Evaluation of Instance Segmentation Methods in the Context of Construction Waste

Emil Sandelin

Master's thesis
2021:E70

Lund Institute of Technology
Lund University

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

Abstract

With the progress made within the field of computer vision the last few years, it is starting to become possible to perform instance segmentation in real time with relatively cheap hardware. This paper's main purpose is to investigate different methods of instance segmentation in the specific case of segmenting construction waste with only a single class. The paper experiments with two models, Mask RCNN and SOLOv2, using ResNet and DenseNet backbones of different depths, and evaluates it according to average precision and prediction time. The author recommends a ResNet50-FPN Mask RCNN model, due to having a good precision-time tradeoff and being easy to implement in the current system.

Acknowledgements

First I want to thank Fredrik Brandt for giving me the opportunity to work on this interesting problem and for his support, and Microsoft for financing the project.

I also want to thank my supervisors Mikael Nilsson and Carl Olsson, for their valuable feedback and suggestions. I especially appreciate the work they did during late spring that allowed me to start this project during the summer.

Lastly, I want to thank my friends and family for their encouragement and support throughout my time at university.

Emil Sandelin
Lund, 18th November, 2021

Contents

Chapter 1

Introduction

The thesis was done in cooperation with OP Teknik and Innovation Skåne. OP Teknik provided the data and valuable input during the process. The company currently produces and sells a robot model called SELMA, which is capable of sorting construction waste on a conveyor belt into several classes. Sorted waste can be sold for recycling, a necessity for a circular economy and sustainable society, but the segmentation algorithms utilized in SELMA are lacking when objects appear close to, or on top of, each other. Algorithms that could improve the segmentation would mean increased throughput and efficiency.

1.1 Purpose

The purpose of this thesis is to investigate and evaluate methods of instance segmentation, in the context of close and/or overlapping pieces of construction waste on a conveyor belt. Evaluation of these methods will be based on both quality of the segmentations and the time it takes to perform the segmentations. The results of OP Teknik's current methods will be used as a baseline for comparison.

1.2 Limitations

The goal of this thesis is to test how well the current algorithm compares to other state-of-the-art methods. The goal of the thesis is not to produce a new form of architecture for instance segmentation, nor is it to classify objects into classes. Classification and implementation can be done if time allows, as it is expected to produce better results.

Chapter 2

Background

In this chapter we discuss how SELMA operates and how it relates to the problem at hand. The chapter then introduces the relevant areas of research that can provide a solution to the problem. Finally, the chapter ends with a discussion on how this can be implemented in the robot.

2.1 SELMA

SELMA is a robot consisting of a conveyor belt, 6 robot arms and 6 bins, each for one type of recycled material. An image of SELMA operating can be seen in Figure 2.1

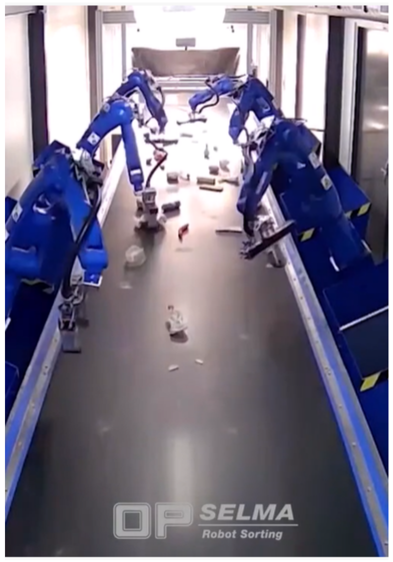


Figure 2.1: An image of SELMA in action.

The current process utilized in SELMA can be divided as the flowchart in Figure 2.2.

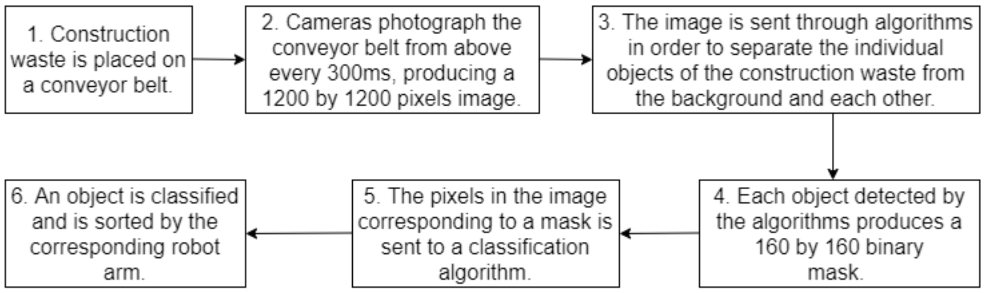

1. Construction waste is placed on a conveyor belt.
2. Cameras photograph the conveyor belt from above every 300ms, producing a 1200 by 1200 pixels image.
3. The image is sent through algorithms in order to separate the individual objects of the construction waste from the background and each other.
4. Each object detected by the algorithms produces a 160 by 160 binary mask.
5. The pixels in the image corresponding to a mask is sent to a classification algorithm.
6. An object is classified and is sorted by the corresponding robot arm.

Figure 2.2: SELMA segmentation and sorting process.

The flowchart can also be visualized as in Figure 2.3

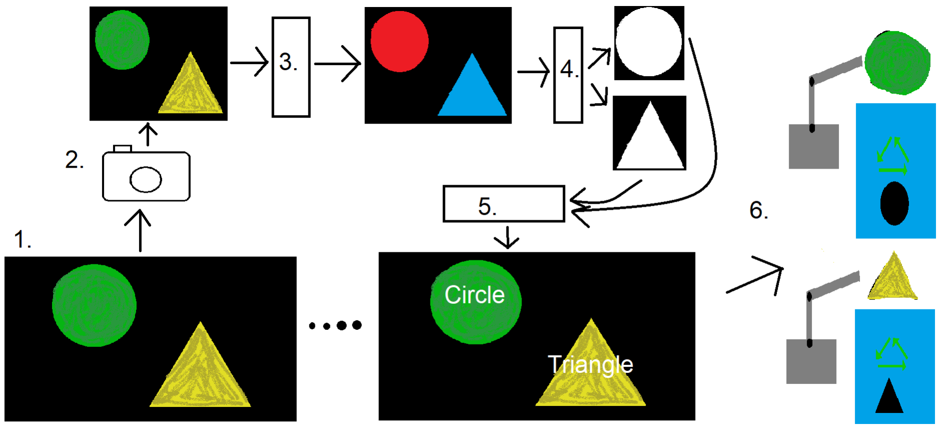


Figure 2.3: Graphic visualization of SELMA sorting process.

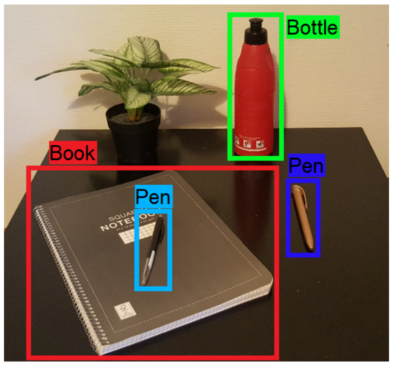
The current algorithms used by OP Teknik in step (3) consist of classical morphological operations, meaning that they are quick but produce unsatisfactory results. The company hopes to improve the results by introducing algorithms that are better at separating objects that are close and/or overlapping. The different computer vision tasks concerned with classification, separate object from background, and localization, where is the object, will be introduced in the following section.

2.2 Computer vision tasks

Image classification is a task, in which an image is classified with a label. The possible labels vary depending on the problem. Image classification is insufficient for this project, since it can only produce one label per image and gives no information about where the objects responsible for that label are located, but is the foundation on which more complex tasks are built.

Object detection is the task that, for an image, produces a bounding box and label for each instance of the detected objects. An example of this task can be seen in Figure 2.4.

(a) Input image

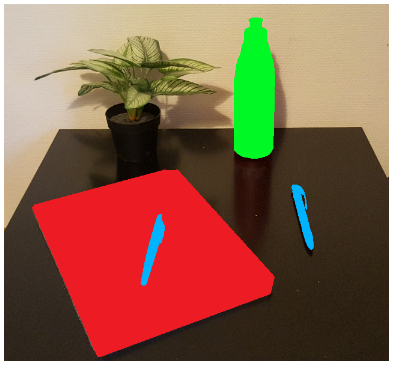


(b) Output image

Figure 2.4: Example of object detection.

Semantic segmentation can be seen as an extension of object detection in which, instead of finding the smallest box containing an object, the task is to mark which class every individual pixel belongs to. An example can be seen in Figure 2.5, note that both pens are marked with the same color.

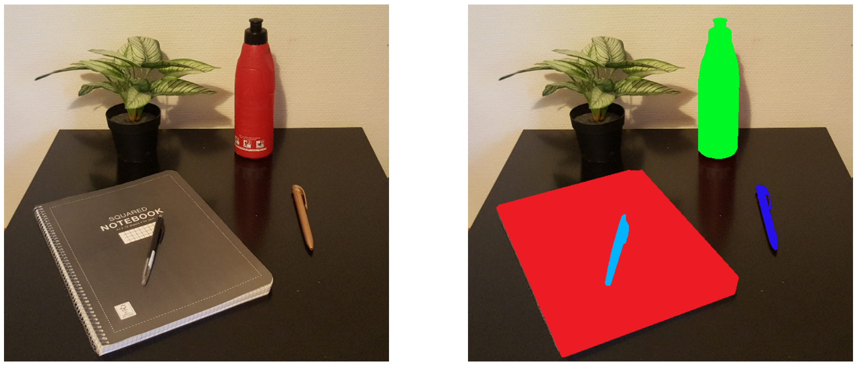
(a) Input image



(b) Output image

Figure 2.5: Example of semantic segmentation.

Instance segmentation extends semantic segmentation, in which every pixel belonging to an *instance* of a class is marked. All resulting pixels belonging to an instance is called a *mask*. An example of instance segmentation can be seen in Figure 2.6. Note that the pens have different colors, meaning they have been assigned different masks.



(a) Input image *(b) Output image*

Figure 2.6: Example of instance segmentation.

In the case of Figure 2.6, the resulting output image could have easily been produced by performing semantic segmentation, as in Figure 2.5, and then producing a mask for each isolated component per class. However, this method would not work well in cases when objects of the same class are occluding one another. So it is clear why semantic segmentation and instance segmentation are regarded as different tasks.

2.3 Neural networks

Over the past two decades, a model that has shown good performance on the four tasks described above is the *neural network*, more specifically the *convolutional neural network*. A short description of those models will now follow.

An artificial neural network (ANN) consists of layers which in turn consists of nodes [1]. Each node has an associated activator function and weight. In essence, given an input value, a node is *activated* if the input fulfills the condition of the activator function. An example of an activator function is the *Rectified linear unit* (ReLU), which can be written as

$$f(x) = max(0, x).$$

The node then outputs its weight times its function value, $y = w \cdot f(x)$, to a set of nodes in the next layer. For a node in the next layer, its input value is the sum of all outputs of the nodes in the previous layer that connect to it. In other words, the input value $x_{i,k}$ to the i:th node in the k:th layer is calculated as

$$x_{i,k} = \sum_j w_{j,k-1} y_{j,k-1},$$

where $w_{j,k-1}$, $y_{j,k-1}$ are the associated weights and outputs from the nodes in the previous layer. The node then produces its output, $y_{i,k}$, according to

$$y_{i,k} = w_{i,k} f_k(x_{i,k}),$$

for the layer function f_k.

For a task as complex as instance segmentation, a neural network can have millions of nodes in hundreds of layers. For such a network it is not feasible to adjust the weights manually, instead the neural network is implemented as a form of machine learning model. This means that the network will undergo a *training phase* during which it processes a *training set*, where it tries to predict each example and adjusts the weights by minimizing a loss function using back-propagation and a numerical optimization method. A loss function should in some way represent the task at hand, for instance by having a term for each subtask. A suitable loss function for instance segmentation could include a term for the number of wrongly classified masks and another term for the number of pixels wrongly classified as background.

2.3.1 Convolutional neural networks

A convolutional neural network is an ANN containing one or several *convolutional layers*, meaning that the output is calculated as convolution with the input and a set filters. Convolution is a mathematical operation which can be seen as equivalent to filtering. When an image is convoluted with a filter, the filter will "slide across" the image and output large values where the image matches the filter, and small values where the image does not match the filter. This is a form of feature extraction. One key difference between a convolutional layer and a traditional layer is that a convolutional layer only needs one filter in order to detect that feature everywhere in the input image, where as a traditional layer could need hundreds of thousands of parameters. So a convolutional layer can extract more features equivariant to translation with far fewer parameters. The features extracted from one layer are quite simple, but by chaining several convolutional layers together, the network is capable of extracting more refined features in an efficient manner. When trained, a CNN will learn the filter weights based on the data it is given, as opposed to being manually set filters. This makes CNNs suitable for computer vision tasks.

2.4 Dataset

The images provided by OP Teknik are 1200 by 1200 pixels, depicting a black conveyor belt with construction waste moving along it, an example can be seen in

Figure 2.7. Briefly mentioned in the previous section, a standard procedure used when developing approximate models for real-world applications is to create two datasets, training and testing. Data is extracted from the training dataset and used to construct the model. The model is then applied to the testing dataset, which allows the model to be evaluated according to selected evaluation metrics.

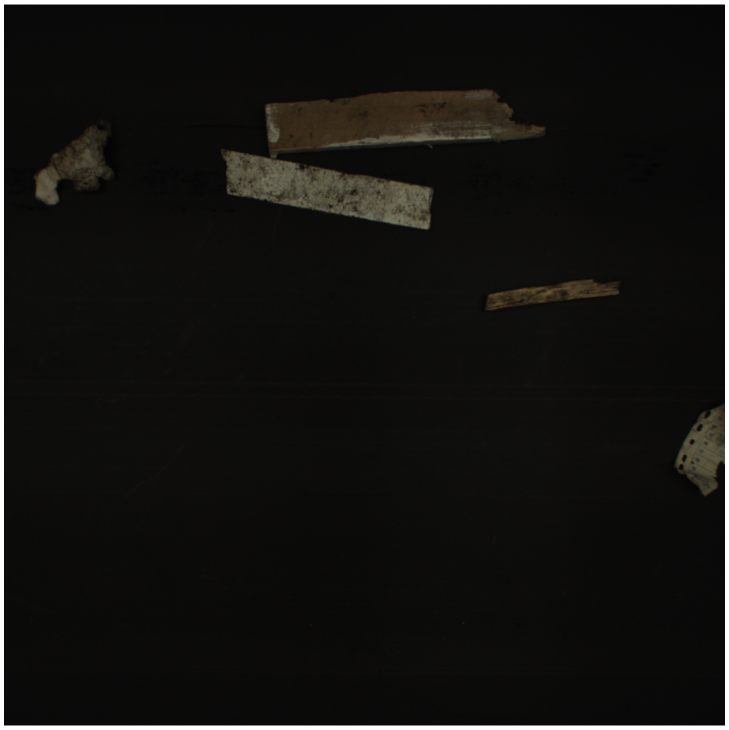


Figure 2.7: An example of an image in the dataset, depicting 5 objects on a conveyor belt.

Manual annotation is a tedious and time consuming process, which is why there are many data sets of different classes and contexts available to the public. There are those that are general [2] or specifically represent trash and waste [3]. Since the objects that are to be segmented in this thesis all are on a black conveyor belt, the background class of contextual data sets are of little use. These datasets could be used in order to generalize a model, making it less prone to be overfit, either by including the datasets in training or starting with a model already trained on such a dataset. While there are mentions of data sets of waste objects on a conveyor belt, they are often kept as intellectual property (IP). I could not find any data sets that are similar to the images SELMA could face.

An object does not need to be of the same material. It could for instance be two different material fused together, as can be seen in Figure 2.8

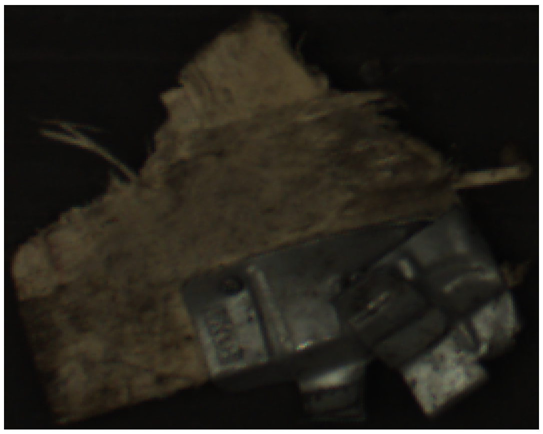


Figure 2.8: An example from the training data, showing one object consisting of wood and metal.

Since the materials of such an object as in Figure 2.8 cannot be separated by SELMA, an useful model should treat this as one object. Otherwise, the sorted wood container could be contaminated with metal which means that less material is recycled and money is lost.

2.5 Overview of the system

The problem at hand is to determine which pixels do no belong to the conveyor belt and then grouping the pixels into individual items. The first part of the problem can be seen as a semantic segmentation problem with one class called *object*, for pixels not belonging to the conveyor belt. In Figure 2.7, one can see 5 distinct items, each belonging to the class object, meaning that each item represents one *instance* of the object class. The rest of the report will use "object" and "instance of the class object" interchangeably.

A *model* refers to a method or network aimed to solve an instance segmentation problem, either a general form or the one treated in this thesis. The difference should be clear from context. The masks produced by some models might not be as accurate as in Figure 2.6, and in that case, *post-processing* can be applied in order to refine the masks. Post-processing will be treated in chapter 4.5. A visualization of the new process for SELMA can be seen in Figure 2.9.

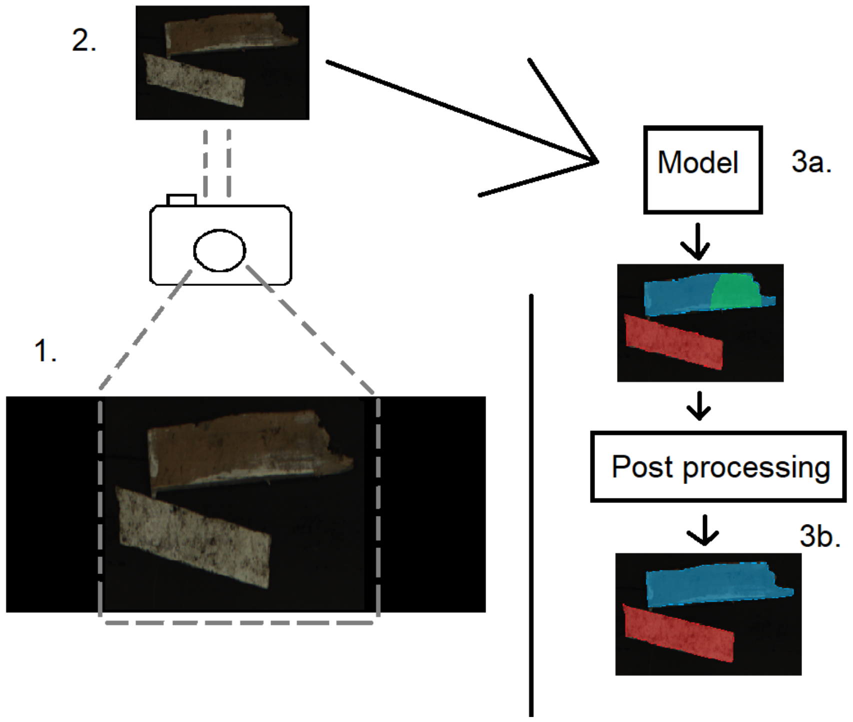


Figure 2.9: An overview of the SELMA work process.

Chapter 3

Related work

In this chapter I give an introduction to how different datasets are used in research and for evaluation of instance segmentation models. I also discuss which models of CNN that will be used in this thesis and their different parts.

3.1 Microsoft Common Objects in Context

Many of the articles cited in this thesis will mention the Microsoft Common Objects in Context (COCO) dataset and related tasks. For ease of reading, a short explanation of COCO will follow.

The COCO [2] dataset is a publicly available data set consisting of over 300.000 images of common objects, such as humans, oranges or zebras, in their expected environments. The data set is publicly available so that anyone can construct generalized models for different challenges, such as instance segmentation. Models can be evaluated for its respective task on a secret validation set according to several metrics. Although this thesis uses its own data set, evaluating according to the COCO challenges will make results more comparable due to its frequent use in research.

3.2 Backbones

As mentioned in Section 2.2, the different problems can be seen as subsets of one another. Since if one had a method to solve instance segmentation, the method would also be able to solve semantic segmentation trivially. Object detection and image classification would then also be solved trivially, simply extract the smallest box containing a mask and recycle the label. Naturally, instance segmentation is a very difficult problem and there is, at the moment, no method that can solve it perfectly, not even humans. But if one is willing to settle for less than perfect it is possible to reduce the complexity of the problem. Instead of trying to predict masks by directly using the image as data, the image is instead sent through a *backbone network*, which extracts features that can be used as data. Backbone networks are in this case CNN:s with the nomenclature *Network-Depth-Features*, describing the network architecture, number of layers and which features that are extracted. Another advantage of using backbones is that they can be *pretrained* when used in models. Having a pretrained backbone means that the weights have already been optimized, so training must only optimize the final layers. Pretrained backbones are essential in research so there is a large incentive to keep them available to the public. When it comes to instance segmentation, most backbones have been pretrained on the ImageNet dataset [4].

3.2.1 Network - ResNet

Major breakthroughs in computer vision related tasks over the past decade have been due increasingly deeper neural networks, and one might think that better results can be achieved simply by stacking more layers. But this has been shown to not be the case, as there comes a point when adding more layers makes the model worse. Given a network of N layers, He et al. [5] hypothesize that this happens when the model has "found" all useful features in the first M layers. The remaining $N - M$ layers must then represent an identity mapping but since layers are often non-linear, this is especially hard to optimize, causing the accuracy of the model to degrade. Let each remaining layer be represented as a function f_i, then the mapping of all remaining layers can be described as

$$F(x) = w_{M+1}f_{M+1}(x) \circ w_{M+1}f_{M+1}(x) \circ ... \circ w_N f_N(x).$$

It is easy to realize that optimizing the weights w_i so that $F(x) = x$ is a non-trivial problem, especially when the function are non-linear. However, optimizing the weights so that $F(x) + x = x$ is trivially solved by setting all the weights to zero. Even though the identity mapping is an extreme case, He et al. show that this reformulation produces better neural networks. In practice, this means that the input to a series of layers is added to the output of those layers. It is not feasible to perform this addition for every layer, so the network is built up by *blocks*, each consisting of several layers. The addition is performed for every block.

Networks that operate under this principle are called ResNets. The article shows that such a networks allows for deeper and better models.

3.2.2 Network - DenseNet

DenseNet is another neural network architecture, proposed by Huang et al. [6]. The main idea is to "recycle" features from previous layers in order to increase information flow in the network and avoid redundant layers. This is done by constructing the network as a number of *dense blocks*. Each dense block consists of l layers, where each layer performs a normalization, rectified linear unit and a 3x3 convolution. This composite operation is called $H_l(\cdot)$ for the l:th layer and its output is denoted x_l. In order to recycle previous features, the input tensor consists of the concatenation of all previous layers, so the output of the l:th layer is computed as

$$x_l = H([x_0, x_1, ..., x_{l-1}]).$$

A visualization of a dense block with 3 layers can be seen in Figure 3.1

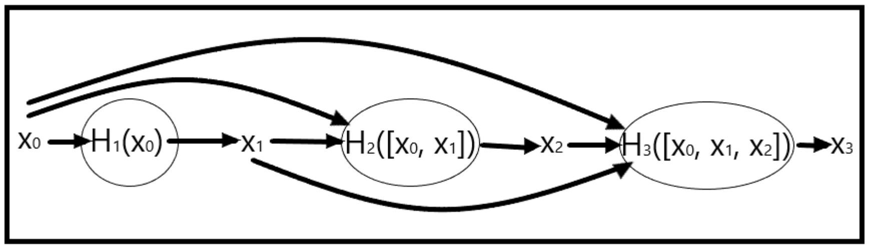


Figure 3.1: An overview of the information flow in a dense block with 3 layers, depicted as circles.

3.2.3 Depth

The depth is simply the amount of layers in the backbone. As all backbones used in this thesis are invented/constructed by researchers, the layers that are included in, say, ResNet50 is unambiguous and should be clear from context.

3.2.4 Features - FPN

Feature pyramid network [7] allows for features to be extracted across scales, which is useful when the size of objects vary. It works by making a prediction based on the features extracted from each block, and by including these features in the prediction for the next block. This means that the lower-level, detail, features is combined with the higher-level, contextual, features to make predictions. Compare this to just letting the low-level features construct the higher-level features, which in turn would make a final prediction, which would put much emphasis on the contextual information and disregarding the finer details.

As an example of the importance of both lower-level and higher-level features, one could imagine wheels and legs as two lower-level features and the number of appendages (wheels and legs) as a higher-level feature. Alone, the lower-level features are enough to differentiate animals and vehicles while the higher-level feature can make no such distinction. Combined, it is possible to differentiate motorbikes, cars, horses and chickens.

3.3 Mask R-CNN

One could imagine that quite good masks could be produced by first performing object detection and then try to create a mask for each bounding box. After all, object detection is a well researched task and would provide a solution to the problem of occluding instances, as described in section 2.2. He et al. states that this is the intuitive idea behind Mask R-CNN [8], a neural network that, when published, out-performed all current state-of-the-art methods in the COCO challenges. On top of this, it also made great improvements in inference time under certain conditions, such as using a fast and good RPN.

Mask R-CNN is very flexible, meaning that is a extension/modification of a two-stage object detection and classification model and not an architecture that can not be altered. The first stage is called a Region Proposal Network (RPN), which outputs Regions of Interest (RoI) which in turn are just bounding boxes where the network believes an object could be. The second stage then performs classification and "improves" the bounding box for each RoI. The Mask R-CNN extension is then a parallell branch in the second stage that computes a binary mask for every class and RoI, with the final output being the classification, bounding box and the binary mask belonging to the predicted class. Experiments showed [8] that predicting a mask for every class led to better segmentations overall, at a small cost in computation time.

3.3.1 Loss function

As Mask R-CNN performs 3 predictions, classification, bounding box and segmentation, its loss function is simply

$$L = L_{classification} + L_{boundingbox} + L_{mask}.$$

$L_{classification}$ and $L_{boundingbox}$ is defined the same as the chosen second stage method. As mentioned, Mask R-CNN produces one mask for every class but only the mask for the correct class contributes to L_{mask}, which is a pixelwise average binary cross-entropy loss.

3.4 SOLO

While the most accurate models developed for instance segmentation in recent years have been so called "two-stage methods", meaning that the model first performs object detection on the image and then predicts a mask for each detected object, Wang et al. claims SOLO [9] is the first "one-stage method" capable of producing results equal to those of the best two-stage method on the COCO Detection challenge, at the time of writing. A one-stage method has the potential to be much faster than a two-stage method, which will be explored in the SOLOv2 subsection. A fast prediction is of course desirable for real-time tasks, such as the task outlined in this thesis. SOLO is a neural network and, as is common praxis, also implements a backbone network for the first layers and builds custom layers on top of it, called heads.

3.4.1 Backbone

In the article, SOLO was evaluated using ResNet-X-FPN backbones.

3.4.2 Heads

The central idea of SOLO is to divide the input image into grids of size SxS. If the backbone detects the center of an object in a grid cell, that grid cell "is responsible" for predicting the class and generating the instance, the specific objects, mask. This is done by constructing SOLO as a fully convolutional network (FCN), meaning that the network only consists of convolutional, and possibly some up- and down-sampling layers. Since convolutional operations are spatially invariant and the position of an object is of importance, the network must have some way to access the pixel coordinates. This is solved by simply concatenating two images of equal size of the input image, one containing the normalized x coordinate and the other containing the normalized y coordinate.

3.4.3 Loss function

The loss function for SOLO is defined as

$$L = L_{cate} + \lambda L_{mask},$$

where L_{cate} is Focal Loss [10], a modifcation of Standard Cross Entropy that allows for better training on adjacent objects. λ is set to 3 in the original paper and

$$L_{mask} = \frac{1}{N_{pos}} \sum_k \mathbf{1}_{\{\mathbf{p}_k^* > 0\}} d_{mask}(\mathbf{m}_k, \mathbf{m}_k^*)$$

where N_{pos} is the number of *positive samples.* A positive sample is every grid cell that falls within the "center region", a box containing the center of mass with axes scaled proportionally to height and width, of a ground-truth mask. Index k represents a grid, starting left to right and top to bottom. A superscript asterisk represents ground-truth/target, so $\mathbf{m}_k$ is the mask produced by the k:th grid and $\mathbf{m}_k^*$ is the ground-truth mask for the k:th grid. If the labels can be represented by positive integers and the background corresponds to 0, then $\mathbf{p}_k^*$ is the ground-truth label/class of the k:th grid. The indicator function [11] $\mathbf{1}$ is equal to one for the elements in its subscript set, i.e. it is equal to one when there is no background. d_{mask} can be chosen as any function that compares a predicted mask to the ground-truth, in the article it was chosen as $d_{mask}(x, y) = 1 - D(x, y)$, where $D(\cdot, \cdot)$ is the Sorensson-Dice coefficient [12]. A visualization of the variables can be seen in Figure 3.2.

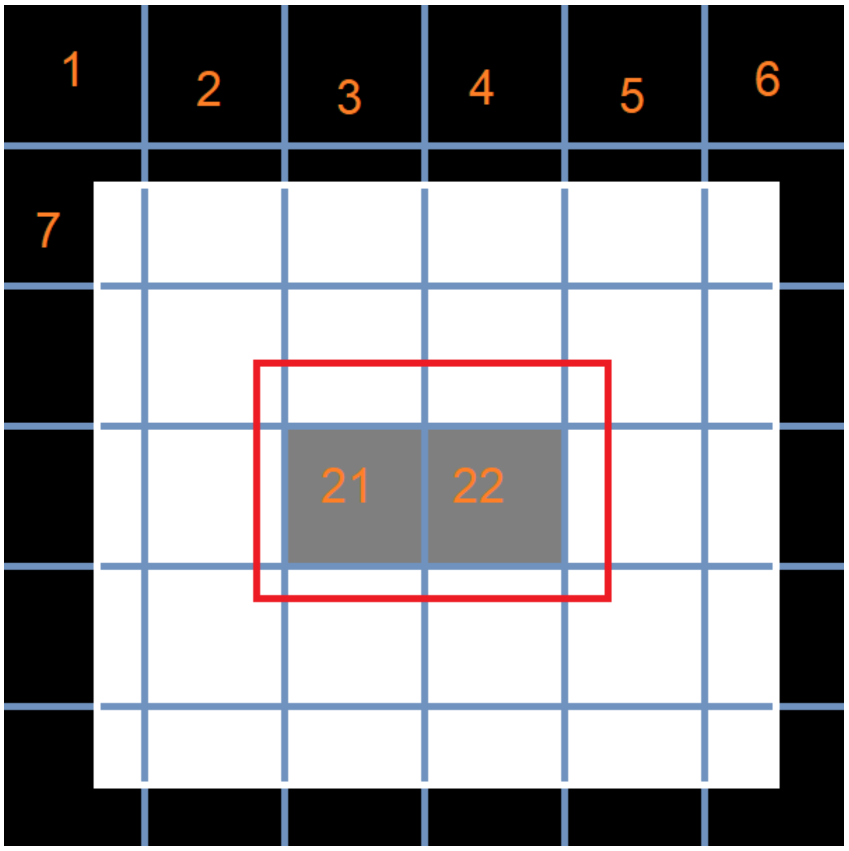


Figure 3.2: Given an input image and labels 0 (background) and 1 (foreground), it is divided into a 6-by-6 grid. The grids are then ordered according to the orange number in each cell (some numbers are omitted to make the image clear, but one should be able to interpolate all grid numbers). The white rectangle represents a ground-truth mask, and the red box its center region. Only two grids fall within this center region, cells 21 and 22, marked in grey. This means that $\boldsymbol{p}^*_{21} = \boldsymbol{p}^*_{22} = 1$ and $\boldsymbol{p}^*_k = 0$ for the other cells and $N_{pos} = 2$. Each of these two cells will produce a mask, $\boldsymbol{m}_{21}$ and $\boldsymbol{m}_{22}$ respectively. Since the positive samples fall in the same center region, $\boldsymbol{m}^*_{21} = \boldsymbol{m}^*_{22}$ which represents the entire ground-truth mask.

3.5 SOLOv2

In its article, SOLO managed to produce equal results, in terms of precision and inference speed, to that of Mask R-CNN with *potential* to be faster. Wang et al. later published SOLOv2 [13], an article describing three major bottlenecks for SOLOs performance and solving them with minimal precision loss. The improved inference speed makes SOLOv2 a suitable model for the task at hand.

Chapter 4

Methods

In this chapter I present how a dataset was created, which parameters that the Mask RCNN and SOLO models used during training, how these networks will be evaluated and the underlying theory. I also discuss how the resulting output image can be processed in order to improve the results.

4.1 Creating the dataset

4.1.1 Manual annotation

For this particular application there were no useful public datasets available. Therefore one had to be built manually. With images of SELMA operating at runtime, provided by OP Teknik, and personally crafted Python scripts, it was possible to automatically filter away images containing no information, i.e. images with no objects. The next step was to select images representative of the task at hand, meaning images that contain several adjacent objects. The selected images were then annotated using VGG Image Annotator [14], and the annotations were formatted according to its own format. A total of 682 images were included in the final dataset and split into training/validation according to 0.8/0.2.

4.1.2 Image augmentation

With a dataset constructed, it is possible to expand it using relatively trivial methods such as rotating, rescaling etc commonly called image augmentation. Performing such augmentations is usually good as it leads to more training data and better generalization. Image augmentation can either be performed before model training and incorporated in the dataset or during. Performing the augmentations beforehand allows for different models to be compared as they have all used the same training data. This also allows for hand-tuning of the hyperparameters, which could mean better results as well as overfit. This can be prevented by augmenting at runtime at the cost of making comparisions more difficult. Augmenting at runtime also has the advantages of using less memory and does not make the context of the dataset more specific. Which method of augmentation to use depends on the situation and for this thesis it was performed at runtime. This is to avoid overfitting and the consequences it can cause if the company chooses to implement a model.

4.2 Mask RCNN

For Mask RCNN I constructed models using ResNet18/34/50/101-FPN and DenseNet63/121-FPN backbones. For each model underwent training twice, one with images rescaled to 448 by 448 pixels and the other with images rescaled to 600 by 600 pixels. That makes 12 the total number of Mask RCNN models. Each model was trained for 20000 iterations with 2 images per batch with a learning rate of 0.001 with a linear warmup schedule for the first 1000 iterations. The momentum was set to 0.9. The DenseNet models had 64 initial features and the drop rate was set to 0.2. DenseNet63 used a (3, 6, 12, 8) block configuration, and DenseNet121 used a (6, 12, 24, 16) block configuration. The weights in the backbone were unfrozen, as early experiments showed that a frozen backbone produced worse models.

4.3 SOLOv2

I constructed 12 SOLOv2 models in the same manner as for Mask RCNN. Hyperparameters that are shared with Mask RCNN were set to the same values. The hyperparameters that are specific to SOLOv2 were set according to the article [13].

4.4 Evaluation methods

Next we present a number of evaluation metrics that measure how well the predicted masks compare to the ground-truth masks, like which pixels were correctly separated from the background, and how many predicted masks correspond to an object.

4.4.1 Intersection-over-Union

Intersection-over-Union, or Jaccard index, is a statistic for measuring the similarity of two sets, defined [15] as

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}, \tag{4.1}$$

for two sets A and B. Clearly $0 \leq J(A,B) \leq 1$, and a larger value represents a better result. With words, Equation (4.1) is the ratio of the size of the intersection of sets A and B and the size of the union of A and B. The Jaccard index can be visualized as shown in Figure 4.1.

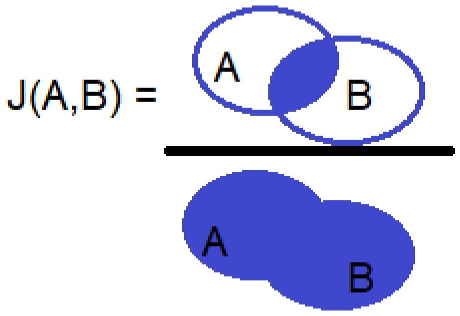


Figure 4.1: Visualization of Intersection-over-Union.

In this project, the set A is the binarized mask of the ground-truth segmentation and the set B is the binarized mask of the predicted segmentation.

4.4.2 AP-metrics

Since the intended application of a segmentation is for a robot to pick out the objects, the Jaccard index is not enough to evaluate the quality of a method. After all, what might be a single object in the ground-truth segmentation could be segmented as several objects in the prediction while still having a perfect Jaccard index. Likewise, two adjacent objects in the ground-truth segmentation could be predicted to just be one object while still having a perfect Jaccard index. It is then clear why a measurement of how many "adequately" segmented objects is needed, and why it is not as trivial as just counting the number of segmented objects. A way of quantitatively measuring the number of adequately segmented objects is by using *average precision*.

Since the models described in earlier sections have been evaluated on the COCO-dataset, it is reasonable to utilize the same evaluation metrics [16] for our problem. Although the metric is described for bounding boxes on the COCO website, Arnab et al. propose an analogue metric for instance segmentation by just using the predicted/GT masks in place of bounding boxes [17]. These metrics provide insight in how well the model handles objects of different sizes and how well it works when we only consider results with a minimum IoU threshold. In order to understand what this means we must first introduce the concept of precision and recall.

Everingham et al. [18] uses the following definitions for true/false classifications in computer vision. Let t be a threshold, $0 \leq t \leq 1$, and let a predicted mask be considered *True Positive* (TP) if it has a IoU-score greater or equal to t with atleast one ground-truth mask and *False Positive* (FP) if it does not. *True Negative* (TN) and *False Negative* (FN) are defined in the same way but with predicted mask and ground-truth mask interchanged, e.g a ground-truth mask is considered TN if it has a IoU-score $\geq t$ with atleast one predicted mask.. The precision and recall is then defined as [19]

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

The *average precision* (*AvgP[t]*) is then calculated as the area under the precision-recall curve for a given threshold t. Let r, $p(r)$ represent recall and the corresponding precision value and this can be formulated as

$$AvgP[t] = \int_0^1 p(r)dr.$$

The order in which the masks are evaluated can influence the precision-recall curve, so we instead use the interpolated precision

$$p_i(r) = max_{R \geq r}\ p(R)$$

to reduce this variation. Following the naming and sampling convention of COCO, the *interpolated average precision* ($AP[t]$) is calculated as the area under the precision-recall curve at 100 equally spaced recall values for a given threshold t. Finally, the COCO definition of AP is

$$AP = \frac{1}{10} \sum_{i=0}^{9} AP[50 + 5i].$$

The AP can be further specified to only account for objects of certain area, meaning the number of pixels in mask. This is useful since large objects have the largest "value", which implies more recycled material/more money. The default categories are

- AP^{small}, for objects with area $< 32^2$.
- AP^{medium}, for objects with $32^2 <$ area $< 96^2$.
- AP^{large}, for objects with area $> 96^2$.

reflecting a 41/34/24 distribution of object sizes in the COCO dataset.

It is common to see mean average precision (mAP) mentioned in literature and used interchangeably with AP. Mean average precision is calculated as the AP averaged over all classes and the difference between mAP and AP is often clear from context. Since our case has only one class, $mAP = AP$.

4.4.3 Time required to segment

The previously discussed measurements only treat the quality of the segmentations produced by the models. It is important for the company to know whether such a model could be implemented without changes in other places of the sorting process. One important aspect, from the whole system perspective, is that of inference time required for the segmentation. A reasonable measurement for the time required to segment is to take the average time it takes to segment a large set of images on a specific hardware setup. As SELMA takes a photograph of the conveyor belt every 0.3 seconds, a good average time to segment should be small enough so that SELMA can continue to operate at this speed. In other words, no individual segmentation should take more than 0.3 seconds. One of the main variables that influence this metric is the image size, instances are inferenced more quickly on a smaller image than a larger one but resizing an image also takes time. It is then plausible that an image could be scaled down, predicted on, and scaled up at a faster rate with minimal accuracy loss than performing the prediction without this pre/post-processing. Thus, the time required to segment is divided into two parts, the time required to predict directly and the time required to rescale and predict.

4.5 Non-maximum suppression

One problem Wang et al. discuss [9][13] is duplicate predictions of the SOLO architecture. Without any post-processing, SOLO produces far too many masks which greatly overlap. An example can be seen in Figure 4.2, which contains 18 masks but it is only possible to distinguish 3.

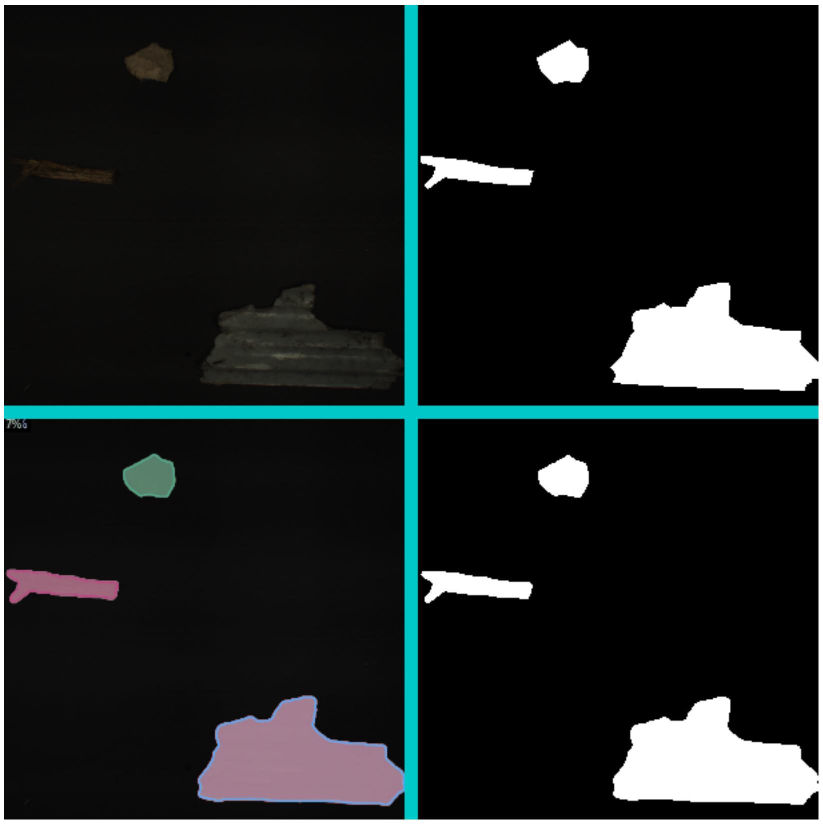
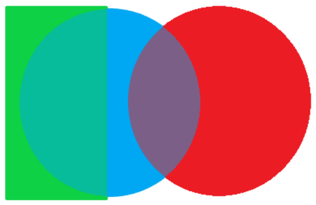


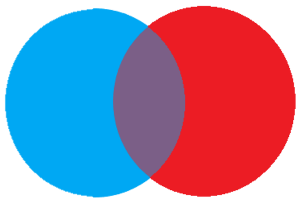
Figure 4.2: Results of a SOLOv2 model without using NMS, the prediction was the best one with regards to IoU. IoU-score: 0.964, Nr instances: 18, GT objects: 3. Top left is the input image. Top right is the ground-truth polygons in a binarized mask. Bottom left is the predicted instance masks overlayed on the input image, each color represents one mask. Bottom right is all masks binarized for easy comparison to ground-truth.

The consequences of having those redundant masks, besides a longer processing time, is that SELMA will be unable to prioritize which objects to sort first. The solution to this problem is called *non-maximum suppression* (NMS) which is used in several computer vision tasks and even in Mask RCNN. NMS works by choosing a threshold, usually 0.5, and sorting the masks by decreasing confidence. The IoU between the mask with highest confidence and all other masks is then computed and if this value is larger than the threshold, the two masks are deemed to represent the same object and the lower confidence mask is suppressed. This process is repeated for the remaining masks.

The NMS algorithm described above returns results with minimal accuracy loss but is quite slow, since it is scales quadratically and is not possible to vectorize. The algorithm Fast NMS, proposed by Bolya et al. [20] relaxes the conditions required to suppress a mask in such a way that it is possible to vectorize. The effect is a faster algorithm that removes a few too many masks, decreasing the accuracy. Early experiments showed that Fast NMS was necessary in order for SOLOv2 to predict the right number of masks in reasonable time.

While traditional NMS and Fast NMS are commonly used in computer vision tasks, they pose a problem in this context and especially when using SOLOv2. The problem is that the remaining non-suppressed masks can still share pixels, which would make it harder for the classification algorithm. After all, if some pixels are shared between two masks, at least one of those masks must contain information about a second object, possibly of a different material. That is why I developed modified algorithms of traditional NMS and Fast NMS, where the Fast NMS version is only approximately 60ms slower than the original. I call these algorithms Pixelwise NMS and Fast Pixelwise NMS. The difference between traditional NMS and Pixelwise NMS is that when two masks have an overlap but not enough to warrant a suppression, the pixels in the overlap are reserved for the higher confidence mask. The difference is visualized in Figure 4.3.


(a) Possible segmentation without NMS.


(b) Effects of traditional NMS.

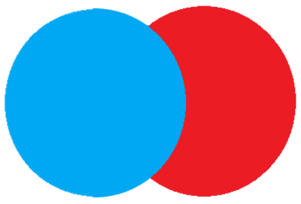

(c) Effects of Pixelwise NMS.

Figure 4.3: Comparisons of different NMS-algorithms.

Theoretically, an NMS-algorithm can either increase or decrease AP-scores, but due to the amount of suggested masks it is far more likely to decrease the AP-scores.

Code for Pixelwise NMS and Fast Pixelwise NMS is provided in the appendix.

Chapter 5

Implementation details

In this chapter I present the specific way the models were implemented and issues that occurred and were consequently fixed.

5.1 Implementation details

The selected language for programming in this thesis was Python 3, as the many libraries related to machine learning and simple syntax was expected to reduce the time spent programming. The Detectron2 [21] library with its extension AdelaiDet [22] was utilized for its easy to use framework and well-documented code meant that experiments could be performed quickly and new models could be introduced with little effort if necessary. Additionally, having all models constructed in the same framework makes inference times more comparable, as different frameworks have different throughput[1].

The annotation format produced by VGG Image Annotator is not the same format that is used in Detectron2. A Python function to convert the VGG format accordingly is provided in the Detectron2 Beginner's Tutorial [23].

Furthermore, Detectron2 provides commonly used evaluation metrics. An *int* type conversion implemented on two lines in the library pycocotools was necessary to solve some issues relating to a newer numpy version.
Detectron2 is not supported on Windows, so a workaround was used [24].

Pixelwise NMS was implemented by initializing a matrix of ones of equal size of an image, and setting elements to zero when the corresponding pixel is reserved. Then elementwise multiplication can be used to truncate masks, which is quickly done on a GPU.

[1] https://detectron2.readthedocs.io/en/latest/notes/benchmarks.html

Densenet was implemented by adapting the authors code, provided here[2], to the Detectron2 framework. Code is provided in the appendix.

[2]https://github.com/gpleiss/efficient_densenet_pytorch

Chapter 6

Results

In this chapter I present the results of different models according to the evaluation metrics described in Chapter 4, Intersection-over-Union, AP and time required to segment.

All training and testing was performed on a GTX 980 TI. Since SELMA operates using a similar graphics card, this makes the time to segment a good measurement of whether implementation is possible. Experiments were performed on images rescaled to 448x448 and 600x600 pixels. Attempts were made on 1200x1200 pixels but the GPU memory was insufficient for images of this size. Time measurements were all averaged over 5 runs, unless otherwise specified. The final training dataset consisted of 545 images and the final validation dataset consisted of 137 images.

6.1 Baseline results

OP Tekniks current algorithm was evaluated on both the final training and validation dataset, the results can be seen in Table 6.1.

Model	Avg IoU	AP	AP50	AP75	APs	APm	APl
OP Teknik Algos	55.86	23.88	50.38	20.17	0	3.3	42.28

Table 6.1: Results from using OP Tekniks algorithms.

6.2 Mask RCNN results

All Mask RCNN models were trained on the final training dataset and evaluated on the validation dataset. Models with a ResNet backbone was initialized with pretrained weights on ImageNet, other backbones were initialized randomly. The IoU and AP results can be seen in Table 6.2.

Backbone	resize (pixels)	Avg IoU	AP	AP50	AP75	APs	APm	APl
ResNet18-FPN	448	87.94	63.94	86.60	74.80	0	54.09	70.07
ResNet18-FPN	600	88.13	65.86	88.35	77.31	0	58.44	70.47
ResNet34-FPN	448	88.07	64.77	87.76	76.10	4.49	53.70	71.30
ResNet34-FPN	600	88.73	65.78	87.47	77.31	3.37	56.09	71.80
ResNet50-FPN	448	90.18	70.43	91.11	**82.92**	0	58.41	77.23
ResNet50-FPN	600	**90.60**	**71.10**	**91.23**	82.37	0	**60.59**	**77.29**
ResNet101-FPN	448	90.10	69.59	90.67	81.40	8.42	55.55	77.01
ResNet101-FPN	600	90.47	70.26	89.97	81.17	**14.31**	58.75	77.10
DenseNet63-FPN	448	50.27	33.22	51.28	39.30	0	27.56	36.77
DenseNet63-FPN	600	15.14	45.96	63.73	55.64	0	39.28	70.70
DenseNet121-FPN	448	39.67	52.98	73.39	63.01	0	45.03	58.20
DenseNet121-FPN	600	42.23	38.60	54.49	46.46	0	33.79	41.90
OP Teknik Algos	-	55.86	23.88	50.38	20.17	0	3.3	42.28

Table 6.2: IoU and AP results of different Mask RCNN models. The best score in each category is written in bold. The baseline results are written on the bottom row.

The average time to segment is divided in two parts, average inference time and average prediction time. The average inference time is the average time it takes for an image from the validation dataset to be processed through the network. The average prediction time also includes the time it takes to rescale an image. The results can be seen in Table 6.3.

Backbone	resize (pixels)	Avg inference time (ms)	Avg prediction time (ms)
ResNet18-FPN	448	**53**	**83**
ResNet18-FPN	600	63	93
ResNet34-FPN	448	58	87
ResNet34-FPN	600	69	98
ResNet50-FPN	448	64	96
ResNet50-FPN	600	81	110
ResNet101-FPN	448	74	105
ResNet101-FPN	600	98	127
DenseNet63-FPN	448	68	87
DenseNet63-FPN	600	76	114
DenseNet121-FPN	448	85	118
DenseNet121-FPN	600	88	110

Table 6.3: Resulting average inference and prediction time, averaged over 5 runs, for the different Mask RCNN models. The best score is written in bold.

A precision-recall curve of the ResNet50-FPN 600x600 pixels model with a IoU-threshold can be seen in Figure 6.1.

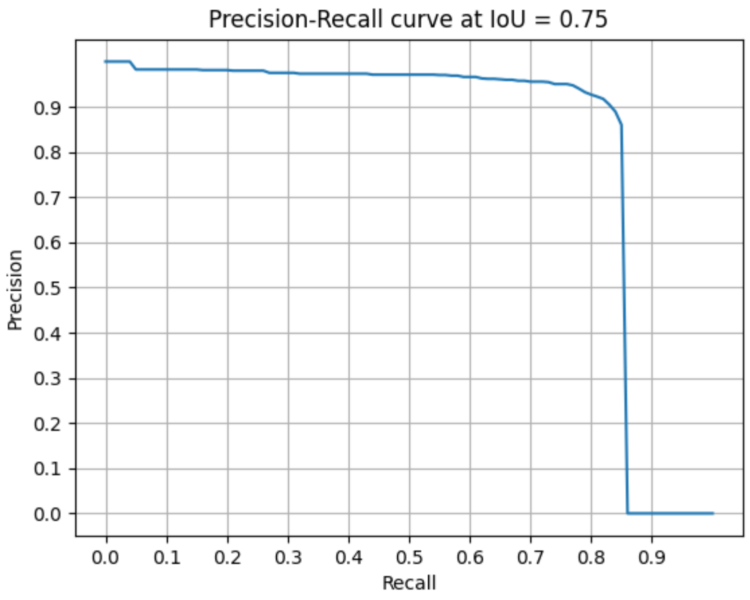


Figure 6.1: Precision recall curve for ResNet50-FPN Mask RCNN rescaled inferenced on 600 by 600 pixels

An example of a training curve for a ResNet50 Mask RCNN network can be seen in Figure 6.2.

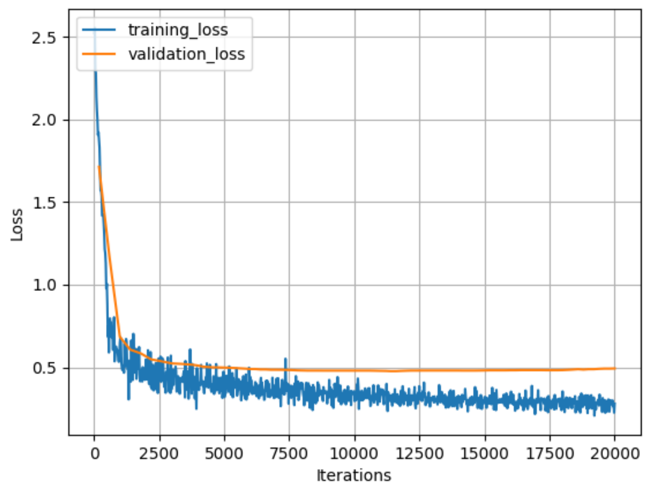


Figure 6.2: Training curve for ResNet50 Mask RCNN. The training loss was evaluated for every iteration and the validation loss was evaluated every 200 iterations.

An example of a training curve for a DenseNet63 Mask RCNN network can be seen in Figure 6.3.

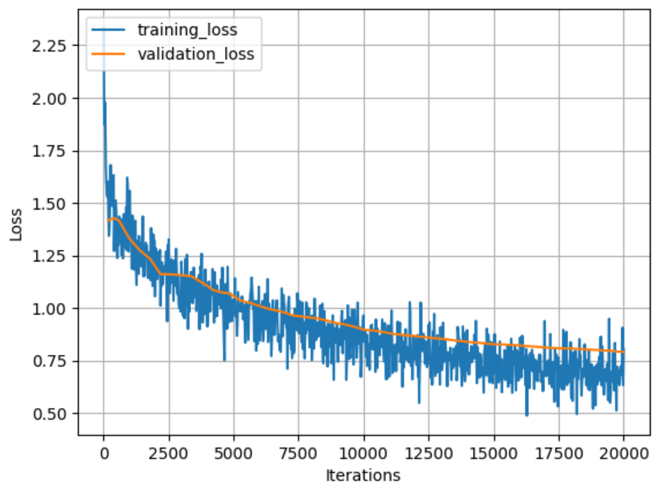


Figure 6.3: Training curve for DenseNet. The training loss was evaluated for every iteration and the validation loss was evaluated every 200 iterations.

6.3 SOLOv2 results

All SOLOv2 models were trained on the final training dataset and evaluated on the validation dataset. Models with a ResNet backbone was initialized with pretrained weights on ImageNet, other backbones were initialized randomly. The ResNet18 and ResNet34 models were trained several times and results were averaged since the optimizer often converged on predicting the entire image as one mask. The results in Table 6.4 are post-processed with traditional NMS and should serve as a baseline when comparing the accuracy-time trade-off of the other NMS methods.

Backbone	resize (pixels)	Avg IoU	AP	AP50	AP75	APs	APm	APl
ResNet18-FPN avg of 5	448	78.43	31.66	60.52	31.89	0	20.45	38.09
ResNet18-FPN avg of 5	600	87.56	55.41	81.85	62.74	0	44.61	62.01
ResNet34-FPN avg of 5	448	87.15	57.10	85.24	66.65	0	44.81	64.68
ResNet34-FPN avg of 5	600	88.28	54.23	81.66	61.75	0	43.17	61.11
ResNet50-FPN	448	88.15	64.93	**91.54**	77.65	0	50.55	73.46
ResNet50-FPN	600	89.14	67.46	91.35	**80.80**	0	**54.18**	75.61
ResNet101-FPN	448	88.29	64.16	90.95	76.79	0	49.52	72.76
ResNet101-FPN	600	**89.33**	**67.65**	91.13	79.76	**3.37**	53.70	**76.05**
DenseNet63-FPN	448	59.10	15.97	34.34	13.38	0	5.18	23.01
DenseNet63-FPN	600	29.08	6.97	15.06	5.88	0	3.14	9.67
DenseNet121-FPN	448	38.29	13.63	27.64	12.08	0	2.53	20.59
DenseNet121-FPN	600	64.64	23.84	53.20	30.16	0	24.03	33.83
OP Teknik Algos	-	55.86	23.88	50.38	20.17	0	3.3	42.28

Table 6.4: Results of SOLOv2 with traditional NMS. The baseline results are written on the bottom row.

The average prediction and processing time and AP-scores for Pixelwise NMS and Fast Pixelwise NMS can be seen in Tables 6.5 and 6.6 respectively.

Backbone	resize (pixels)	Time (ms) Pixelwise	AP	AP50	AP75	APs	APm	APl
ResNet18-FPN	448	219	30.30	56.96	31.51	0	18.80	36.86
ResNet18-FPN	600	192	54.35	78.68	61.99	0	44.51	60.52
ResNet34-FPN	448	191	43.14	70.36	47.24	0	29.97	50.37
ResNet34-FPN	600	184	53.31	77.90	62.51	0	41.55	60.20
ResNet50-FPN	448	**160**	63.82	88.81	76.87	0	49.28	72.44
ResNet50-FPN	600	182	66.65	**89.46**	**80.02**	0	**53.59**	74.89
ResNet101-FPN	448	176	63.48	89.21	76.03	0	48.85	71.82
ResNet101-FPN	600	206	**66.67**	89.25	78.94	**1.12**	52.34	**75.39**
DenseNet63-FPN	448	241	14.13	30.42	12.39	0	4.85	20.44
DenseNet63-FPN	600	254	6.22	13.19	5.36	0	3.01	8.56
DenseNet121-FPN	448	266	12.27	24.23	10.94	0	2.04	18.63
DenseNet121-FPN	600	266	26.00	45.51	26.33	0	20.36	30.23

Table 6.5: Results of SOLOv2 with Pixelwise NMS.

Backbone	resize (pixels)	Time (ms)/ Fast Pixelwise	AP	AP50	AP75	APs	APm	APl
ResNet18-FPN	448	**89**	30.47	56.95	31.66	0	18.98	36.86
ResNet18-FPN	600	99	54.51	79.59	62.08	0	44.60	60.55
ResNet34-FPN	448	94	43.15	69.64	47.44	0	30.19	50.49
ResNet34-FPN	600	101	53.35	77.97	62.54	0	41.85	60.19
ResNet50-FPN	448	95	63.81	88.81	76.87	0	49.41	72.62
ResNet50-FPN	600	110	**66.84**	**89.49**	**80.03**	0	**53.67**	74.99
ResNet101-FPN	448	106	63.48	89.20	76.03	0	48.93	71.88
ResNet101-FPN	600	128	66.73	89.28	79.72	**1.12**	52.26	**75.48**
DenseNet63-FPN	448	107	14.29	30.85	12.42	0	4.91	20.48
DenseNet63-FPN	600	126	6.29	13.24	5.37	0	3.02	8.59
DenseNet121-FPN	448	136	12.32	24.24	11.08	0	2.09	18.53
DenseNet121-FPN	600	136	26.07	45.38	27.05	0	20.43	30.04

Table 6.6: Results of SOLOv2 with Fast Pixelwise NMS.

Chapter 7

Discussion, conclusions and future work

In this section I compare and discuss the results and which conclusions that can be drawn from them. I also suggest different options to continue this work.

7.1 Baseline results

The results of OP Teknik's current algorithms can be seen in Table 6.1. At first glance, these results seem poor. But when the dataset was created, I mainly selected images with adjacent or overlapping objects. It was common for the corresponding masks to be of poor quality, which is the reason this thesis was offered in the first place. The current algorithms work well in cases of large, isolated components, so the results of the algorithms should not be seen as an evaluation of its quality but rather a baseline to compare the other methods with.

Although the company does not wish to reveal the speed of their algorithms, it can be assumed to be faster than 50ms per frame for the sake of discussion.

7.2 Mask RCNN

It is easy to see from Table 6.2 that for every network with a ResNet backbone, a larger input image size produces better results. This comes as no surprise, since an image resized to 448x448 pixels has lost more information than one resized to 600x600 pixels. It can also be seen that IoU- and AP-scores increase with depth for the ResNet backbones, up to ResNet50. For the two networks with a ResNet101 backbone, the results are slightly worse than the ones with a ResNet50 backbone, implying that more than 50 layers has little effect on the accuracy of the models. The most striking difference between 50 and 101 layers is the APs-score. The increased depth seems to make the network better at predicting masks across scales. But since APs is for objects smaller than 32x32 pixels, it is hard to justify the increased processing time in order to recycle very little extra material. All ResNet models outperform the current algorithms when it comes to IoU and AP but no model is faster than the baseline.

The DenseNet backbones produce varying results, with no clear trend to distinguish. One explanation for this difference compared to the ResNet backbones is that the ResNet weights were pretrained. This is clear from the loss curves in Figures 6.2 and 6.3. The ResNet50 model is quickly overfit after just a few thousand iterations, while the validation loss for the DenseNet model follows the training loss quite well for all 20000 iterations. It is then possible for a DenseNet network produce on-par or even better results if training is performed under the same conditions of the ResNet network.

Due to time and resource restrictions, such a training has not been attempted.

Overall, a ResNet50-FPN backbone and resizing images to 600x600 pixels produces the best results in 5 out of 7 categories.

7.3 SOLOv2

The results in Table 6.4 show that the ResNet models gain significantly increased AP-scores with increased depth, up to 50 layers. These AP-scores however, are worse than the corresponding Mask RCNN models and are outclassed with the ResNet18 and ResNet34 backbones. This suggests that SOLOv2 is an overengineered model that is capable of outperforming Mask RCNN in specific cases such as the COCO challenges. This is certainly a possible explanation, supported by the quote in the SOLO article:

”*We start by rethinking a question: What are the fundamental differences between object instances in an image? Take the challenging MS COCO dataset [16] for example. There are in total 36,780 objects in the validation subset, 98.3% of object pairs have center distance greater than 30 pixels. As for the rest 1.7% of object pairs, 40.5% of them have size ratio greater than 1.5×. To conclude, in most cases two instances in an image either have different center locations or have different object sizes. This observation makes one wonder whether we could directly distinguish instances by the center locations and object sizes?*”

Nonetheless, there are still interesting observations to make, such that larger input images provide greater AP-gains for SOLOv2 than their respective Mask RCNN counterparts, except for ResNet34. As described in Chapter 4.3, there were no hyperparameters altered with regards to this dataset so it is possible that the object instances in a larger image better match the underlying idea of

SOLO.

When it comes to different NMS methods, both Pixelwise and Fast Pixelwise NMS produce worse results than traditional NMS, which was expected. What is interesting is that Pixelwise NMS and Fast Pixelwise NMS produce very similar results, and no method is superior in all categories when it comes to AP, which can be seen in Tables 6.5 and 6.6. The methods do behave differently when it comes to prediction time. Contrary to what is expected, the prediction time is greater for smaller ResNet networks when Pixelwise NMS is used. This is explained by noting that Pixelwise NMS has quadratic time complexity with regards to the number of masks and that the poor performance of the shallow networks is partly due to a large number of predicted masks, see Figure 4.2. As the models improve with increased backbone depth, the inference speed is slower and fewer masks are predicted, thus a faster post-processing time. Both of these factors mean that the overall prediction time mainly depends on the inference speed. The prediction time behaves similar to that of Mask RCNN when Fast Pixelwise NMS is used. This is because Fast Pixelwise NMS consists only of vectorized operations, meaning that the post processing time is insignificant when compared to the inference speed.

The SOLOv2 models with DenseNet backbones give no new information regarding the possible performance of using such a backbone. As such, the conclusions regarding DenseNet in the Mask RCNN discussion still stands. Instead of repeating what was said, see Chapter 7.2.

The SOLOv2 models perform worse than the corresponding Mask RCNN models. The results do hint that a SOLOv2 model could outperform Mask RCNN with with hyperparameters adjusted to the given dataset.

7.4 Future work

The results show that a ResNet50 backbone produces the best Mask RCNN and SOLOv2 model. It would be interesting to see just how good one could make these models by adjusting the hyperparameters in accordance with the underlying theory.

Although the DenseNet models were noticeably slower than the ResNet models, they should in theory be able to produce better results. One could attempt this either by configuring a good training scheme or by finding a way to convert pretrained DenseNet models to Detectron2.

As mentioned in the Introduction, SELMA performs classification on the segmented masks. Although this thesis was focused on single class instance segmentation, all models are able to support classification. Performing classification would most likely mean a slightly decrease in prediction speed for the model but overall time would be saved. Since this would replace the entire sorting process, it could be possible to make time gains elsewhere, e.g smaller camera resolutions to avoid re-scaling images.

Chapter 8

References

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* `http://www.deeplearningbook.org`. MIT Press, 2016.

[2] Common Objects in COntext (COCO). *COCO Dataset overview.* `https://cocodataset.org/#overview`. Accessed: 14092021. 2015.

[3] Trash Annotations in COntext (TACO). *TACO Dataset.* `http://tacodataset.org/`. Accessed: 14092021. 2021.

[4] Olga Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: `10.1007/s11263-015-0816-y`.

[5] Kaiming He et al. *Deep Residual Learning for Image Recognition.* 2015. arXiv: `1512.03385 [cs.CV]`.

[6] Gao Huang et al. *Densely Connected Convolutional Networks.* 2018. arXiv: `1608.06993 [cs.CV]`.

[7] Tsung-Yi Lin et al. *Feature Pyramid Networks for Object Detection.* 2017. arXiv: `1612.03144 [cs.CV]`.

[8] Kaiming He et al. *Mask R-CNN.* 2018. arXiv: `1703.06870 [cs.CV]`.

[9] Xinlong Wang et al. *SOLO: Segmenting Objects by Locations.* 2020. arXiv: `1912.04488 [cs.CV]`.

[10] Tsung-Yi Lin et al. *Focal Loss for Dense Object Detection.* 2018. arXiv: `1708.02002 [cs.CV]`.

[11] G.B. Folland. *Real Analysis: Modern Techniques and Their Applications.* John Wiley & Sons, Inc., 1999. ISBN: 978-0-471-317-16-6.

[12] Lee R. Dice. "Measures of the Amount of Ecologic Association Between Species". In: *Ecology* 26.3 (1945), pp. 297–302. DOI: `10.2307/1932409`.

[13] Xinlong Wang et al. *SOLOv2: Dynamic and Fast Instance Segmentation.* 2020. arXiv: `2003.10152 [cs.CV]`.

[14] A. Dutta, A. Gupta, and A. Zissermann. *VGG Image Annotator (VIA)*. http://www.robots.ox.ac.uk/ vgg/software/via/. Version: 2.0.11, Accessed: 13092021. 2016.

[15] Paul Jaccard. "THE DISTRIBUTION OF THE FLORA IN THE ALPINE ZONE". In: *New Phytologist* 11.2 (1912). [Online; accessed 01-November-2021], pp. 37–50. DOI: `10.1111/j.1469-8137.1912.tb05611.x.`. URL: `https://nph.onlinelibrary.wiley.com/doi/epdf/10.1111/j.1469-8137.1912.tb05611.x`.

[16] Common Objects in COntext (COCO). *COCO Detection evaluation*. `https://cocodataset.org/#detection-eval`. Accessed: 14092021. 2015.

[17] Anurag Arnab and Philip H. S Torr. *Pixelwise Instance Segmentation with a Dynamically Instantiated Network*. 2017. arXiv: `1704.02386 [cs.CV]`.

[18] M. Everingham et al. "The Pascal Visual Object Classes (VOC) Challenge". In: *International Journal of Computer Vision* 88.2 (June 2010), pp. 303–338.

[19] Tom Fawcett. "An Introduction to ROC Analysis". In: *Pattern Recognition Letters* 27.8 (2006). [Online; accessed 01-November-2021], pp. 861–874. DOI: `10.1016/j.patrec.2005.10.010`. URL: `https://people.inf.elte.hu/kiss/11dwhdm/roc.pdf`.

[20] Daniel Bolya et al. *YOLACT: Real-time Instance Segmentation*. 2019. arXiv: `1904.02689 [cs.CV]`.

[21] Yuxin Wu et al. *Detectron2*. `https://github.com/facebookresearch/detectron2`. 2019.

[22] Zhi Tian et al. *AdelaiDet: A Toolbox for Instance-level Recognition Tasks*. `https://git.io/adelaidet`. 2019.

[23] Detectron2. *Detectron2 Beginner's Tutorial*. `https://colab.research.google.com/drive/16jcaJoc6bCFAQ96jDe2HwtXj7BMD_-m5`. Accessed: 13092021. 2019.

[24] DG Maxime. *How to easily install Detectron2 on Windows 10?* [Online; accessed 24-October-2021]. 2020. URL: `https://dgmaxime.medium.com/how-to-easily-install-detectron2-on-windows-10-39186139101c`.

Chapter 9

Appendix

Code for Densenet, Pixelwise NMS and Fast Pixelwise NMS is provided here:
https://gist.github.com/em7650sa/a9f7f78de657a929f1edace8dcc91795

Master’s Theses in Mathematical Sciences 2021:E70
ISSN 1404-6342

LUTFMA-3459-2021

Mathematics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lth.se/