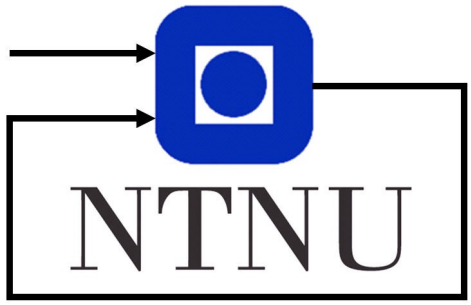
Challenges of image segmentation of corrosion damages using Mask R-CNN



Author:
Helene Semb

Supervisor:
Annette Stahl

Specialization project
Department of Engineering Cybernetics
Norwegian University of Science and Technology

December 19, 2022

Executive summary

The goal of this thesis is to present and evaluate the challenges related to automatic image segmentation of bridge damages, specifically related to corrosion. Robotic systems, with the combination of unmanned aerial vehicles and artificial neural networks, have the potential to replace manual inspection of corrosion damages to make them more effective, where the network architecture Mask R-CNN is the recent state-of-the-art model for image segmentation. To this end, a total of 1990 images have been used to train a segmentation model for detecting corrosion. The network performs at a varying degree, often detecting other objects like forests, water and clouds. Detecting corrosion damage is a complex problem, requiring the network to adapt to its varying patterns to perform properly. This paper will review methods of improving the image segmentation model based on a comprehensive literature study of similar tasks and challenges, to increase performance given specific metrics. The conclusion is that an automated hyperparameter tuning can be used to better adapt the network to the given dataset training, including making changes in the network backbone to better extract information from the feature maps, another idea is to extend the architecture to incorporate more than one network to further process images.

Table of Contents

Abbreviations

Abbreviation	Description
NPRA	Norwegian Public Road Administration
IoT	Internet of Things
CNN	Convolutional Neural Network
UAV	Unmanned Aerial Vehicle
ANN	Artificial Neural Network
FPN	Feature Pyramid Network
RPN	Region Proposal Network
FCN	Fully Convolutional Network
ROI	Region of Interest
NMS	Non-Maximum Suppression
IoU	Intersection over Union

1

Introduction

1.1 Motivation

Corrosion of steel is a significant concern in all fields of infrastructure, specifically related to bridges and their carrying capacity, with lasting economic and environmental impact [1]. If left unaddressed, severe degradation can lead to catastrophic failure of the construction, as seen in the 2019 collapse of the Nangfang'ao bridge in Taiwan [2]. The bridge, which was designed to last for 50 years, collapsed after only 25 years of operation. An investigation revealed that the cause of the collapse was a lack of maintenance. The global cost of corrosion is estimated to be 25 trillion Norwegian kroner (NOK) per year, equivalent to 3.4% of the global GDP [3]. It has been suggested that between 25% and 30% of annual corrosion costs could be saved through the use of optimal corrosion management practices [1].

In Norway, the Norwegian Public Road Administration(NPRA) [4] is responsible for the inspection and maintenance of over 1000 steel bridges. To withstand corrosion and achieve the design lifetime of a steel bridge, NPRA applies a zinc coating along with a layer of paint. Regular recoating is needed, and NPRA is obligated to inspect a bridge every five years. Today, inspection is done manually on-site, where qualified engineers and inspectors take pictures and write down the damage evaluation. This process is labor-intensive, costly, and time-consuming. There is therefore a significant opportunity to automate inspections using unmanned aerial vehicles (UAVs) and image classification. By developing a segmentation algorithm to detect and classify the severity of corrosion on steel bridges, inspections can be done thoroughly and more efficiently. By removing human subjectivity, it can also be more certainly classified when maintenance is needed, saving potentially millions of NOK.

1.2 Background

The subject of machine learning and image segmentation is constantly growing and improving. In just the last two years there has been a growing number of publications about Mask R-CNN, as seen in fig. 1.1 [5]. There are several projects related to classifying and segmenting corrosion on steel structures. The latest thesis written for SINTEF Industry on this current project was published in June 2020, which concluded great promise in the state-of-the-art Mask R-CNN [6] architecture for instance segmentation of bridge corrosion [7]. The project was done on a dataset of 608 annotated pictures, which is quite small for such an extensive task as image segmentation. Two years later, the size of the dataset has tripled, containing almost 2000 pictures with a wide range of corrosion damages on bridges and other steel structures. There is also constantly new research on how to improve the accuracy of neural networks with data augmentation and parameter tuning. It is therefore safe to say that the model has the potential for greater performance than what it currently performs today.

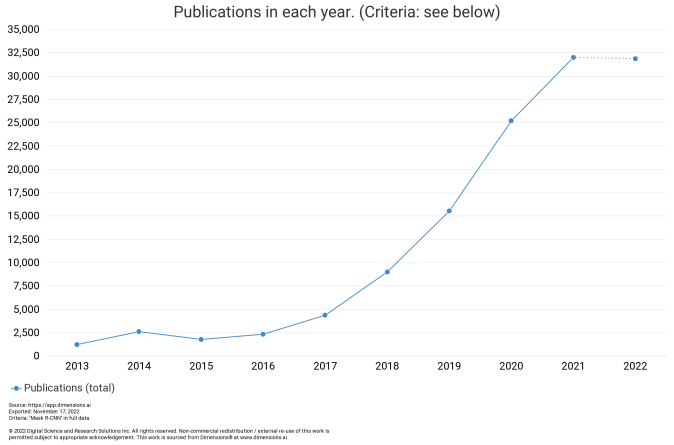


Figure 1.1: Number of publications mentioning Mask R-CNN per year, from dimension.ai[5]

1.3 Objectives

The objective of this project thesis is to present the current performance of the network, and propose methods of improvement based on a comprehensive literature study. The proposed methods are not implemented in this work, therefore the improvements are only hypothesized based on performance from former research. As the use of Mask R-CNN on corrosion segmentation is a narrow research field, the current research used in this thesis also is based on other datasets and problem formulations. This thesis mostly focuses on published work over the last two years and evaluates how well these changes compare to the corrosion dataset.

1.4 Structure of the report

Chapter 2 introduces the fundamental theories of convolutional neural networks (CNNs), which will be used to explain the architecture of Mask R-CNN and the tools it uses to improve performance. It is assumed that the reader has a basic understanding of artificial neural networks, and we will only briefly discuss these concepts. Chapter 3 discusses the use of computer vision techniques for the inspection of steel bridges and the challenges involved. Chapter 4 presents the current implementation of our algorithm and dataset, and demonstrate some results. Chapter 5 delves into the challenges faced by our current implementation and suggest potential improvements, ultimately concluding with the most promising ones.

2

Theoretical background

2.1 Classification, object detection and segmentation

Computer vision relates to how researchers can automate tasks of the human visual cortex by extracting useful information from image data [8]. Advances in the field began in the 1960s, and sought to extract shape information about simple objects such as edges and boxes. Later methods experimented with more complex problems with the development of different representations of image patterns. Breakthroughs such as Optical Character recognition [9] served purposes like automated license plate recognition, already showing potential for real use-cases.

The classical problem in computer vision has always been determining whether or not the image contains some specific object or feature, and to be able to correctly detect and classify the object. With the growth of artificial neural networks (ANNs) and specifically Convolutional Neural Networks, algorithms have been able to build high-quality perception systems for complex visual problems.

The common tasks for computer vision has been

- **Object classification**: The task of assessing the correct class of an object
- **Object detection**: the task of detecting instances of objects of a certain class within an image. This object classification combined with determining the location of the object while other object classes may be present.
- **Object segmentation**: Classifying every pixel related to an object, creating ”masks” and determining the area covered by that object in the image.
  - **Semantic segmentation**: Objects of the same class are regarded as one entity.
  - **Instance segmentation**: Identifies the number of individual objects (instances) within one class.

2.2 Convolutional Neural Networks

For a traditional ANN to be trained on an image, each pixel will be regarded as a single input to one neuron. Using an example with a 1028x1028 RGB image, the total amount of weights per neuron would be $3 * 1028 * 1028 = 3170352$ which makes the structure of an ANN impractical to work with on complex image classification tasks.
Convolutional Neural Networks (CNNs) [10] are a type of ANN specialized for image processing, first introduced in the 1980s by Yahn LeCun [11] to classify handwritten digits. It is similar to a traditional neural network in that it is composed of neurons that self-optimize during training, with the key difference being in the organization of the neurons in dimensions; the spatial dimensionality of the input (height and width of the image) and the depth. Unlike standard ANNs, the output of a given layer is only connected to a small region of the input, instead of being fully connected.

2.2.1 Overall architecture

The goal of a CNN is to produce feature maps, also known as activation maps, with reduced dimensionality instead of a vectored output. A feature map corresponds to the activation of different parts of the image, a high activation means a certain feature has been found. This type of feature extraction is done in a convolutional layer, where a nxn dimensional filter, also known as kernel, slides over the input image computing convolutions, where the number of feature maps is decided by the depth of the current layer. Figure 2.1 visualizes the procedure.
Another important layer of CNNs is the pooling layer, where the goal is to reduce the spatial dimension of the feature maps. The two most common pooling layers, shown in fig. 2.2, are MaxPool, which extracts the strongest features in an activation map, and AvgPool, which averages the input from the kernel, giving a more smooth feature extraction.
A common CNN architecture aims to reduce the spatial dimension while increasing the depth for each layer, where an example can be seen in fig. 2.1. A common tactic is to flatten all the feature maps to the usual neural network structure, with a fully connected layer at the end, to perform classification with softmax or sigmoid. It is also possible to replace this layer with another convolutional layer using 1x1 kernels, in which we would call the architecture a fully convolutional network (FCN).

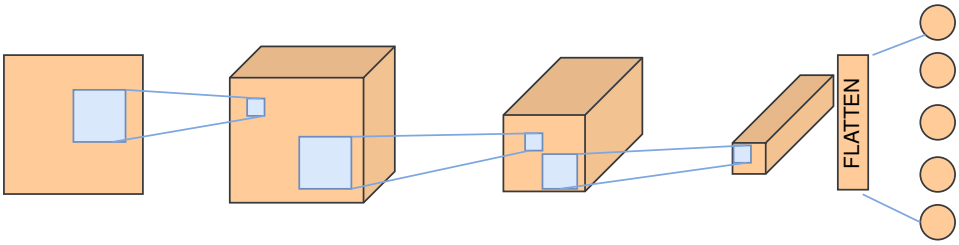


Figure 2.1: A typical CNN architecture. the blue field to the left represents the kernel, and the field to the right represents the output pixel from one convolution.

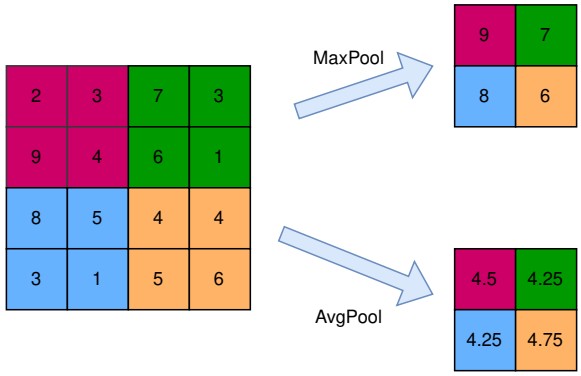


Figure 2.2: An example of the outputs of a MaxPool layer versus an AvgPool layer with a 2x2 kernel and stride 2. The colors represent the receptive field of the output pixel

2.3 Mask R-CNN

Mask R-CNN [6] is a deep learning based technique developed in 2017 by researchers at Meta AI (changed name from Facebook AI in 2021) for instance segmentation. It is a two-stage process, where a region proposal network generates regions of interest, which are then further classified and segmented.
Mask R-CNN is built on top of the Faster R-CNN [12] object detection architecture and adds a mask branch for pixel-level object segmentation. It extends the Faster R-CNN architecture by adding a branch for predicting an object mask in addition to the existing branch for bounding box recognition. The mask branch takes feature maps from the backbone CNN as input and predicts a mask of shape (MM) for each ROI, where M is the resolution of the mask. The output is then three folded for each candidate object; the class label, bounding box offset and the object mask.
The main advantages of Mask R-CNN are its accuracy and speed. Compared to other instance segmentation techniques, Mask R-CNN has been found to be more accurate and faster. It also has the advantage of being well-suited for training on large-scale datasets.

2.3.1 Architecture

The Mask R-CNN pipeline can be divided into four main parts, visualized in fig. 2.3. To shortly describe each part:

1. **Backbone**: The input image is pre-processed in a feature extraction backbone network (in fig. 2.3 we use ResNet-101 + FPN). The goal is to produce feature maps for further detection.

2. **RPN**: The feature maps obtain a large number of candidate frames, known as regions of interest (ROI) through a regional proposal network. The ROIs are given two binary scores of objectness along with a bounding box offset for each frame.

3. **ROIAlign**: The feature maps and the remaining ROI are sent to the ROIAlign layer, so that each ROI generates a fixed size feature map.

4. **Segmentation + classification**: The flow goes through two branches, one fully connected layer to perform object classification, and one branch entering a fully convolutional network (FCN) for pixel segmentation.

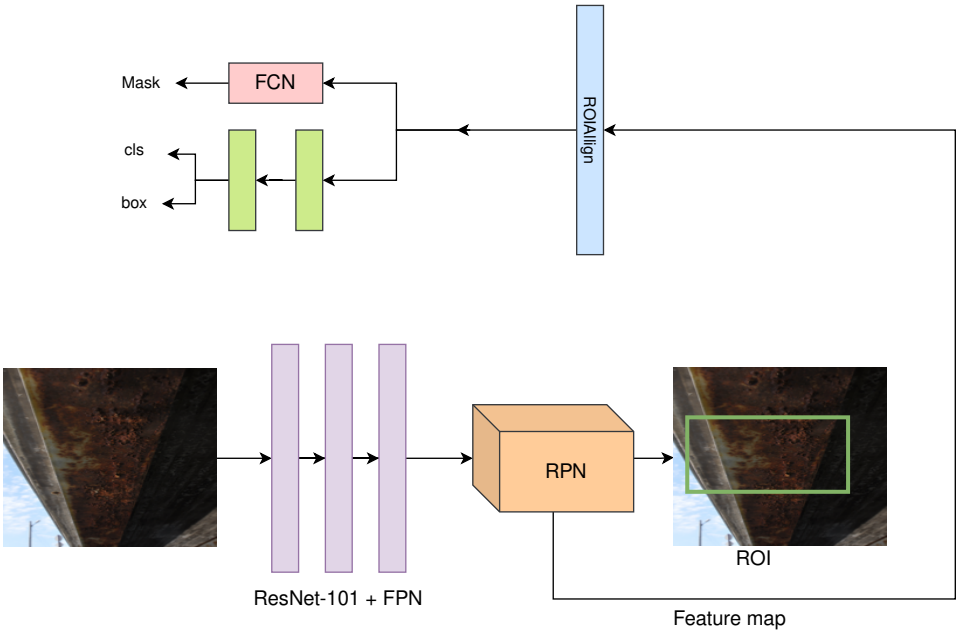


Figure 2.3: Mask R-CNN architecture

Backbone: ResNet + FPN

The first section of Mask R-CNN is a backbone model that serves as a feature extractor. This network follows the same principles desribed in section 2.2 and fig. 2.1. Performance of feature extraction is directly related to the depth of the convolutional neural network, but increasing network size can lead to deprecation of the network, including problems such as disappearing gradients. The solution to this common problem is to introduce residual blocks [13], creating what is known as Residual Neural Networks, or ResNets. In ResNets, a residual block is convolutional block with an added skip connection, a ”shortcut” for the gradient to be directly propagated to earlier layers, making us able to extend the amount of layers of our feature extractor. In [6] ResNet with different amount of layers is tested, which is why we use this as our backbone in further work as well. The size of the network can vary, but in this assignment we focus on the 101-layered residual network, known as ResNet-101, where fig. 2.4 shows the different blocks of layers.
The CNN architecture forms a contracting path with increasing semantic value and decreasing resolution. Scaling the final feature maps. The decreased resolution makes it challenging to detect objects, in particular for small features. To have both high semantic value and resolution, Mask R-CNN has implemented a Feature Pyramid Network (FPN)

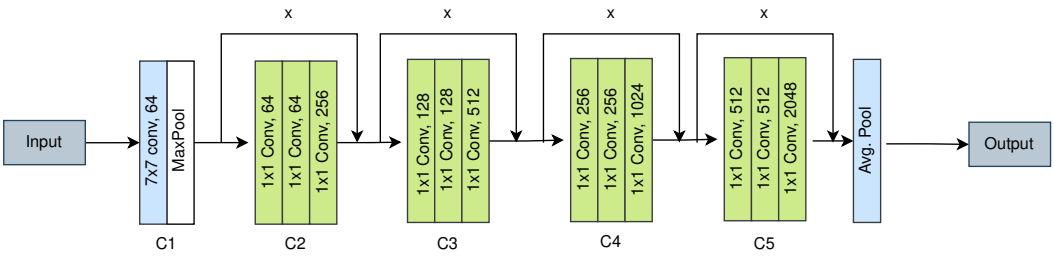


Figure 2.4: ResNet layers: Here the skip connections are shown in the identity of x "skipping" the convolution layer and is added at the end of each block. There are five unique blocks in the network, where each layer is shown with their nxn size filter and depth. C1 is performed without skip connection once followed by a MaxPool layer, C2 is repeated 3 times, C3 four times, C4 23 times and C5 3 times, the final step is then a average pool layer.

[14] to generate multi-scale feature maps with better quality information. It is composed of a bottom-up and top-down pathway as visualized in fig. 2.5. The bottom-up is the usual ResNet, while the top-down pathway constructs high resolution layers from a semantic rich layer. The reconstructed layers, or the M-blocks in fig. 2.5 are created from adding the up scaled output of the previous block and the corresponding C block. Each scale level is then sent to a Regional Proposal Network, which in fig. 2.5 is represented by the P-blocks.

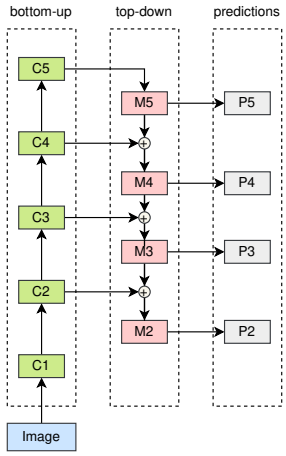


Figure 2.5: Feature Pyramid Network

Region Proposal Network

A Region Proposal Network (RPN)[12] takes the feature maps produced from the feature extractor as input, and outputs a range of ROIs where an object has been detected. The

ROIs come each with an objectness score and a bounding box offset. The objectness score is two-dimensional and computed with Softmax, and show one probability that there is an object in the particular region, and one probability that the region only contains background.

The process is shown in fig. 2.6. A $3x3$ kernel moves over the feature map and generates a particular set of anchors for each pixel coordinate (x_a, y_a). In this case the network generates anchors with three different aspect ratios (AR) for the height and width (h_a, w_a), and three different scales (S_1, S_2, S_3). This gives a total of 9 anchors for each position of the feature map. For each feature map with width W and height H this gives us a total of W*H*9 anchors.

The next step for RPN is to classify and localize the anchor box to produce region proposals. Classification is done by a Bounding Box Classifier, which compares the anchor box with the Ground truth, and outputs the objectness score.

The regions that have been classified as foreground are then localized with a Bounding Box Regressor layer, which computes the offset between the region and ground truth box, and outputs w,h,x,y as the bounding box offset. Where (x,y) is the center of the box, w and h are width and height. RPN will filter out the regions with low objectness score and overlapping regions using non-maximum suppression(NMS) [12], and output the number of ROIs up to the maximum count, which also can be configured.

It should be noted that RPN does not care about the class of the object, only if there is an object in the box or not. Given the level of confidence in our network

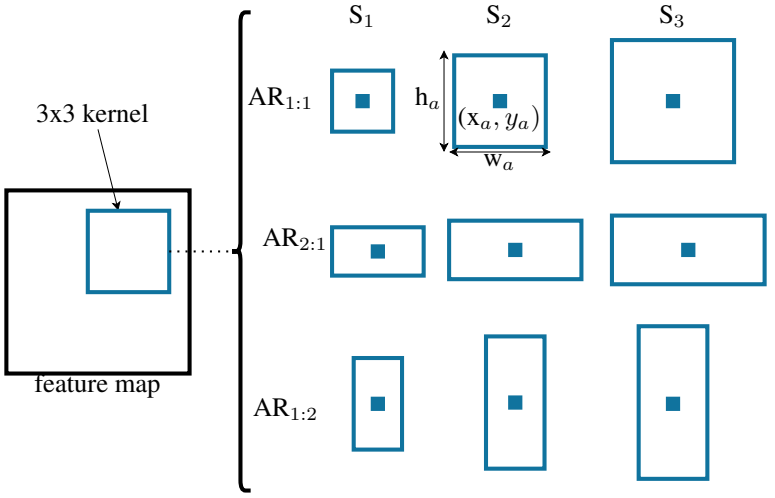


Figure 2.6: Region proposal network

ROIAlign

As the ROIs are of different scale and ratio, and the classification network expects a fixed size input, a conversion is needed before the regions are sent to the next network. Faster

R-CNN used what is known as ROIpooling [12], but due to quantization the regions can lose a lot of important information.
Meta AI developed ROIAlign instead [6]. The algorithm is designed to improve the performance of object detection by providing a more accurate alignment of the object's features in the image. It works by dividing the ROI into an evenly-spaced grid and computing the features at each location by bilinear interpolation. This interpolation method produces more accurate features that can be used for better object detection. Additionally, ROIAlign reduces the amount of time required to process a large image since it only needs to process the ROI instead of the entire image. This makes it an efficient and effective solution for object detection tasks.

Classification + segmentation

The final step of Mask R-CNN is running two parallel networks for the predicted output. This is where the model differs from its previous versions. While Faster R-CNN only has one classification network built by fully connected layers, which will again produce the probability of a given class and a refined bounding box regressor from the ROIs produced by RPN, Mask R-CNN adds a FCN to produce k mxm binary masks for each ROI, where k is the number of classes, meaning one mask is generated for each class leading to less competition between classes. This two-network structure allows Mask R-CNN to decouples mask and class prediction, meaning they do not depend on each other, which increases performance significantly according to He[6].

2.3.2 Loss function

During training, the complete loss function is calculated as a sum of the tasks of the last network predictions. For each ROI, the loss is defined as follows

$$L = L_{cls} + L_{box} + L_{mask} \tag{2.1}$$

Where L_{cls} represents the classification loss, L_{box} represents regression loss of the predicted box and L_{mask} represents the summed binary cross entropy loss per pixel

$$L_{cls}(p_i, p_i^*) = -[p_i^* \log(p_i) + (1 - p_i*) \log(1 - p_i)] \tag{2.2}$$

$$L_{box}(t_i, t_i^*) = R(t_i - t_i^*) \tag{2.3}$$

$$L_{mask} = -\frac{1}{N} \sum_i [x_i^* \log p(x_i) - (1 - x_i^*) \log(1 - p(x_i))] \tag{2.4}$$

For the classification loss in eq. (2.2) is defined as the log loss function with 2 classes, whether the detected object is the target or not, here p_i represents the predicted probability for the class of an object in anchor i, while p_i^* represents a binary ground truth (1 for target object, 0 for not). The regression loss function R in eq. (2.3) is the robust loss function (smooth L_1)[15], where t_i is a vector of 4 coordinates of the predicted bounding box, while t_i^* is the corresponding ground truth coordinates of the bounding box. The regression loss

is only activated for $p_i^* = 1$. L_{mask} in eq. (2.4) is computed by applying a per-pixel sigmoid on each of the k binary masks and compute the average cross entropy loss. It is only the mask associated with the ROI class that contribute to the loss. N represents the number of pixels in the mask (m^2), $p(x_i)$ is the predicted value of the pixel given by sigmoid, and x_i^* is the binary ground truth value for that pixel.

2.3.3 Evaluation metric

Reducing the performance of a model down to a single number (the loss) can obscure detail in the model results that may be important. It is therefore possible to use other metrics when measuring the performance of Mask R-CNN, mainly in computing scores for how many pixels the prediction scored correctly. By using the definition for binary classification defined in a confusion matrix seen in fig. 2.7, it is often the percentage of true positives we want to evaluate, which can be done through precision or recall. They are defined as

$$PRECISION = \frac{TP}{TP + FP} \tag{2.5}$$

$$RECALL = \frac{TP}{TP + FN} \tag{2.6}$$

In other words accuracy measures how many predicted positive values that are actually correct, while recall computes how much of Ground Truth the algorithm detected. Though easy to use, their drawbacks are that it can be easy to achieve high precision or recall without improving the overall network performance. With precision it is trivial to simply predict any output as negative, while for recall one could predict any pixel as positive. One metric that solves this issue and is a very popular metric for object segmentation, is Intersection over Union (IoU). It is defined as the intersection between Ground Truth and the prediction mask, divided by their union. A better representation is shown in fig. 2.8. We can also derive it as

$$IoU = \frac{TP}{TP + FP + FN} \tag{2.7}$$

We can see this evaluation metric being used often, such as in the PASCAL VOC image challenge[16]. The metric allows for the bounding box prediction to be rewarded when it heavily overlaps with the ground truth box even if the coordinates do not match 100%.

2.3.4 Transfer learning

Training a Mask R-CNN model from scratch can be time-consuming and require a large amount of labeled data. Initializing weights can also be a difficult challenge and lead to local minima or poor training results. Transfer learning is a technique often utilized in training deep neural networks. It involves using the knowledge and weights learned by a pre-trained model on a large dataset and applying them to a new task and dataset. This can be useful when training a Mask R-CNN model for a specific task, as it can allow the

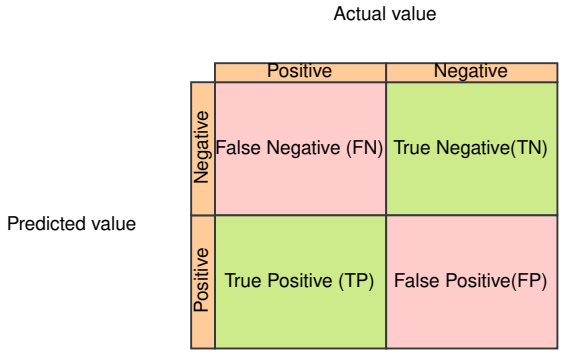


Figure 2.7: Confusion matrix

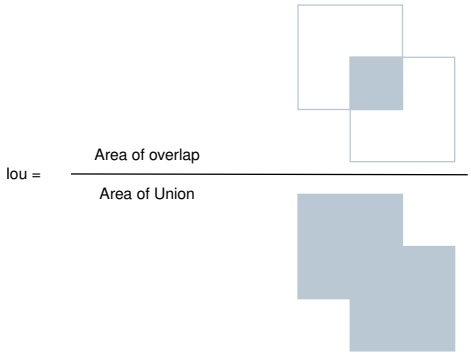


Figure 2.8: Visualization of how IoU is computed

model to make use of the knowledge learned by the pre-trained model and converge to a good solution faster and with less data.

Pre-trained models are often trained on large, high-quality datasets that contain a diverse range of examples. This can provide the pre-trained model with a strong foundation of knowledge that can be useful for many different tasks. For feature extraction, there are many common features for images even as they are not related to bridges, such as edges. Transfer learning can also help to prevent overfitting. When training a Mask R-CNN model from scratch, it is possible for the model to overfit to the training data and perform poorly on new, unseen data. By using transfer learning, the model can learn more generalizable features that are less likely to overfit to the training data.

One of the most popular existing datasets used for transfer learning is the Microsoft COCO: Common Objects in Context [17], the dataset contains 91 object classes of common objects that are easily recognizable, e.g ”Cat”, ”Bicycle” or ”Pizza”. The COCO weights for Mask R-CNN are already trained and easily accessible [18].

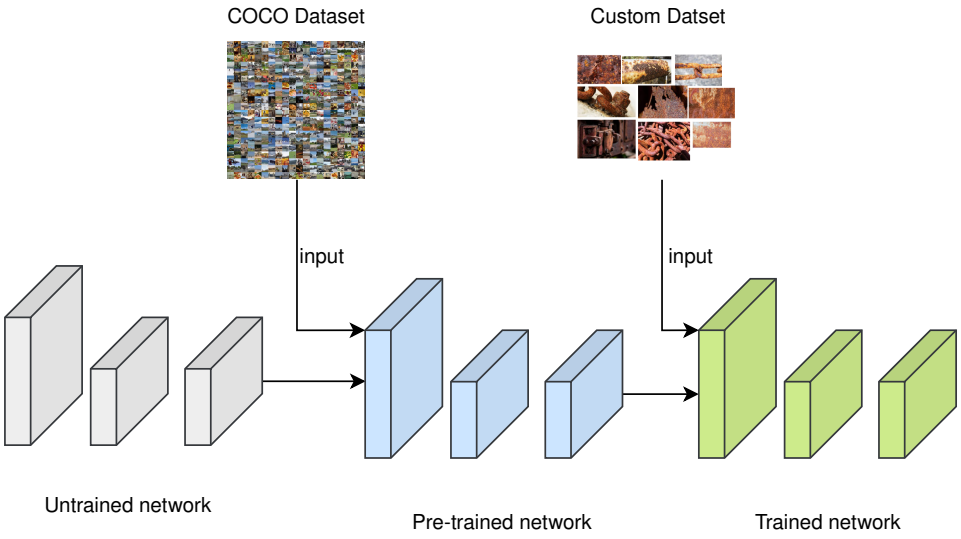


Figure 2.9: Transfer learning for Mask R-CNN

2.3.5 Hyperparameters

Hyperparameters are adjustable parameters that control the model's behavior and performance. These parameters are typically set before training the model, and they can have a significant impact on the model's ability to learn from the data and make accurate predictions.

Mask R-CNN has many of the standard deep learning hyperparameters, such as number of epochs, batch size, learning rate and learning momentum. Batch size is only chosen based on the limitations of the GPU used for training, while epochs and learning rate can have major consequences for our network, as too few epochs and wrong learning rate can lead to a non-optimal network. These are important to test for the correct assignment of values. Other parameters that can have an effect on the training is image size and loss weights, where a good resolution on the images will make it simpler for Mask R-CNN to detect small features. The loss weights are the weights assigned to the different parameters in eq. (2.1) and the loss function for RPN, which affects which loss function we want to evaluate higher. This could become useful after further testing.

The hyperparameters of RPN are related mostly to the number of proposals generated, their size and overlap threshold, along with the minimum confidence for the ROI to be classified as having an object or not. As the last part of Mask R-CNN related to detection and mask generation are very similar to RPN. the hyperparameters are very alike, such as confidence threshold, maximum number of detections and maximum number of ground truth instances. Mask R-CNN has about 18 hyperparameters that have potential effect on the performance [19]. Tuning these can be challenging, as they can interact with each other in complex ways. In general, it is best to try out a range of different settings and see which ones work best for the specific task, which we will discuss more about in chapter 5.

3

Image segmentation on structure damages

This chapter will give an overview of the goal of using computer vision based techniques in infrastructure inspection, along with the recent research in the field. It will in addition address to why we have chosen instance segmentation with Mask R-CNN- The chapter will also introduce some common challenges with using neural networks.

3.1 Goal of automated inspection

The long-term goal is to eliminate human intervention in infrastructure inspections in order to obtain an objective evaluation of damages, such as cracks, paint flaking, asphalt damage, and corrosion. This could potentially be enhanced with the use of digital twins [20], which create a digital 3D model of the structure. With frequent automated inspections, the digital twin would be constantly updated with an overview of damages and their progression.
In later years there has been a wish to digitize the maintenance of infrastructure as a part of Internet of Things [21]. By implementing an automated classification algorithm, this can later be extended to automated inspection with UAVs along with monitors, sensors and cameras to get automated continuous inspection of infrastructure, which is the goal of Structural Health Monitoring [22]

3.2 Current state

Research on automated inspection of infrastructure based on computer vision methods is in constant development, and offers great promise even though existing algorithm have not fully matured yet to be fully realized out in the field. Spencer et al. [8] provides a brief overview of the advances in in computer vision based techniques related to civil infrastructure.

Early research in corrosion detection delved in traditional computer vision methods like using wavelet [23], but with the recent development and use of neural networks in medical imaging, autonomous driving and facial recognition, the research suggests that CNNs vastly outperform traditional computer vision algorithms in terms of object detection [8]. When choosing the architecture for the problem, one could choose between a variety of suitable algorithms. One choice lies in whether to do simple object detection or detection and segmentation of the damage. Relating to previous section with the goal of an continuous supervision, it is more practical with segmenting the area of corrosion to evaluate the graveness and damage of the bridge. A clear segmentation of areas also aids a potential automated inspection drone in orientating itself with the framework of Visual-Simultaneous Localization and Mapping(VSLAM) [24]. As Mask R-CNN is the recent state-of-the-art algorithm for segmentation, and its two-step architecture with both feature extraction and RPN, makes it a good choice in the complex assignment of corrosion segmentation. As it builds on the Faster R-CNN and R-CNN architecture, it solves many of the challenges in facing object detection, by adding multi-task loss for both localization and classification error. The RPN structure aids in the problem of objects appearing in different scales and ratios. It would also be easy to switch back to object detection without segmentation by using Faster R-CNN instead.

3.3 Related research

Mask R-CNN is one of the most widely used segmentation algorithms in computer vision research today. The algorithm has been applied in several areas such as facial recognition, autonomous driving and damage inspection. The most researched field is by far in medical imaging, where Mask R-CNN has been used for organ classification and tumor detection in CTs, MRI and ultrasound [25].

3.3.1 Object detection of damages

The wide field of damage detection using neural networks have long been regarded as the primary focus for digitizing the infrastructure section. The most popular research by far has been crack detection, with the invention of CrackNet [26] on 3D images showing promising results. Detection of cracks in pavements have also been tested with Mask R-CNN and Faster R-CNN [27], with a relatively small dataset. Showing promising results for simple damages like straight cracks, even with interference from sunlight, but performed slightly worse for cracks with more complex structure. Testing on the CRACK500 dataset, it showed a clear need for a larger dataset for it to perform good detections.
Mask R-CNN has also been used for a variety of other damage detection tasks, such as aircraft dents [28], car inspection [29] and inspection netting damage [30]

3.3.2 Corrosion detection

To this day research on segmentation of corrosion is sparse, due to the lack of training data. There has however been some new research the last two years, where several have tested other network architectures than Mask R-CNN. Shi et al. implemented a corrosion

Figure 3.1: Nordhordland bridge

segmentation algorithm using squashing and cropping based on VGG-Unet [31]. With only 200 corroded images, cropping segmentation can increase the dataset majorly. This involved a lot of images where no corrosion was presents, so to improve the capability of the network, Background Data Drop Rate was defined to control the proportion of pixels in each category, randomly dropping some of the images where only background was present. It was shown that with an increased background drop rate, the mIOU increased.
Rahman et al. developed both a tool for automatic image annotation using a small set of labeled images, using a texture based segmentation method integrated with red-green-blue feature based classifier optimization, and a standard CNN model for training and segmentation. [32]
Burton et al. proposed in 2022 RustSEG [33], a deep learning algorithm for producing segmentation masks without annotation. The schematic for RustSEG was using a CNN-based classifier with a 50% confidence rate, where the images classified with corrosion are passed to a localization function, which is an adaption of the Grad-CAM++ method and returns a heatmap of corrosion in the image. A threshold filter is then used to produce the final output mask. Several refinement techniques were considered when no Ground Truth was present, such as conditional random fields (CRF), which considers the boundary of the mask given colour, texture and contiguity.

3.4 Challenges

3.4.1 Bridges

The structure of bridges is mostly recognizable for a computer vision algorithm. This is due to their contrast with the background, which tends to be green nature, blue skies or water, and sharp edges which can be detected by feature extraction. It is however difficult

Figure 3.2: Stavne bridge

to generalize bridges for one single algorithm, as the type and structure of a bridge can vary immensely. Take for example the Nordhordland bridge, visualized in fig. 3.1, which is a type of pontoon bridge floating on top of the water. There are barely any pillars and the distance between the bridge and the ocean is short. Compared to a typical bridge structure where pillars are taken into consideration, this can produce other type of images and different scales to every image. Additional to shape and type, bridges also come in varying colors, as displayed with the Langøy bridge in fig. 4.3 which is blue and Stavne bridge in fig. 3.2, which is both grey and red. The important lesson to take from this is to acquire training sets from all types of bridges, in order to not overfit the algorithm to one specific type or colored bridge.

Images are captured both with UAVs and handheld cameras with varying quality, both due to the resolution of the camera and the photography skills of the inspectors. Images are also collected without regards to light or angles, where some images will be of only concrete with shadows, and some will include blue sky, water and animals. One thing to also notice is the disturbance of other brown-like objects on bridges which is not corrosion, here the picture in fig. 3.2 is a good example, with objects like dirt and graffiti which could look like corrosion.

An idea to reduce the distribution would be to standardize the way inspections are done and have a pre-planned route for the UAVs, but this could prove challenging due to multiple causes. One is the different shapes and sizes of bridges, making the route unique for each bridge. The other is that new corrosion can develop outside of the planned route, needing specific inspection either way. If one were to develop a monitoring process where the model could predict further development of corrosion, it is beneficial either way to calibrate images the same way for optimal performance.

3.4.2 Detecting corrosion

All though corrosion is easy to spot for the human eye, it can be difficult for a deep learning algorithm to fully comprehend the patterns they pose, due to their sporadic growth. It can also make for some complex annotating Examples where the work of detecting corrosion is easy, can be seen in fig. 3.3. This is because of the clear boundary between corroded and non-corroded material, and the contrast to the background. We can verify this by algorithms like RustSEG and VGG-Unet having high accuracy for these kinds of photos. These images are a good starting point for training the algorithm, but realistically there are many outliers.

Figure 3.4 shows examples of where both annotation and classification can be difficult. Either by scattered corrosion dots with a small diameter or soft corrosion without well-defined edges. Bridges are also subject to other types of damages such as white corrosion, which forms on the zinc material[4], or flaking paint. These are also classified as corrosion, which can disturb the finding of global optimum in the training, and lead to other misclassifications. Corrosion can also be of different colors, with the severity of corrosion not being classified. A possible solution could be to identify several classes of damage and degree of corrosion, but this would also depend on the weighted balance of the dataset for each class, as the amount of data ultimately skews towards brown corrosion.

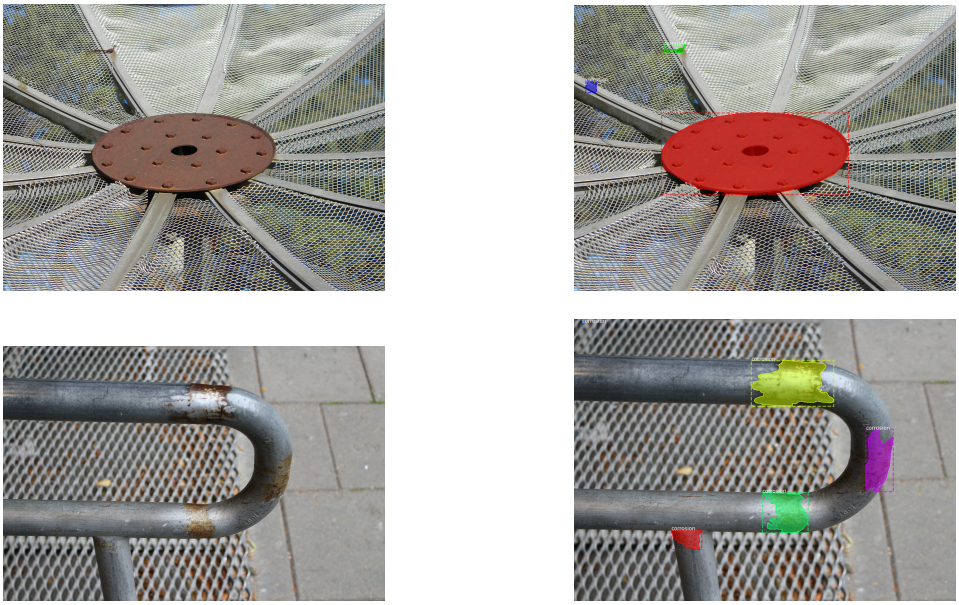
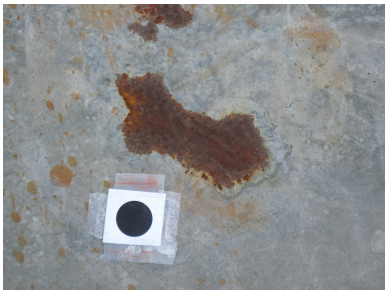


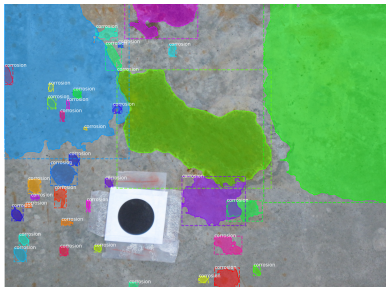
Figure 3.3: Examples from dataset with clear boundary lines

3.4.3 Dataset

The crux of any CNN model performance is the lack of available high quality training data. The more complex the problem is, the higher the need is for accurate data representation, which is highly relevant for this problem. Both section 3.4.1 and section 3.4.2 talk about the high variance in types of corrosion and bridges, which concludes that the quality of data is imperative to the success of the algorithm.
Another reason that speaks the importance of the dataset is the overfitting vs underfitting problem, which is a common issue for all neural networks. The combination of a too small dataset along with a highly complex model such as a 101-layered Residual Neural Network can lead to the model being biased towards the training images and failing to generalize to the features, meaning the network overfits. However, too many outliers in the dataset can make the network struggle to do proper optimization, i.e it underfits.
However much one can talk about the importance of gathering enough data, it is a complex and time consuming process of acquiring good data and to annotate it. A tool many computer vision engineers use is data augmentation, which is to artificially create new data from existing by flipping, cropping, blurring etc. so that the dataset is increased without having to gather new data. A survey testing smaller networks for corrosion damage [34] used cropped images 128x128 creating a total training set of 50,000 images and 4856 images for validation from 926 acquired corroded images.
The conclusion most researchers achieve in their studies [33, 28, 29] is that new data should be gathered to increase performance. This is a continuous process that will be constantly worked on, but data augmentation is a method that can decrease this need.



(a) Scattered corrosion

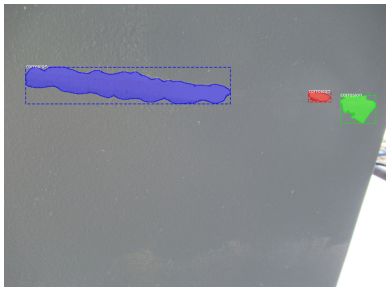


(b) Corresponding mask to (a)

(c) Unclear boundaries

(d) Corresponding mask to (c)

(e) White corrosion



(f) Corresponding mask to (e)

(g) Heavy corrosion sorrounded by light corrosion

(h) Corresponding mask to (g)

Figure 3.4: Selection of images from dataset

4

Implementation

Fondevik implemented the code for training Mask R-CNN on a small corrosion dataset [7]. All though the dataset have increased, a lot of the code from 2020 will be reused for the purpose of testing the performance as a starting ground. Very little work have been done with the code by the author, which can be found in section A, and this is a brief overview of the implementation we have as of now, along with some results.

4.1 Dataset

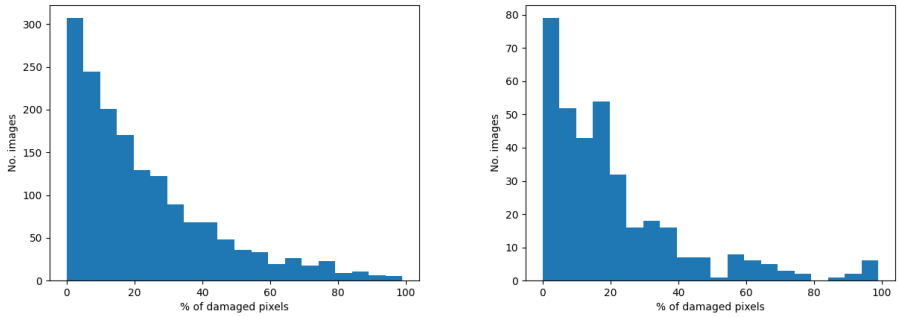
The complete dataset consists of a total of 1990 images and 15030 instances, with a 22.3% average of corroded pixels. The train-val split done is better described in table 4.1.

4.1.1 Image acquisition

As there is no pre-existing dataset for bridge corrosion, the pictures being used for training have been collected over the last years from several sources, with a total of 1632 images for training, and 358 images for validation. This includes objects from which are not bridges, as in fig. 3.3, but since corrosion develops universally the same regardless of object, this should not be an issue. Images have also been fetched from real inspections by the Norwegian Public Roads Administration, as exemplified in fig. 3.4. The distribution of damage is shown in fig. 4.1, where we see that most of the images have a very small percentage of actual corroded pixels. This is an important factor when evaluating

Dataset	# Images	# Instances	Avg. corroded pixels
Train	1632	12279	22.14%
Val	358	2751	20.45%

Table 4.1: Metadata statistics for the dataset



(a) Train dataset **(b)** Val dataset

Figure 4.1: Histogram of images with % corroded pixels

4.1.2 Image processing

For training, a small amount of data augmentation has been done, mostly in horizontal and vertical flipping, as corrosion is invariant of direction. 50% of all images will be flipped vertically, and 50% will be flipped horizontally. For future work, more data augmentation methods can also be considered such as blurring, rotating and the methods suggested in section 3.4.3.

4.1.3 Image annotation

Annotations have been done using AI assisted tools from V7 labs [35]. The new toolkits have made the time-consuming annotation part of supervised learning tasks significantly more effective. With V7, one simply has to envelop the parts of the image, and the instance mask is generated almost automatically. The time of annotating one picture varies based on the complexity of the image and number of instances, but the mean time is an estimated 2 minutes, which is significantly lower than the time for other tools such as LabelBox, according to their own website.

4.2 Distributed code

4.2.1 Mask R-CNN

The implementation of Mask RCNN is developed by Waleed Abdullah (Github user Matterport), made in 2017 for Python, Tensorflow and Keras [18]. The code is naturally outdated after version updates from Tensorflow 1.x to Tensorflow 2.x. Kamlesh Kumar contributed to the repository so that the model could support Tensorflow 2.7. Mask RCNN comes with `model.py`, which contains the model supported by either resnet101FPN or resnet50FPN backbone. This restriction makes it challenging to test other backbone architectures or changes in layers without making large changes to the code, which would

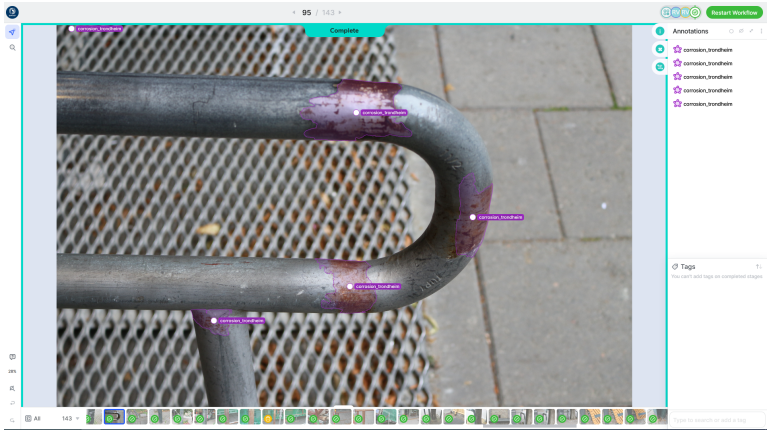


Figure 4.2: Screenshot of V7 annotation tool

be too time consuming for this project assignment. As the dataset has grown to a substantial enough size to avoid overfitting, ResNet-101 is the chosen backbone for further development and testing.

The library also comes with `utils.py` and `visualization.py` to help with assessment of performance. These helperfunctions have been used for inference, IoU computation and visualization of the predicted masks. The code for evaluating the network is in section A.

The last scripts are `config.py` and `parallel_model.py`. The config-script contains all configurations for Mask R-CNN,such as backbone, number of GPUs, optimizer and the hyperparameters discussed in section 2.3.5. The custom class `DamageConfig` overwrites the standard configurations provided by the model, as seen in section A. The script `parallel_model.py` opens up the possibility of training the network on multiple GPUs. As we only use one GPU for our training, this will not be used.

4.2.2 Custom dataset

An important aspect of training neural neural network is the correct configuration and loading of the dataset. To adapt to the corrosion dataset, Fondevik implemented the class `DamageDataset` inherited from the dataset in `utils`, which stems from the implementation by Matterport [18]. The code is mainly inspired by Dhruvil Shah [36], who showed how Mask R-CNN could be implemented for custom datasets. The class loads the images along with the ground truth mask as a binary numpy matrix, as the code for now only supports binary classification (corrosion vs non-corrosion). The class can be viewed in section A.

4.3 Training and configuration of the corrosion dataset

The specific configurations for training the corrosion dataset is given in `DamageConfig.py` and in table 4.2, all other parameters are unchanged and given in `config.py`. A batch size of 2 is chosen along with the dimensions of images between $960x960$ and $1280x1280$, given the restriction of a 12GB GPU. We have chosen to run through the full dataset per epoch for optimal training, at the expense of a slower network. Epoch size was chosen to be 80, but stopped early due to convergence.
From previous testing, the conclusion was that a good initial learning rate for the optimizer was between 0.0001 and 0.001, therefore the value has been set to 0.005. From theory it is shown that to get optimal results, learning rate decay should be used to avoid the optimizer getting stuck in local minima while also avoiding plateauing. The function `lr_scheduler` provides with learning rate decay and is therefore also being used to reduce the learning rate by 50% per tenth epoch.
The confidence threshold for object detection went from 0.7 to 0.9 to avoid a high rate of false positives. Training is done by preparing a train and val dataset and using the model's own train function, with the COCO-weights already loaded[17]. When training we save weights per epoch to use for inference, where the val dataset is then used for mask prediction and IoU computation. We can also choose any image to visualize mask predictions.

Parameter	Configuration
NAME	damage
IMAGES_PER_GPU	2
NUM_CLASSES	2
STEPS_PER_EPOCH	1632
VALIDATION_STEPS	358
DETECTION_MIN_CONFIDENCE	0.9
LEARNING_RATE	0.0005
OPTIMIZER	SGD
BACKBONE	resnet101
IMAGE_MIN_DIM	960
IMAGE_MAX_DIM	1280
MAX_GT_INSTANCES	150
RPN_CLASS_LOSS	1.0
RPN_BBOX_LOSS	1.0
LOSS_WEIGHTS	{1.0, 1.0, 1.5, 1.0, 1.0, 1.0}
RPN_ANCHOR_SCALES	{16, 64, 128, 512, 1024}

Table 4.2: configuration of the custom MASK R-CNN

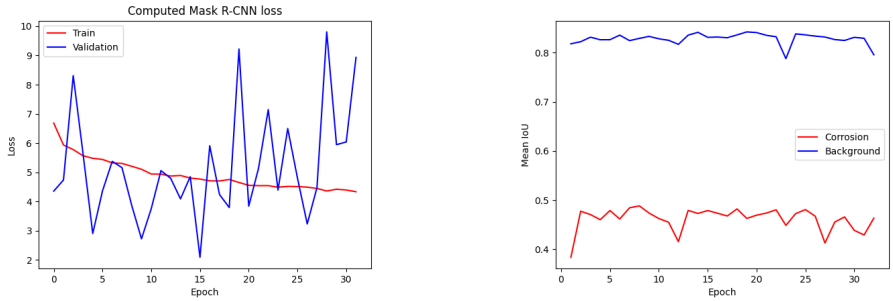
4.4 Testing

A small run has been done on the model using the dataset as we have now, with configurations in table 4.2. No changes have been done to the code other than a version update of the

MRCNN architecture. The goal is to give an understanding of the major challenges of corrosion detection with the model now before suggesting improvements. The performance is given by evaluating the validation dataset.

A small detecting test on images has also been done to a dataset of images collected by the Public Road Administration from an inspection earlier this fall on the Langøy bridge in Møre and Romsdal county fig. 4.3 [4]. Inspection has been done using controlled drone, and no quality check has been done to the images.

Figure 4.3: The Langøy bridge



(a) Train and val loss

(b) Mean IoU for the val dataset per epoch

Figure 4.4: Results from one run through the MRCNN network

4.4.1 Results

The mean IoU of the validation dataset per epoch can be seen in fig. 4.4b. It shows the corrosion IoU never coming higher than 0.5, meaning the predicted mask only covers 50% of the Ground Truth. This leaves a lot to be desired in terms of performance. Evaluating the loss from fig. 4.4a, the train loss follows a traditional asymptotically improvement as expected, but the validation loss is constantly diverging. This could be from several reasons other than bad performance, so it is more valuable to look at the mean IoU, which still shows room for improvement for the network. An interesting part to take from both fig. 4.4b and fig. 4.4a is that the network does not seem to improve much with further training. This could be a sign of overfitting to the training dataset, which we have discussed in previous section. One possible reason for the low IoU in this training compared to other research such as Fondevik [7] and RustSEG [33] is that this dataset is larger and more challenging for the model to perform on, leading to lower accuracy. Adding more diverse, high-quality data or using additional types of data augmentation could potentially improve performance.
When testing the model on unseen images of the Langøy bridge, a common issue was the detection of false positives. Figure 4.5 shows some examples of the model incorrectly classifying other areas of the image as corrosion, including high-textured objects like clouds, stones, and columns, as well as simple objects like paint stains. Figure 4.6 also illustrates instances where the model detects vastly different objects in similar images, which would not be suitable for a potential VSLAM (Visual Simultaneous Localization and Mapping) integration. In both cases, the algorithm appears to struggle with mistaking clouds for corrosion. It is worth noting that the quality of these images may be a factor, as in many cases the bridge is too far away for the camera to capture the corrosion clearly, and in Figure 4.6 the sun obscures and darkens the bridge. Ensuring proper pre-processing and quality checks of the images is an important part of image acquisition, as using low-quality images could hinder the performance of the algorithm. Additionally, if no corrosion is visible in the images, as in fig. 4.6, the algorithm should not react at all.
As previously discussed in sections 3.4.1 and 3.4.2, the results of running the model demonstrate the challenges that Mask R-CNN faces in accurately detecting corrosion. Increasing the amount of data is a simple way to improve performance, but image acquisition and annotation is a time-consuming process, even with tools like V7. Other potential solutions, such as optimizing hyperparameters and using a different backbone network, will be discussed further in chapter 5.

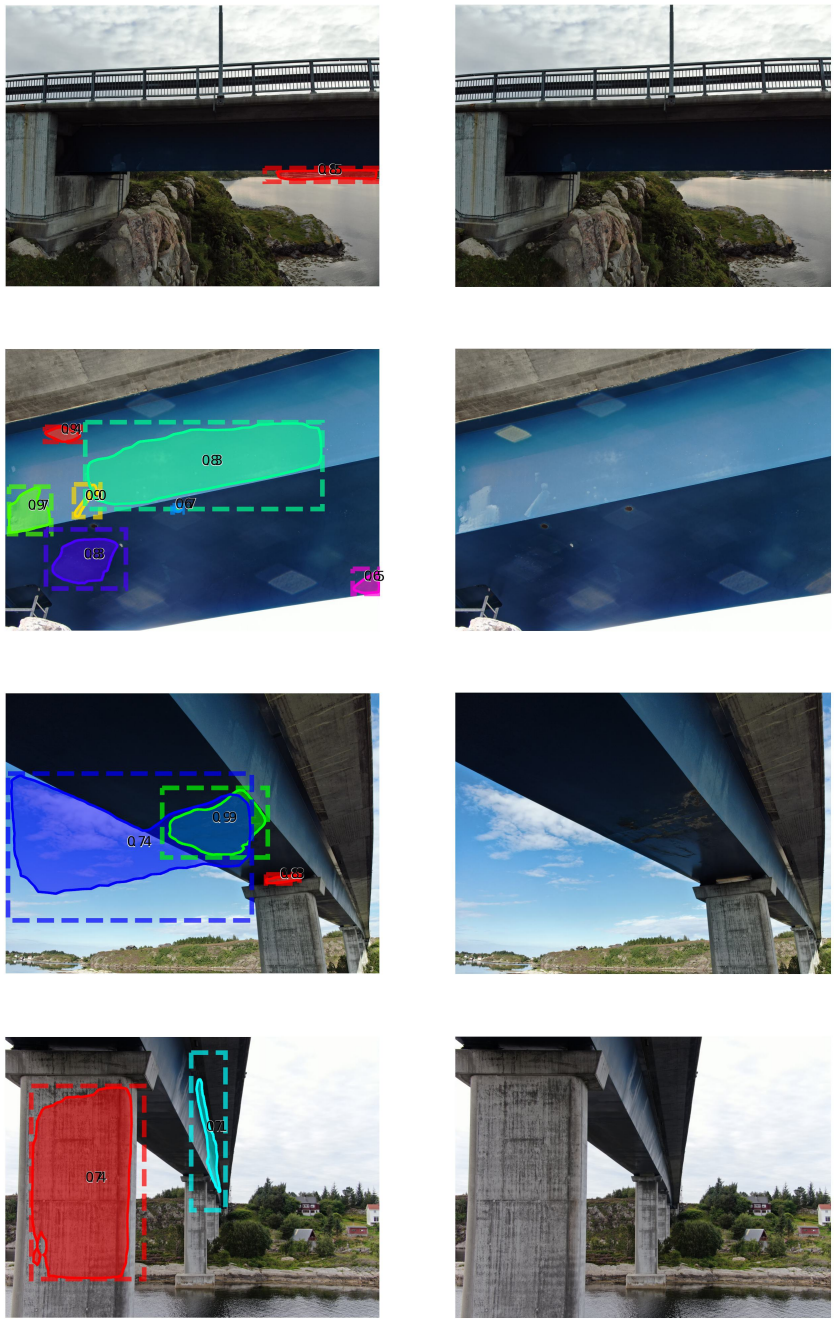


Figure 4.5: Samples of detections by the network. Prediction is shown with confidence level

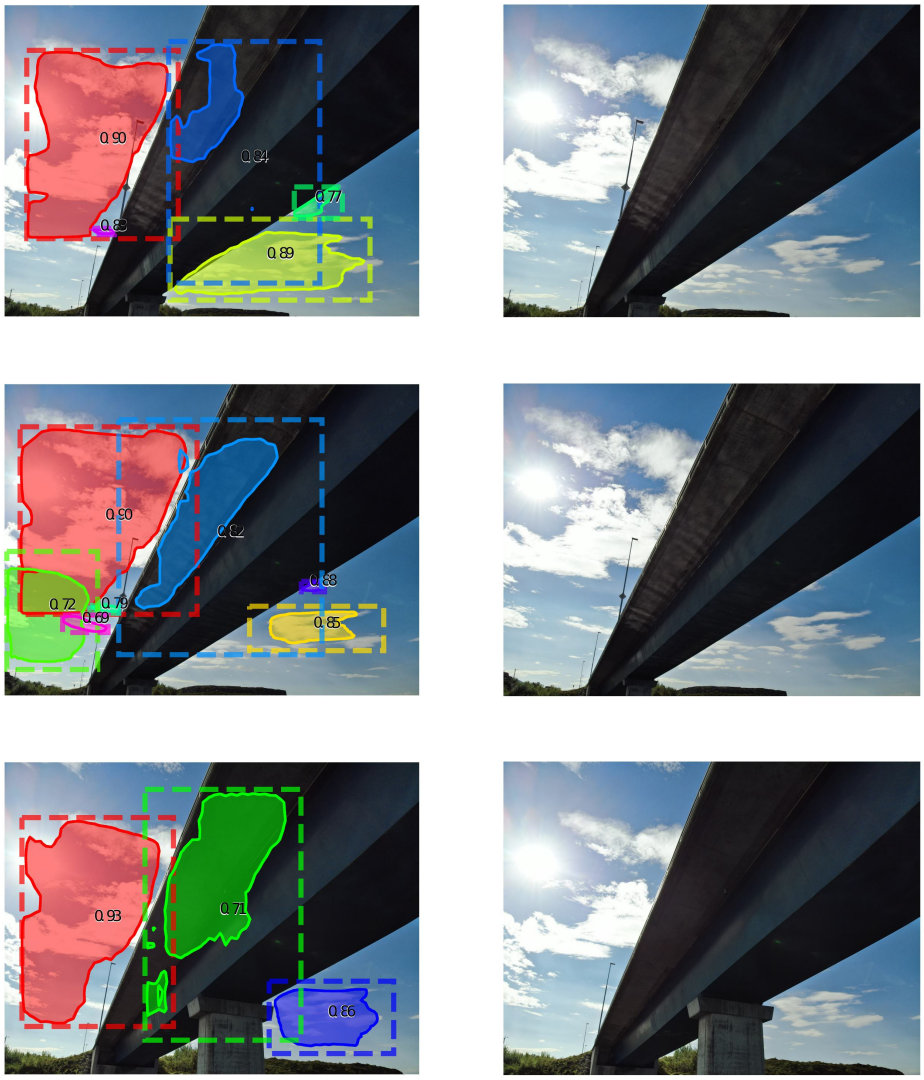


Figure 4.6: Detecting different objects from the same angle

5

Discussion and suggestions of improvements

Figures 4.5 and 4.6 show us different examples of the model failing to detect anything of importance, and how it gives us a range of false positives on the bridges. Specifically textured areas regardless of color or form is classified as corrosion, which could imply low quality feature extraction or low performing RPN.
Mask R-CNN is not without its challenges, as it is highly dependent on the ROIs being accurate to actually perform high quality segmentation, which one step algorithms like YOLO [37] are not dependent on. Seeing as Mask R-CNN is the current state-of-the-art instance segmentation algorithm, we will try to evaluate some methods of increasing performance based on recent research, before we could consider using other types of architectures like RustSEG or VGG-Unet which were discussed previously.

5.1 Refinement and tuning of hyperparameters

One problem with Mask R-CNN is the large number of parameters that needs to be tuned. The Config class presents a variety of hyper parameters that can be manually decided and is dependable on the custom dataset for which it is going to be used, as seen in section 2.3.5. Given the small research on corrosion segmentation, there are no certified intervals for the parameters that are proven to work better than others. The safest way is to manually tune the hyperparameters and test network performance, which is a tiresome process. But by evaluating the different parameters, we can have a clearer view on which ones are the most important regarding this dataset.
A study was done on quantifying common rust severity on maize leaves with Mask R-CNN, where the Genetic Algorithm [38] was examined for automated hyperparameter tuning [19]. The point of the algorithm was to effectively evaluate values for hyperparameters effectively. The algorithm created a population of individuals where each individual represents a potential problem solution. Each individual had a chromosome with multiple

genes where the number of genes in the chromosome is equal to the number of hyperparameters to be optimized. Each gene represents a different hyper-parameter in the Mask R-CNN. In the study 18 hyper-parameters were identified, and the chromosome would look like table 5.1. Each individual is then evaluated and assigned a fitness score representing on how well the solution performed. The Fitness function was given as the overall loss associated with a training run of the Mask R-CNN, where a large loss would lead to a low scoring individual.

The well-performing individuals would have genetic operations applied to them which created new individuals for the next generation. The genetic operators would be one of two

- **Mutation**: Genes are split into 6 subsets, random boolean value is generated for each subset. If true, all genes in that subset are set to have new random values within the bounds. If false. all genes stay unchanged.

- **Crossover**: Two parents are individually selected from existing population. The offspring has its chromosome traversed, at each gene a bool is generated. If true, the offspring gets the gene value from parent 1, if false, parent 2.

The paper initialized a population of 12 with randomly generated values for genes given a certain interval, with a mutation rate of 50% and crossover rate of 50%. This was chosen empirically to make sure the search space was covered, and to guarantee convergence within 30 generations without the algorithm stagnating. The resulting Mask R-CNN perform the same as a manually tuned model or better, with vastly less time used, meaning there is a clear case for using automated hyperparameter tuning.

A lot of the principles that the article discusses, can be applied in this problem. As this describes a general algorithm that can be applied to better the Mask R-CNN network given a particular dataset, we can generalize the objective enough to apply it in our Mask R-CNN model for infrastructure damages. The results are promising, all though it requires substantial computational power to converge properly. It is an idea to also use the Genetic algorithm to better adapt the network to certain bridges or datasets, making it better equipped for large alterations. As everything is done automatically, it does not require expertise knowledge about tuning and would make the network easy accessible for inspectors to improve on their own.

A note is to evaluate the fitness landscape, which the article also recommends, as it is uncertain whether the loss function is the best use for the fitness function in the Genetic Algorithm. As seen in figure (loss function) the loss can be drastically higher while the IOU performance is good.

Hyperparameter tuning in of itself is not enough to properly improve the network, but a nice thing about it is that it probably can't make the network worse. Determining a lower maximum number for region proposals or detections could possibly lower the risk of false positives as we see in fig. 4.5.

RPN_ANCHOR_STRIDE	2
RPN_NMS_THRESHOLD	0.89
RPN_TRAIN_ANCHORS_PER_IMAGE	150
PRE_NMS_LIMIT	5980
POST_NMS_ROIS_TRAINING	1100
POST_NMS_ROIS_INFERENCE	1568
MEAN_PIXEL	[127, 126.45, 134.45]
DETECTION_NMS_THRESHOLD	0.7
DETECTION_MAX_INSTANCES	100
DETECTION_MIN_CONFIDENCE	0.9
LEARNING_RATE	0.0005
ROI_POSITIVE_RATIO	0.4
LEARNING_MOMENTUM	0.85
WEIGHT_DECAY	0.0002
GRADIENT_CLIP_NORM	4.0
MAX_GT_DISTANCES	150
LOSS_WEIGHTS	1.5, 1.0, 1.0, 1.0

Table 5.1: An example of an individual's chromosome in GA algorithm

5.2 Improving backbone network

Quality object detection is determined by the quality in the feature extraction, making the design of the backbone network crucial. As mentioned in section 5.2, we add a Feature Pyramid Network to increase resolution so that small objects, such as corrosion stains, can be detected. The Mask R-CNN network to detect damages in fishnets [30] implements an improvement of the FPN structure by using a Recursive Feature Pyramid (RFP) instead. Inspired by the human thinking of *looking and thinking twice*, RFP incorparates feedback connections into the bottom-up backbone of FPN, extending it to a two-step sequential network, as visualized in fig. 5.1, wherein the network is run again, but merged together with the results from the previous iteration.
The final step is to fuse together the two feature maps together, and with the RFP-incorporated feedback connections containing the gradient signals of classification and regression at the previous iteration, it is possible to update the backbone parameters directly.
The paper also incorporates a Deformable Convolution Network (DCN) to deal with irregular mesh holes in the fishnets, and it is reason to believe that we can relate this to the irregular patterns of corrosion. By replacing the last ResNet block with a DCN structure, the researches believed it could improve the efficiency and accuracy of feature extraction. The goal is to create a convolution kernel more adapted to the formation of objects.
It is difficult to evaluate how well these changes will perform for the corrosion dataset without testing it. There is a clear need for improving feature extraction in order for the network to distinguish between corrosion and other textured areas in the image, but the changes applied here have a huge impact on the runtime of the network, as the backbone runs on the same image twice, and a copy of the feature map must also be saved for each

propagation. It is worth considering applying these changes, but it is not realizable if the changes doesn't lead to major improvements at the cost of computational complexity.

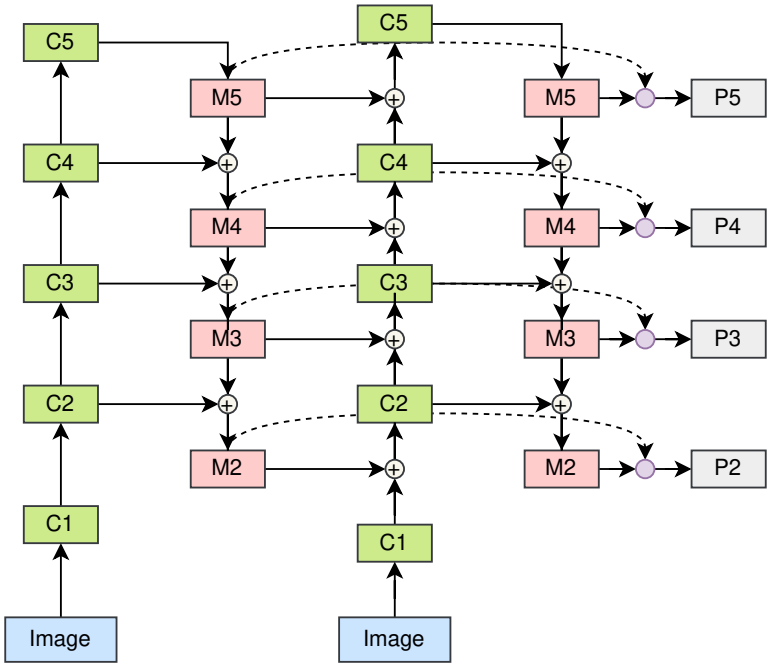


Figure 5.1: An example of Recursive Feature Pyramid

5.3 Two stage neural networks

As seen in fig. 4.5 and fig. 4.6, the algorithm struggles when faced with images with a lot of objects in the background, such as nature, clouds and sunlight. We previously discussed the possibility of simply broadening the dataset with noisy images, lowering the network's confidence rate in detecting objects. Another possibility is rather than training one network to keep track of a large task, we create separate networks responsible for smaller subtasks, making it simpler for each network to adapt to their given goal. While using Mask R-CNN for the actual corrosion detection and segmentation, there could be other smaller networks used to support its assignment by doing other tasks. While there are multiple solution to what these networks could do and their architecture, we have chosen to focus on a two stage pipeline where a second network is tasked with cropping out or removing background.

A suggestion for the implementation of this two stage network is to use a Faster R-CNN network for detecting the bridge, in the use of cropping out any background, before sending the bounding boxes to the network responsible for segmenting corrosion. A similar study have been done in [39] related to segmenting the Optic Nerve Head, where a first stage Mask R-CNN produces cropped RoIs that are sent to a second stage Mask R-CNN

for further segmentation. Here both networks are trained to locate the optic nerve head, while we would train the first stage to look specifically for bridges, as they are easier to detect for a region proposal network due to their contrasted structure in comparison to the nature around. In fig. 5.2, there is a suggestion of how the pipeline would look like.

This vision could be extended in the use of an automated drone inspection, and aid the UAV in taking better pictures. In order for the drone to localize suitable places to inspect for corrosion, it needs to have the overview of the type and size of bridge. Using multilevel networks to first classify the bridge, the drone can place itself closer the relevant parts of the bridge, removing unqualified images too far away or with plenty of nature like the images in fig. 4.5 show.

Yu and Nishio [40] created a bridge inspection algorithm compromised of three neural networks, a ResNet50 for bridge type classification, a YoloV3 [37] for component detection, and Mask R-CNN for segmentation. All though showing promise, additional networks require extra work. This means the time consuming labour of collecting high-quality data and labeling it, as well as looking at optimization techniques for each separate network. This would also require much higher computational complexity, making it harder for real-time detection with a drone.

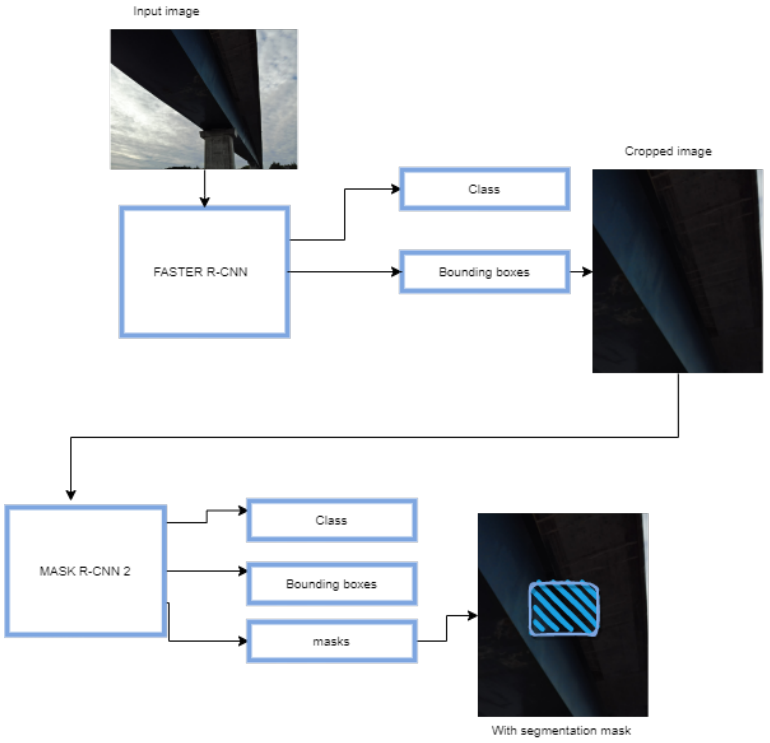


Figure 5.2: An example of two stage network

5.4 Other suggestions

This section is tied to other suggestions of improvement that haven't been fully considered, but can be researched further. They have little work related to them, and are mostly brainstormed between the author, supervisors and other students.

5.4.1 Transfer learning

The Matterport version of Mask R-CNN [18] uses the COCO dataset for pre-training the network. All though a good general starting point for any network, it is worth considering that the subject of corrosion is vastly different from objects like cats, bicycles and pizzas. It could be useful to use other datasets for pretraining, such as medical images of tumors and organs, due to their deformative and unusual structure. There does not exist any large scale dataset on medical images on par with the COCO dataset, but with the rapidly developing machine learning techniques in medical imaging, the datasets like Br35H by Ahmed Hamada [41] are also being developed. Medical images can however be vastly different from images of corrosion, as they are colorless and often without a lot of disturbances, including that medical images are often taken from the same angle. It is therefore also worth testing other datasets for pre-training, for example of other similar damages.

5.4.2 Multi-class detection

Fondevik suggested for further work to introduce multiple damage classes in the network, examples could be white corrosion, cracks, paint flaking or heavy/light corrosion. This could help alleviate some of the work of detecting the specified corrosion, as the network can specify the patterns more into each class. This does again require further data acquisition to get an evenly weighted dataset for each class, which can be challenging due to brown corrosion vastly outnumbering white corrosion in terms of instances. Pictures of paint flaking are also rare. A suggestion could be to perform data augmentation on the rarer classes, but without diverse enough images, we can face the same problems like discussed in section 3.2. This is an improvement that can be evaluated further with the acquisition of more data. An short-term idea is to still use multiple classes with the small dataset, but still only consider the performance of brown corrosion. This way avoids further disturbances by white corrosion for the network to learn.

5.4.3 Increase efficiency and training time

An additional task which is always relevant, especially for real time segmentation, is how to increase efficiency in training and validation. This can be highly applicable if one chooses to implement section 5.2 or section 5.3, or the dataset is vastly increased. Zimmermann and Siems proposes a method for faster training of Mask R-CNN by adding a network head known as the Edge Agreement Head [42]. This head would use classical edge detection filters on the instance masks to calculate the loss between the predicted and ground truth mask contours. The point was to create sharper edges for the predicted mask and increase speed and performance early on in the training procedure. This is a small improvement which could be added on in later processes.

5.4.4 Automatic image annotation

We have mentioned before that one of the challenges for collecting new data and introducing new classes was the work of image annotation. A solution could be even more automated image annotation than V7 [35]. The paper from [32] presents an image labeling tool for unsupervised image segmentation based on texture and RGB feature-based classifier optimization. This means that a large portion of images could be automatically annotated from just a small set of samples, with some time spent on verifying the annotations.

6

Conclusion and further work

6.1 Conclusion

In this thesis, we addressed the task of corrosion detection and the challenges involved, and provided suggestions for improving the performance of the network in chapter 5. Despite the complex structure and varied results of Mask R-CNN, it is still the most suitable network for this task, and we will continue to work with it. However, if performance does not improve, it may be necessary to consider alternative networks. The wide range of corrosion development and bridge images can present significant challenges as more data is collected and tested, as seen in the results presented in chapter 4. These results also highlight the need for improvements to the algorithm in order for it to be effectively used in the field.

Based on this thesis, the following conclusions can be drawn:

6.1.1 Dataset

The quality of the dataset has a significant impact on the performance of a large network like Mask R-CNN. In conclusion, while the current dataset contains useful information, there is still a need for more data that represents the diverse environments in which corrosion can occur on bridge infrastructure. Pre-processing the images to remove distractions such as sunlight and to ensure that the bridges are captured clearly is also important.

Continuous gathering of high-quality corrosion data is crucial for the future of this network. This could involve data from different bridges, different types of corrosion, or different types of damage. As this work is both time-consuming and always improvable, the current dataset may not be complete enough for a master's thesis by spring 2023. To address this issue in the short term, we suggest using data augmentation techniques such as rotation and flipping, and cropping the images to increase the size of the dataset by a factor of ten.

Other suggestions for improving the data quality include implementing specific routes for the UAVs or using a second neural network for bridge detection or background removal.

These approaches may be more applicable in the future when the network is more mature and ready for integration with VSLAM.

6.1.2 Implementation

The results in fig. 4.4 shows sub-optimal performance, with an IoU of only 50% and no improvement overall for the 30 epochs. It is difficult to evaluate where the inefficiency comes from, but research suggests poor feature extraction and poor hyperparameter tuning, in addition to a faulty dataset. The recent research suggests using the Genetic Algorithm for automatic hyperparameter tuning, making the time-consuming work of manual tuning way more efficient and easier. To get a better feature extraction backbone, suggestions like adding a Deformable Convolutional Network and a Recursive Feature Pyramid have also increased perfomance for some related issues.
These suggestions have not been tested, so it is difficult to evaluate how well these changes will perform on our current dataset. Many of these changes are however possible to implement independently of each other, along with continuous work of gathering and annotating new data. Specifically automated hyperparameter tuning and backbone improvement are changes that should be easy to implement and test. The suggestions discussed in section 5.4 can be further researches for future work.
The potential and societal need for an automatic segmentation algorithm is present, and with the skyrocketing amount of research done on Mask R-CNN and other segmentation algorithms, improvement of the network is imminent.

6.2 Further work

The whole of chapter 5 is a discussion of potential further work. Shortly summarized, future work should be focused on gathering data, create good data augmentation such as cropping algorithms in order for the training to be optimized, and then test any of the suggestion in chapter 5. When performance has improved, one could start the work of fulfilling the goal of an fully automated inspection drone by integrating the algorithm with VSLAM.

6.3 Delimitations

The training and testing of neural networks is a complex assignment subject to many pitfalls. This is especially relevant given the age of the code and the lack of documentation. The goal was to develop a more comprehensive understanding of the hyperparameters and introduce them here, and to have more results to compare. The lack of training has been due to multiple issues regarding version control, 0% accuracy and increasing loss function. The problems probably stem from the attempted upgrade to Tensorflow 2.7, which will be investigated further in the master's thesis. The results presented was therefore only based around one training of 32 epochs.

Bibliography

[1] Jozef Gocál and Jaroslav Odrobiňák. On the influence of corrosion on the load-carrying capacity of old riveted bridges. *Materials*, 13(3):717, Feb 2020. ISSN 1996-1944. doi: 10.3390/ma13030717. URL http://dx.doi.org/10.3390/ma13030717.

[2] Vivi Yang. Final report released on nanfangao sea-crossing bridge collapse. *Taiwan Transportation Safety Board*. URL https://www.ttsb.gov.tw/english/16051/16113/16114/28249/post.

[3] Gerhardus Koch, Jeff Varney, Neil Thompson, Oliver Moghissi, Melissa Gould, and Joe Payer. International measures of prevention, application, and economics of corrosion technologies study. Technical report, NACE International.

[4] Norwegian public roads administration. URL https://www.vegvesen.no/en/?lang=en.

[5] Dimensions ai. URL https://www.dimensions.ai/.

[6] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn, 2017. URL https://arxiv.org/abs/1703.06870.

[7] Simen Keiland Fondevik. Image segmentation of corrosion damages in industrial inspections using state-of-the-art neural networks. 2020. URL https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2780881?locale-attribute=no.

[8] Billie F. Spencer, Vedhus Hoskere, and Yasutaka Narazaki. Advances in computer vision-based civil infrastructure inspection and monitoring. *Engineering*, 5 (2):199–222, 2019. ISSN 2095-8099. doi: https://doi.org/10.1016/j.eng.2018.11.030. URL https://www.sciencedirect.com/science/article/pii/S2095809918308130.

[9] Deepa Berchmans and S S Kumar. *Optical character recognition: An overview and an insight*. 2014. doi: 10.1109/ICCICCT.2014.6993174.

[10] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks, 2015. URL https://arxiv.org/abs/1511.08458.

[11] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989. doi: 10.1162/neco.1989.1.4.541.

[12] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. URL https://arxiv.org/abs/1506.01497.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

[14] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection, 2016. URL https://arxiv.org/abs/1612.03144.

[15] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[16] Mark Everingham, Luc Van Gool, Christopher K. I. Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. page 303–338, 2010. doi: https://doi.org/10.1007/s11263-009-0275-4.

[17] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2014. URL https://arxiv.org/abs/1405.0312.

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

[19] Mia Gerber, Nelishia Pillay, Katerina Holan, Steven A. Whitham, and Dave K. Berger. Automated hyper-parameter tuning of a mask r-cnn for quantifying common rust severity in maize, 2021.

[20] Feng Jiang, Ling Ma, Tim Broyd, and Ke Chen. Digital twin and its implementations in the civil engineering sector. *Automation in Construction*, 130: 103838, 2021. ISSN 0926-5805. doi: https://doi.org/10.1016/j.autcon.2021.103838. URL https://www.sciencedirect.com/science/article/pii/S0926580521002892.

[21] Industrial internet of things: Unleashing the potential of connected products and services. Technical report, World Economic Forum, January 2015. URL https://www3.weforum.org/docs/WEFUSA_IndustrialInternet_Report2015.pdf.

[22] Mayank Mishra, Paulo B. Lourenço, and G.V. Ramana. Structural health monitoring of civil engineering structures by using the internet of things: A review. *Journal of Building Engineering*, 48:103954, 2022. ISSN 2352-7102. doi: https://doi.org/10.1016/j.jobe.2021.103954. URL https://www.sciencedirect.com/science/article/pii/S235271022101812X.

[23] Sindhu Ghanta, Tanja Karp, and Sangwook Lee. Wavelet domain detection of rust in steel bridge images. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1033–1036, 2011. doi: 10.1109/ICASSP.2011.5946583.

[24] Ahmed Mahmoud and Mohamed Atia. Improved visual slam using semantic segmentation and layout estimation. *Robotics*, 11(5):91, Sep 2022. ISSN 2218-6581. doi: 10.3390/robotics11050091. URL http://dx.doi.org/10.3390/robotics11050091.

[25] T. Padma, Ch Usha Kumari, Dommeti Yamini, Kapilavai Pravallika, Konduru Bhargavi, and Mula Nithya. Image segmentation using mask r-cnn for tumor detection from medical images. In *2022 International Conference on Electronics and Renewable Systems (ICEARS)*, pages 1015–1021, 2022. doi: 10.1109/ICEARS53579.2022.9751891.

[26] Allen Zhang, Kelvin C. P. Wang, Baoxian Li, Enhui Yang, Xianxing Dai, Yi Peng, Yue Fei, Yang Liu, Joshua Q. Li, and Cheng Chen. Automated pixel-level pavement crack detection on 3d asphalt surfaces using a deep-learning network. *Computer-Aided Civil and Infrastructure Engineering*, 32(10):805–819, 2017. doi: https://doi.org/10.1111/mice.12297. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/mice.12297.

[27] Xiangyang Xu, Mian Zhao, Peixin Shi, Ruiqi Ren, Xuhui He, Xiaojun Wei, and Hao Yang. Crack detection and comparison study based on faster r-cnn and mask r-cnn. *Sensors*, 22(3), 2022. ISSN 1424-8220. doi: 10.3390/s22031215. URL https://www.mdpi.com/1424-8220/22/3/1215.

[28] Soufiane Bouarfa, Anil Doğru, Ridwan Arizar, Reyhan Aydoğan, and Joselito Serafico. *Towards Automated Aircraft Maintenance Inspection. A use case of detecting aircraft dents using Mask R-CNN*. doi: 10.2514/6.2020-0389. URL https://arc.aiaa.org/doi/abs/10.2514/6.2020-0389.

[29] Qinghui Zhang, Xianing Chang, and Shanfeng Bian Bian. Vehicle-damage-detection segmentation algorithm based on improved mask rcnn. *IEEE Access*, 8:6997–7004, 2020. doi: 10.1109/ACCESS.2020.2964055.

[30] Ziliang Zhang, Fukun Gui, Xiaoyu Qu, and Dejun Feng. Netting damage detection for marine aquaculture facilities based on improved mask r-cnn. *Journal of Marine Science and Engineering*, 10(7):996, Jul 2022. ISSN 2077-1312. doi: 10.3390/jmse10070996. URL http://dx.doi.org/10.3390/jmse10070996.

[31] Jiyuan Shi, Ji Dang, Mida Cui, Rongzhi Zuo, Kazuhiro Shimizu, Akira Tsunoda, and Yasuhiro Suzuki. Improvement of damage segmentation based on pixel-level data balance using vgg-unet. *Applied Sciences*, 11(2), 2021. ISSN 2076-3417. doi: 10.3390/app11020518. URL https://www.mdpi.com/2076-3417/11/2/518.

[32] Atiqur Rahman, Zheng Yi Wu, and Rony Kalfarisi. Semantic deep learning integrated with rgb feature-based rule optimization for facility surface corrosion detection and evaluation. *Journal of Computing in Civil Engineering*, 35(6):04021018, 2021. doi: 10.1061/(ASCE)CP.1943-5487.0000982. URL https://ascelibrary.org/doi/abs/10.1061/%28ASCE%29CP.1943-5487.0000982.

[33] B. Burton, W. T. Nash, and N. Birbilis. Rustseg – automated segmentation of corrosion using deep learning, 2022. URL https://arxiv.org/abs/2205.05426.

[34] Deegan J Atha and Mohammad R Jahanshahi. Evaluation of deep learning approaches based on convolutional neural networks for corrosion detection. *Structural Health Monitoring*, 17(5):1110–1128, 2018. doi: 10.1177/1475921717737051. URL https://doi.org/10.1177/1475921717737051.

[35] Darwin from v7 labs. URL https://darwin.v7labs.com.

[36] Dhruvil Shah. Mask r-cnn on a custom dataset! https://github.com/jackfrost1411/MaskRCNN, 2020.

[37] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement, 2018. URL https://arxiv.org/abs/1804.02767.

[38] Annu Lambora, Kunal Gupta, and Kriti Chopra. Genetic algorithm- a literature review. In *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*, pages 380–384, 2019. doi: 10.1109/COMITCon.2019.8862255.

[39] Haidar Almubarak, Yakoub Bazi, and Naif Alajlan. Two-stage mask-rcnn approach for detecting and segmenting the optic nerve head, optic disc, and optic cup in fundus images. *Applied Sciences*, 10(11):3833, May 2020. ISSN 2076-3417. doi: 10.3390/app10113833. URL http://dx.doi.org/10.3390/app10113833.

[40] Weilei Yu and Mayuko Nishio. Multilevel structural components detection and segmentation toward computer vision-based bridge inspection. *Sensors*, 22(9), 2022. ISSN 1424-8220. doi: 10.3390/s22093502. URL https://www.mdpi.com/1424-8220/22/9/3502.

[41] Ahmed Hamada. Brain tumor detection 2020. URL https://www.kaggle.com/datasets/ahmedhamada0/brain-tumor-detection.

[42] Roland S. Zimmermann and Julien N. Siems. Faster training of mask r-CNN by focusing on instance boundaries. *Computer Vision and Image Understanding*, 188: 102795, nov 2019. doi: 10.1016/j.cviu.2019.102795. URL https://doi.org/10.1016%2Fj.cviu.2019.102795.

Appendix

A Distributed code

Custom dataset and training

```

def aug_flipping():
    return iaa.Sequential([iaa.Fliplr(0.5), iaa.Flipud(0.5), ])
    # horizontally f l i p 50\% of all images# vertically f l i p 50\% of all images


class DamageConfig(Config):
    """Configuration for training on the toy dataset .
    Derives from the base Config class and overrides some values .
    """
    # Give the configuration a recognizable name
    NAME = 'damage'
    # We use a GPU with 12GB memory, which can fit two images .
    # Adjust down if you use a smaller GPU.
    IMAGES_PER_GPU = 2
    # Number of classes ( including background)
    NUM_CLASSES = 1 + 1  # +1 for background
    # Number of training steps per epochDataset
    STEPS_PER_EPOCH = 1632  # 508
    VALIDATION_STEPS = 358  # 50
    # Skip detections with < 90% confidence
    DETECTION_MIN_CONFIDENCE = 0.9
    LEARNING_RATE = 0.0005  # 0.001 seems too big, 0.000001 is too small
    OPTIMIZER = "SGD"  # default is SGD
    # Reducing memory
    BACKBONE = "resnet101"
    IMAGE_MIN_DIM = 960  #
    IMAGE_MAX_DIM = 1280
    MAX_GT_INSTANCES = 150
    LOSS_WEIGHTS = {
        "rpn_class_loss": 3,
        "rpn_bbox_loss": 1.,
        "mrcnn_class_loss": 3.,
        "mrcnn_bbox_loss": 1.,
        "mrcnn_mask_loss": 1.
    }
    RPN_ANCHOR_SCALES = (16, 64, 128, 512, 1024)  # can only select 5 anchor scale
    # RPN_ANCHOR_RATIOS = [0.25, 0.5, 1, 2, 4]


############################################################
# Dataset
############################################################
class DamageDataset(utils.Dataset):
    def load_damage(self, dataset_dir, subset):
        """Load a subset of the damage dataset .
        dataset_dir : Root directory of the dataset .
        subset : Subset to load : train or val
        """
        # Add classes.
        self.add_class('damage', 1, 'corrosion')
        # +1 since 0=BG
        # Train or validation dataset
        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_example_dir = os.path.join(dataset_dir, image_id)
            (_, _, file_names) = next(os.walk(image_example_dir))
            file_name = file_names[0]
            image_path = os.path.join(image_example_dir, file_name)
            image = skimage.io.imread(image_path)
            height, width = image.shape[:2]
            self.add_image(
                'damage',
                image_id=image_id,  # use ids from LabeBox
                path=image_path,
                width=width, height=height)

    def load_mask(self, image_id):
        """Generate instance masks for an image .
        Returns :
        masks: A bool array of shape [height , width , instance count] with
        one mask per instance .
        class_ids : a 1D array of class IDs of the instance masks .
        """
        info = self.image_info[image_id]
        # Get mask directory from image path
        mask_dir = os.path.join(os.path.dirname(info['path']), 'masks')
        # Read mask files from .png images
        mask = []
        for f in next(os.walk(mask_dir))[2]:
            if f.endswith('.png') and ('corrosion' or 'grov_merking' in f):
                m = skimage.io.imread(os.path.join(mask_dir, f))
                if len(m.shape) > 2:
                    m = m[:, :, 0]
                m = m.astype(bool)
                # as_gray=True).astype(bool)
                mask.append(m)
        mask = np.stack(mask, axis=-1)
        class_ids = np.ones([mask.shape[-1]], dtype=np.int32)
        # Return mask, and array of class IDs of each instance .
        return mask.astype(bool), class_ids,

    def load_mask_semantic(self, image_id):
        """Generate instance masks for an image .
        Returns :
        masks: A bool array of shape [height , width , instance count] with
        one mask per instance .
        class_ids : a 1D array of class IDs of the instance masks .
        """
        info = self.image_info[image_id]
        # Get mask directory from image path
        mask_dir = os.path.join(os.path.dirname(info['path']), "masks")
        # Read mask f i l e s from .png images
        mask = None
        mask_initialized = False
        for f in next(os.walk(mask_dir))[2]:
            if f.endswith(".png"):
                if not mask_initialized:
                    mask = skimage.io.imread(os.path.join(mask_dir, f),
                                             as_gray=True).astype(np.bool)
                    mask_initialized = True
                else:
                    m = skimage.io.imread(os.path.join(mask_dir, f),
                                          as_gray=True).astype(np.bool)
                    mask = np.add(mask, m)
        mask = [mask]
        mask = np.stack(mask, axis=-1)
        class_ids = np.ones([mask.shape[-1]], dtype=np.int32)
        # Return mask, and array of class IDs of each instance .
        return mask.astype(np.bool), class_ids
```

```
    def image_reference(self, image_id):
        """Return the path of the image . """
        info = self.image_info[image_id]
        if info[" source "] == " damage ":
            return info["path"]
        else:
            super(self.__class__, self).image_reference(image_id)


def lr_scheduler(epoch, lr):
    decay_rate = 0.5
    decay_step = 10
    print(epoch, lr)
    if int(epoch) % decay_step == 0 and epoch != 0:
        print(lr * decay_rate)
        return float(lr * decay_rate)
    else:
        print(lr)
        return float(lr)


def train(model):
    """Train the model."""
    dataset_train = DamageDataset()
    dataset_train.load_damage(ROOT_DIR, 'train')
    dataset_train.prepare()

    # Validation dataset
    dataset_val = DamageDataset()
    dataset_val.load_damage(ROOT_DIR, 'val')
    dataset_val.prepare()
    # Custom callbacks
    change_lr = LearningRateScheduler(lr_scheduler, verbose=1)
    print(" Training network heads")
    print(model.config.LEARNING_RATE)
    model.train(dataset_train, dataset_val,
                learning_rate=model.config.LEARNING_RATE,
                epochs=80,
                custom_callbacks=[change_lr],
                augmentation=aug_flipping(),
                layers='all')
```

Inference and Visualization

```
local_class_colors = [(0, 0, 0), (0, 0, 255)]
mask_rcnn_colors = local_class_colors
def apply_inference(model, image_path=None):
    # Load image
    print(image_path)
    image = skimage.io.imread(image_path)
    # Run detection
    results = model.detect([image], verbose=1)
    # Visualize results
    r = results[0]
    class_names = ["BG", "corrosion"]
    visualize.display_instances(image, r['rois'], r['masks'],
                                r['class_ids'],
                                class_names, r['scores'])

def evaluate_model(model, dataset_val):
    dataset_val.load_damage(ROOT_DIR, "val")
    dataset_val.prepare()
    image_ids = dataset_val.image_ids
    iou_corr_list = []
    iou_bg_list = []
    for image_id in image_ids:
```

```
        image = dataset_val.load_image(image_id)
        mask_gt, class_ids = dataset_val.load_mask(image_id)
        mask_gt = combine_masks_to_one(mask_gt)
        result = model.detect([image], verbose=0)[0]
        #frame = Image.open(image_path).convert('RGB')
        frame = np.array(image)
        CLASS_NAMES_MASKRCNN = ['background', 'corrosion']
        r = result
        fig, (ax1, ax2) = plt.subplots(1, 2)

        visualize.display_instances(frame, r['rois'],
                                    r['masks'], r['class_ids'],
                                    CLASS_NAMES_MASKRCNN, r['scores'],
                                    figAx=(fig, ax1) ,show_caption=False)
        predicted_masks = result["masks"]
        if predicted_masks.shape[-1] == 0:
            continue
        mask_pred = combine_masks_to_one(predicted_masks)
        iou_corr = compute_overlaps_masks(mask_gt, mask_pred)[0][0]
        iou_bg = compute_overlaps_masks(mask_gt, mask_pred, BG=True)[0][0]
        print(image_id, "IoU =", (iou_corr, iou_bg))
        iou_corr_list.append(iou_corr)
        iou_bg_list.append(iou_bg)
    mean_corr_iou = sum(iou_corr_list) / len(iou_corr_list)
    mean_bg_iou = sum(iou_bg_list) / len(iou_bg_list)
    print("Total mean values ")
    print(" Corrosion IoU =", mean_corr_iou)
    print("BG IoU=", mean_bg_iou)
    print("Mean IoU =", (mean_corr_iou + mean_bg_iou) / 2)

def combine_masks_to_one(masks):
    combined_mask = masks[:, :, 0]
    for i in range(masks.shape[-1]):
        combined_mask += masks[:, :, i]
    return np.expand_dims(combined_mask, 2)

def compute_overlaps_masks(masks1, masks2, BG=False):
    """Computes IoU overlaps between two sets of masks .
    masks1, masks2: [Height , Width , instances ]
    """
    # If either set of masks is empty return empty result
    if masks1.shape[-1] == 0 or masks2.shape[-1] == 0:
        return np.zeros((masks1.shape[-1], masks2.shape[-1]))
    # f l a t t e n masks and compute their areas
    if BG:
        masks1 = np.reshape(masks1 < .5,
                            (-1, masks1.shape[-1])).astype(np.float32)
        masks2 = np.reshape(masks2 < .5,
                            (-1, masks2.shape[-1])).astype(np.float32)
    else:
        masks1 = np.reshape(masks1 > .5,
                            (-1, masks1.shape[-1])).astype(np.float32)
        masks2 = np.reshape(masks2 > .5,
                            (-1, masks2.shape[-1])).astype(np.float32)
    area1 = np.sum(masks1, axis=0)
    area2 = np.sum(masks2, axis=0)

    # intersections and union
    intersections = np.dot(masks1.T, masks2)
    union = area1[:, None] + area2[None, :] - intersections
    overlaps = intersections / union

    return overlaps

def overlay_prediction_single_maskrcnn(pr=None, inp=None,
                                       out_dir=None,
                                       overlay_img=True,
                                       colors=mask_rcnn_colors):
```

```
    if isinstance(pr, six.string_types):
        pr = cv2.imread(pr, 0)
    if isinstance(inp, six.string_types):
        out_fname = os.path.join(out_dir, os.path.basename(inp))
        inp = cv2.imread(inp[:-4] + ".jpg")

    assert len(inp.shape) == 3, "Image should be h,w,3 "
    seg_img = visualize_segmentation(pr, inp, colors=colors,
                                     overlay_img=overlay_img, )
    if out_fname is not None:
        cv2.imwrite(out_fname, seg_img)
    return pr

def overlay_predictions_all_maskrcnn(pr_dir=None, inp_dir=None,
                                     out_dir=None,
                                     overlay_img=True, colors=mask_rcnn_colors):
    for f in next(os.walk(pr_dir))[2]:
        print("File name =", f)
        if " DS_Store " in f:
            print(" Skipping DS_Store file")
        continue
        if f.endswith(".png"):
            pr = os.path.join(pr_dir, f)
            inp = os.path.join(inp_dir, f)
            overlay_prediction_single_maskrcnn(pr, inp,
                                               out_dir, overlay_img, colors)
```