

**Bachelor's Thesis**



**Czech  
Technical  
University  
in Prague**

**F3**

**Faculty of Electrical Engineering  
Department of Cybernetics**

# **Using Machine Learning and Computer Vision for Defects Detection in Manufacturing**

**Josef Losos**

**Supervisor: Ing. David Kadleček, Ph.D.**

**Study program: Open Informatics**

**Specialisation: Artificial Intelligence and Computer Science**

**May 2023**





## I. Personal and study details

Student's name: **Losos Josef** Personal ID number: **499267**  
Faculty / Institute: **Faculty of Electrical Engineering**  
Department / Institute: **Department of Cybernetics**  
Study program: **Open Informatics**  
Specialisation: **Artificial Intelligence and Computer Science**

## II. Bachelor's thesis details

Bachelor's thesis title in English:

**Using Machine Learning and Computer Vision for Defects Detection in Manufacturing**

Bachelor's thesis title in Czech:

**Využití strojového učení a počítačového vidění pro detekci vad v průmyslové výrobě**

Guidelines:

Nowadays, there is a great emphasis on reducing human resources in production and replacing them with automation. One of the important approaches is quality inspection using machine learning and computer vision, which leads both to saving human resources and to increasing the quality of inspection and its objectivity in general.

The objectives of this thesis are:

- 1) Make a survey of the types of defects that occur in different types of industrial manufacturing.
- 2) Make a survey of the leading machine learning and computer vision methods used to solve this problem.
- 3) Choose at least two defect detection methods, make them work, verify their functionality on data, and compare them.
- 4) For the most appropriate method, perform model fine-tuning.
- 5) For the trained model, objectively evaluate its quality on the provided data and evaluate the results.

For the purpose of the bachelor thesis, a subset of existing annotated images of inserts used for metal machining will be provided. This dataset will be provided at the start of the bachelor thesis.

Bibliography / sources:

- [1] Park, S.-S.; Tran, V.-T.; Lee, D.-E. Application of Various YOLO Models for Computer Vision-Based Real-Time Pothole Detection. Appl. Sci. 2021, 11, 11229. <https://doi.org/10.3390/app112311229>
- [2] Fei-Fei, L., Jiajun, W., Ruohan, G., Stanford University CS231n: Deep Learning for Computer Vision, 2022
- [3] LeCun, Y., Bengio, Y. & Hinton, G. Deep learning. Nature 521, 436–444, 2015.
- [4] Jiadong, C., Wei, L., Xiang, W., Jingjing, F., Liwei, W., Rui, Z., MIAD: A Maintenance Inspection Dataset for Unsupervised Anomaly Detection Tianpeng Bao, 2022.

Name and workplace of bachelor's thesis supervisor:

**Ing. David Kadlec, Ph.D. Centre for Knowledge Management FEE**

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **01.02.2023** Deadline for bachelor thesis submission: **26.05.2023**

Assignment valid until: **22.09.2024**

Ing. David Kadlec, Ph.D.  
Supervisor's signature

prof. Ing. Tomáš Svoboda, Ph.D.  
Head of department's signature

prof. Mgr. Petr Páta, Ph.D.  
Dean's signature

### III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

\_\_\_\_\_  
Date of assignment receipt

\_\_\_\_\_  
Student's signature

## Acknowledgements

I thank my supervisor for the topic he had chosen for me and for pieces of advice from his co-worker Ing. Tomáš Paleček, thanks to whom I successfully did my thesis.

Computational resources were provided by the e-INFRA CZ project (ID:90140), supported by the Ministry of Education, Youth and Sports of the Czech Republic.

## Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, May 26, 2023

Losos Josef

## Abstract

This work aimed to analyze the types of defects on metallic surfaces, to search for the leading object detection methods that could be suitable for detecting these defects, and to find out which method is more suitable. DNAI and Severstal datasets were used for the evaluation task. The first dataset is from a DNAI client, with images taken from a production line. The second dataset is from a Kaggle competition. YOLO and U-net were selected for comparison. The YOLO model performs better than the U-net model on both datasets. The main difference was in the number of false positives. On the Severstal dataset, the YOLO model achieved a recall of 90 % and a precision of 69.3 % with the benevolent metric. On the DNAI dataset, the results were 72.2 % recall and 70.5 % precision.

**Keywords:** Computer Vision, defects, object detection, YOLO, U-net

**Supervisor:** Ing. David Kadleček, Ph.D.

## Abstrakt

Cílem této práce bylo analyzovat typy defektů na kovových površích, provést rešerši hlavních metod detekce objektů, které by mohly být vhodné pro detekci těchto vad a zjistit, která metoda je vhodnější. Pro vyhodnocení byly použity datasety DNAI a Severstal. První dataset je od zákazníka DNAI, snímky přímo z produkce. Druhý dataset je z Kaggle soutěže. Metody YOLO a U-net byly vybrány pro porovnání. Model YOLO je na obou datech úspěšnější než model U-net. Hlavním rozdílem byl počet falešně pozitivních predikcí. Na datasetu Severstal dosáhl model YOLO s benevolentní metrikou recall 90 % a přesností 69,3 %. Na datasetu DNAI byly výsledky 72,2 % recall a 70,5 % přesnost.

**Klíčová slova:** počítačové vidění, vady, detekce objektů, YOLO, U-net

**Překlad názvu:** Využití strojového učení a počítačového vidění pro detekci vad v průmyslové výrobě

# Contents

<b>1 Introduction</b>	<b>1</b>	5.2 Conversion of the CSV annotations to RGB mask . . . . .	30
<b>2 Theoretical background</b>	<b>3</b>	5.3 Conversion of the annotations from the RGB mask to the COCO . . . . .	30
2.1 Computer Vision techniques . . . . .	3	5.4 Conversion of the annotations from the COCO to the YOLOv5 format . . . . .	31
2.1.1 Image Classification . . . . .	3	5.5 Conversion of the annotations from the COCO to the CSV format . . . . .	31
2.1.2 Object Detection . . . . .	3	5.6 How to train and predict with YOLOv5 model . . . . .	31
2.1.3 Segmentation . . . . .	4	5.6.1 How to train . . . . .	31
2.2 Annotation formats . . . . .	5	5.6.2 How to run inference . . . . .	32
2.2.1 COCO format . . . . .	5	5.7 Viewing the dataset using FiftyOne . . . . .	33
2.2.2 YOLOv5 format . . . . .	6	<b>6 Severstal dataset</b>	<b>35</b>
2.2.3 Severstal format . . . . .	6	6.1 Comparison of methods . . . . .	35
2.3 Evaluation metrics for object detection . . . . .	7	6.1.1 U-net . . . . .	35
2.3.1 IoU . . . . .	7	6.1.2 YOLO . . . . .	37
2.3.2 Precision (P) and Recall (R) . . . . .	7	6.1.3 Summary comparison . . . . .	37
2.3.3 Average Precision (AP) . . . . .	8	6.2 Fine-tuning . . . . .	40
2.3.4 Mean Average Precision (mAP) . . . . .	8	6.2.1 Data preparation . . . . .	40
2.3.5 F1 score . . . . .	8	6.2.2 Training and prediction parameters . . . . .	41
2.3.6 Benevolent metric . . . . .	9	6.2.3 Baseline model using no technique . . . . .	41
2.3.7 Number of detected defects . . . . .	9	6.2.4 Slicing to 256 pixels sized slices . . . . .	42
<b>3 Defect types in manufacturing</b>	<b>11</b>	6.2.5 Minimal area ratio of defect . . . . .	43
3.1 DNAI dataset . . . . .	13	6.2.6 Dataset with/without negative samples . . . . .	44
3.1.1 Jagged edge . . . . .	13	6.2.7 Slicing to 400 pixels width slices . . . . .	45
3.1.2 Material Remains . . . . .	14	6.2.8 Slicing to 800 pixels width slices . . . . .	46
3.1.3 Material on the edge . . . . .	15	6.2.9 Using a pre-trained model or random initialization . . . . .	49
3.1.4 Crack . . . . .	16	6.2.10 Pre-trained model size . . . . .	50
3.1.5 Extra material glued . . . . .	17	6.2.11 Using various confidence threshold . . . . .	50
3.1.6 Shipping defects . . . . .	18	6.2.12 Comparison of fine-tuning measurements . . . . .	52
3.2 Severstal dataset . . . . .	19	6.2.13 Evaluation of results . . . . .	53
3.2.1 Dimple . . . . .	19	<b>7 DNAI dataset</b>	<b>55</b>
3.2.2 Scratch . . . . .	19	7.1 Fine-tuning . . . . .	55
3.2.3 Abrasion . . . . .	20	7.1.1 Data preparation . . . . .	56
3.2.4 Blister . . . . .	20	7.1.2 Training and prediction parameters . . . . .	56
<b>4 Object detection methods</b>	<b>21</b>		
4.1 YOLO . . . . .	21		
4.1.1 YOLOv1 . . . . .	22		
4.1.2 YOLOv2 . . . . .	23		
4.1.3 YOLOv3 . . . . .	24		
4.1.4 YOLOv4 . . . . .	24		
4.1.5 YOLOv5 . . . . .	25		
4.1.6 YOLOv8 . . . . .	25		
4.2 Faster R-CNN . . . . .	26		
4.3 SSD . . . . .	27		
4.4 U-net . . . . .	28		
<b>5 Necessary prerequisites</b>	<b>29</b>		
5.1 Selection of methods used . . . . .	29		

7.1.3 Baseline model .....	57
7.1.4 Slicing to 1280 pixels .....	58
7.1.5 Slicing to 640 without overlap	59
7.1.6 Slicing to 640 with overlap ..	60
7.1.7 Dataset with negative samples	61
7.1.8 Minimal area ratio of defect .	63
7.1.9 Pre-train model size .....	64
7.1.10 Using random initialization or pre-trained model .....	64
7.1.11 Using various confidence thresholds .....	65
7.1.12 Comparison and summary of measurements .....	67
7.1.13 Evaluation of results .....	70
7.2 Comparison of methods .....	71
7.2.1 U-net.....	71
7.2.2 YOLO.....	72
7.2.3 Summary comparison.....	73
<b>8 Conclusion</b>	<b>77</b>
<b>A Bibliography</b>	<b>79</b>
<b>B Abbreviations</b>	<b>83</b>
<b>C User guide</b>	<b>85</b>
C.1 Code .....	85
C.2 Datasets .....	86
C.2.1 Severstal .....	86
C.2.2 DNAI .....	86

## Figures

2.1 Difference Semantic and Instance Segmentation, taken from [13]. . . . .	4	3.11 On the left side is extra material glued example. The typical placement of the extra material glued is shown in the annotation heat-map generated by Roboflow [8] on the right side. . . . .	17
2.2 COCO folder structure. . . . .	5	3.12 Extra material glued annotations distribution. . . . .	17
2.3 YOLOv5 annotation notation in .txt file. . . . .	6	3.13 A shipping defect example is on the left side. On the right side is an annotation heat-map generated by Roboflow [8], which shows the placement of shipping defects. . . . .	18
2.4 CSV annotation file example. . . . .	6	3.14 Shipping defect annotations distribution. . . . .	18
2.5 The formula for calculating precision. . . . .	7	3.15 Dimple example. . . . .	19
2.6 The formula for calculating recall. . . . .	8	3.16 Scratch example. . . . .	19
2.7 Examples of problematic predictions for IoU metric. . . . .	9	3.17 Abrasion example. . . . .	20
3.1 Graph shows the count of each defect class in the private dataset given by DNAI. . . . .	12	3.18 Blister example. . . . .	20
3.2 Representation of each defect type in the public Severstal steel dataset. . . . .	12	4.1 YOLO object detector, taken from [5]. . . . .	21
3.3 On the left side is a jagged edge example. Annotation Heatmap, generated by Roboflow [8], which shows the typical placement of the jagged edge, is on the right side. . . . .	13	4.2 Object detection sequence of YOLO, taken from [18]. . . . .	22
3.4 Jagged edge annotations distribution. . . . .	13	4.3 YOLOv1 architecture, taken from [28]. . . . .	22
3.5 On the left side is a material remains example. On the right side is the annotation Heat-map, which shows that the material remains defects can be located on the surface of any part of the product. The heat-map was generated by Roboflow [8]. . . . .	14	4.4 Faster R-CNN unified network, taken from [22]. . . . .	26
3.6 Material remains annotations distribution. . . . .	14	4.5 SSD architecture, taken from [19]. . . . .	27
3.7 On the left side is material on the edge example. Annotation heat-map, generated by Roboflow [8] on the right side shows the typical placement of the material on the edge. . . . .	15	4.6 U-net architecture, taken from [24]. . . . .	28
3.8 Material on the edge annotations distribution. . . . .	15	5.1 The annotations in CSV format from the Kaggle . . . . .	30
3.9 A crack example is on the left side. Annotation heat-map, generated by Roboflow [8] on the right side, shows the typical location of this defect. . . . .	16	5.2 On the left side of the Figure is shown the settings of category ids, and on the right side are category colors corresponding to colors in the previous step. 5.2 . . . . .	30
3.10 Crack annotations distribution. . . . .	16	6.1 Example predictions performed by the U-net model. . . . .	36
		6.2 Example predictions performed by YOLOv5 model. . . . .	37
		6.3 On the left side are predictions from YOLO, and on the right side from U-net. . . . .	38

6.4 On the left side are predictions from YOLO, and on the right side are from U-net. This Figure shows differences in false positive prediction. ....	38	7.1 Prediction example of baseline model on the private dataset.....	57
6.5 Prediction example for each defect type. ....	41	7.2 Prediction example of the model using slicing to 1 280 pixel-sized slices. ....	58
6.6 Prediction example while using slicing to 256-pixel sized slices. ...	42	7.3 Comparison of predictions for slicing to 1 280 on the left side and 640 pixel-sized slices on the right side. ....	59
6.7 Prediction examples where the left side displays prediction with the model trained on slices with 20 % minimal area of defect and on the right side are predictions made with the model trained on slices with 5 % minimal area of defect. ....	43	7.4 Comparison of predictions for 640 pixel-sized slices without overlap on the left and with an overlap of 25 % on the right side. ....	60
6.8 Left side displays the prediction performed by the model trained on data containing only positive samples, and the right side shows the prediction made by the model trained on data with negative samples. ...	44	7.5 Comparison of predictions for the training set with 30 % in the first row and keeping all available negative samples in the last row.....	62
6.9 The prediction examples while using slicing to 400-pixel width slices. ....	45	7.6 Comparison of predictions for 20 % minimal area ratio on the left and 5 % on the right side. ....	63
6.10 The prediction example shows the difference between using slicing to 400-pixel width slices on the right side and 800-pixel width slices on the left side for large defects. ....	46	7.7 Comparison of predictions using F1 max as confidence threshold on the left and confidence threshold 0.1 on the right side. ....	65
6.11 This Figure compares predictions made with slicing to 800 width slices, but on the left are predictions made without an overlap, and on the right side are predictions made with an overlap of 0.3. ....	47	7.8 Prediction example of material on edge on the left side baseline method without any improvement and on the right side using slicing to 640-pixel-sized slices with overlap.	67
6.12 Prediction example of using random initialization on the left side and using pre-trained model yolov5m.pt on the right side. ....	49	7.9 Prediction example of jagged edge on the left side baseline method without any improvement and on the right side using slicing to 640-pixel-sized slices with overlap.	68
6.13 Prediction example of using confidence threshold corresponding to F1 max on the left side and using 0.15 as confidence threshold on the right side.....	51	7.10 Example predictions performed by U-net. ....	71
		7.11 Example predictions performed by YOLOv5. ....	72
		7.12 Comparison of predictions performed by U-net on the left side and YOLO on the right side. ....	73
		7.13 Comparison of predictions performed by U-net on the left side and YOLO on the right side. ....	74



## Tables

6.1 Tables with the comparison of the number of detected defects, TP, and FP for the YOLO and U-net model in the first two tables, where the second one is for the more benevolent metric. The mAP results are described in the last table. The metrics in columns are described in Section 2.3. ....	39
6.2 Tables show the number of detected defects, TP, and FP for slicing to 256-pixel-sized slices in the first two tables. The table below shows the results of mAP. ....	42
6.3 The first two tables shows the number of detected defects, TP, and FP for different minimal area ratios of a defect. The last table shows the results of mAP. ....	43
6.4 The number of detected defects, TP, and FP for the model trained with only positive samples in the row named 'positive' and with negative samples in the row named 'negative' are shown in the first two tables. The last table shows the results of mAP. ....	45
6.5 The first two tables show the number of detected defects, TP, and FP for the model trained with 256-pixel-sized slices and 400-pixel width slices. The last table shows the results of mAP. ....	46
6.6 Tables demonstrating the number of detected defects, TP, and FP for the model trained with 400-pixel width slices and 800-pixel width slices. ....	47
6.7 The table displays the results of mAP. ....	48
6.8 The first two tables show the number of detected defects, TP, and FP for the model trained with 800-pixel width slices where one row is the case of prediction without overlap and the second one is with overlap. The third table shows the results of mAP. ....	48
6.9 The first two tables show the number of detected defects, TP, and FP for a model trained from random initialization or from pre-trained yolov5m.pt model. The table below shows the results of mAP. ....	49
6.10 The first two tables show the number of detected defects, TP, and FP for three different-sized pre-trained yolov5 models. The table below shows the results of mAP. ...	50
6.11 Table with differences between results predicted with various confidence thresholds. In the columns header are values of confidence thresholds that were used, where the first one is the F1 max. ....	51
6.12 Table with result metrics for the public dataset, where measurements labeled from M1 to M4 are described below. ....	52
7.1 The first two tables show the number of detected defects, TP, and FP for the baseline model. The table below shows the results of mAP. The metrics in columns are described in Section 2.3. ....	57
7.2 The first two tables show the number of detected defects, TP, and FP for the baseline model compared to using 1 280-pixel-sized slices. The table below shows the results of mAP. ....	58
7.3 Tables demonstrating the number of detected defects, TP, and FP for the model using slicing into 1 280 pixel-sized and 640 pixel-sized slices. ....	59
7.4 The table displays the results of mAP. ....	60
7.5 The first two tables show the number of detected defects, TP, and FP for the model using slicing with an overlap of 0.25 and for the model without an overlap. The table below shows the results of mAP. ....	61

7.6 The first two tables show the number of detected defects, TP, and FP for the training set with 0 %, 30 %, or keeping all available negative samples. The table below shows the results of mAP. ....	62	7.14 The first two tables are the number of detected defects, TP, and FP for the YOLO and U-net models in the first table. The mAP results are described in the last table. ....	75
7.7 The first two tables show the number of detected defects, TP, and FP for the model trained on data using 20 % or 5% of minimal area ratio. The table below shows the results of mAP. ....	63		
7.8 The first two tables show the number of detected defects, TP, and FP for medium-sized and extra-large pre-trained models. The table below shows the results of mAP. ....	64		
7.9 The first two tables show the number of detected defects, TP, and FP for the model trained from random initialization or from pre-trained yolov5m weights. The table below shows the results of mAP. ....	65		
7.10 Table with differences between results predicted with various confidence thresholds. Metrics in rows are explained in section 2.3. In the columns header are values of confidence thresholds that were used, where the first one is the F1 max. .	66		
7.11 Table with result metrics for material remains measurements labeled from M1 to M7, which are described at the end of this section.	67		
7.12 Table with result metrics for material on edge measurements labeled from M1 to M7, which are described at the end of this section.	68		
7.13 Table with result metrics for jagged edge measurements labeled from M1 to M7, which are described at the end of this section. ....	69		

# Chapter 1

## Introduction

The following work aims to utilize computer vision methods in the domain of quality inspection. The main objective is to implement an algorithm that is capable of autonomous defect detection on metallic parts. This is achieved by utilizing neural network-based computer vision techniques. The work compares a YOLO-based solution [15] with a U-net-based solution [4]. YOLO is widely used in practice for multiple object detection tasks, while U-net was one of the best solutions for the Kaggle competition[3]. The goal is to find out which of these methods provides better results. These methods are tested on two datasets, one from DNAI and the second from the Kaggle competition.

Chapter 2 provides an overview of the computer vision techniques, annotation formats, and evaluation metrics. The survey of the defect types from industrial manufacturing is provided in Chapter 3, where each subsection describes one defect type. An overview of the leading computer vision methods is provided in Chapter 4. This overview focuses on the YOLO method, which is currently one of the most widely used methods for object detection. The following Chapter 5 provides the reasons for using the YOLO and U-net methods in Section 5.1 and procedures needed to convert data between individual annotations formats or a guide on how to use YOLO in all rest sections. Chapter 6 provides a comparison of the chosen methods on the Severstal dataset in Section 6.1. First, typical predictions for both methods are described, followed by a comparison of predictions with significant differences. The subsections of Section 6.2 describe individual measurements using techniques such as slicing, overlapping slices, minimum defect area ratio, using negative samples for training, and starting training from randomly initialized weights or a pre-trained model. Subsection 6.2.13 describes the evaluation of the achieved results. Chapter 7 provides results reached on the DNAI dataset. Section 7.1 evaluates the measurements with the same model improvements as for the Severstal dataset. Section 7.2 describes the comparison of YOLO and U-net. Individual subsections contain the most common predictions for individual methods and their differences. The overall evaluation of the achieved results contains Chapter 8.



## Chapter 2

### Theoretical background

#### 2.1 Computer Vision techniques

##### 2.1.1 Image Classification

Image classification represents a basic technique in the field of Computer Vision, with the objective of assigning single or multiple categories to a given image. An image classifier has an image as input and afterward identifies various objects in the image, such as people, animals, and plants. However, image classification alone does not provide detailed information about the image content, such as the number of people or their position in the image. Other computer vision techniques are needed to get such information.

There are basically two types of image classification. binary classification and multiclass classification. As the name suggests, binary image classification looks for a single class in the given image and provides results based on whether the image has that object.

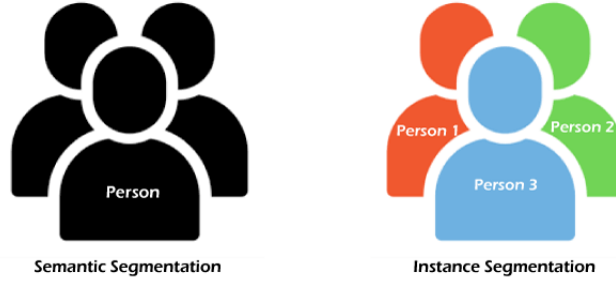
##### 2.1.2 Object Detection

Object detection is an important technique in Computer Vision that can be used either as a step after image classification or as an approach that uses image classification to detect objects in images. Object detection aims to recognize objects within the bounding boxes and identify their relevant classes. The goal is to imitate human intelligence in machines to accurately identify and locate objects. Deep learning and machine learning technologies are typically used by object detection to achieve accurate and efficient results. The goal is to imitate human intelligence in machines to accurately identify and locate objects.

Object detection has multiple uses, including object tracking, searching, video surveillance, and captioning. Various techniques can be used to perform object detection, including R-CNN, YOLO, etc.

### 2.1.3 Segmentation

As one of the most classic tasks in the Computer Vision field, image segmentation aims to label pixels into different groups according to the objects to which each pixel belongs. Image segmentation is considered a clustering problem. Earlier segmentation techniques usually used methods of splitting and merging regions. Later segmentation algorithms used consistency indicators such as intra-regional consistency and inter-regional dissimilarity. [30]



**Figure 2.1:** Difference Semantic and Instance Segmentation, taken from [13].

Segmentation can be separated into two subcategories. Semantic Segmentation classifies similar objects as the same class from the pixel levels. It gives meaning to each pixel in an image. Instance Segmentation can classify the object in an image similar to Semantic Segmentation but with more detailed information. This technique can show that there are more instances of a detected object.[13]

A commonly used loss function for deep-learning segmentation algorithms is the intersection over union (IoU).

## 2.2 Annotation formats

### 2.2.1 COCO format

The COCO (Common Objects in Context) format [6] is a widely used file format for object detection tasks in computer vision. It is a standardized format that is used to represent datasets for various computer vision tasks, including object detection, segmentation, and keypoint detection.

The COCO format consists of two main parts, as shown in Figure 2.2. The JSON file contains metadata about the dataset, and an image directory contains the images. The JSON file includes information about the images in the dataset, the objects within each image, and the annotations for each object.

Overview of each field in JSON file of COCO format:

1. **info**: This field contains general information about the dataset, such as the dataset name, version, and description.
2. **licenses**: This field contains information about the licenses under which the dataset is released. This can include the license name, URL, and any specific terms and conditions.
3. **categories**: This field defines the categories of objects that are present in the dataset. Each category is defined by a unique identifier, a name, and a supercategory (if applicable).
4. **images**: This field contains information about the images in the dataset, such as their filename, unique identifier, width, and height.
5. **annotations**: This field contains information about the objects in the images, including their category label, bounding box coordinates, and a unique identifier. Each annotation is associated with a specific image, and the image ID links the annotations to their respective images.

```
<dataset_dir>/
  data/
    <filename0>.<ext>
    <filename1>.<ext>
    ...
  labels.json
```

**Figure 2.2:** COCO folder structure.

### 2.2.2 YOLOv5 format

The YOLOv5 format is a custom format used by the YOLOv5 algorithm to store the annotations for object detection tasks. This format creates a .txt file for each image with the same name. Each text file is a simple text file where every line represents an annotation for an object in the image with the same name as the text file. The structure of each line in the text file is shown in the following Figure 2.3 YOLOv5 format uses a YAML file for

```
<object-class><x><y><width><height>
```

**Figure 2.3:** YOLOv5 annotation notation in .txt file.

data configuration to define parameters and settings for training. The YAML file contains the following parameters:

1. Class names: This parameter lists the names of the classes that the model needs to detect.
2. Number of classes: This parameter specifies the number of classes the model must detect.
3. Train and validation dataset paths: These parameters specify the paths to the training and validation datasets in the system.

### 2.2.3 Severstal format

The format of annotations used on Kaggle is a CSV file containing the file name in the 'ImageId' column, RLE (Run Length Encoding) in the column 'EncodedPixels', and class id of the defects described in RLE in column 'ClassId'.

ImageId	ClassId	EncodedPixels
0002cc93b.jpg	1	29102 12 29346 24 29602 24 29858 24 30114 24 30370 24 3
0007a71bf.jpg	3	18661 28 18863 82 19091 110 19347 110 19603 110 19859 1
000a4bcdd.jpg	1	37607 3 37858 8 38108 14 38359 20 38610 25 38863 28 391
000f6bf48.jpg	4	131973 1 132228 4 132483 6 132738 8 132993 11 133248 13
0014fce06.jpg	3	229501 11 229741 33 229981 55 230221 77 230468 92 23062

**Figure 2.4:** CSV annotation file example.



## 2.3 Evaluation metrics for object detection

### 2.3.1 IoU

Intersection over Union (IoU) measures the overlap between the predicted and ground truth bounding boxes. It is calculated as the area of intersection between the two boxes divided by the area of union. A threshold value is usually set to determine whether a prediction is a true positive or a false positive. If the IoU is greater than the threshold value, then the prediction is considered a true positive otherwise, it is considered a false positive. [11]

- True Positive (TP): A predicted bounding box is a true positive if it has an IoU score greater than a certain threshold with a ground truth bounding box. In other words, it correctly identifies an object as belonging to a particular class.
- False Positive (FP): A predicted bounding box is a false positive if it has an IoU score less than the threshold with all ground truth bounding boxes of that class. In other words, it incorrectly identifies a background region as belonging to a particular class.
- False Negative (FN): A ground truth bounding box is a false negative if there is no predicted bounding box with an IoU score greater than the threshold. In other words, the model fails to detect an object that belongs to a particular class.
- True Negative (TN): This term represents a correct misdetection. Many possible bounding boxes should not be detected, so the metrics do not commonly use TN. However, in some cases, it may be relevant to consider the true negatives as the correctly identified background regions.

### 2.3.2 Precision (P) and Recall (R)

Precision (P) and Recall (R) are two metrics used to measure the performance of object detection models. Precision is the fraction of true positives out of all the predictions, while recall is the fraction of true positives out of all the ground truth annotations.

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

**Figure 2.5:** The formula for calculating precision.

Precision and Recall are important evaluation metrics because they can help determine how well an object detection model correctly identifies objects in the image. A higher precision value indicates that the model produces fewer false positives, while a higher recall value indicates that the model detects more ground true annotations.

$$Recall = \frac{TP}{TP + FN} = \frac{TP}{\text{all ground truths}}$$

**Figure 2.6:** The formula for calculating recall.

### 2.3.3 Average Precision (AP)

Average Precision (AP) is a metric for evaluating object detection models. It is calculated by computing the precision and recall values for different IoU thresholds and then computing the area under the precision-recall curve. The AP metric is used to measure how well the object detection model performs across all the IoU thresholds. A higher AP value indicates better performance of the object detection model. [7]

### 2.3.4 Mean Average Precision (mAP)

Mean Average Precision (mAP) is a commonly used metric for evaluating object detection models. It is the average of the AP values across multiple object classes. The mAP metric is used to measure the overall performance of the object detection model. [25]

mAP.50 is a variation of the mean average precision (mAP) metric where the AP values are averaged only over the IoU threshold of 0.5. Same for mAP.75 or other specific IoU thresholds.  $mAP_s$ ,  $mAP_m$ , and  $mAP_l$  measure the average precision of the object detection model on objects of different scales by computing the AP values separately for each size category. Specifically,  $mAP_s$  computes the AP for small objects with bounding box areas below 32x32 pixels,  $mAP_m$  computes the AP for medium objects with bounding box areas between 32x32 and 96x96 pixels, and  $mAP_l$  computes the AP for large objects which have bounding box areas above 96x96 pixels.

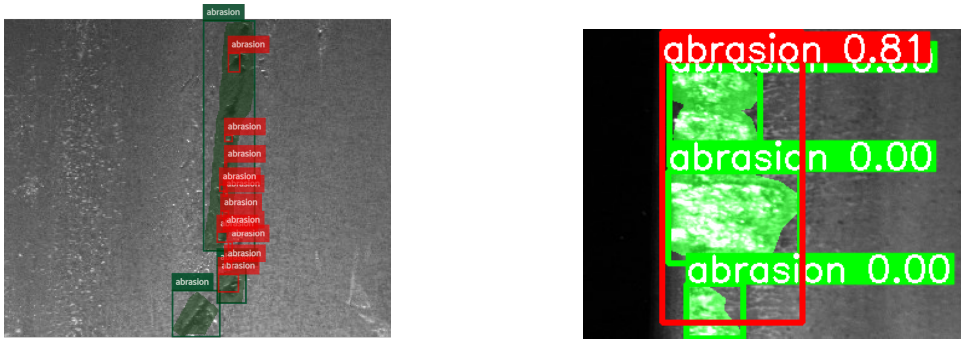
### 2.3.5 F1 score

F1 score is a metric used to evaluate the overall performance of the object detection model. It is the harmonic mean of precision and recall and is used to balance both precision and recall. It is useful when the classes are imbalanced, and each class needs to balance precision and recall. A higher F1 score indicates better performance of the object detection model.

### 2.3.6 Benevolent metric

Here we are defining a more benevolent way of deciding if a prediction is true positive or false positive. It uses instead of IoU only intersection of bounding boxes. This metric has the following criteria:

- When a prediction is smaller than the ground truth annotation, the intersection of these two bounding boxes must be greater than 75 % of the prediction area to mark the prediction as true positive.
- For predictions larger than the ground truth annotation, the intersection of these bounding boxes must be greater than 75 % of the ground truth defect area to mark this prediction as true positive. Otherwise, it is a false positive.



**Figure 2.7:** Examples of problematic predictions for IoU metric.

In Figure 2.7, red rectangles illustrate the predictions, and the green one ground truth annotation. The left side of this figure shows the case when the prediction is smaller than the real defect annotation, and the right side displays the case when one large prediction contains more ground truth annotations.

The example on the left shows much smaller predictions, but according to IoU, it is not detected. To say this is detected, the IoU threshold must be extremely low. Similarly, on the right side, one large prediction contains all three ground truth annotations. And none of them is detected, according to IoU. However, in practice, these detections would be sufficient. Thus, we propose a custom metric to account for this fact.

This metric is in this work tagged with the suffix '-B' as TP-B for true positives and FP-B for false positives calculated this way.

### 2.3.7 Number of detected defects

The number of detected defects is a metric used to evaluate the success rate of predictions. If there is at least one true positive prediction for a defect annotation, it is marked as detected.

This metric is in this work tagged as DC (Detected Count). DC calculated with IoU threshold is tagged DC and, for the benevolent way described in the previous subsection, is tagged DC-B.



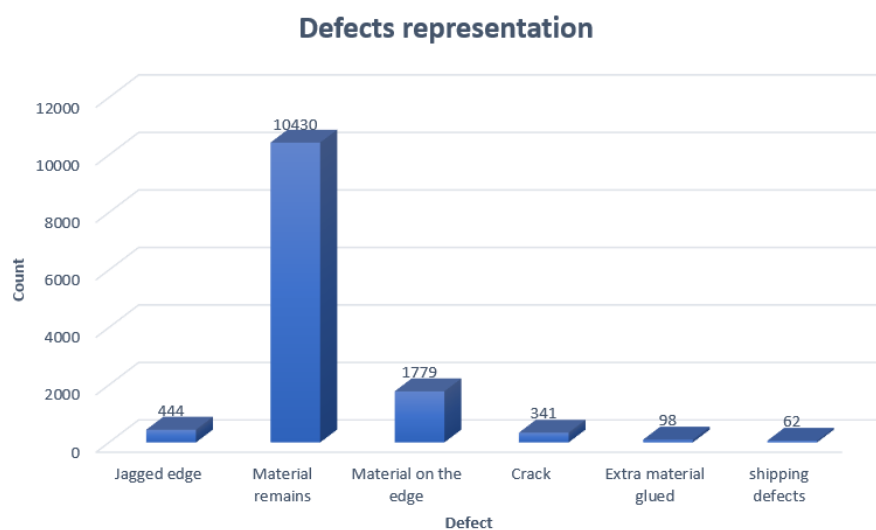
## Chapter 3

### Defect types in manufacturing

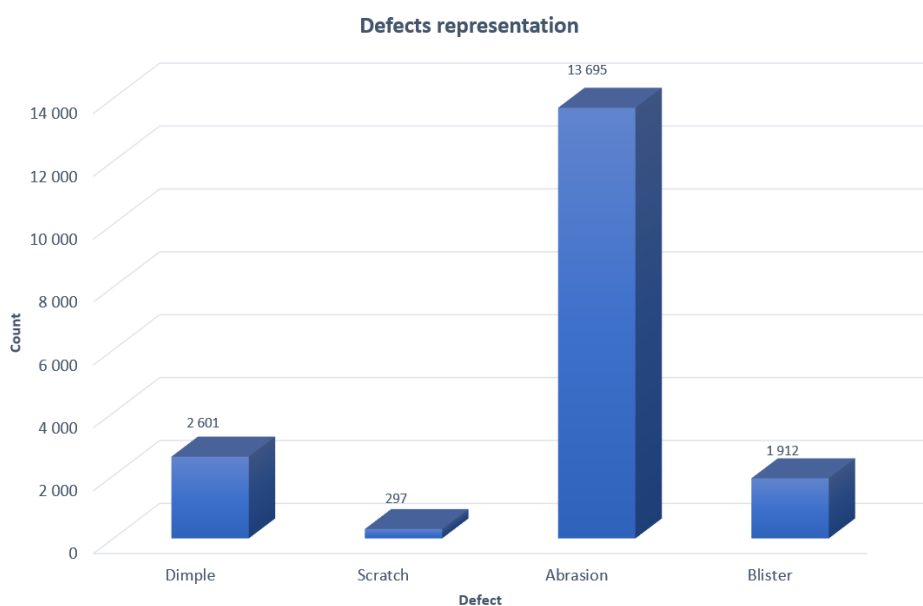
This section will describe all defects in the datasets that will be used in this Bachelor's thesis.

The first one is a private dataset given by DNAI s.r.o., which contains 113 high-resolution examples with 13 154 annotations from the manufacturing industry. Products in these 113 images can be divided into 28 different types. All 113 images contain at least one annotation of defect from any category.

The second dataset is a public Severstal: Steel Defect Detection dataset from the Kaggle competition. Used are only images from training set, because for these images are available ground truth annotations. This subset consists of 12 568 images with 18 505 annotations. There are 5 902 images without any annotation and 6 666 images with at least one annotation of defect from any category. All of them have resolution  $1\,600 \times 256$  pixels, which is much lower than the private dataset.



**Figure 3.1:** Graph shows the count of each defect class in the private dataset given by DNAI.

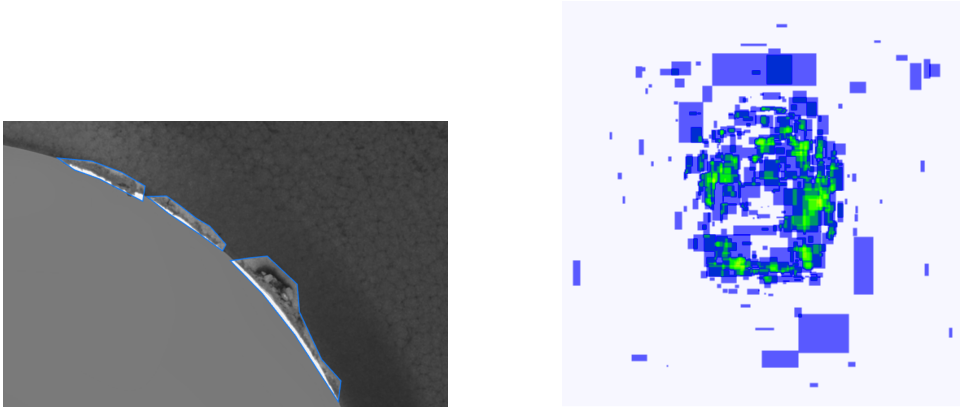


**Figure 3.2:** Representation of each defect type in the public Severstal steel dataset.

## 3.1 DNAI dataset

### 3.1.1 Jagged edge

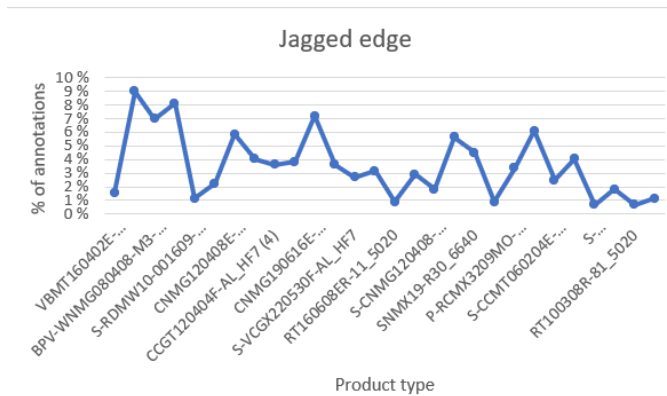
The first defect type from the private dataset is the jagged edge. The edges of the product should be sharp. Breaking off the edge causes a jagged edge with a typically slightly darker color. The shape of the defect at the edge is in the shape of the product, but on the other side, it is very diverse. The defect is mostly medium-sized.



**Figure 3.3:** On the left side is a jagged edge example. Annotation Heatmap, generated by Roboflow [8], which shows the typical placement of the jagged edge, is on the right side.

The given dataset has jagged edge 444 annotations in 101 images. The jagged edge in most cases has 2 to 5 instances by image. The average jagged edge area is 5 451 pixels.

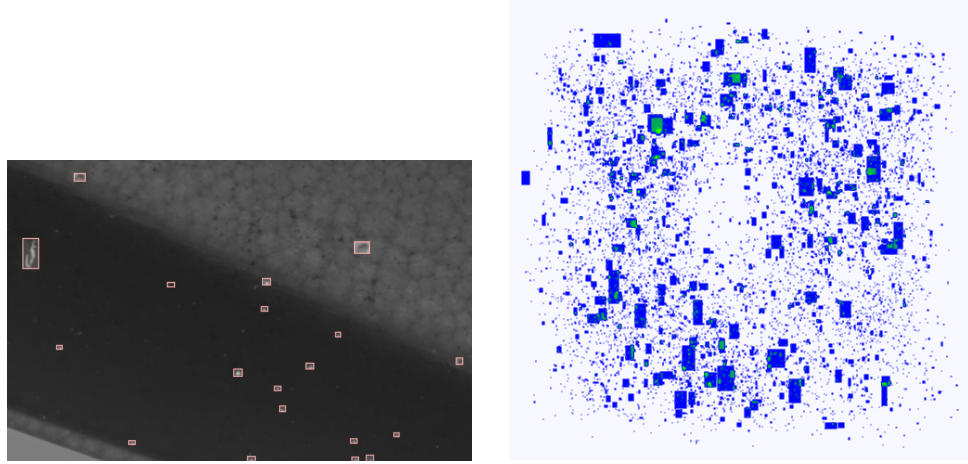
The distribution of annotations of jagged edge defects shows in the graph below. Types with the most annotations are DCMT11T and TNMG1604. On the other hand, the lowest are S-APMT11 and RT100308.



**Figure 3.4:** Jagged edge annotations distribution.

### 3.1.2 Material Remains

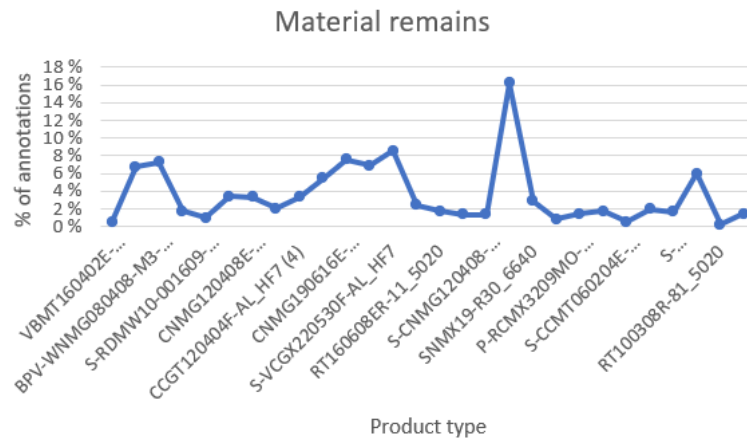
The second defect type from the private dataset is material remains, which are minimal areas on the product surface covered with the remaining material. These areas are lighter or darker than the remaining surface of the product.



**Figure 3.5:** On the left side is a material remains example. On the right side is the annotation Heat-map, which shows that the material remains defects can be located on the surface of any part of the product. The heat-map was generated by Roboflow [8].

The given dataset has material remaining 10 430 annotations in all 113 images. The remaining material defects generally have 2 to 154 instances by image. Several samples have even 461 to 511 instances by image. The average area of this defect is 354 pixels.

The distribution of annotations of this defect shows in the graph below. Types with the most annotations are CNMM2509 and S-VCGX2205. RT100308 and S-CCMT0602 have the least annotations.

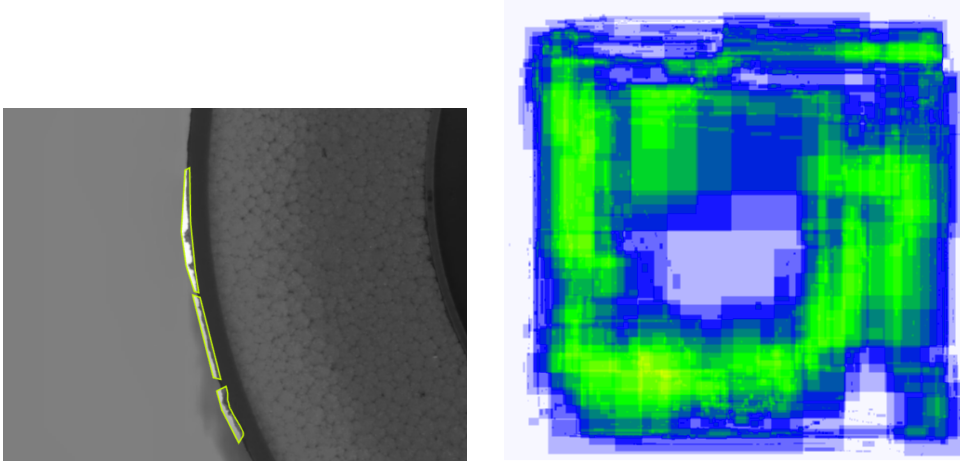


**Figure 3.6:** Material remains annotations distribution.



### 3.1.3 Material on the edge

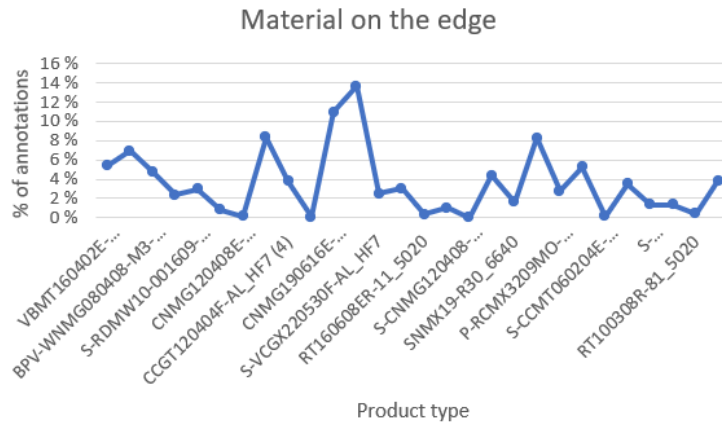
Material on the edge is another type of defect from the private dataset. It is the same defect as in the previous section but is located only on product edges. The shape is typically narrow and long. The edge where the material is located could be more perfectly straight but slightly wavy.



**Figure 3.7:** On the left side is material on the edge example. Annotation heat-map, generated by Roboflow [8] on the right side shows the typical placement of the material on the edge.

The given dataset has material on edge 1 779 annotations in 100 images. The material on the edge count is mostly 2 to 23 by image but rarely is it even 101 to 111 defect count by image. The average area for this defect type is 4 117 pixels.

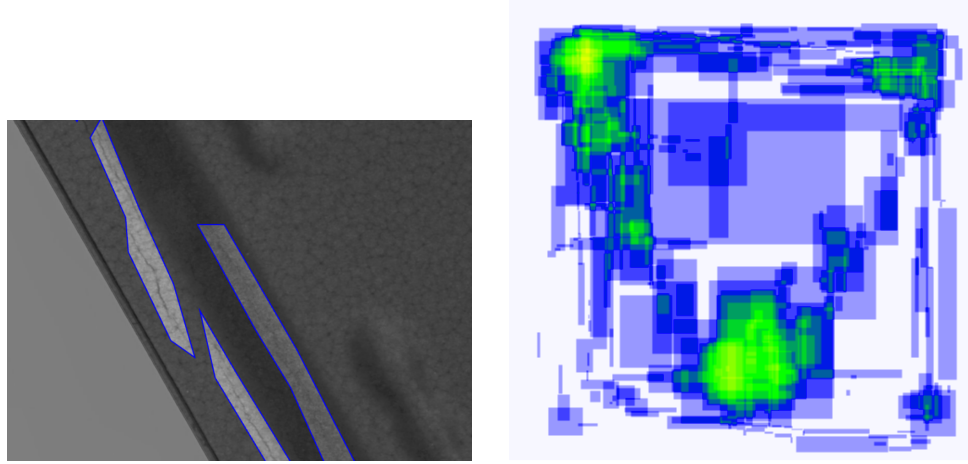
The distribution of annotations of this defect shows in the graph below. Types that have the most annotations with material on edge are SNMG2509 and CNMG1906. S-CMG1204 and SNMG1204 have the least annotations.



**Figure 3.8:** Material on the edge annotations distribution.

### 3.1.4 Crack

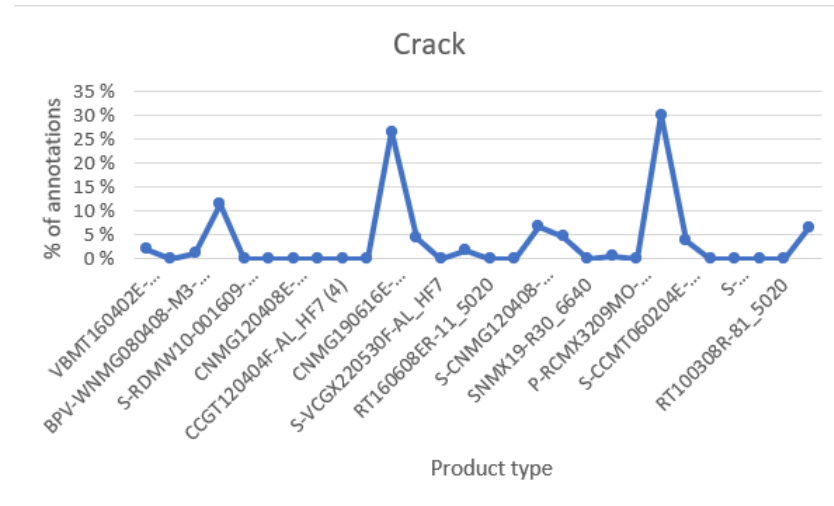
The private dataset contains a defect type called a crack, a small space in the material that can be seen in images as a line darker than its surroundings. And the crack has minimal surface area but may occur over a large portion of the product surface.



**Figure 3.9:** A crack example is on the left side. Annotation heat-map, generated by Roboflow [8] on the right side, shows the typical location of this defect.

The given dataset has 341 crack annotations in 31 images. The crack mostly has 2 to 11 instances per image, but some samples with 42 to 46 instances exist. The average area of the crack defect is 19 603 pixels.

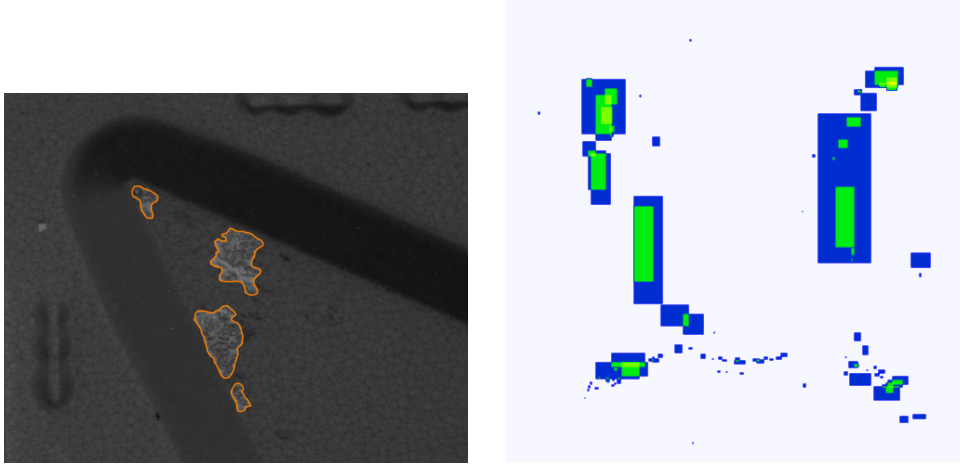
The distribution of this defect shows in the graph below. Types with the most annotations are S-TNMM3309 and CNMG1906 are types with the least annotations of the crack defect.



**Figure 3.10:** Crack annotations distribution.

### 3.1.5 Extra material glued

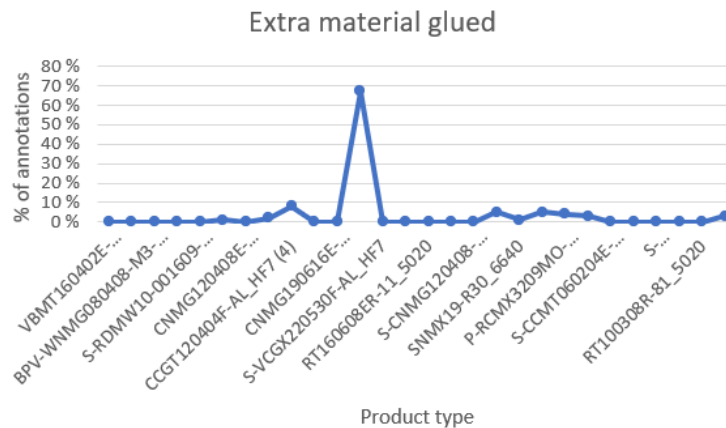
Extra material glued is another type of defect from the private dataset occurring on the surface of the product. It has various sizes and shapes and can be located on a surface in any part of the product. The structure of the surface differs from the surroundings and has more wrinkles and darker colors.



**Figure 3.11:** On the left side is extra material glued example. The typical placement of the extra material glued is shown in the annotation heat-map generated by Roboflow [8] on the right side.

The given dataset has extra material glued 98 annotations in 26 images. This defect count by image is mostly 1 to 4 but in rare cases, it is up to 25. The average area for this defect type is 13 107 pixels.

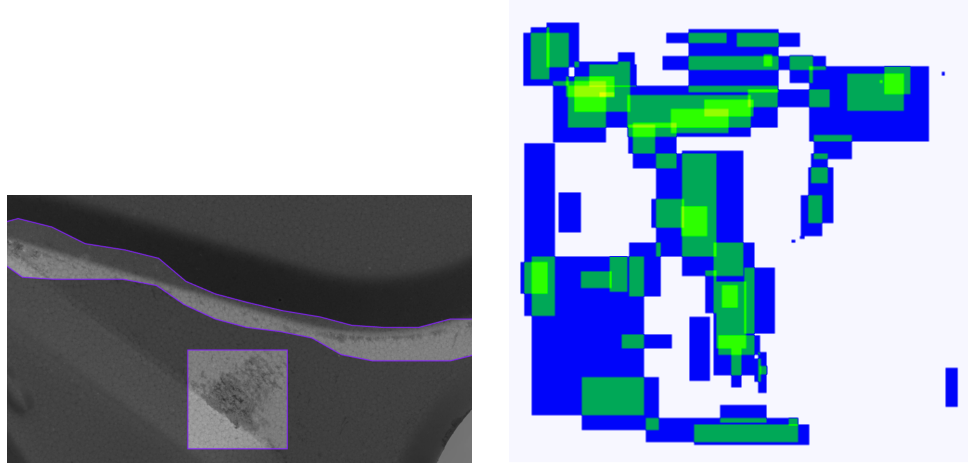
The distribution of annotations of extra material glued is shown in the graph below. The type which has the most annotations is SNMG2509. Most of the product types have 0 % of annotations of this defect.



**Figure 3.12:** Extra material glued annotations distribution.

### 3.1.6 Shipping defects

Defects from shipping are the last defect type from the private dataset. They can be abrasions or scratches anywhere on the product. These defects are typically irregularly shaped and large but rarely can be smaller too.



**Figure 3.13:** A shipping defect example is on the left side. On the right side is an annotation heat-map generated by Roboflow [8], which shows the placement of shipping defects.

The given dataset has shipping defect 62 annotations in 23 images. Shipping defect count by image is mainly 1 to 3. The average area of the shipping defect is 137 539 pixels.

The distribution of annotations of this defect shows in the graph below. Most annotations of shipping defects have S-TNMM3309. Many types have 0 % of annotations.

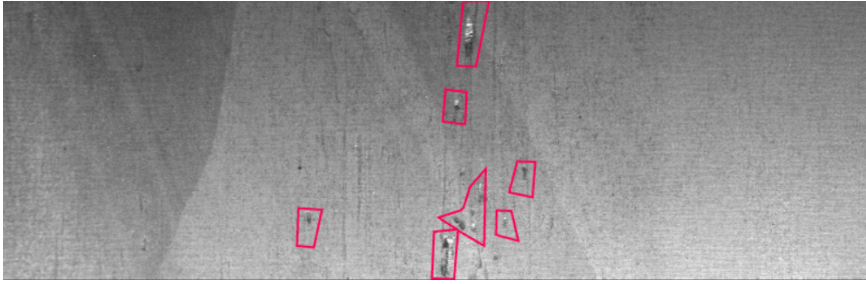


**Figure 3.14:** Shipping defect annotations distribution.

## 3.2 Severstal dataset

### 3.2.1 Dimple

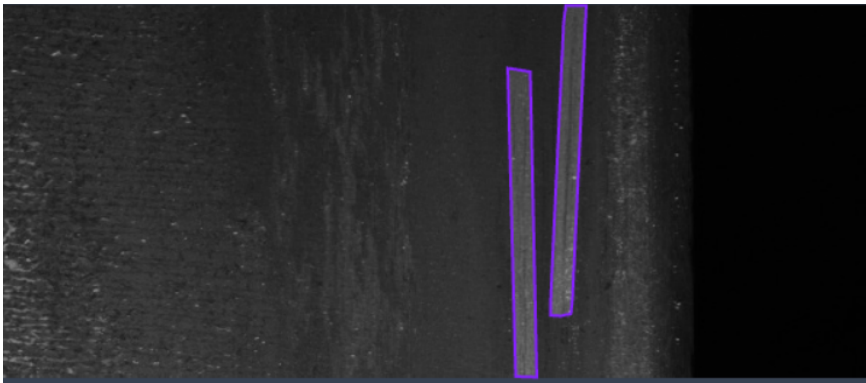
Small patches of a missing or incomplete surface are the first defect type from the Severstal steel dataset. They typically appear as small, irregularly-shaped regions with a color that differs from the surrounding area, ranging from a few pixels to a few tens of pixels in diameter. They can be scattered randomly across the image and can appear anywhere on the surface of the steel. The count of annotations per image is mostly 1 to 4.



**Figure 3.15:** Dimple example.

### 3.2.2 Scratch

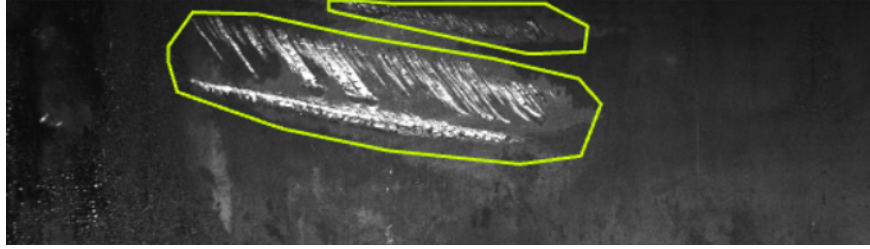
The second defect type from the Severstal steel dataset is elongated and narrow, typically appearing as long and dark thin lines on the surface of the steel. These defects are typically a few tens of pixels long and a few pixels wide and can have various shapes and orientations. It can appear in clusters or isolated instances. There is mainly one annotation per image of the scratch defect type.



**Figure 3.16:** Scratch example.

### 3.2.3 Abrasion

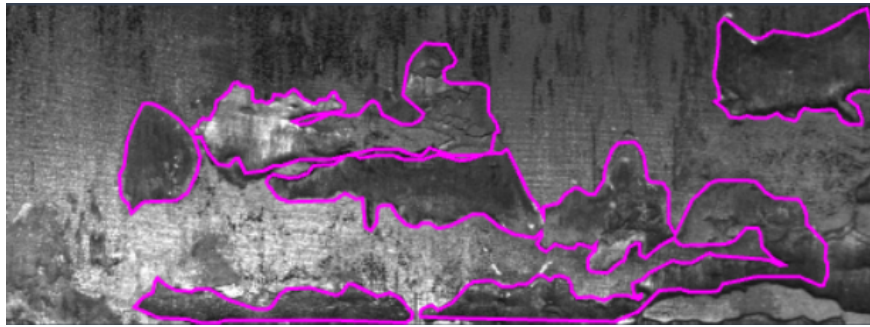
These defects can be irregular, with no specific pattern. This defect type from the Severstal steel dataset has defects that are typically larger than dimples and can appear as irregularly shaped patches on the surface of the steel. They can range in size from a few tens of pixels to a few hundred pixels in diameter. The number of abrasion annotations is 1 to 3 in most cases.



**Figure 3.17:** Abrasion example.

### 3.2.4 Blister

The fourth defect type from the Severstal steel dataset can be irregular and complex, with no specific pattern. These defects are typically much larger than abrasions and can appear as irregularly shaped patches or areas of missing surface on the steel. They can range in size from a few hundred pixels to a few thousand pixels in diameter. The majority of images contain 1 to 3 annotations of the blister.



**Figure 3.18:** Blister example.

## Chapter 4

### Object detection methods

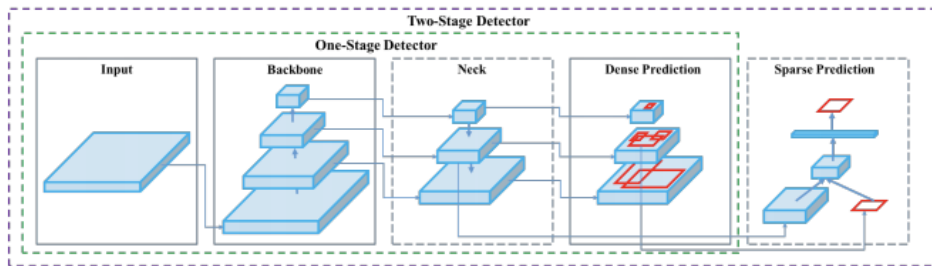
This chapter provides a basic description of one object detection method in each section. The Faster R-CNN, YOLO, SSD, or U-net are one of the most popular methods for object detection, so these are included in this survey.

#### 4.1 YOLO

The YOLO (You Only Look Once) is a family of Computer Vision models which provides real-time object detection. Following subsections describes individual versions of YOLO.

Its network consists of three main components [26].

- **Backbone:** A convolutional neural network that aggregates and forms image features at various granularities.
- **Neck:** A set of layers used to merge and combine image features to pass to prediction.
- **Head:** Consumes features from the neck and takes a box and class prediction steps.



**Figure 4.1:** YOLO object detector, taken from [5].

The YOLO was realized similarly to the human object recognition system. The front part of the YOLO network structure is a modified structure of GoogleNet. The deeper the Convolution Neural Network, the better the performance with more layers. However, the number of parameters to be learned







### ■ 4.1.2 YOLOv2

YOLOv2 (sometimes called YOLO9000) was introduced in 2016. This variant of the YOLO algorithm can predict up to 9 000 different object categories while still running in real-time from its derived name. An effort was made to address the weaknesses of the previous version of YOLOv1, which had problems detecting small objects and had a low recall.[17]

YOLOv2 improvements are better and faster functionality. The main differences in YOLOv2 are as follows, taken from [28]

- Batch Normalization: Normalize the input layer with a slight change and zoom activation. Batch normalization reduces the offset of unit values in the hidden layer. Adding batch normalization to the convolutional layer increases the architectural mAP by 2 %.
- • Higher Resolution Classifier: The input size in YOLOv2 has been increased from  $224 * 224$  to  $448 * 448$ . An increase in the image input size increases the mAP by 4 %.
- Anchor Boxes: One of the most obvious changes in YOLO v2 is the introduction of the anchor box. YOLOv2 is classified and predicted in a single framework. These anchor boxes are responsible for predicting the bounding box and designing this anchoring box for a particular data set by using clustering (k-means clustering).
- Fine-Grained Features: One of the main problems that YOLOv1 must solve is to detect smaller objects on the image. This problem has been resolved in YOLOv2, which divides the image into smaller  $13 * 13$  grid cells. Enabling YOLOv2 to recognize or locate smaller objects in the image is also effective for larger objects.
- Multi-Scale Training: YOLOv2 uses a random image of different sizes ranging from  $320 * 320$  to  $608 * 608$  for training.
- Darknet 19: YOLOv2 uses the Darknet 19 architecture with 19 convolutional layers and 5 maximum pooling layers, and a SoftMax layer for classifying objects.

### ■ 4.1.3 YOLOv3

YOLOv3 was released in 2018. This version was based on the Darknet-53 architecture, which consisted of 53 convolutional layers, as opposed to the Darknet-19 architecture used in YOLOv2.

One of the most significant changes in YOLOv3 was the introduction of predictions at three different scales. This has enabled YOLOv3 to detect and classify small objects, resulting in a higher average accuracy (AP) score compared to previous versions. [17]

YOLOv3 improvements compare to the previous version, taken from [28]

- **Bounding Box Predictions:** The YOLOv3 method gives the score for the objects for each bounding box. It uses logistic regression to predict the objectiveness score.
- **Class Predictions:** YOLOv3 uses a logical classifier for each class instead of the SoftMax used in the previous YOLOv2. By doing this in YOLOv3, we can do multi-label classification.
- **Feature Pyramid Networks (FPN):** The predictions made by YOLOv3 are similar to FPN, in which three predictions are extracted for each location of the input image and features, and features are extracted from each prediction. In this way, YOLOv3 has better features on different scales. Each prediction consists of a bounding box, objectivity, and 80 category scores.
- **Darknet-53:** Previous YOLOv2 used Darknet-19 as a feature extractor, while YOLOv3 uses the Darknet-53 network as a feature extractor with 53 convolution layers. It is deeper than YOLOv2 and has a quick connection. The Darknet-53 consists primarily of 3x3 and 1x1 filters with quick connections.

### ■ 4.1.4 YOLOv4

The YOLOv4 model is an optimized model based on YOLOv3. In comparison to the network structure of YOLOv3, YOLOv4 introduces a modification to the Darknet53 network, which is replaced with a variant called CSP Darknet53. YOLOv4 significantly improves Average Precision (AP) and Frames Per Second (FPS). In comparison to YOLOv3, YOLOv4 demonstrates an important advancement, with a 10 % increase in AP and a 12 % boost in FPS. [23]

Improvements to make the detector more suitable for training on a single GPU, taken from [5]

- **New method of data augmentation Mosaic** which mixes 4 training images. Thus 4 different contexts are mixed, while CutMix combines only 2 input images. This allows the detection of objects outside their normal context.
- **The second new augmentation method Self-Adversarial Training (SAT)** operates in 2 forward-backward stages.

### ■ 4.1.5 YOLOv5

YOLOv5 [14] is a model in the You Only Look Once (YOLO) family of Computer Vision models. YOLOv5 is commonly used for detecting objects. YOLOv5 comes in four main versions: small (s), medium (m), large (l), and extra-large (x), each offering progressively higher accuracy rates. Each variant also takes a different amount of time to train.

YOLOv5 keeps P5 architecture for training images with a size of 640 pixels from YOLOv4 and adds P6 models for images with 1280 pixels. P6 models include an extra P6/64 output layer for the detection of larger objects and benefit the most from training at higher resolution.

The release of v7.0 YOLOv5 adds segmentation models, for instance segmentation.

### ■ 4.1.6 YOLOv8

The latest addition to the YOLO (You Only Look Once) family of object detection models is the YOLOv8 [27], released in January 2023, representing the current state-of-the-art deep learning-based object detection. YOLOv8 can be utilized for a wide range of computer vision tasks, such as object detection, image classification, and instance segmentation. Ultralytics, the developers of YOLOv8, have a proven track record in creating influential and industry-defining models, with YOLOv5 being one such example.

YOLOv8 may be used directly in the Command Line Interface (CLI) with a 'yolo' command or may also be used directly in a Python environment and accepts the same arguments as in the CLI, which are Task (detect, segment, classify), Mode (train, val, predict, export, track) and ARGS are any number of custom arg=value pairs like imgsz=320.

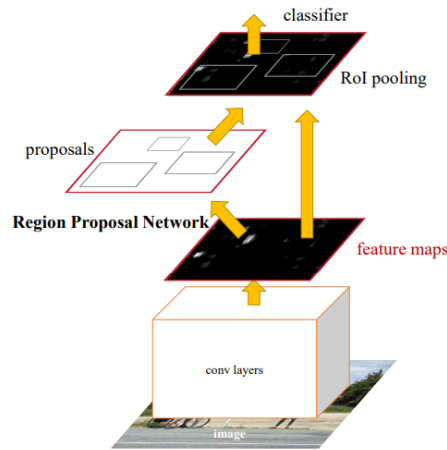
YOLOv8 pretrained models are available for detect, segment, and pose tasks, which are pretrained on the COCO dataset, while models for classification are pretrained on the ImageNet dataset.

At the time of this writing, there was no paper about YOLOv8.

## 4.2 Faster R-CNN

The Faster Region-based Convolutional Neural Network (Faster R-CNN) has been developed as a faster alternative to the Selective Search (SS) algorithm in the original R-CNN (Region-based Convolutional Neural Network) model. To generate regions of interest (ROIs), the Faster R-CNN model uses a compact convolutional network called the Region Proposal Network (RPN). Faster R-CNN introduces the concept of anchor boxes to handle variations in aspect ratio, which are used to scale the ROIs. When it comes to performance, Faster R-CNN is 250 times faster than R-CNN.

The first module of Faster R-CNN is a deep fully convolutional network that proposes regions, and the second module is a Fast R-CNN detector that uses the proposed regions. The entire system is a single unified object detection network. [22]



**Figure 4.4:** Faster R-CNN unified network, taken from [22].

The RPN module tells the Fast R-CNN where to search for objects. The network produces a convolutional feature map after processing the whole image with several convolutional and max pooling layers. From a crop of the image, called Region of Interest (RoI), one obtains a vector containing fixed-length features determined from the map of features of each object. This vector is sent to a sequence of fully connected layers that are fed finally to two sibling output layers. One layer produces the maximum probability of estimation over a number of object classes, whereas the other layer outputs 4 real numbers for each class. [10]

## 4.3 SSD

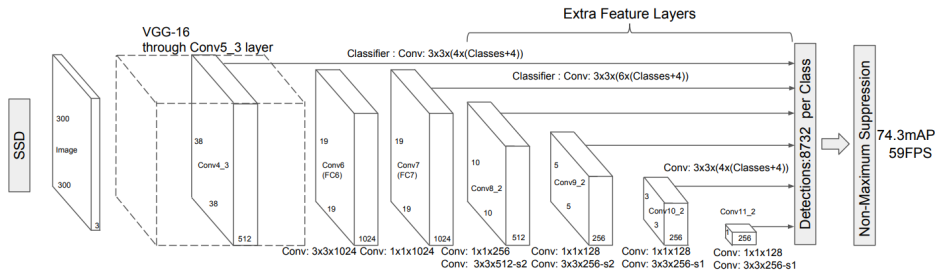
The Single Shot Detector (SSD) is a detection framework capable of identifying multiple object categories in a single pass. The core of SSD is predicting category scores and box offsets for a fixed set of default bounding boxes using small convolutional filters applied to feature maps.

Convolutional feature layers were added to the end of the truncated base to detect objects at various scales. These layers decrease in size and enable object detection predictions at multiple scales.

SSD only needs an input image and ground truth boxes for each object during training.

The SSD object detection composes of 2 parts [12]:

- Extract feature maps
- Apply convolution filters to detect objects.



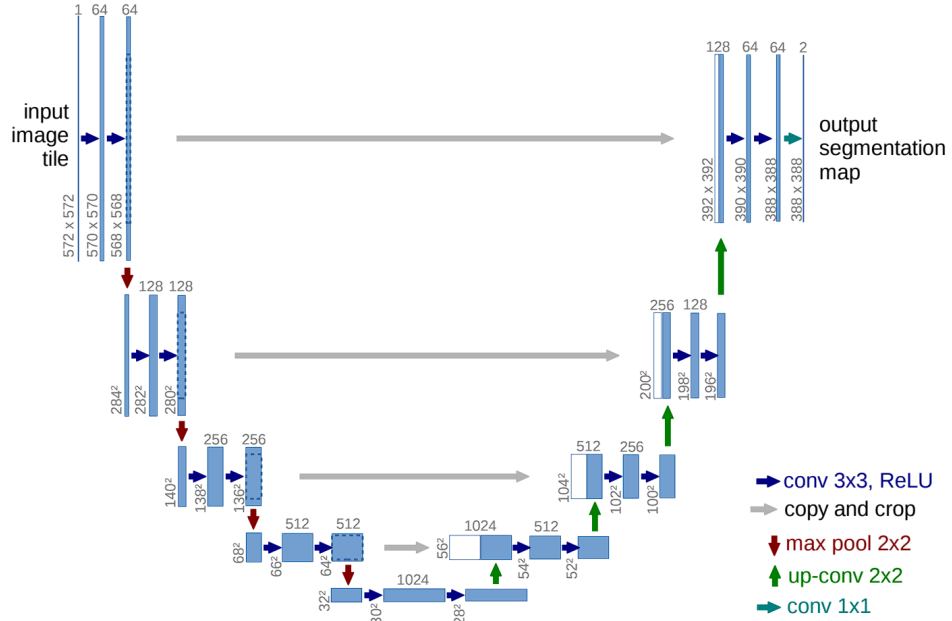
**Figure 4.5:** SSD architecture, taken from [19].

The additional structure that was added to the network, taken from [19]

- Multi-scale feature maps for detection: Convolutional feature layers were added to the end of the truncated base network. These layers decrease in size progressively and allow predictions of detections at multiple scales. The convolutional model for predicting detections is different for each feature layer.
- Convolutional predictors for detection: Each added feature layer can produce a fixed set of detection predictions using a set of convolutional filters.
- Default boxes and aspects ratios: Default bounding boxes are associated with each feature map cell for multiple feature maps at the top of the network. The default boxes tile the feature map in a convolutional manner so that the position of each box relative to its corresponding cell is fixed.

## 4.4 U-net

U-Net is a convolutional neural network architecture designed for image segmentation tasks. It is called "U-Net" because of its U-shaped architecture, consisting of a contracting path followed by an expanding path.



**Figure 4.6:** U-net architecture, taken from [24].

The first part of the U-Net architecture is the contracting path, composed of convolutional and pooling layers that progressively reduce the resolution of the input image while increasing the number of feature channels. Each convolutional layer is typically followed by a rectified linear unit (ReLU) activation function.

The last part of the contracting path is a bottleneck layer consisting of a convolutional layer that combines spatial information from the input image with the learned feature representation. Many filters are there to catch the most important features.

The remaining half of the U-Net architecture is an expanding path that reflects the contracting path. It is upsampling the feature maps back to the input image size while decreasing the number of feature channels. Transposed convolution layers are used in upsampling steps.

At the end of the expanding path, a convolutional layer with a sigmoid activation function is used to get the final segmentation map with the same resolution as the input image. The output of the U-Net model is a binary mask for each class. [29] [24].

## Chapter 5

### Necessary prerequisites

This chapter describes the necessary steps before doing any experiments. Some of these steps are the preparation of data format and choosing which methods we want to use. The methods utilized are selected in 5.1. It is important to know how to run training and inference for chosen methods, which is provided in Section 5.6. The private dataset was already received in COCO format, so all needed to use YOLO was to convert the dataset from the COCO format 2.2.1 to YOLOv5 format 2.2.2 and know how to train and run inference. The public dataset has available annotations in CSV format 2.2.3. To use this dataset for YOLO was necessary to convert it with steps in Sections 5.2, 5.3, and 5.4. While using U-net, the private dataset must be converted using steps provided in Section 5.5. The public dataset does not need to use any conversion to use it on U-net. But for both datasets, the output of U-net must be converted by steps from Section 5.3 to get result bounding boxes to compare.

#### 5.1 Selection of methods used

The important condition in my selection of methods was that a method is nowadays being used and to be able to compare their results.

The first used method is YOLOv5[15] because it is widely used in practice for object detection nowadays. The main advantages of the YOLOv5 are that it is easy to use and also easy to install because it only requires the installation of a torch and some Python libraries from requirements.txt. Another benefit is that the YOLOv5 models are training pretty quickly.

The second used method is Vanilla U-net, specifically the implementation from the Kaggle competition [4]. The Severstal dataset is from the Kaggle competition, which has already ended, but many solutions are there. So I decided to use one of the best solutions for its great success in this competition.

Chosen YOLO generates prediction bounding boxes, and U-net performs segmentation masks. Converting the result masks from the U-net to bounding boxes is necessary to make the results comparable.

## 5.2 Conversion of the CSV annotations to RGB mask

The annotations with train images were downloaded from the Kaggle competition [3] in the format shown in the following Figure. A mask was created for

	ImageId	ClassId	EncodedPixels
0	0002cc93b.jpg	1	29102 12 29346 24 29602 24 29858 24 301...
1	0007a71bf.jpg	3	18661 28 18863 82 19091 110 19347 110 1...
2	000a4bcd.jpg	1	37607 3 37858 8 38108 14 38359 20 38610...
3	000f6bf48.jpg	4	131973 1 132228 4 132483 6 132738 8 132...
4	0014fce06.jpg	3	229501 11 229741 33 229981 55 230221 77...

**Figure 5.1:** The annotations in CSV format from the Kaggle

each image in the folder, which has a black color ('(0, 0, 0)' in RGB), where is no annotation. Areas, where annotations were placed, were following colors according to defect class id. The '(255, 0, 0)' in RGB for the defect which has id 1, '(0, 255, 0)' for the defect class with id 2, '(0, 0, 255)' for the defect class with id 3 and '(128, 0, 255)' for defect class with id 4. These RGB masks were saved as .png files. After these steps, CSV annotations were successfully converted to RGB masks. The notebook [16] inspired this process.

## 5.3 Conversion of the annotations from the RGB mask to the COCO

The git repository [9] was used to convert the RGB masks to the COCO format. The 'create-custom-coco-dataset.ipynb' file is needed, but some modifications must be made first. Change the category ids in the format 'defect\_name': id' and category colors in the format '(R, G, B)': id' as shown in the Figure below.

```
category_ids = {
    "dimple": 0,
    "scratch": 1,
    "abrasion": 2,
    "blister": 3
}
```

```
category_colors = {
    "(255, 0, 0)": 0, # dimple
    "(0, 255, 0)": 1, # scratch
    "(0, 0, 255)": 2, # abrasion
    "(128, 0, 255)": 3 # blister
}
```

**Figure 5.2:** On the left side of the Figure is shown the settings of category ids, and on the right side are category colors corresponding to colors in the previous step. 5.2



## 5.4 Conversion of the annotations from the COCO to the YOLOv5 format

The conversion of the data in the COCO format to the YOLOv5 format is performed in this work using the SAHI framework. This framework has the following command:

```
sahi coco yolov5 --image_dir dir/to/images --dataset_json_path dataset.json
--train_split x [2]
```

, which will convert the given coco dataset to yolov5 format. The 'x' parameter for train\_split is between 0 and 1, with a default value of 0.9. Instead of using this command, it can be imported as a Python function 'from sahi.scripts.coco2yolov5 import main'.

## 5.5 Conversion of the annotations from the COCO to the CSV format

The conversion of the data in the COCO to the CSV format is using the pycocotools function called 'annToMask', which makes a mask from the annotation. But all annotations need to contain the field 'segmentation'. Some of the annotations from the private dataset do not contain segmentation, it is needed to add it in the shape of the bounding box. If all annotations have segmentation, it is appropriate to create a mask for each defect type per image. And finally, for each mask generate RLE (Running Length Encoding) and write it into a CSV file in column 'EncodedPixels'.

## 5.6 How to train and predict with YOLOv5 model

### 5.6.1 How to train

To run training of YOLOv5 is necessary to clone the git repository of YOLOv5 and install requirements in requirements.txt. Command to run training in basic is

```
python train.py --img img_size --batch batch_size --epochs N --data path/to/yaml --weights path/to/model.pt
```

where

- '--img': The size of the input image for the model (e.g., 640, 1280, etc.).
- '--batch\_size': The number of images to include in each training batch (e.g., -1 for automatic calculation or 8, 16, 32, 64, etc.).
- '--epochs': The number of epochs to train the model for (e.g., 30, 100, 1000, etc.).
- '--data': The path to the data config file (YAML file 2.2.2) containing information about the training and validation data

- '--weights': The path to the initial weights file to use for training the model (e.g., " for random initial weight or yolov5s.pt, yolov5m.pt, yolov5l.pt, etc.)
- '--cfg': The path to the model configuration file (YAML file) that defines the architecture of the YOLOv5 model. This parameter has to be set if are used randomly initialized weights.
- '--hyp': the path to the hyperparameters file (YAML file) containing values for various hyperparameters used during training (e.g., learning rate, momentum, etc.).

### ■ 5.6.2 How to run inference

Inference can run directly from YOLOv5, but to use techniques like slicing, SAHI (Slicing Aided Hyper Inference) [1] will be used. Inference with YOLOv5 detect.py:

```
python detect.py --img img_size --weights path/to/model.pt --source path/to/imgs --conf K
```

where

- '--img': This specifies the input image size for the model. The default value is 640, and larger values will increase the detection accuracy but also increase the processing time and memory requirements.
- '--weights': This specifies the path to the trained YOLOv5 weights file to be used for inference.
- '--source': This specifies the input source for the detection, which can be a path to an image or video file, a directory containing images, or a camera index.
- '--conf': This sets the detection confidence threshold for filtering out low-confidence detections. The default value is 0.25, and higher values will result in fewer detections but higher precision.

Inference with SAHI is detailed described here [2], but important is SAHI predict command:

```
sahi predict --slice_width slice_size --slice_height slice_size --overlap_height_ratio overlap_ratio --overlap_width_ratio overlap_ratio --model_confidence_threshold T --source image/file/or/folder --model_path path/to/model.pt where
```

- '--slice\_width or height': This sets the width or height of each slice to the given integer value. The default value is 512 pixels.
- '--overlap\_width\_ratio or height': This specifies the fractional overlap in the height of each window (e.g., an overlap of 0.2 for a window of size 512 yields an overlap of 102 pixels). The default value is 0.2.
- '--model\_confidence\_threshold': This is the same as the 'conf' parameter while using directly detect.py from YOLO.

- '--source': This is the same as the 'source' parameter while using directly detect.py from YOLO.
- '--model\_path': This is the same as the 'weights' parameter while using directly detect.py from YOLO.

## ■ 5.7 Viewing the dataset using FiftyOne

In this work, the FiftyOne open-sourced toolkit was employed to facilitate the examination of datasets. Specifically, a comprehensive analysis of all samples was conducted to identify and describe all types of defects types present within the data. The FiftyOne toolkit offers a multitude of additional features, including the ability to generate customized datasets from scratch or evaluate results and much more. [21]



## Chapter 6

### Severstal dataset

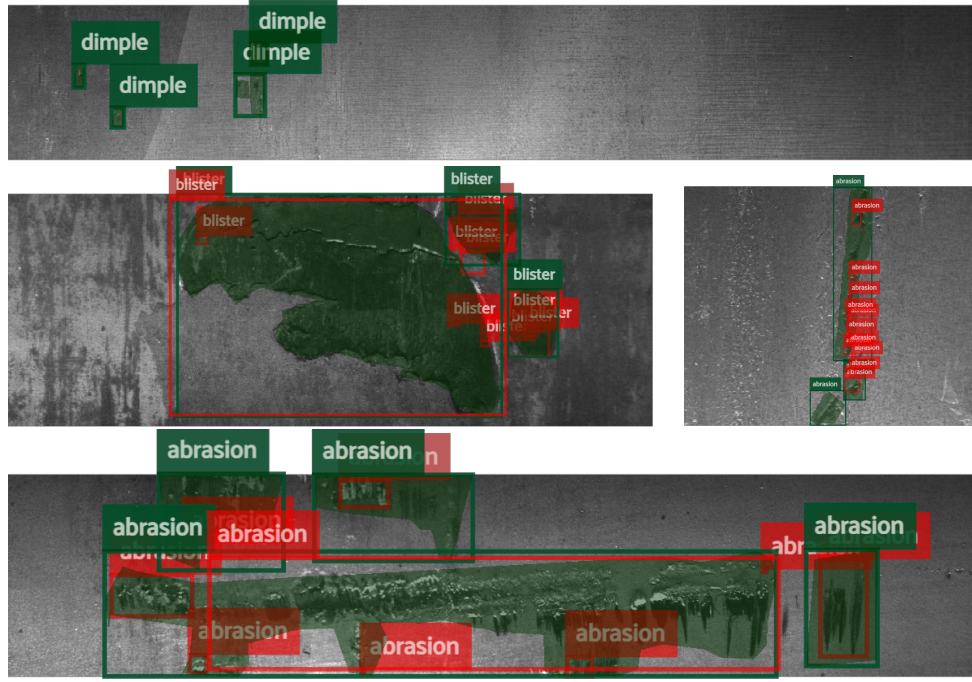
This chapter describes the results of measurements on the Severstal dataset. These results are compared between YOLO and U-net in Section 6.1 and then are experiments with the YOLO model to improve its performance in Section 6.2. Achieved results are provided in Section 6.2.13. All figures with example predictions in this chapter are consistent with green annotations indicating ground truth, and the red-colored annotations represent predictions. The testing set contains 1 006 images with 1 434 ground truth annotations, of which is 201 dimple, 20 scratch, 1069 abrasion, and 144 blister defects.

#### 6.1 Comparison of methods

Subsections 6.1.1 and 6.1.2 provide typical predictions reached by the relevant method. But the Subsection 6.1.3 contains a comparison of the main differences between YOLO and U-net predictions.

##### 6.1.1 U-net

Results reached while using the U-net method from the Kaggle notebook on the Severstal dataset are described here. The aim is to describe the most common problems or achievements reached with U-net. The following results were reached using U-net from the Kaggle competition trained with origin source images with the  $1\,600 \times 256$  pixels resolution for 30 epochs with 200 steps per epoch.



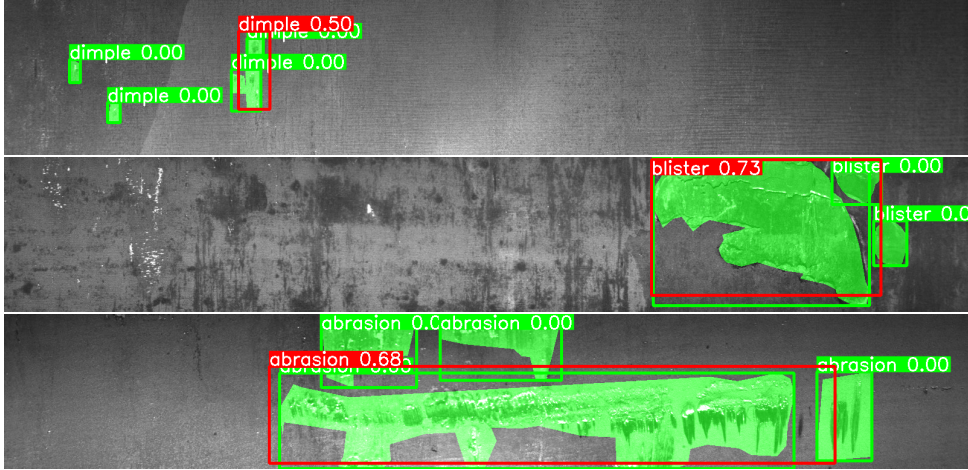
**Figure 6.1:** Example predictions performed by the U-net model.

The first prediction example in Figure 6.1 shows no dimple is detected. On the second row, the left example shows the blister defects, where all of them are detected. All predictions except one are true positives according to a softer metric consistent with visible predictions. The example on the right side shows the case when many small predictions are performed along the whole core of the defect. The example shows that two defects are correctly detected, and one is not. The last row illustrates the prediction of the large defect. The large defect is pretty well detected even when prediction separated this defect into one smaller in the left part on the defect and the rest of the defect on the right. The other one on the right side is well detected of the three remaining smaller defects. Only a small bottom part of the defect is detected on the left top part of the example, which is enough according to the more moderate metric. And the last one in the top middle of the example defect has detected the core of the defect.

From the observations above, it is evident that small defects are a problem to detect, as shown by no dimple prediction. Scratch defect type is also not detected at all, but scratch annotations are underrepresented in this dataset, which may also cause it. U-net has good results on large defects with pronounced edges or cores of medium-sized defects.

### 6.1.2 YOLO

The predictions of the YOLOv5 model trained on the Severstal dataset are provided in this subsection. Example predictions are here, showing typical results of the YOLO model. Training and prediction are performed on source images with an origin resolution of  $1600 \times 256$  pixels for 1000 epochs with the patience of 100, batch 32, and used randomly initialized weights with config file yolo5m.yaml.



**Figure 6.2:** Example predictions performed by YOLOv5 model.

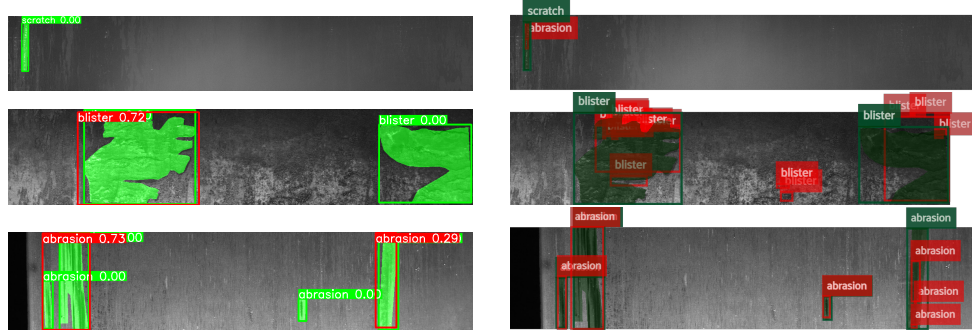
The first example in Figure 6.2 shows the prediction of dimple defects, where two defects are not detected, and the remaining two are detected in one, a slightly bigger prediction. The middle example displays a blister defect. One prediction detected two ground truth annotations pretty well, but the smaller defect on the right was not detected because the prediction bounding box does not contain a core of the defect and enough area of the defect. The last example shows a good prediction of large abrasion defect but all smaller defects are not detected.

In conclusion, from observation, small defects, such as dimple defect type, are difficult to detect. Also problem with the detection of the underrepresented scratch defect type, which was not detected at all. In the case of large or medium-sized defects, the predictions are quite accurate.

### 6.1.3 Summary comparison

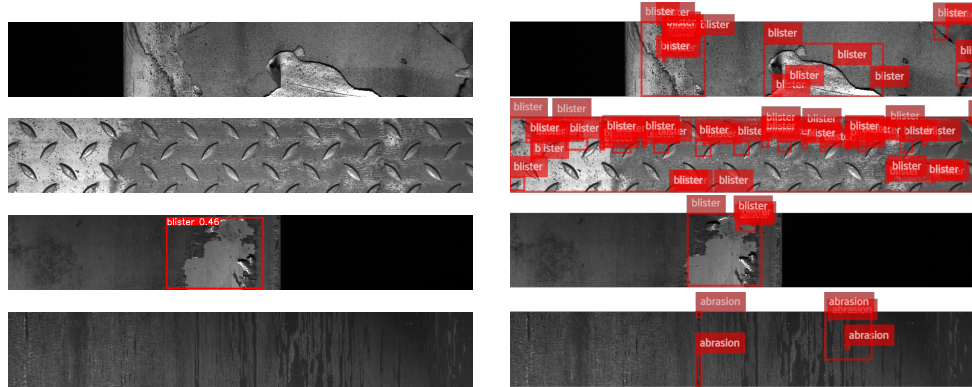
This subsection describes differences in results and predictions of both methods. The comparison is divided into three parts, where the first one compares reached predictions, which have quite good results. The second part of the comparison provides examples of false positives or unannotated defects. And the last third part of the comparison takes numeric results and checks that it is consistent with the observations made.





**Figure 6.3:** On the left side are predictions from YOLO, and on the right side from U-net.

Figure 6.3 shows a pretty good predictions. The first row compares the prediction of scratch defect where YOLO performed no prediction. In the case of U-net, the scratch is not predicted, but the abrasion is at the place where is a ground truth annotation of scratch. So U-net recognized the defect but assigned it to a bad defect type. There are more wrong defect type assignments, but they are rare. YOLO also has this problem but in smaller quantities. The second row shows that YOLO detected one blister defect with high accuracy, but the other one did not. In contrast, U-net detected both defects but with more prediction bounding boxes less accurately and with 2 false positives. The last row displays predictions of abrasion. YOLO performed 2 predictions which detected 4 defects. Many smaller predictions are performed by U-net detecting all defects, which may cause mark some of these predictions as not accurate enough while using the IoU metric.



**Figure 6.4:** On the left side are predictions from YOLO, and on the right side are from U-net. This Figure shows differences in false positive prediction.

Figure 6.4 shows differences in false positive predictions. The first example shows the example which is according to the ground truth without defects. As can be seen, the surface is peeled off, which looks like a blister defect. Because there are more images like this, false positives are high in the case of U-net. The example below shows the case when the regular pattern is on the whole surface. The whole image is covered with predictions performed by U-



net, which are false positives according to the ground truth. YOLO performed no prediction that can be considered as true negative. The third row is an example of a damaged surface looking like a blister defect that is not a defect according to ground truth annotations. Both methods detected this defect. We believe it is an unannotated defect, and more cases like this are in this dataset. The last row illustrates the problem when the differently colorized surface looks like a defect. U-net performed predictions in places with these different colors. So U-net got confused with this compared to YOLO, which deals with it.

Metric	DC	P	R	TP	FP
U-net	485	6.8 %	33.8 %	485	6 635
YOLO	505	50.9 %	35.2 %	507	490

Metric	DC-B	P-B	R-B	TP-B	FP-B
U-net	913	20.1 %	63.7 %	1 428	5 818
YOLO	930	81.7 %	64.9 %	815	182

Metric	mAP	mAP.50	mAP.75	$mAP50_s$	$mAP50_m$	$mAP50_l$
U-net	0.009	0.02	0.007	0.0	0.039	0.097
YOLO	0.098	0.21	0.076	0.03	0.215	0.365

**Table 6.1:** Tables with the comparison of the number of detected defects, TP, and FP for the YOLO and U-net model in the first two tables, where the second one is for the more benevolent metric. The mAP results are described in the last table. The metrics in columns are described in Section 2.3.

The tables with numeric results reflex the problems from the observations, like not detecting small defects. Both methods must deal with unannotated defects or parts of the surface similar to defects, but YOLO handles these cases much better for available ground truth annotations. Because it causes a higher number of false positive predictions from U-net. U-net makes more of the smaller predictions, which results in lower prediction accuracy.

Considering that unannotated defects are not annotated for some sensible reason. Probably because this defect type is not important to detect. Therefore we chose YOLO as a more appropriate algorithm. Despite the fact that, in my opinion, some unannotated defects are of the type we want to detect.

## 6.2 Fine-tuning

This section describes experiments on the public dataset. These experiments are performed on several datasets, which are variants of the public dataset. The goal is to discover which techniques help improve the precision of models for this dataset.

In experiments, the following techniques are used. Slicing source images into smaller slices to improve the detection of small defects. Prediction is made with or without overlap of the slices. Because slicing can cause splitting defect annotation into more slices, find out how much of the defect area at least needs to be kept on the slice to improve detection. Determine the effect of negative samples in the training set. And verify if using the pre-trained model is useful, and if so, find out which one is the most appropriate for this dataset.

This section describes the effects of these techniques on the public dataset and the differences between them.

### 6.2.1 Data preparation

Several datasets for experiments were created with different parameters such as slice size, minimal object area, or the number of negative samples in the training dataset. These datasets must be converted to yolov5 format with steps mentioned in Chapter 5. Datasets for measurements were generated with the following parameters:

- Slice size 800, 400, or 256 pixels
- Overlap of sliced images 0 %
- Minimal area of defect in sliced image 5 % or 20 %
- Percentage of negative samples in training set 0 % or keep all

### 6.2.2 Training and prediction parameters

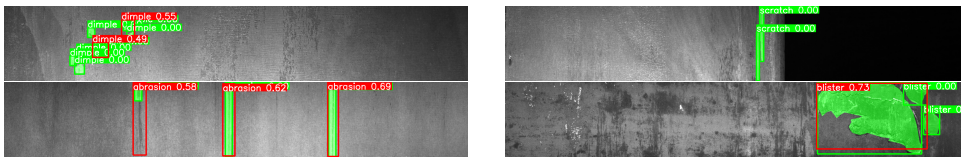
These results were reached with YOLOv5 v7.0.11[15], SAHI v0.11.13[1], Python 3.9.6 with torch-1.12.1+cu116. Parameters used while training:

- Batch: 32
- Weights: random initialized
- Cache: RAM
- Epochs: 1 000
- Patience: 100
- Config file: yolov5m.yaml

and parameter '--rect' is used to enable the training model on images with different width and height. Predictions were made with the same parameters as training and with 30% overlap, and the confidence threshold is chosen as a maximum of the F1 curve unless otherwise mentioned. The DC (Detected Count) metric is evaluated at the IoU threshold of 0.5.

### 6.2.3 Baseline model using no technique

As the first measurement is made training and prediction on unchanged data, images with resolution  $1\,600 \times 256$  pixels. The training was run with parameters from Section 6.2.2. It is the same model with the same result metrics as compared to U-net in Subsection 6.1.2.



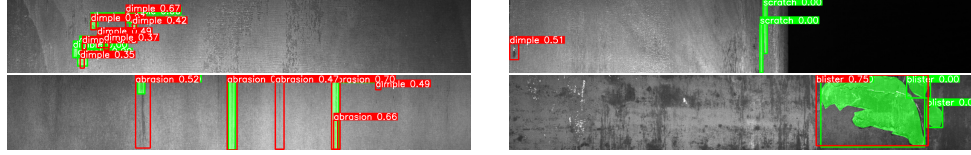
**Figure 6.5:** Prediction example for each defect type.

The top left example shows that not all defects are detected, which are quite small. The top right example shows that the underrepresented defect type scratch is not detected at all. In the left bottom example is visible correct detection of all defects. According to the ground truth, the left defect is small at only the top of the image, but we can see that defect same as the prediction is made, so maybe not all ground truth annotations must be completely accurate. In the right bottom example is shown one prediction of the blister defect over 2 ground truth annotations, and the third is mostly out of the predicted area, so it is not detected.

### 6.2.4 Slicing to 256 pixels sized slices

The first technique is slicing source images into squared slices with a  $256 \times 256$  pixels resolution. Slices are without overlap, and all negative samples are kept in the training set. The training was run with parameters from Section 6.2.2.

This technique should improve the prediction of small defects, which are especially annotations belonging to the dimple defect type.



**Figure 6.6:** Prediction example while using slicing to 256-pixel sized slices.

In the top left example, an increased number of detections of dimple annotations can be seen. Almost all of them are detected, although not exactly. In the right top example, the scratch defect type is again not detected, but one dimple prediction has appeared. According to the ground truth, no defect is there. But after zooming that image, we noticed some defect is there. In the example at the bottom left, two false positive predictions have occurred there, and one defect is detected twice, which is not desirable. The place on the product where the false positive abrasion prediction is has a darker line there if it is that unannotated defect, we can not say. The last example in the right bottom is almost without changes compared to using no techniques. The only difference is the higher confidence of prediction, and its bounding box is slightly larger.

Metric	DC	P	R	TP	FP
baseline	505	50.8 %	35.2 %	507	490
slice 256	694	36.1 %	48.4 %	697	1 235

Metric	DC-B	P-B	R-B	TP-B	FP-B
baseline	930	81.7 %	64.9 %	815	182
slice 256	1 081	68.3 %	75.4 %	1 320	612

Metric	mAP	mAP.50	mAP.75	$mAP50_s$	$mAP50_m$	$mAP50_l$
baseline	0.098	0.21	0.076	0.03	0.215	0.365
slice 256	0.133	0.257	0.12	0.11	0.21	0.402

**Table 6.2:** Tables show the number of detected defects, TP, and FP for slicing to 256-pixel-sized slices in the first two tables. The table below shows the results of mAP.

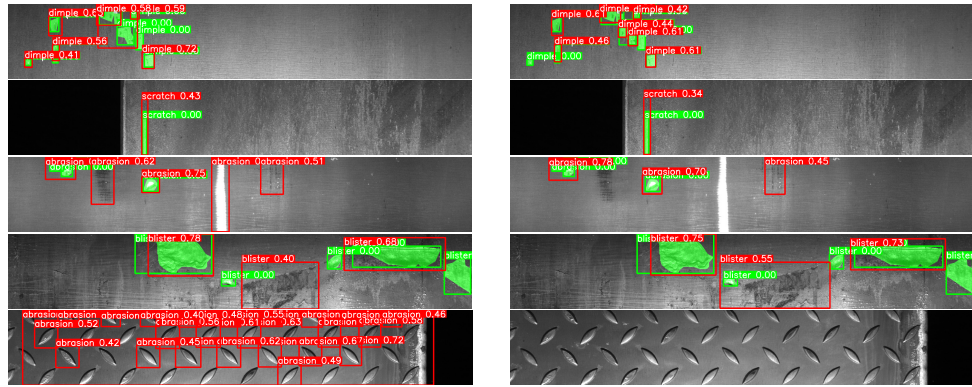
This technique is great for increasing the number of detected small defects but causes a very high number of false positives, even when some of them can be only unannotated defects.



This experiment shows that 5 % has higher mAP and precision but a little lower recall against the 20 % for minimal area ratio. For the following experiments is used 5 % of the minimal defect area. The reason is that the number of detected defects for 20 % is not that much higher than for 5 %. To increase the true positives, I'll try to find a more appropriate technique, and this technique will help to improve the precision of predictions.

### 6.2.6 Dataset with/without negative samples

This subsection determines the effect of the negative samples in the training set. Almost half of the training set are negative samples. The training was run for 200 epochs with the patience of 40, batch 32, randomly initialized weights, and with yolov5m.yaml config file for both. The first difference is already during training when the training time for the dataset, which consists only of positive samples, is very shorter. The training time for the dataset containing negative samples is 10 hours and 37 minutes, against the dataset from only positive samples, which took only 2 hours and 56 minutes.



**Figure 6.8:** Left side displays the prediction performed by the model trained on data containing only positive samples, and the right side shows the prediction made by the model trained on data with negative samples.

The first row with dimple defect type on the right side shows the prediction of more bounding boxes, which fit better the ground truth. In the case of only positive samples, there are fewer bigger predictions that include more ground truth annotations. The only difference for the scratch defect type is lower confidence in the case of using negative samples. The abrasion prediction on the row below has confused with different colors a defect on the left. The prediction of blister defect type affected the confidence of predictions, and false positive prediction has expanded and included one small ground truth annotation in the case of training with negative samples. The last row illustrates a sample that is according to the ground truth without defects. And there is a big difference. There are many false positives while using only positive samples. There is no false positive prediction if the negative samples are used while training.

Training with all negative samples helps increase the precision by 72 %

Metric	DC	P	R	TP	FP
negative	809	54.9 %	56.4 %	814	668
positive	739	31.8 %	51.5 %	742	1 588

Metric	DC-B	P-B	R-B	TP-B	FP-B
negative	1 145	74.6 %	79.8 %	1 105	377
positive	1 171	46.1 %	81.7 %	1 075	1 255

Metric	mAP	mAP.50	mAP.75	$mAP_{50_s}$	$mAP_{50_m}$	$mAP_{50_l}$
negative	0.171	0.378	0.143	0.198	0.376	0.419
positive	0.112	0.251	0.095	0.245	0.316	0.189

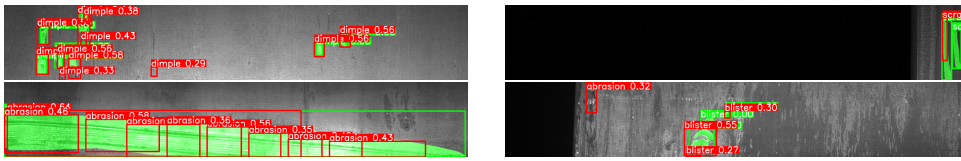
**Table 6.4:** The number of detected defects, TP, and FP for the model trained with only positive samples in the row named 'positive' and with negative samples in the row named 'negative' are shown in the first two tables. The last table shows the results of mAP.

with higher mAP for stricter metrics using IoU. The disadvantage of using negative samples is lower confidence of predictions, but it can be minimized by training for more epochs. In conclusion, using negative samples in training is very helpful to get better results.

### 6.2.7 Slicing to 400 pixels width slices

The next technique is slicing into bigger slices with a resolution of  $400 \times 256$  pixels. Slices are without overlap, and all negative samples are kept in the dataset. The train ran with parameters from Section 6.2.2.

This technique should help increase the accuracy of predicting small defects, but compared to slicing into 256-pixel-sized slices, this technique will create fewer slices and thus reduce the number of problems at their edges.



**Figure 6.9:** The prediction examples while using slicing to 400-pixel width slices.

At the top left image is shown dimple prediction, which predicted all defects with a few false positives according to the ground truth. if we take a closer look, is possible to see that the false positive predictions are at the places where something is different from the normal surface. In the case of scratch again, no significant changes. But for abrasion defect type there is shown how predictions are made in each slice. According to stricter metrics, only one of the predictions is a true positive. The rest of them are false positives because they are below IoU 0.5. But for the softer metrics, all of them are true positives, and that's why we use this metric because we think so too.



The last example of blister defect shows slightly good accuracy, but one false positive blister and one abrasion annotation exist. Both false positives look like they were confused by the color of the surface.

Metric	DC	P	R	TP	FP
slice 256	694	36.1 %	48.4 %	697	1 235
slice 400	842	40.6 %	58.7 %	907	1 328

Metric	DC-B	P-B	R-B	TP-B	FP-B
slice 256	1 081	68.3 %	75.4 %	1 320	612
slice 400	1 213	68.2 %	84.6 %	1 525	710

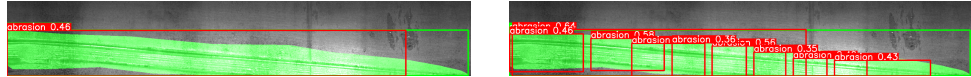
Metric	mAP	mAP.50	mAP.75	$mAP50_s$	$mAP50_m$	$mAP50_l$
slice 256	0.133	0.257	0.12	0.11	0.21	0.402
slice 400	0.166	0.38	0.113	0.358	0.355	0.339

**Table 6.5:** The first two tables show the number of detected defects, TP, and FP for the model trained with 256-pixel-sized slices and 400-pixel width slices. The last table shows the results of mAP.

Slicing to 400-pixel width slices improved predictions of small defects and helped to detect more defects. The  $mAP50_s$  column shows that mAP for small objects is significantly better. But the number of false positives has slightly increased too. And the detection accuracy for large defects is not ideal, as shown in the abrasion example.

### 6.2.8 Slicing to 800 pixels width slices

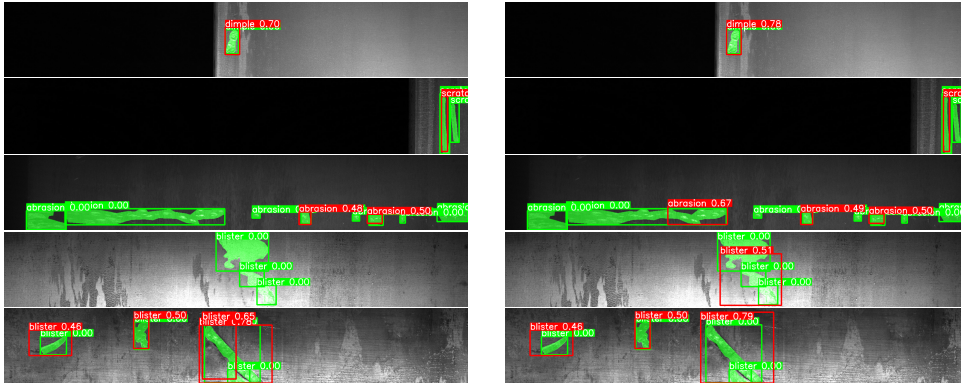
This technique tries slicing images into only two slices with a resolution of  $800 \times 256$  pixels to keep better results for the detection of small defects but reduce problems with large ones. In the training set are, all negative samples kept. This subsection also describes the difference in using an overlap in prediction, which should help detect defects at the edge of the slices with higher accuracy. The training was run with parameters from Section 6.2.2



**Figure 6.10:** The prediction example shows the difference between using slicing to 400-pixel width slices on the right side and 800-pixel width slices on the left side for large defects.

Using slicing to larger slices eliminated the multiple predictions of the same ground truth annotation but with lower confidence.





**Figure 6.11:** This Figure compares predictions made with slicing to 800 width slices, but on the left are predictions made without an overlap, and on the right side are predictions made with an overlap of 0.3.

The first row in Figure 6.11 illustrates the difference on the dimple defect type, which helped increase the prediction confidence. The second row displays the scratch defect, but this defect type is in the testing set, never in the middle of the image, which is the only place where this technique can make some changes for slicing to 800-pixel width slices. So all annotations of all defect types, which are not at the edge of the slices, can not be affected by this technique. The third row displays the case when using an overlap in prediction can help to detect defects that are not detected without overlap. Unfortunately, this annotation is evaluated here as a false positive because its IoU is lower than 0.5, but it is a true positive for softer metrics. The next row shows the blister defect, which is not detected at all using prediction without prediction, but with an overlap, it predicted one blister annotation, which is taken as one true positive. And at the last row can be seen in the case of not using an overlap duplication of prediction at the edge of the slices, but while using an overlap only one, a slightly larger prediction is performed.

Metric	DC	P	R	TP	FP
slice 400	842	40.6 %	58.7 %	907	1 328
slice 800	788	63.1 %	55 %	790	462

Metric	DC-B	P-B	R-B	TP-B	FP-B
slice 400	1 213	68.2 %	84.6 %	1 525	710
slice 800	1 105	83.1 %	77.1 %	1 041	211

**Table 6.6:** Tables demonstrating the number of detected defects, TP, and FP for the model trained with 400-pixel width slices and 800-pixel width slices.

Metric	mAP	mAP.50	mAP.75	$mAP50_s$	$mAP50_m$	$mAP50_l$
slice 400	0.166	0.38	0.113	0.358	0.355	0.339
slice 800	0.182	0.395	0.137	0.272	0.421	0.403

**Table 6.7:** The table displays the results of mAP.

Metric	DC	P	R	TP	FP
no overlap	775	63.6 %	54 %	778	446
overlap	788	63.1 %	55 %	790	462

Metric	DC-B	P-B	R-B	TP-B	FP-B
no overlap	1 083	84 %	75.5 %	1 028	196
overlap	1 105	83.1 %	77.1 %	1 041	211

Metric	mAP	mAP.50	mAP.75	$mAP50_s$	$mAP50_m$	$mAP50_l$
no overlap	0.179	0.386	0.135	0.269	0.401	0.393
overlap	0.182	0.395	0.137	0.272	0.421	0.403

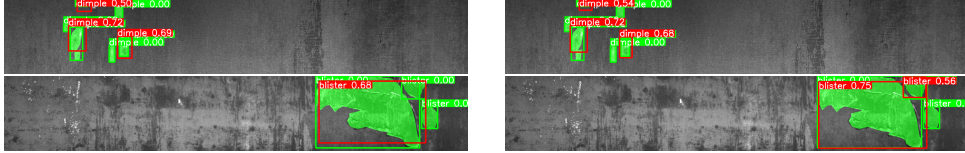
**Table 6.8:** The first two tables show the number of detected defects, TP, and FP for the model trained with 800-pixel width slices where one row is the case of prediction without overlap and the second one is with overlap. The third table shows the results of mAP.

Using slicing to 800-pixel width slices versus 400-pixel width slices helps to reduce multiple predictions for large defects and slightly decreases the recall but with significantly higher precision. And still helps to detect small defects against not using any slicing. Deciding which of these techniques is better depends on what is more important in the current task. If maximizing the recall, slicing to 400-pixel width slices is better for sure. But if we are worried about a significantly higher number of false positives, slicing to 800-pixel width slices is the right technique.

Has been confirmed that performing predictions with overlap solves many detection problems at the edges of the slices.

### 6.2.9 Using a pre-trained model or random initialization

This subsection compared the training without any techniques, with only different, which is using randomly initialized weights of a model or using an already pre-trained YOLO model. Pre-trained models from YOLO are pre-trained on the COCO dataset, which contains 80 classes, and most of the images have a small count of annotations, which mostly have medium to large areas.



**Figure 6.12:** Prediction example of using random initialization on the left side and using pre-trained model yolov5m.pt on the right side.

The first row of Figure 6.12 has almost no difference, only slightly narrower predicted bounding boxes. But the second row shows the case when the pre-trained model predicted defects more exactly.

Using a pre-trained model should be better if this dataset's data has similar features. Otherwise, it may have a bad effect on the accuracy of the model. The pre-trained model might be more appropriate since the COCO dataset, and this one has low data resolution and mostly fewer annotations per image.

Metric	DC	P	R	TP	FP
random init	505	50.9 %	35.2 %	507	490
yolov5m.pt	540	66.3 %	37.7 %	540	274

Metric	DC-B	P-B	R-B	TP-B	FP-B
random init	930	81.7 %	64.9 %	815	182
yolov5m.pt	831	89.7 %	58 %	730	84

Metric	mAP	mAP.50	mAP.75	$mAP50_s$	$mAP50_m$	$mAP50_l$
random init	0.098	0.21	0.076	0.03	0.215	0.365
yolov5m.pt	0.116	0.256	0.088	0.032	0.261	0.408

**Table 6.9:** The first two tables show the number of detected defects, TP, and FP for a model trained from random initialization or from pre-trained yolov5m.pt model. The table below shows the results of mAP.

Using a pre-trained model increased the precision of predictions and help to detect more defects while evaluating by stricter metrics. The next improvement reached by the pre-trained model is decreased number of false positives. In conclusion, using the pre-trained model really helped to get more satisfactory results.

### 6.2.10 Pre-trained model size

This subsection compares the metrics between individual pre-trained models. The main difference between the pre-trained models is in the complexity of the models. The yolov5m.pt model is approximately 2 times more complex than yolov5n.pt, and yolov5x.pt is approximately almost 5 times more complex than yolov5n.pt. The higher complexity of the model can help to increase precision when the data have a higher resolution, or the specific features of defects are very difficult.

Metric	DC	P	R	TP	FP
yolov5n	566	58.2 %	39.5 %	567	407
yolov5m	540	66.3 %	37.7 %	540	274
yolov5x	519	67 %	36.2 %	519	256

Metric	DC-B	P-B	R-B	TP-B	FP-B
yolov5n	906	83.2 %	63.2 %	810	164
yolov5m	831	89.7 %	58 %	730	84
yolov5x	776	90.2 %	54.1 %	699	76

Metric	mAP	mAP.50	mAP.75	$mAP_{50_s}$	$mAP_{50_m}$	$mAP_{50_l}$
yolov5n	0.115	0.252	0.09	0.053	0.268	0.389
yolov5m	0.116	0.256	0.088	0.032	0.261	0.408
yolov5x	0.117	0.23	0.101	0.049	0.23	0.389

**Table 6.10:** The first two tables show the number of detected defects, TP, and FP for three different-sized pre-trained yolov5 models. The table below shows the results of mAP.

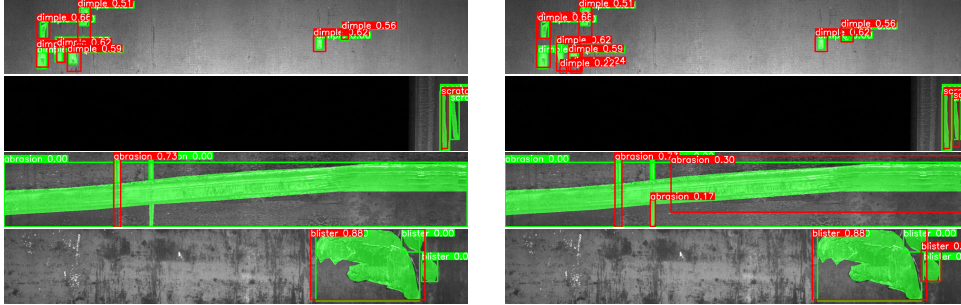
All 3 models were trained with parameters from Section 6.2.2. Training times were 7 hours and 52 minutes for the nano model, 7 hours and 59 minutes for the medium, and 21 hours and 46 minutes for the extra large model. Trained on origin data, same as in Section 6.2.3.

From the results can be seen that the smaller model has better performance on the Severstal dataset. A possible explanation for this observation is that this dataset contains only 4 classes, and the number of annotations on the images is typically less than 5. Plus, these annotations are very often of the same class. Additionally, the features of the defects in this dataset are quite obvious and not very complicated, combined with the lower resolution are these probable reasons. In summary, the complexity of the model has a minimal effect on its performance.

### 6.2.11 Using various confidence threshold

This subsection uses the model from Subsection 6.2.8, which is trained on 800-pixel width slices. But predictions are performed with different confidence thresholds. Decreasing confidence thresholds cause using predictions with

lower confidence in results. That will increase the number of false positives. The number of true positives should get higher too. Using a lower confidence threshold is useful if the number of true positives increases. Hopefully, the precision remains the same or is reduced as little as possible.



**Figure 6.13:** Prediction example of using confidence threshold corresponding to F1 max on the left side and using 0.15 as confidence threshold on the right side.

The first row in Figure 6.13 shows that a lower confidence threshold produces more false positives. The remaining two rows show when a lower confidence threshold helps detect more defects.

Metric	0.311	0.3	0.25	0.2	0.15	0.1
DC	788	791	808	824	827	833
P	63.1 %	62.4 %	58.8 %	55.1 %	50.1 %	53.8 %
R	55 %	55.2 %	56.3 %	57.5 %	57.7 %	58.1 %
TP	790	793	810	826	831	977
FP	462	478	567	672	799	839
DC-B	1 105	1 120	1 162	1 205	1 244	1 290
P-B	83.1 %	82.7 %	80.1 %	77.6 %	73.6 %	69.3 %
R-B	77.1 %	78.1 %	81 %	84 %	86.8 %	90 %
TP-B	1 041	1 051	1 103	1 162	1 200	1 258
FP-B	211	220	274	336	430	558
mAP	0.182	0.181	0.18	0.185	0.186	0.173
mAP.50	0.395	0.394	0.4	0.422	0.426	0.413

**Table 6.11:** Table with differences between results predicted with various confidence thresholds. In the columns header are values of confidence thresholds that were used, where the first one is the F1 max.

According to these results, the best confidence threshold for this dataset is 0.2 if the goal is to achieve the highest possible mAP and precision. If even a slightly lower precision and mAP do not matter, the 0.1 confidence threshold is the best choice.

### 6.2.12 Comparison of fine-tuning measurements

This section shows the differences in results between the most relevant measurements.

Metric	M1	M2	M3	M4
DC	505	694	842	833
P	50.8 %	36.1 %	40.6 %	53.8 %
R	35.2 %	48.4 %	58.7 %	58.1 %
TP	507	697	907	977
FP	490	1 235	1 328	839
DC-B	930	1 081	1 213	1 290
P-B	81.7 %	68.3 %	68.2 %	69.3 %
R-B	64.9 %	75.4 %	84.6 %	90 %
TP-B	815	1 320	1 525	1 258
FP-B	182	612	710	558
mAP	0.098	0.133	0.166	0.173
mAP.50	0.21	0.257	0.38	0.413

**Table 6.12:** Table with result metrics for the public dataset, where measurements labeled from M1 to M4 are described below.

M1 - prediction made and model trained without any technique

M2 - model trained on sliced samples with a width and height of 256 pixels, 5 % minimal area ratio, and all negative samples.

M3 - model trained on sliced samples with a width of 400 pixels, 5 % minimal area ratio, and containing all negative samples.

M4 - model trained on sliced samples with a width of 800 pixels, 5 % minimal area ratio, and containing all negative samples and prediction performed with a confidence threshold of 0.1

From Table 6.12 can be seen that slicing to 400-pixel or 800-pixel width slices combined with negative samples in the training set, 5 % minimal defect area ratio, and decreased confidence threshold performed much better results. Deciding which of these is better depends on the importance of detecting more defects with respect to precision.

### 6.2.13 Evaluation of results

From the measurements above, the following conclusions were reached. Slicing source images is important because data contains many small defects. Slicing to slices with a width of 800 pixels and height of 256 pixels improved the DC-B from 930 to 1244, which is a 33.8 % improvement. The technique met expectations by improving the detection of small defects and not just them. Performing prediction with overlap has proven to be an effective way how to deal with problems at the edges of slices, like creating more annotations for one defect or recognizing the defect at the edge of the slices. In this dataset, a lower minimal defect area ratio helps to increase the precision of predictions. Specifically, it has improved the mAP by 15.5 % and precision by 8.2 %. Negative samples are necessary while training to reduce false positive predictions. Using negative samples increased the precision from 46.1 % to 74.6 %, which is a 61.8 % improvement. The dataset contains almost half of the negative samples. Increasing the number of negative samples may help to achieve even better results. Using a nano-pre-trained YOLO model reaches the best performance among pre-trained models. It has detected 63.2 % of ground truth defects instead of medium-sized, which has 58 %. But in summary, differences between individual sizes are minimal. So our decision to use randomly initialized weights of a medium-sized model was not appropriate for this dataset.

In conclusion, our improvements of the YOLO model increased the mAP by 76.5 % (from 0.098 to 0.173). The recall grew from 35.2 % to 58.7 % for the metric using IoU 0.5 threshold. In the case of the benevolent metric, it increased from 64.9 % to 90 %. With this result, such a model could be applicable in manufacturing because not all defects are correctly labeled. Which could be further inspected in the next work.





## Chapter 7

### DNAI dataset

This chapter provides the results of measurements on the DNAI dataset. Experiments using various techniques are described in Section 7.1, where subsections contain results for relevant techniques. The obtained results are described in Subsection 7.1.13. YOLO with U-net is compared while using slicing to 640-pixel-sized slices of source images for this dataset in Section 7.2. All figures with example predictions in this chapter are consistent, with green annotations indicating ground truth and red-colored annotations representing predictions.

#### 7.1 Fine-tuning

This section describes experiments on the private dataset. That dataset is divided into 3 datasets, each containing only annotations of one defect type. Material remains jagged edge, and material on edge are the defect types for which datasets are created. For each dataset, measurements are made to determine which techniques help improve precision for that defect type. The testing set for material remains has 2 151 annotations, the material on edge 267 and jagged edge 61.

The experiments are performed with the following techniques. Slicing source images into smaller samples with the goal of improving the detection of small defects. While slicing, find out the influence of an overlap of the slices. If the annotation is divided into multiple samples, find out how much of the defect area needs to be on the sample to keep the annotation there. And determine the effect of the negative samples in the training set on its detection capabilities. Also, investigate whether using a pre-trained model is more suitable than randomly initialized weights. See the effect of a more complex model on model performance.

These techniques are described in full detail for material remains defect, and for the other two defect types are measurements compared in summary Subsection 7.1.12.

### ■ 7.1.1 Data preparation

Creation of various datasets with different parameters such as slice size, overlap ratio, minimal object area, or percentage of negative samples in the dataset. These datasets need to be converted to yolov5 format as mentioned in the Chapter 5. Datasets for measurements were generated with the following parameters:

- Slice size 640 or 1280 pixels
- Overlaps of sliced images 0 %, 10 %, or 25 %
- Minimal area of defect in sliced image 5 % or 20 %
- Percentage of negative samples in dataset 0 %, 30 %, or keep all

### ■ 7.1.2 Training and prediction parameters

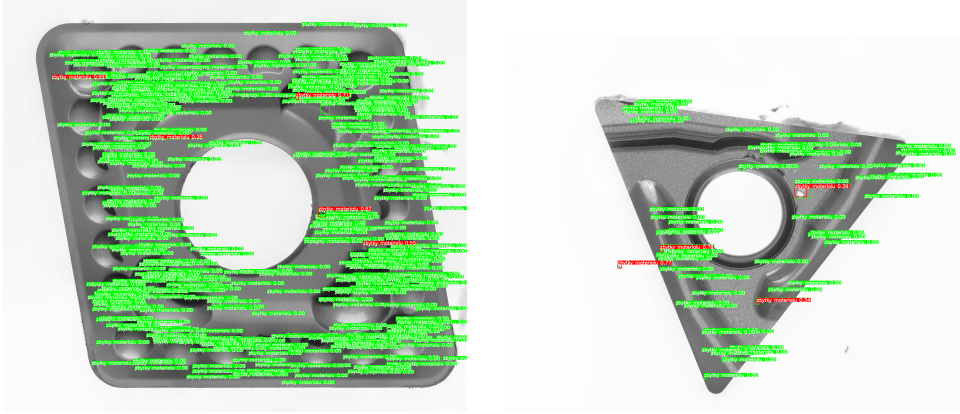
Following results were reached with YOLOv5 v6.2.3[15], SAHI v.0.11.6[1], Python-3.9.6 with torch-1.12.1+cu116. Training parameters used:

- Batch: -1 for automatically computing batch size for an assigned GPU by Metacentrum [20].
- Weights: yolov5m.pt pre-trained model
- Cache: RAM
- Epochs: 1000
- Patience: 300, to stop training if the better result is not reached in the last 300 epochs

Unless otherwise mentioned, the confidence threshold is chosen as a maximum of the F1 curve. The DC (Detected Count) metric is evaluated at the IoU threshold of 0.35 for a jagged edge, 0.33 for material on edge, and 0.5 for the material remains defect type.

### 7.1.3 Baseline model

Firstly training is run on origin data down-scaled to 2560 pixels. YOLO pre-train models are trained on images with a size of 640 pixels. The material remains defect has a very small area, and down-scale origin images to 640-pixel-sized samples would cause all small defects to disappear. So keeping a higher training resolution ensure that all annotated defects are kept. The disadvantage of training at higher resolution is longer training and prediction time. For this dataset containing only 113 images, it is not too big a problem.



**Figure 7.1:** Prediction example of baseline model on the private dataset.

The example predictions in Figure 7.1 illustrate a problem with an extremely small area of this defect type, which causes no detection of most of the defects.

Metric	DC	P	R	TP	FP
baseline	15	20.3 %	0.7 %	15	59

Metric	DC-B	P-B	R-B	TP-B	FP-B
baseline	50	68.9 %	2.3 %	51	23

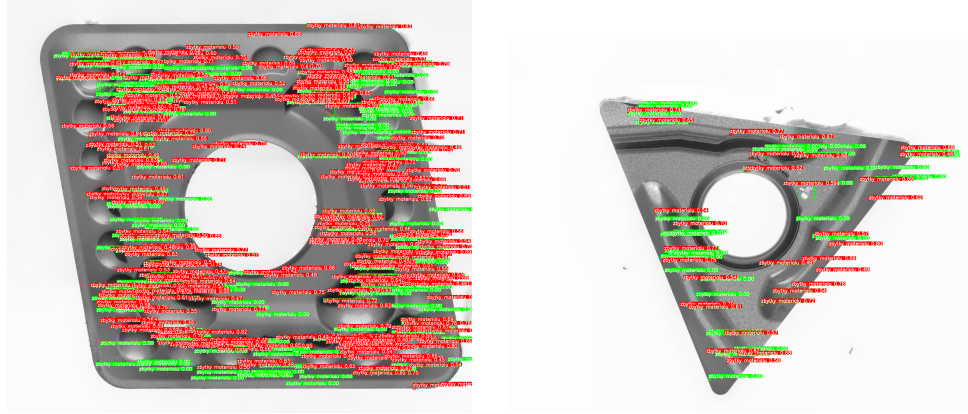
Metric	mAP	mAP.50	mAP.75	$mAP50_s$	$mAP50_m$	$mAP50_l$
baseline	0.003	0.005	0.0	0.01	0.028	0.026

**Table 7.1:** The first two tables show the number of detected defects, TP, and FP for the baseline model. The table below shows the results of mAP. The metrics in columns are described in Section 2.3.

Numeric results confirm the observation that almost all of the defects are not detected. Using any technique to improve the ability to detect small defects is necessary.

### 7.1.4 Slicing to 1280 pixels

The first technique used is slicing source images to 1 280 pixel-sized samples without overlap for training with parameters from Subsection 7.1.2. It should help to recognize these small defects because then small defects take up more space in ratio to the size of the product surface in one sample.



**Figure 7.2:** Prediction example of the model using slicing to 1 280 pixel-sized slices.

Figure 7.2 shows an increased number of predictions, most of which are true positives. But many ground truth defects are still not detected.

Metric	DC	P	R	TP	FP
baseline	15	20.3 %	0.7 %	15	59
slice 1 280	1 048	76.6 %	48.7 %	1 048	321

Metric	DC-B	P-B	R-B	TP-B	FP-B
baseline	50	68.9 %	2.3 %	51	23
slice 1 280	1 129	82.4 %	52.5 %	1 128	241

Metric	mAP	mAP.50	mAP.75	$mAP_{50_s}$	$mAP_{50_m}$	$mAP_{50_l}$
baseline	0.003	0.005	0.0	0.01	0.028	0.026
slice 1 280	0.171	0.415	0.1	0.433	0.471	0.298

**Table 7.2:** The first two tables show the number of detected defects, TP, and FP for the baseline model compared to using 1 280-pixel-sized slices. The table below shows the results of mAP.

All metrics are consistent with achieving more true positive predictions with high precision, but recall is still low.

### 7.1.5 Slicing to 640 without overlap

This subsection uses smaller slices with a resolution of  $640 \times 640$  pixels. Smaller slices should further improve the detection of more small defects. Slices are without overlap, and only positive samples are used for training with parameters from Subsection 7.1.2.



**Figure 7.3:** Comparison of predictions for slicing to 1 280 on the left side and 640 pixel-sized slices on the right side.

Figure 7.3 shows that slicing to 640 pixel-sized slices helped detect more defects but with more false positive predictions. And the problem with detecting dirt on the camera as a defect is shown in the example at the bottom right corner.

Metric	DC	P	R	TP	FP
slice 1 280	1 048	76.6 %	48.7 %	1 048	321
slice 640	1 504	68.5 %	69.9 %	1 504	692

Metric	DC-B	P-B	R-B	TP-B	FP-B
slice 1 280	1 129	82.4 %	52.5 %	1 128	241
slice 640	1 569	71.6 %	72.9 %	1 573	623

**Table 7.3:** Tables demonstrating the number of detected defects, TP, and FP for the model using slicing into 1 280 pixel-sized and 640 pixel-sized slices.

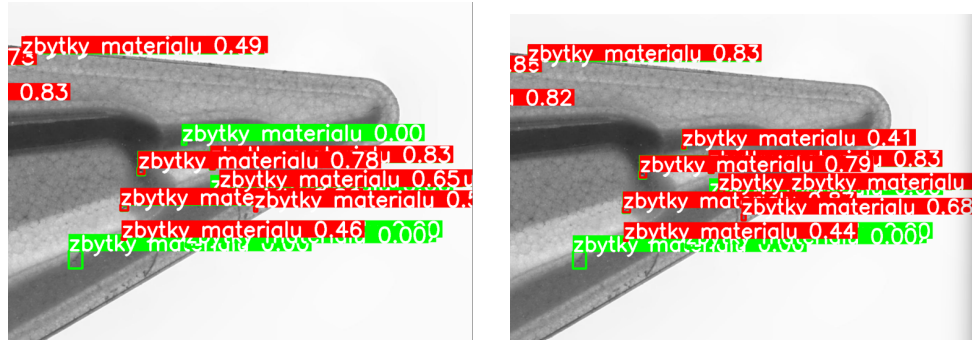
Metric	mAP	mAP.50	mAP.75	$mAP50_s$	$mAP50_m$	$mAP50_l$
slice 1 280	0.171	0.415	0.1	0.433	0.471	0.298
slice 640	0.261	0.573	0.185	0.61	0.423	0.122

**Table 7.4:** The table displays the results of mAP.

Slicing to 640 pixel-sized slices helps increase the number of detected defects with slightly lower precision. The worse result is only with large defects, which make up only about 0.5 % of all remaining material annotations.

### 7.1.6 Slicing to 640 with overlap

Using the same slicing to 640 pixel-sized slices but with an overlap of 25 %. For overlapping slices should be easier to recognize the defects which are at the edge of the slice.



**Figure 7.4:** Comparison of predictions for 640 pixel-sized slices without overlap on the left and with an overlap of 25 % on the right side.

Figure 7.3 shows that slicing help detects a defect that is not detected without overlap. Using overlap even increased the confidence of some predictions on the edges of slices.

Metric	DC	P	R	TP	FP
no overlap	1 504	68.5 %	69.9 %	1 504	692
with overlap	1 598	68.6 %	74.3 %	1 600	731

Metric	DC-B	P-B	R-B	TP-B	FP-B
no overlap	1 569	71.6 %	72.9 %	1 573	623
with overlap	1 639	70.3 %	76.2 %	1 639	692

Metric	mAP	mAP.50	mAP.75	$mAP50_s$	$mAP50_m$	$mAP50_l$
no overlap	0.261	0.573	0.185	0.61	0.423	0.122
with overlap	0.296	0.627	0.222	0.647	0.471	0.224

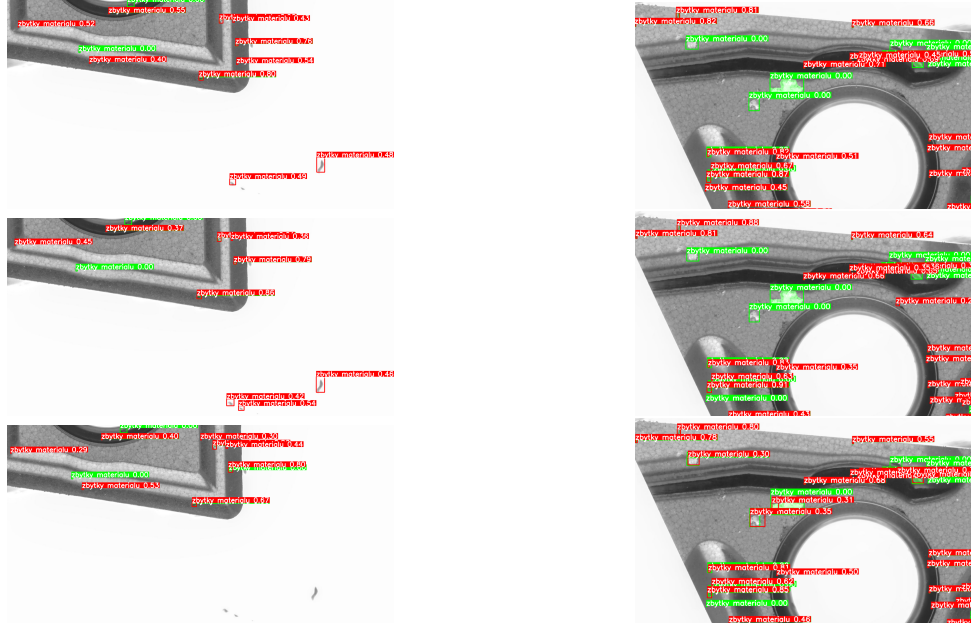
**Table 7.5:** The first two tables show the number of detected defects, TP, and FP for the model using slicing with an overlap of 0.25 and for the model without an overlap. The table below shows the results of mAP.

From tables with results and observations can be seen that using overlap help to increase recall and keep the same or slightly decrease the precision.

### 7.1.7 Dataset with negative samples

This subsection determines the effect of the negative samples in the training set. The private dataset does not contain any negative samples. As negative samples, we consider samples after slicing which are without defects. Three variants of negative samples in the training set are used. Firstly training set with no negative samples, then the training set consists of 30 % negative samples and lastly training set which keeps all available negative samples.

In Figure 7.5 are shown 2 examples. The one on the left demonstrates elimination predicting the dirt on the camera as defects. Interestingly while 30 % of the training set are negative samples, it misclassified even more. The example on the right illustrates examples of predictions that are not detected while using negative samples. On the other hand, larger defects are at least partially detected in this case which without negative samples were not.



**Figure 7.5:** Comparison of predictions for the training set with 30 % in the first row and keeping all available negative samples in the last row.

Metric	DC	P	R	TP	FP
0 %	1 598	68.6 %	74.3 %	1 600	731
30 %	1 627	68.5 %	75.6 %	1 628	747
all	1 625	68.6 %	75.5 %	1 626	744

Metric	DC-B	P-B	R-B	TP-B	FP-B
0 %	1 639	70.3 %	76.2 %	1 639	692
30 %	1 664	70.2 %	77.4 %	1 668	707
all	1 684	70.8 %	78.3 %	1 678	692

Metric	mAP	mAP.50	mAP.75	$mAP50_s$	$mAP50_m$	$mAP50_l$
0 %	0.296	0.627	0.222	0.647	0.471	0.224
30 %	0.301	0.631	0.225	0.671	0.474	0.083
all	0.307	0.631	0.241	0.669	0.421	0.367

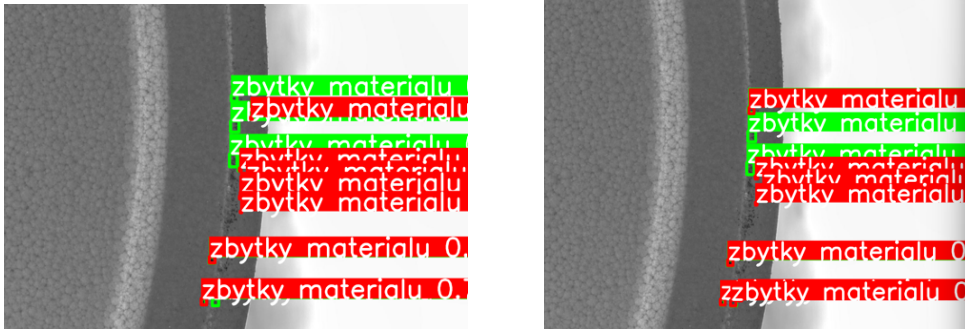
**Table 7.6:** The first two tables show the number of detected defects, TP, and FP for the training set with 0 %, 30 %, or keeping all available negative samples. The table below shows the results of mAP.

In summary, using negative samples has a positive effect on results. It eliminates false positives, which can be even better if more images of these products without defects were available. And predict defects or at least part of them which were not detected.



### 7.1.8 Minimal area ratio of defect

A comparison of keeping defect annotations on slices when at least 5 % of the area is in the slice or up to 20 % is enough is determined in this subsection. Slices with the size of 640 pixels are used with an overlap of 25 % and trained with parameters from Subsection 7.1.2. A higher minimal ratio of defect area may cause unannotated parts of some defects. Due to the reason that they are below this ratio. A model trained on such slices may not predict these parts of the defects. This parameter should help with defects on the edges of the slices similarly, like using overlapping slices.



**Figure 7.6:** Comparison of predictions for 20 % minimal area ratio on the left and 5 % on the right side.

Example prediction in Figure 7.6 shows that using a 5 % minimal area ratio help to detect more material remains defects. Because the bounding boxes fit the real defect well for the material remains. Keeping even a small 5 % of the annotation area can help to detect more defects.

Metric	DC	P	R	TP	FP
20 %	1 497	72.6 %	69.6 %	1 498	564
5 %	1 680	68 %	78.1 %	1 681	792

Metric	DC-B	P-B	R-B	TP-B	FP-B
20 %	1 559	75.9 %	72.5 %	1 566	496
5 %	1 702	68.7 %	79.1 %	1 699	774

Metric	mAP	mAP.50	mAP.75	$mAP_{50_s}$	$mAP_{50_m}$	$mAP_{50_l}$
20 %	0.287	0.595	0.225	0.632	0.4	0.304
5 %	0.303	0.642	0.241	0.68	0.472	0.044

**Table 7.7:** The first two tables show the number of detected defects, TP, and FP for the model trained on data using 20 % or 5% of minimal area ratio. The table below shows the results of mAP.

In conclusion, for the remaining material, 5 % of the minimal ratio provides significantly more detected ground truth annotations.

### 7.1.9 Pre-train model size

This subsection checks whether using a more complex pre-trained model positively affects its performance. Compared are models yolov5m.pt (medium-sized) a yolov5x.pt (extra large sized). The more complex model should help with learning specific features, which are very diverse or with high-resolution input.

Metric	DC	P	R	TP	FP
yolov5m	1 498	70.6 %	69.6 %	1 499	625
yolov5x	1 446	69.4 %	67.2 %	1 446	638

Metric	DC-B	P-B	R-B	TP-B	FP-B
yolov5m	1 537	72.6 %	71.5 %	1 543	581
yolov5x	1 504	72.5 %	69.9 %	1 510	574

Metric	mAP	mAP.50	mAP.75	$mAP_{50_s}$	$mAP_{50_m}$	$mAP_{50_l}$
yolov5m	0.27	0.588	0.202	0.621	0.479	0.107
yolov5x	0.256	0.569	0.18	0.598	0.475	0.149

**Table 7.8:** The first two tables show the number of detected defects, TP, and FP for medium-sized and extra-large pre-trained models. The table below shows the results of mAP.

Using the extra-large model has barely improved the prediction of large defects but has degraded the accuracy of most of the remaining material annotations. Mostly there are extremely small differences in the predictions of both models. Because the extra-large model is more consuming of computing resources and its results are moderately worse, the medium-sized model provides better performance for material remains defect type in this dataset.

### 7.1.10 Using random initialization or pre-trained model

Here are compared models where one is trained from randomly initialized weights and the other from a pre-trained model yolov5m. YOLO pre-trained models are trained on the COCO dataset, which contains images with 80 classes and only a few objects per image.

Metric	DC	P	R	TP	FP
pre-trained	1 498	70.6 %	69.6 %	1 499	625
random init	1 583	70.8 %	73.6 %	1 584	654

Metric	DC-B	P-B	R-B	TP-B	FP-B
pre-trained	1 537	72.6 %	71.5 %	1 543	581
random init	1 613	71.9 %	75 %	1 609	629

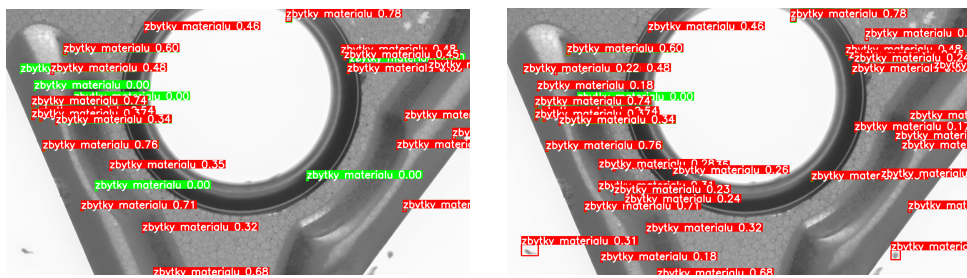
Metric	mAP	mAP.50	mAP.75	$mAP_{50_s}$	$mAP_{50_m}$	$mAP_{50_l}$
pre-trained	0.27	0.588	0.202	0.621	0.479	0.107
random init	0.284	0.615	0.205	0.651	0.49	0.144

**Table 7.9:** The first two tables show the number of detected defects, TP, and FP for the model trained from random initialization or from pre-trained yolov5m weights. The table below shows the results of mAP.

Differences in predictions between these two models are minimal. Overall, the model trained from randomly initialized weights provides slightly better performance in the number of detected defects, even in the accuracy of predictions.

### 7.1.11 Using various confidence thresholds

Testing the effect of prediction at different confidence thresholds is described here. Predictions below confidence thresholds are ignored, which causes decreasing number of false positives and may reduce the number of true positives in case of a higher confidence threshold. On the other hand, decreasing the confidence threshold increases the overall number of predictions which can increase true positives but false positives too.



**Figure 7.7:** Comparison of predictions using F1 max as confidence threshold on the left and confidence threshold 0.1 on the right side.

Figure 7.7 shows the correct detection of more defects using a lower confidence threshold and false positive detections. Dirt on the camera is detected as a defect too.

Metric	0.313	0.25	0.20	0.15	0.10
DC	1 680	1 754	1 816	1 846	1 902
P	68 %	63.4 %	59.8 %	54.8 %	48.9 %
R	78.1 %	81.5 %	84.4 %	85.8 %	88.4 %
TP	1 681	1 755	1 817	1 847	1 903
FP	792	1 011	1 223	1 522	1 987
DC-B	1 702	1 776	1 842	1 884	1 946
P-B	68.7 %	64.1 %	60.5 %	55.8 %	50 %
R-B	79.1 %	82.6 %	85.6 %	87.6 %	90.5 %
TP-B	1 699	1 773	1 839	1 881	1 944
FP-B	774	993	1 201	1 488	1 946
mAP	0.303	0.308	0.309	0.302	0.3
mAP.50	0.642	0.653	0.66	0.655	0.654

**Table 7.10:** Table with differences between results predicted with various confidence thresholds. Metrics in rows are explained in section 2.3. In the columns header are values of confidence thresholds that were used, where the first one is the F1 max.

From Table 7.10 with the results can be seen that decreasing the confidence threshold can help to detect up to 90 % of the defects but with many false positives, which cause a reduction of precision. If the goal is to detect as many defects as possible, using the 0.1 threshold is acceptable. But in the case precision should be greater, the 0.2 threshold is a suitable option, in my opinion.

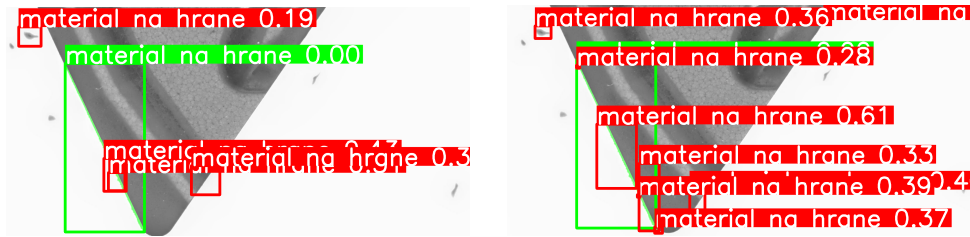
### 7.1.12 Comparison and summary of measurements

This subsection shows the summary results of measurements for all 3 defect types. There is only a summary Table 7.11 for material remains. In the case of jagged edge and material on edge, Tables 7.12 and 7.13 contain their results. Several prediction examples of material on the edge and jagged edge are provided here.

Metric	M1	M2	M3	M4	M5	M6	M7
DC	15	1 048	1 504	1 598	1 627	1 625	1 680
P	20.3 %	76.6 %	68.5 %	68.6 %	68.5 %	68.6 %	68 %
R	0.7 %	48.7 %	69.9 %	74.3 %	75.6 %	75.5 %	78.1 %
TP	15	1 048	1 504	1 600	1 628	1 626	1 681
FP	59	321	692	731	747	744	792
DC-B	50	1 129	1 569	1 639	1 664	1 684	1 702
P-B	68.9 %	82.4 %	71.6 %	70.3 %	70.2 %	70.8 %	68.7 %
R-B	2.3 %	52.5 %	72.9 %	76.2 %	77.4 %	78.3 %	79.1 %
TP-B	51	1 128	1 573	1 639	1 668	1 678	1 699
FP-B	23	241	623	692	707	692	774
mAP	0.003	0.171	0.261	0.296	0.301	0.307	0.303
mAP.50	0.05	0.415	0.573	0.627	0.631	0.631	0.642

**Table 7.11:** Table with result metrics for material remains measurements labeled from M1 to M7, which are described at the end of this section.

In Table 7.11, the results show that using slicing with an overlap of source images, including all negative samples combined with 5 % of minimal defect area ratio, improved the model's performance the most.



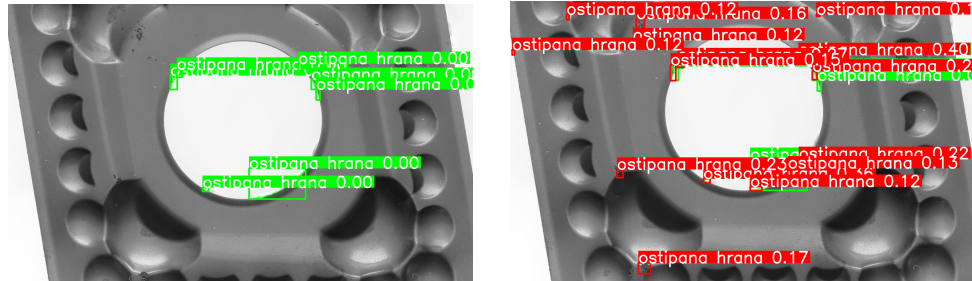
**Figure 7.8:** Prediction example of material on edge on the left side baseline method without any improvement and on the right side using slicing to 640-pixel-sized slices with overlap.

Example prediction in Figure 7.8 on the left side where baseline model predicted only part of the defect. The dirt on the camera confused model and another false positive prediction occurs there. The improved model on the right side performed much more predictions. The problem with dirt persists, but a larger part of the ground truth defect is detected correctly. Overall, many of the predictions are false positives according to stricter metrics.

Metric	M1	M2	M3	M4	M5	M6	M7
DC	4	49	78	101	94	87	83
P	7.3 %	23 %	25.3 %	17.8 %	19.8 %	20.9 %	25.3 %
R	1.5 %	18.4 %	29.2 %	37.8 %	35.2 %	32.6 %	31.1 %
TP	4	50	78	105	96	88	83
FP	51	168	230	484	389	334	245
DC-B	24	87	123	199	178	179	160
P-B	51 %	55 %	41.2 %	36.2 %	37.5 %	41.5 %	47.9 %
R-B	9 %	32.6 %	46.1 %	74.5 %	66.7 %	67 %	60 %
TP-B	28	120	127	213	182	175	157
FP-B	27	98	181	376	303	247	171
mAP	0.0	0.011	0.021	0.011	0.015	0.011	0.016
mAP.50	0.0	0.029	0.056	0.039	0.05	0.038	0.05

**Table 7.12:** Table with result metrics for material on edge measurements labeled from M1 to M7, which are described at the end of this section.

Table 7.12 reflects observations that slicing to 640-pixel-sized with overlap helps detect more annotations of material on edge. Using negative samples helps decrease the number. The minimal area ratio needs to be greater because the area of bounding boxes is much larger than the actual size of the defect hence the 20 % minimal area ratio is a little better. Overall, very poor precision of predictions with slightly better recall.



**Figure 7.9:** Prediction example of jagged edge on the left side baseline method without any improvement and on the right side using slicing to 640-pixel-sized slices with overlap.

Predictions of the baseline model on the left side in Figure 7.9 show that using no technique causes no predictions to be performed. In the case of the improved model, slicing to 640-pixel-sized slices with overlap and keeping all available negative samples performs much more predictions. These predictions detect almost all ground truth defects but with many false positives. Some false positives are not even at the edges of the product. If we take a closer look at some false positive predictions, they are where some defects really are, but according to the ground truth, it is unannotated. In this dataset are more source images with unannotated defects.

Metric	M1	M2	M3	M4	M5	M6	M7
DC	5	23	29	38	40	31	19
P	75 %	46.9 %	14.1 %	18.7 %	22.8 %	37.1 %	40.4 %
R	8.2 %	37.7 %	47.5 %	62.3 %	65.6 %	50.8 %	31.2 %
TP	6	23	29	39	42	33	19
FP	2	26	176	170	142	56	28
DC-B	5	30	43	48	48	36	23
P-B	75 %	63.3 %	23.4 %	24.4 %	28.8 %	43.8 %	51.1 %
R-B	8.2 %	49.2 %	70.5 %	78.7 %	78.7 %	59 %	37.7 %
TP-B	6	31	48	51	53	39	24
FP-B	2	18	157	158	131	50	23
mAP	0.016	0.1	0.066	0.124	0.214	0.174	0.116
mAP.50	0.059	0.166	0.137	0.261	0.414	0.321	0.196

**Table 7.13:** Table with result metrics for jagged edge measurements labeled from M1 to M7, which are described at the end of this section.

From Table 7.13 can be seen that slicing to 640-pixel-sized slices helps get more acceptable results. The highest recall for the jagged edge defect is reached when 30 % of the training set are negative samples, but the number of false positives is extremely high. Using all available negative samples helps reduce it, but recall is also reduced. Same as material on edge, bounding boxes have a much larger area than the area of the actual defect. For that reason, using 20 % as the minimal area ratio is much better for the jagged edge defect type.

Explanation of the labels for measurements:

- M1 - baseline model
- M2 - mode using slicing into 1280-pixel-sized slices without overlap and minimal object area ratio is 20 %
- M3 - the same as M2, but with slicing into 640-pixel-sized slices
- M4 - using slicing into 640-pixel-sized slices with overlap and 0 % negative samples
- M5 - using slicing into 640-pixel-sized slices with 30 % negative samples
- M6 - using slicing into 640-pixel-sized slices with all negative samples kept, and minimum object area ratio is 20 %
- M7 - the same as M6, but with 5 % minimum object area ratio

### 7.1.13 Evaluation of results

The following conclusions were made from all the previous measurements and observations. Using slicing is necessary to get acceptable results. Slicing source images into 640-pixel-sized slices reached the best results. It has improved the recall for material remains defects from 2.3 % to 90.5 % in softer metrics. The use of this technique met expectations with improved detection of small defects and not just them. Add overlap to slicing solved some problems with defects on the edges, especially with the larger defects. It resulted in the minimum area ratio should be smaller for small defects and larger for medium and large defects. Specifically, 5 % for remaining material and 20 % for jagged edge and material on edge. Using negative samples increased recall by 2.8 % and barely precision, but the private dataset contains no images with a larger area without defects. This dataset should ideally include images with each product type completely without defects. Additionally, this dataset containing only 113 images with underrepresented jagged edge defect type annotations is not ideal. Another problem is unannotated defects, mostly very small areas which are probably material remains defect type. Further findings are that the size of the pre-train model has minimal effect on results, while using slicing and randomly initialized weights has a slightly better performance.

In conclusion, the YOLO model for material remains reached 79.1 % recall with 68.7 % precision from 2.3 %. The recall was improved for material on edge from 9 % to 60 %, with 47.9 % precision. In the case of the jagged edge, we improved recall from 8.2 % to 59 % with 43.8% precision. The mAP.50 has significantly increased in the case of material remains and from 0.05 to 0.642.

A significant improvement in the model's detection performance was achieved for all defect types. These results are not so poor, but the results still have imperfections, especially in the case of material on the edge and jagged edge for application in manufacturing. This showed me that real data from manufacturing are significantly more challenging.

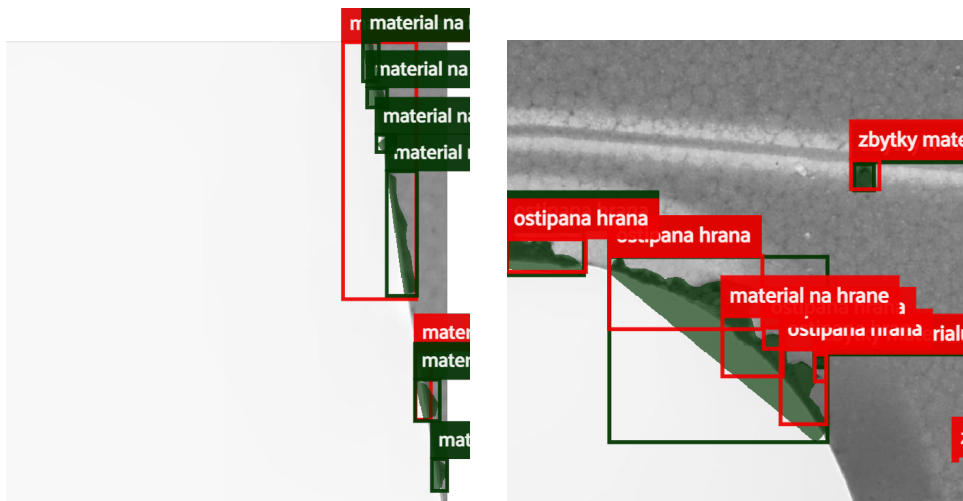


## 7.2 Comparison of methods

This section describes the comparison of the YOLO and U-net models on the private dataset. From the experiments in Section 7.1, we found out which techniques help with detecting each defect type, so they are used in the comparison. The training was run on 640-pixel-sized samples, all negative samples were kept in the training set. The samples were with an overlap of 25 % and 5 % of minimal area ratio. Subsections 7.2.1 and 7.2.2 provide typical predictions of the relevant method. In Subsection 7.2.3 are compared the differences between predictions made by U-net and YOLO.

### 7.2.1 U-net

Predictions performed using U-net mode from the Kaggle notebook on the private dataset are described here. It is focused on usual predictions with their advantages and disadvantages. The training was run for 70 epochs with 200 steps per epoch on 640-pixel-sized slices with overlap, keeping all negative samples and 5 % of the minimal object area.



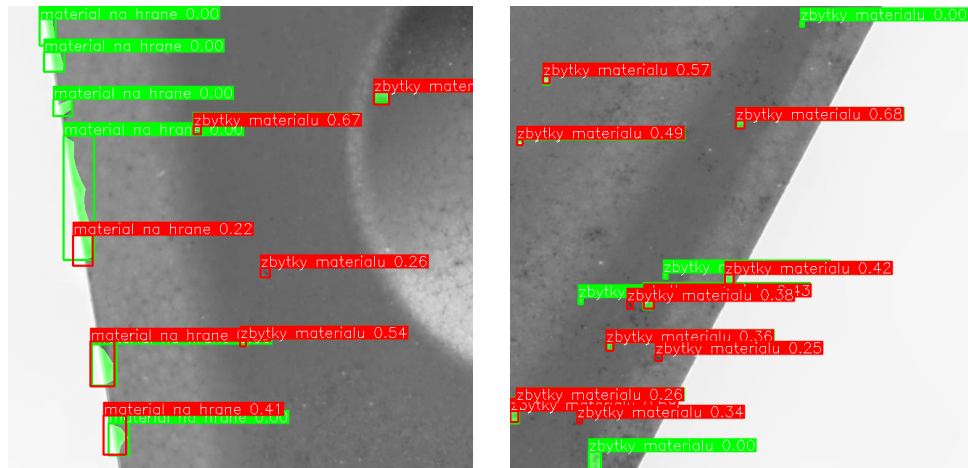
**Figure 7.10:** Example predictions performed by U-net.

Figure 7.10 shows on the left detection of almost all of the ground truth defects for the softer metric. For the stricter metric using IoU, there is only one detected defect. On the right can be seen the detection of a large jagged edge, which is detected in several smaller bounding boxes.

Prediction of medium-sized or large defects is quite accurate. But small defects are not detected so well, which is described in Subsection 7.2.3 with comparison.

### 7.2.2 YOLO

Results of common predictions reached with the YOLOv5 model trained on the private dataset are provided in this subsection. Example predictions demonstrate here the most common problems. The training was run on the same sliced images as U-net for 500 epochs with patience 100 from randomly initialized weights using yolov5m.yaml config file.



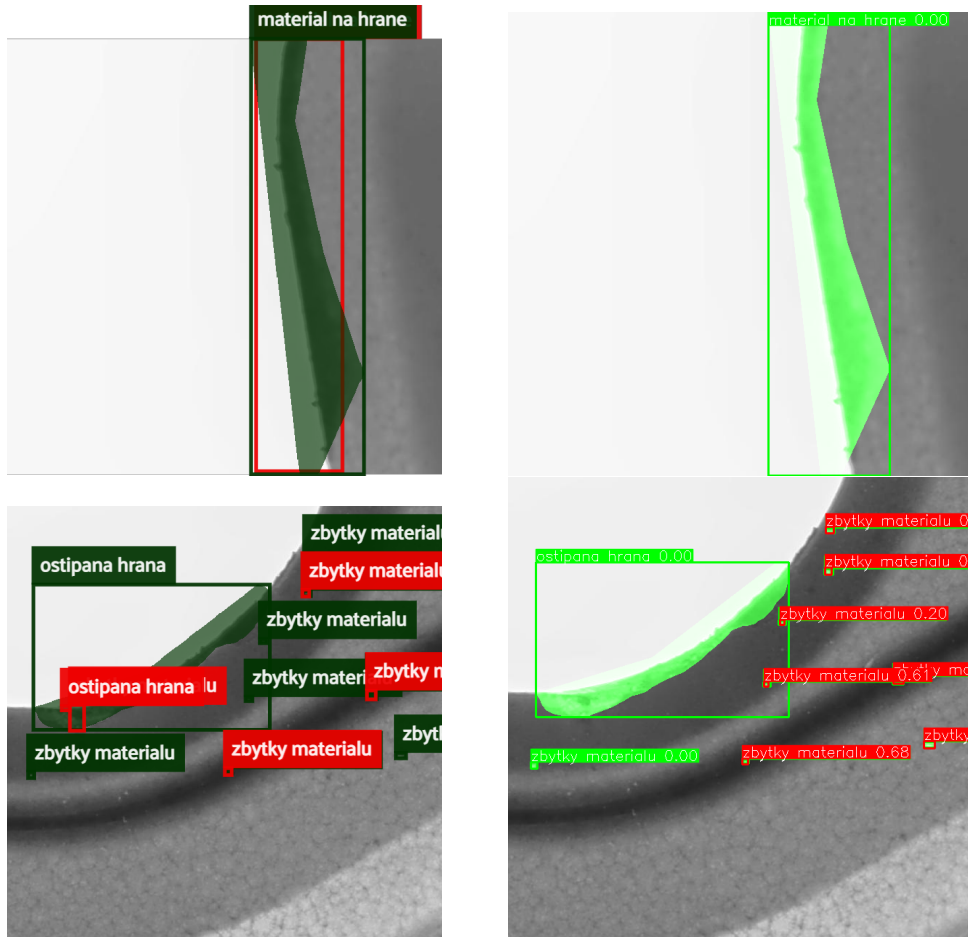
**Figure 7.11:** Example predictions performed by YOLOv5.

The first example in Figure 7.11 shows that the prediction of the jagged edge is quite accurate or not predicted at all. It is also possible to see here the prediction of the unannotated defect of the remaining material, which makes up a few percent of the false positives. The second one shows this situation again. Additionally, it is visible that most of the small defects are detected.

In summary, medium-sized or large defects are detected accurately but only if recognized at all, mainly related to jagged edge and material on edge. Small annotations of material remains are detected acceptably well.

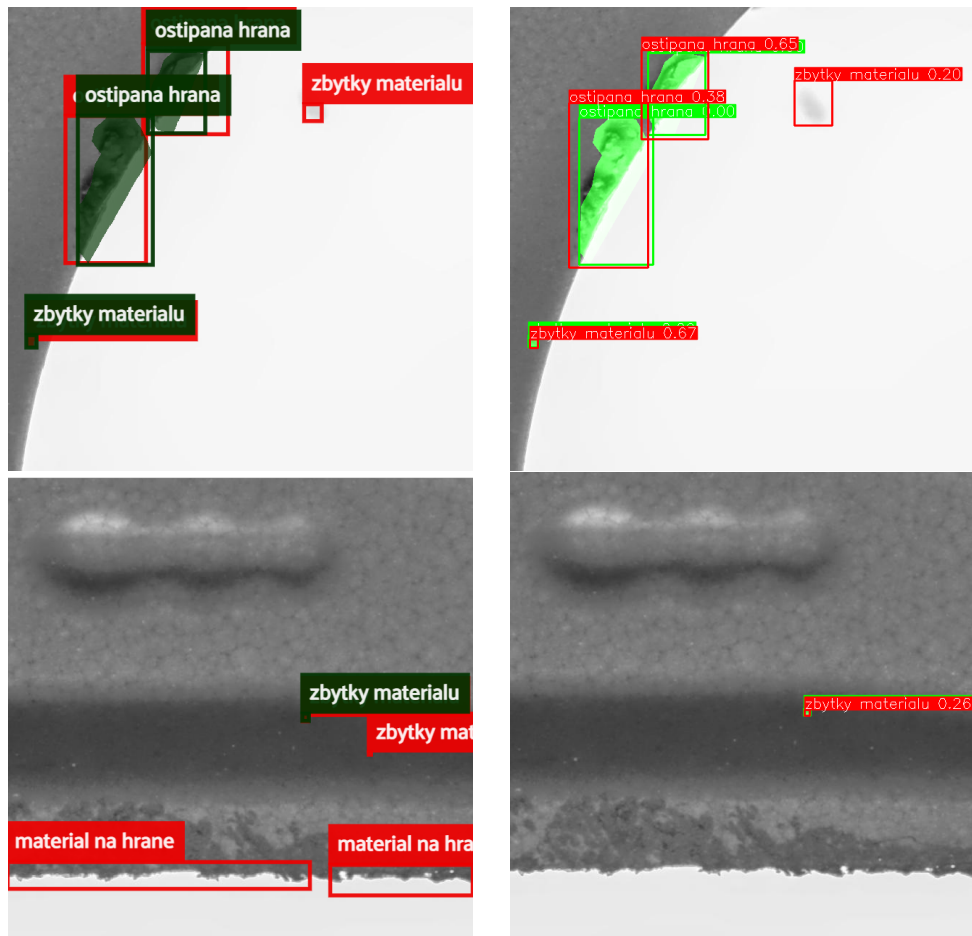
### 7.2.3 Summary comparison

Here are provided the significant differences between the results of YOLO and U-net. Firstly compare differences in performed predictions followed by comparing the resulting metrics to see if they are consistent with observations.



**Figure 7.12:** Comparison of predictions performed by U-net on the left side and YOLO on the right side.

The first row in Figure 7.12 shows a better detection ability for large defects in the case of U-net. The example in the following row shows again not detecting a large jagged edge defect. In the case of YOLO predictions, close to all material remains are detected. U-net detected barely half of them.



**Figure 7.13:** Comparison of predictions performed by U-net on the left side and YOLO on the right side.

In Figure 7.13, the first row demonstrates very good predictions whose bounding boxes fit the ground truth more accurately in the case of YOLO. Both methods detected dirt on the camera as the remaining material defect. The last example shows the damaged edge of the product, which is not annotated as a defect. U-net predicted it as a defect which, according to my opinion correct, but according to the ground truth not. Also, there is only one ground truth remaining material annotation, but more very small white areas. we believe these areas should be annotated as remaining material. U-net recognized one of them but YOLO has results corresponding better with ground truth.

Metric	DC	P	R	TP	FP
U-net	1 294	27 %	33.1 %	1 294	3 491
YOLO	2 685	67.4 %	68.7 %	2 685	1 301

Metric	DC-B	P-B	R-B	TP-B	FP-B
U-net	2 428	54.4 %	62.1 %	2 604	2 181
YOLO	2 824	70.5 %	72.2 %	2 812	1 174

Metric	mAP	mAP.50	mAP.75	$mAP50_s$	$mAP50_m$	$mAP50_l$
U-net	0.013	0.051	0.002	0.046	0.049	0.076
YOLO	0.215	0.441	0.188	0.295	0.434	0.404

**Table 7.14:** The first two tables are the number of detected defects, TP, and FP for the YOLO and U-net models in the first table. The mAP results are described in the last table.

The tables above are consistent with the observations made. YOLO has detected much more defects with significantly fewer false positives. Both methods must deal with unannotated defects, U-net these defects predicted, but even if they are included, YOLO still has significantly better precision and recall. Because U-net performs mostly multiple smaller predictions for one defect, that results in lower mAP.



## Chapter 8

### Conclusion

The thesis aimed to analyze the types of manufacturing defects and find suitable methods to detect them. After researching the types of manufacturing defects from both datasets, we decided to use YOLO and U-net from the range of computer vision methods. In order to use these methods, it was necessary to convert the given datasets into the required format.

First, a comparison of YOLO and U-net was performed. This comparison on the Severstal dataset shows that YOLO provides 1.9 % more detected defects with 81.7 % precision against U-net with 20.1 % precision. For the DNAI dataset, the comparison shows that YOLO achieves much better results according to ground truth. YOLO reached 72.2 % recall, which is 10.1 % more than U-net. But the more significant difference is that YOLO provided 70.5 % precision against the 54.4 % precision of U-net while using the benevolent metric.

To improve the results of both methods, we fine-tuned the following parameters. It is slicing the source images into slices of different sizes with or without overlap, a minimum area ratio of the defect annotation in the slice to keep it, including the negative samples in the training set, and training from the randomly initialized weights or from a pre-trained model.

The most suitable for the Severstal dataset is slicing source images into 800-pixel-width slices, keeping annotations with at least 5 % area in the slice, keeping all negative samples in the training set, and starting training weights from the pre-trained model. Using these techniques, the YOLO model improves by 76.5 % in the mAP, from 0.098 to 0.173. The recall reaches 90 % with 69.3 % precision for the benevolent metric.

We then attempted to identify techniques that would improve the performance of the DNAI dataset. We evaluated performance on three defect types: material remains, material on the edge, and jagged edge. For the material remains, defect type proved to be the most suitable for slicing into 640-pixel-sized slices with an overlap of 25 % combined with 5 % minimum object area in the slice, keeping all available negative slices and starting training from the randomly initialized weights and using the yolov5m config file. This helped to achieve 90.5 % recall with 50 % precision. For the material on the edge, the same techniques helped with the difference of 20 % minimum object area in a slice instead of 5 %. So the model for material on edge reached recall

67 % with a precision of 41.5 %. In the case of the jagged edge, the most suitable techniques are the same as for material on edge. These techniques improved jagged edge recall to 59 % with 43.8 % precision. Slicing of source images is the most significant improvement for all defect types.

In conclusion, we showed that YOLO outperforms the U-net on both datasets. With improvements, the YOLO reached 72.2 % recall and 70.5 % precision on the whole DNAI dataset. On the Severstal dataset it achieved up to 90 % recall with 50 % precision according to benevolent metric. We found that real data from manufacturing are significantly more challenging.



## Appendix A

### Bibliography

- [1] Fatih Cagatay Akyon, Cemil Cengiz, Sinan Onur Altinuc, Devrim Cavusoglu, Kadir Sahin, and Ogulcan Eryuksel. SAHI: A lightweight vision library for performing large scale object detection and instance segmentation, November 2021.
- [2] Fatih Cagatay Akyon and Burak Maden. Cli commands from <https://github.com/obss/sahi/blob/main/docs/cli.md#predict-command-usage>.
- [3] Alexey Grishin, BorisV, iBardintsev, inversion, Oleg. Severstal: Steel defect detection from <https://kaggle.com/competitions/severstal-steel-defect-detection>, 2019.
- [4] K. Amano. Solution on kaggle competition from <https://www.kaggle.com/code/amanooo/defect-detection-starter-u-net>.
- [5] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. YOLOv4: Optimal speed and accuracy of object detection, 2020.
- [6] COCO Consortium. Data format from <https://cocodataset.org/#format-data>.
- [7] Vijay Dubey. Evaluation metrics for object detection algorithms from <https://medium.com/@vijayshankerdubey550/evaluation-metrics-for-object-detection-algorithms-b0d6489879f3>, 2020.
- [8] Dwyer, B., Nelson, J.(2022), Solawetz, J., et. al. Roboflow (version 1.0) [software] available from <https://roboflow.com>. computer vision.
- [9] Chris Eijgenstein. Convert segmentation rgb mask images to coco json format from <https://github.com/chrise96/image-to-coco-json-converter>.
- [10] Raducu Gavrilescu, Cristian Zet, Cristian Foşalău, Marcin Skoczylas, and David Cotovanu. Faster r-cnn:an approach to real-time object detection. In *2018 International Conference and Exposition on Electrical And Power Engineering (EPE)*, pages 0165–0168, 2018.

- [11] Eric Hofesmann. Iou a better detection evaluation metric from <https://towardsdatascience.com/iou-a-better-detection-evaluation-metric-45a511185be1>, 2020.
- [12] Jonathan Hui. Ssd object detection: Single shot multibox detector for real-time processing, 2018.
- [13] Computer vision techniques from <https://www.javatpoint.com/computer-vision-techniques>.
- [14] Glenn Jocher. YOLOv5 git repository from <https://github.com/ultralytics/yolov5>, 2022.
- [15] Glenn Jocher, Ayush Chaurasia, Alex Stoken, Jirka Borovec, NanoCode012, Yonghye Kwon, TaoXie, Kalen Michael, Jiacong Fang, imyhxy, Lorna, Colin Wong, Zeng Yifu, Abhiram V, Diego Montes, Zhiqiang Wang, Cristi Fati, Jebastin Nadar, Laughing, UnglvKitDe, tkianai, yxNONG, Piotr Skalski, Adam Hogan, Max Strobel, Mrinal Jain, Lorenzo Mammana, and xylieong. ultralytics/yolov5: v6.2 - YOLOv5 Classification Models, Apple M1, Reproducibility, ClearML and Deci.ai integrations, August 2022.
- [16] Gilberto Titericz Junior. Building and visualizing masks from <https://www.kaggle.com/code/titericz/building-and-visualizing-masks/notebook>, 2019.
- [17] Ph.D. Jędrzej Świeżewski. Yolo algorithm and yolo object detection from <https://appsilon.com/object-detection-yolo-algorithm/>, 2020.
- [18] Jeong-ah Kim, Ju-Yeong Sung, and Se-ho Park. Comparison of faster-rcnn, yolo, and ssd for real-time vehicle type recognition. In *2020 IEEE International Conference on Consumer Electronics - Asia (ICCE-Asia)*, pages 1–4, 2020.
- [19] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: Single shot MultiBox detector. In *Computer Vision – ECCV 2016*, pages 21–37. Springer International Publishing, 2016.
- [20] Computational resources were supplied by the project "e-infrastruktura cz" (e-infra cz lm2018140 ) supported by the ministry of education, youth and sports of the czech republic.
- [21] B. E. Moore and J. J. Corso. Fiftyone. *GitHub. Note:* <https://github.com/voxel51/fiftyone>, 2020.
- [22] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2015.

- [23] Gai Rongli, Chen Na, and Yuan Hai. A detection algorithm for cherry fruits based on the improved yolo-v4 model. *Neural Computing and Applications*, 7, 2021.
- [24] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.
- [25] Aditya Sharma. Mean average precision (map) using the coco evaluator from <https://pyimagesearch.com/2022/05/02/mean-average-precision-map-using-the-coco-evaluator/>, 2022.
- [26] Jacob Solawetz. What is yolov5? from <https://blog.roboflow.com/yolov5-improvements-and-evaluation/#what-is-yolov5>, 2020.
- [27] Ultralytics. YOLOv8 from <https://github.com/ultralytics/ultralytics>.
- [28] Chia-Chin Wang, Hooman Samani, and Chan-Yun Yang. Object detection with deep learning for underwater environment. In *2019 4th International Conference on Information Technology Research (ICITR)*, pages 1–6, 2019.
- [29] Jeremy Zhang. Unet — line by line explanation from <https://towardsdatascience.com/unet-line-by-line-explanation-9b191c76baf5>, 2019.
- [30] Longfei Zhou, Lin Zhang, and Nicholas Konz. Computer vision techniques in manufacturing. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 53(1):105–117, 2023.



## Appendix B

### Abbreviations

- **AP** - Average Precision
- **COCO** - Common Objects in Context
- **CSV** - Comma-separated values
- **DC** - Detected Count
- **DC-B** - Detected Count for benevolent metric
- **FP** - False Positive
- **FP-B** - False Positive for benevolent metric
- **IoU** - Intersection over Union
- **JSON** - JavaScript Object Notation
- **mAP** - Mean Average Precision
- **P** - Precision
- **P-B** - Precision for benevolent metric
- **R** - Recall
- **R-B** - Recall for benevolent metric
- **RLE** - Run Length Encoding
- **TP** - True Positive
- **TP-B** - True Positive for benevolent metric
- **YOLO** - You Only Look Once



## Appendix C

### User guide

#### C.1 Code

In this work, some helper functions are needed for data preparation or evaluation of results. Some git repositories were used with my edits. All of that can be found in the git repository <https://gitlab.fel.cvut.cz/lososjos/bachelorthesis>. The project is using Python 3.9.6 and Torch 1.12.1+cu116. The git structure is described here:



The code for the project can be divided into three parts. The first one provides Jupyter Notebooks with code for converting, training, viewing and evaluating datasets in the main folder. The second part is used git repositories, 'cocosplit', for splitting datasets into a train, valid, and test sets. The 'image-to-coco-json-converter' was used for converting masks into COCO annotations. The last part contains scripts for data analysis, conversions, evaluation of results according to IoU or benevolent metric, and an example script to run training of YOLO on MetaCentrum.

## ■ C.2 Datasets

### ■ C.2.1 Severstal

The Severstal dataset was used from the Kaggle competition <https://www.kaggle.com/c/severstal-steel-defect-detection>. The dataset is described in Section 3.2. Data are available at <https://www.kaggle.com/competitions/severstal-steel-defect-detection/data>. The dataset in COCO format is provided at <https://www.kaggle.com/datasets/joseflosos/severstal-coco>.

### ■ C.2.2 DNAI

DNAI dataset is a dataset from a client of the DNAI, s.r.o. firm. The description of the dataset is provided in Section 3.1. Data is available upon request.