Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of ICT and Natural Sciences

Basir Sedighi

Deep learning for fault detection of guardrails

Master's thesis in Simulation and Visualization
Supervisor: Ottar L. Osen, Robin T. Bye
June 2020

NTNU
Norwegian University of Science and Technology

Basir Sedighi

Deep learning for fault detection of guardrails

Master's thesis in Simulation and Visualization
Supervisor: Ottar L. Osen, Robin T. Bye
June 2020

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of ICT and Natural Sciences

NTNU
Norwegian University of
Science and Technology

Summary

Humans are immeasurably good at extracting information from images in complex scenery to detect and classify an object. In recent years, algorithms and methods have been presented to do the same. Advanced algorithms are used for complex tasks, famously in areas related to self-driving cars, tracking, classifying, etc. The field in machine learning called computer vision heads out to extract the vast information present in images. The exponential advances in numbers of computing units in GPUs (Graphical Computing Units), have made it possible to create algorithms that were unimaginable a decade ago without supercomputers.

Recent years have seen an increase in neural network for solving a variety of tasks, where the Convolutional Neural Network is known for its performance on image processing. Also, resourceful IT-companies and research faculties having the advantage of available computational power, have contributed with state-of-the-art, costume tailored CNN models for a variety of computer vision tasks. This thesis investigated the state-of-the-art CNN models to aid skilled workers in maintaining guardrails across Norway to automize the visual inspection done by these workers. Today's visual inspection is done by driving in speeds between 1-15 km/h and performing visual inspection through the camera or the car window to detect the faults.

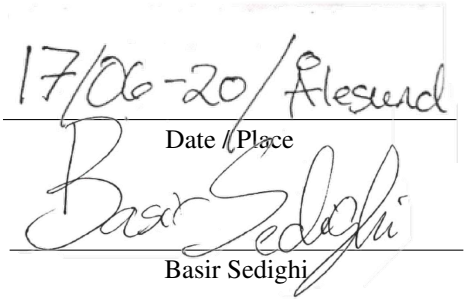
This work sets out to implement a state-of-the-art architecture, which is chosen by evaluating a variety of architectures according to the objectives set by the thesis. The experimentation was done by collecting data, pre-processing, and implementing the neural network. The model achieved a promising $mAP_{IOU=50}$ of 71%.

Preface

This report is a result of the master's thesis given at the Norwegian University of Science and Technology (NTNU), autumn 2020. The report is part of the Master's program in Simulation and Visualization at the Department of ICT and Engineering.

The Master's thesis is weighted to 30 out of 120 credits. I was assigned the project by Arvid Gjerde AS in collaboration with iSi AS. The purpose of this report is to give the reader an insight into how machine learning model was implemented and modified for detection of faults in guardrails.

This report is addressed to Arvid Gjerde AS, iSi AS, NTNU and others who are interested in the topics covered by the report. I want to thank local supervisor Ottar L. Osen and Robin T. Bye of NTNU and external supervisors Nils Tarjei Hjelme and Bård Indredavik of iSi. In addition, I would like to thank the employees of the at Arvid Gjerde for the construction of the dataset. This came in very handy.



Date / Place

Basir Sedighi

Table of Contents

List of Tables

List of Figures

Abbreviations

AI	=	Artificial Intelligence
AP	=	Average Precision
CNN	=	Convolutional Neural Network
COCO	=	COmmon object in COntext
DL	=	Deep Learning
GPU	=	Graphical Processing Units
HOG	=	Histogram of Oriented Gradients
IOU	=	Intersection Over Union
IT	=	Information Technologies
mAP	=	mean Average Precision
ML	=	Machine Learning
ResNet	=	Residual neural Network
ROI	=	Region Of Interest
RPN	=	Region Proposal Network
SSD	=	Single Shot MultiBox
TP	=	True Positive
FP	=	False Positive
VOC	=	Visual Objects Classes
YOLO	=	You Only Look Once algorithm

Chapter 1

Introduction

One of the most recognizable cars in Norway to date are the ones produced by Tesla. They became well known for being the first company to develop premium sports cars that weren't in anyway reminiscent of the two-seat small electric cars of that time. In later years they got big recognition for the technology provided in their automobiles, especially the autopilot which could make the car drive itself from place to place [72]. How is it possible for a car to drive itself?

One could believe we could program sets of rules from the computer, like stay within the lines in the road or start driving when the lights turn green, but this is high level human thinking. Driving a car for a mere human requires a lot of information which mostly is provided from our sensory organs. Writing every rule for the computer to follow would be very time and resource consuming. What if instead of giving it sets of rules to follow, one could teach the computer to learn?, in other words, make it perform a specific task without explicitly being programmed for it.

Machine learning is an application of Artificial Intelligence (AI) that does specifically this. The aim of machine learning is to allow the computers to learn automatically without human intervention or assistance and adjust actions accordingly [52]. As an example, the machine learning algorithm take an input which could be pixels of an image or human measurements and predict if a certain object is in the picture or the human's Body Mass Index (BMI).

Machine learning is very demanding computationally, but in recent years the advances in Graphical Processing Units (GPU) has made it easier to run this model on a local machine [50]. Also, Google and other big companies provide free services like Google Collab to train a model on an virtual computer. The goal here is to utilize these services to introduce machine learning for road maintenance, for detecting faults in road fences.

1.1 Background

"*Improper auto protection may have contributed to the death of 18-year-olds*"
– Norwegian Broadcasting Channel (NRK), 2018 [57]

"*In Norway, there are at least 1745 railing on bridges with vulnerabilities, faults and deficiencies that can affect road safety.*"
– National Newspaper VG, 2018 [40]

The Norwegian Public Roads Administration is working on a number of road safety measures and campaigns against traffic accidents. The National Road Safety Action Plan for Road Safety 2018–2021 is a four-year plan for road safety work in Norway [46]. The plan is a collaboration between the Norwegian Public Roads Administration, the police, the Directorate of Health, the Directorate for Education, Trygg Trafikk, the county municipalities and seven metropolitan municipalities. In addition, a number of other public actors at the national level and about 20 interest organizations have contributed to the plan.

The plan contains 136 targeted measures. These will contribute to the Storting's target for 2030 of a maximum of 350 died or severely injured in traffic per year. In 2016, 791 people were died or seriously injured on Norwegian roads[4].

The operating contractors are currently responsible for checking the guardrails annually. The check seeks if any bolts are missing, any damage has been done, they are inclined and if they are in accordance with today's norm. There is good reason to believe that this either does not happen today, or that it is done in a superficial way. As of today, there is absolutely no requirement for competence on those who are going to install road fences in Norway! Unfortunately, this causes incorrect installation of some railings, wrong ends choices, or railings mounted in places where one could have safer side terrain, and sometimes even use of cheaper measures than railings.

The conclusion from Supervision Case 2018-19 is that "Missing bolts in railings are a road safety problem with high damage potential. The Norwegian Public Roads Administration's management system does not capture this problem, either in terms of risk assessment of objects, requirements for inspections, specifications of control activities, causal analysis or experience sharing/learning. Without system and practice changes, this problem will continue. "[76]

Apparently, there is some misconception of how the inspections of the road fences should be done. This has contributed to unsafe road railings in Norway, this is not the only contribution, but also the fact that Norway has four seasons. The plowing of the snow on the roads during the winter does a lot of wear and tear to the railings and can also make the bolts pop out from their sockets.

1.2 Problem description

During autumn, the client has been commissioned to do the maintenance, and has done so in the most efficient way possible, but, at the same time, in such a way that it is done in a qualitatively manner. The fences must be mounted correctly and not damaged in order to function properly. Today the client has driven 1 car with driver and controller at 1-15 km / h and done visual inspection through the window or with camera mounted on the car. They have also repaired any damage that has been affordable to repair on site. In addition, they have driven with a rear-seat cushion car to ensure safety.

The Norwegian Public Roads Administration has taken the problem seriously, and want all the contractors to identify fault in road fences and report them to their database. As of today, they don't really know many faults there are in the road fences, because the numbers that have been estimated are based on observations done in a fraction of the roads. However, assigning skilled workers to drive 1-15 km/h through Norwegian roads is neither scalable nor safe. Usually, it requires three people and two cars, and it must be done at night for busy roads. A company like Arvid Gjerde which is responsible for many of the roads in their area must dedicate a lot of resources to ensure they are safe by monitoring them manually. This sector is in need of innovation to make the identifying of faults cheaper and faster for their customers. This is important as the Norwegian government has a vision of no casualties in traffic accidents, but then a drastic change is needed in order to improve the identification of faults on the road fences. Also, it would be more attractable for the workers to work during daytime, be able to drive the same speed as the traffic and without being danger for other drivers.

1.3 Objectives

The method proposed should be a resource for helping the company to maintain the railing which they are responsible for. In the context of common fault on railings today, it is necessary to review computer vision theory, algorithms, and methods to propose a solution aiding the workers. Given that the system does not need to be realtime, the pursuit of a solution should be in terms of accuracy, and especially consider missing bolts, as it is the most common faults in railings. Aiding the company to collect data, and preparing the data according to standards used in the computer vision field. The chosen state-of-the-art model should be implemented and modified to the specific need for fault detection in railings. Experimentation is to be done on a costume dataset, and the result should be presented and discussed in a way, giving the company and the interested reader the insight needed to understand the limitations of using the proposed model. The methodology used in the thesis should be explained so that if one wishes to implement the proposed solution, the implementation can easily be modified for the specific needs of the individual or company. Finally, the resulting model from experimentation should be a starting point for the future projects the company has regarding fault detection through a camera.

1.4 Scope

This project is a part of a bigger project, where the outcome will have Industry grade GPS to remember the exact location within X-meters of where the picture was taken. Also, there will be a lidar in the result to find the angle of inclination of the road behind the fences. These are the requirements for the inspections and will not be touched upon this thesis. A new architecture will not be created either as it requires big resources and should be left for the big companies like Google and Facebook. Rather, in this thesis an implementation of the state-of-the-art architectures will be used and fitted to be able to detect faults on the road fences.

Chapter 2

Theory

2.1 Railing

In event of a road accident, the purpose of rails and cushions is to reduce the extent of damage to humans and material as much as possible. According to The Norwegian Public Roads Administration's manual [77], rails and cushions are expected to:

- Prevent dangerous side obstructions.
- Prevent exit on high and steep slopes, deep ditches, water, etc.
- Prevent collisions between oncoming vehicles.
- Protect road users and others who are on or near the road from vehicles on the road.
- Protect special facilities near the road, eg. rail, fuel tanks etc. against vehicles on the road.
- Prevent damage to road structures that can result in very serious consequential damage, like bridges.
- Prevent vehicles from falling off the road or rail, into a river that goes under the road.

Railings shall function so that when hit by a vehicle, it will guide the vehicle along the railing, until it stops, or lead the vehicle back to the roadway, but no longer than to avoid colliding with oncoming vehicles.

2.2 Maintenance

This section is a superficial analysis of the maintenance done on the railings today. In maintenance, there are two main subcategories. These are corrective and preventive

maintenance, however, this section will focus on corrective as the preventive maintenance is not relative for this thesis.

2.2.1 Corrective maintenance

The main difference between corrective maintenance and preventive maintenance is when the maintenance is done. Corrective maintenance can either be planned or unplanned depending on whether a maintenance plan has been created or not. But, nowadays, it is more associated with the unplanned as this is more interesting from a cost-analysis perspective. However, corrective maintenance is done after failure while preventive is done before failure. In mechanical engineering, this approach is the costly approach, because the cost is associated with danger to life in this project and not the physical cost of the railings. [62]

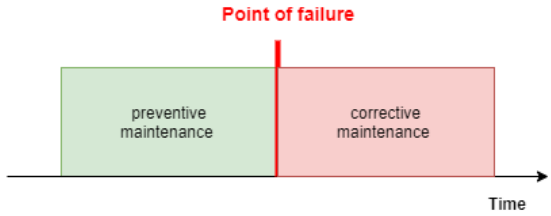


Figure 2.1: Preventive vs. corrective maintenance

These types of maintenance from mechanical engineering do not translate perfectly to our project, as in this thesis the cost is not the measurement which is wished to be reduced, but rather safety, as mentioned above. However, the mechanical instruments are usually in a plant, where its task is dependent on another instrument and so on, and when one of the instruments fails, the production of the entire plant stops. In the case of detecting faults in railings an additional time is added for detecting the fault. If a fault goes undetected the fault remains until the next control, which can take seasons to happen.

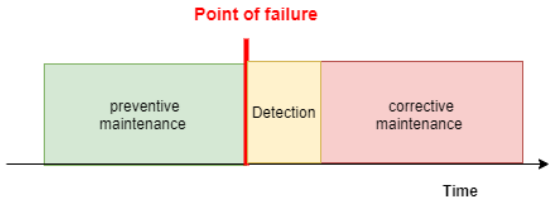


Figure 2.2: Preventive vs. corrective maintenance

2.3 Deep Learning

An algorithm is a set of rules to be followed when solving problems. In AI, Deep Learning (DL) and Machine Learning (ML), algorithms take in data and perform calculations to find an answer[47]. To understand deep learning, these terms need to be explained as it is a subset of machine learning and artificial intelligence, as depicted in the figure 2.3.

In mass media, these terms are used interchangeably but they are in fact not the same thing. AI is a broader concept than machine leraning and includes every algorithm which addresses the use of a computer to mimic the cognitive function of the human brain. Machine learning, on the other hand, includes algorithms that take sets of input data and through adaptive learning and automatically can learn the relationship between the input and output data. This is done through sets of transformations to the input data and then evaluating the transformations to further improve upon. Overtime its performance will improve.[22] [47]

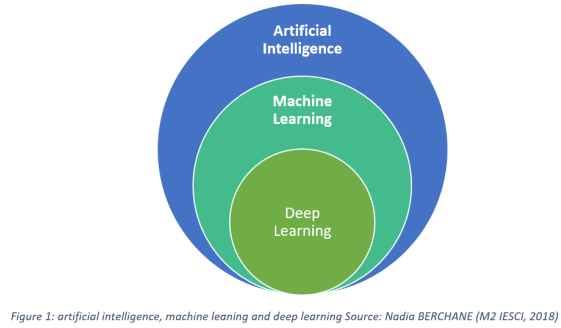


Figure 2.3: Subsets of Artificial Intelligence [53]

Deep learning functions and algorithms are similar to those in machine learning, but the difference between traditional machine learning and deep learning is the numerous layers of these algorithms. Each of these layers provides a different interpretation of the data it is fed on. In short, deep learning has several layers on processing which makes it be able to recognize more complex patterns. Deep learning is not a new concept, but its popularity can be summed up to two main reasons: The amount of data available and the increase of raw power in the graphics processing unit[35]. Creating deep learning models was, a decade ago, exclusively preserved to resourceful IT companies or universities that could afford expensive GPUs to do the calculations needed to make a successful model. However, today this can be done with a mid-tier GPU for a reasonable price at a students dorm. Also, the amount of data for an arbitrary problem is widely available. Consider a hospital doing X-rays to detect fractures, it would typical, some decades ago, to produce the images on a film, but, as for today, this would be done digitally, stored in a database and over the years a dataset could be created out of this for teaching a deep learning model to detect fractures.[20] [55]

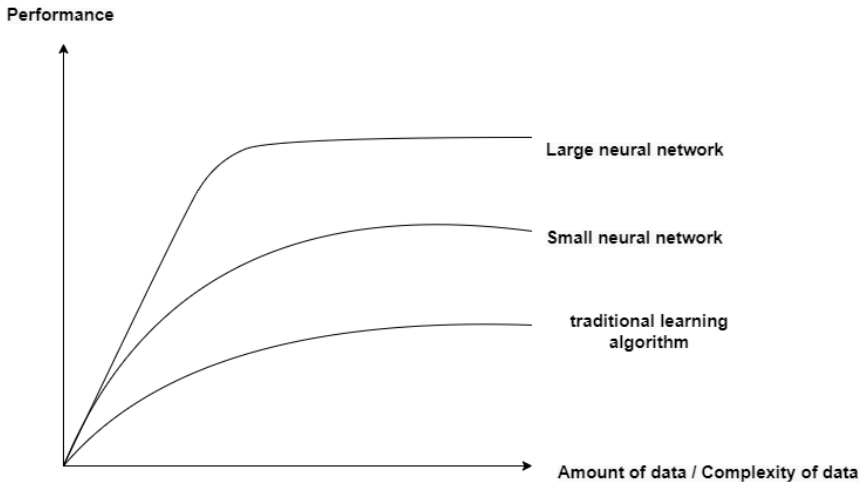


Figure 2.4: Advantages of deep learning [55]

One way products like Netflix and Amazon can make you spend more time/money on their websites is that they have millions of active users that leave a lot of data about their preferences, giving them the necessary data to make an accurate model that predicts what one as a costumer would spend money on and not. As depicted in the figure 2.4, the deep learning model will have high performance compared to other algorithms as the data is vastly available.[55]

A variety of tasks are solved by deep learning, from language processing in the form of virtual assistants on phones (Siri, Google Assistant), to playing games like Dota 2 or chess. The models are based on an artificial neural network which is processing units in conjunction, inspired by a biological neurals to do transformations to data. One very popular and standard ML algorithm which was used in this research project was the feed-forward neural network, and it will be explained in section 2.5.1.[55]

2.4 Categories of Machine Learning algorithms

2.4.1 Supervised learning

In supervised learning, the dataset is the collection of labelled samples $(x_i, y_i)^{(N)}$. Each element x_i among N is called a feature. A feature is a vector in which each dimension $j = 1, ..., D$. The label y_i can be either an element belonging to a finite set of classes $1, 2.....C$, or a real number. As an example $x^{(j=1)}$ could be the weight of a person, $x^{(j=2)}$ could be the height and $x^{(j=3)}$ could be muscle mass then y_i could belong to $C_i = \{overweight, normalweight, underweight\}$ or just a real number that indicated the individual's BMI. The goal of the supervised algorithm is to use the dataset to produce a set of transformations that takes the features x and y_i. [12]

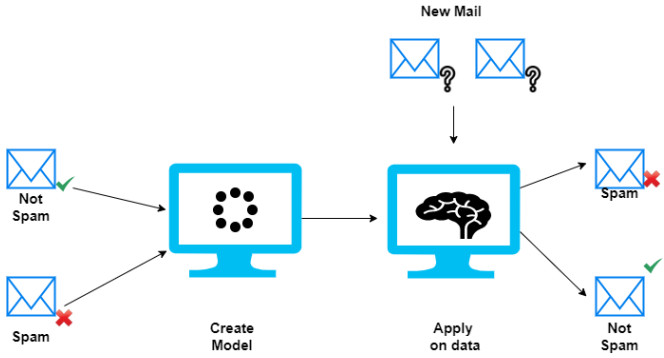


Figure 2.5: Supervised learning

2.4.2 Unsupervised learning

In unsupervised learning there is a collection of data just like supervised learning, however, now it is not needed to have the collection of label y_i. Meaning the dataset is a collection of unlabelled samples $\{x_i\}$. x_i is still the feature, and the goal of the algorithm is to create a model where x is transformed into another vector or a value that is useful for the current practical problem. This can, for example, be a clustering problem. After passing a feature vector x_i to the model, it will return which cluster the feature vector belongs to. This can also be used in dimensionality reduction where the model receives ${x_i}^N$ and it returns $\{x_i\}^M$ where $M < N$, in other words, it returns fewer features as the neglected features are unneeded for solving the problem. [12]

2.4.3 Reinforcement learning

Reinforcement learning have a long-term objective to maximize a numerical performance measure. In contrast to supervised learning, reinforcement learning "lives" in an environment and can understand the state of the environment. The goal is to learn a policy such

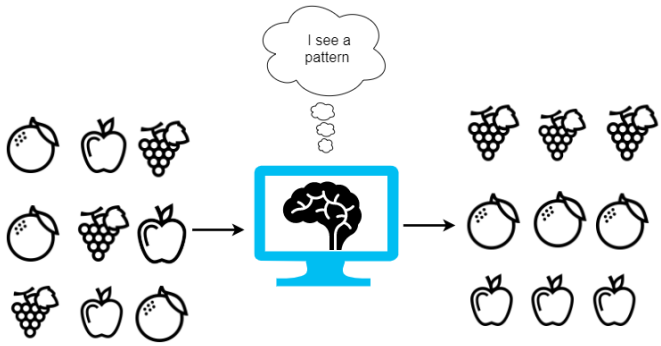


Figure 2.6: Unsupervised learning

that in every stage, the model decides an action that is optmal (maximizes the expected reward). [12][71]

2.5 Deep learning algorithms

2.5.1 Feedforward neural network

The feedforward neural network is the simplest of the different types of neural networks [12]. The information flows in one direction from input to output, through neurons (also called nodes). The nodes are simple processing units, which will transform sets of inputs into one output. The transformed signal from a neuron then becomes one of several input signals to the neurons in the next layer. Between the layers, the neurons have a weighted connection. The neurons get activated by the signal from the previous layer, except for the input layer. The output of a neuron is the sum of the weighted signal between the layers, passed through an activation function. Considering the highlighted section in figure 2.7, this will result in equation 2.1. [14]

$$y = \sigma(\sum W_i \times x_i - b) \tag{2.1}$$

x_1 are the inputs or the activation from previous layer, W_i are the weights between the layers, b_i is the associated bias for the neuron and the activation for the output of the neuron is in this case equals to y and σ is our activation function. This is the general equation for a neuron but in practice, a matrix representation is more suitable for computers. Many of the libraries using matrix operation for modern programming language is very well optimised for this, also, matrix operations are well suited for running calcu-

lations on the GPU as this component is specialised for this task. [56]

$$a^{(1)} = \sigma\left(\begin{bmatrix} w_{0,0} & w_{0,1} & x_{0,2} & \dots & w_{0,n} \\ w_{1,0} & w_{1,1} & w_{1,2} & \dots & w_{1,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{k,0} & w_{k,1} & w_{k,2} & \dots & w_{k,n} \end{bmatrix} * \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ \vdots \\ a_n^{(0)} \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \right) \tag{2.2}$$

$$a^{(1)} = \sigma(W * a^{(0)} + b) \tag{2.3}$$

In programming languages, object programming implementation of the neural network would look similar to equation 2.2. The same equation is applied over and over again until the signal reaches the output layer. The indexes n and k indicates nodes in their respective layers. $w_{k=1,n=2}$ is the weight between second node in current layer and first node in previous layer . Sigma is again our activation function and b is the bias. [56]

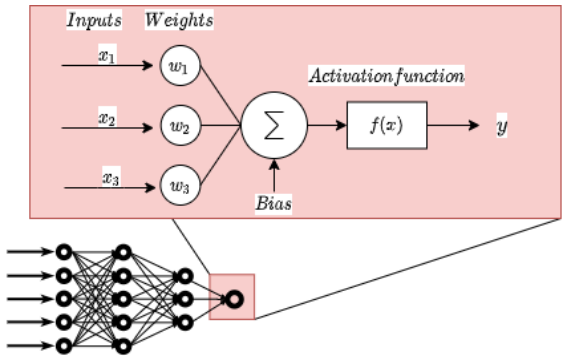
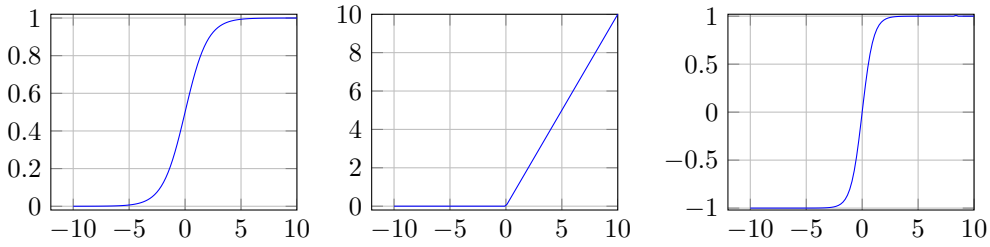


Figure 2.7: Neuron in a neural network



(a) *Sigmoid* **(b)** *Relu* **(c)** *Tanh*

Figure 2.8: Three activation functions

What is the purpose of the activation function?
The activation function defines the output of a neuron given a set of inputs. The neuron

in a neural network is inspired by the activity in the human brains, where a specific neuron is fired by the "right" stimuli[56]. Some of the most known activation functions, as depicted in figure 2.8, is the Sigmoid, Relu, and Tanh. The Sigmoid outputs 1 if the input is much larger than 1, and output 0 if the input is much less than 0. Another way of thinking about it is, it will squeeze the input x in a range between zero and one. The Tanh is very similar to the Sigmoid function, however, the range of the input will be transformed into a range between -1 and 1. Relu, on the other hand, is a bit more unique. The Relu function lets all values pass-through as long as it's bigger than 0 or else the neuron will output 0.

The main reason to use the Sigmoid function is that it is especially good for a model that seeks to predict probability as an output[66]. As the probability of anything exists between the range of 0 and 1, Sigmoid is the right choice. The Tanh or hyperbolic tangent activation function can be used for the classification of two classes[66]. Relu is the most widely used activation function, and that is for good reason. In the training of Imagnet dataset, which is an image database of 200 classes, Relu had almost 6x improvement in the training process compared to Tanh[37]. The equation for the activation functions are given by:

$$f_{sigmoid}(x) = \frac{1}{1 + e^{-x}} \tag{2.4}$$

$$f_{relu}(x) = max(0, x) \tag{2.5}$$

$$f_{tanh}(x) = \frac{e^{x} - e^{-z}}{e^{x} - e^{-z}} \tag{2.6}$$

The output of a neural network is the result of numerous activations and biases, however,it is necessary to train the neural network to do the right transformations to the data throughout all the layers. During the start of the training, there are some clever ways to initialize the weights and biases for faster training time, or for simplicity can be initialized randomly. After initializing both weights and biases, the network needs to know how precise the predictions are. This is done by defining a cost/loss function, simply being how far of the prediction was to the target or desired value. Then the process of finding the optimal weights is to minimize this loss function. Considering a neural network with one node in each layer, results in loss function in equation 2.7.[56]

$$C(w, b) = (y - a^{(L)})^2 \tag{2.7}$$

Using the gradient of the cost function, the network can find out how much each weight and bias is affecting the overall cost. Backpropagation is a technique where the model uses the chain rule to find the partial derivative of each weight and bias to calculate the

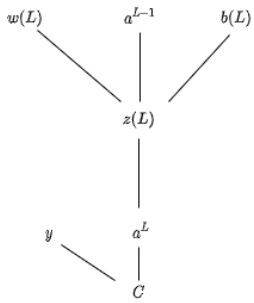


Figure 2.9: Neuron in a neural network

gradient or slope of the cost function, and then, take a "small step" down the slope. Sometimes, there is a way to explicitly derive the minimum of a function, however, this can be hard or impossible for functions that take as input thousands of parameters. Initializing the weights and biases, can be considered as starting in an arbitrary point in the nth dimensional surface and, calculating the slope downhill from the arbitrary point is, finding the gradient of the cost with respect of the weights and biases.[56] [55]

In the simple one node neural network in figure 2.9, it is shown some new terms like $z(l)$, and C. The term z indicated the value before activation and a is after the activation function has been applied to the input. In this network, using backpropagation calculates the weight $w(l)$ effect on the overall cost. One training example results in equation 2.8.

$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} + \frac{\partial a^{(L)}}{\partial z^{(L)}} + \frac{\partial C^{(L)}}{\partial a^{(L)}} \tag{2.8}$$

The equation is averaged across all training examples for each weight and bias to compute gradient vector ∇C, as shown in equation 2.9.

$$\nabla C(w) = \begin{bmatrix} \frac{\partial C}{\partial w^{(1)}} \\ \frac{\partial C}{\partial w^{(2)}} \\ \vdots \\ \frac{\partial C}{\partial w^{(L)}} \end{bmatrix}, where \frac{\partial C}{\partial w^{(L)}} = \sum_{k=0}^{n-1} \frac{\partial C_k}{\partial w^{(L)}} \tag{2.9}$$

Earlier in this thesis, it was mentioned that, when calculating the gradient vector, the weights and biases are updated as if taking a "small step" in the direction of the vector. This was referring to the learning rate μ, a user-defined parameter which determines how big step the network should take in direction of the gradient vector. This does not need

intensive explanation about its relevance, and can be illustrated simply as depicted in figure 2.10. The weights can then be updated with following equations:

$$\triangle w_{t+1} = \mu \nabla E(w_t) \tag{2.10}$$

$$w_{t+1} = w_t + \triangle w_{t+1} \tag{2.11}$$

Introducing more than one node each layer there is a need of more indices, as the weights can affect several nodes, and also, there might be more than one output. Equation 2.12 accounts for these changes by averaging the N paths affecting the cost function.

$$C(w,b) = \frac{1}{N} \sum_{i=0}^{N-1} (y_i - a_i^{(L)})^2 \tag{2.12}$$

Another change which is emphasized in equation 2.13, is how a single training example affects to the cost with respect of previous activations by applying the chain rule.

$$\frac{\partial C_0}{\partial a_k^{(L-1)}} = \sum_{j=0}^{n_L - 1} \frac{\partial z_j^{(L)}}{\partial a_k^{(L-1)}} + \frac{\partial a^{(L)}}{\partial z^{(L)}} + \frac{\partial C_0^{(L)}}{\partial a_j^{(L)}} \tag{2.13}$$

The network and equations described above research published in the 80s and 90s, in data science we call this network ”vanilla neural network” but research has come far since then. Other proposals that have become ”standard” is the introduction to momentum term for the error calculation. Instead of using only the gradient of the current step to guide the search, momentum also accumulates the gradient of the past steps to determine the direction to go. The equations of gradient descent are revised as follows.[56] [49]

$$\triangle w_{t+1} = (1 - \alpha)\mu \nabla E(w_t) + \alpha \triangle w_{t-1} \tag{2.14}$$

$$w_{t+1} = w_t + \triangle w_{t+1} \tag{2.15}$$

The momentum term alpha is a value that determines how much of the gradient and/or previous weight adjustments shall be used. In the equation above alpha equals zero means the weight update is solely determined by the gradient without considering previous weight, and alpha equals one means the opposite. [13]

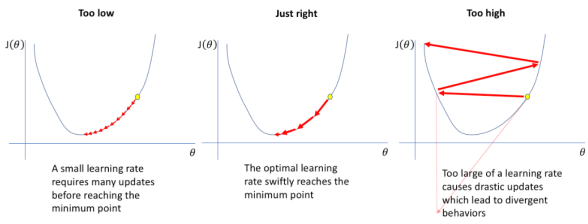


Figure 2.10: Learning rate

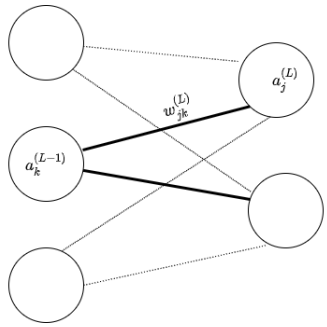


Figure 2.11: Neuron in a neural network

Regularization and overfitting

One common problem in machine learning is overfitting or the network memorizing the data instead of making a valid generalization of the problem [17]. The network should work on not only the data it was introduced to, but also data the network never has seen before. The common way to detect overfitting is training the network on a larger subset and withhold a smaller part. While training on the larger subset and calculating the error to adjust the weights, the smaller subset is only for calculating the error for validation.

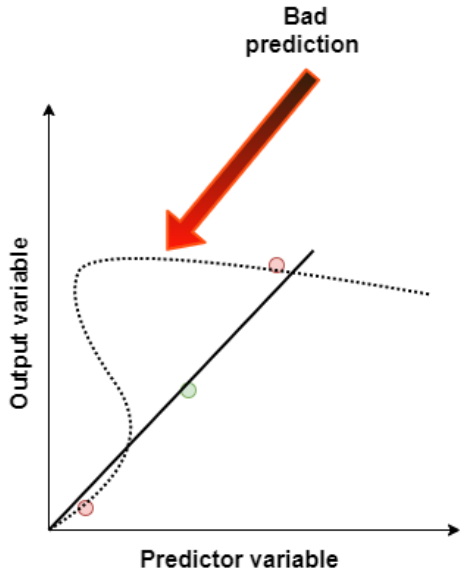


Figure 2.12: Overfitted model

In figure 2.12, the two data points the model can train on are depicted with red color and the data which the model can not train on is depicted with the color green. Between the two red data points, it is infinite possible ways the model could fit a line through, however

upholding some of the data a check can be done to find out if the model is overfitting or not. When the loss of both training and the upheld data decreases, the model can safely continue to train, however, if the loss of the training continues decreasing while the upheld data increases, it is an indication that the model is overfitting.

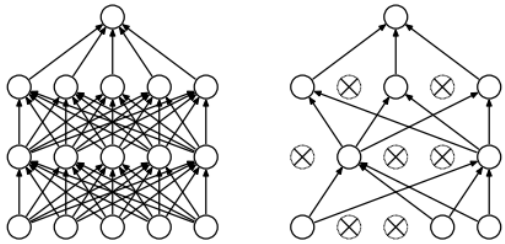
In deep learning, regularly the best performing models tend to be large models trained in a way that restricts the utilization of their entire potential. In other words, encouraging the models to have a preference towards simpler models. This reduces the risk of overfitting, and one way this can be achieved is by adding a weight penalty term to the cost function. [38, 32]

$$C(w,b) = \frac{1}{N}\sum_{i=0}^{N-1}(y_i - a_i^{(L)})^2 + (weightpenalty) \tag{2.16}$$

$$(weightpenalty) = \alpha \sum w_i^2 \tag{2.17}$$

In equation 2.17 an additional term is introduced which is the squared sum of the weights. When the model is being trained the optimization algorithm will minimize both the original loss function and the weight penalty term, and expressing a preference towards smaller weights.

Another simple regularization technique is the dropout method. This method randomly drops out a portion of the nodes in the network with some probability during training. This means that, in each iteration of the training, a subset of the network is trained. This approach encourages the nodes in the network to learn useful features on their own, without being to heavily dependent on nodes.[32]



(a) Standard Neural Net (b) After applying dropout.

Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

Figure 2.13: Dropout [70]

2.5.2 Convolutional Neural Network (CNN)

The computer vision field enables machines to view or perceive the world as humans do. This is what is used in videorecognition, image analysis, classification, etc. One of the main algorithms is the Convolutional Neural Network, a deep learning algorithm that has methods for extracting important features by sharpening, blurring, enhancing and detecting the edges, and other high order processing on images. This is primarily done by assigning importance (learnable weights and biases) to areas or pixels in an image and let the network itself learn the importance of these features or, better said, which features to look for.[26]

The image will go through four different types of layers, which have different purposes. First, it needs to extract the important features, which is done in the convolution layer. From this stage, the network will subsample the output (a feature map) which will preserve the important features and discard the unnecessary data. This will, in other words, give the network fewer parameters to work with. Usually, several of these two different layers are used in conjunction depending on the problem before passing the data to the flattening layer. The flattening layer converts the higher-order representation into a 1D representation so the data can be passed through the fully connected layer, which is a Feedforward neural network described in section 2.2.2.[8]

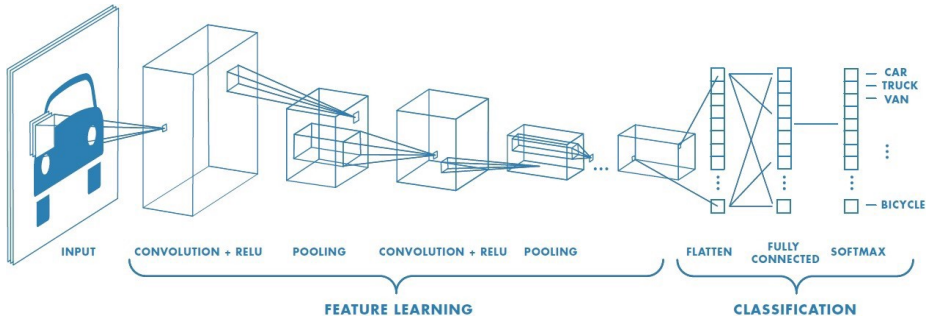


Figure 2.14: Convolutional Neural Network [3]

Convolution layer

The process of filtering out the important features is done by kernel convolution, and it is not only a key element in the Deep learning algorithm CNNs but in many computer vision algorithms. In the convolution layer, a process is happening where a smaller matrix of numbers called the kernel or filter is passed over an image to transform it based on the value of the filter. The output is called a feature map and is calculated based on equation 2.18, where the input image is denoted by f and the filter by h. The indexes of rows and columns of the resulting matrix are noted with m and n.[69]

$$G[m,n] = (f * h)[m,n] = \sum_j \sum_k h[j,k] f(m-j, n-k) \tag{2.18}$$

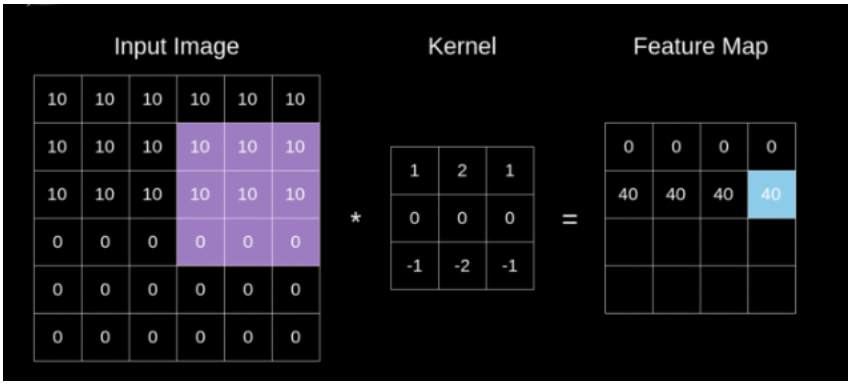


Figure 2.15: Kernel convolutional example [69]

One can imagine placing the kernel over a pixel, like depicted in figure 2.15 with purple color. The corresponding value pair from the kernel and pixel of the picture and multiply them. Finally, sum all products produced in the previous step. G[0,3] (blue color) in the feature map in figure 2.15 would have the following solution.

$$G[0,3] = 10 \times 1 + 10 \times 2 + 10 \times 1..... + 0 \times -1 + 0 \times -2 + 0 \times -1 = 40 \tag{2.19}$$

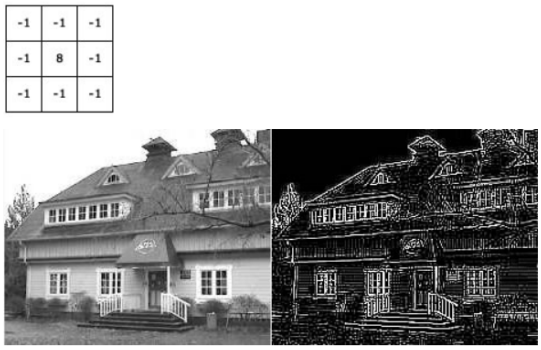


Figure 2.16: Kernel for edge detection[68]

The algorithm will, through error minimization, find itself the kernels which it deems important, and if knowing where the edges are is important for solving the task, then the deep learning algorithm might come to a kernel similar to figure 2.16.

Pooling layer

One thing the pooling layer has in common with the convolutional layer is the dimensionality reduction; this is, decreasing the computational power required, by compressing the image to a smaller size while maintaining the important features. Also, the feature maps generated from the previous layer are dominant features that are positional invariant. In other words, even though the image is rotated, sheered or other transformations have been applied to it, features can still be detected, like edges in figure 2.19. This leads to a lower chance of overfitting and more effective training.
The two common types of pooling are average pooling and max pooling[8]. Max pooling returns the maximum value from a portion of the kernel, while average pooling returns the average of all the values in a portion of the kernel. The portion or pool size of the kernel is chosen by the user.[73]

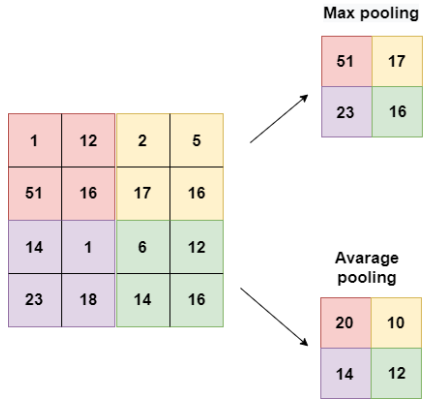


Figure 2.17: Max pooling and average pooling

The term stride [x,y] in the pooling layer tells the algorithm how the pooling window should be moved horizontal and vertical direction. In figure 2.17 the pool-size or size of the sliding window is (2,2) and the stride is [2,2].

Flattening layer

This layer is rather simple, and its purpose is to prepare the data to be fed into a feedforward neural network. This is done by converting the matrices into a vector[59]. The result from the pooling layer in figure 2.17 after flattening would convert to be a vector with 4 samples.

Fully connected layer

This is the layer where the vector is fed to an artificial neural network, like the feedforward network from section 2.2.2. This layer is responsible for performing classification or regression on images. [59]

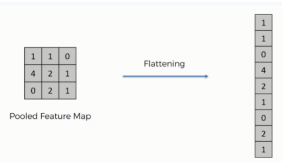


Figure 2.18: Flattening layer

2.6 Computer vision

There have been big improvements in the field of computer vision. A good part of this is from big companies like Google, Microsoft, and Facebook contributing in the form of datasets to benchmark the newly developed networks, or develop networks. However, computer vision gives the computer the ability to extract information from pictures or videos to understand the scene as humans do. In other words, a camera just captures the moment while with computer vision a computer can ”see”. This section, explains the main categories of computer vision tasks.[64]

2.6.1 Classification

The lectures from Fei-Fei Li [65] describe the four main classes of problems in detection and segmentation. These are classification, semantic segmentation, object detection, and instance segmentation. Classification trains the computer to distinguish pictures from one another, by labelling them into different classes. The network should, with high certainty, output if there is a cat in the picture or not, given that it was trained to find pictures with cats. Figure 2.19 shows what the output from a network designed for classification looks like. [64]

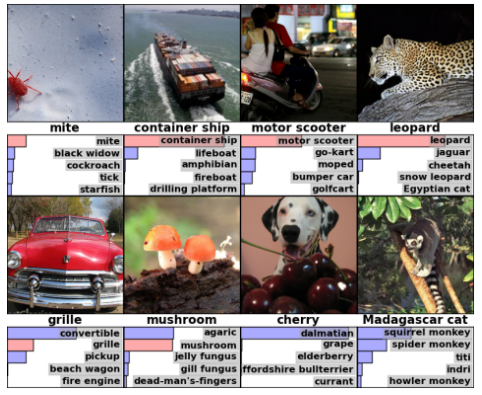


Figure 2.19: Classification [37]

2.6.2 Semantic segmentation

In computer vision segmentation is the process of the partitioning of an image into several segments. This is done by fully convolutional networks [45]. The goal is to simplify the image for further processing. Usually, each pixel is assigned to one of the predefined classes like background, cat, etc. An application of this is the portrait mode on modern phones in the camera application, where the camera knows what is the foreground and background, and blurs the background.[64]

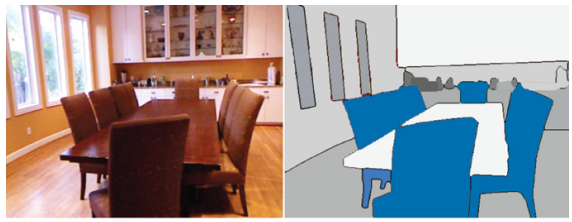


Figure 2.20: Semantic segmentation [67]

2.6.3 Object detection and instance segmentation

As the name implies, object detection finds the instances of the class it is trained on within the image. Given that a network was trained for detecting people in a picture, then, the network should confidently recognize all the people in the picture and their location, while accounting for overlapping. This is usually depicted with boundary boxes on the output pictures where the instance is located.[64]

Instance segmentation is similar to object detection, their main difference is that it gives one step higher accuracy as the network should also know all the pixels that belong to the instance. It should, as the object detection, know how many different instances of a class are in an image and keep track of them. [64]

Chapter 3

Related work

3.1 Datasets

One of the primary goals of computer vision is the understanding of visual scenes, which involves localization, recognition, characterization, and relationships between objects; however, it requires excessive data to train a computer vision algorithm. The ranked searches of a dog on Google images or Flickr will result in uncompromised, well-composed images of a dog[43]. The problem is that, when training the computer vision model, the dataset should be representative of the problem one wants to solve. Detecting dogs in CCTV footage on a common street requires labelled images in similar scenarios. Studio-quality and well-centered pictures of dogs will not be sufficient, and this is where contributions to the open-source community from well-established IT-companies come in [43].

3.1.1 Common Objects in context dataset

This dataset has been described as following in the COmmon object in COntext (COCO) paper [43]: ”We present a new dataset with the goal of advancing the state-of-the-art in object recognition by placing the question of object recognition in the context of the broader question of scene understanding. This is achieved by gathering images of complex everyday scenes containing common objects in their natural context.” The images were mostly gathered by hiring from Amazon Mechanical Turk. What is unique with the COCO dataset is that it provides instance-level segmentation and has a high level of instances per class compared to other datasets.

3.1.2 VOC dataset

The Pascal Visual Object Classes (VOC) challenge was presented in 2005 and associated with the benchmark test for object detection. Their diverse development kit provided an easy way to evaluate object detection models, which today have become standard. The

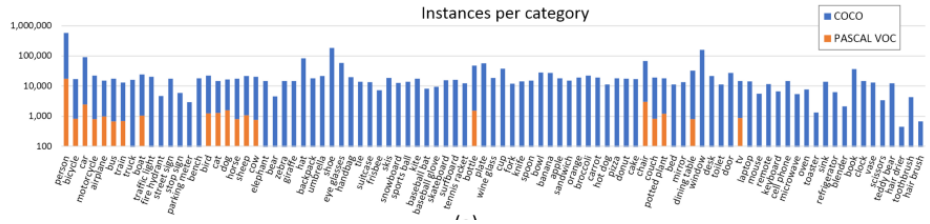


Figure 3.1: Object detection vs instance segmentation [43]

dataset consists of images collected from Flickr, which has been annotated and made ready for training [19]. The dataset contained necessary data for both classification and object detection, however, it consisted of only 20 classes and, as the technologies advanced within machine learning, the necessity of making a new challenging dataset arose. Review papers that evaluate different networks, usually list metrics generated for VOC and COCO [19] [43].

3.2 Feature Pyramid Network

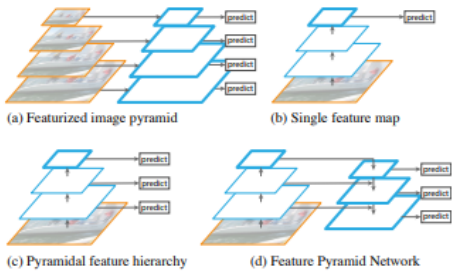


Figure 3.2: Feature Pyramid [41]

The Facebook AI research team proposed the feature pyramid network in 2017 [41]. The problem addressed in their paper was that recognizing objects at vastly different scales is a fundamental challenge. The solution until then was solved using an image pyramid (figure 3.2 a) to build a feature pyramid. However, using a single picture in different scales to generate sets of feature maps was computationally heavy and required a lot of memory. The proposal for this problem was the feature pyramid network in figure 3.2d, which is computationally similar in speed as generating a single feature map, as in the CNN after the convolution layer described in chapter 2.2.2 generated [41].

Buttom-up pathway

A feedforward convolutional neural network (often called the backbone), as described in chapter 2 is the bottom-up pathway of the feature pyramid network. When using transfer learning to initialize the backbone of the feature pyramid, there might be many layers after each other with outputs the same size. This is defined in the pyramid network as one stage. As shown in figure 3.10, the feature maps fed through the stages in the bottom-up pathway, the spatial dimension of the image is reduced by 1/2. The output of each convolution module is later used for the top-down pathway.[31]

Top-down Pathway

The top-down pathway ”simulates” higher resolution features by upsampling, by a factor of two from previous pyramid layers. The features from the higher levels are spatially coarser but have semantically stronger features[41]. The lateral connections merge feature maps of the same size from the bottom-up pathway and the top-down pathway. The reason for merging the feature maps is that the activation localized in the bottom-up pathway is more accurate as it has been subsampled fewer times. In figure 3.3 the pyramid feature P1 does not exist, or upsampling M2 and merging it with conv1, this is due to the spatial dimension of C1 is too large, which would result in a slower process.[31]

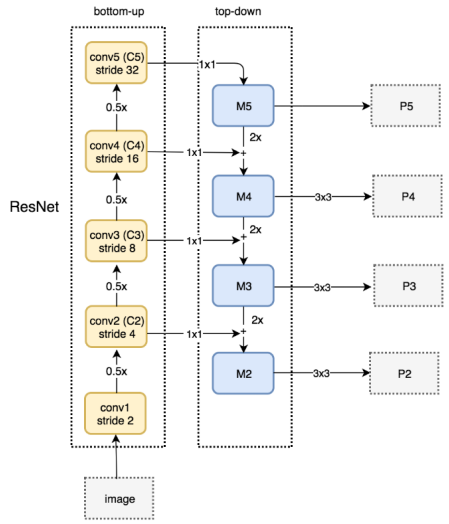


Figure 3.3: Bottom-up and top down pathways for feature pyramid network [31]

3.3 Transfer learning

Transfer learning in Machine Learning refers to the transferring of knowledge between networks. The knowledge gained while solving one problem could be reapplied to solving another similar problem. For example, the knowledge gained while training a network on voice recognition could be applied to trigger word recognition when activating a virtual assistant like Siri on iPhones or Google Assistant on Android phones. This is not far from how humans learn, as humans don't start from scratch every time learning something but rather build on from past experiences. One of the main reasons to use transfer learning is due to insufficient data for a new domain, or the overall problem is dependent on a sub-problem that has been solved efficiently before with state-of-the-art deep learning algorithms [54].

There are different forms of transfer learning strategies. Moreover, in computer vision, the most common is off-the-shelf pre-trained models [58], as in the case of this thesis. A subset of the previously trained architectures can be used for a new problem where the output layer is substituted with the desired output. The criteria to use transfer learning can be summarized to [54]:

- There is not enough labelled training data to train a network from scratch.
- There already exists a network that is pre-trained on a similar task, which is usually trained on massive amounts of data.
- When task 1 and task 2 have the same input.

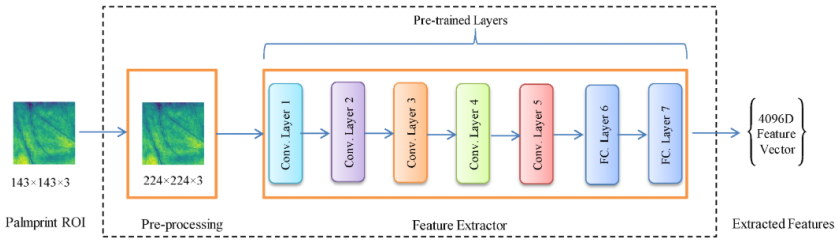


Figure 3.4: Pre-trained Deep Learning Models as Feature Extractors [33]

3.3.1 Residual Neural Network (ResNet)

When deciding the number of layers in a neural network and nodes, at first glance, it might seem that more layers and nodes are better. This is not a bad assumption, after all, the amount of neurons is higher, which gives more activations to estimate a function. However, a deeper neural network leads to a more strenuous training process. This is due to the famously known vanishing gradient problem. When the network is too deep during backpropagation, the cost function shrinks to zero for earlier layers. More details for vanish gradient problem can be found at [29].

The ResNet, or Residual neaural Network, can have variable sizes; in this thesis, both ResNet101 and ResNet50 were experimented with, but this section will consider ResNet34, as in the official Microsoft research paper [28]. In the figure, the ResNet consists of one convolution and one pooling step. After these steps, it repeats the same pattern: perform 3x3 convolution with fixed map dimensions [64,128,256,512] and bypasses every other convolution. The dotted line represents a change in the dimension of the input volume. The reduction is not like in chapter two, where the pooling operation was responsible for the reduction, but rather, the reduction is achieved by increasing the strides from one to two in these steps.

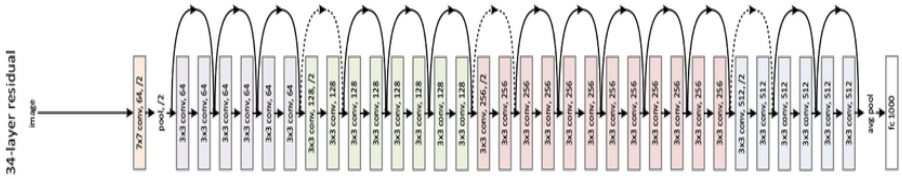


Figure 3.5: ResNet34 architecture [28]

This network has a better learning representation, it is shown that, when adapted to deep learning object detection algorithms, it significantly improves accuracy. The result of one of the states-of-the-art object detection algorithms, called Faster R-CNN [61], had the authors achieve a relative improvement of 28% compared to other CNN models for feature extraction. By using the ResNet as a backbone for the feature pyramid, the model achieved 1st place in multiple categories in the COCO2015 challenge.

3.4 A brief summary of object detection algorithms

A feature descriptor is a representation of an image that simplifies the image, most commonly by extracting useful information and discarding counterproductive features. One of the famous feature descriptors, Histogram of Oriented Gradients (HOG) [16], would take an image as an input in the form of a 3-D array (considering RGB), and output a 1-D array, or a vector.

In the early stages of object detection, the task was divided into three main stages.

- Proposal generation.
- Feature vector extraction.
- Region classification.

During the proposal stage, the idea is to use techniques to find regions of interest or, in other words, regions that might contain an object. Vedaldi et al. [75] suggested using multiple kernels as sliding windows to scan a portion of the image at the time. To

account for objects in different scales and aspect ratios, the image was resized multiple times, and also set into different scales before sliding over the images. Each location that was retrieved in the first stage from the sliding windows was, in the second stage, used for extracting the important feature vectors. Methods like HOG were used for the feature vector extraction. Features from covered regions were then, in the third step, assigned categorical labels, most commonly by using support vector machine which was known for its good performance.[80]

The most successful traditional object detection algorithms like Zhang et al. [81] from the Institute of Automation of the Chinese Academy of Science had a recurrent theme going on. They all based on carefully hand engineering feature descriptors. With the help of good feature descriptors, in 2010, they achieved the state-of-the-art result in the VOC dataset. However, these traditional methods only achieved incremental progress.[80]

Surprisingly, in 1998, there were attempts to adapt deep neural networks for digit recognition and showed promising results; however, it was not further explored for many years. This is, perhaps, because widely used dataset was not introduced for benchmarking, which would have revealed its potential; therefore, algorithms like support vector machines were prominently adopted. Alex Krizhevsky et al [37]. introduced a deep convolutional neural network trained on ImageNet, a dataset consisting of 1.2 million high-resolution images. They achieved first place with an error rate of 15.3%, a marginal difference compared to the second place with 26.2%. After these promising results, deep learning techniques were quickly adapted compared to traditional methods [80].

Today, object detection has two categories of deep learning frameworks: the two-stage detectors and single-stage detectors. The two-stage detectors are incremental improvements from the R-CNN first proposed by Ross Girshick et al [24].

The two-stage detectors used a generator to generate its proposals; in the case of R-CNN, it was done by Selective Search[24], but later advances use CNN as a region proposal network to generate the proposals [27, 61], which are classified in the following step. One-stage detectors make a categorical prediction on each location of the feature map (without the region classification step). A common theme was that the two-stage detector would achieve state-of-the-art results on public benchmarks; however, the one-stage detector as still superior in real-time detection [80].

Important two-stage detectors

- **R-CNN** is the pioneering two-stage object detector which achieved 54.7% mAP on VOC dataset which was significantly higher than the second place with 13%. R-CNN uses a combination of traditional object detection (such as Selective search or SVM) and a convolutional neural network to achieve this. More details can be found at [24].
- **Fast R-CNN** addressed the biggest shortcoming of R-CNN. Arguably, the most

significant difference between R-CNN and fast R-CNN is the generation of regions. The region proposals are generated by the output (feature map) of a convolutional neural network, instead of the input image; this bypasses approximately 2000 passes through the convolutional neural network. Then, the regions of proposals are generated and wrapped into fixed sizes from the feature map. The most significant achievement was that it only needed approximately 9 hours to train, compared to the 84 hours of the R-CNN. Also, the detection time for a single image dramatically reduced from 49 seconds to 2.3. More details can be found at [23].

- **Faster R-CNN** achieved an end to end training (that is, training the entire network as a whole). Both the R-CNN and the Fast variant relied on selective search on the input image or the feature map to generate a region of proposals. Faster R-CNN introduced a fully connected convolutional neural network to generate the region proposals. This also achieved state-of-the-art mAP on the COCO dataset while reducing the detection time by 10 times. More details can be found at [61].
- **Mask R-CNN**, while achieving the state-of-the-art mAP on COCO dataset, it was also a extension of Faster-RCNN that allowed instance segmentation. The mask is generated by a separate, fully connected convolutional neural network parallel with the classification and bounding box regression. More details can be found at [27].

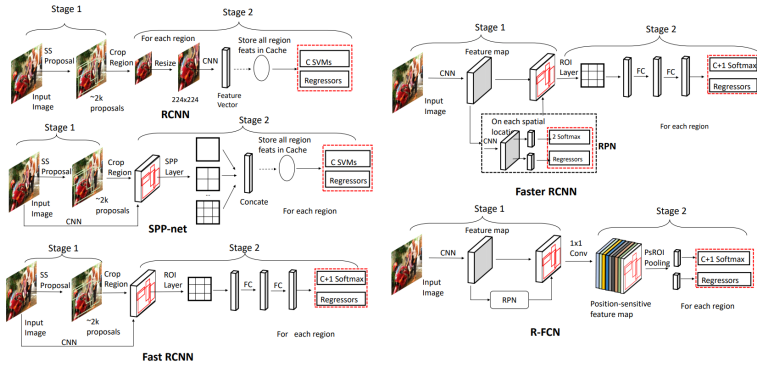


Figure 3.6: Two-stage detectors[80]

Important single-stage detectors

- **YOLO** or You Only Look Once real-time object detection algorithm, spatially divided the input image using a 7x7 grid, which made sub-parts (cells) of the image equally sized, and was later used as region proposal for one or more objects [21]. YOLO considers object detection as a regression problem; then, for each cell, there is calculation of bounding boxes, classification of the objects, and determination

of whether the location had an object or not. It also had a performance of 45 FPS to 155 FPS (simplified backbone). Its limitation was crowded objects and small objects. This algorithm is not suitable for predicting object at multiple scales either [60].

- **SSD** or Single Shot MultiBox Detector addressed the limitations of YOLO, especially having a fixed-sized proposal. SSD similarly divided images into grid cells, however for each sell a set of anchor were generated with different aspect ratios (i.e.[1:1, 1:2, 2:1]) and diffrent scales[1, 0.5, 2]; SSD also detected objects from multiple feature maps to make its prediction; and it achieved detection accuracy comparable to Faster R-CNN. More details can be found at [44].

- **CornerNet** delivered something special, as networks before this one initialized anchors in which objects were fitted into, while CornerNet didn't need initialization of anchor. This anchor-free approach detected objects like a pair of corners. More details can be found at [39].

- **RetinaNet** addressed further limitations that single-stage detector had until this point. The models with region proposal network have the advantage of having less negative samples to filter negative samples, the networks until now had a class imbalance between foreground and background. RetinaNet solved this with a customized loss function; this network also implemented feature pyramid network. More details can be found at [42].

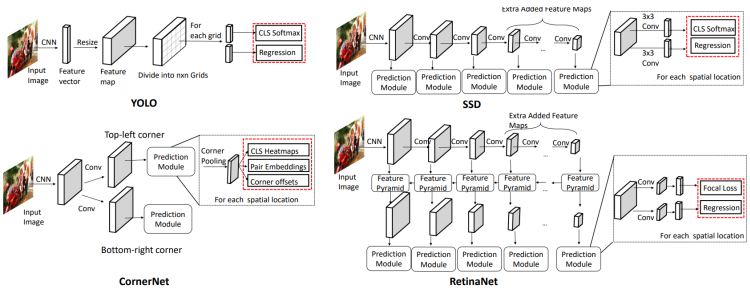


Figure 3.7: Single-stage detectors [80]

3.5 Choosing network - A Review

There is no right answer when choosing a network, as it is quite a problem-dependent decision and, as already mentioned, while some networks have good accuracy, other networks sacrifice accuracy for speed. However, there are papers published were the state-of-the-art networks are reviewed and benchmarked. One such paper was published by Zhao et al. [82], which has been a key source in the stage of choosing a network. Also, Wu et al. [80], has been a good source to review the development of the two types

TABLE III
COMPARATIVE RESULTS ON VOC 2012 TEST SET (%).

Methods	Trained on	areo	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	mAP
R-CNN(Alex) [15]	12	71.8	65.8	52.0	34.1	32.6	59.6	60.0	69.8	27.6	52.0	41.7	69.6	61.3	68.3	57.8	29.6	57.8	40.9	59.3	54.1	53.3
R-CNN(VGG16) [15]	12	79.6	72.7	61.9	41.2	41.9	65.9	66.4	84.6	38.5	67.2	46.7	82.0	74.8	76.0	65.2	35.6	65.4	54.2	67.4	60.3	62.4
Bayes [85]	12	82.9	76.1	64.1	44.6	49.4	70.3	71.2	84.6	42.7	68.6	55.8	82.7	77.1	79.9	68.7	41.4	69.0	60.0	72.0	66.2	66.4
Fast R-CNN [16]	07++12	82.3	78.4	70.8	52.3	38.7	77.8	71.6	89.3	44.2	73.0	55.0	87.5	80.5	80.8	72.0	35.1	68.3	65.7	80.4	64.2	68.4
SutffNet30 [100]	12	83.0	76.9	71.2	51.6	50.1	76.4	75.7	87.8	48.3	74.8	55.7	85.7	81.2	80.3	79.5	44.2	71.8	61.0	78.5	65.4	70.0
NOC [114]	07+12	82.8	79.0	71.6	52.3	53.7	74.1	69.0	84.9	46.9	74.3	53.1	85.0	81.3	79.5	72.2	38.9	72.4	59.5	76.7	68.1	68.8
MR-CNN&S-CNN [110]	07++12	85.5	82.9	76.6	57.8	62.7	79.4	77.2	86.6	55.0	79.1	62.2	87.0	83.4	84.7	78.9	45.3	73.4	65.8	80.3	74.0	73.9
HyperNet [101]	07++12	84.2	78.5	73.6	55.6	53.7	78.7	79.8	87.7	49.6	74.9	52.1	86.0	81.7	83.3	81.8	48.6	73.5	59.4	79.9	65.7	71.4
OHEM+Fast R-CNN [113]	07++12+coco	90.1	87.4	79.9	65.8	66.3	86.1	85.0	92.9	62.4	83.4	69.5	90.6	88.9	88.9	83.6	59.0	82.0	74.7	88.2	77.3	80.1
ION [95]	07+12+S	87.5	84.7	76.8	63.8	58.3	82.6	79.0	90.9	57.8	82.0	64.7	88.9	86.5	84.7	82.3	51.4	78.2	69.2	85.2	73.5	76.4
Faster R-CNN [18]	07++12	84.9	79.8	74.3	53.9	49.8	77.5	75.9	88.5	45.6	77.1	55.3	86.9	81.7	80.9	79.6	40.1	72.6	60.9	81.2	61.5	70.4
Faster R-CNN [18]	07++12+coco	87.4	83.6	76.8	62.9	59.6	81.9	82.0	91.3	54.9	82.6	59.0	89.0	85.5	84.7	84.1	52.2	78.9	65.5	85.4	70.2	75.9
YOLO [17]	07++12	77.0	67.2	57.7	38.3	22.7	68.3	55.9	81.4	36.2	60.8	48.5	77.2	72.3	71.3	63.5	28.9	52.2	54.8	73.9	50.8	57.9
YOLO+Fast R-CNN [17]	07++12	83.4	78.5	73.5	55.8	43.4	79.1	73.1	89.4	49.4	75.5	57.0	87.5	80.9	81.0	74.7	41.8	71.5	68.5	82.1	67.2	70.7
YOLOv2 [72]	07++12+coco	88.8	87.0	77.8	64.9	51.8	85.2	79.3	93.1	64.4	81.4	**70.2**	91.3	88.1	87.2	81.0	57.7	78.1	71.0	88.5	76.8	78.2
SSD300 [71]	07++12+coco	91.0	86.0	78.1	65.0	55.4	84.9	84.0	93.4	62.1	83.6	67.3	91.3	88.9	88.6	85.6	54.7	83.8	77.3	88.3	76.5	79.3
SSD512 [71]	07++12+coco	91.4	88.6	82.6	71.4	63.1	**87.4**	88.1	93.9	66.9	86.6	66.3	92.0	91.7	90.8	88.5	60.9	87.0	75.4	90.2	80.4	82.2
R-FCN (ResNet101) [16]	07++12+coco	**92.3**	**89.9**	**86.7**	**74.7**	**75.2**	86.7	**89.0**	**95.8**	**70.2**	**90.4**	66.5	**95.0**	**93.2**	**92.1**	**91.1**	**71.0**	**89.7**	**76.0**	**92.0**	**83.4**	**85.0**

* '07++12': union of VOC2007 trainval and test and VOC2012 trainval. '07++12+COCO': trained on COCO trainval35k at first then fine-tuned on 07++12.

Figure 3.8: VOC 2012 test set results Zhao et al. [82]

of object detection networks throughout the years.

Earlier in this chapter, the VOC and COCO datasets were introduced; and figure 3.9 from [82] shows the result on the 2012 VOC's dataset. The front runners are the SSD network[44], faster R-CNN network [61], and faster R-CNN based network. At first sight, SSD-network might seem outperforming (in terms of accuracy) Faster R-CNN based network; but, luckily, there is more benchmarking available.

In figure 3.9, there is a clear indication that, if incorporated properly, more powerful CNN models definitely improve object detection like ResNet, VGG, etc. in terms of accuracy; the figure shows Faster R-CNN based networks outperforming SSD or single-stage networks. Still, a deliberate decision has to be made based on the problem one wishes to solve, as approximately ten percentage points loss in terms of accuracy might be admirable for a gain in performance of 200 times (0.2 vs. 45 frames per second). [82]

TABLE V
COMPARISON OF TESTING CONSUMPTION ON VOC 07 TEST SET.

Methods	Trained on	mAP(%)	Test time(sec/img)	Rate(FPS)
SS+R-CNN [15]	07	66.0	32.84	0.03
SS+SPP-net [64]	07	63.1	2.3	0.44
SS+FRCN [16]	07+12	66.9	1.72	0.6
SDP+CRC [33]	07	68.9	0.47	2.1
SS+HyperNet* [101]	07+12	76.3	0.20	5
MR-CNN&S-CNN [110]	07+12	78.2	30	0.03
ION [95]	07+12+S	79.2	1.92	0.5
Faster R-CNN(VGG16) [18]	07+12	73.2	0.11	9.1
Faster R-CNN(ResNet101) [18]	07+12	**83.8**	2.24	0.4
YOLO [17]	07+12	63.4	**0.02**	**45**
SSD300 [71]	07+12	74.3	**0.02**	**46**
SSD512 [71]	07+12	76.8	0.05	19
R-FCN(ResNet101) [65]	07+12+coco	83.6	0.17	5.9
YOLOv2(544*544) [72]	07+12	78.6	0.03	40
DSSD321(ResNet101) [73]	07+12	78.6	0.07	13.6
DSOD300 [74]	07+12+coco	81.7	0.06	17.4
PVANET+ [116]	07+12+coco	**83.8**	0.05	21.7
PVANET+(compress) [116]	07+12+coco	82.9	0.03	31.3

*

Figure 3.9: VOC 2007 test set results Zhao et al. [82]

The improvement in the performance obtained by the single-stage networks carry a cost evident on the results of the COCO dataset. This dataset has more crowded and smaller objects, for this dataset contains images in context, as mentioned earlier in this chapter. The bottleneck of SSD is small objects; as illustrated in figure 3.9, it has a significant accuracy loss compared to two-stage networks for small objects. This bottleneck is still

prominent, even after feeding the networks with higher resolution images (SSD300 vs. SSD512). The clear winner, in terms of accuracy is the Mask R-CNN, with its robust backbone CNN architecture (FPN and ResNet) beating the competition in every category. In chapter two, categories of computer vision were described, and Mask R-CNN was an instance segmentation algorithm, considering that the problem in this thesis was object detection. In the Mask R-CNN paper [27], the author produced the numbers by letting the network work in object detection, where the mask outputs are ignored; and used, for evaluation, the bounding box coordinates. Also, in the paper, it is mentioned that the extra mask output's performance cost is negligible.

TABLE IV
COMPARATIVE RESULTS ON MICROSOFT COCO TEST DEV SET (%).

Methods	Trained on	0.5:0.95	0.5	0.75	S	M	L	1	10	100	S	M	L
Fast R-CNN [16]	train	20.5	39.9	19.4	4.1	20.0	35.8	21.3	29.4	30.1	7.3	32.1	52.0
ION [95]	train	23.6	43.2	23.6	6.4	24.1	38.3	23.2	32.7	33.5	10.1	37.7	53.6
NOC+FRCN(VGG16) [114]	train	21.2	41.5	19.7	-	-	-	-	-	-	-	-	-
NOC+FRCN(Google) [114]	train	24.8	44.4	25.2	-	-	-	-	-	-	-	-	-
NOC+FRCN (ResNet101) [114]	train	27.2	48.4	27.6	-	-	-	-	-	-	-	-	-
GBD-Net [109]	train	27.0	45.8	-	-	-	-	-	-	-	-	-	-
OHEM+FRCN [113]	train	22.6	42.5	22.2	5.0	23.7	34.6	-	-	-	-	-	-
OHEM+FRCN* [113]	train	24.4	44.4	24.8	7.1	26.4	37.9	-	-	-	-	-	-
OHEM+FRCN* [113]	trainval	25.5	45.9	26.1	7.4	27.7	38.5	-	-	-	-	-	-
Faster R-CNN [18]	trainval	24.2	45.3	23.5	7.7	26.4	37.1	23.8	34.0	34.6	12.0	38.5	54.4
YOLOv2 [72]	trainval35k	21.6	44.0	19.2	5.0	22.4	35.5	20.7	31.6	33.3	9.8	36.5	54.4
SSD300 [71]	trainval35k	23.2	41.2	23.4	5.3	23.2	39.6	22.5	33.2	35.3	9.6	37.6	56.5
SSD512 [71]	trainval35k	26.8	46.5	27.8	9.0	28.9	41.9	24.8	37.5	39.8	14.0	43.5	59.0
R-FCN (ResNet101) [65]	trainval	29.2	51.5	-	10.8	32.8	45.0	-	-	-	-	-	-
R-FCN*(ResNet101) [65]	trainval	29.9	51.9	-	10.4	32.4	43.3	-	-	-	-	-	-
R-FCN**(ResNet101) [65]	trainval	31.5	53.2	-	14.3	35.5	44.2	-	-	-	-	-	-
Multi-path [112]	trainval	33.2	51.9	36.3	13.6	37.2	47.8	29.9	46.0	48.3	23.4	56.0	66.4
FPN (ResNet101) [66]	trainval35k	36.2	59.1	39.0	18.2	39.0	48.2	-	-	-	-	-	-
Mask (ResNet101+FPN) [67]	trainval35k	38.2	60.3	41.7	20.1	41.1	50.2	-	-	-	-	-	-
Mask (ResNeXt101+FPN) [67]	trainval35k	**39.8**	**62.3**	**43.4**	**22.1**	**43.2**	**51.2**	-	-	-	-	-	-
DSSD513 (ResNet101) [73]	trainval35k	33.2	53.3	35.2	13.0	35.4	51.1	28.9	43.5	46.2	21.8	49.1	66.4
DSOD300 [74]	trainval	29.3	47.3	30.6	9.4	31.5	47.0	27.3	40.7	43.0	16.7	47.1	65.0

Figure 3.10: COCO test set results Zhao et al. [82]

Chapter 4

Methodology

The thesis's main objective is to explore how deep learning can automate the maintenance of railings across Norway's roads. The previous chapters addressed the goals, relevant theory considering image processing in deep learning, categories within it, and related work considering object detection with traditional and deep learning and its development throughout the years. Also, the introduction of state-of-the-art networks considering object detection was previously presented. This chapter describes the data collection, data preparation, and implementation details for the experimentation of the thesis.

4.1 Data

There was no available dataset found about the railings for the roads in Norway, so it was determined that it does not currently exist (not publicly available). Therefore, this thesis dataset was primarily collected by iSi AS and Arvid Gjerde AS and annotated by the skilled employees of Arvid Gjerde AS.

4.1.1 Labelling and preparation

While training the network, it is necessary to rate the accuracy of the predictions the system managed to produce. The output that the system generates is bounding boxes (exact location) around the object, assigning a label to the area previously located, and the exact pixels belonging to the class. Labelling in this project was done by creating a lookup table with the necessary information of each image, like the class and bounding box. The software used to generate this lookup table was VGG annotation tool by [18].

Image annotated such as illustrated in figure 4.1, the generation of the bounding boxes' corners is done by extracting the x_{min}, x_{max} , y_{min}, and y_{max} and adding a small offset the coordinates. By this annotation, the mask is also generated, by having a representation of the image in the form of a 2-D array and assigning 1 in the array where the object

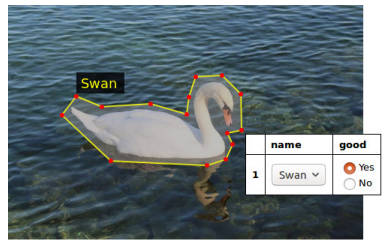


Figure 4.1: VGG annotation tool [18]

is present and 0 otherwise. The label has to be assigned manually (i.e., cat/dog).

After training on the initial dataset, the network didn't seem to generalize and make useful predictions; therefore, an analysis of the dataset was required. One methodology for preparing the datasets was the manipulation (adding, deleting, cropping, etc.) of the training set and validation set was acceptable. Still, the testing dataset should not be tampered. Any modifications that will be done should be with the purpose of better generalization (that is, performing better on data that the network has never seen before).

Visual inspection

This section, considers one class in which the visual inspection was applied, but the same analogy can be used for the other classes. Class A, or "damaged railing", has a variety of subclasses, meaning a "damaged rail" can be bulk, rust, deformation, etc., compared to the class "wrong end". The images were imported to a vector drawing application with a red or green border around the image to indicate that it belongs to the training set or validation set. Then, they were sorted into their class and their subclass. As discussed

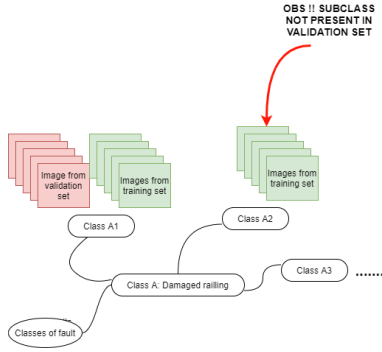


Figure 4.2: Visual inspection

in chapter two, the performance of the validation set allows the user to indicate whether the model is overfitting or not. Sorting them (figure 4.3) allowed it to find instances in the training set, which is not represented in the validation set. The solution was as

simple as moving some of these images to the validation set. Moreover, this also worked as an indication of the correlation between the training and validation set. This method allowed finding some instances of wrongly labelled images. In a particular case, the exact scene in two different photos had two different labels. Even though the dataset was labelled by a professional, mistakes can be made, considering that it takes one misclick on a dropbox to mislabel it. Both were removed, as there was no professional to label them again. Another interesting finding was that the class "other mistakes" had almost no correlation between the datasets and could be assigned to various subclasses (every image was its own subclass); that is why this class and the instances belonging to them were removed.

4.2 Implementation details

4.2.1 Local computing

Initially, it was thought to develop model and train it on a local machine; however, training a CNN requires a specific hardware that can do it efficiently. If training it on an average PC, networks like Faster R-CNN or Mask R-CNN takes weeks to train without dedicated GPU. Also, there must be many hyperparameters adjusted to train the network, meaning that trying and failing is a part of the procedure for making the network work. The CNNs have an exponential training performance gain by training it on GPU, which currently the libraries supported by the machine learning libraries (Tensorflow and Pytorch) are CUDA developed by Nvidia; hence, the GPU required is an Nvidia GPU. Moreover, if using cutting edge algorithms, one has to take care of many dependencies like drivers, configurations, etc., which makes it hard to recommend over a cloud computing for the company, as the goal is to make it as simple as possible. However, if the local machine is desired, one can use these sources [2] [1] for Windows, Linux and macOS, respectively, for the correct setup. However, both local (for evaluation) and cloud (for training) computing were used in this thesis.

4.2.2 Cloud computing

The solution for extreme training time in this project was to do the computations using cloud services solutions that already exist, in this case, Amazon web services. Many cloud computing solutions exist; and some of the most prominent players in this field are Google Cloud, AWS, and Microsoft's Azure. The instances available are vast, and one can import instances already initialized for Deep Learning. Still, as there are many dependencies to make a state-of-the-art algorithm work, the decision fell upon configuring it specially for this thesis;which resulted in an instance called p3.2xlarge; which includes a 16GB of video ram, 8 virtual CPUs, a memory of 61 GB, and a bandwidth up to 10 Gbps. Further installation and modifications were:

- CUDA 9 and cuDNN 7 [79]
  - Nvidia libaries used to run computations on the GPU

- Python 3.7.0 [74]
  - Object oriented programming language
- Tensorflow 1.8.0 [5]
  - A free and open-source software library for dataflow and differentiable programming across a range of tasks.
- Keras 2.2.0 [15]
  - Keras is an open-source neural-network library written in Python.
- NumPy 1.18.0 [78]
  - Python library for scientific computing.
- Imgaug 0.4.0 [34]
  - Python library image augmentation.
- Tensorboard 0.4.0 [48]
  - Provides the visualization and tooling needed for machine learning experimentation
- Matterport/mask R-CNN [6]
  - framework for easier implementation of Mask R-CNN

4.3 Mask R-CNN

4.3.1 Model details

Region proposal

Feature pyramid network [48] and ResNet [43] was introduced in chapter two; and they are the backbone of the Mask R-CNN [27] implemented in this project. The network used for the bottom-up pathway in the feature pyramid network for Mask R-CNN was the ResNet, which produced the pyramid features for the Region Proposal Network (RPN). The purpose of using RPN is to provide regions of interest, by obtaining bounding boxes for each feature, and a probability for the area that contains an object.
The regions are generated by sliding a window over the image, and the center of each position that is stored is called the anchor. Pre-defining aspect ratios (usually 1:1, 1:2 and 2:1) and sizes (usually 64, 128, 256) are used to define bounding boxes for each anchor (9 anchor boxes) [7].
The RPN takes the anchor boxes as input and outputs the probability of an object is present (foreground or background) in the box, and refinement adjusts the coordinate boxes. This is done because of anchor boxes that have a high probability of containing an object might not be well centered. The following step after getting the two outputs is

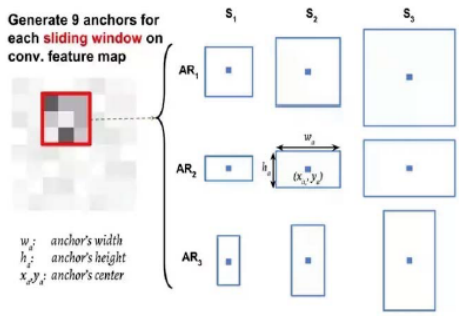
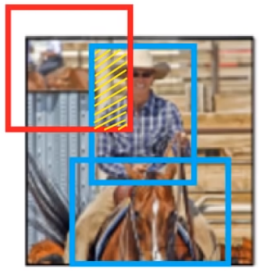
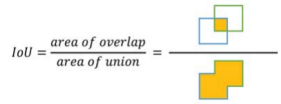


Figure 4.3: Anchor boxes [63]

discarding redundant anchor boxes. The discarded anchor boxes are chosen by selecting the boxes with the lowest probability, invalid boxes, and rejecting sets of boxes with low Intersection Over Union (IOU) [63].



(a) Invalid proposal



(b) High IOU value (i.e. same proposal)

Figure 4.4: Discarding of proposal [21]

Region of interest align

The second stage is similar to the Faster R-CNN, as it was the predecessor to the Mask R-CNN. After the RPN output, the adjusted anchor boxes with the probability of containing an object (proposal) serves as the input in the next stage. Mask R-CNN made some modifications to improve instance segmentation and performance, as the Faster R-CNN was not designed for pixel-to-pixel alignment between the network input and output. To fix this misalignment, it was proposed a modification to this module called ROIAlign [25] [27].

When generating anchor boxes, the RPN generates them with different size and aspect ratios. A Convolutional Neural Network does not perform well on variable input; hence, transformation to the input image is done in networks with CNN. The job of the module Region of interest align (similarly to the Region of interest Pool in Faster R-CNN) is to

transform all proposals to the same size. When scaling the feature map that corresponds to the Regions Of Interest (ROI) by ROI pooling, the first step is to divide each ROI into the same number of bins. The boundaries of the ROI, feature map, and bins may not match; so, quantization (transforming a broad set of values to a discrete set) of the bins was used for aligning the ROI. Max-pooling is used on all of the bins; however, quantization causes a significant drop in performance for tasks that need higher accuracy, like predicting the mask, and the author of Mask R-CNN proposed a more accurate ROI align [27].

ROI pooling solved the mismatch by using quantization on the bins that do not correspond to the feature maps. On the other hand, the ROI align solved this by using bilinear interpolation, which also allowed the generation of bins of the same size within each ROI. The number of points used for the bilinear interpolation is found to be best when using 4 points according to the paper [27].

Predictions

The last module of the Mask R-CNN are the network heads, that take the ROI with the same dimension, which was constructed on the previous step to make its predictions. The final predictions are the bounding boxes, classification, and prediction of the object's mask. It was mentioned in chapter two that Mask R-CNN has an additional branch of a fully convolutional layer for predicting the mask. Knowing the regions of interest on the feature pyramid network's original feature map, the regions are used to predict the classes and bounding boxes.

4.3.2 Training details

Tensorboard

In chapter two, regularisation techniques were presented, the techniques that were used in this thesis (dropout [70], and weight decay [32]); this was for combating the network remembering the instances from the dataset instead of learning to generalize from the instances. The training and validation sets uses have also been explained, as it is an upper bound of how much can the network learn from training before starting to remember. Tensorboard was used for visualizing the losses over the training and validation set. Additional to this, it is possible to, in order to save computational power, use a subset of the validation set for each iteration to indicate if the model is overfitting. The size of the subset depends on the dataset's size; so, if there is availability of vast datasets, then 30% - 40% of the validation set should be enough. In this project, 75% of the validation set was used, as the available dataset was not vast enough. This approach might cause high fluctuation in the cost function of the validation, and the exponential moving average can be used (smoothing in Tensorboard) to give an indication of the true cost.

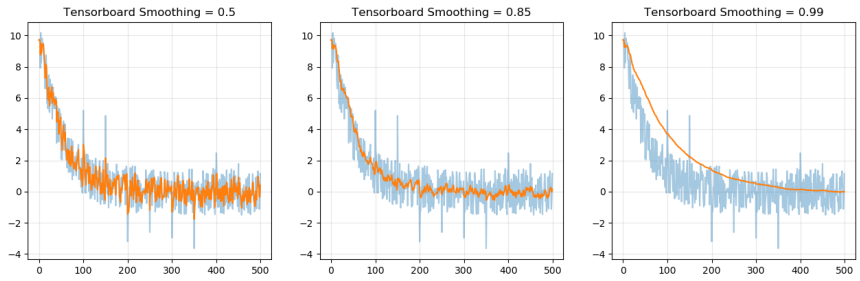


Figure 4.5: Smoothing

Saving the weights

The intuition behind saving the weights after each iteration (update to the weights) is to be able to go back in through the iterations and pick the best weights. The training takes approximately 3-6 hours depending on the chosen parameters. To make sure that the iteration with the best weights haven't "passed", it should be able to go back and pick the desired weights.

The best weights can be chosen by finding the iteration where the validation loss starts to increase while the training loss decreases.

Freezing the network

Freezing a layer or a subset of the neural network is about controlling how the weights should be updated; it means that the subset cannot be modified during training, and it is usually done layer-wise (in contrast to dropout). This technique provides a trade-off between computational time and accuracy.

However, some entries like [51] on the Kaggle challenge have had great success with Mask R-CNN. An investigation into their code shows the applied approach chosen was freezing a subset of the network until the network cannot learn any further, and then training the entire network.

The intuition behind this is that, by applying transfer learning, the network has learned low-level feature extractions, while the head of the network has where classification, box-regression, and mask prediction happens, and which relies on information further down the chain, will perform worse. This approach can limit the search space, and can then fine-tune the entire network. The experiments regarding this were carried out as following:

- Training parts of the network.
- Training entire network.

Figure 4.6: Optimal weights after training

- Training parts of the network + fine-tuning by training the entire network.

Transfer learning and backbone

As explained, Transfer learning refers to giving the network a ”head start”; the weights and biases from the COCO dataset are applied to initialize the Mask R-CNN network. In the case of the backbone, ResNet50 and ResNet100 were employed through experimentation.

4.3.3 Hyperparameter tuning

The weights and biases are derived via training, but parameters that cannot be obtained by training are called hyperparameters. These are used to control the learning process. Even though it was not explicitly stated earlier in this thesis, the learning rate is a hyperparameter. The user chooses this parameter in contrast to the weights and biases. Another known hyperparameter is the batch size. There are techniques to obtain the best hyperparameter, as an optimization algorithm like Particle Swarm Optimization or Genetic algorithm. Probably, the most known in machine learning is grid search, which iterates through every combination and stores a model for each combination. This approach is intuitive, and works well for function estimation in regression or simple artificial neural network grid search [10]; however, it is not freezable for network architectures like CNN

and, especially, Mask R-CNN, as this experimentation uses 3-6 hours to train. Proper tuning of the hyperparameters is necessary to achieve optimal results. To tune the hyperparameters, it is essential to know their effect on the overall prediction.

Batch size

Using the entire dataset before updating the gradient makes the calculated gradient precisely aligned with the true one ;like, for instance, the case of gradient descent, which was discussed in chapter two. As the entire dataset is employed to calculate the gradient, this method is less prone to randomness compared to using a subset (mini-batch) of the dataset. Using a mini-batch is affected by the randomness of selecting the batch each iteration. As a result, using the entire dataset means taking smooth steps towards the globally optimal of the cost function. This results with the cost of the entire dataset having to be retained in memory.

Depending on the size of the mini-batch, its learning is exposed to more or less randomness. The smaller the batch-size is, the greater the randomness resulting. However, if choosing an optimal batch size, they train much faster than full batch learning. Also, picking a very high batch-size on a challenging problem (i.e., image recognition) is known to cause significant degradation in the quality of the model. The inability of generalization occurs because of full-batch converges (gets stuck) in sharp local minimums of the training functions loss [36].

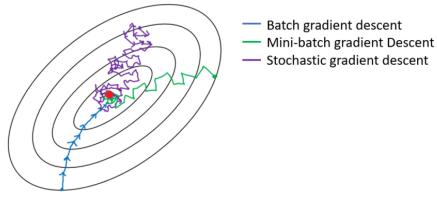


Figure 4.7: Gradient decent on entire data set vs. Mini-batch vs. Single training example

Anchor size and ratio

Anchor boxes, are the generated boxes used for sliding over the image for further processing for the RPN. The boxes are generated in different sizes and aspect ratios.

The size ranges should reflect the problem that shall be slowed through the dataset. Choosing a large range of scales (i.e. [8,16,32,64]) for a dataset consisting of a small object of 8px by 8px would mean more unnecessary computation when generating the larger anchor boxes. More accurate predictions can also be made by choosing the appropriate ratios. If the dataset consists of long and narrow objects, wider aspect ratios (i.e., 1:2, 1:3) should be applied in order to better capture the objects.

Region Of Interest per image

This integer number indicates the maximum number of Regions Of Interest the region proposal network can generate. When training on an annotated dataset, it should be possible to extract the maximum number of instances on the images; then, assigning the "ROI per image" to it can help in the reduction of False Positives and training time. However, if the number of instances is not known in the application of the model (i.e people detection in Times square), this number should be put to a reasonably high number.

Detection threshold

Confidence level threshold is a way to tell how certain the network should be to consider the detection a True Positive. If detecting all of the faults is of the highest importance, it can be lower; however, this comes with the cost of False Positive generation. If the accuracy of the detection is essential, then increase the threshold to minimize false positives and ensure only proper quality detection.

Image size

Image size is controlled by two integer values that control the width and height. The default value is 1024x1024, but smaller images can reduce the memory requirement and training time. However, downscaling the images will result in the loss of potentially essential details.

Weighted loss functions

Mask R-CNN predicts box locations, class of the object, and each pixel corresponding to the object (mask); and, the loss function is the weighted sum of the different losses at each stage of the model. A weight is assigned to the different losses corresponding to a specific loss in the main stages of the network [11].

- **Region proposal classification loss:** This loss consists on the improper classification (foreground, background) on the proposals by the Region Proposal Network. Increasing this hyperparameter, leads the network to emphasize on generating better proposals, and should be only increased when the RPN does not detect objects.
- **Region proposal box loss:** This corresponds to the localization loss for the region proposals. This weight is to be tuned in case that the object is detected, but the bounding boxes are misaligned.
- **Mask R-CNN classification loss:** Indicates an improper classification done after the region of interest passthrough the fully connected layers, which is classification of objects that are present in the region proposal. This is to be increased if the localization of the object is precise but misclassified.
- **Mask R-CNN box loss:** This is assigned to the localization of the final output. It is to be increased if classified correctly, but the localization of the object is not precise.

- **Mask R-CNN mask loss:** This implies the accuracy of the identification of every pixel belonging to the object. This is to be increased if localization and classification are precise, but the mask generated is not.

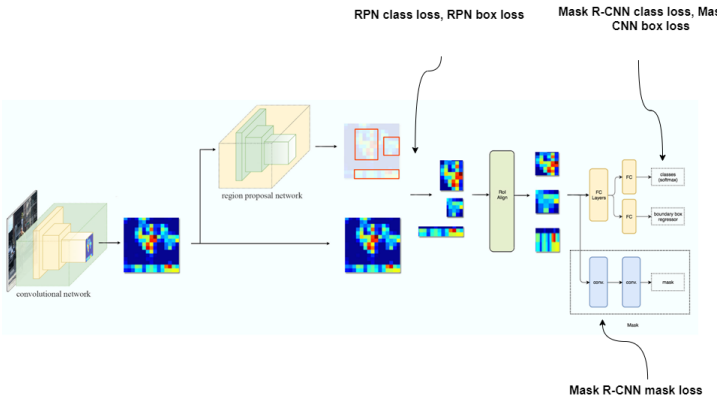


Figure 4.8: Mask R-CNN specific losses [11]

4.4 Evaluation

Single-stage and two-stage detectors were compared in chapter three, and the performance of different models was introduced. The metric employed to evaluate the models is mean Average Precision (mAP). Competitions such as PASCAL VOC, ImageNet and COCO have exclusively adopted average precision as their performance metric. As a result, mean average precision has become the most used metric when evaluating segmentation and object detection models. This is calculated by computing the Average Precision (AP) for each class in the dataset (i.e., missing bolt, or damaged foot) and then average between all the classes; hence, the name mean Average Precision.

Object detection and other categories of algorithms in computer vision, are not meant straight forward to evaluate the performance. In the case of object detection, location and classification indicated the performance of the output. What if the bounding box is predicted correctly but not the class, or the other way around? What AP solves is the evaluation if the network's performance is given a single number. A single number makes it trivial to assess the performance, especially when the algorithm performs well in one of the subtasks while lacking in others.

The AP is dependent on precision and recall. The underlying metric's precision, measures the quality (accuracy) of the predictions. In other words, how many of the predic-

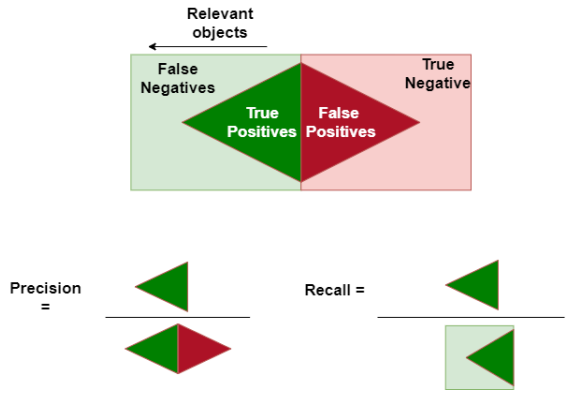


Figure 4.9: Precision and recall

tions were relevant.

$$Precision = \frac{TP}{TP + FP} \tag{4.1}$$

In the equation above TP (True positive) is the total numbers of correct predictions and FP (False positive) it the total number of incorrect predictions. In contrast, recall measures the models ability of detecting the relevant cases among all the data.

$$Recall = \frac{TP}{TP + FN} \tag{4.2}$$

A perfect model would have both recall and precision close to 1. Whereas a high precision meaning that, when the neural network makes a prediction, it is most likely to be correct. A high recall indicates that the model detects close to all objects in the dataset.

In the case of classification, it is trivial knowing when the algorithm predicts wrong. i.e., the model predicted a cat but should've predicted a dog. However, in the subtasks of Mask R-CNN like object detection and mask prediction, it is a bit more complicated. Perfect bounding boxes where each corner perfectly matches the bounding boxes of the dataset is almost impossible to reproduce. It is up to the user to decide when the predicted box is a positive prediction. In the case of object detection, this is done by setting a threshold to the comparison between the prediction and the ground truth. The comparison is made by using IOU.

$$IoU = \frac{area\ of\ overlap}{area\ of\ union} = \tag{4.3}$$

For each object detection, if the IOU values are higher than the set threshold and the predicted classes are correctly labelled, then the prediction is set to true positive. On the other hand, if the predictions fall short to meet the threshold value, the prediction is deemed false positive. False negative is measured by the presence of an object in the dataset or ground truth that was not detected. For a certain IOU, precision and recall

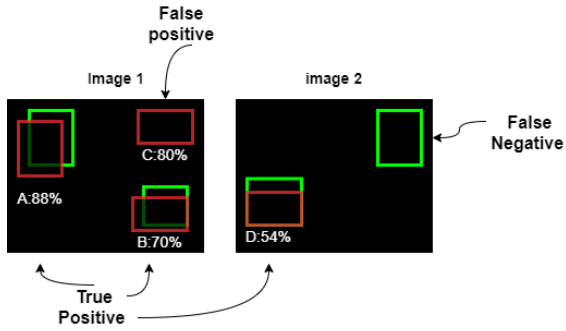


Figure 4.10: TP, FP, FN given threshold of 0.5

can be computed for each class in the dataset by, first, determining true positive, false positive, and false negative of the predictions; then, sorting by the confidence distributed prediction with the corresponding type of prediction (TP or FP); and, finally, calculating a pair for precision-recall for each rank. I.e. rank B's precision-Recall pair is calculated by proportion of TP $2/3 = 0.67$, and proportion of TP out of the possible positive ground truth $2/4 = 0.5$. The AP for a specific class (i.e., missing bolt, damaged foot)

Rank	Correct?	Precision	Recall
A	Yes	1	0.25
C	No	0.5	0.25
B	Yes	0.67	0.5
D	Yes	0.75	0.75

Table 4.1: Complementary table for AP calculations [30]

is calculated through precision-recall pairs, by measuring the area under their curve. To measure the area under the key-value pairs, the precision value is interpolated by equation 4.4.[30]

$$p_{interp}(r) = \max_{\tilde{r} \geq r} p(\tilde{r}) \tag{4.4}$$

Finally, the AP can be calculated by equation 4.5 to measure the area under the curve. The mAP is the average of each class' AP; and it is also listed in the comparison of the different object detection algorithms in chapter three, the metric 0.5:0.95. This metric is determined by calculating the mAP for 10 different thresholds between 0.5 and 0.95 and averaging them. However, in this thesis, threshold of 0.5 has been exclusively used.

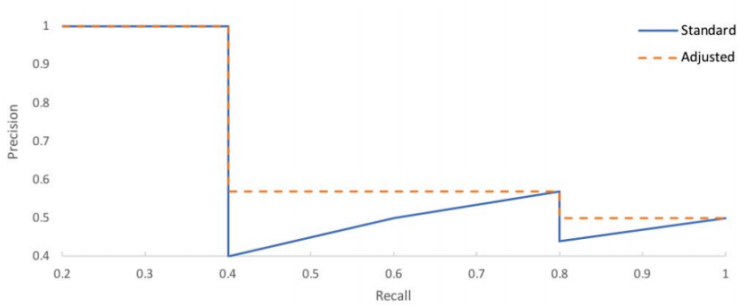


Figure 4.11: Interpolated precision-recall curve [30]

$$AP = \int_0^1 p_{interp}(r)dr \tag{4.5}$$

Chapter 5

Experiment and results

The current chapter, explains the different experiments, and all the approaches are for incremental changes to get the most out of the network's performance. The evaluation of the models were stated in the fourth chapter, and are performed on the untampered test dataset.

5.1 Assumptions

From the understanding of how the Mask R-CNN and its components work, it is viable to make educated assumptions, in order to focus on the most significant experiments.

1. Due to a lack of quality data, the most significant approach for increasing the performance would be the use of transfer learning to utilize the information extraction of a previously trained Mask R-CNN on a vast dataset.

2. Allowing the network to generate small anchors will significantly increase the performance in the detection of missing bolts.

3. Allowing the network to generate anchors with wider ratio will increase the performance of images of a damaged railing.

4. A combination of training parts of the network and finetuning by training the entire network would be the best approach for further increasing the performance across all classes. This is because limiting the search space allows the network to learn valuable information extraction before attacking a harder problem.

5. Generating extra data by augmenting the training dataset will be essential to combat the lack of data, and to increase the performance across all classes.

6. ResNet100 as the backbone, will outperform ResNet50 as presented in their paper, ResNet architecture is robust against vanishing gradients.

5.2 Baseline architecture

The purpose of this section is to make a baseline Mask R-CNN model as a reference point. All other modifications to the network will be compared to the baseline model. In this way, the improvements of each modification, and their effect on the different branches of the network can be isolated. The model will be trained with the following configuration.

Configuration	
Max Iterations	30
Anchor sizes	64,128,256
Anchor ratios	1:1, 1:2, 2:1
Backbone	ResNet50
Detection threshold	90%
Image size	1024x1024
Batch size	2
Validation steps	75%
Schedule	Head
Augmentation	No
Transfer Learning	No
Weighted loss adjusted	No

Table 5.1: Configuration for baseline model

The dataset details are as follows. The same dataset has been used on all of the different experiments in this chapter.

Dataset details				
Dataset	# Missing bolt	# Damaged foot	# Wrong end	# Damaged railling
Training	200	200	200	200
Validation	50	50	50	50
Testing	100	100	100	100

Table 5.2: Configuration for baseline model

5.2.1 Results

The training iterations were set to a maximum of 30, and the training stopped after 17 iterations as the model couldn't learn any further (the dataset was used up). The optimal weights were located at iteration 13 and used for evaluating the model. The detection results for the different classes are shown in table 5.3. Also, an prediction example is shown in figure 5.1

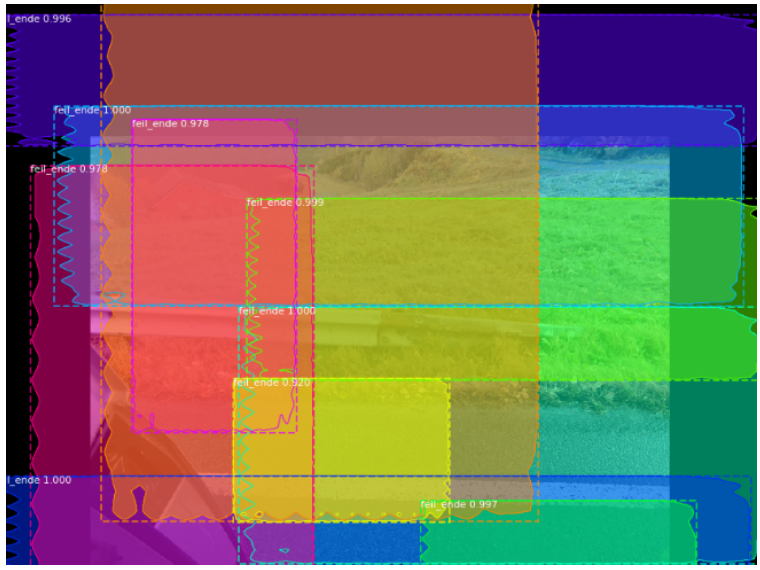


Figure 5.1: Prediction made by the baseline model

Detection results					
Metric	Missing bolt	Damaged foot	Wrong end	Damaged railling	Testset
AP	~ 0	~ 0	0.1	~ 0	0
Recall	~ 0	~ 0	0.1	~ 0	0
Precision	~ 0	~ 0	0.1	~ 0	0

Table 5.3: Detection result for baseline model

5.3 Transfer learning with Mask R-CNN

This section presents the effects of transfer learning in the detection results on the dataset. All of the configurations are initialized as the baseline model, excepting the weights and biases, that are initialized from a previously trained Mask R-CNN model on the COCO dataset. The learning rate had to be adjusted through experimentation to find the optimal achievable weights with the current configuration. The configuration is shown in the table below.

Results

The full extent of the dataset was used after 24 iterations; and, during the following iterations, the model started to overfit. Figure 5.3 illustrates an example of the detection result. Usually, these models make many detections with low confidence (i.e., missing bolt 30%). These detections are filtered out by a confidence filter set by the user. Figure 5.2 shows the predictions before the filtering is applied to the detection. Meaning that the detections that the model keeps, are because it is more than 90% certain that it is correct.

Configuration	
Max Iterations	30
Anchor sizes	64,128,256
Anchor ratios	1:1, 1:2, 2:1
Backbone	ResNet50
Detection threshold	90%
Image size	1024x1024
Batch size	2
Validation steps	75%
Schedule	Head
Augmentation	No
Transfer Learning	Yes
Weighted loss adjusted	No

Table 5.4: Configuration for model with transfer learning

The regions generated by the RPN are illustrated in figure 5.4, and the final detection results can be found in table 5.5.

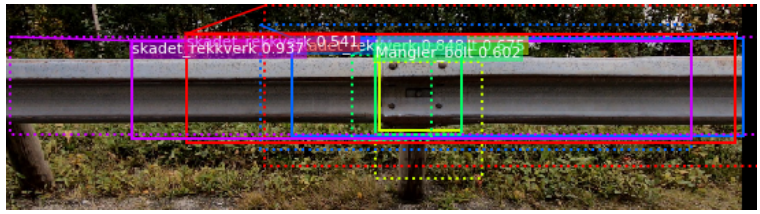


Figure 5.2: Prediction made by the transfer model without confidence filtering

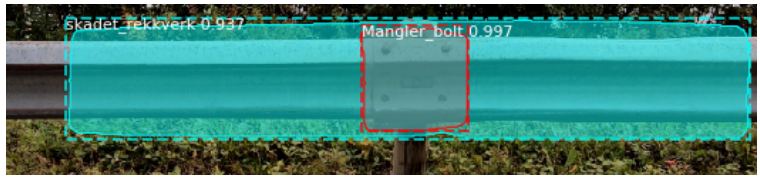


Figure 5.3: Prediction made by the transfer model with confidence filtering

5.4 Split training schedule

In previous experiments, parts of the network were frozen, and only the heads of the network (RPN, classification, mask, and box-regressor) were trained. However, this experiment explored training heads compared to the entire network, and a combination of training both the heads of the network and the entire network.

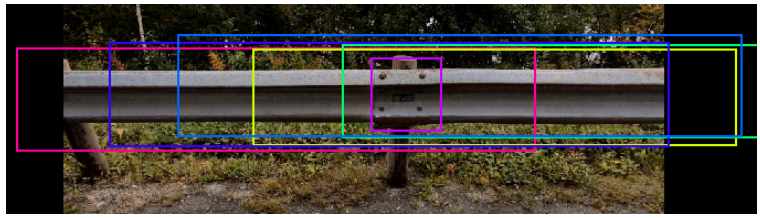


Figure 5.4: Region proposals

Detection results					
Metric	Missing bolt	Damaged foot	Wrong end	Damaged railling	Testset
AP	0.30	0.26	0.84	0.26	0.43
Recall	0.36	0.33	0.86	0.31	0.44
Precision	0.23	0.22	0.84	0.19	0.38

Table 5.5: Detection result for Mask R-CNN with transfer learning

In the section that presents the results, training the entire network from scratch is excluded, for no interesting observations were made, and the model's performance was very similar to training only the heads of the network (baseline model) from scratch.

5.4.1 Results

Baseline model training schedule

The baseline model was trained similarly to the process explained in section 5.1; but, after 13 iterations, where approximately the optimal weights could be found, the model was trained for five more iterations, where the model could train the entire network. Some minor modification needed was that, when starting to train the whole network, the learning rate set for training the head was too large and had to be reduced by a factor of ten.

Table 5.7 presents the detection results of the model, and figure 5.5 shows the final region proposals where the RPN is more than 70% confident about the areas that contain an object, and these proposals will be further processed by the model. Finally, figure 5.6 and 5.7 show the detection done by the model and the detections that are kept after confidence filtering.

Transfer learning with split training schedule

Similarly to the transfer learning model train in section 5.2, the model was frozen, with the exception of the heads, and after 24 iterations, the model was trained for another 15 iterations. The model's configuration is listed in table 5.7; also, this model had to reduce

Configuration	
Max Iterations	30
Anchor sizes	64,128,256
Anchor ratios	1:1, 1:2, 2:1
Backbone	ResNet50
Detection threshold	90%
Image size	1024x1024
Batch size	2
Validation steps	75%
Schedule	Head + Entire network
Augmentation	No
Transfer Learning	No
Weighted loss adjusted	No

Table 5.6: Configuration for model with transfer learning

Detection results					
Metric	Missing bolt	Damaged foot	Wrong end	Damaged railling	Testset
AP	0	0	0	0	0
Recall	0	0	0	0	0
Precision	0	0	0	0	0

Table 5.7: Detection result for Mask R-CNN train the head + entire network

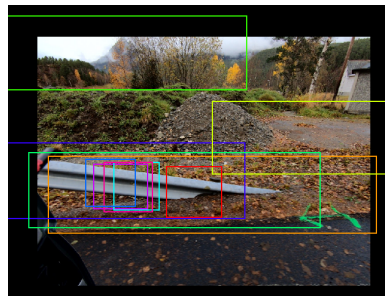


Figure 5.5: Baseline models region proposal

its learning rate by a factor of ten when training the entire model.

Table 5.9 presents the results of the precision, recall, and AP/mAP for the model; and notes, next to the metrics, the changes in performance. The table presents a comparison of the transfer model when only the heads are trained, and when the model head + entire network is trained. Figure 5.8 illustrates the final region proposals made from the RPN.Then, figure 5.9 is a prediction done where the low confident predictions are not filtered; and figure 5.10 shows predictions after filtering low confident predictions.

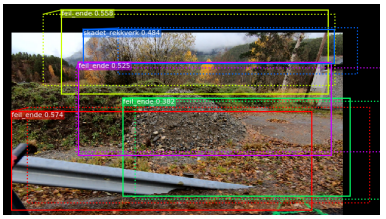


Figure 5.6: Baseline models detections without confidence filtering

Figure 5.7: Baseline models detections with confidence filtering

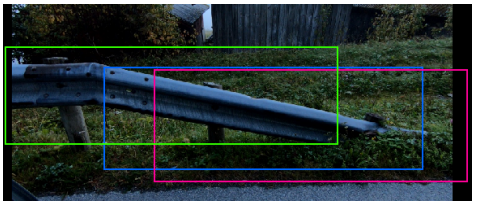


Figure 5.8: Final region proposal from model head+entire network trained

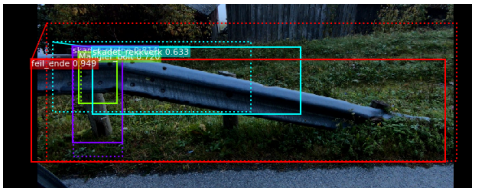


Figure 5.9: Final region proposal from transfer model where head+entire network trained

5.5 Data augmentation

Data augmentation is a technique for producing more data to combat cases where data is insufficient. The images created by the augmentation generator exist for precisely one passthrough of the network. The procedure for data augmentation for the transfer learning model is listed below.

1. Randomly generate a discrete number in an interval between $[x = 0, y = 2]$ and assign it to a variable c.

Configuration	
Max Iterations	30
Anchor sizes	64,128,256
Anchor ratios	1:1, 1:2, 2:1
Backbone	ResNet50
Detection threshold	90%
Image size	1024x1024
Batch size	2
Validation steps	75%
Schedule	Head + Entire network
Augmentation	No
Transfer Learning	Yes
Weighted loss adjusted	No

Table 5.8: Configuration for model with transfer learning

Detection results					
Metric	Missing bolt	Damaged foot	Wrong end	Damaged railling	Testset
AP(%)	40 (+10)	40 (+14)	35 (-49)	36(+12)	40(-3)
Recall(%)	50 (+14)	55 (+22)	36.6 (-49.5)	38(+7)	46 (+2)
Precision(%)	27 (-4)	33 (+11)	35(-49)	38(+19)	35(-3)

Table 5.9: Detection result for Mask R-CNN with transfer learning with split training schedule

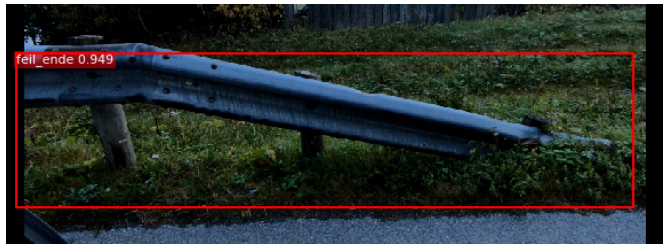


Figure 5.10: Final detection from transfer model where head+entire network trained

2. Arbitrarily choose c augmentations to do from the list below

   (a) In 50% of cases flip the image from left to right

   (b) In 50% of cases flip the image from up to down

   (c) Arbitrarily choose to rotate the image by either $90°$, $180°$ or $270°$

   (d) Alter the brightness of the image by a number d between $[0.8, 1.5]$

Similar augmentation was done to the baseline model; however, a more intense augmentation procedure was chosen, by setting the interval [x=0,y=4] in its first step. The result of this experimentation is listed in table 5.10 and 5.11.

Detection results					
Metric	Missing bolt	Damaged foot	Wrong end	Damaged railing	Testset
AP(%)	20	22	42	3	20.7
Recall(%)	25	26	45	5	26
Precision(%)	18	20	41	2	19.75

Table 5.10: Detection result for baseline model with augmentation

Detection results					
Metric	Missing bolt	Damaged foot	Wrong end	Damaged railling	Testset
AP	37(+7)	34(+8)	75.6(-7.4)	33(+7)	50.5(+7.5)
Recall	44(+4)	43(+10)	77(-9)	40.3(+9.3)	52(+8)
Precision	28(+5)	28.6(+8.6)	75(-9)	24.5(+5.5)	46(+8)

Table 5.11: Detection result for transfer learning model with augmentation

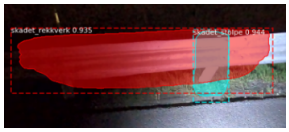
5.6 Final model

Previously in this chapter, the most significant approaches for increasing the performance were presented, like transfer learning, anchor generation, and training part of the network compared to the entire network and a combination. This section presents the consolidation of previous methods that have been experimented with to produce the final model for the company.

The full configuration is listed in table 5.10; then, figure 5.11 shows the predictions that the final model did correctly; figure 5.12 the ones that the model did partially correct; and 5.13 where the model failed.

Configuration	
Max Iterations	70
Anchor sizes	16, 32, 64, 128, 256, 512
Anchor ratios	1:1, 1:2, 2:1, 3:1
Backbone	ResNet101
Detection threshold	90%
Image size	1024x1024
Batch size	2
Validation steps	75%
Schedule	Head + Entire network
Augmentation	Yes
Transfer Learning	Yes
Weighted loss adjusted	Yes

Table 5.12: Configuration for model with transfer learning

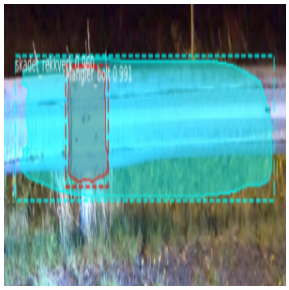


(a) Correctly predicted damaged railing, correctly predicted damaged foot

(b) Correctly predicted damaged foot

(c) Correctly predicted damaged foot

Figure 5.11: Correct predictions



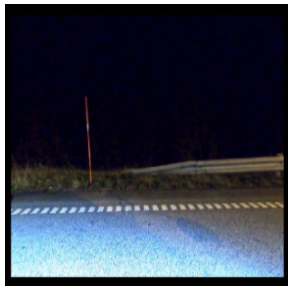
(a) Correctly predicted missing bolt; falsely predicted damaged railing

(b) Correctly predicted damaged railing, falsely predicted damaged foot

(c) Correctly predicted wrong end, falsely predicted damaged foot

Figure 5.12: Correctly predicted wrong end, falsely predicted damaged foot

(a) No prediction on a image where the bolt/bolts are missing



(b) No prediction on an image with wrong end

(c) Falsely predicted wrong end on an image of damaged railling

Figure 5.13: Wrong prediction

Detection results					
Metric	Missing bolt	Damaged foot	Wrong end	Damaged railling	Testset
AP(%)	71	67	84	56	71
Recall(%)	75	71	92	61	77
Precision(%)	67	60	75	51	67

Table 5.13: Detection result for final Mask R-CNN with transfer learning

Anchor boxes

A model was trained with a more diverse anchor generation with and without transfer learning. The model without transfer learning will not be listed in this section, because no significant observations were made from it. The smallest anchor generated had a width of 16, and the largest had a width of 512. The anchors increased in size by two, and the aspect ratios were 1:1, 1:2, 2:1, and 3:1. In table 5.12 is the result of this experimentation.

Detection results					
Metric	Missing bolt	Damaged foot	Wrong end	Damaged railling	Testset
AP(%)	60 (+30)	33.8(+7.8)	67(-17)	39(+13)	48(+7)
Recall(%)	70(+34)	40(+7)	65(-21)	46.5(+15.5)	55.8(+11.8)
Precision(%)	48(+25)	30(+8)	62(-22)	30(+11)	41(+3)

Table 5.14: Detection result for Mask R-CNN with transfer learning and diverse anchor generation

Adjusted loss function and backbone

It was attempted to adjust the loss function. The result shows that adjusting the loss function with respect to the mask loss improves the performance. The performance on the classes ”missing bolt” and ”damaged foot” remained after the change of the loss, but there was an improvement on the classes ”damaged railing” and ”wrong end”. The optimal change to the loss function was 20%; any lower than 20% would worsen the performance again. In table 5.13 the change in performance is listed.

Changing the backbone from ResNet50 to ResNet100 improved the performance after many attempts to use the optimal configuration; however, the observed improvement was negligible.

Improvement on detection results		
Metric	Damaged railing	Wrong end
Δ AP(%)	3	8
Δ Recall(%)	5	10
Δ Precision(%)	2	5

Table 5.15: Improvement on detection result by adjusting the loss function

Chapter 6

Discussion

6.1 Baseline architecture

The basic Mask R-CNN model achieved optimal results after 13 iterations. No adjustment was made to the model, and it didn't manage to archive significant results. The latter was clearly the first sign of insufficient data. The figure in 5.1 showed that the region proposal didn't achieve generating the regions of interest through the training procedure. Interestingly, the model achieved some of the detections; giving the models score of 0.1 in the class "Wrong end", and giving the first sign of the class "wrong end" is the easiest class in the dataset to detect. Moreover, instances in the dataset containing the class "Wrong end," were usually labelled correctly, but the model had no clue about where the fault was located in the image, similar to classification in computer vision. The same figure also shows the model labelling the image correctly, but not managing to localize it in the image.

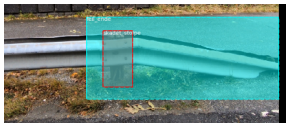
6.2 Transfer learning

Transfer learning model was an experiment where the objective was to repurpose knowledge of a Mask R-CNN model learning to solve a different task, like finding faults in railing. This was the first experiment where the model learned features necessary to solve the task this thesis sought to do.

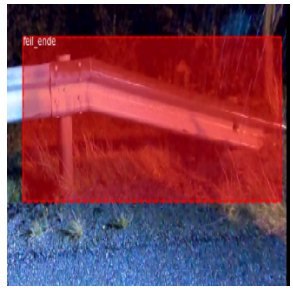
This model achieved a remarkable mAP of 43%, which was a clear indication that some of the transferred knowledge helps in case of limited data. The region proposal networks' generalization abilities far exceed the baseline model. Figure 5.4 illustrates the regions proposed by the model, which shows what the model decided to emphasize on. Every region is capturing some aspect of the railing, and the proposal in purple showing that the model also learned the square section in important regions where the bolts should be.

Although the model has recognized what regions are important, it did not achieve promising results on classifying the region and finding the section where the fault was. Approximately, the model managed to find the regions and classify correctly 25% of the times for ”missing bolt”,”damaged foot” and ”damaged railing”. The quality of the prediction (the precision) was a small incremental worse than its ability to find all the faults (recall). However, the detection results have a significant outlier.

The model achieved an impressive 85% accuracy to find all the images where the railing had a wrong end and label them correctly. It is not trivial, knowing exactly what a deep learning model learns; however, a visual inspection of the dataset was carried out to understand why the model achieved significant results on the images with the wrong end on the railings.



(a) Correctly predicted damaged railing, correctly predicted damaged foot



(b) Correctly predicted damaged foot

(c) Correctly predicted damaged foot

Figure 6.1: Correct predictions of wrong end

The class ”wrong end” can roughly be divided into two subclasses; and, these two subclasses, are represented abundantly with many instances in all three datasets (training, validation, and testing). Figure 6.1.a shows the two subclasses in the first subclass; a horizontal segment of the railing is connected to the inclining segment by an angle. The other subclass is if the railing has a small declining segment towards the ground (Figure 6.1.c).

Training a human to do this detection would be quite trivial, and the reason for that is that the classification of this region is dependent on natural and big features. Similarly, it is trivial for the neural network, as these big features are not easily obstructed by other factors like the brightness or bad resolution of an image, and the image taken from far away.

6.3 Split training schedule

Baseline

In table 5.2, the detection results for the baseline model trained with a split schedule is listed. The model heads were trained until it achieved the optimal results, and then, the entire network was trained.

At first sight, it might seem no improvements were made by using the split training schedule. The model achieved a mAP of 0%, similar to the baseline model without split schedule training. However, the improvements were observed in the region proposals. Contrary to the baseline that was not trained with this schedule, the model neither achieved generating proposals, localizing the object, nor classifying it. Still, the region proposal for this model is far superior compared to the baseline model. Figure 5.5 showed the output from the region proposal network, and the model achieved capturing the railing on the majority of the proposals. Due to the manner the criteria initialized, the model should only output predictions the model is certain of; but figure 5.6 shows that the model made many predictions on that particular instance and even achieved to localize and label the predictions correctly compared to the ground truth. However, the model was too uncertain of its prediction (57%) and was, in the final output, filtered out.

Transfer model

For 24 iterations the transfer model trained only the heads; then, it trained the entire network for 15 iterations. Figure 5.8 depicts the RPN output, that has understood the essential parts of the railing, where it has no abundant proposals. All proposals capture the different parts of the railing, which is valid for all instances in the datasets.

This model achieved a mAP of 40%, which was 3% less than in the transfer model that was trained without split schedule. The model managed to find 46% of all faults in the dataset, which was an improvement of 2 %, but a decrease of 3% in the quality. However, the model achieved better results in all of the classes except for ”Wrong end”, which was a significant outlier in the previous experiment.

Averagely, the model did approximately 10% better at finding all the faults and labelling them correctly (excluding ”Wrong end”). I.e., damaged railing had an increase of 19% in its quality of predictions, and an increase of 7% in finding all faults. Similarly, the damaged foot had a mAP increase of 14%, where the model achieved finding 55% of all faults in the testing set. Also, the model had a precision increase of 11%.

The model achieved approximately the same result as without split schedule training, even with the significant improvements observed; this, due to the bog drop in detection result considering ”wrong end”. It is important to mention that this class was trained optimally before the entire network’s training started; the training regarding this class began to overfit after trained further, giving this class a significant drop in the different

metrics used for evaluation. This caused concern in the project, knowing that the problem's difficulty is reflected in the complexity of the model, and the dataset has the same number of instances but different difficulties.

6.4 Augmentation

Baseline model

The experimentation shows that augmentation is an essential part for achieving a effective model when data is the limiting factor. Even the baseline model achieved mAP above 0 (20.7%) considering all other experiments done on this model. Somewhat, the data generation solved the problem with limited data points for a model.

Again, it is prominent that the ”wrong end” is the easiest class in the dataset. The model achieved an AP of 42 %, where it managed to find 45% of these instances in the test set and a precision of 41%. Similarly, the model achieved to learn features to correctly localize and label approximately 20% of the cases in the classes ”missing bolt” and ”damaged foot”.

The model achieved to learn some patterns to detect damaged railings; however it only accomplished an AP of 3% where it managed to find 5% of all instances in the dataset, correctly classify and localize them 2% of the time. Showing the model did achieve to learn something but not to a significant extent. It is relevant to note that an AP of 3% is far superior compared to an AP of 0%. However, the definition of a damaged railing is wide and could be divided into many subclasses; thererore, a good assumption would be that, as this class is so vast, it is possible that this class's learning can't efficiently utilize the generation of more data as when the image is manipulated, the vast boundary of the defintion of ”damaged railing”, is widened. Also, it is important to mention that when less aggressive augmentation is used, it significantly affects the performance of the other classes, but the class ”damaged railing” seems to be unaffected by it.

Transfer model

The transfer model with augmentation has utilized the generation of more data. The mAP had an increase of 7.5%, and also, averagely, AP had an increase of approximately 7%. Furthermore, like in other experiments regarding the transfer model, this experiment's detection takes a hit for the class ”wrong end”, but not as significant as in the others.

Still, the reason for the drop is that it is starting to overfit. Additionally, the augmentation limits the remembering as the data, meaning the chance of training on an instance the model already trained on, is far less than without the augmentation. Interestingly, the class ”damaged railing” managed to utilize the generation of more data, in the same rate as the other classes with an improvement of 7% AP. This might contradict the previous section's assumption, but it must be mentioned that this model didn't start with no knowledge of the problem it tries to solve. It seems like there is a balance between how

much the network knows before starting and the widening of the definition of ”damaged railing”. The threshold for learning has increased by manipulating images in an already highly diverse class, making it harder to utilize the generation of more data. The transfer learning model has previously learned features and feature extraction, making it easier to overcome the threshold.

6.5 Final model

The final model is a combination of early experiments, but, with minor modifications like generating varied anchors and adjusting the loss function. This model achieved to find 77% of all faults in the test set with a precision of 67%, giving it an mAP score of 71%. This model trained the heads only for 30 iterations, and the entire network was trained for 20 iterations.

The ”damaged railing” was the class with the least impressive results, for it achieved an AP score of 56%, where it managed to find 61% of all instances in the test set and a precision of 51%. Eventhough it achieved much better than previous experiments, it still had some shortcomings. An interesting observation is that the model achieved finding railings with severe damage, like illustrated in the figure below. However, the model struggles to find minor damages. Moreover, this class has inconsistency in the way it is labelled; when labelling this fault, the operator has chosen to label the entire segment of the railing from one foot to the other; then, halfway through the datasets, decided to label only the section where the fault is. This labelling makes the model also struggle with detection, which is technically correct, but comes short in the IOU of 50% threshold the model was initialized with. There are also special instances that the model struggles with achieving high enough confidence level, like in figure 6.2 b. The figure shows an image of a railing from an angle, the false negative is understandable given that these instances are not well represented in the training set.

(a) Correct detection with IOU less than 50%

(b) False negative

(c) Sever

Figure 6.2: Severe damage to the railing

The ”missing bolt” and ”damaged foot” classes are rather aggressive, which reduces their precision score. Interestingly, when the image is taken from far, the model does not output these classes, but when the image is unclear, the model predicts these classes with

confidence. Besides, the ”damaged foot” class seems to be predicted far more correctly when the foot is made from wood than metal. This case would be true for humans also, as the tree foot cracks while the metal foot gets bent, which makes it easier to detect in an image.
From the previous experiments, the class ”missing bolt” has a huge spike in its metric scores. As depicted in table 5.14, the class has an increase of 30% in AP. The reason is that the model can now use smaller, semantically richer feature maps to detect this class when generating smaller anchor boxes. The final model (table 5.13) had another increase in AP with 11% showing that the model could utilize the generation of more data with extracting more certain regions from the feature maps.

Adjusting the loss function seemed to only affect the classes' ”damaged railing” and ”wrong end”. A feasible assumption for this would be that this is due to the generation of the mask; which is generated from the bounding boxes. Damaged foot and missing bolt have very accurate labelled bounding boxes. Generating the mask from these bounding boxes has a small error compared to the original mask that the model should use during training; however, this is not true for the classes ”wrong end” and ”damaged railing.” The bounding boxes the operator labelled cover a lot of the surroundings of the faults (i.e., grass, or buildings). Reducing the loss for the mask is a way to tell the model that it should emphasize on the bounding boxes and not the masks.

6.6 Uncertainty and limitation

In chapter four and five, respectively, detection result from the different experiments was presented, as well as the evaluation of the model. The definition of the mAP is defined as the mean over the Average Precision for each class. However, when the number for the evaluation metrics for each class was generated, images with multiple instances were filtered out (approx. 25% of the dataset). The generation for the metrics to do the evaluation was not done in a costly cloud server, but on a local computer without a dedicated GPU. The evaluation for a single experiment took several hours without GPU from the experimentation done in this thesis. Each class was compared to each other throughout the different experiments; but, this should not affect the analysis, as the main concern was the improvement/decrease in the evaluation metrics. Nonetheless, the evaluation for the entire dataset was done calculating its AP (mPA), and provided a good indication of the previous calculations accuracy.

Another critical fact to discuss is the class ”missing foot” that was removed from the datasets; as the instances in the dataset which had a missing foot, the model couldn't learn at all. It seems that it is too much to ask from the network. to find the absence of an object. This might sound like a contradiction, because then, how is it possible for the model to find instances of a missing bolt? The model doesn't really find the area that is missing a bolt, but rather a hole where the bolt should've been. Given the figure 6.3, the operator probably did some calculations like the length between two poles, the height of the pole relative to the ground, to understand where the pole should've been. In other

words, how do you tell the model that the annotated region of interest is dependent on two different parts of the image? And then ask it to calculate the height of the railing.

Figure 6.3: Missing foot

One last interesting finding that evidences the limitation of the model is the instance in figure 6.4. Because of the way the model was trained (data of ”wrong end” fed to the model), the prediction in green is technically correct, but by the law and regulation criterion, it is not. The definition of ”wrong end” in a railing, by new regulations, is not allowed because the railing works as a ramp. With this definition, the model predicted correctly, then why did not the operator label it? Well, the operator is an expert in this field; given that the image is taken from a car, this is certainly from a traffic road (the one in blue). While the railing predicted in green is in another category of a road (walkway), these roads have another set of laws and regulations, which does not dictate railing with a declining endpiece is prohibited; so, to conclude this, the model can predict accurately from visual cues, but the fact it does not know Norwegian laws and regulations for safety on the roads makes this image especially hard to predict correctly.

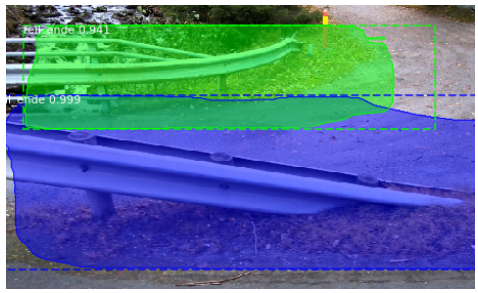


Figure 6.4: Questionable prediction

Chapter 7

Conclusion

This thesis presented the research carried out to implement and prepare the state-of-the-art-model based on literature review, and the task the company wishes to solve. The most common fault is ”missing bolt”, according to the company and vegtilsynet [76]; and, given that accuracy is the highest priority, the network choice was Mask R-CNN (He et al. [27]).

A lot has changed since the project started, for images were a labor-intensive task to capture and annotate for the company, which was coordinated by the author. The company executed this task by capturing images with a GoPro camera and annotating them with VGG [18]. However, they also got support from Innovation Norway (Innovasjon Norge), which give them the resources to implement a project more representative to the task that shall be solved in the future.

There were certain limitations observed as the research and experimentation was carried out; especially the limited data, and inconsistency of labelling. With the newfound resources and support, the company will install a camera system on the car, and obtain far more data than was available throughout the thesis. In addition, the company is currently working on making a software where more relying labelling is proposed. The intention is that an image has to be annotated by several operators, and those annotations must have an IOU above 90% to be considered a fault, similar to True Positive definition in object detection from chapter 4.

The final model achieved a promising $mAP_{iou=50} = 71\%$, where it managed to find 77% of all faults in the dataset and predict correct labels 67% of the time. It managed to find an essential region of the image, emphasizing the areas containing the railing, and part of the railing where the classes of faults are usually located. As stated in the thesis, driving between $1 - 15km/h$ is neither safe nor efficient. The limiting factor for the driving speed is solely dependent on the chosen cameras' ability to capture images, frequency of the lidar, and GPS. Also, the economic aspect hasn't been stated either. The

model can run $6FPS$ for each 6GB video memory; which means that, a server instance of 32 GB is needed to match 30FPS. The hour cost in the server is 4.56 dollars per hour [9] (worst case, high demand), making a factor of 5 times cheaper than the per hour rate of skilled workers. Also, the car would drive 2x faster(worst case).

However, the test data in which the model was trained on is insufficient to represent the full extent of the problem. This was clear even before starting this thesis; yet the limiting factors, a successful model was trained. Even though the problem isn't completely solved, the model is the right step towards solving it, and the network is a great starting point for the company. The weight of this model (and the model itself) has been stored for its use in transfer learning. The approaches are clearly explained (chapter 4), and their effects on the class prediction accuracy (chapter 5, 6), such that the company or other parties can implement the model to their specific need or for modification. From experimentation, transfer learning has proven to be an essential part of finding faults in railings; and, now it does exist a network with pre-knowledge of detecting faults in railings.

7.1 Future work

This thesis presented a proof of concept that object detection algorithms can aid the workers in keeping the Norwegian roads safe. The list below suggests work related to this thesis that should be investigated in a close future.

- It is apparent that some classes are harder to detect than others, as discussed in chapter 6. An interesting approach to this would be implementing **sampling strategies**; i.e., in each training iteration, higher percentage of difficult classes are trained for overcoming clasewise overfitting.

- Another approach to the variety of difficulties between classes would be similar to reducing mask loss to make the network emphasize on the bounding box predictions; however, the implementation would be for defining a **loss for each class** for a given dataset.

- As mentioned in the scope, this thesis is one part of a larger project. The metadata of the images will be annotated with GPS location, considering that a working model will eventually be developed, and faults in railings across Norway documented extensively. Analysis of the results can be made moving towards **predictive maintenance**.

- One class that the model didn't achieve to detect was "missing foot". The company decided to use height obtained by the lidar to detect "missing foot" faults, but this might also be obtainable through other means.

Bibliography

[1] , a. Build from source : Tensorflow. URL: https://www.tensorflow.org/install/source.

[2] , b. Build from source on windows : Tensorflow. URL: https://www.tensorflow.org/install/source_windows.

[3] , . Cs231n: Convolutional neural networks for visual recognition. URL: http://cs231n.stanford.edu/.

[4] , . Meld. st. 33, nasjonal transportplan 2018–2029. Statens vegvesen. URL: https://www.regjeringen.no/contentassets/7c52fd2938ca42209e4286fe86bb28bd/no/pdfs/stm201620170033000dddpdfs.pdf.

[5] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X., 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. URL: https://www.tensorflow.org/. software available from tensorflow.org.

[6] Abdulla, W., 2017. Mask r-cnn for object detection and instance segmentation on keras and tensorflow. https://github.com/matterport/Mask_RCNN.

[7] Abdulla, W., 2018. Splash of color: Instance segmentation with mask r-cnn and tensorflow. URL: https://engineering. matterport. com/splash-of-color-instance-segmentation-with-maskr-cnn-and-tensorflow-7c761e238b46 .

[8] Albawi, S., Mohammed, T.A., Al-Zawi, S., 2017. Understanding of a convolutional neural network, in: 2017 International Conference on Engineering and Technology (ICET), pp. 1–6.

[9] Amazon, . Amazon EC2 G3 Instances, year = 2020, url = https://aws.amazon.com/ec2/instance-types/g3/,.

[10] Bergstra, J., Bengio, Y., 2012. Random search for hyper-parameter optimization. Journal of machine learning research 13, 281–305.

[11] Bobba, R., . Taming the Hyper-Parameters of Mask RCNN, year = 2019, url = https://medium.com/analytics-vidhya/taming-the-hyper-parameters-of-mask-rcnn-3742cb3f0e1b, urldate = 2019-12-14.

[12] Burkov, A., 2019. The Hundred-Page Machine Learning Book. Andriy Burkov, author@themlbook.com.

[13] Bushaev, V., . Stochastic Gradient Descent with momentum, year = 2017, url = https://towardsdatascience.com/stochastic-gradient-descent-with-momentum-a84097641a5d, urldate = 2017-12-04.

[14] de Castro, L.N., 2007. Fundamentals of Natural Computing: Basic Concepts, Algorithms, and Applications.

[15] Chollet, F., et al., 2015. Keras. https://keras.io.

[16] Dalal, N., Triggs, B., 2005. Histograms of oriented gradients for human detection, in: 2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05), IEEE. pp. 886–893.

[17] Dietterich, T., 1995. Overfitting and undercomputing in machine learning. ACM computing surveys (CSUR) 27, 326–327.

[18] Dutta, A., Zisserman, A., 2019. The VIA annotation software for images, audio and video, in: Proceedings of the 27th ACM International Conference on Multimedia, ACM, New York, NY, USA. URL: https://doi.org/10.1145/3343031.3350535, doi:10.1145/3343031.3350535.

[19] Everingham, M., Van Gool, L., Williams, C.K., Winn, J., Zisserman, A., 2007. The pascal visual object classes challenge 2007 (voc2007) results .

[20] Fridman, L., . lecture: MIT Deep Learning Basics: Introduction and Overview, year = 2020, url = https://deeplearning.mit.edu/, urldate = 2020.

[21] Gandhi, R., 2018. R-cnn, fast r-cnn, faster r-cnn, yolo object detection algorithms. july 9, 2018. Retrieved September 20, 2019.

[22] Garbade, M.J., 2018. Clearing the confusion: Ai vs machine learning vs deep learning differences. URL: https://towardsdatascience.com/clearing-the-confusion-ai-vs-machine-learning-vs-deep-learning-differences-fce69b21d5eb.

[23] Girshick, R., 2015. Fast r-cnn, in: Proceedings of the IEEE international conference on computer vision, pp. 1440–1448.

[24] Girshick, R., Donahue, J., Darrell, T., Malik, J., 2014. Rich feature hierarchies for accurate object detection and semantic segmentation, in: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 580–587.

[25] Group, M.V.R., 2019. Mask r-cnn unmasked. URL: https://medium.com/@fractaldle/mask-r-cnn-unmasked-c029aa2f1296.

[26] Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., Liu, T., Wang, X., Wang, G., Cai, J., et al., 2018. Recent advances in convolutional neural networks. Pattern Recognition 77, 354–377.

[27] He, K., Gkioxari, G., Dollár, P., Girshick, R., 2017. Mask r-cnn, in: Proceedings of the IEEE international conference on computer vision, pp. 2961–2969.

[28] He, K., Zhang, X., Ren, S., Sun, J., 2015. Deep residual learning for image recognition. arXiv:1512.03385.

[29] Hochreiter, S., 1998. The vanishing gradient problem during learning recurrent neural nets and problem solutions. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 6, 107–116.

[30] Hui, J., 2018. map (mean average precision) for object detection. Jonathan Hui .

[31] Hui, J., 2020. Understanding feature pyramid networks for object detection (fpn). URL: https://medium.com/@jonathan_hui/understanding-feature-pyramid-networks-for-object-detection-fpn-45b227b9106c.

[32] Isikdogan, L., . Regularization, year = 2018, url = https://www.isikdogan.com/blog/regularization.html, urldate = 2017-12-04.

[33] Izadpanahkakhk, M., Razavi, S.M., Taghipour-Gorjikolaie, M., Zahiri, S.H., Uncini, A., 2018. Deep region of interest and feature extraction models for palmprint verification using convolutional neural networks transfer learning. Applied Sciences 8, 1210.

[34] Jung, A., 2019. Imgaug documentation. Readthedocs. io .

[35] Kapoor, A., 2019. Clearing the confusion: Ai vs machine learning vs deep learning differences. URL: https://hackernoon.com/deep-learning-vs-machine-learning-a-simple-explanation-47405b3eef08.

[36] Keskar, N.S., Mudigere, D., Nocedal, J., Smelyanskiy, M., Tang, P.T.P., 2016. On large-batch training for deep learning: Generalization gap and sharp minima. arXiv:1609.04836.

[37] Krizhevsky, A., Sutskever, I., Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks, in: Advances in neural information processing systems, pp. 1097–1105.

[38] Kukačka, J., Golkov, V., Cremers, D., 2017. Regularization for deep learning: A taxonomy. arXiv preprint arXiv:1710.10686 .

[39] Law, H., Deng, J., 2018. Cornernet: Detecting objects as paired keypoints, in: Proceedings of the European Conference on Computer Vision (ECCV), pp. 734–750.

[40] LERAAN, O., ENGAN, S., 2018. Sp om fiksing av farlige rekkverk: – det kan ikke utsettes! VG.

[41] Lin, T.Y., Dollár, P., Girshick, R., He, K., Hariharan, B., Belongie, S., 2017a. Feature pyramid networks for object detection, in: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 2117–2125.

[42] Lin, T.Y., Goyal, P., Girshick, R., He, K., Dollár, P., 2017b. Focal loss for dense object detection, in: Proceedings of the IEEE international conference on computer vision, pp. 2980–2988.

[43] Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L., 2014. Microsoft coco: Common objects in context, in: European conference on computer vision, Springer. pp. 740–755.

[44] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.Y., Berg, A.C., 2016. Ssd: Single shot multibox detector, in: European conference on computer vision, Springer. pp. 21–37.

[45] Long, J., Shelhamer, E., Darrell, T., 2015. Fully convolutional networks for semantic segmentation, in: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 3431–3440.

[46] Løtveit, S., Ådlandsvik, L.F., Gullbrå, E.H., Oksnes, T., Sandvik, T.F., Midtgård, F., Grytli, T., Kvanvik, M., Munch-Olsen, Y., Tronsmoen, T., Hendbukt, M., 2017. Nasjonal tiltaksplan for trafikksikkerhet på veg 2018-2021. Statens vegvesen.

[47] M., V., . Artificial Intelligence vs. Machine Learning vs. Deep Learning, year = 2018, url = https://www.datasciencecentral.com/profiles/blogs/artificial-intelligence-vs-machine-learning-vs-deep-learning, urldate = 2018-05.

[48] Mané, D., et al., . Tensorboard: Tensorflow's visualization toolkit, 2015.

[49] Marsland, S., 2009. Machine Learning: An Algorithmic Perspective.

[50] Mesquita, D., . How to use NVIDIA GPUs for Machine Learning with the new Data Science PC from Maingear, year = 2019, url = https://towardsdatascience.com/how-to-use-gpus-for-machine-learning-with-the-new-nvidia-data-science-workstation-64ef37460fa0, urldate = 2019-10.

[51] mirzaevinom, 2018. data science bowl 2018. `https://github.com/mirzaevinom/data_science_bowl_2018`.

[52] Mulonda, Y., . What is Machine Learning? “In Simple English”, year = 2018, url = https://medium.com/@yannmjl/what-is-machine-learning-in-simple-english-b0aaa251cb60, urldate = 2018-11-17.

[53] Nada, Berchane, N., . Artificial Intelligence, Machine Learning, and Deep Learning: Same context, Different concepts, year = 2018, url = https://master-iesc-angers.com/artificial-intelligence-machine-learning-and-deep-learning-same-context-different-concepts/, urldate = 2018-0403-16.

[54] Ng, A., 2017. Lecture:transfer learning (c3w2l07). deeplearning.ai. URL: https://www.youtube.com/watch?v=yofjFQddwHE&t=1s.

[55] Ng, A., Katanforoosh, K., 2018. Stanford cs230: Deep learning autumn 2018 lecture 2 - deep learning intuition. URL: https://www.youtube.com/watch?v=AwQHqWyHRpU&list=PLoROMvodv4rOABXSygHTsbvUz4G_YQhOb.

[56] Nielsen, M., Michael Nielsen. Neural Networks and Deep Learning.

[57] NRK, 2018. Feil autovern kan ha bidratt til at 18-åring omkom. URL: https://www.nrk.no/vestland/feil-autovern-kan-ha-bidratt-til-at-18-aring-omkom-1.14296659.

[58] Pan, S.J., Yang, Q., 2009. A survey on transfer learning. IEEE Transactions on knowledge and data engineering 22, 1345–1359.

[59] Prabhu, R., . Understanding of Convolutional Neural Network (CNN) — Deep Learning, year = 2018, url = https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148, urldate = 2017-03-04.

[60] Redmon, J., Divvala, S., Girshick, R., Farhadi, A., 2016. You only look once: Unified, real-time object detection, in: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 779–788.

[61] Ren, S., He, K., Girshick, R., Sun, J., 2015. Faster r-cnn: Towards real-time object detection with region proposal networks, in: Advances in neural information processing systems, pp. 91–99.

[62] Sagnier, C., . TYPES OF MAINTENANCE : 5 DEFINITIONS YOU SHOULD KNOW, year = 2018, url = https://www.mobility-work.com/blog/5-types-maintenance-you-should-know, urldate = 2018-11-15.

[63] Sambasivarao, 2019. Region proposal network-a detailed view. URL: https://towardsdatascience.com/region-proposal-network-a-detailed-view-1305c7875853.

[64] Sedighi, B., 2018. Deep learning for fault detection in fences. Norwegian university of science and technology.

[65] Serena Yeung, F.J.., 2018. Lecture 11:detection and segmentation. Norwegian university of science and technology.

[66] SHARMA, S., . Activation Functions in Neural Networks, year = 2017, url = https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6/, urldate = 2017-09-06.

[67] Silberman, N., Sontag, D., Fergus, R., 2014. Instance segmentation of indoor scenes using a coverage loss, in: European Conference on Computer Vision, Springer. pp. 616–631.

[68] Sinha, U., . Convolutions, year = 2017, url = https://aishack.in/tutorials/image-convolution-examples/,.

[69] Skalski, P., . Gentle Dive into Math Behind Convolutional Neural Networks, year = 2019, url = https://towardsdatascience.com/gentle-dive-into-math-behind-convolutional-neural-networks-79a07dd44cf9, urldate = 2019-04-12.

[70] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R., 2014. Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research 15, 1929–1958.

[71] Szepesvári, C., 2010. Algorithms for reinforcement learning. Synthesis lectures on artificial intelligence and machine learning 4, 1–103.

[72] Tesla, . Tesla's mission is to accelerate the world's transition to sustainable energy. URL: https://www.tesla.com/about.

[73] Thomas, S., 2019. PyTorch Deep Learning Hands-On: Build CNNs, RNNs, GANs, reinforcement ... Packt Publishing.

[74] Van Rossum, G., Drake, F.L., 2009. Python 3 Reference Manual. CreateSpace, Scotts Valley, CA.

[75] Vedaldi, A., Gulshan, V., Varma, M., Zisserman, A., 2009. Multiple kernels for object detection, in: 2009 IEEE 12th International Conference on Computer Vision, pp. 606–613.

[76] vegtilsynet, . System for oppfølging av avvik på rekkverk statens vegvesen. URL: https://vegtilsynet.com/tilsyn/tilsynsrapporter/system-for-oppfolging-av-avvik-pa-rekkverk.

[77] vegvesen, S., 2014. Rekkverk og vegens sideområder, håndbok n 101. URL: https://www.vegvesen.no/_attachment/69909.

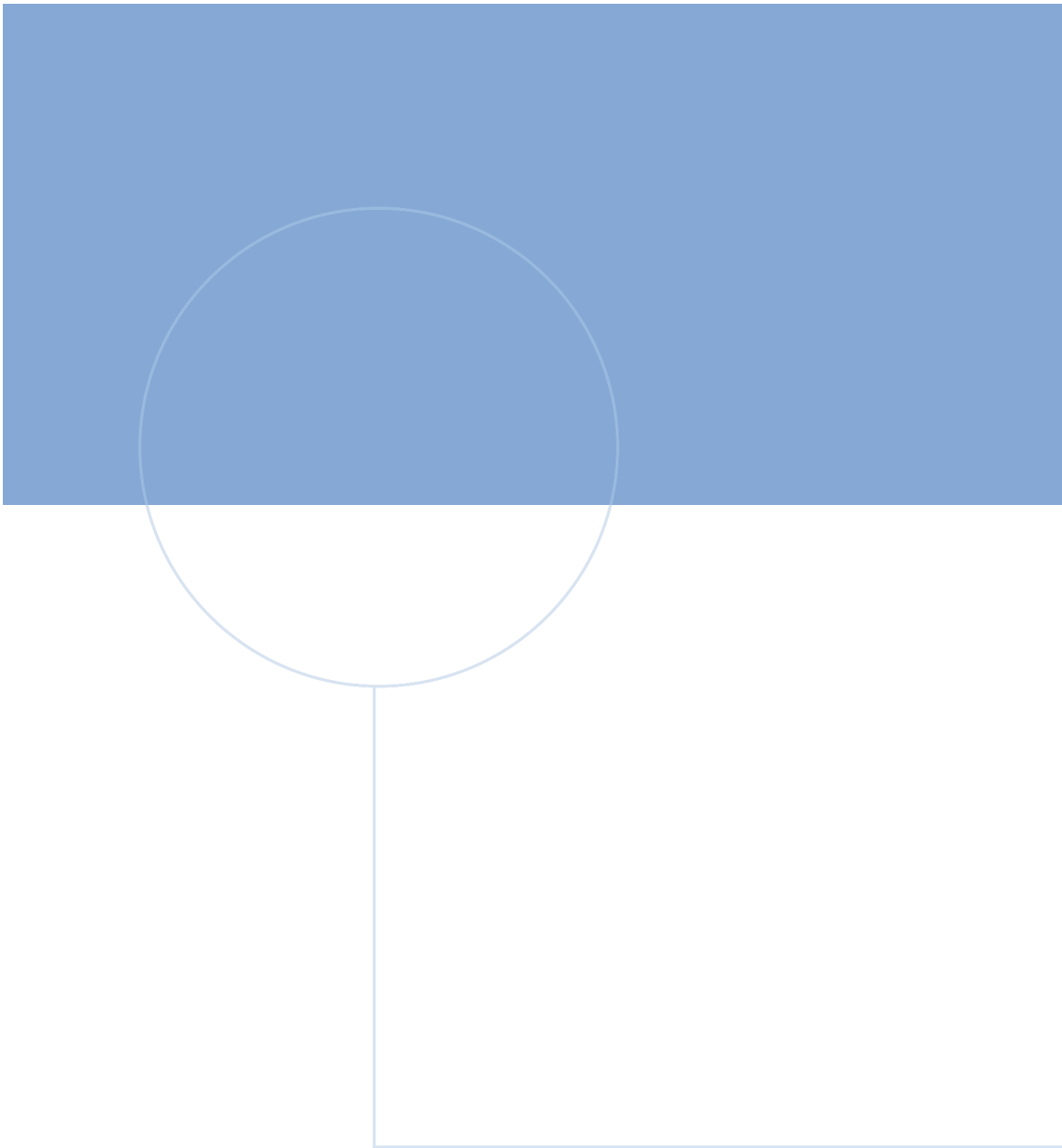
[78] Walt, S.v.d., Colbert, S.C., Varoquaux, G., 2011. The numpy array: a structure for efficient numerical computation. Computing in Science & Engineering 13, 22–30.

[79] Whitehead, N., Fit-Florea, A., 2017. Floating point and ieee-754 compliance for nvidia gpus. Nvidia Whitepaper .

[80] Wu, X., Sahoo, D., Hoi, S.C.H., 2019. Recent advances in deep learning for object detection. arXiv:1908.03673.

[81] Zhang, J., Huang, K., Yu, Y., Tan, T., 2011. Boosted local structured hog-lbp for object localization, in: CVPR 2011, pp. 1393–1400.

[82] Zhao, Z.Q., Zheng, P., tao Xu, S., Wu, X., 2019. Object detection with deep learning: A review. arXiv:1807.05511.



NTNU

Norwegian University of
Science and Technology