Aalborg University
Software Engineering

AALBORG UNIVERSITET

Title:
Mask R-CNN for Segmentation of Aerial Data with Edge Aware Loss

Project-period:
2020 Spring Semester

Project Group:
DT102F20

Group Members:
Frederik Østerby Hansen
Mikkel Vestergaard Hem
Matias Dahlin Holst

Supervisor:
Chenjuan Guo

Page Count: 8

Abstract: In this project we tackle the problem of image segmentation in aerial data, where the goal is to draw accurate segmentation masks unto buildings. We utilize an existing machine learning model, Mask R-CNN, and attempt to optimize the mask loss function, such as to improve the accuracy of the segmentation masks. We therefore propose and implement three different loss functions for the model, to measure their effects on performance, specifically when applied on our dataset. We also produce a dataset, derived from aerial photographical data and LiDAR data. We find that the Edge Aware loss function results in a noteworthy improvement in output masks.

Mask R-CNN for Segmentation of Aerial Data with Edge Aware Loss

Hansen, Frederik Østerby , **Hem, Mikkel Vestergaard** and **Holst, Matias Dahlin**
Aalborg University
faha15@student.aau.dk, mhem15@student.aau.dk, mholst15@student.aau.dk

Abstract

In this project we tackle the problem of image segmentation in aerial data, where the goal is to draw accurate segmentation masks unto buildings. We utilize an existing machine learning model, Mask R-CNN, and attempt to optimize the mask loss function, such as to improve the accuracy of the segmentation masks. We therefore propose and implement three different loss functions for the model, to measure their effects on performance, specifically when applied on our dataset. We also produce a dataset, derived from aerial photographical data and LiDAR data. We find that the Edge Aware loss function results in a noteworthy improvement in output masks.

1 Introduction

The process of extracting information from images has gained a lot of attention in the previous decade, as it has a wide range of applications, depending on the given type of image. To perform this task, neural networks are a popular choice, as they have a large collection of tools, and can be trained to perform a variety of tasks. Such tasks could be *image classification*, labeling the object in an image, *object detection*, marking the general area of an object e.g. by a square, or *image segmentation*, creating a pixel mask for the object in the image.

For the image segmentation task, the goal is to produce a pixel mask, henceforth mask, where each pixel is associated with a chosen label. Any given pixel will then either be classified as part of a given object instance, or not.

One specific area is the extraction of buildings from aerial photos, which provides data that can be used by areas such, as real estate, industry, homeland security, flood management, and city planning, e.g. to create a model of the city. While this task can be done manually by hand, it is a tedious and costly process, and as such automating this process could provide a cheaper and faster alternative. This however comes with a set of challenges; e.g. noise from the image can make the end product blurry or oddly shaped. In this paper we aim to alleviate this problem, in order to produce a sharper and more accurate representation of the building's location. We do this by taking an existing image segmentation model, and attempt to optimize its loss function for mask generation, to achieve improved results.

Existing work concerning this problem tries to account for some of the difficulties in various ways. One example is [Chen *et al.*, 2014], where the focus is put on detecting the shadows cast on buildings, as these could obscure the roofs, making it hard to effectively mark the building. This is done by first separating the image into smaller sections, or clusters, and then using an algorithm to classify these sections. The disadvantage of their method is that the method consists of multiple parts, that are independent of each other, an therefore unable to work together.

Other works use LiDAR data, containing information about the height of each pixel in the image. In [Rottensteiner and Briese, 2003] LiDAR data is used to construct a 3D model of a group of buildings. This is done by using the LiDAR data to get a rough estimate of the roof of the buildings, passing this information to an algorithm separating the different buildings, and then finding the slope of each roof to construct a 3D model. However only using LiDAR data may not provide enough information, especially when the task involves detecting a certain type of building, e.g. residences.

In [Lee *et al.*, 2008] LiDAR data is used together with aerial images to find the locations of buildings. This is done by first proposing a set of candidate regions, where buildings might be present, and then using the aerial images to find the best candidates. This method exploits the benefits of the two data types, however as the many processes applied to the data are not aware of each other, it may be hard to fully utilise both data types, as their information can not be connected more directly.

To process images and produce a useful output, a model is needed that is designed to extract information from images. Fortunately, such a model has gained a lot of popularity lately, this being the Regional Convolutions Neural Network (R-CNN). This network model, together with the more recent descendants of it, proposes a set of regions in an image, and processes the information in the regions separately to produce an output, thus providing both image classification and image segmentation.

However, the Regional Convolutions Neural Network is a general solution for image processing, and is thus unlikely to achieve outstanding performance on a specific problem. In this paper we therefore propose to improve the model for

extracting buildings from aerial data, by optimizing the loss function. For our problem we choose a variation of the R-CNN, the Mask R-CNN, designed to produce pixel masks. To achieve better performance we test a series of alternate loss functions, to investigate what works best for our problem domain. We also construct our own dataset using aerial data to test the performance of the network, using aerial images together with LiDAR data to increase the amount of information available for the network. In this paper we make the following contributions; 1) Create our own dataset for building detection, using aerial images and LiDAR data. 2) Show the performance of Mask R-CNN using different loss functions.

2 Related Works

2.1 Regional Convolutional Neural Network

The Regional Convolutional Neural Network (R-CNN), has gone through several iterations, from the most simple model [Girshick *et al.*, 2014], to the newer faster versions [Ren *et al.*, 2015] [He *et al.*, 2017].

The idea behind R-CNN is that instead of looking through every possible region, of the image, a set of regions, or areas within the image, are proposed, in which objects might be present. These regions are then processed further, depending on the desired output, such as *image classification*, labeling the object in the region, *object detection*, refining the region the in which the object is located, or *image segmentation*, creating a pixel mask for the object.

The ability to create masks for the object in a region was introduced in the Mask R-CNN version from [He *et al.*, 2017]. This process was made to run parallel to the other tasks, such that as little overhead was added as possible, ensuring that the fast performance of the network was not reduced.

2.2 Edge Aware Models

To increase the performance of a model used to compute a semantic segmentation task, some papers have experimented with creating a line marking the edge, or boundary, of the object, in addition to creating a pixel mask for each object. One reason for computing the edge of an object is to help the model differentiate between overlapping, or touching, objects, by making it not only locate the objects, but also create an outline of them.

This was the idea in [Mou and Zhu, 2018], where the goal was to create a mask, as well as an outline of each individual car, from an aerial photo, to improve the performance of the segmentation task. To perform this task, the model was constructed to compute these two output from data processed by a series of convolutions and residual layers, which could then be used to create the individual masks.

In [Kirillov *et al.*, 2017] they also use edge detection to separate different objects, in photos taken from a human perspective, which could be defined as a more complicated task, as a greater overlap between the objects is possible. To create masks for each object, they calculate the probability for each pixel to belong to a certain class, and the probability of it touching another object. They then divide the image into small clusters and use this, together with the previous probabilities, to construct masks for each object.

The idea behind edge detection is to differentiate objects within an image, by looking for a sharp change in intensity or color. A common method for finding the edge of an object is by using the *Sobel Operator*, that is split into a horizontal and a vertical directions (also referred to as the x and y directions). A kernel is then created for each direction, to find edges, or changes in the image, for that orientation. The output of each direction can then combined to one value, representing how big the change is.

3 Preliminaries

The data we use consists of the following three different data types; Aerial photographic data (orthophoto) O, which has three color channels red, green and blue. A height map H derived from LiDAR data, including a surface-map H_S (buildings, trees and other objects), and a terrain-map H_T, with data points excluding surface objects. There is also a set of building polygons B (representing building footprints), where each individual polygon consists of a series of points outlining the building. We assume a complete geographical overlap, between layers O, H and B.

Problem Definition

We provide a formal description of the problem, such as to precisely define the input and output of this paper.

Input

The information available to us is the orthophoto, LiDAR, and polygonal data, which will create the foundation for this project, and are used to train the model to produce our desired output. See Equation 1.

$$Problem = (O, H, B) \tag{1}$$

Output

The output we wish to achieve, is a set of bitmasks, where each mask represents a building within the input data. See Equation 2.

$$Solution = \{bitmask_1, .., bitmask_i \mid i > 0\} \tag{2}$$

4 Method

From preliminary experimentation, we quickly observed that the Mask R-CNN model exhibited weak performance in regards to mask accuracy, when applied to our dataset. The model is very good at recognizing buildings, and does quite a good job at drawing a bounding boxes. However, the produced masks are not representative of the building polygons. It became evident that the masks tended to have rounded shapes, and thus did not fit most buildings well. Buildings tend to have right angles, and straight sides. These seem difficult for the Mask R-CNN model to accurate produce masks for, the same observation can be reflected in the *mrcnn_mask* loss metric. In this section we will describe our process for constructing our dataset using areal images and LiDAR data.

We then give a more detailed description of the architecture of the Mask R-CNN, and its components. Lastly, we present the loss functions we will train the model with.

4.1 Model Input & Output

We provide a formal description such as to precisely define the input and output of the model. When talking about the dataset, it is important to first understand what input the Mask R-CNN model works on, and what output it produces.

Model Input

The model (during training) takes two inputs, an image (tile) t and its associated masks (ground truth) $< mask_1, .., mask_i >$. Section 4.2 covers how we derive data in this form. In inference mode, the model only takes t. See Equation 3.

$$Input = (t, < mask_1, .., mask_i >) \textit{ where } i > 0 \quad (3)$$

Note that i must be greater than zero, otherwise there is no mask, ground truth(s), associated with the input, and thus training is impossible. Tiles without masks must therefore be filtered out before training.

Model Output

The model produces three outputs for each detected instance: a classification label $class$, a bounding box $bbox$ and a segmentation mask $mask$. The model is trained to optimize the prediction accuracy of these three outputs. See Equation 4.

$$Output = \{(class_i, bbox_i, mask_i) \mid i \geq 0\} \quad (4)$$

The output contains i predictions, each representing a detected building. If no buildings are detected, the output set may be empty, $i = 0$.

4.2 Dataset Processing

This section covers the sourcing and processing of the data used to derive our dataset. A brief explanation is followed by a formalized description of the processing.

Our data source offers height map data in three formats, a point cloud (raw LiDAR data points), a terrain map (raster) and a surface map (raster). We utilize the terrain H_T and surface H_S rasters to derive a new height map, which we denote as *relative height map* H_R. The terrain map only includes the terrain (hills, ditches, etc.), and the surface map includes surface features (buildings, bridges, vegetation, etc.). Both of these height maps are relative to sea level. Therefore we can compute a relative height map by subtracting the terrain from the surface. By doing this we can remove the terrain (which does not hold relevant information, with regards to our problem), and effectively put all surface features on a level ground. Note that height values are now relative the ground level, as opposed to sea level.

The data was sourced from Kortforsyningen (Danish Map Supply, Agency for Data Supply and Efficiency), and is publicly available. This includes surface and terrain height maps, orthophoto and building polygons.

The orthophoto data is usually provided as arbitrarily large blocks, with extremely high pixel resolution, the same being true for the height data. Because of this high resolution, we process it to produce a set of smaller tiles, with reasonable spatial and pixel resolution dimensions. We also compute the relative height map, correct for the differences in resolution between the data, and append the height map data to the orthophoto data. This is done by using a 4th channel in the orthophoto raster. Some image formats support a 4th channel, often interpreted as transparency (e.g. PNG). We utilize this, to place the height map data in said 4th channel, alongside the RGB channels from the aerial imagery. An illustration of this can be seen in Figure 2.

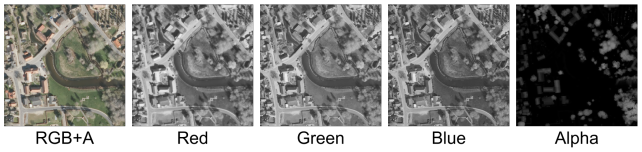

Figure 2: Illustration of channels in an image with appended relative height data, in the fourth channel. Individual channels are rendered in greyscale.

The resulting imagery is then sliced up into smaller tiles of reduced spatial size and image resolution. The building polygons (ground truths) are sliced likewise, such that we have a set of tiles, and their corresponding ground truths. For each tile, a set of bitmasks is produced where each mask corresponds to a single building feature in said tile.

The following is a step-by-step explanation of how we derive the dataset from the source data. We have four input data: The terrain height map H_T, the surface height map H_S, the orthophoto O and building polygons B. Note that we assume a complete overlap between these inputs, such that they all cover the same geographical area. From these inputs we produce a set of image tiles T, where each tile t_n has an associated set of bitmasks $< mask_1, ...mask_i >$. The set T is the output of this procedure, and thus describe the form of the dataset. See Equation 5.

$$T = \{(t_1, < mask_1, .., mask_i >), .., (t_n, < mask_1, .., mask_j >) \mid n, i, j > 0\} \quad (5)$$

The following steps one (1) through four (4), explains the process used to produce T:

Step 1 - Compute Relative Height Map

Using the terrain and surface height maps, H_T and H_S we derive the relative height map H_R by subtraction.

$$H_R = H_S - H_T \quad (6)$$

Step 2 - Merge

We now append H_R into the 4th channel of O. As this is a rather simple process, it will not be explained in detail, but

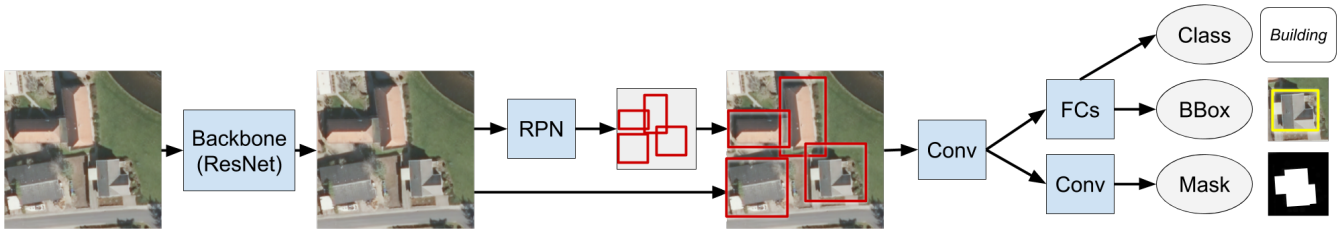

Figure 1: Mask R-CNN model architecture illustrated with image example.

simply denoted as the function *append*. From this operation we obtain an image M, with four channels (R, G, B, A), where the alpha channel contains the data from H_R.

$$M = append(O, H_R) \tag{7}$$

Step 3 - Slice to Tiles

We now cut the image M into a set of tiles T'. Each tile $t \in T'$ has dimensions $N \times N$, while M has dimensions $L \times L$. N must be selected such that $(L \mod N = 0)$. This ensures that it is only possible to produce tiles such that all tiles fit within M. The tiling algorithm is denoted as $\rightarrow$.

$$M \rightarrow T' = \{t_1, .., t_n \mid n > 0\} \tag{8}$$

The same procedure is performed on the buildings B.

$$B \rightarrow B' = \{b_1, .., b_n \mid n > 0\} \tag{9}$$

Given the sets T' and B' we can produce pairs, such that each tile t is associated with the set of buildings b located within the spatial area of t.

$$\{(t_1, b_1), (t_2, b_2).., (t_n, b_n) \mid n > 0\} \tag{10}$$

Step 4 - Produce Bitmasks

For each tile/buildings pair (t_j, b_j) we can now produce a bitmask for each building feature in b_j. This produces a number of bitmasks $masks$, which remains associated in-place, with the tile t_j. Each bitmask has the same dimensions $N * N$ as the tile. This algorithm is denoted with the arrow $\rightsquigarrow$.

$$(t_j, b_j) \rightsquigarrow (t_j, < mask_1, .., mask_i >) \mid j, i > 0 \tag{11}$$

Thus we have arrived at the output as described in equation 5.

4.3 Architecture Overview

Mask R-CNN [He *et al.*, 2017] expands on the architecture of Faster R-CNN, by adding the ability for the network to create pixel masks for the object within the image. The network consists of five different parts, those being the backbone, RPN, and three *heads* performing classification, bounding boxes and masks respectively.

In figure 1 the architecture of Mask R-CNN can be seen, shown with intermediate image representations. The input image is processed by the backbone network, consisting of a series of residual convolutional layers. The output is then passed to the *Region Proposal Network* (RPN), which proposes a series of *Regions of Interest* (ROIs). Each ROI is fed through an additional convolutional network, before it branches off to the *heads*. These heads will then work individually, to produce either a classification, a bounding box or a segmentation mask, for all ROIs. Each part of the model has its own respective loss metric (e.g. RPN, mask generator, classification).

In this project we have utilized an implementation of Mask R-CNN, using Keras on Tensorflow, provided by Github user Matterport [Matterport, 2019].

Architecture Modules

The Mask R-CNN model, as described previously, consists mainly of five components, see figure 1. While these are connected with additional layers (e.g. convolutions), we disregard these and provide an explanation of the significant and distinct parts as follows:

Backbone

The first component in the model is the backbone. An interchangeable subnet, whose responsibility is to analyze the input image for objects. Any object recognition model may be used, however the Mask R-CNN implementation [Matterport, 2019] we utilize in this project has ResNet-50 and ResNet-101 implemented (we use the latter). The backbone outputs a set of *feature maps*, each of which is then used to construct the later *feature pyramid* in the Region Proposal Network.

RPN

The second component is the RPN (Region Proposal Network). This works on the output from the backbone, determining whether recognized objects are of significant importance. In other words, it produces proposed regions of interest (ROIs), in the input image. Each of these region areas are then passed forward as input for each of the following heads (class, bbox and mask heads). This is done by computing a feature pyramid, where each level of the pyramid contains interesting features and different depths in the image. At each depth in the pyramid, regions of interest are selected and passed on.

Class Head
The class head takes a ROI from the RPN as input, and performs classification on the given object, associating it with a label from a set of classes. In our case we only have one class, which is *building*.

BBox Head
The bbox head (bounding box) takes a ROI as input from the RPN, and attempts to fit a minimum bounding box around the given object.

Mask Head
The mask head takes a ROI as input from the RPN. This head performs classification on each pixel in the input region, in order to determine which pixels belongs to a given object, and which does not. This results in a pixel mask (bitmask) segmentation of the object in question.

Thus, when all three heads are employed, the model outputs, for each recognized object: a classification label, a bounding box and a segmentation mask.

4.4 Loss functions

The default implementation of the Mask R-CNN utilizes Cross Entropy (CE) to calculate the loss for the mask generation. Intuitively, this a rather simplistic approach to image segmentation loss, and we hypothesize that employing a different loss function, will yield a better performing model, with regards to our problem. We therefore describe two additional loss functions, which we implement in Mask R-CNN, to explore whether they can improve the mask output of the model.

The first described is CE, the loss function implemented in Mask R-CNN, which we believe may contribute to less than optimal mask generation. Second is the squared hinge loss (SH), and third is Edge Aware (EA) loss. We run experiments using these three functions as losses for the Mask head of Mask R-CNN, to determine their effects on the model's performance.

For the loss functions we use the notation $\hat{y}$ to denote the predicted value, being the value produced by the model. We use the notation y to denote the actual value, or ground truth, to which the predicted value is compared.

Cross Entropy
Cross Entropy is a loss function based on calculating the entropy across two distributions. In supervised learning these two distributions can be thought of as a know distribution y and a presumed distribution $\hat{y}$.

$$CE = -\frac{1}{N}\sum_{i=1}^{N} y_i * log(\hat{y}_i) + (1 - y_i) * log(1 - \hat{y}_i) \tag{12}$$

To calculation the loss the network predict a value $\hat{y}_i$ between zero and one, where zero would be the case of the data not belonging to the class, and one being the data belonging to the class. The function, found in Equation 12, uses either the left or right hand term, if the ground truth y_i is either true or negative respectively.

The loss itself, is the reverse of the logarithmic function, and thus grows at an increasing rate when approaching zero, punishing predictions that are far away from the truth more, in respect to rewarding the predictions that are closer to the truth.

Squared Hinge
Hinge loss builds on the theory that a model should not focus on the predictions that are correct, specifically on those it is very certain of, but instead should instead focus the incorrect predictions. For this reason all predictions which are correct are treated the same, providing zero loss, while the loss for the incorrect predictions are based on their difference to the correct predictions.

$$SH = \frac{1}{N}\sum_{i=1}^{N}(max(0, 1 - y_i * \hat{y}_i)^2) \tag{13}$$

For the Squared Hinge loss function, which can be seen in Equation 13, the labels y_i for the data are either 1 or -1. As the multiplicative of the correct and the predicted values are subtracted from one, this means that for the cost to be zero, the multiplicative of the the two values have to result in a number equal or greater than one.

As we are working with values between 0 and 1, we will have to stretch this range, such that it matches the expected range of SH, -1 and 1.

Edge Aware
The Edge Aware loss function EA utilizes convolutions to detect edges in both the ground truth y and the predicted value $\hat{y}$. L2 loss is then applied on the convoluted values $y\prime$ and $\hat{y}\prime$, after which each filter layer is merged into a single layer by the mean product. Then an optional weight is applied, and finally the mean loss is outputted as shown in Equation 14.

The reason to make the loss function edge aware lies in the assumption that most buildings have distinct consecutive edges, that often conclude in right angles. Thus by making the model aware of these edges, we will be able to improve the mask generation.

$$EA = \frac{1}{N}\sum_{i=1}^{N} W * \frac{1}{M}\sum_{j=1}^{M}(f(y_i)_j - f(\hat{y}_i)_j)^2 \tag{14}$$

In Equation 14 the function f is the filters applied on the inputs y and $\hat{y}$ as shown in Equation 15.

$$f(y) = f * y, \text{ where } * \text{ is the convolution operation} \tag{15}$$

Edge aware loss can utilize most edge filtering operators, but for this project the *Sobel Operator* and the *Prewitt Operator*, have been chosen. Both operators are represented by two $3x3$ kernels, the *horizontal* (x) and *vertical* (y), shown in Equation 16 and Equation 17.

The Sobel operator computes an approximate derivative of the input, which represents the gradient of the image.

This is useful because it allows EA to pick up on spatial qualities of the mask, that another loss function, such as CE, cannot.

$$sobel_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix}, \; sobel_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{16}$$

The purpose and intuition of the *Prewitt Operator* (equation 17) follows that of the Sobel operator.

$$prewitt_x = \begin{bmatrix} +1 & 0 & -1 \\ +1 & 0 & -1 \\ +1 & 0 & -1 \end{bmatrix}, \; prewitt_y = \begin{bmatrix} +1 & +1 & +1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \tag{17}$$

A problem with using Edge aware loss is that the output is the approximate derivative, and does not wholly represent the correct ground truth. This can be mitigated by applying the *identity* operator, shown in Equation 18. As this operator will output the identity, consequently making the function also output an MSE loss.

$$identity = \begin{bmatrix} 0 & 0 & 0 \\ 0 & +1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{18}$$

5 Experiments

This section covers the experiments we have conducted. We present training parameters and our strategy, as well as details about the datasets. Lastly, we present the results yielded from the experiments.

5.1 Dataset

We have produced a dataset, encompassing both rural, suburban, and urban features. The dataset covers an area in and around the city of Copenhagen, the capital of Denmark. The location was chosen due to the aerial imagery and LiDAR captures being as temporally close as possible. Also, a large variety of building types can be found, and little area is wasted on non-populated areas. Note that the dataset does not have a perfect temporal synchronization between the capture dates of LiDAR and aerial imagery.

Dataset	Copenhagen
Area	600 km^2
# of Buildings	396,543
Orthophoto year	2019
LiDAR year	2019
# of Tiles	9,600
Tile Size	250×250 m
Tile Resolution	1024×1024 px

Table 1: Our dataset, and its respective attributes

5.2 Training Parameters

We apply a training/test split of 75/25, such that a quarter of the dataset is reserved for testing. As our datasets are fairly large, we take the liberty to use a rather large test set. If we used a small test set, we would risk that the set contains an unbalanced variety of building types. This would make the test set biased towards certain buildings types. The significant training parameters used, can be found in table 2.

Parameter	Value
Train/Test split	75%/25%
Backbone	ResNet-101
Mini mask shape	56×56 px
Image resize	512×512 px
Train ROIs per image	200
Learning rate	1×10^{-4}
Min. detection confidence	0.9

Table 2: Training parameter configuration

The approach we have chosen for the training process, is to run it for an undefined number of epochs, until overfitting starts to occur. We then use the weights, taken imminently before the exhibition of overfitting behavior. We do this for sake the of fairness, as different model variations may require longer or shorter training phases to reach their potential. Thus, the metrics used in this project reflects the best achieved losses, with no overfitting. Furthermore, the training strategy and parameters (see figure 2) are kept constant across all model training sessions. We initialize the model with pre-trained ResNet-101 weights.

Note in table 2 that input images are resized (downsampled), from their original size ($1024x1024$). This is to cut down the memory requirements and training time.

5.3 Training Strategy

Before we train the entire model, we need to perform some initial training on different components. Due to our input image having four channels, we cannot use pre-trained weights for the first convolutional layer in the model. Furthermore, it is recomended when training the Mask R-CNN model, to train the backbone and the heads individually initially. We therefore employ a simple training strategy, as follows:

Stage 0, (1 epoch)
We train the first convolutional layer. All other layer weights are frozen. This stage uses a higher learning rate of 1×10^{-3}.

Stage 1, (3 epochs)
We train the backbone subnet (ResNet-101), with all other layers remaining frozen. This stage also uses a higher learning rate of 1×10^{-3}.

Stage 2, (1 epoch)
We train the network heads (class, bbox and mask heads).

Stage 3, (∞ epochs)
We train the entire model, for an unbounded number of epochs, until overfitting occurs.

	mIoU (%)	F1
MRCNN+CrossEntropy*	81.68	0.3346
MRCNN+SquaredHinge	79.84	**0.3686**
MRCNN+EdgeAware	**83.27**	0.3273

Table 3: Comparison of MRCNN with chosen evaluation metrics, mIoU (mean Intersection over Union) and the F1 score. *(Baseline).

5.4 Results

After having conducted three experiments, by training the Mask R-CNN model with three different loss functions for its mask generation, and having performed testing, we have arrived at some evaluation metrics (shown in table 3). The mask losses are the result of different loss functions, and thus cannot be compared directly. We therefore calculate a mean Intersection over Union (mIoU) and F1 scores, when testing in order to yield comparable metrics. The most relevant metric is mIoU, as it directly computes the overlap of masks, between predicted and ground truth.

The results shown that the best mIoU performance is achieved by EA, and the worst performance is SH. The best performance of F1 score is achieved by SH, while the worst is EA.

We use two evaluation metrics in table 3, which are defined as follows:

mIoU

Mean Intersection over Union, computes the overlap between predicted masks and ground truth masks, as a percentage. This is done for each prediction/ground truth pairs, and averaged to a single metric.

$$mIoU = \frac{1}{N}\sum_{i=1}^{N}\frac{|mask_i^{gt} \bigcap mask_i^{pred}|}{|mask_i^{gt} \bigcup mask_i^{pred}|} \quad (19)$$

F1 score

The F1 score is the harmonic mean of precision and recall calculated for each test sample.

$$\textit{F1 score} = \frac{1}{N}\sum_{i=1}^{N}(2 * \frac{precision_i * recall_i}{precision_i + recall_i}) \quad (20)$$

6 Conclusion

It follows from the results in section 5.4 in Table 3, that the Squared Hinge loss function outperforms both the baseline and Edge Aware, by a significant margin in F1 score ($\sim 10\%$). This is a positively surprising result, however in terms of mask generation, the Squared Hinge loss produces slightly weaker results than the baseline.

We *had* expected to see an improvement over both the baseline and Squared Hinge, using Edge Aware loss. While the resulting F1 score for Edge Aware is comparable with the baseline (albeit slightly worse), the mIoU shows a noteworthy improvement ($\sim 1.9\%$) relative to the baseline.

It is important to note that we do *not* believe the evaluation metrics to be grounds for an absolute conclusion. There is too much uncertainty in the training strategy and hyperparameters, which remains unexplored, and may shift the results. However, we *do* observe a clear advantage for both the Squared Hinge and Edge Aware loss functions. Importantly, we observe a noteworthy improvement in the mask generation, using the Edge Aware loss, from looking at the mIoU evaluation metric.

We can therefore conclude that the Edge Aware loss function should be applied in the Mask R-CNN model, when working with our dataset, as it retains a similar F1 score, but grants improved mIoU score. It might also hold true that Edge Aware Mask R-CNN may show significant improvements elsewhere in other problems, where the segmentation boundary accuracy may be of importance.

7 Future work

This section describes a few different directions which potential continued work may explore.

7.1 Combine Squared Hinge with Edge Aware loss

From the experiments we have observed that both the introduction of Squared Hinge and Edge Aware separately, contribute to an improvement in model performance. Thus, we propose that the model may be trained with parallel weighted loss functions, where both Squared Hinge and Edge Aware are employed simultaneously, potentially achieving the benefit of both.

7.2 Modified Backbone Net

As for this project, we have utilized the ResNet-101 as the backbone component in Mask R-CNN. It seemed to perform well, however, we are open to the possibility that a different backbone may exhibit improved performance, in terms of its capacity to accurately recognize buildings (and potentially other surface features), and even differentiate between building types. This project was concerned with the improvement of the mask generation component, and as such, the backbone has went largely ignored.

7.3 A GAN Approach to Mask Generator Improvement

Through the duration of this project, we have explored different avenues to improving the mask generation component of the Mask R-CNN model. One of these was the introduction of a *Discriminator* component, inspired by its successful usage in GAN models (Generative Adversarial Networks). The basic idea is that a generator and a discriminator acts as each others adversaries, such as to improve their performance. Based on this intuition, we explored the introduction of a discriminator as an adversary to the Mask generator in Mask R-CNN. However, due to time constraints, this idea was scrapped. However, we think this approach may be worth looking further into, in the future.

7.4 Additional Filters for Edge Aware Loss

While this project has worked with the application of filters, the filters we have chosen to apply are fairly simple. Future experimentation with various other filter types, e.g. the

Laplace operator, may prove an interesting endeavour for the Mask R-CNN mask head loss function.

7.5 Better Training Strategy

The training protocol described and applied in this project could have been conducted better. We propose that future work utilizes a pre-trained Mask R-CNN model, trained without the mask head. Said trained model should then be frozen, and the mask head should be trained thrice, with the three loss functions. Thus each experiment would have the same pre-trained weights for the entire model, with the exception of the mask head.

References

[Chen *et al.*, 2014] Dongyue Chen, Shibo Shang, and Chengdong Wu. Shadow-based building detection and segmentation in high-resolution remote sensing image. *journal of multimedia*, 9(1):181–188, 2014.

[Girshick *et al.*, 2014] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[He *et al.*, 2017] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[Kirillov *et al.*, 2017] Alexander Kirillov, Evgeny Levinkov, Bjoern Andres, Bogdan Savchynskyy, and Carsten Rother. Instancecut: from edges to instances with multicut. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5008–5017, 2017.

[Lee *et al.*, 2008] Dong Hyuk Lee, Kyoung Mu Lee, and Sang Uk Lee. Fusion of lidar and imagery for reliable building extraction. *Photogrammetric Engineering & Remote Sensing*, 74(2):215–225, 2008.

[Matterport, 2019] Matterport. Mask r-cnn. https://github.com/matterport/Mask_RCNN, 2019.

[Mou and Zhu, 2018] Lichao Mou and Xiao Xiang Zhu. Vehicle instance segmentation from aerial image and video using a multitask learning residual fully convolutional network. *IEEE Transactions on Geoscience and Remote Sensing*, 56(11):6699–6711, 2018.

[Ren *et al.*, 2015] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[Rottensteiner and Briese, 2003] Franz Rottensteiner and Christian Briese. Automatic generation of building models from lidar data and the integration of aerial images. 2003.