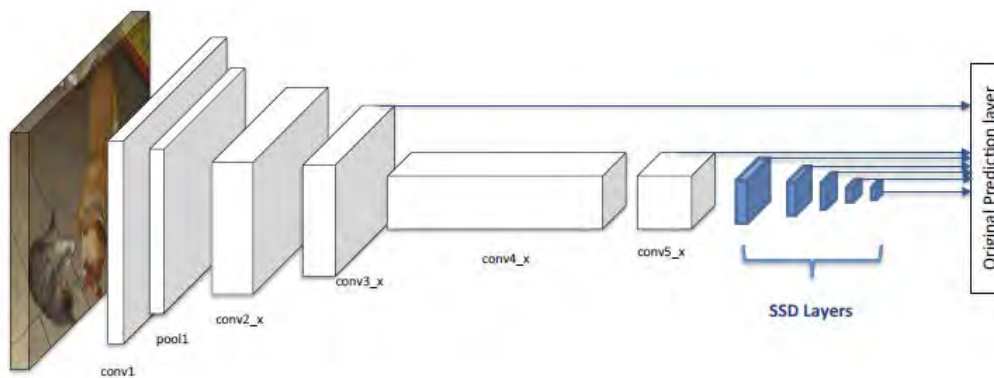


FIGURE B.4: SSD (Tsang, 2018).

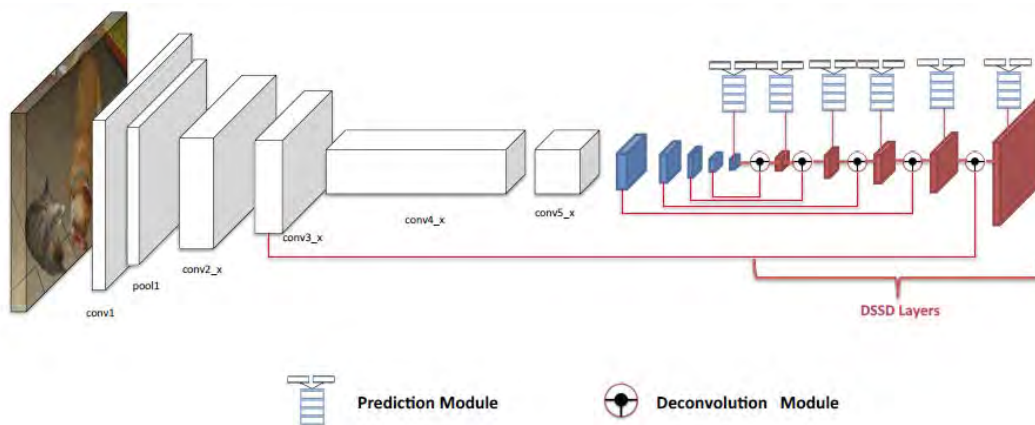


FIGURE B.5: DSSD (Tsang, 2018).

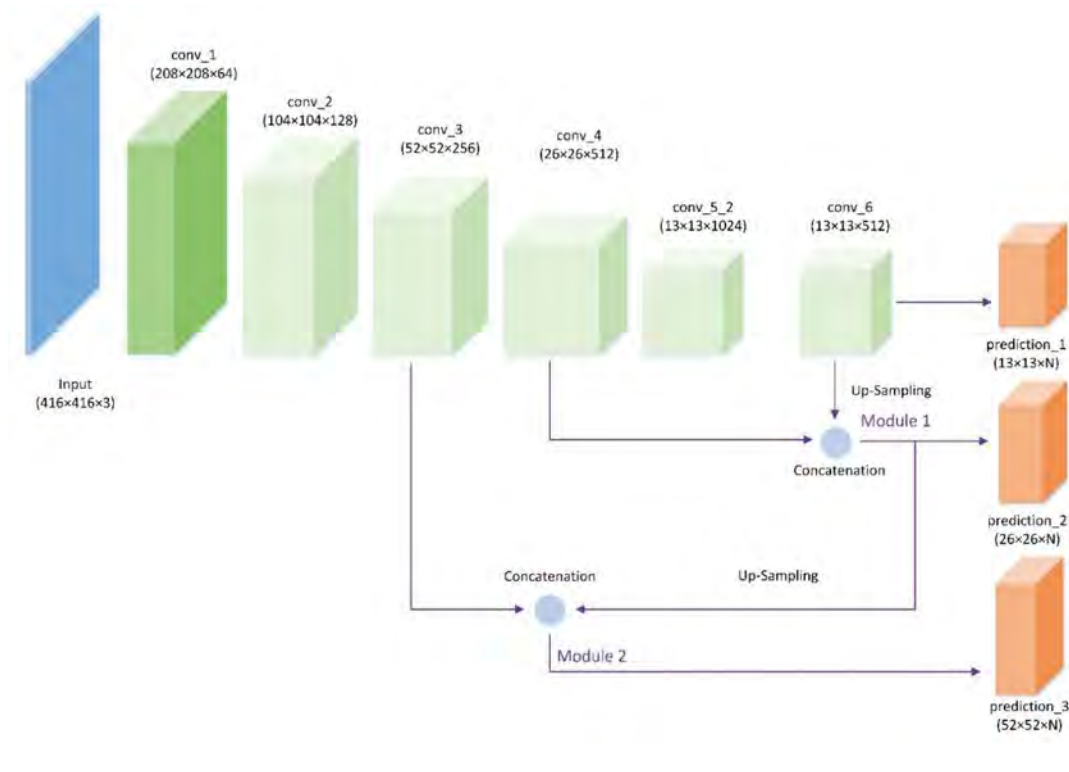


FIGURE B.6: FSSD (Zhuang, Wang, Jiang, Wang, & Wang, 2019).

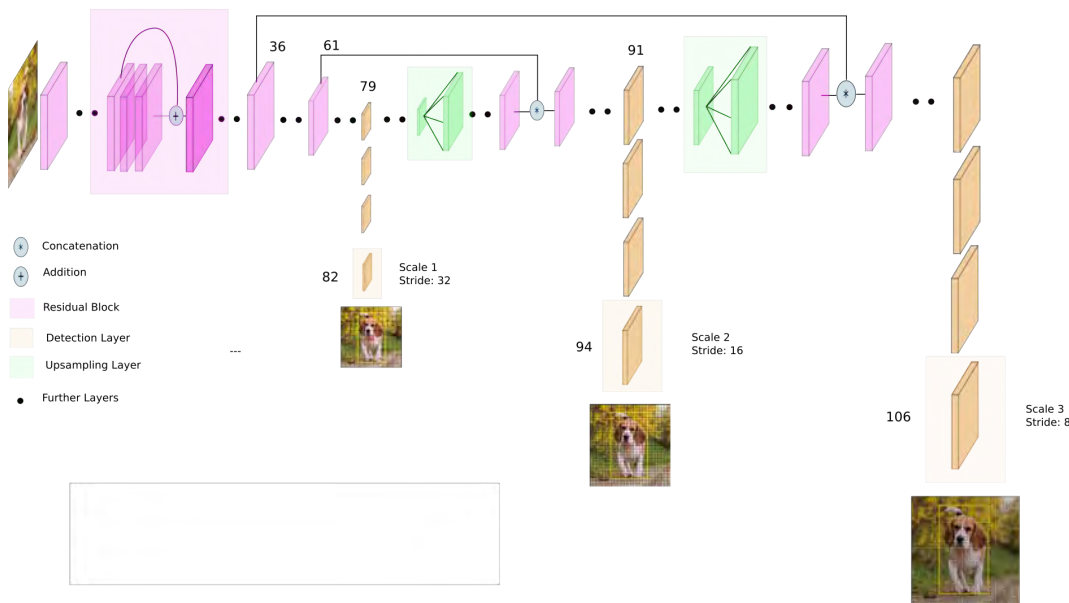


FIGURE B.7: YOLO v3 (Kathuria, 2018).

Appendix C

Web Image Extraction

TABLE C.1: Image search queries.

Category	Query	Image limit
Dent	Car dent	400
Dent	Car dent front	200
Dent	Car dent back	200
Dent	Car dent rear	200
Dent	Car dent bumper	200
Dent	Car dent side	200
Dent	Car dent door	200
Dent	Car dent crease	200
Dent	Car dent small	200
Dent	Car dent big	200
Dent	Car dent roof	100
Hail	Car Hail	200
Window	Car window crack	200
Window	Car windshield crack	200
Window	Car window broken	300
Window	Car window damage	100
Scratch	Car scratch	100
Scratch	Car scratch	100
Scratch	Car scratch minor	100
Scratch	Car scratch small	100
Scratch	Car damage scratch	100
Scratch	Car roof scratch	100
Scratch	Car key scratch	100
Scratch	Auto lakschade (Dutch search)	100
Scratch	Auto kras (Dutch search)	100
General	Car minor damage	200
General	Car small damage	200
General	Car damage rear	100

Appendix D

Excerpts from the Damage Datasets

D.1 Damage Dossiers

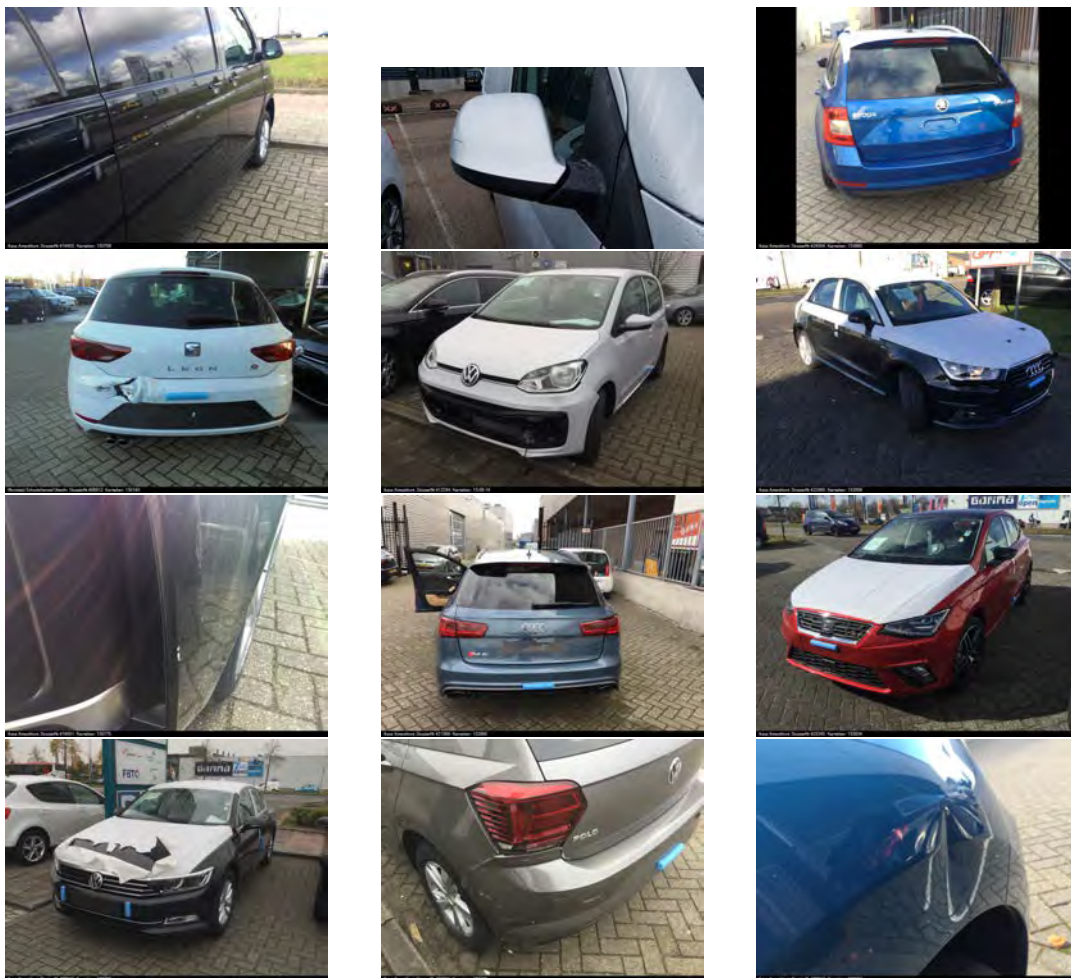


FIGURE D.1: Excerpts from the Damage Dossiers dataset.

D.2 Damage Web

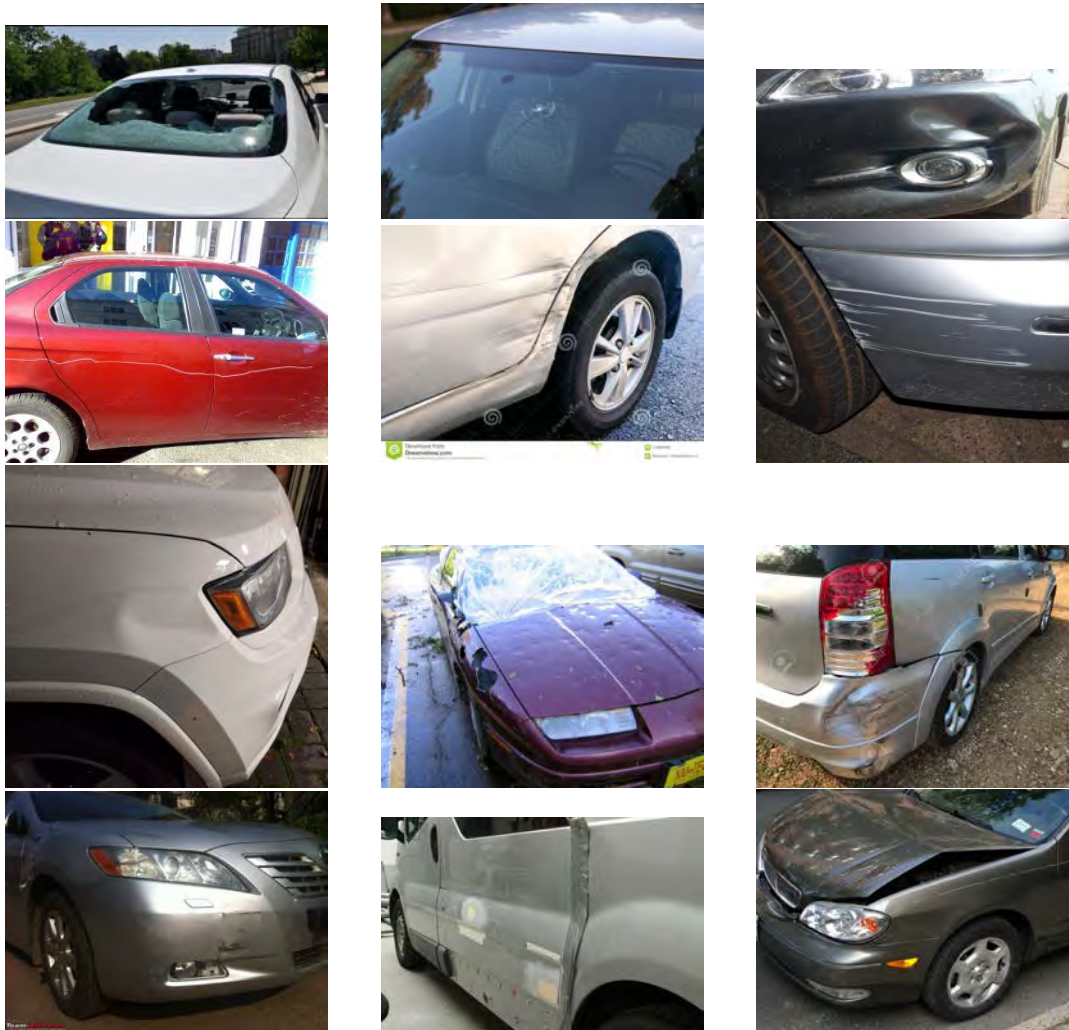


FIGURE D.2: Excerpts from the Damage Web dataset.

Appendix E

Annotations

E.1 Annotation Classes

1. **Bent:** A bending in vehicle components or sheet metal transition.
2. **Builen:** A dent to the outside of the sheet metal instead of inside.
3. **Cover Damage:** Torn or partly removed covering foil.
4. **Crack:** Sheet metal torn in two parts.
5. **Dent:** Inward bending of sheet metal.
6. **Glass Shatter:** Broken glass or cracks in the glass.
7. **Hail:** Damage due to hail.
8. **Light Broken:** Broken glass of either the front or rear light.
9. **Missing:** A part or component of the vehicle not being present.
10. **Rust:** Rust on the vehicle metal.
11. **Scratch:** Damage on the car paint.
12. **Tire Crack:** Crack in the surface of the tire.

E.2 Bounding Box Dimensions

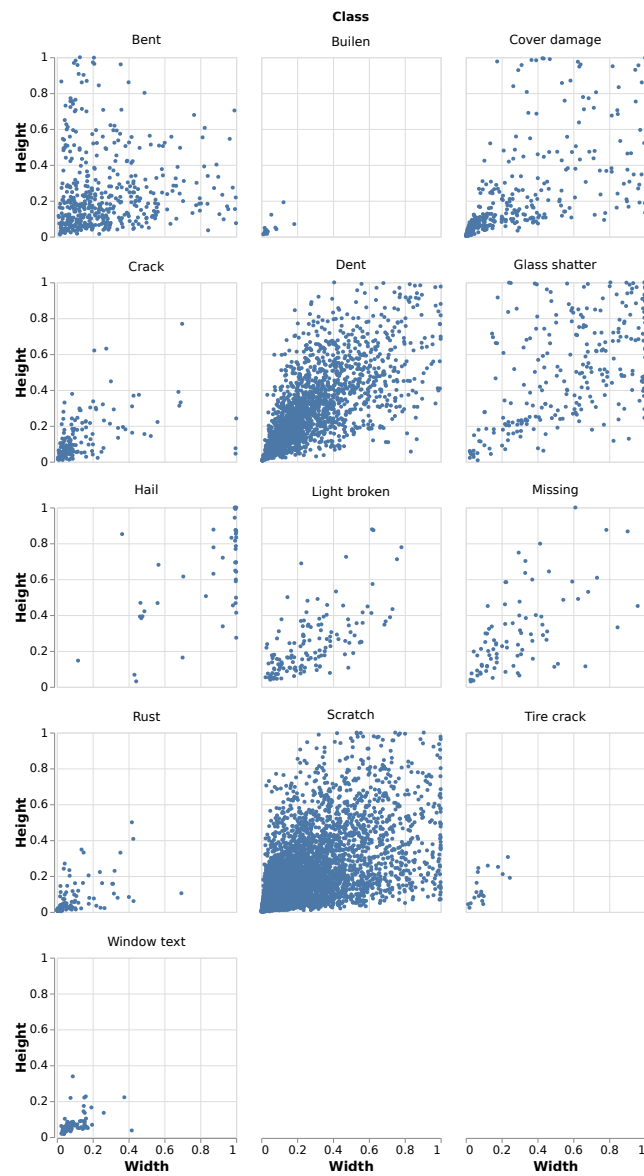


FIGURE E.1: Bounding box dimensions per class.

Appendix F

Excerpts from the Augmented Data

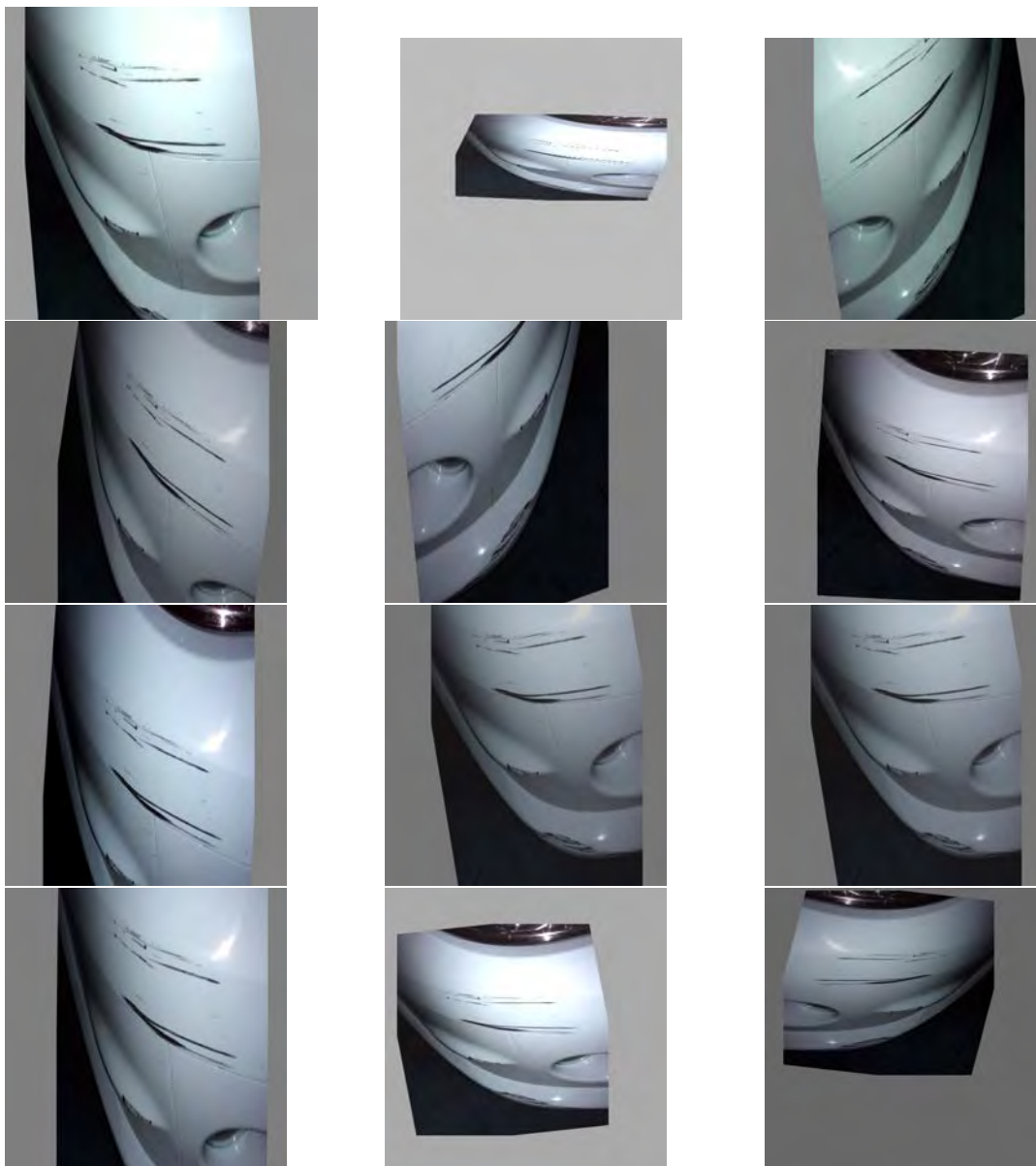


FIGURE F.1: Excerpts from the augmented data with: cropping, expansion, rotation, and multiply.

Appendix G

Anchor Priors

TABLE G.1: Anchor prior comparison across datasets for YOLO v3. COCO as reported by Redmon and Farhadi (2018), PASCAL VOC by K-means clustering as no reliable source was found. Damage Web, Damage Dossiers, and Damage as constructed by K-means clustering.

Scale	COCO	PASCAL VOC	Damage Web	Damage Dossiers	Damage
I	[0.024, 0.031]	[0.050, 0.071]	[0.051, 0.045]	[0.042, 0.040]	[0.047, 0.043]
	[0.038, 0.072]	[0.095, 0.175]	[0.159, 0.143]	[0.226, 0.116]	[0.110, 0.174]
	[0.079, 0.055]	[0.142, 0.385]	[0.386, 0.178]	[0.101, 0.154]	[0.237, 0.116]
II	[0.072, 0.146]	[0.242, 0.596]	[0.747, 0.226]	[0.221, 0.355]	[0.240, 0.369]
	[0.146, 0.108]	[0.254, 0.202]	[0.224, 0.355]	[0.837, 0.301]	[0.536, 0.478]
	[0.141, 0.286]	[0.356, 0.375]	[0.493, 0.480]	[0.450, 0.182]	[0.311, 0.716]
III	[0.278, 0.216]	[0.446, 0.681]	[0.291, 0.705]	[0.525, 0.458]	[0.536, 0.478]
	[0.375, 0.475]	[0.679, 0.402]	[0.838, 0.557]	[0.307, 0.710]	[0.311, 0.716]
	[0.896, 0.783]	[0.748, 0.748]	[0.786, 0.853]	[0.787, 0.793]	[0.809, 0.786]

TABLE G.2: Anchor prior comparison for SSD Liu et al. (2015), FSSD Li and Zhou (2017), and RFB-SSD Liu, Huang, and Wang (2018), based on the authors reported default anchors. All models are using aspect ratio: $[\frac{1}{3}, \frac{1}{2}, 1, 2, 3]$.

SSD	FSSD	RFB-SSD
[16, 16]	[8, 8]	[16, 16]
[32, 32]	[16, 16]	[32, 32]
[64, 64]	[32, 32]	[64, 64]
[100, 100]	[64, 64]	[100, 100]
[150, 150]	[100, 100]	[150, 150]
[300, 300]	[300, 300]	[300, 300]

Appendix H

Excerpts from the Predictions

H.1 Damage Web

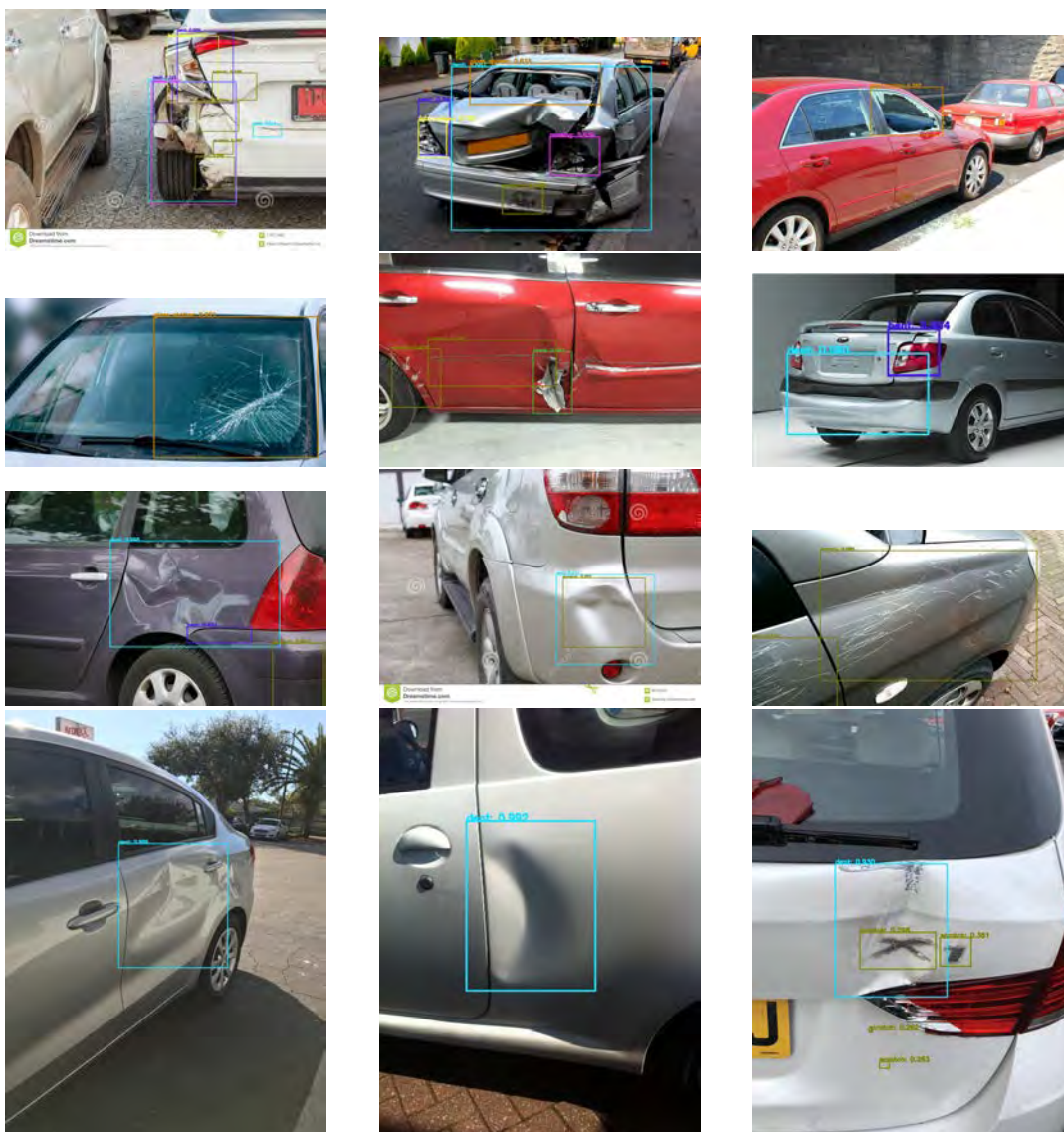


FIGURE H.1: Excerpts from the predictions on the Damage Web data with YOLO v3 and Darknet-53.

H.2 Damage Dossiers



FIGURE H.2: Excerpts from the predictions on the Damage dossiers data with FSSD and Darknet-53.