Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Helene Semb

Optimizing a neural network with genetic algorithm and background removal for instance segmentation of infrastructural damages

Master's thesis in Cybernetics and Robotics
Supervisor: Annette Stahl
Co-supervisor: Marius Andersen, Ole Øystein Knudsen, Robin Vacher
June 2023

NTNU
Norwegian University of Science and Technology

Helene Semb

Optimizing a neural network with genetic algorithm and background removal for instance segmentation of infrastructural damages

Master’s thesis in Cybernetics and Robotics
Supervisor: Annette Stahl
Co-supervisor: Marius Andersen, Ole Øystein Knudsen, Robin Vacher
June 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Abstract

This thesis aims to study and assess strategies of improving a an instance segmentation model for detecting and segmenting corrosion damages from bridge inspections. To evaluate the impact of the images, the dataset was narrowed down to include only images with red corrosion, i.e steel corrosion. Subsequently, three strategies were employed to assess the potential improvements in accuracy: data augmentation, hyperparameter optimization using the Genetic Algorithm, and a background removal network specifically designed to eliminate sky regions from the images. These strategies were implemented and tested on five backbone models using Mask R-CNN.

The results indicate that each method has its own strengths and limitations for detecting different types of red corrosion, but none of them are able replace the deficient quality of the dataset, as the only remarkable improvement comes from the reduction in dataset. While both automatic hyperparameter tuning and change of backbone indicate some improvement, there are still challenges of generalizing such a broad dataset, that the images should either be expanded threefold to improve the network's capabilities, or sharpen its focus on one specific damage. The Genetic Algorithm does have potential for further development of the network, but suffers from a small search space and limited population size. The sky removal network did no visible improvement to the accuracy, making the pre-processing step redundant. Further research should be done on a better augmentation scheme than the one presented in this assignment. The sky removal network did no visible improvement to the accuracy, making the pre-processing step redundant. Other further improvements suggested are to use RGB-D images, synthetic data or to consider other networks than Mask R-CNN.

Sammendrag

Denne oppgaven tar sikte på å studere og vurdere strategier for å forbedre et nevralt for å oppdage og segmentere korrosjonsskader fra broinspeksjoner. For å evaluere effekten av bildene, ble datasettet innsnevret til kun å inkludere bilder med rød korrosjon. Deretter ble tre strategier brukt for å vurdere potensielle forbedringer i nøyaktighet: dataforstørrelse, hyperparameteroptimalisering ved bruk av den genetiske algoritmen, og et bakgrunnsfjerningsnettverk spesielt designet for å eliminere himmelregioner fra bildene. Disse strategiene ble implementert og testet på fem ryggradsmodeller ved bruk av Mask R-CNN.

Resultatene indikerer at hver metode har sine egne styrker og begrensninger for å oppdage ulike typer rød korrosjon, men ingen er i stand til å erstatte den mangelfulle kvaliteten på datasettet, da den eneste bemerkelsesverdige forbedringen kommer fra reduksjonen i datasettet. Mens både automatisk hyperparameterinnstilling og endring av ryggrad indikerer en viss forbedring, er det fortsatt utfordringer med å generalisere seg til et så bredt datasett. For å løse disse utfordringene bør datasettet utvides tre ganger for å forbedre nettverkets muligheter, eller begrense til en skade for skjerpe fokuset til nettverket. Den genetiske algoritmen har potensial for videreutvikling av nettverket, men lider av liten søkeplass og begrenset populasjonsstørrelse. Det bør forskes videre på bedre dataforstørrelse teknikker enn det som er presentert i denne oppgaven. Skyfjerningsnettverket gjorde ingen synlig forbedring av nøyaktigheten, noe som gjorde forbehandlingstrinnet overflødig. Andre ytterligere forbedringer som foreslås er å bruke RGB-D-bilder, syntetiske data eller å vurdere andre nettverk som har prestert bedre enn Mask R-CNN i andre eksperimenter.

Preface

This master's thesis marks the conclusion of my five-year degree at the Department of Engineering Cybernetics, and is the result of a collaboration between NTNU and SINTEF. The task was assigned January 16th 2023 and was delivered June 12th 2023, and is a continuation of my pre-project assignment written in the fall of 2022. The thesis is written entirely by me. All contributions, text and figures are original works by me. The dataset is provided by SINTEF, with the Norwegian Public Roads Administration providing images from real bridge inspections, along with relevant corrosion images taken from stock image collections, and code implementation for Mask R-CNN and YOLO is provided by Meta AI and Ultralytics respectively. Relevant code contributions can be found in the appendices and on my GitHub for those interested. The pre-project contains a lot of important background information and theory that this thesis is based on, and it is recommended to read through it before starting on this thesis. Citations are provided in the specific sections, and the project thesis is delivered along with this thesis.

The work has been trisected. The first part was spent on setting up the network with a new framework, Detectron2 and training backbone models on the existing and evaluating current solution paths, basically picking up where I left off in the pre-project, which fell short in some training evaluation. The second part of the project consisted of implementing the Genetic Algorithm and the sky segmentation network with YOLO and training them. The last month was spent on running the final networks and analyzing and summarizing the results.

This has been at times a frustrating thesis, and exploring the world of deep learning networks has been a challenging, but fun task nonetheless. Because this project operates in such a narrow field, a lot of freedom was given to me to explore the methods I deemed interesting, however little impact they had on the network. All network training was done on the Idun cluster provided by the HPC group at NTNU, and I am grateful to them for providing me with lifesaving software. Without Idun, this assignment would not be possible.

I would like to thank my four supervisors Annette Stahl (NTNU), Marius Andersen (SINTEF), Ole Øystein Knudsen (SINTEF) and Robin Vacher (SINTEF). Andersen and Vacher have provided me with guidance and ideas when I have felt stuck, while Knudsen and Stahl have provided me with valuable discussion points and guidance on how to structure a good master's thesis.

Deep learning is a field in rapid development, and I am afraid this thesis may already be irrelevant by the time it is published. But hopefully, it may provide some insights into working with neural networks and what we can learn from the challenges.

Helene Semb
NTNU, Trondheim
June 13th, 2023

Contents

Figures

Tables

Chapter 1

Introduction

1.1 Motivation and Background

Bridges are critical components of any transportation infrastructure, and ensuring their safety and longevity is of paramount importance. The Norwegian Public Roads Administration (NPRA) is responsible for maintaining and inspecting over 6000 bridge constructions in Norway, spending over 500 million NOK each year[1]. However, one of the significant concerns in relation to steel bridges is corrosion, which can severely impact the bridge's carrying capacity and deteriorate its lifetime. The most common examples of this are bridges close to seawater and in harsh weather conditions, which includes many on the coast of Norway, like the Nordhordaland bridge, seen in Figure 1.1.

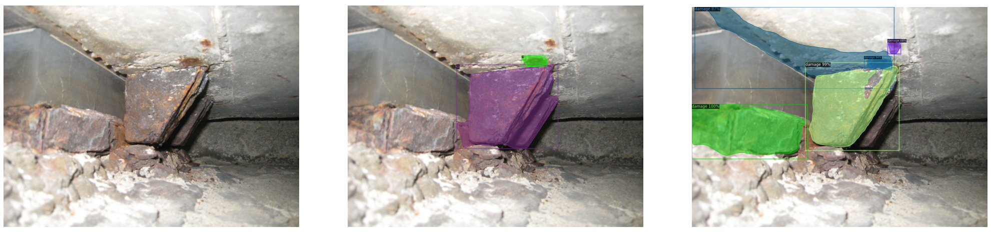
When NPRA deems a bridge to be critically damaged, it needs to be closed for maintenance, leading to huge repair costs and high CO2 emissions. Therefore, NPRA needs to achieve the designated lifetime of a bridge and discover corrosion and other damages before they become critical. To prevent corrosion, the NPRA applies a zinc coating on the steel structure as sacrificial protection, in addition to a protective layer of paint. However, regular recoating is an expensive task that requires a thorough assessment of when it is needed.

Currently, the main inspection is done every 5 years, and it involves manual on-site inspections by qualified engineers and inspectors[1]. This process is time-consuming, requiring extensive gear and personnel, and can be prone to human errors, leading to conflicting views on the degree of damage.

Recently, advancements in computer vision have made it a realistic goal to implement techniques such as object detection and image segmentation into infrastructural tasks, mostly related to inspection[2, 3]. An automated algorithm to detect corrosion on images would save hours of tedious inspection and remove the issue of human subjectivity. With the increasing use of drones, there is a potential for an end-to-end automated inspection, a cost-efficient method that reduces time and risk, as the drone can access spots on the bridge that would usually require scaffolding, climbing gear and an underbridge inspection truck. Moreover, more frequent inspections can lead to early detection of damages, reducing the need for huge repairs, as studies from the World Corrosion Organization suggests that between 25% and 30% of costs related to repair can be reduced in optimal damage reduction [4].

Although many recent advances have been made in models and algorithms that can segment corrosion and similar damages, the focus of this thesis is on using Mask R-CNN developed by He et al. at Meta AI [5], which Fondevik further develops in order to segment corrosion damages in the spring of 2020[6]. During the pre-project conducted in the fall of 2022, an assessment of the current network revealed suboptimal performance, characterized by low accuracy and a high rate of false predictions[7]. A notable instance of these misclassifications is illustrated in fig. 1.2. The pre-project identified several challenges faced by the network, including significant data variation, untuned hyperparameters, and the presence of noisy backgrounds. Potential solutions were discussed to address these issues.

Figure 1.1: Nordhordland bridge



(a) Corroded bridge part **(b)** Ground truth **(c)** Current model prediction

Figure 1.2: The goal of the model versus the detections the model currently produces

1.2 Aim and scope of the thesis

The project thesis conducted in the previous fall offers a comprehensive examination of potential enhancements to the network, drawing upon a thorough literature review. From the array of solutions presented, a careful selection was made, considering the project's time constraints and the potential for promising outcomes. A notable challenge identified in the pre-project phase revolved around the substantial variation among images within the dataset, despite its relatively small size. These variations encompass differences in corrosion proximity, damage types, structural characteristics, and viewing angles, collectively

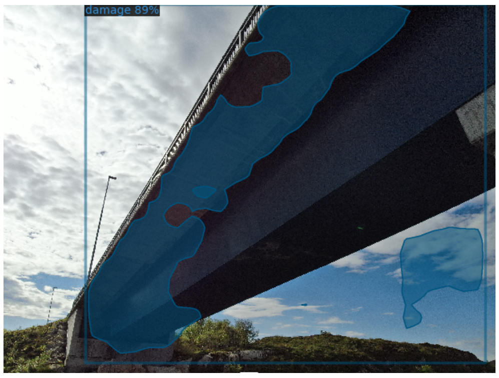


Figure 1.3: False prediction of sky from the model

posing a challenge to the model's generalization capability. Although expanding the dataset remains the optimal approach for any neural network task, this thesis proposes data augmentation as a pragmatic short-term solution. The second point the pre-project emphasizes is the importance of hyperparameters, but as manual tuning is a time-consuming labour, the potential of using the Genetic Algorithm to automatically tune the model's hyperparameters could save both time and provide a better result. Lastly, from reviewing the test images it seems that the model struggles with incorrectly detecting parts of the background as corrosion. These false predictions are specially related to clouds when they are present in the images, or water reflections of the sky. An example can be viewed in fig. 1.3, where the model mistakes a cloud for corrosion. The idea is to implement an independent network to segment the background, specifically the sky, to remove the pixels as a pre-processing step for the images. The result would ideally look like fig. 1.4. Based on the limited time for the experiment and reviewing the different methods, this thesis will implement and assess these methods:

- The Genetic Algorithm for hyperparameter tuning
- A second independent network for detecting and removing sky regions from the images.

The primary objective of this thesis is to evaluate the potential performance enhancements achieved through these methods, specifically in terms of increased IoU, which serves as the primary evaluation metric for the entire project. Building upon the results obtained from the corrosion detection model, this thesis aims to determine whether these solutions effectively enhance accuracy and warrant further consideration for future applications, or if alternative approaches should be explored. The methods under investigation are grounded in theoretical foundations established in similar tasks, as outlined in Chapter 5 of the pre-project. However, there is limited empirical evidence available regarding the extent to which these methods can improve the network's performance in real-world scenarios. Consequently, a comparative analysis will be conducted among these strategies, traditional data augmentation techniques, and various backbone configurations of the Mask R-CNN model.

1.3 Previous work

This section presents an overview of previous work on which these solutions are based. A review of previous work on corrosion damages has been covered extensively in section

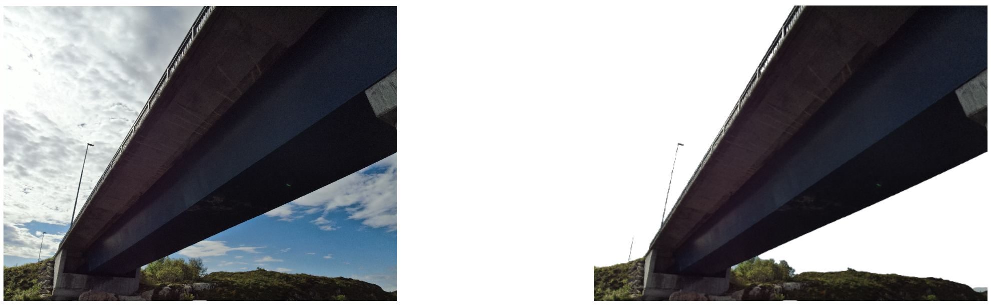

(a) Image before the preprocessing **(b)** Image after preprocessing

Figure 1.4: The goal of a sky network would be to remove unimportant pixels from images to have less disturbances for the Mask R-CNN.

3.3 by the pre-project[7].

1.3.1 Damage segmentation

After the literature review conducted in December 2022[7], there has been limited recent work on corrosion detection using Mask R-CNN. The only notable project in this domain is the study conducted by Lemos et al., which focuses on detecting corrosion on industrial building rooftops[8]. They achieve a recall of 85.8% and precision of 84.0% by leveraging a monotonous and large dataset comprising of images captured from a fixed distance. This approach simplifies the task compared to this project's current dataset, which requires detecting corrosion from various angles and structures. Multi-class segmentation for structure inspection seems to be the primary objective in current research, specifically crack and delamination detection. Bai et al.[9] in their project provide the most recent insight on automatic inspection where they use several deep learning methods and models for Structural Damage Detection(SSD).

1.3.2 Automatic hyperparameter tuning

Automatic hyperparameter optimization is a common procedure when tuning neural networks. The most significant complex algorithm used especially for Convolutional Neural Networks is the Genetic Algorithm. The work on the Genetic Algorithm for this project is mostly based on the paper by Gerbet et al. for segmentation of rust maize on plants, as the project also used Mask R-CNN. The paper is discussed in section 5.1 in the pre-project[7]. Another notable project which has used the Genetic Algorithm for tuning is by Rodrigues et al., who compared the Genetic Algorithm and other optimization methods on three fundamental parameters; learning rate, dropout and momentum, for the purpose of classifying acute lymphs[10]. In the paper, they concluded that the Genetic Algorithm achieved the best results with a 98.46% accuracy compared to Bayesian optimization (80.39%) and Random search(62.31%).

Lee et al. presented another way of implementing the Genetic Algorithm, by using it to design and optimize the architecture for their own Convolutional Neural Network, testing it on a brain image dataset used for Alzheimer's disease diagnosis[11]. Their model created its own architecture by first encoding the connections between convolutional layers as potential candidates. Then, it applied the Genetic Algorithm to select candidates with the best performance.

1.4 Contributions

1.4.1 Dataset construction for sky segmentation

To save time annotating a whole new dataset for sky segmentation, the network primarily used existing datasets which contain sky regions as a labelled class. The largest dataset is ADE20K, providing around 9,000 images with the class sky, along with 149 other classes. A simple code was constructed to extract one class so that "sky" was the only class being segmented. The code can easily be transferred to extract other classes in the dataset. A small custom dataset is also created from existing images of bridges taken by Orbiton[12] of typical bridges in Norway, in order to improve the model's capabilities to detect sky for these types of images.

1.4.2 Code and implementation

Mask R-CNN

This thesis changed the framework from the original Mask R-CNN implementation from the pre-project[7], replacing it with Meta AIs own library Detectron2[13]. The model was implemented with customized configurations and augmentations, and with modules for training, evaluating and predicting the corrosion dataset.

Genetic Algorithm

The Genetic Algorithm implemented in this project was a modified version of an implementation regarding the knapsack problem [14]. The changes allowed a structure for more complex genes other than a binary value, making the chromosome type dictionary instead of a list. In addition, I have implemented two different mutation functions, one that allows for random value assignments in a hyperparameter, and one which chooses a random value in a range from the parameter's original value, where the range shrinks according to its generation.

Sky segmentation and removal

This assignment involved utilizing the YOLO model (You Only Look Once) to detect and segment sky regions in the corrosion dataset. These identified sky regions were then transformed into binary masks which were used to black out the sky pixels in each image. This network was called SkySeg for the sake of simplicity.

1.4.3 Format conversion

The annotations for both datasets are in the form of binary mask images, while the models require a specific text format (COCO for Detectron2, YOLO for YOLO). This thesis implemented the two types of dataset conversions for adapting the binary masks to the assorted texted format by locating the contours of the mask images and deriving the polygon coordinates.

Chapter 2

Preliminaries

The chapter will introduce background theory on deep learning optimization and new concepts implemented in this thesis. Theory on Convolutional Neural Networks and Mask R-CNN has been covered in chapter 2 of the pre-project[7].

2.1 Neural Network Optimization

Handling large Convolutional Neural Networks (CNNs) such as Mask R-CNN with over 60 million parameters[5] can be difficult, as there are several factors, including dataset, weights, activation functions, that contribute to the performance. The only way to evaluate where the model underperforms is through metrics such as training or validation loss. Even these metrics aren't able to provide a clear direction on how to improve the network. The literature behind neural networks does however present a variety of ways to improve and test accuracy, some of which the pre-project also presents in chapter 3 and chapter 5[7]. This section provides a short summary of the background on neural network improvements, which also provides the context of implementations further in this thesis.

2.1.1 Evaluation metric

Evaluation metrics are an important factor in assessing the performance of deep learning models. These metrics provide quantitative measures to understand the model's capabilities, identify areas for improvement, and compare different approaches.

Selecting the appropriate evaluation metric is a critical decision during the pre-work phase as it ensures that the chosen metric aligns with the specific task requirements and accurately measures the model's performance.

In instance segmentation and classification, it is common to use the terms true positives(TP), i.e correctly predicted pixels as corrosion, true negatives (TN) which are pixels correctly predicted as background, and false positives(FP) and false negatives(FN) which are incorrect predictions in each class. From these terms, several important metrics are defined. The comparison of the three metrics discussed in this thesis can be seen in fig. 2.1

Accuracy

Accuracy is a measure of the overall correctness of a classifier. It represents the proportion of correctly predicted instances (both positive and negative) out of the total instances.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Accuracy is a commonly used evaluation metric, but it can be misleading when the classes in the dataset are imbalanced. If the dataset has a large number of instances from one class and very few from the other, a classifier that always predicts the majority class can achieve high accuracy without actually performing well. For example in this corrosion dataset, where in general the image consist of 80% background[7], the accuracy can reach over 90% accuracy without actually predicting corrosion correctly.

Precision

Precision, shown in fig. 2.1a, measures the proportion of correctly predicted positive instances (TP) out of the total instances predicted as positive (TP + FP). It focuses on the accuracy of positive predictions. High precision indicates that the classifier has a low rate of falsely predicting positive instances.

$$\text{Precision} = \frac{TP}{TP + FP}$$

Precision is useful when the cost of false positives is high, such as in medical treatment. But to achieve 100% precision comes at the cost of risking a negative output, i.e. predict no pixel as corrosion.

Recall

Recall(fig. 2.1b), also known as sensitivity, measures the proportion of correctly predicted positive instances (TP) out of the total actual positive instances (TP + FN). It focuses on the classifier's ability to find all positive instances. It is in many ways the inverse of precision, and sense to reduce the number of false negatives.

$$\text{Recall} = \frac{TP}{TP + FN}$$

As with precision, the risk of getting 100% recall is to set all pixels as positive, which serves little practical use.

The weight of precision versus recall depends on the computer vision task, as methods done on the network can have positive implications on one metric, but negative on the other. It's therefore more valuable to find other metrics to evaluate the optimal value for both precision and recall.

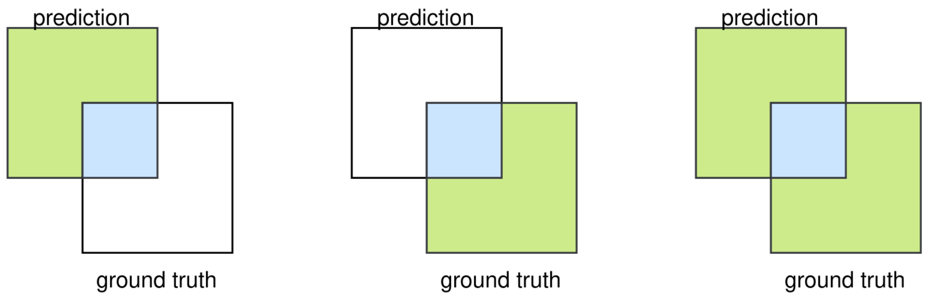
Intersection over Unit

Intersection over Union (IoU), shown in fig. 2.1c, is a commonly used evaluation metric in computer vision tasks, particularly in object detection and segmentation. It measures the overlap between the predicted bounding box or mask and the ground truth annotation, providing a quantitative assessment of the accuracy of the predictions. IoU is calculated by dividing the area of intersection between the predicted and ground truth regions by the area of their union. The resulting value ranges from 0 to 1, where a value of 1 indicates a perfect overlap between the predicted and ground truth regions, while a value of 0 indicates no overlap at all.

$$\text{IoU} = \frac{TP}{TP + FN + FP}$$

IoU is an effective metric for assessing the localization accuracy of objects or regions of interest in an image. It evaluates how well the model captures the spatial extent of the objects being detected or segmented. A higher IoU indicates a better alignment between the predicted and ground truth regions, reflecting a higher level of precision in the model's predictions.

IoU is often reported as a mean IoU (mIoU), which is the average IoU across multiple objects or classes. It provides an overall measure of the model's performance by considering the IoU values for all the predicted objects or classes.



(a) Precision **(b)** Recall **(c)** Intersection over Unit

Figure 2.1: Visualization of the three primary evaluation metric equations. Blue denotes the area of the nominator (true positive), while green plus blue denotes the area of the denominator

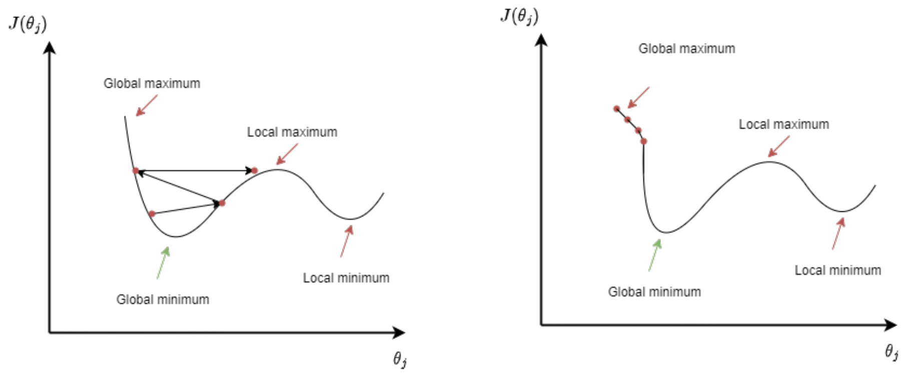
2.1.2 Hyperparameter optimization

Hyperparameters, settings configured external to the model, contribute largely to how the model performs on different data patterns. For a machine learning model to optimally solve the given task, hyperparameter optimization, also known as tuning, is an essential part of working with neural networks[15].

An important hyperparameter is the learning rate. During weight optimization with Stochastic Gradient Descent(SGD), which is the most popular optimization technique and the one used in Mask R-CNN[5], the learning rate is the step size taken in the gradient's direction. The simplest equation of updating the parameter θ_j with SGD is shown in eq. (2.1). $J(\theta)$ is the loss function, and α is the learning rate. Deciding the step size can have a large impact on the quality of optimization. An example in fig. 2.2 shows the graph of running SGD with two different learning rates. Choosing a large learning rate, especially close to the global minima, can cause the optimization to "jump" over the optimal parameter value as seen in fig. 2.2a. Choosing a small learning rate will however cause the model to slow down, risking the potential of not reaching an optimal value before training is over, as seen in fig. 2.2b. A small step size also increases the risk of the parameter reaching a local optimal value.

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_j) \tag{2.1}$$

Another notable hyperparameter is the minimum confidence threshold. As all predictions come with a class probability, it is up to us to decide the threshold for the model to



(a) α too large **(b)** α too small

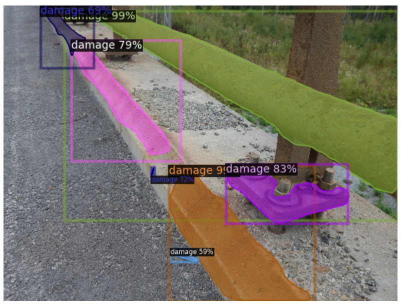
Figure 2.2: Example comparison of two learning rates on four iterations of SGD.

classify the detection as corrosion. Setting the threshold higher lowers the probability for faulty background detections for example, but if the model already fails to detect the target, a high threshold can lead to degradation of performance, as fig. 2.3b shows. Lowering the threshold allows for more detections, but increases the risk of wrong predictions, as in fig. 2.3a. The point should therefore be to find the optimal threshold between false predictions. This could be done with normal experimentation, as in fig. 2.3, in order to assign an optimal value which does not degrade the IoU. The training and dataset quality impact this hyperparameter and sets a roof for the highest IoU achieved by optimizing the threshold, as class probability can fail due to poor training.

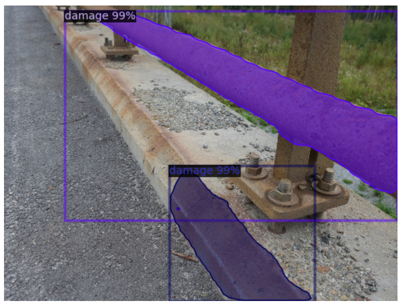
All though hyperparameter tuning is important, the task can be time-consuming and tedious, as it involves manually setting and evaluating each hyperparameter against each other, requiring countless training hours that are often wasted on tuning redundant values. It is therefore often better to use automatic optimization methods. The most used traditional methods for automatic tuning are grid search and random search, both visualized in fig. 2.4. Grid search is a systematic brute-force application that trains a model with a combination of all hyperparameter values within a given interval in order to cover the entire search space, seen in fig. 2.4a. Random search, depicted in Figure 2.4b, employs a straightforward approach by randomly sampling values from the search space in order to discover the optimal solution. While this technique offers a time-efficient approach, the limited search space can restrict the exploration to only local minima.

While simple algorithms can be effective for small models with limited hyperparameters, they may not scale well to more complex optimization problems with larger search spaces. As the need for efficiency increases, the use of metaheuristic algorithms becomes more prominent.

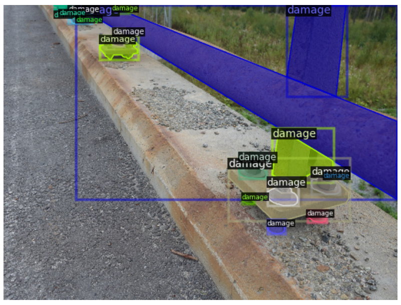
Metaheuristic algorithms are techniques that guide the search for optimal solutions in complex and large-scale optimization problems[16]. They offer an effective approach for hyperparameter optimization, outperforming the traditional techniques[15]. By employing heuristic rules and randomization, metaheuristics can efficiently navigate through the search space, searching for promising regions and avoiding getting trapped in local optima. Examples of popular metaheuristic algorithms for hyperparameter optimization include the Genetic Algorithm, which is discussed further in section 2.3, Particle Swarm Optimization and Grey Wolf Optimization.



(a) Threshold 0.5



(b) Threshold 0.9



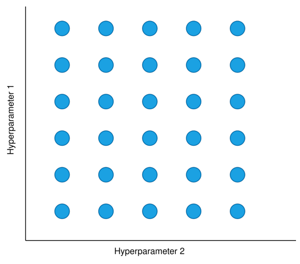
(c) Ground truth

Figure 2.3: Comparing predictions with two thresholds.

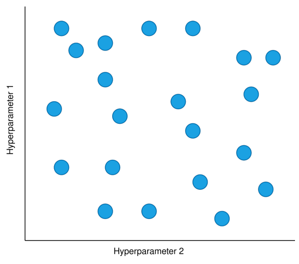
2.1.3 Choice of backbone

Section 2.2 of the pre-project[7] explained the architecture of Convolutional Neural Networks and introduces the concept of a backbone network in Mask R-CNN. As Mask R-CNN is a two-stage network[17], it consists of backbone CNN whose goal is to perform basic feature extraction. The backbone chosen can be a variety of architectures, but the most used are residual networks[5], or ResNets[18], which use a simple module architecture of stacking layers of the same size on top of each other and adds a residual block at the end of a convolution to avoid disappearing gradients in backpropagation. Mask R-CNN generally uses ResNet-50 or ResNet-101, where 50 and 101 denote the number of layers in the network.

The ResNeXt architecture, developed by Xi et al. in 2017[19], serves as an extension to the ResNet framework and has gained popularity as a backbone model for Mask



(a) Grid search



(b) Random search

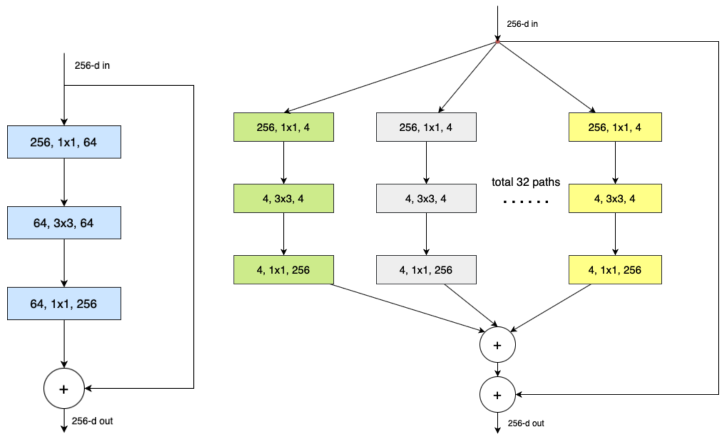
Figure 2.4: Visualization of how automatic tuning operates

R-CNN in various applications[5]. Its primary objective is to enhance classification accuracy without increasing the complexity of the network. ResNeXt achieves this by introducing the concept of "cardinality," which serves as an additional dimension to the width and depth of the network. By combining the repeating-layer strategy of ResNet with the split-transform-merge strategy known from Inception networks[20], ResNeXt achieves improved efficiency and performance. The block comparison between ResNet and ResNeXt can be viewed in fig. 2.5

This approach addresses some limitations of ResNet. The structure of stacking simple layers on top of each other can face challenges in adapting to more complex tasks due to the increasing number of hyperparameters associated with deeper networks. In contrast, Inception networks have carefully designed topologies to achieve good accuracy with low theoretical complexity. In one module, the input is split into lower-dimensional embeddings with 1x1 convolutions, transformed by a set of filters such as 3x3 or 5x5 convolutions, and merged by concatenation. that balances accuracy and computational complexity[20].

While ResNet has shown strong performance in image classification tasks, the ResNeXt architecture has emerged as the best-performing backbone model in combination with Mask R-CNN on the COCO dataset[5, 21]. Its utilization in the corrosion detection model is a promising avenue for future exploration.

The choice of backbone architecture plays a crucial role in the performance of Mask R-CNN, with larger and more complex networks generally being suitable for larger datasets, while simpler networks may suffice for smaller datasets.



(a) Block of ResNet **(b)** Block of ResNeXt with cardinality 32

Figure 2.5: Side-by-side comparison of the two backbone networks, a layer is shown with (# in-channels, filter size, # out-channels).

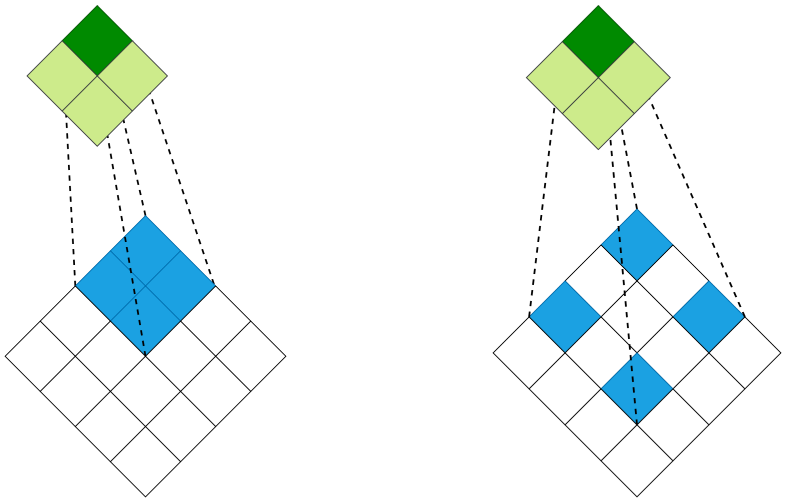
Feature extraction

How the backbones perform feature extraction is also an additional architecture choice. The simplest option is a normal convolutional extraction as seen in fig. 2.6a, at the final layer of the fourth block in ResNet, denoted as C4. Faster R-CNN network utilizes this type[17], but there are no remarkable results from the operator being used with Mask R-CNN.

A method that the pre-project discussed in section 2.2[7] is the Feature Pyramid Network(FPN) developed by He et. al[22]. Mask R-CNN uses this feature extraction as default in the original paper[5] and in an experiment testing the robustness of these feature extractors, FPN showed the most promising results[23].

Another popular convolutional operation is using dilated convolutions at the fifth block in ResNet, denoted as DC5[24]. Dilated convolutions stand apart from standard convolutions by introducing gaps between the filter elements, allowing for an expanded receptive field without increasing the number of parameters or the computational cost significantly, shown in fig. 2.6b. The "dilation factor" determines the spacing between the elements of the filter. By using dilated convolutions, models can capture both local and global information in an efficient manner. It has been particularly useful in tasks that require a broader context.

Testing these feature extractors with different backbones against each other can provide useful information on what methods work best with the given task.



(a) Regular convolution **(b)** Dilated convolution

Figure 2.6: Comparing a regular 2x2 convolution wtih a dilated convolution with dilation factor 1

2.1.4 Image processing

In many cases when a deep learning model struggles with accuracy, the cause usually falls on the training data instead of the model itself. The quality of images is an important factor that hugely impacts the model performance, either because the dataset is too monotonous or too small. Increasing the dataset is always a viable option, but data acquisition is a process that takes time, as the pre-project also mentioned in section 3.4.3[7]. Therefore, the alternatives are to work with the available images instead.

Data augmentation

Data augmentation is a widely used technique in computer vision that involves applying various transformations to the training data to artificially increase its size and diversity[25]. By augmenting the data, deep learning models can improve robustness, generalization, and performance. The most popular augmentation techniques are listed below

- **Color Augmentation**:
  - **Brightness adjustment**: Altering the brightness level of the image by scaling the pixel values up or down.
  - **Contrast adjustment**: Modifying the contrast of the image by rescaling the pixel values to increase or decrease the difference between the bright and dark regions.
  - **Saturation adjustment**: Changing the saturation level of the image by scaling the color intensities.
  - **Hue adjustment**: Shifting the hue values of the image, which alters the color appearance.
- **Geometric Transformations**:
  - **Rotation**: Rotating the image by a certain angle, either in a clockwise or counterclockwise direction.
  - **Scaling**: Rescaling the image by either enlarging or reducing its size while maintaining the aspect ratio.
  - **Translation**: Shifting the image horizontally and vertically by a certain number of pixels.
  - **Flipping**: Flipping the image horizontally or vertically to create a mirror image.
  - **Shearing**: Distorting the image by tilting it along one of the axes.

These transformations can be combined and applied in various ways to generate diverse training samples. For example, multiple transformations can be randomly applied to each image during training. The parameters for each transformation, such as rotation angle, scaling factor, or brightness range, can also be randomized to introduce further variation in the augmented data.

Background removal

Background removal is an underutilized technique in image pre-processing, but previous studies have demonstrated its successful application in improving the performance of simple image classification tasks[26, 27]. The hypothesis is that by removing the background, which reduces irrelevant information and clutter, from an image, the classifier can focus solely on the foreground objects or regions of interest, leading to enhanced classification accuracy.

The effectiveness of background removal depends however on the specific dataset, the nature of the background, and the characteristics of the objects being classified. In some cases, the background may provide contextual information that is crucial for accurate classification. In cases with complex tasks, it can therefore be more practical to instead introduce more data in the training with a variety of background elements to make the model more robust to background elements.

There exist several methods for background removal, depending on the complexity of the dataset, ranging from simple thresholding[28] to fully fleshed neural networks. BackgroundRemover by nadermx[29] is a popular tool for standard background removal and uses U^2-net[30]. Choosing the right method could be difficult given the wide variety, but it also opens up possibilities to choose according to the preference of system, library and framework.

2.2 The YOLO network

You Only Look Once (YOLO) is a state-of-the-art object detection algorithm, introduced in 2015 by Redmon et al. in their famous research paper "You Only Look Once: Unified, Real-Time Object Detection"[31]. While Mask R-CNN, like the other R-CNN architectures, is a two-stage model. meaning they use the Regional Proposal Network to select certain regions of interest and then perform classification on those regions with an additional network head. YOLO, as its name indicates, uses a single network to perform detection from the entire image in one evaluation. This makes the YOLO algorithm outperform many neural network models in terms of speed, with the original version reaching a speed of 45 frames per second(FPS), in comparison Faster R-CNN reaches about 5 FPS[32].

2.2.1 Detection

YOLO differs from Mask R-CNN and other detection models in that it frames object detection as a regression problem instead of a classification task. Instead of dividing the task into separate classification and localization steps, YOLO treats directly predicts the bounding boxes and associated class probabilities in a single pass.

The system divides the input image into a S x S grid, in which the grid cell containing the centre of an object is responsible for detecting that object, each grid cell predicts B bounding boxes with four predictions, the x- and y- coordinate relative to the bounds of the cell, the width and height of the box relative to the whole image and associated confidence scores. The confidence score is calculated as Pr(Object) * IoU, in which Pr(Object) is the probability that the cell contains an object.

Each cell generates C class predictions, where C represents the number of classes, indicating the probabilities of containing each class. The network makes class-specific confidence scores at test time by multiplying the class probability with the object confidence score, and the class corresponding to the highest confidence score gets assigned to the box. YOLO also applies Non-minimum Max Suppression(NMS) as in Mask R-CNN to remove and concatenate overlapping prediction boxes. The detection task is visualized in fig. 2.7.

2.2.2 Architecture

The architecture behind YOLO builds on 24 convolutional layers followed by 2 fully connected layers which varies based on the version and model of the algorithm, but bases the architecture on the same pipeline as normal CNNs. The authors specify the main attributes of the architecture like this:

- Resize input image to $448x448$, to save computational power
- For each block, apply 1x1 convolution to reduce the number of channels, followed by a 3x3 convolution to generate a cuboidal output.
- The activation function mainly used is leaky-ReLU, except for the last layer which uses a linear function.
- Add additional techniques to reduce overfitting, such as dropout and batch normalization

2.2.3 Loss function

To train the network, The YOLO loss function combines the following:

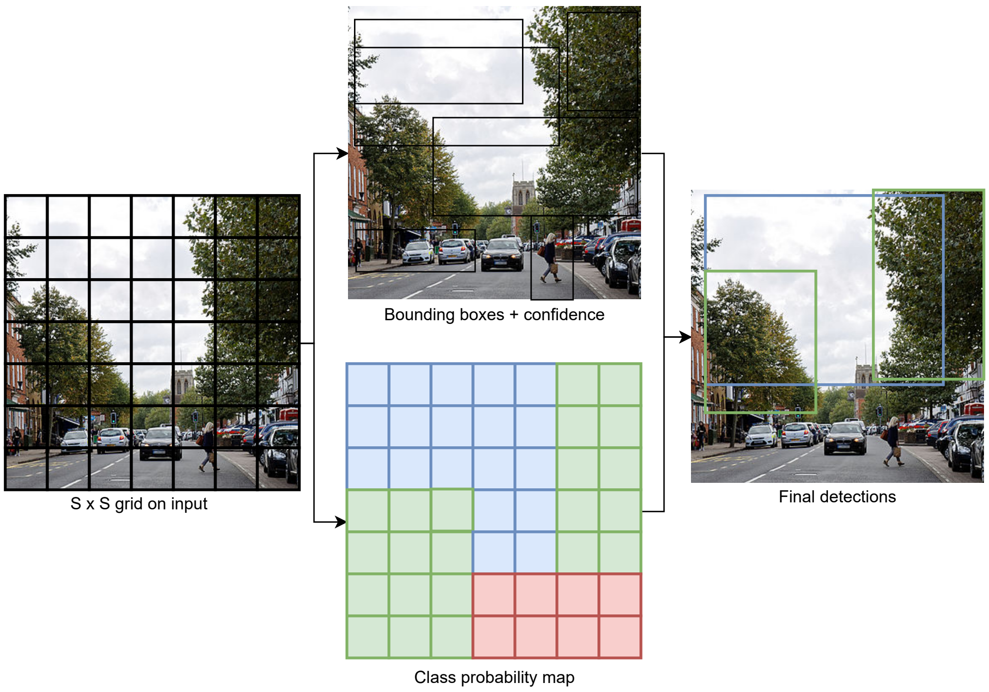


Figure 2.7: The detection pipeline of YOLO

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{loc}} + \mathcal{L}_{\text{conf}} + \mathcal{L}_{\text{cls}}$$

The total loss of the YOLO network is the sum of the localization loss($\mathcal{L}_{\text{loc}}$]), confidence loss($\mathcal{L}_{\text{conf}}$), and class loss($\mathcal{L}_{\text{cls}}$), each weighted by hyperparameters determined during training.

The localization loss:

$$\mathcal{L}_{\text{loc}} = \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 + (\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right]$$

The confidence loss:

$$\mathcal{L}_{\text{conf}} = \sum_{i=0}^{S^2} \sum_{j=0}^{B} \left[\mathbb{1}_{ij}^{obj} \left(C_i - \hat{C}_i \right)^2 + \lambda_{\text{noobj}} \mathbb{1}_{ij}^{noobj} \left(C_i - \hat{C}_i \right)^2 \right]$$

The class loss:

$$\mathcal{L}_{\text{cls}} = \lambda_{\text{cls}} \sum_{i=0}^{S^2} \mathbb{1}_{i}^{obj} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2$$

where:

- λ_{coord} is the localization loss weight, set to 5 in the paper
- λ_{noobj} is the weight for confidence loss on background (no object) predictions, set to 0.5 in the paper
- S is the grid size

- B is the number of anchor boxes per grid cell
- $\mathbb{1}_{ij}^{obj}$ is an indicator function that evaluates to 1 if the jth anchor box in the ith grid cell is responsible for detecting an object
- $\mathbb{1}_{ij}^{noobj}$ is an indicator function that evaluates to 1 if the jth anchor box in the ith grid cell does not contain an object
- x_i, y_i, w_i, h_i are the predicted bounding box coordinates of the ith grid cell
- $\hat{x}_i$, $\hat{y}_i$, $\hat{w}_i$, $\hat{h}_i$ are the ground truth bounding box coordinates of the ith grid cell
- C_i is the predicted objectness score of the ith grid cell
- $\hat{C}_i$ is the ground truth objectness score of the ith grid cell

The localization loss measures the error in predicting the bounding box coordinates (x, y, width, height) of the detected objects. It uses the mean squared error (MSE) to calculate loss, but uses the square root on width and height to remedy that small deviations in large boxes should not matter as much as with small boxes.

The confidence loss is responsible for evaluating the accuracy of the predicted confidence score associated with each bounding box. YOLO calculates the confidence loss using the binary cross-entropy loss between the predicted objectness scores and the ground truth.

The class loss measures the error in predicting the class probabilities of the detected objects. YOLO employs the categorical cross-entropy loss to compute the class loss between the predicted class probabilities and the ground truth class probabilities.

The ultimate objective of the network is to optimize average precision across all classes, which entails giving the highest weight to the localization loss among the three loss functions. By carefully balancing these loss functions together, YOLO achieves a remarkable balance between speed and accuracy, resulting in high-performance object detection.

2.3 Genetic Algorithm

The Genetic Algorithm (GA) is a well-known metaheuristic algorithm used to solve complex optimization problems in engineering, economics, and management[33]. First proposed by J.H. Holland in 1992 the algorithm takes inspiration from the Darwin evolutionary process, in which only the individuals best adapted to their environment survive. The advantage of this algorithm and other population-based algorithms is that they propose diversification methods to escape local minima[15].

There are different variants to the Genetic Algorithm, but the main idea remains the same. GA consists of a population of individuals, where each individual represents a potential solution to the optimization problem built up by a combination of parameter values, which are called genes[33]. The population is visualized in fig. 2.8. The individuals compete for the chance to survive and mate, but only the fittest individuals are allowed to create more offspring than others and are able to continue to the next generation. The genes from the fittest individuals propagate to their offspring, meaning one offspring will have a combination of genes from two well-performing parents. Thus each successive generation is better adapted to their environment, outperforming their parents. Once the offspring has no significant difference from their parents and they perform mostly the same, the algorithm has converged. To avoid premature convergence in local minima, mutation can be applied to certain offspring, with the change of random genes.

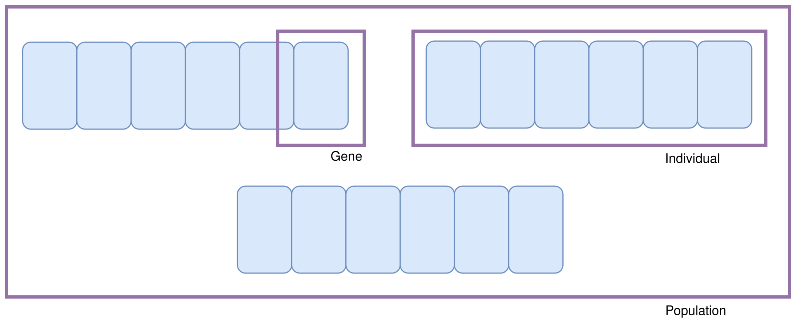
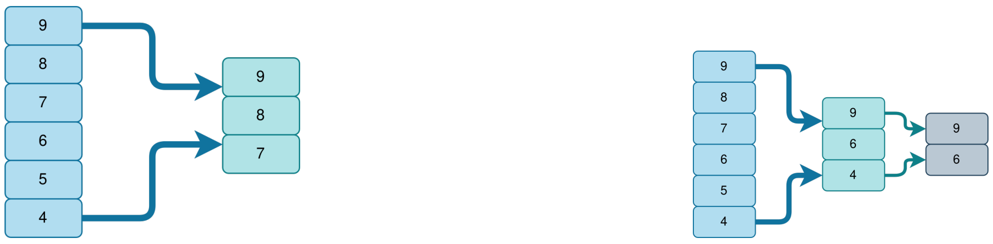


Figure 2.8: Representation of population, individual and gene in the Genetic Algorithm

2.3.1 Genetic operators

- **Selection:** The selection operator, shown in fig. 2.9 determines which individuals are chosen to reproduce and continue to the next generation. Two common methods are fitness-based selection, where the fittest half of the population mates randomly with each other, and tournament selection, where for each new individual a random subset of individuals competes for reproduction, with the two fittest chosen as parents.
- **Crossover:** Mating of the two parents once they are selected. As in normal mating, the offspring is a combination of genes from both parents, chosen at random. Crossover could be done either by choosing a random crossover site, where the offspring gets 50% of parent one's genes on the left of the crossover site, and 50% from parent two on the right, or by iterating through each gene and randomly picking one of the parent's gene, as visualized in fig. 2.10.
- **Mutation:** To maintain diversification in the population, the algorithm includes a mutation operator to allow for changes to some of the genes during the creation of new individuals. A mutation, visualized in fig. 2.11, will mean that for every gene in the individual, there is a small percentage chance that the value will be random, instead of inherited from one of the parents.



(a) Fitness-based **(b)** Tournament-based

Figure 2.9: Representation of the selection operator on a set of 6 individuals with a given fitness score. In (a) only the individuals with the best fitness can reproduce, while in (b) random individuals are first picked, and then sorted.

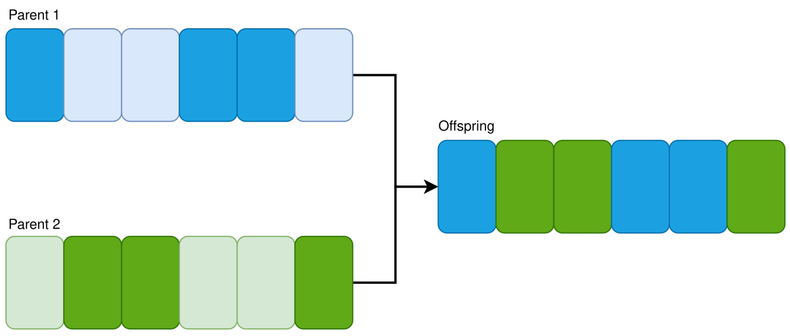


Figure 2.10: Representation of the crossover operator in GA

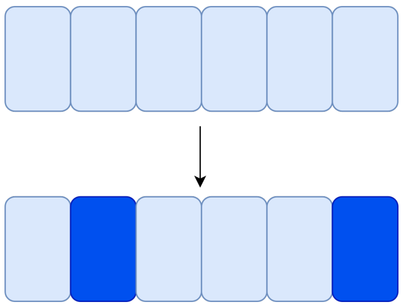


Figure 2.11: Representation of the mutation operator in GA, which chooses random genes and replaces them with a new value

2.3.2 Fitness function

The fitness function is a custom function whose purpose is to calculate the individual's ability to compete and adapt to the environment. The function assigns a fitness score to each individual, where the algorithm chooses the individuals with the optimal fitness score. Choosing a score and function depends on the problem GA is trying to optimize. The problem can either be to minimize a loss function or maximize accuracy. In hyper-parameter optimization (HPO) problems, the score could be an evaluation metric such as average precision or IoU. As long as the score is easily computed and measurable, it is possible to use it in the Genetic Algorithm.

Chapter 3

Neural network for corrosion segmentation

This chapter presents the corrosion network as it is and the current results. The goal is to provide an extended overview of the challenges the network faces and a base of comparison when trying out the new strategies

3.1 Dataset

The dataset is identical to the one presented in section 4.1 in the pre-project[7]. This dataset consisted of 1632 images for training, and 358 images for validation, annotated in Darwin by V7 labs[34]. Corrupted images, where the mask did not correspond to the correct image, were removed before training started, resulting in 1584 images for training and 323 images for validation. No new images have been added to the dataset, and by default there was no data augmentation besides horizontal flipping with a 50% rate

3.2 Model

The model currently in use is Mask R-CNN from Meta AI[5] as used throughout the pre-project. Mask R-CNN is a state-of-the-art model for instance segmentation, and although newer models such as YOLOv8[35] have surpassed it in speed and accuracy for the COCO-dataset[21], the complexity and the size of the network give an advantage for corrosion detection. As the focus of the pre-project revolved around Mask R-CNN, it is the sole model trained and tested for the corrosion task. However, the methods discussed in future chapters also apply to other convolutional neural networks.

3.3 Implementation

In the pre-project, the research primarily relied on the Mask R-CNN implementation by Matterport Inc.[36], which is based on Python, Keras, and the TensorFlow framework. However, the current repository has not been updated since April 2019, which poses challenges due to subsequent upgrades in TensorFlow from version 1.9 to 2.12. Keeping up with the rapid advancements in the field of computer vision, adapting libraries, datasets, and custom functions to recent research becomes increasingly challenging, particularly

```
{
        "image_id": 0,
        "width": 200,
        "height": 200,
        "file_name": "path/to/image",
        "annotations" : [{
            "iscrowd": 0,
            "segmentation": [p1.x, p1.y, p2.x, p2.y, ... ],
            "bbox": [x, y, w, h],
            "bbox_mode": BoxMode.XYWH_ABS,
            "category_id": 0,
        }]
}
```

Figure 3.1: Example of one image written in the COCO-format

when working with outdated implementations, which is why this thesis disregards this version of Mask R-CNN.

Two new frameworks are instead proposed; Detectron2, developed by Meta AI[13] and MMdetection by MMlab[37]. Both are platforms based on PyTorch[38] for object detection and segmentation. They provide toolboxes and support a variety of models and algorithms and come with an easy-to-use interface and several pre-trained models, including Faster R-CNN and Mask R-CNN. Both platforms are continuously updated to support new models and library updates, which make them adaptable to recent and forthcoming changes in further research. In the end, it was an arbitrary choice between the two toolboxes, but as Detectron2 is developed by the same company behind Mask R-CNN and comes with easier documentation[39], it is the framework for all further neural network training. The implementation of Mask R-CNN using Detectron2 can be viewed in appendix B.

3.3.1 Dataset conversion

The Detectron2 Dataloader requires the COCO-format on all data, which is a JSON-file written like fig. 3.1. To ensure compatibility with the format, the original dataset, which consists of instances represented as binary mask images, needs to undergo a conversion process. The key step in this transformation is converting the binary mask images into polygon format before annotating them in the COCO format. The OpenCV library in Python[40] assists in creating the polygons from the mask images with the inbuilt function `find_contours()`. The full code implementation can be viewed in appendix A. For future work when using Detectron2, labelling should be automatically converted to COCO-format, as it is provided by most labelling softwares such as V7.

3.3.2 Choice of architecture and transfer learning

The model zoo in Detectron2 consists of a collection of pre-trained models that have been trained on large-scale benchmark datasets. These models cover a wide range of vision tasks, including object detection, instance segmentation, and more. Each model in the zoo is pre-configured with the network architecture, weights, and other necessary components for performing specific tasks. The theory behind transfer learning was explained in section 2.3 in the pre-project[7]. Instead of training models from scratch, which can be computationally expensive and time-consuming, Detectron2 provides pre-trained models that suit a variety of tasks. The pre-trained models are typically trained on large-scale datasets such as COCO (Common Objects in Context)[41].

To assess the performance of the network and evaluate the compatibility of the new library with the corrosion task, three main backbone architectures, namely ResNet-50, ResNet-101, and ResNeXt-101, are employed in conjunction with Mask R-CNN. These backbones, as described in the original Mask R-CNN paper[5], run with FPN and are pre-trained on the COCO dataset.

3.3.3 Choice of evaluation metrics

As IoU, and particular mIoU, has been the preferred method for evaluating instance segmentation both in previous literature and in the pre-project, class-wise IoU and mIoU will be reported for each tested model to obtain consistency to the previous work done. The same function for computing IoU described in section 4.2 of the pre-project[7] will be used, as Detectron2 does not produce IoU in its own evaluation. The instance segmentation predictions are all merged into one mask and evaluated the corresponding merged ground truth, as Fondevik developed it[6]. This was to avoid unreasonably bad IoUs because of difference in specific mask predictions.

Detectron2 does not automatically calculate validation loss or perform cross validation, the method where a new subset is chosen for validation per epoch to avoid overfitting. This is unfortunately a major hindrance, as it removes an important metric for evaluation. It is possible to extend the Trainer module to add custom loss calculations, but the version found was heavily reliant on the Detectron2 evaluator, which returned no usable results. The customization was therefore scrapped, but can later be fixed and used if possible. The training results and the custom IoU evaluator, including reviewing pictures, were the metrics used for this dataset.

3.4 Training process

The current training has not undergone significant tuning. To closely match the original version of the network run in the pre-project, a base learning rate was set to 0.0005 along with a learning rate scheduler set to halving the value at every tenth epoch. he confidence threshold is set to 0.8, and other parameters remain at default values. The training will span approximately 25 epochs. The number of epochs is derived from the number of iterations, which is determined by the batch size and the number of GPUs used. In this case, a single GPU with a batch size of 1 is employed, processing one image per iteration. The calculation of epochs can be determined using the equations in eq. (3.1).

$$IMAGES\_PER\_ITERATION = NUM\_GPUS * BATCH\_SIZE \tag{3.1a}$$

$$ITERATION\_PER\_EPOCH = \frac{TOTAL\_NUMBER\_IMAGES}{IMAGES\_PER\_ITERATION} \tag{3.1b}$$

$$ITERATIONS = EPOCHS * ITERATIONS\_PER\_EPOCH \tag{3.1c}$$

For 25 epochs, the corrosion detection model would then use 39600 iterations.

Backbone	Corrosion IoU	Background IoU	Mean Iou
ResNet-50	54.5%	84.1%	69.3%
ResNet-101	57.3%	86.1%	71.7%
ResNeXt-101	55.9%	84.9%	70.4%

Table 3.1: Current results from the Detectron2 Mask R-CNN on the validation dataset

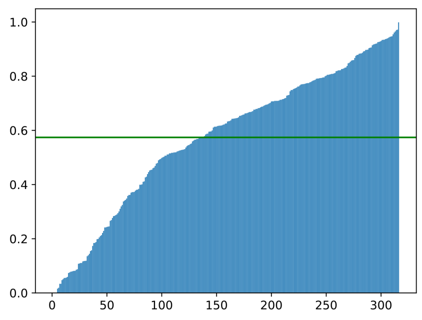


Figure 3.2: Corrosion IoU per image in validation set, with ResNet-101 backbone

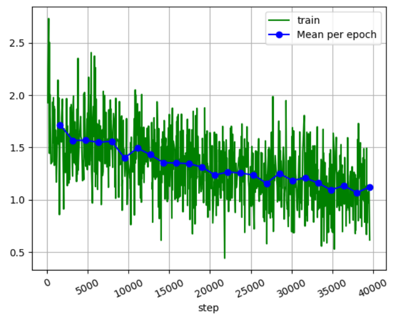
3.5 Results and evaluation of performance

Upon evaluating the validation dataset in table 3.1, it can be observed that the mean IoU ranges approximately between 69% and 72% for all three models. Additionally, the corrosion IoU indicates that 56% of the predicted corrosion mask aligns with the ground truth. These results already showcase an improvement compared to the findings presented in chapter 4 of the pre-project[7]. However, it is likely attributed to the difference in default hyperparameters between Detectron2 and Matterport.

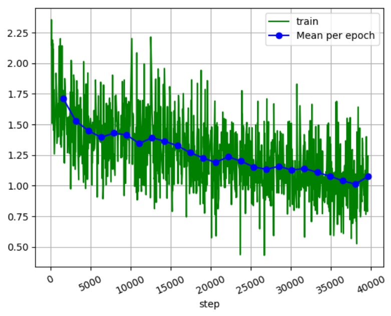
The corrosion IoU, which is the most critical metric, for each image in the validation dataset is illustrated in fig. 3.2. The analysis reveals that the majority of the images exhibit an IoU greater than 0.5, indicating a satisfactory level of accuracy in the predictions. However, the lower section of the figure presents a subset of 50 images with an IoU below 0.25. These images, some depicted in fig. 3.6, demonstrate instances where the model's predictions are notably inaccurate.

Another notable aspect regarding the current results is that the model performs comparably regardless of the backbone network. Considering the results obtained from the COCO dataset[5], one might expect ResNet-101 and ResNeXt-101 to outperform ResNet-50 when applied to a complex task like corrosion segmentation. This observation suggests a potential issue of overfitting across all networks, indicating a requirement for a more extensive dataset to effectively train the larger backbone networks.

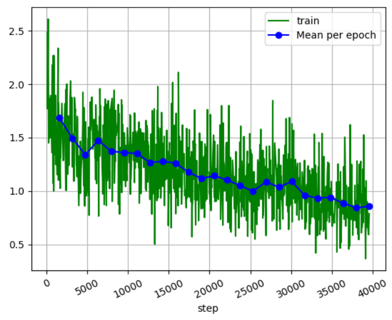
Evaluating the loss functions in fig. 3.3 provides several insights. Firstly, it is evident that there is noise in the output from each iteration due to the batch size of 1, causing the model to calculate the loss for a single image. This notion highlights the presence of outliers within the training dataset, where the model exhibits significantly better performance on certain subsets of images compared to others. While some training progress can be observed, it occurs at a relatively slow pace for all model backbones. Notably, the ResNeXt model demonstrates the most substantial reduction, with an average loss per epoch of approximately 1.7, ultimately converging to a loss below 1.0, a level not achieved by



(a) ResNet-50



(b) ResNet-101

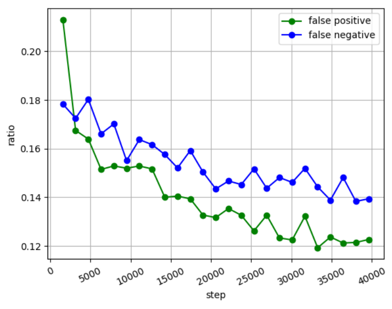


(c) ResNeXt-101

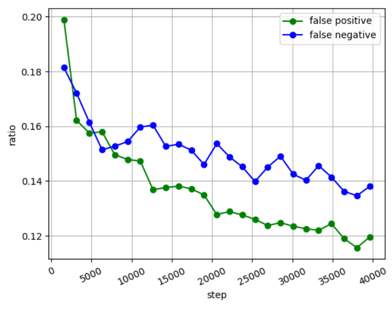
Figure 3.3: Total training loss per step with the backbone structure

any other models. An interesting aspect to consider is the average ratio of false positives and false negatives per image in one epoch. Both metrics play a crucial role in evaluating corrosion segmentation. False detections, such as paint streaks, clouds, and background elements, often lead to significantly low IoU scores. For instance, as seen in fig. 3.6, a flawed mask results in a remarkably low corrosion IoU of 0.8%. However, reducing false negatives is of greater importance since a corrosion network that fails to accurately detect corrosion becomes ineffective. In fig. 3.4, there is a general improvement across all networks, with the rate of false negatives slightly surpassing that of false positives. This observation aligns with the high confidence threshold applied. Generally, the ResNeXt-101 backbone performs the best, with a false negative rate of 0.138 and a false positive rate of 0.108, but the differences between the backbones are marginal. While a small percentage of incorrectly detected pixels can be tolerated due to inconsistent masking and corrosion annotations, the goal is to reduce false positives and false negatives to below 0.1 if possible.

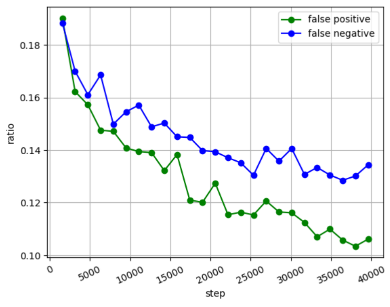
Several observations arise when evaluating the IoU over time in fig. 3.5. For ResNet-50, the accuracy fluctuates, while the IoU score remains stagnant for the two other models. The question remains as to why there seems to be limited improvement in IoU calculations compared to the advancements in training loss, with only moderate improvements observed in the ResNet-101 model. The reason might be again overfitting to some datatypes, which leads to underfitting for other datatypes in the image set.



(a) ResNet-50



(b) ResNet-101



(c) ResNeXt-101

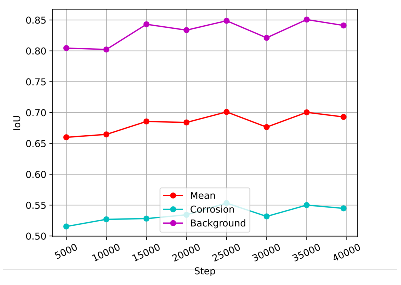
Figure 3.4: Ratio of false negatives vs false positives per epoch with the backbone structure

3.6 Dataset limitations

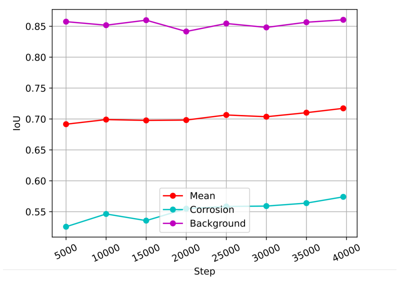
The inability to achieve an accuracy higher than 60% for corrosion masks may be closely tied to the organization and composition of the dataset. As mentioned in chapter 3 of the pre-project[7], the quality of the dataset can have a huge impact on the performance, and there are many factors to the current state of this dataset that makes it suboptimal. In the evaluation of the dataset for detecting corrosion, several faults have been identified and examples of these are visualized in fig. 3.6. These faults have implications for the performance of the model and highlight areas of improvement.

Firstly, the incorporation of other damages such as white corrosion or paint flaking into the dataset confuses the network's learning process. Although the dataset contains a limited number of images depicting these damages, the model is expected to detect them with the same level of accuracy as red corrosion. This imbalance introduces a bias within the dataset against red corrosion, resulting in the model struggling when presented with images showcasing other types of damage. The inclusion of other damages within the same class challenges the model's ability to correctly identify distinctive patterns associated with corrosion. Consequently, false positives may arise, such as misclassifying clouds or background elements as corrosion. Instead of focusing on specific characteristics like colour or structure, the model is trained to generalize that "anything irregular or contrasting with the background is corrosion."

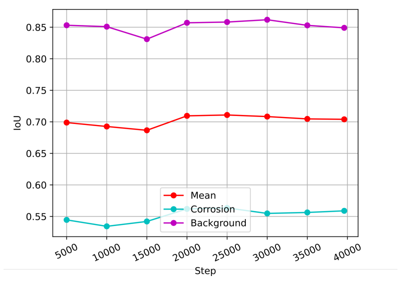
Secondly, the model faces challenges when dealing with images that depict corrosion at a distance. Since the model lacks an understanding of distance in images, it is unable to differentiate between close-up shots of corrosion and images taken from a distance showing the entire structure of a bridge along with background noise. This limitation can result in the model failing to detect subtle instances of corrosion and falsely identifying



(a) ResNet-50



(b) ResNet-101



(c) ResNeXt-101

Figure 3.5: Mean IoU over the iterations on the validation dataset

background elements as corrosion.

Furthermore, disturbances such as sunlight and shadows affect the quality of the images, leading to performance degradation. Degradation could be in the form of the model failing to detect corrosion because of poor lighting or the model falsely detecting the shadows themselves. Enhancing the image quality and avoiding images captured under unfavourable lighting conditions can contribute to alleviating this problem. Additionally, it is crucial to exclude images containing irrelevant objects like tools or humans, which can introduce noise and confuse the model.

Lastly, incorporating more worst-case scenarios into the training dataset is essential. The model struggles to accurately identify corrosion-like patterns, which could be caused by gravel, like in fig. 3.6, moss, or forest elements.

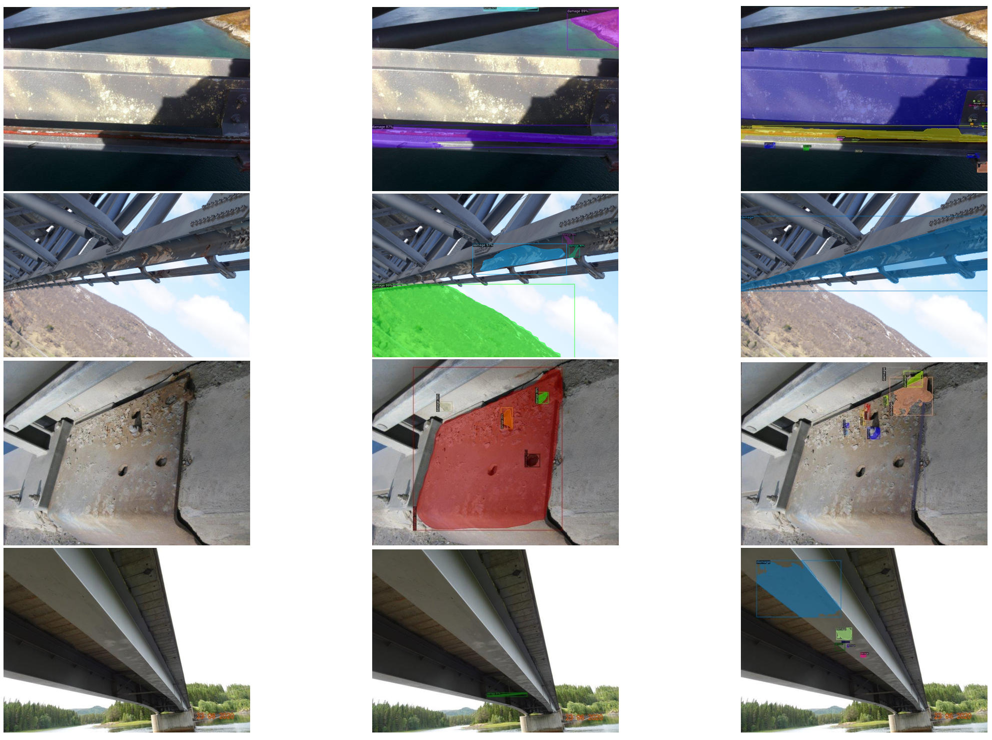


Figure 3.6: Examples of predictions on the validation dataset, ResNet-101+FPN

Chapter 4

Methods and Implementation

4.1 Dataset reduction and augmentation

To address the issues outlined in the discussion section 3.6, it is crucial to ensure the quality of the images before proceeding with the implementation of strategies. One of the objectives is to find solutions for detecting white corrosion or blistering in the images, as well as improving the detection of other types of damages that the current model fails to predict accurately.

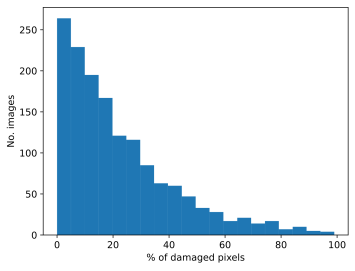
The implementation of a multi-class model has been suggested as a potential solution both by Fondevik[6] and in section 5.2 in the pre-project[7]. However, developing such a model requires an evenly distributed dataset in terms of classes. Therefore, it is necessary to carefully assess the dataset and address any issues related to class imbalances of certain damage types. When manually going through the dataset, 100 images are denoted as "other damages". The amount is not a good enough distribution, and implementing multi-class would not be efficient enough. The short-hand solution is to remove these 100 images from the dataset completely in order to properly assess the improvement of the accuracy, making the model focus more on red corrosion, ignoring other damages.

Removing these images could introduce some other potential challenges. One is that only images where the dominant damage present was not red corrosion. There are still several images with smaller masks that correspond to other damages present, meaning that the model will struggle even more to correctly detect them. The removal could potentially lower the IoU for some images with several different types of damages, but as the masks are scarce and small, it shouldn't affect the mean IoU on a significant scale.

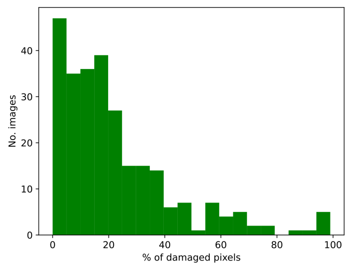
Another challenge is that reducing the dataset increases the risk of overfitting. Although the changes sharpen the model towards red corrosion, it could also make it more vulnerable to corrosion-like background. The ideal solution would be to incorporate more worst-case scenarios and more varied backgrounds with red corrosion, but this study is of course limited by the images at disposal.

The final dataset which was used for the corrosion detection model consisted of a total of 1772 images, with 1503 for training and 269 for validation, in contrast to the old dataset which consisted of 1996 images. The distribution of instance sizes, compared to the size of their images, is shown in fig. 4.1.

Data augmentation served as a temporary solution to compensate for the reduced dataset, mitigating the scarcity of samples. While there exist numerous approaches to augmenting images, the vastness of the search space necessitates a focused selection based on previous research findings [42, 43] and traditional methods.



(a) Train



(b) Validation

Figure 4.1: Histogram of dataset and percent of corroded pixels (bin=20)

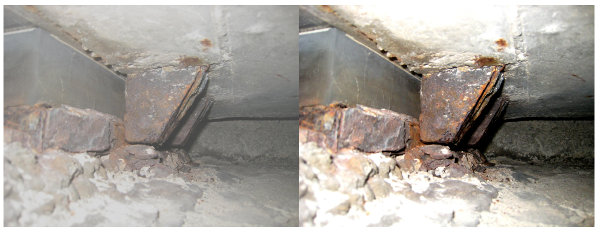
Detectron2 provides a comprehensive set of built-in augmentation methods. Leveraging these functions, we devised two augmentation strategies tailored for our model. The first strategy, denoted as **light augmentation**, consists of conservatively flipping the images horizontally and vertically. Considering that corrosion is generally invariant to direction and the corrosion dataset encompasses images captured from various angles, this scheme serves as a simple yet effective approach to expanding the number of images for training, and represents the most fundamental configuration of the corrosion detection model.

In the second strategy, light augmentation is combined with a more advanced scheme incorporating color transformations, denoted as **heavy augmentation**. This approach aims to enhance the model's resilience to variations in lighting conditions, which can significantly impact the appearance of corrosion patterns. Given that corrosion exhibits variations in red colors and is often influenced by outdoor environmental factors, this augmented scheme holds potential for improving the model's generalization capabilities. The augmentation pipeline comprises various transformations, exemplified in fig. 4.2, which facilitate adjustments to pixel values, contrast, brightness, and other relevant image attributes. The two strategy schemes can be viewed in table 4.1, with the parameters for light augmentation showing the probability of an image being flipped, while for heavy augmentation they show the range of color adjustment, with a random value chosen for every image.

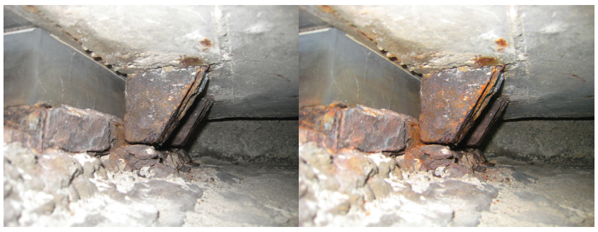
Augmentation	**Value**
Light augmentation	
Resize	800x800
Horizontal flip	0.5
Vertical flip	0.5
Heavy augmentation	
Brightness	0.8-1.8
Contrast	0.6-1.3
Saturation	0.8-1.4

Table 4.1: Augmentation schemes. Heavy augmentation

(a) Brightness



(b) Contrast



(c) Saturation

Figure 4.2: Heavy augmentations of one image

4.2 Genetic Algorithm

The Genetic Algorithm was here specifically implemented for hyperparameter optimization of the Mask R-CNN network, with the implementation inspired from Gerbet et al.[44].

4.2.1 Chromosome

The full list of hyperparameters available for Mask R-CNN in Detectron2 can be found in the configuration file[39], where the most relevant have been chosen as a chromosome. The config file differs from the Matterport version used in previous work, and some parameters were not present in the file, which means the experiment can't fully replicate the project from Gerber et al.[44]. Most of the chosen hyperparameters focus on either the Region Proposal Network, the first stage of Mask R-CNN, or the ROI (Region Of Interest) heads, which are the two branches for predicting boxes and masks at the end of the model. Other relevant hyperparameters not specific to Mask R-CNN are learning rate, learning momentum, weight decay, epochs and image dimensions. The hyperparameters chosen specific to Mask R-CNN are:

- **RPN NMS threshold:** The threshold of the IOU between two proposal boxes. The proposal with the highest confidence score is compared to all other proposals one-by-one. If the IOU is higher than the given threshold, the proposal will be removed to avoid several proposals detecting the same object.

- **RPN batch size:** Number of region proposals per image in the RPN
- **RPN Pre NMS limit:** Upper limit for number of top scoring RPN proposals to keep before applying NMS
- **RPN Post NMS limit:** Number of top proposals of RPN to keep after applying NMS, there are two separate values for training and inference
- **ROI batch size:** Number of region proposals in ROI heads.
- **ROI positive ratio:** target fraction of foreground boxes per ROI batch
- **Detection confidence threshold:** The threshold of confidence for the final box predictions, all box predictions below this threshold will be removed. Removes potential false positives, but at the cost of potentially removing true positives with a low confidence score.
- **ROI IoU threshold:** Overlap threshold for an ROI to be considered foreground (if $>=$ threshold)

The most relevant effects these hyperparameters have on the predictions are the number of the box predictions generated and the threshold of classifying the boxes as objects. It is uncertain of different combinations of the hyperparameters work together, and we have yet to see the effect they have on the accuracy of corrosion detection. The complete chromosome and search space can be viewed in table 4.2. The values from the rust maize project[44] decide the the intervals for the hyperparameters. The table is almost a copy of table 5.1 from the pre-project[7], but differs in that the hyperparameters not available in Detectron2 have been removed.

RPN_NMS_THRESHOLD	0.5-1
RPN_BATCH_SIZE	[64, 128, 256, 512, 1024]
PRE_NMS_LIMIT	4000-8000
POST_NMS_ROIS_TRAINING	1000-3000
POST_NMS_ROIS_INFERENCE	600-2000
ROI_BATCH_SIZE	[64, 128, 256, 512, 1024]
ROI_IOU_THRESHOLD	0.3-0.7
ROI_POSITIVE_RATIO	0.3-0.8
DETECTION_MIN_CONFIDENCE	0.3-0.9
LEARNING_RATE	0.0001-0.001
LEARNING_MOMENTUM	0.75-0.95
WEIGHT_DECAY	0.00007-0.000125
EPOCHS	20-40
IMAGE_MIN_SIZE	500-1000
IMAGE_MAX_SIZE	700-1200

Table 4.2: The hyperparameters chosen for the Genetic Algorithm and their value intervals

4.2.2 Dataset

Using the Genetic Algorithm for hyperparameter optimization involves training a separate network for *N* individuals for *M* generations, resulting in a significantly longer training period compared to conventional training approaches. To mitigate the computational and time requirements associated with this extensive training process, a reduced subset of the available images is employed. Specifically, a subset consisting of 200 images was used for training, while an additional set of 40 images was allocated for validation purposes. This reduced subset strikes a balance between expediting the training process and providing the model with sufficient data to adapt effectively.

4.2.3 Fitness function

Given that the primary objective of this thesis is to enhance the mean Intersection over Union (IoU) metric, the evaluation of IoU serves as the fitness function employed within the Genetic Algorithm. The particular emphasis is placed on assessing the mean corrosion IoU for the validation dataset, as it represents the most crucial metric for evaluating the performance of the algorithm. By prioritizing the mean corrosion IoU, the algorithm focuses on achieving accurate and precise segmentation results specifically for corrosion, which is a critical aspect in the context of this research.

4.2.4 Algorithm pipeline

The pipeline of the algorithm is relatively simple, shown in fig. 4.3 for generation *i* with a population of 4. The same pipeline is used for our run of the algorithm, using the parameters in table 4.3. An initial population of 20 with random hyperparameter values define 20 separate models that will each train on the small dataset independently. The model chosen for training is ResNet-50, as a smaller model is needed to avoid overfitting for the 200 images. The algorithm then evaluates the models on the validation dataset and calculates sorts the fitness scores. When a new generation is made, the selection operator chooses the best half of the previous population, also called the elite, by the highest corrosion IoU. The elite automatically continues to the next generation without any change, and they will also be the parents of the new half of the population.

When the crossover operator creates a new individual, two random individuals from the elite are chosen as the parents, and the new individual inherits one of their value, chosen at random. The individual then undergoes mutation. With a mutation rate of 0.2, there is a 20% chance of a gene being mutated. This mating process is repeated until the algorithm works with a new population of 20 individuals, ready for the next training and evaluation round. At the last generation the fittest individual's hyperparameters will be chosen for the complete corrosion detection model. Either after 15 generations or the fitness function has reached a minima threshold, which in this case is a corrosion IoU of 0.65. This threshold value could also be experimented with, but with the few hyperparameters available, it is not likely that the IoU will rise significantly beyond the point of 0.65.

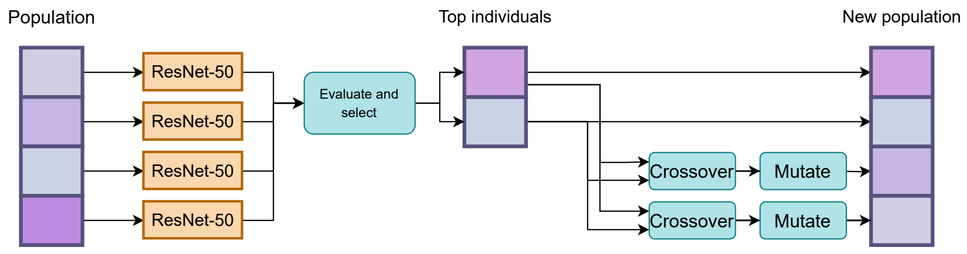


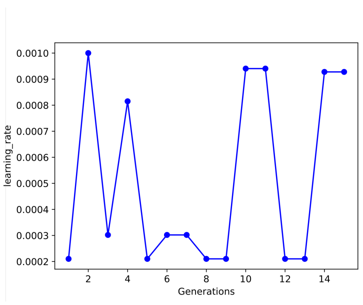
Figure 4.3: The Genetic Algorithm run per generation

Generations	15
Population	20
Selection rate	0.5
Mutation rate	0.2
Train set size	200
Val set size	40

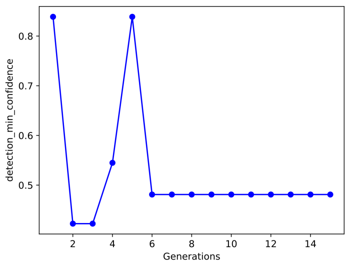
Table 4.3: Genetic Algorithm parameters

Run with standard mutation

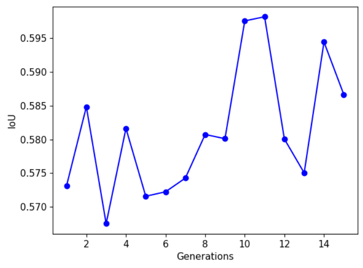
The simple GA mutation function takes 20% of the hyperparameters and change it with a random value in the range set by table 4.6. It was however discovered with small experiments that using this type of mutation led to divergence in the training. Evaluating the graphs in fig. 4.4, the corrosion IoU reaches its maximum in generation 11, but it is then reduced in the succeeding training, meaning the values from the last generation are not the optimal hyperparameters. Some gene values such as the minimum confidence manages to converge to a single value, but the learning rate continuously diverges. the performance can suggest that the mutation is too rapid for such a small population, so the algorithm never finds back to the optimal value after a mutation.



(a) Learning rate



(b) Minimum confidence

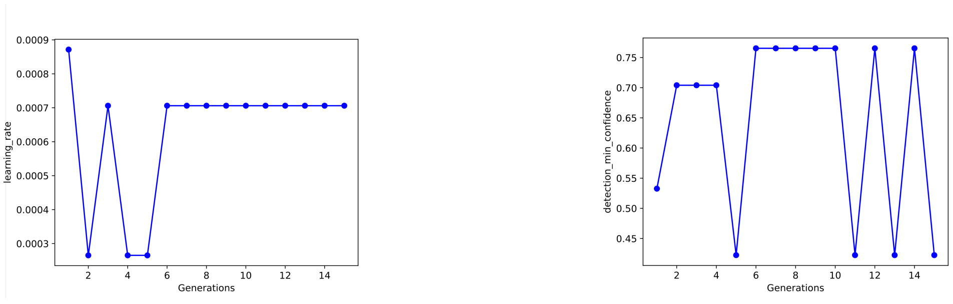


(c) Corrosion IoU

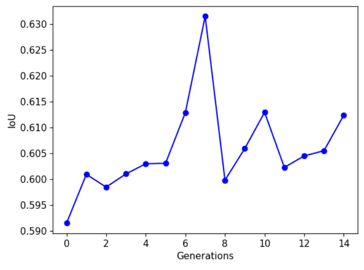
Figure 4.4: Excerpt of genes and results from Genetic Algorithm testing with simple mutation

Run with adaptive mutation

To further ensure that the algorithm converges to the optimal hyperparameters, a more complex mutation function has also been added. Firstly mutation is added after a child has been created from two parents, and the child's gene is then mutated in a range from that given value, instead of it being given a completely new random value. The range of mutation is decided by the current generation, meaning that the higher the generation number, the smaller the mutation range gets. Using this type of logic avoids too high divergence as the algorithm should be free to explore the search space in the beginning, but get closer to the optimal values in the later generations. The graphs in fig. 4.5 show some convergence for the learning rate, while the confidence threshold fluctuates between two values. Comparison of the standard and adaptive mutation shows however that the adaptive is better suited for hyperparameter tuning, and will be used in the tuning going forward. For further details the updated mutation function can be seen in appendix C.



(a) Learning rate　　**(b)** Minimum confidence



(c) corrosion IoU

Figure 4.5: Excerpt of genes and results from Genetic Algorithm testing with adaptive mutation

4.2.5 Results

Analyzing the evaluation of the best performing individuals in fig. 4.5c there is again a president for the Genetic Algorithm not converging towards the optimal value, as it reaches its maximum in generation 7 before quickly deteriorating. The lack of convergence could again be that the population or number of generations is too small to accommodate for the mutation rate. The corrosion detection model will therefore run with results from generation 7, which produces the highest corrosion IoU. The best hyperparameters produced by the algorithm can be viewed in table 4.6.

4.3 Sky segmentation and removal (SkySeg)

This section introduces a novel network, referred to as SkySeg, designed specifically for detecting and removing the sky from the corrosion dataset. The implementation involves selecting, training, and evaluating a separate model independent of the main corrosion detection model. SkySeg has not been extensively optimized for achieving 100% accuracy in sky removal, as doing so would require significant time and resources and is not part of this thesis' main objectives. Instead, the goal is to attain a satisfactory level of accuracy to evaluate the impact of removing the sky from the images and assess if the predictions improve as a result.

4.3.1 Model

SkySeg implements YOLO as its model for instance segmentation of the sky regions. The requirements for a background removal algorithm are primarily efficiency and accuracy. It should add little complexity to add this type of image processing in the damage detection pipeline. The model should be accurate enough in order to not confuse the corrosion detection model any further. YOLO provides remarkable accuracy at a fast rate. It is also cutting edge and easy to implement, making it a contender for future networks to evaluate and use in future projects. Originally YOLO was only intended for box predictions, but the fifth version of the network, YOLOv5, added a mask head identical to the fully connected network in Mask R-CNN to the architecture, making YOLO able to do instance segmentation. The newest version, YOLOv8, was released in January 2023 by Ultralytics[35]. The version further enhances performance and simplifies the process, making the library the suitable choice for further development of SkySeg.

4.3.2 Dataset

As it is time-consuming to collect and annotate a completely new and separate dataset for sky segmentation, data acquisition was instead spent on finding datasets where a sky class already existed. The choice landed on using the ADE20K dataset from MIT CSAIL[45]. The dataset is originally created for semantic segmentation, but it is possible to use the dataset for instance segmentation as well[46]. Originally ADE20K composes of 27,574 images (25,574 for training and 2,000 for testing) spanning 150 classes from indoors and outdoors objects, such as walls, floors, grass, persons and mountains. The dataset presents annotation with a gray scale image mask for each image, where the pixels have one assigned value between 0 and 150 (0 for background, and the number 1-150 corresponding to the given pixel's class). An outdoor image and its mask can be viewed in fig. 4.6.

Out of the total number of images, 9,040 of them contains the class "sky". These specific images were selected in the creation of a new dataset for training the model. To create a single-class dataset, the segmentation masks, as in fig. 4.8a, are reduced to a binary segmentation mask similar to the ones in the damage dataset, with all sky pixel values set 255(white), and the rest set to 0 (black), as seen in fig. 4.8b.

To convert the masks into the YOLO format, seen in fig. 4.8c, each mask is transformed into a text file. The YOLO format uses a line-by-line representation, where each line contains the class number ("0" for sky) followed by the normalized coordinates of the polygon. The dataset metadata, including the paths to the datasets, can be stored in a YAML file[47] for easy organization, exemplified in fig. 4.7. There is a possibility to extend this network to classify more background elements such as water or forest, both of which are present in the ADE20K dataset, but due to time constraints this thesis focused on a

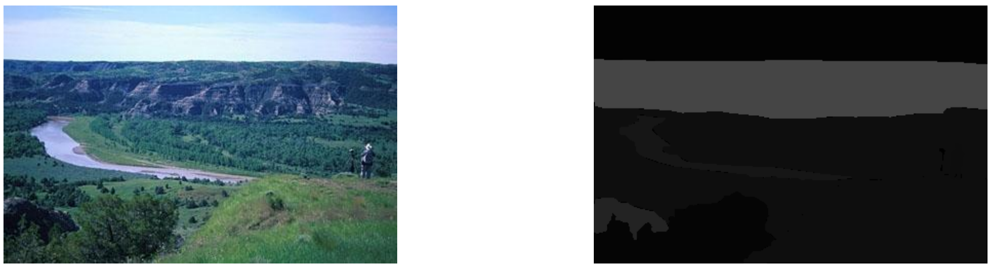

(a) Image **(b)** Corresponding mask

Figure 4.6: Example from ADE20K dataset

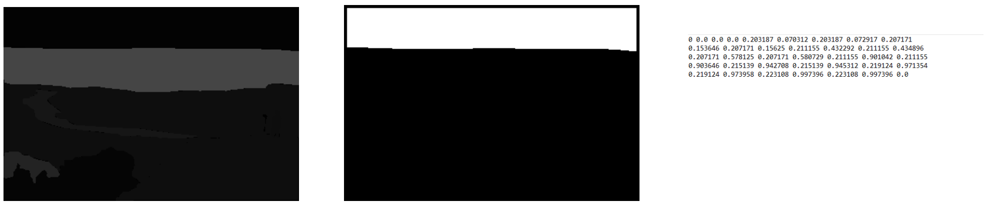
```
path: /cluster/home/helensem/Master/sky_data # dataset root dir
train: images/train  # train images (relative to 'path')
val: images/val  # val images (relative to 'path')
test: /cluster/home/helensem/Master/damage_data # test images

# Classes
names:
  0: sky
```

Figure 4.7: Metadata for SkySeg

single class network. The code implementation of the full dataset conversion in fig. 4.8 can be seen in appendix A.

Additional dataset

To extend the network capabilities to detect sky where a bridge was present, a set of bridge images were added to SkySeg dataset. Adding relevant images aims to mitigate issues with the current ADE20K dataset and narrow the SkySeg model towards the corrosion dataset. The outdoor images are mostly with a straight horizon, where the sky always occurs in the upper parts of the image. This is not the case for many bridge images, where the sky can appear both above and under the bridge, as in fig. 1.3 and fig. 3.6. ADE20K is a dataset originally meant for *semantic* segmentation, and consists mostly of one instance


```
0 0.0 0.0 0.0 0.203187 0.070312 0.203187 0.072917 0.207171
0.153646 0.207171 0.15625 0.211155 0.432292 0.211155 0.434896
0.207171 0.578125 0.207171 0.580729 0.211155 0.901042 0.211155
0.903646 0.215139 0.942708 0.215139 0.945312 0.219124 0.971354
0.219124 0.973958 0.223108 0.997396 0.223108 0.997396 0.0
```

(a) Original mask **(b)** Transformed binary mask **(c)** YOLO format of the mask

Figure 4.8: The pipeline of converting a ADE20K mask to YOLO format. (a) The original mask corresponding to the image in fig. 4.6. (b) The instance mask for the class sky is located, all other classes are removed, making the image a binary mask. (c) The polygon coordinates are normalized and written to a text file in YOLO format.

per image. SkySeg works with *instance* segmentation where the sky regions can appear in several parts of the image and be occluded by the bridge, as in section 3.5.

The additional set of bridge images are from earlier bridge inspections on the Fredrikstad bridge, provided by Orbiton[12]. These are images where no corrosion is present, but are useful in that parts of the sky are occluded by bridge parts. Images were annotated in Roboflow[48], which is powered by Meta AI's new segmentation tool; the Segment Anything Model (SAM)[49]. The model has been trained on a dataset of 11 million images and 1.1 billion masks, allowing for some of the fastest annotation tools today. The example in fig. 4.9 shows how easily the model recognizes the correct polygons, making labeling easy and less time consuming. 156 images of the bridges were annotated and added to the train dataset.

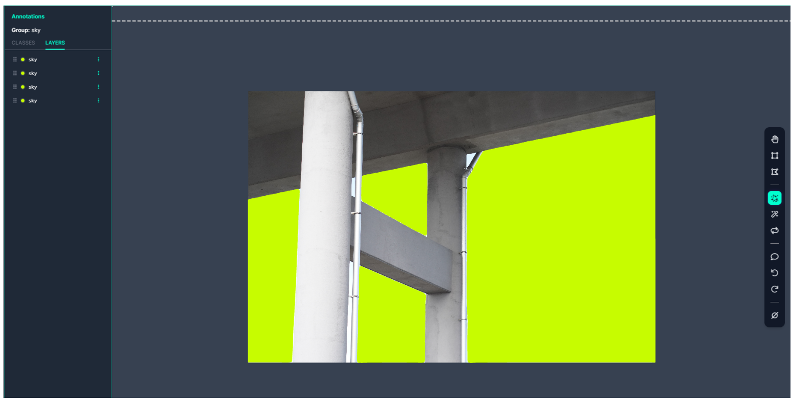


Figure 4.9: Labeling with Roboflow

4.3.3 Training and testing

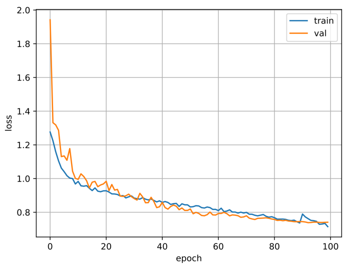
The YOLO network runs with custom sky dataset using the default hyperparameters described in table 4.4. To verify that the model meets the standards of sky segmentation, the results from the validation dataset are briefly evaluated. From the predictions in fig. 4.11 the model performs up to standard when detecting the sky and the loss functions in fig. 4.10 show an optimal training loss decline for the network, with the validation loss showing no signs of overfitting.

Epochs	100
Optimizer	SGD
Learning rate	0.001
Image size	640
Batch size	16
Confidence threshold	0.6

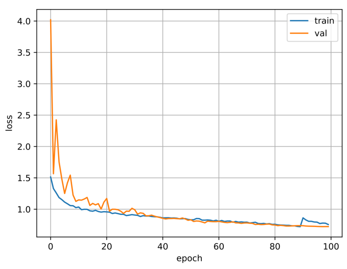
Table 4.4: Hyperparameter values for YOLO network

4.3.4 Background removal

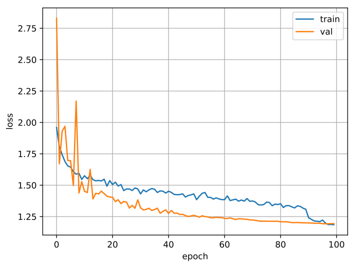
The weights from the pre-trained sky segmentation model will be used to predict masks on any incoming picture. The sky predictions with a confidence higher than a given threshold



(a) Localization loss



(b) Class loss



(c) Segmentation loss

Figure 4.10: Loss functions from training YOLO on the sky dataset

will be used to create binary masks, 0 for sky and 1 for background, which will be multiplied with the original image to zero out any pixel with the sky in it, using the OpenCV library[40]. The process is shown in fig. 4.12 and the implemented code can be seen in appendix D, where the finished input to the corrosion detection model should look like fig. 4.12c.

4.4 Experiment

The experiment table is displayed in table 4.5, each block represent one evaluation. LA denotes light augmentation and HA denotes heavy augmentation.

Backbone	Feature extractor	Method			
		LA	HA	GA-tuned	SkySeg
ResNet-50	DC5				
	FPN				
ResNet-101	DC5				
	FPN				
ResNeXt-101	FPN				

Table 4.5: Total outline over all experiments

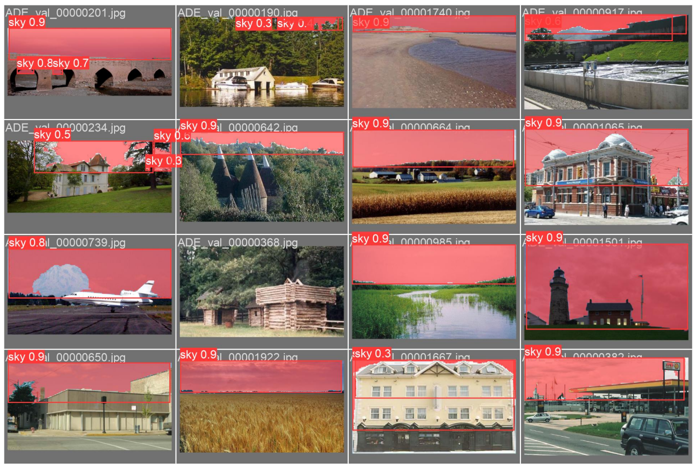


Figure 4.11: Predictions from the SkySeg model on the validation dataset

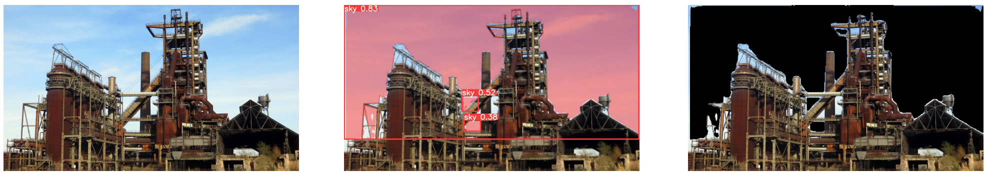

(a) Original image (b) With sky predictions (c) Masking

Figure 4.12: Background removal with SkySeg

4.4.1 Backbones

The corrosion detection model will train with three backbone architectures: ResNet-50, ResNet-101, and ResNeXt-101 (with a cardinality of 32). FPN will serve as the main feature extraction method, but the model will also undergo testing with dilated convolutions, specifically DC5 as discussed in section 2.1.3. This approach aims to widen the scope of the model and assess if there exist any simple techniques that can be employed to improve the network's accuracy.

All these models are available in the model zoo of Detectron2 with pre-trained weights from the COCO dataset[41], as explained in section 3.3.2. ResNext architectures do not have the option of dilated convolutions in Detectron2 as of now, which results in 5 different combinations of backbone structures, which all will be tested with the different strategies below.

4.4.2 Methods

This chapter has so far explained three separate methods, data augmentation with light and heavy schemes, hyperparameter optimization and SkySeg, which results in four independent evaluations on the validation dataset with the corrosion detection model.

Augmentation

Mask R-CNN will train once with light augmentation and once with heavy augmentation. Since the light augmentation scheme is as conservative as it is, it serves as the base of comparison to both the reduction in the dataset and between the backbones. Based on the reduced dataset, it makes however more sense to compare every strategy to results from light augmentation in table 5.1 than those from chapter 3.

Other than augmentations, the model will run on default configuration with little tuning done on the hyperparameters. The only change done between the two augmentation schemes is that the batch size was increased from 1 to 2, with the only noticeable consequence being the number of iterations was cut in half. Both models were trained on 25 epochs with the same learning rate scheduler from chapter 3, with a base rate of 0.0005 which was then halved by the 10th and 20th epoch. All the other hyperparameters can be seen on the left side of the table in table 4.6.

GA tuning

The results from GA will in the end be a comparison of hyperparameters. Looking at table 4.6 the backbone models with normal augmentations will run on default configurations, while a new training run will be done with the configurations resulted by tuning with GA. The hyperparameter values differ quite from another, the goal will be to assert both the effectiveness of changing hyperparameters in regard to corrosion detection, and the effectiveness of the Genetic Algorithm for this goal.

SkySeg

This experiment differs from the others in that no network training is done, with the only variation being the inclusion of SkySeg as a pre-processing step for the validation dataset. Training with SkySeg as a part of the model may result in reduced robustness and potential false positives, which could negatively impact the images. Therefore, it is more appropriate to train the dataset using normal configurations and evaluate the performance on the validation dataset to observe if the model improves in focusing on the structure rather than the background. The only results retained from this experiment are the mean IoU and prediction outputs from the validation dataset.

Parameter	Default value	GA value
RPN_NMS_THRESHOLD	0.7	0.642
RPN_BATCH_SIZE	256	1024
PRE_NMS_LIMIT	12000	6857
POST_NMS_ROIS_TRAINING	2000	2224
POST_NMS_ROIS_INFERENCE	1000	885
ROI_BATCH_SIZE	512	128
ROI_POSITIVE_RATIO	0.25	0.49
ROI_IOU_THRESHOLD	0.5	0.35
DETECTION_MIN_CONFIDENCE	0.8	0.76
LEARNING_RATE	0.0005	0.0007
LEARNING_MOMENTUM	0.9	0.95
WEIGHT_DECAY	0.0001	0.00012
EPOCHS	25	29
IMAGE_MIN_SIZE	800	989
IMAGE_MAX_SIZE	1333	1148

Table 4.6: Hyperparameters comparison between the default values and those used in the GA experiment

4.4.3 System

All experiments were run on the Idun cluster provided by the High Performance Computing Group at NTNU[50]. The cluster provides a variety of cores and GPUs, meaning the hardware configurations varied for each training. Table 4.7 shows the configuration most often used by Idun, while software modules and libraries are shown in table 4.8.

CPU	Intel Xeon Gold 6148 (20 cores @ 2.40 GHz)
GPU	NVIDIA V100 tensor core (16GB)

Table 4.7: Hardware configuration

Compiler	GCC v.11.3.0
Python	v.3.10.4
Pytorch	v.1.13.1
Torchvision	v.0.14.1
CUDA	v.11.7.0
CuDNN	v.8.4.1

Table 4.8: Software configurations

Chapter 5

Results and Discussion

This chapter presents the most relevant results and findings from the 15 training schedules and 20 evaluations done over the course of this thesis. Additional plots related to the network training with light augmentation and GA tuning can be found in appendix E. The three first sections discuss the three different strategies on how they compare to results presented in chapter 3. The summary in compares the strategies to each other and across the backbones, with the key takeaways from the evaluations. Readers that are only interested in the main results may see the mean IoU for all evaluations in table 5.5.

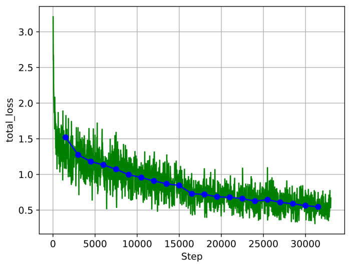
5.1 Augmentations

5.1.1 Light augmentation

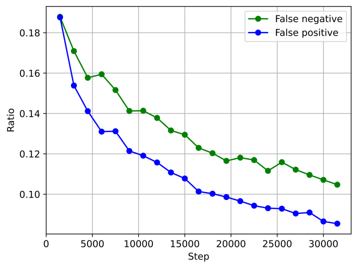
Mask R-CNN was trained using only vertical and horizontal flipping on 25 epochs with a batch size of 1, with the simple default configurations from Detectron2. Resulting class-wise IoU and mean IoU on the validation dataset are listed in table 5.1. Results from the backbone model, ResNet-101+DC5 are plotted in fig. 5.1, where the loss function presents per batch loss and per epoch loss. Predicted segmented masks are visualized in fig. 5.2.

Backbone	**Corrosion IoU**	**Background IoU**	**Mean Iou**
ResNet-50 + FPN	61.4%	86.5%	74.0%
ResNet-101 + FPN	61.2%	86.2%	73.7%
ResNeXt-101 + FPN	62.0%	87.0%	74.5%
ResNet-50 + DC5	60.9%	87.0%	73.9%
ResNet-101 + DC5	62.5%	87.9%	75.1%

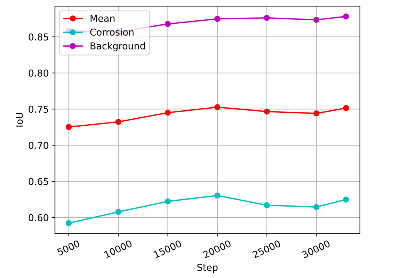
Table 5.1: Results on default configuration, Light augmentation



(a) Total training loss



(b) False predictions



(c) IoU per iteration

Figure 5.1: Plots from default run, Light augmentation, ResNet-101+DC5

Discussion

By comparing table 3.1 and table 5.1, notable improvements can be observed in the FPN models. The results indicate that by reducing the dataset, there is an average increase in mean IoU of 6.78% for ResNet-50, 2.79% for ResNet-101, and 5.68% for ResNeXt-101. This improvement can largely be attributed to the removal of images that previously resulted in the poorest performance by the model. For the FPN models, the best performing by a small margin is ResNeXt-101, which makes sense due to the complexity of the model compared to ResNet-50, and its performance being the best in the original paper on Mask R-CNN[5]. It is however disappointing to see such a small increase in performance between ResNet-50 and ResNeXt-101 with doubling of layers and higher complexity.

By introducing dilated convolutions, there is no remarkable improvement for ResNet-50, but sees another 1.9% increase for ResNet-101 in mean IoU, which makes ResNet-101 with DC5 the best-performing model. As FPN is a more complex convolutional operator than DC5 and is used more widely in literature on Mask R-CNN[5, 21, 23], it was initially expected that FPN would provide better results for the corrosion model. The expectation was that FPN's ability to capture features at various scales would benefit the model by providing more comprehensive information about the image.

However, the results suggest that a larger receptive field, which can capture a broader context of the regions of interest, may have advantages when distinguishing corrosion from the surrounding environment. In the comparison in fig. 5.2 the DC5 predicted images show both a smaller degree of false positives and more coherent masks than with the standard FPN. Analyzing the plots from ResNet-101+DC5 in fig. 5.1, the standard loss curve falls to an impressing 0.5 compared to the models in fig. 3.3 which barely reached below 1. The amount of false positives has also been reduced to below 10%, which was one of our desired goals.

The ratio of false negatives leaves still a lot to be desired, and there are some images that the model struggles with, especially those still with masks from the other damages mentioned earlier. Even if the model has improved on misclassification, it seems to still fail on some hard-to-spot corrosion details. This could be altered by lowering the confidence threshold. The challenges with the validation dataset can also be seen when evaluating the IoU over the epochs in fig. 5.1c, as this model, as the previous models in fig. 3.5, fluctuates in accuracy over the iterations. The reason could be that the model generalizes to one type of image, which in consequence leads it to fail on other types of images. It could be improved by increasing the number of epochs.

All in all, DC5 outperforms FPN in general as a feature extractor for the assignment of corrosion detection, with ResNet-101 as the preferred backbone. FPN does still have some advantages such as high resolution, which is why it may be useful to incorporate it in the architecture alongside DC5, meaning the model extract feature maps from each convolutional block, but use dilated convolutions when extracting. Guan et al. design a similar backbone for the Faster R-CNN network, called dilated convolutional feature pyramid network (DCFPN)[51], for the purpose of detecting thigh fracture. In the new design, the network works as described for FPN[22], but between the stages in the backbone network, normal convolutional blocks are replaced with dilated bottle blocks to enlarge the receptive field of each feature map extracted. The method outperforms the traditional Faster R-CNN with normal FPN. For future research, this might be an extension worth implementing to improve detection of small corrosion spots.

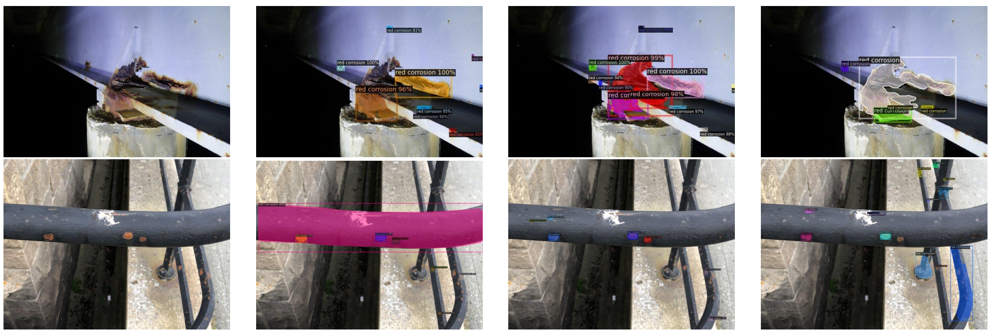


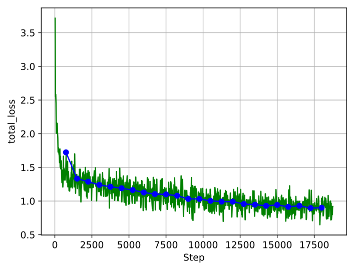
Figure 5.2: Comparing predictions on ResNet-101 backbone

5.1.2 Heavy augmentation

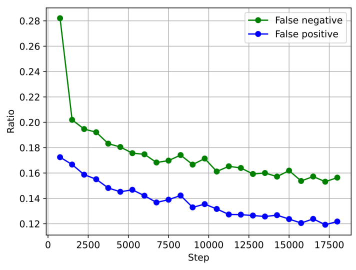
Mask R-CNN was trained using the heavy augmentation scheme in table 4.1 on 25 epochs with a batch size of 2, with the simple default configurations from Detectron2 and a learning rate scheduler. The resulting mean IoU on the validation dataset is listed in table 5.2. Results from the backbone model, ResNet-101+DC5 are plotted in fig. 5.3, where the loss function presents per batch loss and per epoch loss.

Backbone	Corrosion IoU	Background IoU	Mean Iou
ResNet-50 + FPN	58.0%	86.1%	72.0%
ResNet-101 + FPN	59.1%	86.2%	72.6%
ResNeXt-101 + FPN	58.9%	86.4%	72.7%
ResNet-50 + DC5	58.5%	85.7%	72.1%
ResNet-101 + DC5	59.6%	85.5%	72.5%

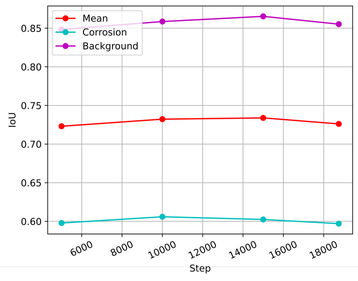
Table 5.2: Results on normal configuration, Heavy augmentation



(a) Total training loss



(b) False predictions



(c) IoU per iteration

Figure 5.3: Plots from default run, Heavy augmentation, ResNet-101+DC5

Discussion

Comparing table 5.1 and table 5.2 shows reduced accuracy for heavy augmentation on all backbone models. When evaluating ResNet-101+DC5 infig. 5.3, the overall performance is degraded, with a larger amount of misclassifications than the light augmentation scheme in fig. 5.1 and less improvement in IoU over the iterations.

The model does also seem to not train very well, showing a clear effect of underfitting when comparing the loss function in fig. 5.1 and fig. 5.3. This seems to be an indication that the variation and colour augmentation of the dataset has grown beyond what the model manages to adapt to. As these types of data augmentation are uncontrolled and performed by Detectron2, it is difficult to fully evaluate the quality of the augmented images and which of the techniques are failing. It could likely be that for example lowering the contrast removes the visible borders between corroded and non-corroded material and that the heavy saturation leads to unnatural images.

Comparing the predictions between light augmentation training and heavy augmentation training also sees a decrease in quality predictions. Taking a few sample images in fig. 5.4 it's evident that the heavy augmentation scheme has led the model to fail in generalizing corrosion patterns, and simply detects large masks of saturated areas. Even with the same confidence threshold of 0.8, the heavy augmentation scheme leads to large amounts of false positives.

As a result, we have decided to discard this particular augmentation approach, which expands the range of potential data augmentation techniques that can be considered to enhance the dataset. One alternative is to explore other geometric transformations, such as rotation, or adjust the scale of colour transformations to make them more versatile. Additionally, more advanced options include generating synthetic data using techniques like Generative Adversarial Networks (GANs)[52]. Furthermore, if the goal is to implement multi-class detection for less common damage types, oversampling techniques like Synthetic Minority Over-sampling Technique (SMOTE)[53] or class weights[54] can be considered. Just as an excessive amount of any substance can be toxic, the same can be said for a deep learning approach. While data augmentation is a widely recognized and effective technique in machine learning, it is essential to carefully consider its purpose, the model being used, and the characteristics of the dataset.

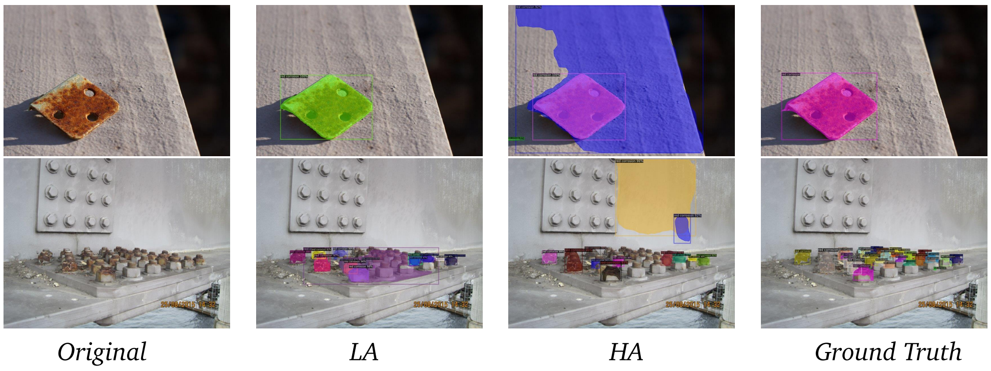


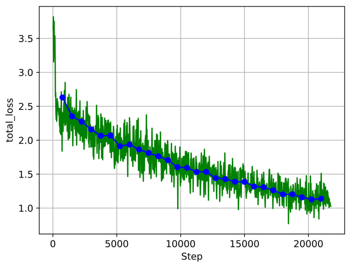
Figure 5.4: Comparing predictions on ResNet-101+DC5 with light augmentation(LA) and heavy augmentation(HA)

5.2 Genetic Algorithm

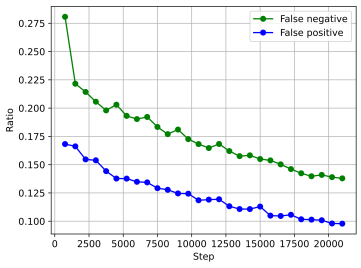
The final results can be viewed in table 5.3, with some visual comparisons between tuned and untuned network predictions in fig. 5.7. The results from ResNet-101+DC5 and ResNet-101+FPN are plotted in fig. 5.5 and fig. 5.6 respectively.

Backbone	**Corrosion IoU**	**Background IoU**	**Mean Iou**
ResNet-50 + FPN	60.1%	86.6%	73.3%
ResNet-101 + FPN	60.7%	85.7%	73.2%
ResNeXt-101 + FPN	62.6%	87.0%	74.8%
ResNet-50 + DC5	60.7%	86.7%	73.7%
ResNet-101 + DC5	62.7%	87.5%	75.1%

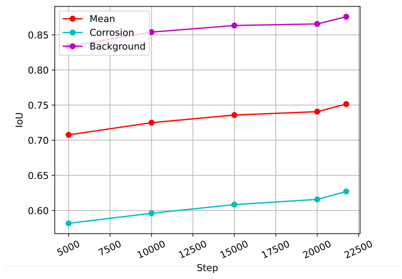
Table 5.3: Results from GA tuned run



(a) Total training loss

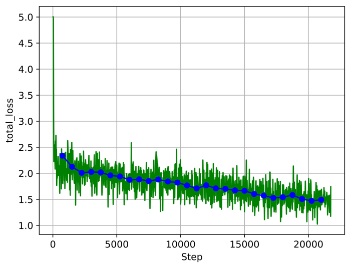


(b) False predictions

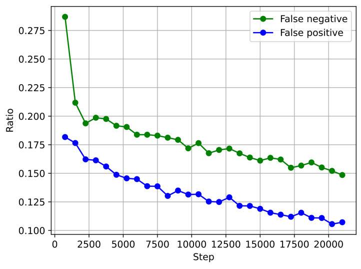


(c) IoU per iteration

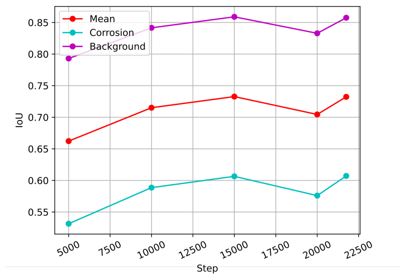
Figure 5.5: Plots from GA-tuned run, ResNet-101+DC5



(a) Total training loss



(b) False predictions



(c) IoU per iteration

Figure 5.6: Plots from GA-tuned run, ResNet-101+FPN

Discussion

The results from table 5.3 show a mean IoU between 73.2% and 75.1%, with the only general improvement from light augmentation in table 5.1 being in ResNeXt-101 + FPN, which had a 0.5% increase in mean IoU.

By only looking at the corrosion IoU, which defines the fitness score of the Genetic Algorithm, there has been an increase for both ResNeXt and ResNet-101+DC5, but the noticeable is that background IoU has stagnated or decreased, resulting in no growth in mean IoU. Utilizing GA tuning did however not have a substantial effect, as the corrosion IoU for ResNet-101+DC5 only increases a mere 0.3%. Overall the average performance of the model decreases.

The difference is nore distinct when overlooking the plots of fig. 5.5 and comparing them to the plots from light augmentation in fig. 5.1. Although the predictions starts of worse with a greater loss and lower IoU, the training curve is more coherent and shows progress, in contrast to the basic run in fig. 5.1 which does not improve IoU that much. This could be a sign of the model escaping local minima, but the plots also suggest that the number of iterations should be higher, since the loss function does not converge as of yet, and does not reach the same minima as the light augmentation training loss in fig. 5.1.

The plots for ResNet-101+FPN in fig. 5.6 still show suboptimal performance, when compared to the DC5 equivalent. One is the loss function converging early, which is a problem also seen earlier in fig. 3.3, but at a worse point than before. The second point is that there is a consistent increase in IoU until the second half of the iteration, when it starts fluctuating again, and the degradation is more severe. The plots show a lot of the same problems as previously discussed, i.e. the model tries to adapt one type of damage, which leads it to fail on other damages.

Taking the example predictions from fig. 5.7, the updated model demonstrates hypersensitivity to various types of red irregularities. It generates multiple small and overlapping masks with confidence levels ranging from 75% to 100%. This outcome indicates that the new approach successfully identifies corrosion instances that were previously overlooked by the traditional model. Furthermore, the generation of smaller masks suggests improved segmentation with enhanced detail. However, a drawback of the new approach is its tendency to incorrectly detect background regions. This can result in overlaps and misclassifications in the predicted masks. To address this issue, one potential solution is to adjust the confidence threshold. By carefully tuning the threshold, it is possible to reduce the number of false positive detections and mitigate the impact of the reduced IOU threshold.

Since the model responds better to red corrosion, tuning mitigates some of the issues with falsely classifying corrosion-like patterns. This issue is particularly evident in the FPN models, which were initially challenged by non-red misclassifications. Conversely, the DC5 models had already addressed this problem and exhibited improved performance in that regard.

The suboptimal output of the algorithm, particularly in the case of the fitness score being highest at generation 7 instead of the intended generation 15, presents uncertainties regarding the factors contributing to this outcome. One possible reason could be the limited number of individuals and generations in the Genetic Algorithm, which restricts the search space and may result in exploring only a few combinations of hyperparameters. Additionally, the algorithm's constraint to run on ResNet-50+FPN and a dataset of only 200 images could lead to overfitting.

To address these issues, several potential solutions can be considered. Firstly, introducing the backbone as an additional gene in the Genetic Algorithm can enhance the exploration of different network architectures and further optimize the model's performance. This allows for the selection of an optimal backbone architecture based on the given task and data.

Furthermore, creating additional datasets and randomly selecting one set for training can help reduce overfitting and improve the model's generalization capabilities. By augmenting the available data with diverse samples, the algorithm can learn from a wider range of variations and better capture the underlying patterns.

Another reason behind GA failing is that there are hyperparameters not available in the Detectron2 config file, such as the loss weights. The pre-project emphasized the importance of these parameters in section 2.3[7]. On an additional note Redmon et al. configured their loss weights for YOLO to best suit its purpose[31]. Including the loss weights could therefore be useful when further expanding the search space for the genetic algorithm. Running with GA tuning, though not as successful as expected, still highlighted the importance of hyperparameters. The predictions in fig. 5.7 indicate that they can affect the model in different ways, but it is obvious that these are not the best-fitted hyperparameters, contrary to what the genetic algorithm outputs. Overall, the new approach shows potential for addressing specific challenges in the original model, but the Genetic Algorithm has produced far from the optimal values for this specific model. Further exploration and refinement of the method could lead to more accurate and robust corrosion detection and segmentation.

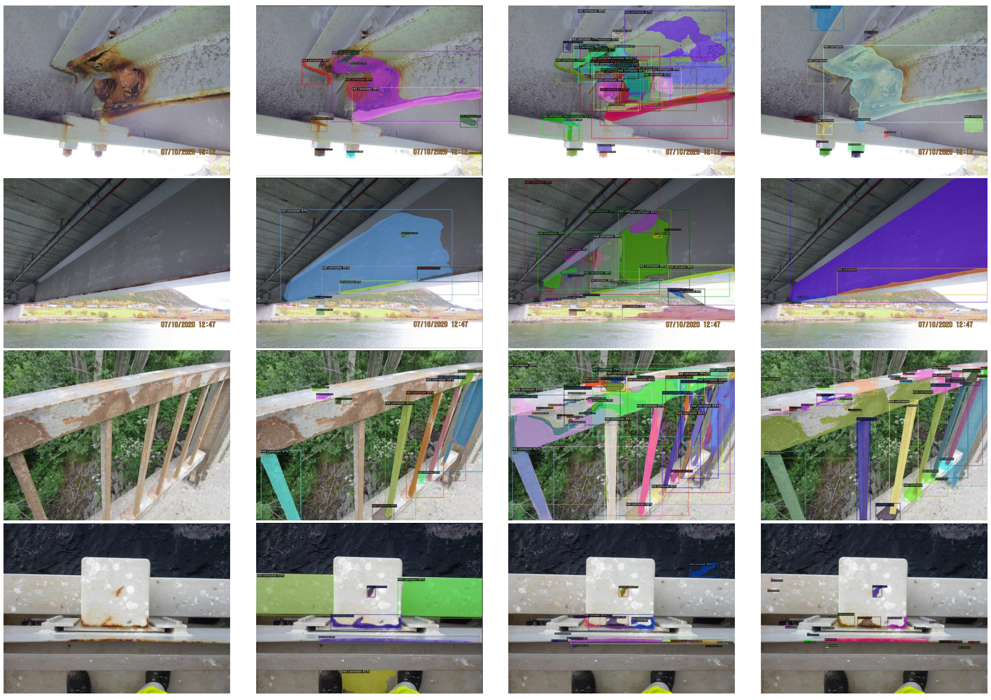


Figure 5.7: Comparing predictions with left-side vs right sided values in table 4.6, with ResNet-101+FPN

5.3 SkySeg

The weights from the light augmentation run in table 5.1 remains unchanged, but the validation dataset pass through SkySeg before the corrosion detection. The results from the new predictions are shown in table 5.4, with output predictions in fig. 5.8, fig. 5.9, fig. 5.10 and fig. 5.11.

Backbone	Corrosion IoU	Background IoU	Mean Iou
ResNet-50 + FPN	61.6%	86.7%	74.1%
ResNet-101 + FPN	61.0%	86.2%	73.5%
ResNeXt-101 + FPN	62.1%	86.9%	74.5%
ResNet-50 + DC5	60.7%	87.0%	73.8%
ResNet-101 + DC5	62.3%	87.8%	75.1%

Table 5.4: Current results on the validation dataset, with SkySeg

Discussion

The results show little to no improvement when comparing table 5.1 and table 5.4. The only way to properly assess SkySeg is to analyze the output pictures, which show several reasons behind the failure of SkySeg to demonstrate any improvement.

The most prominent aspect is SkySeg corrupting the images by removing parts of the bridge, as seen in fig. 5.8 and fig. 5.9. This misclassification leads to two distinct issues, one is that the bridge part removed actually contains labelled corrosion, which the corrosion model is not able to predict because SkySeg has removed them, fig. 5.8 illustrates this challenge. The other is that the corrosion model falsely predicts the removed pixels as corrosion, as seen in fig. 5.9. This degradation in performance despite good results from training is unexpected, as sky segmentation seems at first an easy task for the YOLO model to understand. The challenges stem most likely from limited research on the YOLO model, leading to similar challenges with the dataset and hyperparameters as with the corrosion model.

The corrosion dataset differentiates heavily from the ADE20K dataset. As the training dataset is quite monotonous, SkySeg overfits to images similar to fig. 4.11, but struggles with bridge images where the parts of the sky are occluded. Adding 150 images with bridges does not do enough to mitigate this issue, suggesting that the model requires a larger custom dataset of bridge images.

The model struggles too with images where no sky is present, which is often what leads to false predictions. The model should therefore only perform on images which contain sky in order to avoid misclassifications.

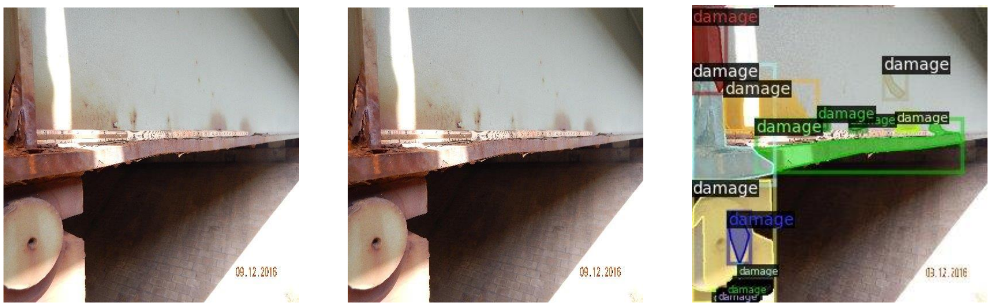
Upon analyzing the images where SkySeg demonstrates successful performance, specifically those with a clear horizon and a distinct border between the sky and the bridge, it is observed that there are only a few instances where the pre-processing has noticeably improved corrosion detection. An example of such a notable difference can be seen in the prediction shown in fig. 5.10. However, in most instances where the SkySeg model performs adequately, the difference in predictions is minimal. This challenges the hypothesis put forth in the pre-project, which suggested that the corrosion detection model made mistakes in detection by misinterpreting the background as the structure and clouds as corrosion. The results from fig. 5.11 indicate the contrary. It is possible that the exclusion of white corrosion images during pre-processing and introducing DC5 models, which exhibit greater robustness to the background, have also enhanced the corrosion model's ability to handle clouds, which relegates SkySeg's role.

In the current state of the images, SkySeg appears to be redundant since the corrosion model can effectively distinguish between the structure and the background in such instances. The poor performance of the sky network complicates the evaluation of its impact on the corrosion model, as it either confuses the network or fails to introduce significant changes to the images. As a result, it becomes challenging to determine how SkySeg truly affects the corrosion model's performance, as it either hampers the network's performance or fails to make a substantial difference.

Alternative approaches

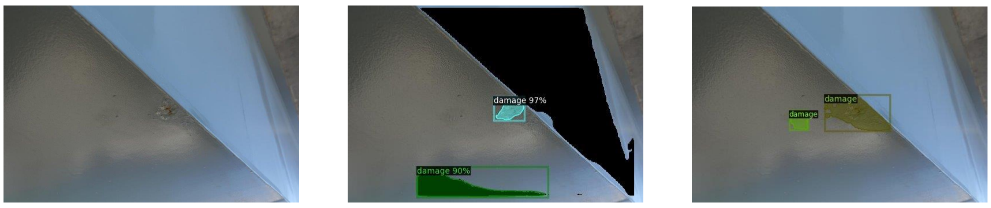
Considering the effort required to implement and fine-tune a sky segmentation network solely for background removal purposes, it may be worthwhile to explore alternative approaches that offer greater efficiency. One possible solution could be expanding the dataset to include more diverse backgrounds. This would provide the network with a broader range of background variations to learn from. Additionally, the DC5 models exhibit greater robustness to false positives, as demonstrated in fig. 5.2, suggesting that they may be a more effective solution for handling background-related challenges.

Another large-scale alternative to removing sky pixels is to introduce depth as an extra dimension to the images, so-called RGB-D (Red Green Blue - Depth) images. As sky and background are an unavoidable part of bridge images and the model as of now has no account of how far away the structure is, it is currently impossible for the model itself to understand the difference between noisy background and actual corrosion. Introducing depth per pixel, where the sky and background would be infinitely away, could lead to the model being more aware of where the structure is and in consequence, where the corrosion is. Depth-aware CNNs have been in development[55, 56], but they suffer from high computational complexity and a lack of datasets with RGB-D images. With the occurrence of higher quality drones, RGB-D images are possible to render either directly from the drone or with depth estimation tools, so this could be a future direction of improvement.



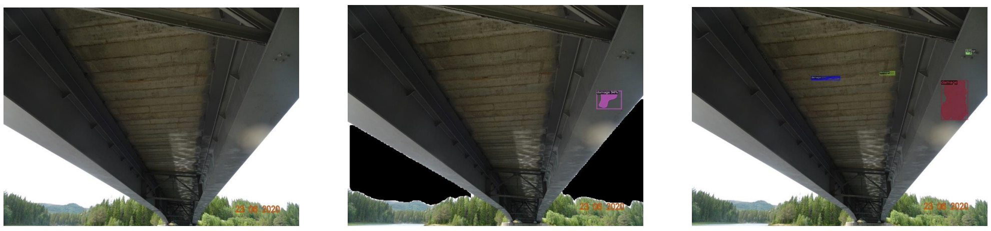
(a) Original image **(b)** Prediction (SkySeg) **(c)** Ground Truth

Figure 5.8: Example of poor pre-processing with SkySeg, the masking removes part of the bridge where corrosion is present



(a) Original image **(b)** Prediction (SkySeg) **(c)** Ground Truth

Figure 5.9: Example where the mask itself is detected as corrosion



(a) Prediction(normal) **(b)** Prediction(SkySeg) **(c)** Ground Truth

Figure 5.10: Comparing predictions with and without SkySeg to the Ground Truth, the prediction with pre-processing performs slightly better

5.4 Summary of results and analysis

Table 5.5 summarizes the mean IoU from all runs discussed in the preceding sections. Comparing the results shows some prominent factors. In general, it is the reduction of the dataset that induces the largest increase in mean IoU, as well as introducing DC5 models, where ResNet-101 with DC5 performs the best across all strategies with an average mIoU of 74.5%. ResNext comes in second place with 74.1%. These are the two models suggested to be used for further work. ResNet-101+FPN performs the worst of all models, which is surprising considering its likeness to ResNet-50+FPN and ResNet-101+DC5.

The differences are small across backbone models and strategies, ranging between 72.0% as the worst value and 75.1% as the best. The stagnant values show the clear diversity in the dataset, where models with a certain strategy work better on some images, such as the GA tuning on hard-to-spot red corrosion, but falter on other types. Even with the reduction in the dataset, there are limitations which reduce the model's full potential. This is evident by the lack of performance over the iterations when evaluating fig. 5.1. There is no "one strategy fixes all" solution, but expanding and focusing on the dataset

Backbone	Feature extractor	Strategy			
		LA	HA	GA-tuned	SkySeg
ResNet-50	FPN	74.0%	72.0%	73.3%	74.1%
	DC5	73.9%	72.1%	73.7%	73.8%
ResNet-101	FPN	73.7%	**72.6%**	73.2%	73.5%
	DC5	**75.1%**	72.5%	**75.1%**	**75.1%**
ResNeXt-101	FPN	74.4%	72.5%	74.8%	74.5%

Table 5.5: Mean IoU comparison for all experiments

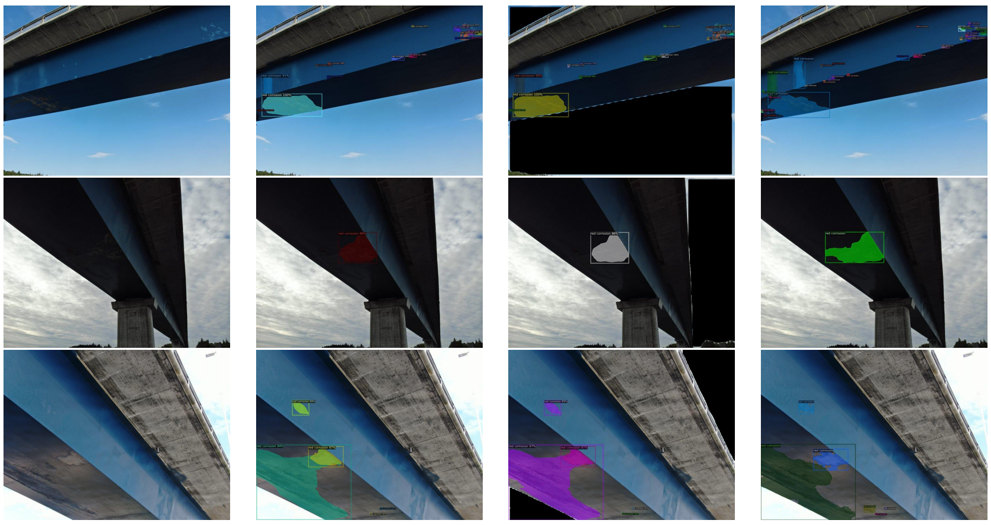


Figure 5.11: Comparing predictions from ResNet-101+DC5 with and without SkySeg to the Ground Truth. the predictions are completely alike.

might strengthen the model going forward.

Using the hyperparameters from GA tuning had little effect on the mean IoU, but showed some increase in the corrosion IoU for ResNet-101+DC5 and ResNeXt, which shows some progress by changing hyperparameters. Hyperparameters still play a big role despite the underperformance of the current GA. The grid search on the confidence threshold, as shown in fig. 5.12, indicates that lowering the threshold below GA's suggested 0.76 improves the model's mean IoU. This suggests that a lower threshold value is preferable for better performance.

Running the final evaluation from the lessons in this discussion and taking the results from fig. 5.12, the threshold is set to 0.65 on ResNet-101+DC5 with light augmentation and default configuration. The final result is shown in table 5.6. With a mean IoU of 75.7% and corrosion IoU of 63.3%, this evaluation is the best result achieved in this thesis and shows that some small tweaking can still improve the network.

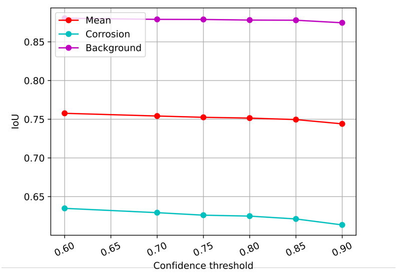


Figure 5.12: IoU per confidence threshold for ResNet-101+DC5

A 63% corrosion IoU is still far from ideal, and there are many challenges that are yet to be solved. Even though the effect of lowering the threshold on average increases mean IoU, it still poses problems with false positive predictions of gravel and red background

Backbone	Corrosion IoU	Background IoU	Mean Iou
ResNet-101 + DC5	63.3%	88.0%	75.7%

Table 5.6: Results from one final run with ResNet-101+DC5, with a lowered threshold

such as in fig. 3.6. This refers again to the diversity in the dataset.

Other challenges where the mIoU is still low can be viewed in fig. 5.13. These examples show images where the corrosion is obvious, but the model still fails to detect it. As of now, there is no clear-cut answer to why this happens or how to fix it, but assessing the model with a larger or more focused dataset may offer up good points for analysis.

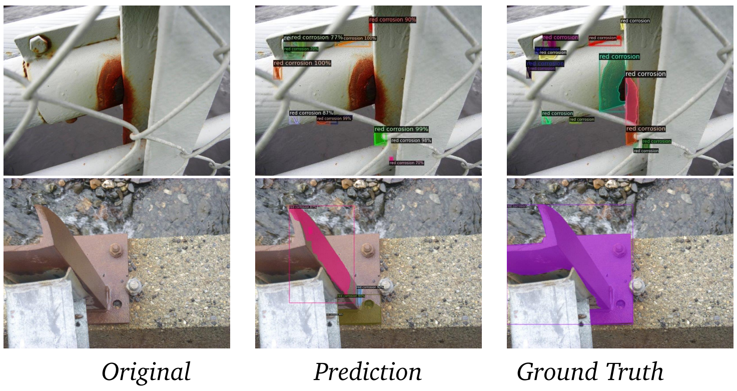


Figure 5.13: Final predictions on validation dataset, with ResNet-101+DC5

Chapter 6

Conclusion and further work

6.1 Conclusion

The goal of this master thesis has been to probe and assess new methods for improving the accuracy of corrosion detection with Mask R-CNN. The work contains three main contributions. Each contribution is concluded below.

6.1.1 Dataset

The work of this thesis started with a dataset consisting of 1990 images with varying degrees of patterns, backgrounds, angles and types of damage. After removing a batch of images because of corruption and then again another batch where the focal damage was not red corrosion, the final number of images was 1772. The reduction in images led to the most obvious improvement in accuracy, which speaks to the importance of good and diverse data when training the network. The quality of the dataset is probably the main contributor to why the accuracy does not improve in a significant way over the course of the thesis. The model's difficulty in generalizing to basic corrosion patterns can be attributed to the significant variety of corrosion damages present in the images. Factors such as large distances to the damage, noisy backgrounds, and disturbances like shadows contribute to the challenges faced by the model in accurately detecting these patterns.

To address these issues, it is necessary to expand the dataset by including images of other damage types, excluding irrelevant objects, and introducing worst-case scenarios for better generalization. By addressing these concerns, the model's performance in detecting corrosion can be significantly improved.

Data augmentation was used as a temporary solution to mitigate these issues, and it was shown that conservative methods of flipping horizontally and vertically had a good effect on accuracy. Trying however to expand the methods of data augmentation with colour transformations such as contrast, brightness and saturation seemed to decrease the IoU and resulted in a higher ratio of false positives, which shows that the search space needs to be opened wider for further experimentations if data augmentations should be further uses.

In conclusion, the goal of the model should guide the selection of images for training. If the model should only detect red corrosion on bridges, it is crucial to curate a dataset that primarily consists of images depicting this specific type of damage. If the goal is to detect several types of damage on other infrastructures, the dataset has to be expanded greatly in order to accommodate those high expectations.

6.1.2 Genetic Algorithm for automatic hyperparameter tuning

This thesis has implemented and adapted the Genetic Algorithm to find the optimal combination of hyperparameters to get the highest accuracy possible, with both simple mutation, which was shown to create divergence in the algorithm, and adaptive mutation which had the most promising results.

The results show only minor performance gain using GA-tuned hyperparameters, but the limitations could also be due to the quality of the dataset as mentioned before, limited access to hyperparameters by Detectron2, too small population and overfitting on the small subset of training images. Overall the Genetic Algorithm shows an effective way of tuning hyperparameters without the need for human interaction, which could be expanded on in further work by introducing more hyperparameters. Indeed, while hyperparameters play a crucial role in optimizing neural networks, it is important to recognize that the model's performance will ultimately be constrained by the quality and diversity of the dataset it is trained on. Therefore, before delving into the development of the Genetic Algorithm for hyperparameter optimization, it is advisable to focus on expanding the dataset.

6.1.3 SkySeg

An independent network called SkySeg, utilizing the state-of-the-art YOLO model, was implemented to detect the sky in the corrosion dataset. The objective was to assess whether removing the background, specifically the sky, would have any impact on the accuracy of the corrosion detection model. The newly constructed dataset was created by incorporating images from the ADE20K dataset.

The results of this experiment did not demonstrate any significant improvement in accuracy. In many images where the sky was not present, YOLO instead falsely detected parts of the bridge and removed crucial segments, leading to a decrease in IoU. In cases in which SkySeg successfully removed the sky, the predicted masks from the corrosion detection model were identical to the masks without background removal, rendering the sky masking process redundant. The results from earlier implementations such as incorporating dilated convolutions and removing images with white corrosion had a greater effect on reducing false positives of clouds.

Overall, the implementation of a background removal network did not contribute to enhancing the accuracy of the corrosion detection model. SkySeg has therefore been concluded to not improve the model in such a way that it is worth the cost of implementing a new network with its own dataset. The easier and more robust solution would be to expand the dataset with more background variety, including the sky. Another solution that has been mentioned for further work is introducing RGB-D images where depth is a fourth dimension, allowing the model to generalize based on the distance to the structure and learn efficiently what parts are background and can be ignored.

6.2 Further work

This thesis has provided insights into the field of image segmentation, particularly in the context of corrosion detection. However, there are several avenues for further exploration and improvement. The following suggestions outline potential directions for future work in this area.

6.2.1 Introduce a combination of FPN and dilated convolutions

All though most projects use FPN as a feature extractor with Mask R-CNN, the results suggest that the corrosion model prefers DC5. As the methods are not mutually exclusive, a potential enhancement is to incorporate the Dilated Convolutional Feature Pyramid Network (DCFPN) into the Mask R-CNN architecture. By combining the strengths of Feature Pyramid Network (FPN) and DC5, the model can potentially achieve improved performance in terms of both accuracy and computational efficiency.

6.2.2 Multiple classes

Multi-class segmentation is a topic mentioned both by Fondevik[6] and in the pre-project[7], where several damage types are added as their own separate class, but has not been implemented due to the lack of data. In this thesis, the focus has primarily been on detecting corrosion as a single class. However, further work can involve extending the model to detect and classify multiple types of damage. The previous results of the network indicate that the damage types differ from each other too much for the model to be able to detect all of them under the same class. By implementing multi-class segmentation, the model would be more effective in generalizing each damage type. To accomplish this, the model requires additional data specific to these damage types. Synthetic data generation techniques, such as GAN, SMOTE, or class weighting, can be explored to augment the dataset and improve the model's ability to detect and classify different damage types.

6.2.3 Change of network

While Mask R-CNN has demonstrated promising performance in this thesis, it is worthwhile to explore alternative instance segmentation models that have shown superior results in other instance segmentation tasks. The COCO test competition is a place to start finding and evaluating new instance segmentation models, and every year there comes a new model or implementation that outperforms the current model.

6.2.4 RGB-D Images and Depth-Aware CNNs

To enhance the model's capability for detecting corrosion, the dataset can be extended to include RGB-D images. By incorporating depth information alongside RGB data, the model can potentially leverage the additional depth cues to improve segmentation accuracy and robustness.

Bibliography

[1] 'Norwegian public roads administration,' [Online]. Available: https://www.vegvesen.no/en/?lang=en (visited on 14/03/2023).

[2] D. J. Atha and M. R. Jahanshahi, 'Evaluation of deep learning approaches based on convolutional neural networks for corrosion detection,' *Structural Health Monitoring*, vol. 17, no. 5, pp. 1110–1128, 2018. DOI: 10.1177/1475921717737051. eprint: https://doi.org/10.1177/1475921717737051. [Online]. Available: https://doi.org/10.1177/1475921717737051.

[3] R. Howells. 'Keeping norwegian bridges safe through intelligent asset monitoring and industry 4.0.' (2020), [Online]. Available: https://www.forbes.com/sites/sap/2020/06/18/keeping-norwegian-bridges-safe-through-intelligent-asset-monitoring-and-industry-40/?sh=e6b5c146b792 (visited on 18/05/2023).

[4] G. Schmitt, M. Schütze, G. Hays, W. Burns, E.-H. Han, A. Pourbaix and G. Jacobson, 'Global needs for knowledge dissemination, research, and development in materials deterioration and corrosion control,' *World Corrosion Organization*, p. 14, Jan. 2009.

[5] K. He, G. Gkioxari, P. Dollár and R. Girshick, 'Mask r-cnn,' 2017. DOI: 10.48550/ARXIV.1703.06870. [Online]. Available: https://arxiv.org/abs/1703.06870.

[6] S. K. Fondevik, *Image segmentation of corrosion damages in industrial inspections using state-of-the-art neural networks*, 2020. [Online]. Available: https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2780881?locale-attribute=no.

[7] H. Semb, *Challenges of image segmentation of corrosion damages using mask r-cnn*, Specialization thesis, Dec. 2022.

[8] R. Lemos, R. Cabral, D. Ribeiro, R. Santos, V. Alves and A. Dias, 'Automatic detection of corrosion in large-scale industrial buildings based on artificial intelligence and unmanned aerial vehicles,' *Applied Sciences*, vol. 13, no. 3, p. 1386, Jan. 2023, ISSN: 2076-3417. DOI: 10.3390/app13031386. [Online]. Available: http://dx.doi.org/10.3390/app13031386.

[9] Y. Bai, B. Zha, H. Sezen and A. Yilmaz, 'Engineering deep learning methods on automatic detection of damage in infrastructure due to extreme events,' *Structural Health Monitoring*, vol. 22, no. 1, pp. 338–352, 2023. DOI: 10.1177/14759217221083649. eprint: https://doi.org/10.1177/14759217221083649. [Online]. Available: https://doi.org/10.1177/14759217221083649.

[10] L. F. Rodrigues, A. R. Backes, B. A. N. Travençolo and G. M. B. de Oliveira, 'Optimizing a deep residual neural network with genetic algorithm for acute lymphoblastic leukemia classification,' *Journal of Digital Imaging*, vol. 35, no. 3, pp. 623–637, 2022.

[11] S. Lee, J. Kim, H. Kang, D.-Y. Kang and J. Park, 'Genetic algorithm based deep learning neural network structure and hyperparameter optimization,' *Applied Sciences*, vol. 11, no. 2, p. 744, Jan. 2021, ISSN: 2076-3417. DOI: 10.3390/app11020744. [Online]. Available: http://dx.doi.org/10.3390/app11020744.

[12] 'Orbitron.' (), [Online]. Available: https://orbiton.no/ (visited on 05/06/2023).

[13] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo and R. Girshick, *Detectron2*, https://github.com/facebookresearch/detectron2, 2019.

[14] A. Bhayani, *Genetic algorithm to solve the knapsack problem*, https://github.com/arpitbbhayani/genetic-knapsack, 2022.

[15] L. Yang and A. Shami, 'On hyperparameter optimization of machine learning algorithms: Theory and practice,' *Neurocomputing*, vol. 415, pp. 295–316, 2020, ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2020.07.061. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231220311693.

[16] M. Abdel-Basset, L. Abdel-Fatah and A. K. Sangaiah, 'Chapter 10 - metaheuristic algorithms: A comprehensive review,' in *Computational Intelligence for Multimedia Big Data on the Cloud with Engineering Applications*, ser. Intelligent Data-Centric Systems, A. K. Sangaiah, M. Sheng and Z. Zhang, Eds., Academic Press, 2018, pp. 185–231, ISBN: 978-0-12-813314-9. DOI: https://doi.org/10.1016/B978-0-12-813314-9.00010-4. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128133149000104.

[17] S. Ren, K. He, R. Girshick and J. Sun, 'Faster r-cnn: Towards real-time object detection with region proposal networks,' DOI: 10.48550/ARXIV.1506.01497. [Online]. Available: https://arxiv.org/abs/1506.01497.

[18] K. He, X. Zhang, S. Ren and J. Sun, 'Deep residual learning for image recognition,' *CoRR*, vol. abs/1512.03385, 2015. arXiv: 1512.03385. [Online]. Available: http://arxiv.org/abs/1512.03385.

[19] S. Xie, R. Girshick, P. Dollár, Z. Tu and K. He, *Aggregated residual transformations for deep neural networks*, 2017. arXiv: 1611.05431 [cs.CV].

[20] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, *Going deeper with convolutions*, 2014. arXiv: 1409.4842 [cs.CV].

[21] 'Instance segmentation on coco test-dev.' (Jan. 2023), [Online]. Available: https://paperswithcode.com/sota/instance-segmentation-on-coco (visited on 01/06/2023).

[22] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan and S. Belongie, *Feature pyramid networks for object detection*, 2017. arXiv: 1612.03144 [cs.CV].

[23] S. Altindis, Y. Dalva and A. Dundar, *Benchmarking the robustness of instance segmentation models*, Sep. 2021.

[24] F. Yu and V. Koltun, *Multi-scale context aggregation by dilated convolutions*, 2016. arXiv: 1511.07122 [cs.CV].

[25] D. A. van Dyk and X.-L. Meng, 'The art of data augmentation,' *Journal of Computational and Graphical Statistics*, vol. 10, no. 1, pp. 1–50, 2001. DOI: 10.1198/10618600152418584. eprint: https://doi.org/10.1198/10618600152418584. [Online]. Available: https://doi.org/10.1198/10618600152418584.

[26] W. Fang, Y. Ding, F. Zhang and V. S. Sheng, 'Dog: A new background removal for object recognition from images,' *Neurocomputing*, vol. 361, pp. 85–91, 2019, ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2019.05.095. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231219308963.

[27] S. B. Park, J. W. Lee and S. K. Kim, 'Content-based image classification using a neural network,' *Pattern Recognition Letters*, vol. 25, no. 3, pp. 287–300, 2004, ISSN: 0167-8655. DOI: https://doi.org/10.1016/j.patrec.2003.10.015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167865503002253.

[28] C. Sun, G. Arr, R. Ramachandran and S. Ritchie, 'Vehicle reidentification using multidetector fusion,' *IEEE Transactions on Intelligent Transportation Systems*, vol. 5, pp. 155–164, Sep. 2004. DOI: 10.1109/TITS.2004.833770.

[29] J. Nader, 'Backgroundremover,' *GitHub repository*, 2021.

[30] X. Qin, Z. Zhang, C. Huang, M. Dehghan, O. R. Zaiane and M. Jagersand, 'U2-net: Going deeper with nested u-structure for salient object detection,' *Pattern recognition*, vol. 106, p. 107 404, 2020.

[31] J. Redmon and A. Farhadi, 'Yolov3: An incremental improvement,' 2018. DOI: 10.48550/ARXIV.1804.02767. [Online]. Available: https://arxiv.org/abs/1804.02767.

[32] S. Sánchez Hernández, H. Romero and A. Morales, 'A review: Comparison of performance metrics of pretrained models for object detection using the tensorflow framework,' *IOP Conference Series: Materials Science and Engineering*, vol. 844, p. 012 024, Jun. 2020. DOI: 10.1088/1757-899X/844/1/012024.

[33] A. Lambora, K. Gupta and K. Chopra, 'Genetic algorithm- a literature review,' in *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*, 2019, pp. 380–384. DOI: 10.1109/COMITCon.2019.8862255.

[34] 'Darwin from v7 labs,' [Online]. Available: https://darwin.v7labs.com (visited on 02/05/2023).

[35] G. Josher, A. Chaurasia and J. Qiu, *Yolov8, developed by ultralytics*, https://github.com/ultralytics/ultralytics, 2023.

[36] W. Abdulla, 'Mask r-cnn for object detection and instance segmentation on keras and tensorflow,' *GitHub repository*, 2017.

[37] K. Chen, J. Wang, J. Pang, Y. Cao, Y. Xiong, X. Li, S. Sun, W. Feng, Z. Liu, J. Xu, Z. Zhang, D. Cheng, C. Zhu, T. Cheng, Q. Zhao, B. Li, X. Lu, R. Zhu, Y. Wu, J. Dai, J. Wang, J. Shi, W. Ouyang, C. C. Loy and D. Lin, 'MMDetection: Open mmlab detection toolbox and benchmark,' *arXiv preprint arXiv:1906.07155*, 2019.

[38] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai and S. Chintala, 'Pytorch: An imperative style, high-performance deep learning library,' in *Advances in Neural Information Processing Systems 32*, Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[39] *Detectron2 documentation*. [Online]. Available: `https://detectron2.readthedocs.io/` (visited on 12/04/2023).

[40] G. Bradski, 'The OpenCV Library,' *Dr. Dobb's Journal of Software Tools*, 2000.

[41] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick and P. Dollár, 'Microsoft coco: Common objects in context,' 2014. DOI: `10.48550/ARXIV.1405.0312`. [Online]. Available: `https://arxiv.org/abs/1405.0312`.

[42] D. A. van Dyk and X.-L. Meng, 'The art of data augmentation,' *Journal of Computational and Graphical Statistics*, vol. 10, no. 1, pp. 1–50, 2001. DOI: `10.1198/10618600152418584`. eprint: `https://doi.org/10.1198/10618600152418584`. [Online]. Available: `https://doi.org/10.1198/10618600152418584`.

[43] C. Shorten and T. M. Khoshgoftaar, 'A survey on image data augmentation for deep learning,' *Journal of big data*, vol. 6, no. 1, pp. 1–48, 2019.

[44] M. Gerber, N. Pillay, K. Holan, S. A. Whitham and D. K. Berger, 'Automated hyper-parameter tuning of a mask r-cnn for quantifying common rust severity in maize,' pp. 1–7, 2021. DOI: `10.1109/IJCNN52387.2021.9534417`.

[45] B. Zhou, H. Zhao, X. Puig, T. Xiao, S. Fidler, A. Barriuso and A. Torralba, *Semantic understanding of scenes through the ade20k dataset*, 2016. DOI: `10.48550/ARXIV.1608.05442`. [Online]. Available: `https://arxiv.org/abs/1608.05442`.

[46] H. Zhao, Z. Yu and B. Shou, *Places challenge*, `https://github.com/CSAILVision/placeschallenge`, 2017.

[47] C. Evans. 'Yaml ain't markup language,' YAML.org. (2001), [Online]. Available: `http://yaml.org`.

[48] J. Nelson, B. Dwyer and J. Solawetz, *Roboflow*, version 1.0. [Online]. Available: `https://roboflow.com/`.

[49] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, P. Dollár and R. Girshick, 'Segment anything,' *arXiv:2304.02643*, 2023.

[50] M. Själander, M. Jahre, G. Tufte and N. Reissmann, *EPIC: An energy-efficient, high-performance GPGPU computing research infrastructure*, 2019. arXiv: `1912.05848 [cs.DC]`.

[51] B. Guan, J. Yao, G. Zhang and X. Wang, 'Thigh fracture detection using deep learning method based on new dilated convolutional feature pyramid network,' *Pattern Recognition Letters*, vol. 125, pp. 521–526, 2019, ISSN: 0167-8655. DOI: `https://doi.org/10.1016/j.patrec.2019.06.015`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0167865519301825`.

[52] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville and Y. Bengio, *Generative adversarial networks*, 2014. arXiv: `1406.2661 [stat.ML]`.

[53] N. V. Chawla, K. W. Bowyer, L. O. Hall and W. P. Kegelmeyer, 'SMOTE: Synthetic minority over-sampling technique,' *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, Jun. 2002. DOI: `10.1613/jair.953`. [Online]. Available: `https://doi.org/10.1613%2Fjair.953`.

[54] Z. Xu, C. Dan, J. Khim and P. Ravikumar, *Class-weighted classification: Trade-offs and robust approaches*, 2020. arXiv: `2005.12914 [stat.ML]`.

[55] W. Wang and U. Neumann, 'Depth-aware cnn for rgb-d segmentation,' in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 135–150.

[56] S. Tu, J. Pang, H. Liu, N. Zhuang, Y. Chen, C. Zheng, H. Wan and Y. Xue, 'Passion fruit detection and counting based on multiple scale faster r-cnn using rgb-d images,' *Precision Agriculture*, vol. 21, pp. 1072–1091, 2020.

Appendix A

Dataset conversion

A.1 Binary masks to COCO format

```
def find_contours(sub_mask):
    """Generates a tuple of points where a contour was found from a binary mask

    Args:
        sub_mask (numpy array): binary mask
    """
    assert sub_mask is not None, "file␣could␣not␣be␣read,␣check␣with␣os.path.exists()"
    imgray = cv2.cvtColor(sub_mask, cv2.COLOR_BGR2GRAY)
    ret, thresh = cv2.threshold(imgray, 127, 255, 0)
    contours, hierarchy = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
    assert len(contours)!= 0, print(contours)
    return contours


def create_image_annotation(file_name, width, height, image_id):
    return {
        "image_id": image_id,
        "width": width,
        "height": height,
        "file_name": file_name,
    }


def create_annotation_format(contour):
    return {
        "iscrowd": 0,
        "segmentation": [contour.flatten().tolist()],
        "bbox": cv2.boundingRect(contour),
        "bbox_mode": BoxMode.XYWH_ABS,
        "category_id": 0,
    }



def load_damage_dicts(dataset_dir, subset):
    """
    Loads the images from a dataset with a dictionary of the annotations in COCO-format
    """
    dataset_dicts = []

    assert subset in ["train", "val"]
    dataset_dir = os.path.join(dataset_dir, subset)
    image_ids = next(os.walk(dataset_dir))[1]
    for image_id in image_ids:

        image_dir = os.path.join(dataset_dir, image_id)
        (_, _, file_names) = next(os.walk(image_dir))
        file_name = file_names[0]

        image_path = os.path.join(image_dir, file_name)
        height, width = cv2.imread(image_path).shape[:2]
        record = create_image_annotation(image_path, width, height, image_id)
        #idx +=1
        mask_dir = os.path.join(image_dir, 'masks')
        objs = []
        for f in next(os.walk(mask_dir))[2]:
            if f.endswith('.png') and ('corrosion' or 'grov_merking' in f):
                mask_path = os.path.join(mask_dir, f)
                #print(mask_path)
                mask = cv2.imread(mask_path)
                if mask is None:
                    print("Couldn't␣retrieve␣mask:␣", mask_path)
                    continue
                if mask.shape[0]!=height:
                    print("MISMATCH:", image_dir)
                if not(255 in mask):
                    continue
                contours = find_contours(mask)
                for contour in contours:
                    if len(contour) < 3:
                        continue
                    obj = create_annotation_format(contour)
                    objs.append(obj)
        record["annotations"] = objs
        dataset_dicts.append(record)
    return dataset_dicts
```

A.2 ADE20K to YOLO format

```
def load_sky_yolo(root, subset,destination):
    """
    Transforms the ADE20K dataset to a binary mask
    """
    assert subset in ['train', 'val']
    if subset == 'train':
        dir = 'training'
    else:
        dir = 'validation'

    source = os.path.join(root, "images", dir)
    mask_dir = os.path.join(root, "annotations", dir)
    print(mask_dir)
    mask_ids = next(os.walk(mask_dir))[2]

    for id in mask_ids:
        mask_path = os.path.join(mask_dir, id)
        mask = cv2.imread(mask_path, cv2.IMREAD_GRAYSCALE)
        height, width = mask.shape
        #print(mask)
        #print(mask.dtype)
        string = ""
        for label in [3]:
            mask = cv2.imread(mask_path, cv2.IMREAD_GRAYSCALE)
            height, width = mask.shape
            mask = np.where(mask==label, 255,0) #3 for sky
            if 255 not in mask:
                continue
            mask = mask.astype(’uint8’)
        #print(mask)

            contours = find_contours(mask)
            for contour in contours:
                contour_list = contour.flatten().tolist()
                if len(contour_list) < 5:
                    continue
                if label == 3:
                    string += "0␣"
                for i in range(1,len(contour_list),2):
                    string += str(round(contour_list[i-1]/width,6)) #x coordinate
                    string += "␣"
                    string += str(round(contour_list[i]/height, 6)) # y coordinate
                    string += "␣"
                string+= "\n"
        if string == "":
            continue
        image_id = os.path.splitext(id)[0] + ’.jpg’
        image_source = os.path.join(source, image_id)
        image_dest = os.path.join(destination, "images", subset, image_id)
        print("destination:␣", image_dest)
        print("source:␣", image_source)
        shutil.copy(image_source, image_dest)
        print(string)
        txt_id = os.path.splitext(id)[0]+’.txt’
        txt_path = os.path.join(destination, "labels", subset, txt_id)
        print(txt_path)
        with open(txt_path, "w") as f:
            f.write(string)
```

Appendix B

Mask R-CNN

B.1 Data augmentation

```
class CustomTrainer(DefaultTrainer):
    @classmethod
    def build_train_loader(cls, cfg):
        mapper = DatasetMapper(cfg, is_train=True,
                               augmentations=
                               [T.RandomFlip(prob=0.5, horizontal=True, vertical=False),
                                T.RandomFlip(prob=0.5, horizontal=False, vertical=True),
                                T.RandomBrightness(0.8, 1.8),
                                T.RandomSaturation(0.8, 1.4),
                                T.RandomContrast(0.6, 1.3),
                                ])
        return build_detection_train_loader(cfg, mapper=mapper)
```

B.2 Configuration

B.3 Main

```
def train_model(cfg, backbone):
    cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url(backbone)
    trainer = DefaultTrainer(cfg)
    trainer.resume_or_load(resume=False)
    trainer.train()


def predict(cfg, damage_metadata, segment_sky = False):
    cfg.MODEL.WEIGHTS = os.path.join(cfg.OUTPUT_DIR, "model_final.pth")
    predictor = DefaultPredictor(cfg)
    val_dict = DatasetCatalog.get("damage_val")
    apply_inference(predictor, damage_metadata, output_dir, d, segment_sky)


def evaluate(cfg, segment_sky = False):
    val_dict = DatasetCatalog.get("damage_val")
    cfg.MODEL.WEIGHTS = os.path.join(cfg.OUTPUT_DIR, "model_final.pth")
    evaluate_model(cfg, val_dict, True, segment_sky)
```

```
def inference(cfg):
    val_dict = DatasetCatalog.get("damage_val")
    cfg.MODEL.WEIGHTS = os.path.join(cfg.OUTPUT_DIR, "model_final.pth")
    evaluate_over_iterations(cfg, val_dict, cfg.OUTPUT_DIR, plot=True, segment_sky=False)


if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="Custom␣Trainer␣Script")
    parser.add_argument("--backbone", type=str, help="Backbone␣model␣to␣use")
    parser.add_argument("--output_dir", type=str, help="Output␣directory")
    parser.add_argument("--data_dir", type=str, help="Dataset␣directory")
    parser.add_argument("--mode", type=str, choices=["train", "predict", "evaluate", "inference"],
                        help="Execution␣mode")
    parser.add_argument("--segment_sky", action="store_true", help="Segment␣sky")

    args = parser.parse_args()

    mode = args.mode
    backbone_model = args.backbone
    output_dir = args.output_dir
    segment_sky = args.segment_sky
    data_dir = args.data_dir
    for d in ["train", "val"]:
        DatasetCatalog.register("damage_" + d, lambda d=d: load_damage_dicts(data_dir,d))
        MetadataCatalog.get("damage_" + d).set(thing_classes=["red␣corrosion"])

    damage_metadata = MetadataCatalog.get("damage_train")

    cfg = config(backbone_model, output_dir)
    os.makedirs(cfg.OUTPUT_DIR, exist_ok=True)

    if mode == "train":
        train_model(cfg, backbone_model)
    elif mode == "predict":
        predict(cfg, damage_metadata, segment_sky)
    elif mode == "evaluate":
        evaluate(cfg, segment_sky)
    elif mode == "inference":
        inference(cfg)
    else:
        print("Invalid␣mode␣chosen")
```

Appendix C

Genetic Algorithm

C.1 Adaptive mutation

```python
def adaptive_mutation(hyperparameters, init_values, generation):
    mutated_hyperparameters = hyperparameters.copy()
    for param, value in mutated_hyperparameters.items():
        # Calculate the mutation range based on the current generation
        mutation_range = 1.0 / (generation + 1)
        if param == "roi_batch_size" or param == "rpn_batch_size":
            mutated_value = random.choice(init_values[param])
        # Mutate the parameter by a random value within the mutation range
        else:
            mutated_value = value + random.uniform(-mutation_range, mutation_range)
            # Clip the mutated value within the parameter's defined range
            mutated_value = np.clip(mutated_value, init_values[param].min(),
                                    init_values[param].max())
            # Update the mutated parameter value
            mutated_hyperparameters[param] = mutated_value
    return mutated_hyperparameters
```

C.2 Genetic algorihtm

```python
def genetic_algorithm(population_size, num_generations, mutation_probability, stop_fitness_score):
    init_values = generate_hyperparameters()

    best_individual = None
    best_fitness = None
    best_per_gen = []

    # Initialize the population
    population = [dict(zip(init_values.keys(), [random.choice(values)
                                               for values in init_values.values()]))
                                               for _ in range(population_size)]

    for generation in range(num_generations):
        # Evaluate the fitness of each individual in the population
        fitness_scores = []
        for idx, individual in enumerate(population):
            fitness = calculate_fitness(idx, individual, generation+1)
            fitness_scores.append((individual, fitness))

        # Sort the population based on fitness scores in descending order
        fitness_scores.sort(key=lambda x: x[1], reverse=True)
        current_best_individual, current_best_fitness = fitness_scores[0]
        best_per_gen.append((current_best_individual, current_best_fitness))
        if current_best_fitness >= stop_fitness_score:
            best_fitness = current_best_fitness
            best_individual = current_best_individual
            break
        # Select the top individuals for reproduction (elitism)
        elite_population = [individual for individual, _ in
                            fitness_scores[:int(0.4 * population_size)]]

        # Create the next generation through crossover and mutation
        next_generation = elite_population.copy()

        while len(next_generation) < population_size:
            # Perform crossover by randomly selecting two parents
            parent1, parent2 = random.choices(elite_population, k=2)

            # Create a new child by combining the hyperparameters of the parents
            child = {}
            for param in init_values.keys():
                # Perform uniform crossover by randomly selecting a parent's value
                if random.random() < 0.5:
                    child[param] = parent1[param]
                else:
                    child[param] = parent2[param]

            # Perform mutation on the child
            if random.random() < mutation_probability:
                child = adaptive_mutation(child, init_values, generation)

            # Add the child to the next generation
            next_generation.append(child)

        # Replace the current population with the next generation

        population = next_generation

    # Evaluate the fitness of the final population
    fitness_scores = []
    for individual in population:
        fitness = calculate_fitness(population_size, individual, num_generations)
        fitness_scores.append((individual, fitness))

    # Sort the final population based on fitness scores in descending order
    fitness_scores.sort(key=lambda x: x[1], reverse=True)

    # Return the best individual (hyperparameters) and its fitness score
    best_individual, best_fitness = fitness_scores[0]
    best_per_gen.append([best_individual, best_fitness])
    return best_individual, best_fitness, best_per_gen
```

Appendix D

SkySeg

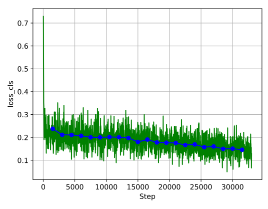
```
def remove_sky(image, pre_trained_model):
    """Removes pixels from an image where a pre-trained YOLO network has detected sky

    Args:
        image (numpy array): cv2 image

    Returns:
        numpy array: processed image
    """
    model = YOLO(pre_trained_model)
    results = model.predict(source=image, save=False, save_txt=False, conf=0.5)
    for result in results:
        if result.masks is None:
            print("no␣detections␣in␣results")
            continue
        masks = result.masks.masks.cpu().numpy()     # masks, (N, H, W)
        masks = np.moveaxis(masks, 0, -1) # masks, (H, W, N)
        # rescale masks to original image
        masks = scale_image(masks.shape[:2], masks, result.masks.orig_shape)
        masks = np.moveaxis(masks, -1, 0) # masks, (N, H, W)
        for mask in masks:
            mask = (mask*255).astype("uint8")
            mask = cv2.bitwise_not(mask)
            image = cv2.bitwise_and(image, image, mask=mask)
    return image
```

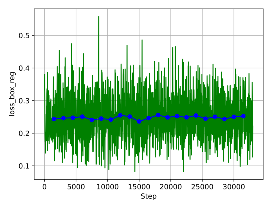
Appendix E

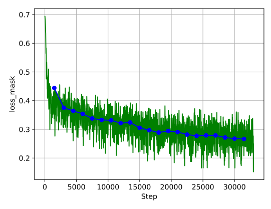
Network plots

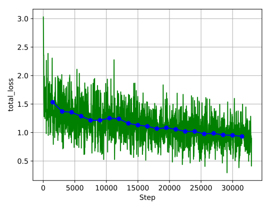
E.1 Light augmentation

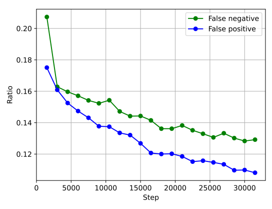
E.1.1 ResNet-50 + FPN

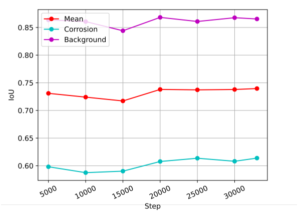

(a) Class loss


(b) Box loss

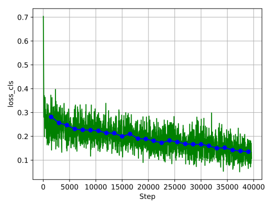

(c) Mask loss


(d) Total loss

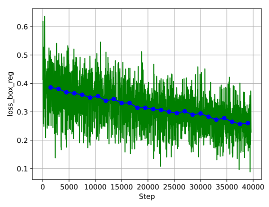

(e) False predictions


(f) IoU per iteration

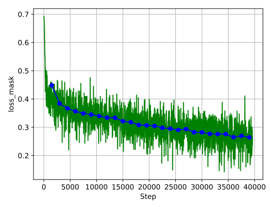
E.1.2 ResNet-101 + FPN



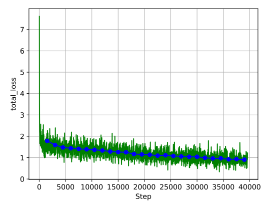
(a) Class loss



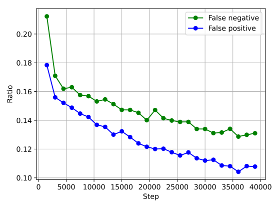
(b) Box loss



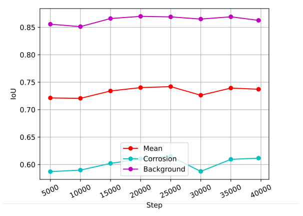
(c) Mask loss



(d) Total loss

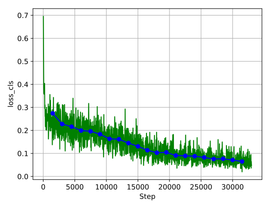


(e) False predictions

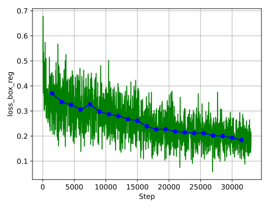


(f) IoU per iteration

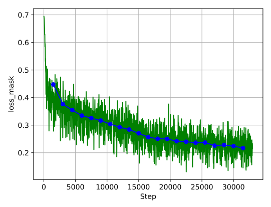
E.1.3 ResNet-50 + DC5



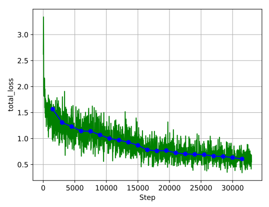
(a) Class loss



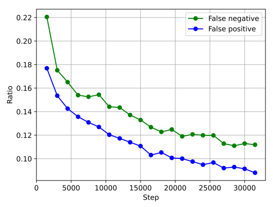
(b) Box loss



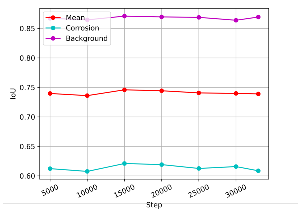
(c) Mask loss



(d) Total loss

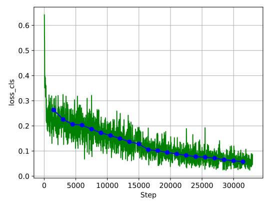


(e) False predictions

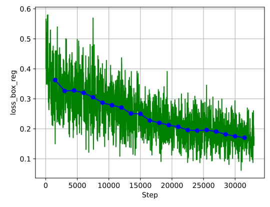


(f) IoU per iteration

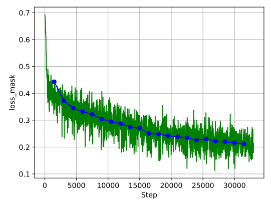
E.1.4 ResNet-101 + DC5



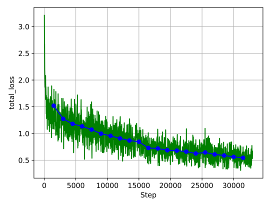
(a) Class loss



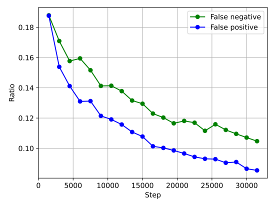
(b) Box loss



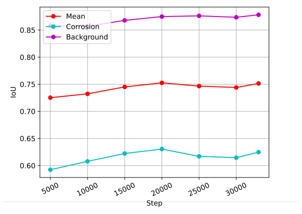
(c) Mask loss



(d) Total loss



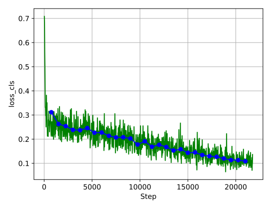
(e) False predictions



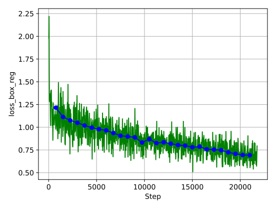
(f) IoU per iteration

E.2 GA tuned

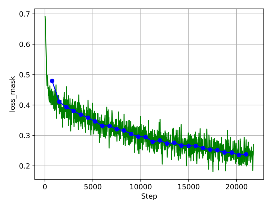
E.2.1 ResNet-50 + FPN



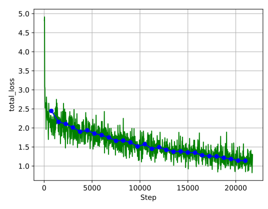
(a) Class loss



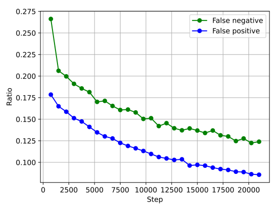
(b) Box loss



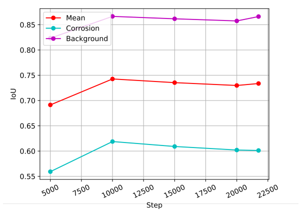
(c) Mask loss



(d) Total loss

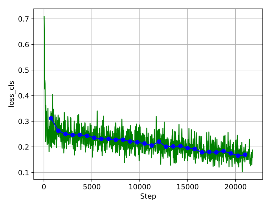


(e) False predictions

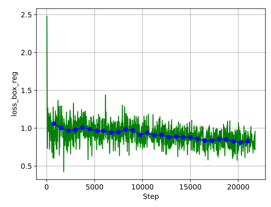


(f) IoU per iteration

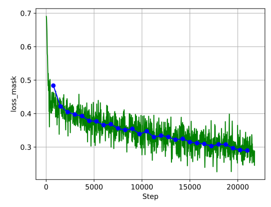
E.2.2 ResNet-101 + FPN



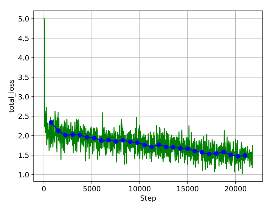
(a) Class loss



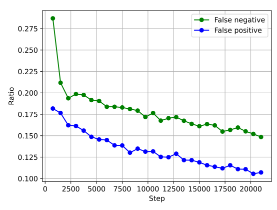
(b) Box loss



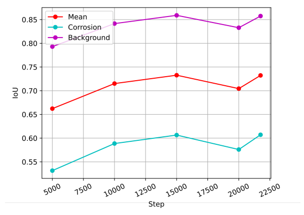
(c) Mask loss



(d) Total loss

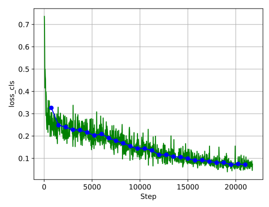


(e) False predictions

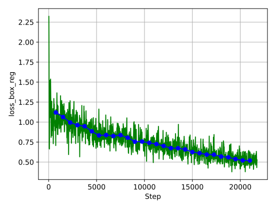


(f) IoU per iteration

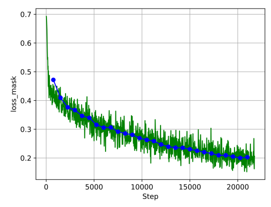
E.2.3 ResNeXt-101 + FPN



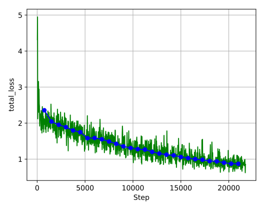
(a) Class loss



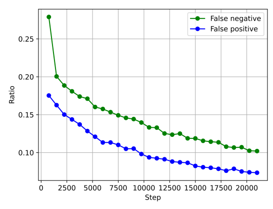
(b) Box loss



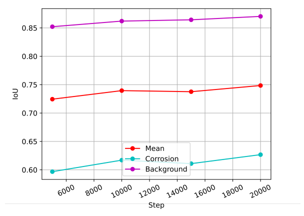
(c) Mask loss



(d) Total loss

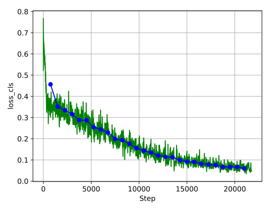


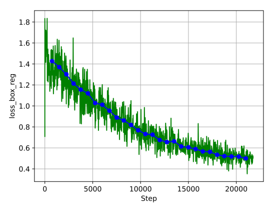
(e) False predictions

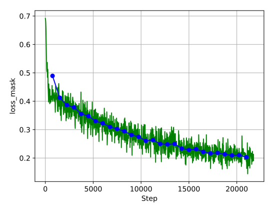


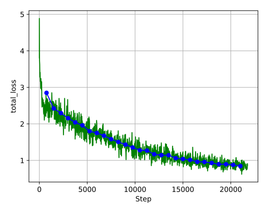
(f) IoU per iteration

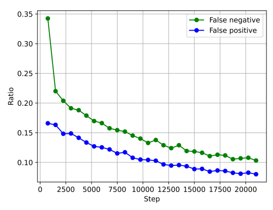
E.2.4 ResNet-50 + DC5

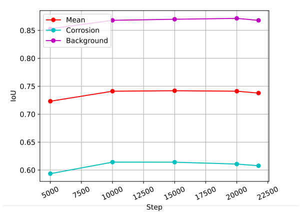

(a) Class loss


(b) Box loss

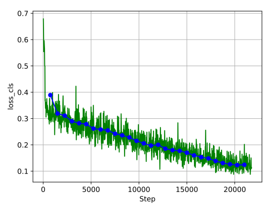

(c) Mask loss

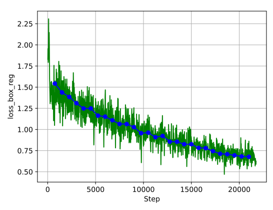

(d) Total loss

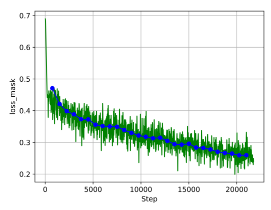

(e) False predictions

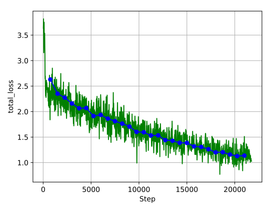

(f) IoU per iteration

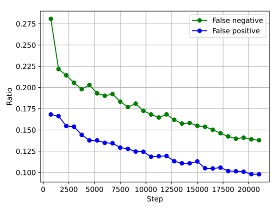
E.2.5 ResNet-101 + DC5

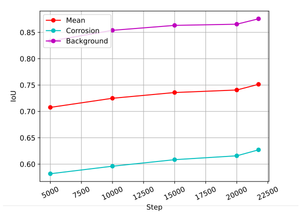

(a) Class loss


(b) Box loss


(c) Mask loss


(d) Total loss


(e) False predictions


(f) IoU per iteration