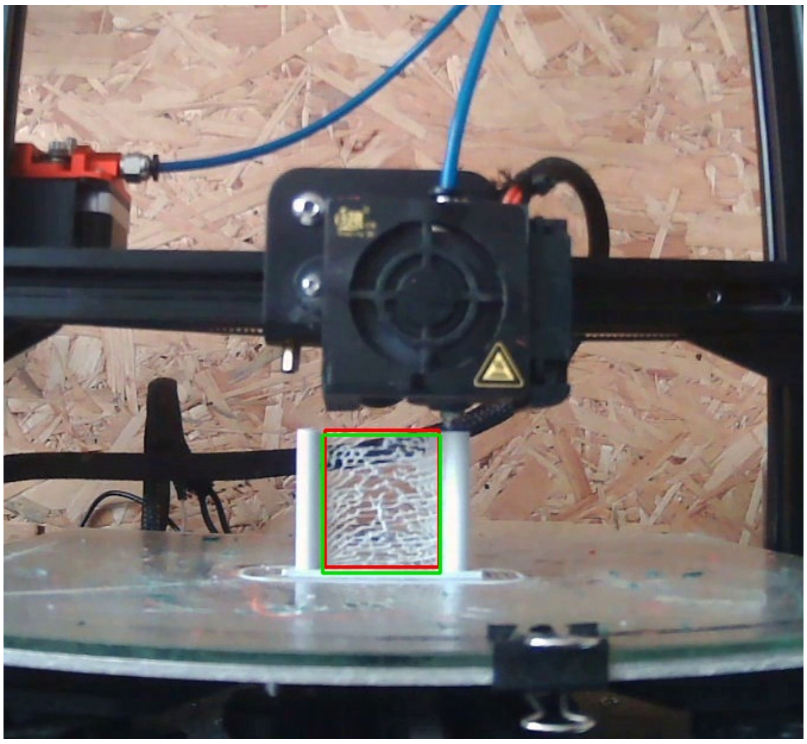
Automatic detection and severity assessment of stringing defect in FDM 3D printing using deep learning

Master's Degree Thesis

Report Damian Herman



Aalborg University
Department of Electronic Systems

AALBORG UNIVERSITY
STUDENT REPORT

Department of Electronic Systems
Aalborg University
http://www.aau.dk

Title:
Automatic detection and severity assessment of stringing defect in FDM 3D printing using deep learning

Theme:
Scientific Theme

Project Period:
Spring Semester 2024

Project Group:
Group 941

Participant(s):
Damian Dariusz Herman

Supervisor(s):
Andreas Aakerberg

Copies: 1

Page Numbers: 73

Date of Completion:
May 31, 2024

Abstract:

The thesis investigates defects occurring during the Additive Manufacturing process and proposes a CNN-based solution to detect and assess the severity of the stringing defect. A YOLOv8 and MobileNetV3 solution to first detect and then assess stringing defects' severity have been proposed to solve the problem. A labeled object detection dataset with severity labels have been created. Tests on the dataset and as well on a real-life capture from a webcam overseeing the printing process have been conducted, to see whether the proposed solution is viable to solve the problem of both manufacturing supervision and defect severity assessment. To find out if the solution is viable, several metrics have been calculated to measure, how well the solution performs, as well resource usage have been measured, to find out if the solution won't interrupt the manufacturing process. On the test split of created dataset, the detection model performs well, however it fails to generalize on the real-life capture and the severity assessment model is not accurate enough.

Contents

Chapter 1

Introduction

3D printing, or Additive Manufacturing steadily gained popularity at homes, allowing makers to rapidly prototype and create their own products. According to Google Trends, the adoption by the home users keep increasing, leading to more and more manufacturers entering home 3d printing market.

The 3D printing process itself requires supervision, to ensure that the operated machine is calibrated and no printing failures occur. Depending on the object size and the process parameters, the manufacturing process can take from a couple of minutes to a couple of hours and days, creating a need to oversee the print process, as some of the 3d printing errors may lead to a complete failure, resulting in lost time, energy and resources or even worse, a potential fire hazard.

The calibration process itself may be cumbersome, requiring the operator to dial in material parameters, which does not necessarily guarantee, that a resulting printout will be flaw-free; visual defects may still occur, which depending printed object's purpose may introduce an additional post-processing step and if the flaw is severe enough, the object will have to be re-printed, leading to material waste.

A potential solution would be to use Computer Vision, to solve the issue of automatic print supervision, flaw detection and flaw severity assessment, potentially relieving the user from the need of constant manufacturing process supervision and simplifying the calibration process, as some of the printer settings could be adjusted on-the-fly.

Commercial solution exist to solve the supervision problem, but those solutions lack the ability to assess whether a flaw is severe enough to justify re-printing the manufactured product.

This master's thesis aim to find out what defects are occurring in 3d printing, what have been done to detect those flaws and extend upon the flaw detection task, by introducing automatic assessment of the detected flaw.

1.1 Initial problem statement

The following project statement have been formed, based on the flaw detection and supervision problem:
What are the 3d printing parameters, 3d printing defects and how can these defects could be detected and assessed?

Chapter 2

Problem Analysis

This chapter aims to analyse what is additive manufacturing, what defects occur during the manufacturing process, establish what are the current state of defect detection in the context of additive manufacturing. Furthermore, defect severity have been investigated, in order to find out why it is important, not only detect the defects but also assess their severity.

2.1 Additive manufacturing

Additive manufacturing, or better known as 3D printing is the process of creating physical objects from a geometrical representation by successive addition of material. 3D printing has its application in broad industry sectors, ranging from medical applications, jewellery, construction, ending on rapid prototyping and design [1].

In additive manufacturing a broad range of materials are being used. Many thermoplastics, ceramics, resins are being used in 3d printing [1]. With open source projects like RepRap and companies creating affordable 3d printers, printing have been adopted by average consumers, enabling them to experiment with 3d printing and manufacture items at home [2] [3].

At homes, two types of 3D printing are typically used - Fused Deposition Modeling (FDM) or Masked Sterelitography (M-SLA) in cojuntion with so-called slicer, which divides and prepares a 3d model into layers and generates so-called Gcode.

2.1.1 Fused deposition modeling

Fused deposition modeling is a 3d printing process that uses a continuous *filament* of a thermoplastic material and deposits it on a (usually) heated *printing bed* using an *extruder* [1] [4]. The thermoplastic is heated up to a temperature that allows the plastic to be in semi-liquid state, enabling extrusion through the *nozzle*. A printing head moves around the printing area and lays the material on a layer-by-layer basis.

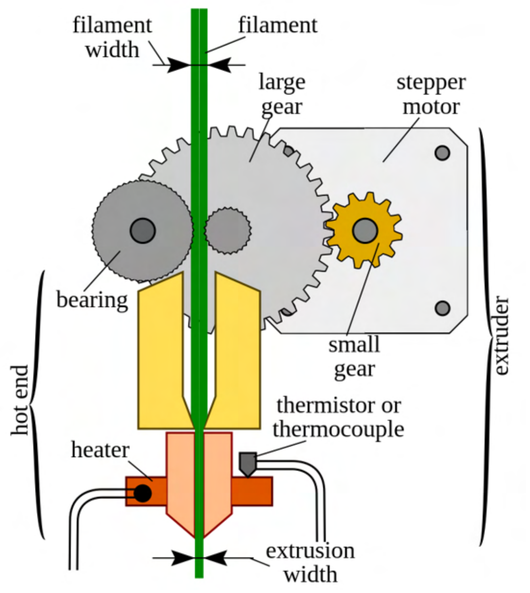
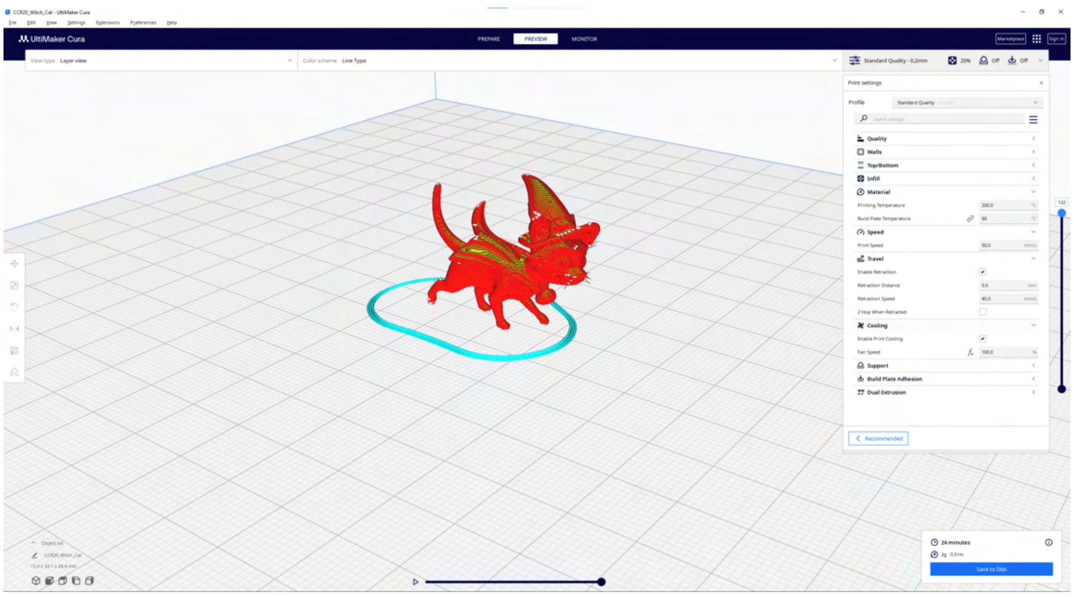


Figure 2.1: Printing head schematic [4]

Figure 2.1 represents an example of a 3D printing head. The extruder pushes filament to the hot-end, to be finally laid on the printing bed.

A wide range of filaments are offered to consumers such as acrylonitrile butadiene styrene (ABS), polylactic acid (PLA), polyethylene terephthalate glycol (PETG), polyethylene terephthalate (PET), high-impact polystyrene (HIPS), thermoplastic polyurethane (TPU) and aliphatic polyamides (Nylon). Physical properties of the print depend on what material is being used. For example - PLA plastic is easier to print with, but is less durable than ABS or PET/PETG and is susceptible to heat [5].

The 3D model has to be processed in a slicer software. A 3D model is being imported to the software and after adjusting parameters specific to the printing hardware and to the used filament, a machine code called GCode is being generated. GCode contains all instructions directed to the printer like axis movements, material extrusion or temperature settings of the printing bed (if heated) and the hotend [4].



View type Layer view
Color scheme Line Type
Standard Quality - 0.2mm 20% Off Off
Profile Standard Quality
Printing Temperature 200.0 °C
Build Plate Temperature 50 °C
Print Speed 50.0 mm/s
Enable Retraction ✓
Retraction Distance 5.0 mm
Retraction Speed 45.0 mm/s
Z Hop When Retracted
Enable Print Cooling ✓
Fan Speed 100.0 %
122
15.9 x 33.1 x 24.4 mm
24 minutes
2g · 0.51m

Figure 2.2: A 3D model being sliced in Ultimaker Cura, in order to process and prepare the model for a production job.

Figure 2.2 showcases a 3D model being sliced and prepared for 3D printing in Ultimaker Cura software. It can be seen how the model is divided into layers and infill. Infill is needed for structural integrity of the model and its fill parameter can be adjusted depending on specific needs.

2.1.2 Mask stereolitography

Mask stereolitography (MSLA) is a 3d printing process that uses a UV resin instead of thermoplastics. Instead of a printing head and movable axes, a lifting platform, a resin container and UV light source is needed. Each layer is being converted into a 2D mask image and is projected onto the resin to cure it [6].

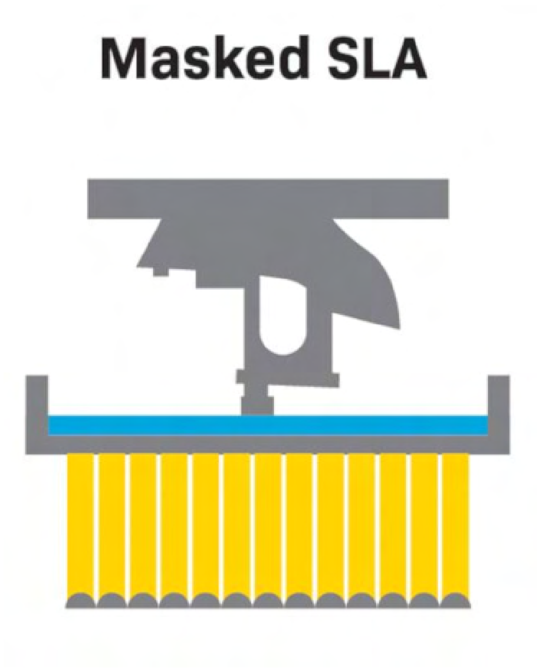
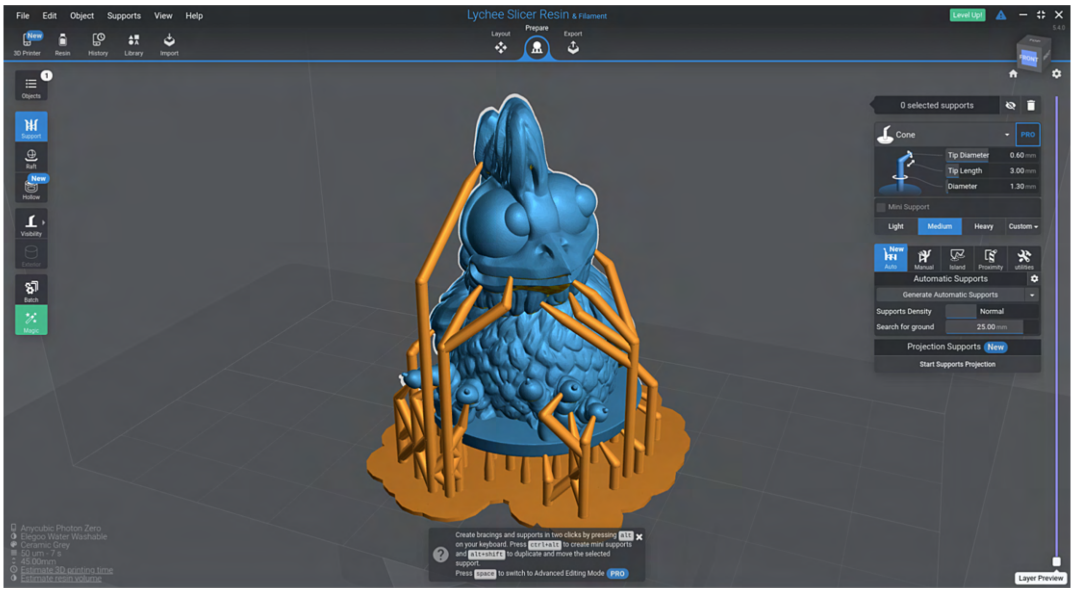


Figure 2.3: MSLA process visualisation [7].

On figure 2.3 the light is being shined onto the resin vat from the bottom. In MSLA an LCD screen is being used to mask areas, where the UV light will come through to the vat and solidify the resin. The lifting platform "peels" the layer from the vat and the process is being repeated until the print is finished [6].

Similarly to the FDM process, slicer software is being used to prepare and slice the print, however the print parameters are different.



0 selected supports

Cone PRO

Tip Diameter 0.60 mm
Tip Length 3.00 mm
Diameter 1.30 mm

Mini Support

Light [✓ Medium] Heavy Custom

[✓ Auto] Manual Island Proximity utilities

Automatic Supports

Generate Automatic Supports

Supports Density Normal
Search for ground 25.00 mm

Projection Supports New

Start Supports Projection

Anycubic Photon Zero
Elegoo Water Washable
Ceramic Grey
50 um - 7 s
45.00mm
Estimate 3D printing time
Estimate resin volume

Create bracings and supports in two clicks by pressing alt on your keyboard. Press ctrl+alt to create mini supports and alt+shift to duplicate and move the selected support.
Press space to switch to Advanced Editing Mode PRO

Figure 2.4: A 3d model being prepared for 3d printing in Lychee Slicer

Figure 2.4 shows a model being prepared for a resin print, with visible support material.

2.2 Slicing process

In order to print an object on a 3D printer, so-called slicing process has to be performed via piece of software called a slicer. A slicer has two jobs to perform: divide the objects into layers, prepare object to be printed and generate GCode, which controls 3d printer movements and the extrusion process. Several software distributions like Slic3r, Ultimaker Cura, Lichee Slicer help with slicing and GCode generation, offering printer parameter adjustment and toolpath previews.

2.2.1 FDM 3D printing parameters

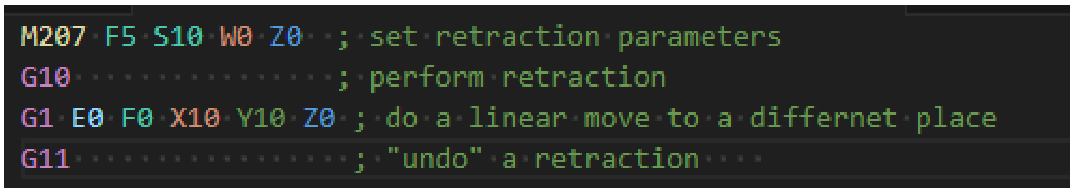
Some printing parameters are crucial for a print to be successful and limit amount of defects occurring in the resulting print. Based on [8] [9] [10], some of the parameters include:

1. **Hotend temperature** - temperature at which the printer will print. Too low temperature may lead to layer adhesion issues and extrusion issues This parameter is set depending on what filament is being used, it varies between filament brands, thermoplastic types and even filament colors.

2. **Print bed temperature** - temperature to which a heated print bed will be heated. Correct temperature will improve first layer adhesion. Incorrect print bed temperature may lead to print defects like warping.
3. **Print speed** - controls how fast will 3D printer move its axes while printing. Print speeds have to be calibrated per machine and per filament basis. Printing too fast will lead to quality issues like ringing.
4. **Retraction, Retraction distance, Retraction Speed** - used to combat stringing. Retraction distance controls "how much" will the printer pull the filament back. Retraction speed controls the rate at which the filament will be pulled back.
5. **Layer height** - controls the height of one layer. In general, the smaller layer height, the finer details can be printed, at the expense of longer print times. However, the layer height has to match nozzle diameter. It is recommended that the layer height should not exceed 80% of the nozzle diameter , otherwise it may lead to issues with inter-layer adhesion.
6. **Infill type and infill density** - infill fills the hollow space in the print and can be adjusted to modify prints' structural strength, or even flexibility if printing with elastic materials like TPU. Higher percentage mean that the print will be less "empty" inside, resulting in a stronger and heavier object.

2.2.1.1 GCode

GCode is a programming language used to control the 3D printer itself, generated by the slicer software, consisting of commands that control axes movement, speeds and at which temperatures 3d printer should operate [11].

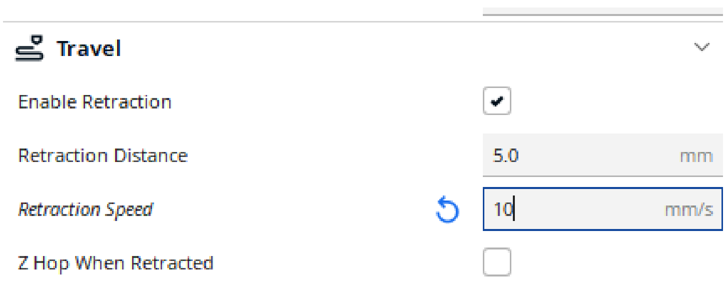


```
M207 F5 S10 W0 Z0  ; set retraction parameters
G10                ; perform retraction
G1 E0 F0 X10 Y10 Z0 ; do a linear move to a differnet place
G11                 ; "undo" a retraction
```

Figure 2.5: Code listing presenting an example retraction.

Listing 2.5 sets necessary parameters for a retraction action (M207 command), performs the retraction (G10 command), performs a linear move (G1 command) and "unretracts" filament back to nozzle [11]. (G11 command) GCode syntax usually consist of the code itself and the code parameters, separated by spaces [11].

Parameters in Gcode are configured via slicer software, for example in previously mentioned Ultimaker Cura.



Travel

Enable Retraction	✓
Retraction Distance	5.0 mm
Retraction Speed	10 mm/s
Z Hop When Retracted	

Figure 2.6: Retraction settings set in slicer directly correspond to the Gcode M207 command.

Because GCode is a text format, it is possible to change the parameters after slicing or even during printing, as commands are being sent one-by-one, enabling the possibility to either inject GCode or modify commands on-the-fly, based on some kind of other input. Such a property may be exploited for automatic print calibration, as users have to calibrate their printers manually before printing using a specially designed object to test printer capabilities like dimensional accuracy, extruder calibration, first layer adhesion, nozzle heater PID controller or retraction settings [12].

2.3 Calibration

Calibrating the 3d printer minimizes the chance of a defect occurring and increases printout quality. Several components can be calibrated in a 3d printer, requiring manual intervention from the user. The printer's bed has to be leveled, ensuring good print adhesion to the platform, so-called E-steps have to be correctly set to make sure, that the extruder is serves the correct amount of filament to the nozzle [13]. Thermoplastic filament settings like its printing temperature (specified by the filament producer) and build platform temperature have to be set correctly in the slicer software.
In order to calibrate the 3d printer several specially designed object have been created. Which model is ought to be used, depends on the 3d printer's component being calibrated.

2.4 Selected FDM 3D printing defects

This section will present some of FDM 3D printing faults and why they occur.

2.4.1 Warping

Warping is a defect where a part of the print loses adhesion to the printing bed, curling upwards. If a part of print gets abruptly cooled, the plastic will shrink and warping may occur. The susceptibility to this issue depends greatly on the material used and the printing environment. For example ABS is much more susceptible to this issue, as it shrinks more than PLA [14], if abruptly cooled. The issue can be mitigated by implementing a 3D printing enclosure over the printer, to stabilize the air temperature around the print [8].

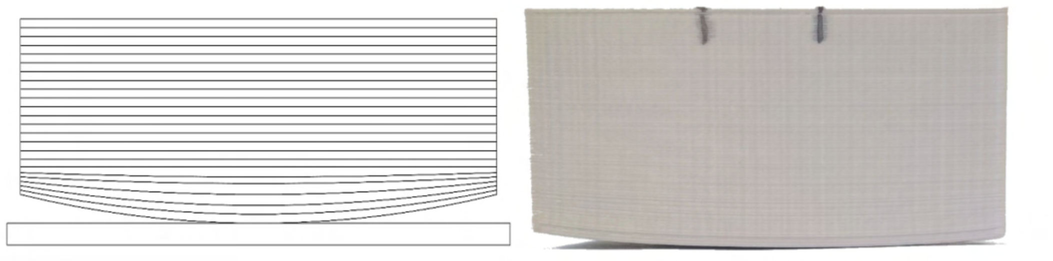


Figure 2.7: A picture demonstrating warping defect. [14]

Figure 2.7 demonstrates how the corners of print lift up from the printing bed, due to abrupt cooling of the printed object [8].

2.4.2 Stringing

Stringing is a defect, in which a small strands of plastic are left on the printed model. Typically the issue occurs, when plastic oozes out of the nozzle (as the material is in semi-liquid state) while the printing head moves from one place to a new location.

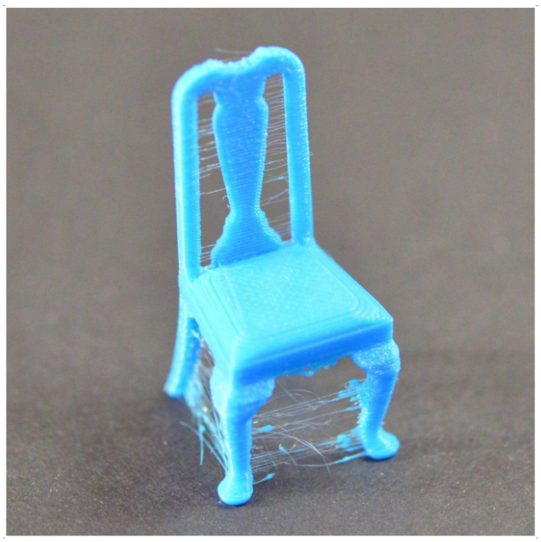


Figure 2.8: Image showcasing stringing defect [8].

Figure 2.8 presents how string looks alike on a print. To prevent stringing, retraction setting in the slicer has to be enabled [8]. However, the retraction setting has to be tuned depending on what filament type is used and how the filament was stored. Plastic filaments are hygroscopic, absorbing surrounding moisture [15].
Furthermore, a humid filament may lose its properties [16] and may be prone to stringing, due to water absorbed by the filament boiling in the nozzle [17] [18], especially if PET-G or ASA filaments are being used [19].
Print speed has influence on stringing. Printing too fast may lead to stringing, as the plastic may have not enough time to solidify and drag along the nozzle, as it moves [8] [20].

2.4.3 Layer Separation and Splitting

Layer splitting occurs when inter-layer lamination is lost. Similarly to warping, the issue may occur when a part of print is being cooled, causing plastic to shrink [21] and when invalid print settings like nozzle diameter/layer height are configured [8]. Resulting print has "cracks" between layers, weakening printed objects' strength and is a visual defect.

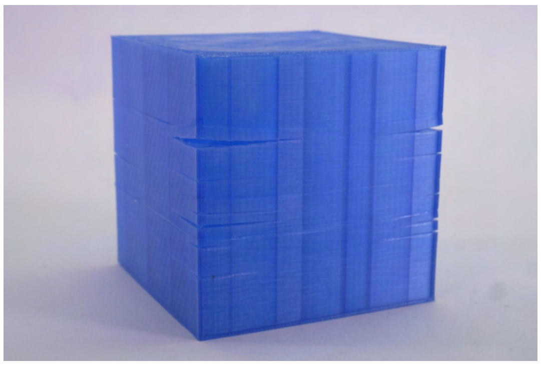


Figure 2.9: Picture showing how layer separation influences a print [21]

To avoid the issue presented in 2.9, a heated enclosure and a heated printing bed is recommended [8] [21]. Making sure that the layer height and printing nozzle diameter is set correctly, as mentioned in section 2.2.1.

2.4.4 Spaghetti

Spaghetti happens when the printer starts to print "in air" and the extruded plastic ends up curling up on the nozzle, instead of being extruded onto the printing bed and the produced item. In result, if print is left for long enough a "spaghetti" of extruded filament is being produced, usually meaning that the print has to be interrupted and started from the beginning. The issue may occur because the print may detach from the printing bed or there is an error in the sliced object [22].

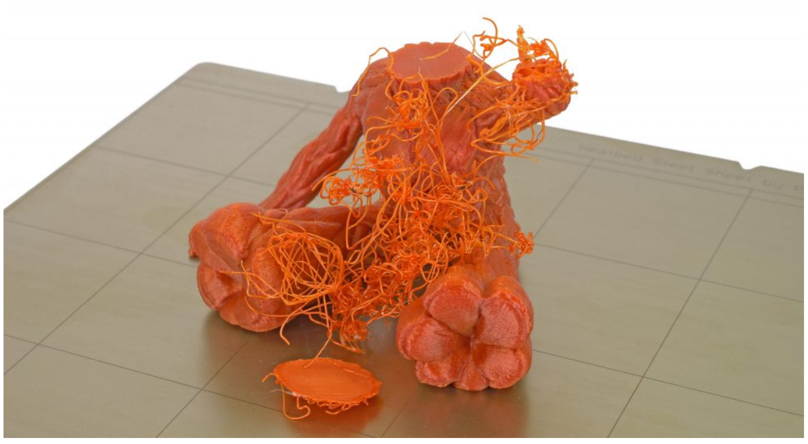


Figure 2.10: A printed object that has spaghetti issue present [22].

2.4.5 Blobs and zits

Blobs and zits are marks and ridges on the outer walls of the print. This issue may be caused by too frequent, too small or too slow retractions in the print (causing the plastic to ooze from the nozzle), inconsistent extrusion caused by worn-out bowden tube (if the 3d printer has a bowden extruder, instead of direct-drive one), too high hot-end temperature and improper cooling, leading the melted material to "slide off", forming a zit when next layer is being laid.

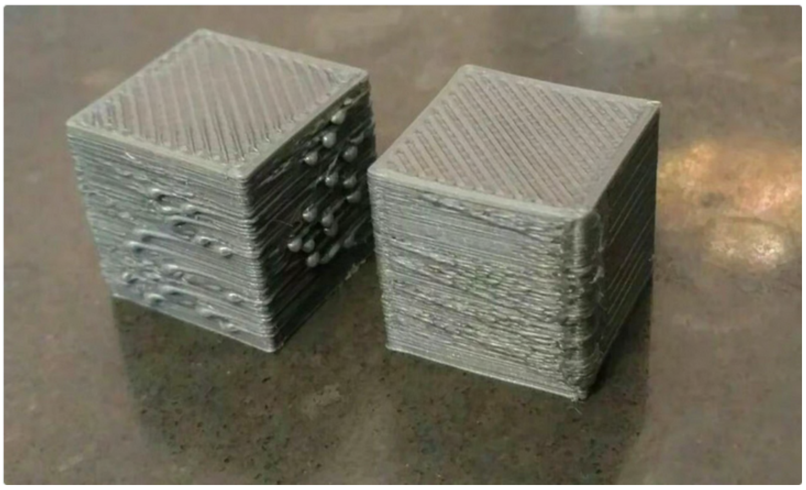


Figure 2.11: Printed objects having blobs and zits on the walls [5]

2.5 Defect severity

3D printing is a process that may lead to defects, as described in 2.4 due to insufficient machine maintenance, incorrect filament storage or other machine and printing environment issues.

Depending on what has fault occurred, a print may need an additional post-processing step to remove excess plastic, like in the case of stringing defect or may needed to be reprinted if the surface of the print cracked, which is the case in inter-layer lamination loss. All of this leads to time, material and energy losses, as the print has to be manually processed or reprinted, for the print to fulfil its use.

Material loss costs rise more, as the production is being scaled and specialized filaments are being used to print. For example, a solid-color PLA filament can be purchased for as low as 20 EUR per 1kg filament spool and filaments with metal dust embedded in the filament may cost from 95 EUR per 1kg spool, meaning that a job restart or even excess material extrusion will lead to increased costs of printing and in a commercial setting lead to income loss.

Energy costs have to be taken into considerations as well. A 3d printer draws considerable amounts of powering the hotend and printing bed, if heated. For instance, Creality Ender 3 V3 SE FDM printer is rated for 350W [23] and the assumed energy price is 0.43 EUR per kWh (3.33 kr) [24], then one hour of printing will cost 0.15 EUR, excluding initial nozzle and printing bed heating process. A print that will take 4 hours will cost 0.60 EUR in just energy costs. Having multiple printers will increase the costs even further.

The presented costs necessitate print supervision, but also automated early print job cancellation, via defect severity assessment, to at least limit energy and material losses. However, a defect may not be severe enough, to justify job cancellation, since a defect may not influence the produced objects' functionality enough.

To further investigate 3D printing defects' severity, several test prints and defect images have been collected, differentiate between defect severity levels.

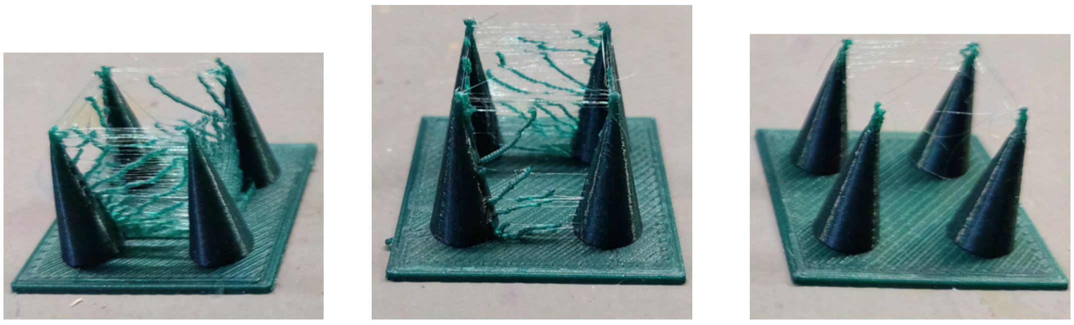


Figure 2.12: Different severity levels of said stringing defect, produced by changing retraction settings, printed by the author.

Figure 2.12 demonstrate different levels of stringing defect, starting from left: "high", "medium", "low". Despite the fact that retraction was configured, some stringing has prevailed. In all cases some material has to be trimmed off. The defect itself does not influence model's functionality, regardless of severity.

Warping described in 2.4 may lead to print failure, meaning that the print will have to be reprinted with changed print parameters, to ensure that the problem will not rise up again.

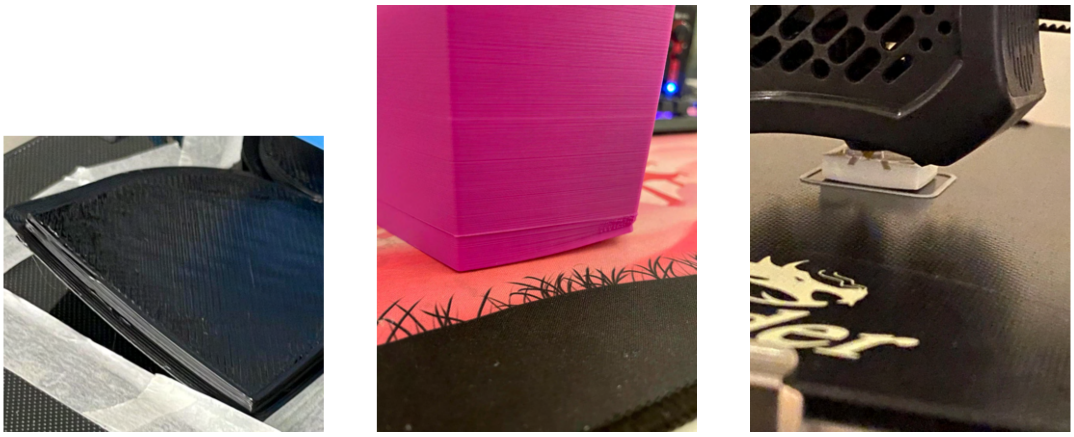


Figure 2.13: Different levels of warping, caused by insufficient print bed heating. Image sources from left: [25], [26], [27]

Figure 2.13 demonstrate different warping levels, from left: "high", "medium" and "low". Different severity levels are produced by either insufficient bed heating, lack of machine thermal insulation via an enclosure if printing with warping-prone filaments like ABS and poor first layer adhesion to the print bed. Depending on print intended use and defect severity, the print may be used or will have to be reprinted, with the correct settings and/or environment.

Blobs and zits defect can show up with different severity levels as well, caused by intermittent overextrusion during printing.

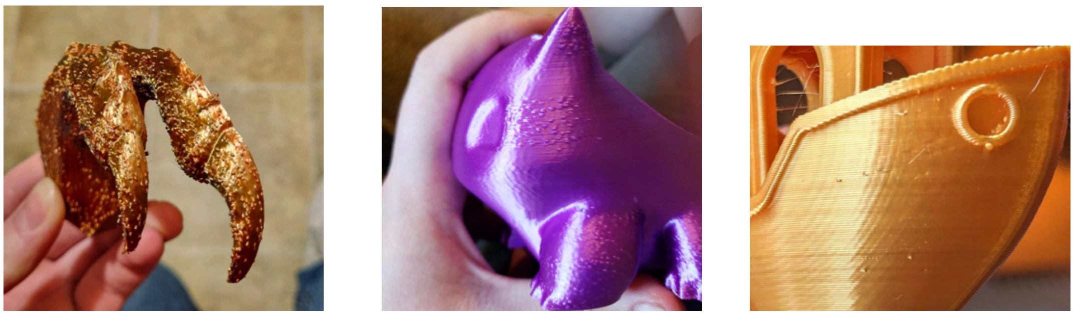


Figure 2.14: Different levels of blobs defect on the surface of print, due to overextrusion. Image sources from left: [28], [29], [30]

Figure 2.14 present varying levels of the blobs and zits defect, from left: "high", "medium" and "low". Blobs and zits usually are a visual defect, introducing an additional post-processing step after printing. The excess material has to be sanded off, if a smoother surface is desired.

Spaghetti defect on the other hand in many cases mean that the print ended up in a total failure, due to the other issue. For example:

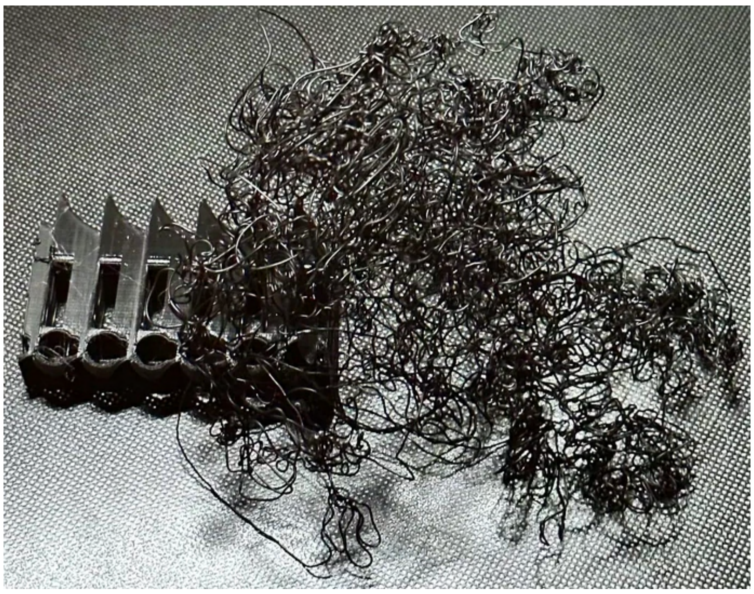


Figure 2.15: Spaghetti defect, which occurred due to loss of bed adhesion. [31]

Here, on figure 2.15, the model has detached during the printing and ended up in a spaghetti defect. That would be a total loss, the print is not finished and necessitate a reprint.

A defect could be assessed incorrectly, either by underestimating its severity or by overestimating its severity. An underestimation may have detrimental effects on the print job, as the defect may be more severe in reality and might need to be restarted, to avoid wasting energy and material. On the other hand, overestimation may stop more prints and save material, at the expense of being overactive and stopping print jobs that didn't need it, leading to material and energy again. One solution would be to inform the printer operator, if a severity threshold will be reached and the operator would make the decision if the print will be stopped, or not.

Stringing defect have been chosen as the defect for the project, due to fact that it may appear without easily visible reason. Wrong filament storage, wrong printing process settings may lead to stringing [17] [18]. Consistent appearance of the said defect may force the machines' operator to verify plastic filament quality, machines' calibration and 3D printing jobs' settings. Despite stringing being a visual defect, it takes time to remove excess material with sharp tools, which in effect if the operator is not careful enough, will lead to the printed objects' damage. As effect, excess plastic waste may be produced.

2.6 Related Works

The following section presents past work on the topic of flaw detection in 3d printing.

2.6.1 Scientific research

Some research have been done on defect detection in 3d printing. In particular, Machine Learning and Deep Learning methods find use in defect detection. Solutions proposed are not only vision-based, but also sensor-based [32] [33]. Deep Learning-based methods involve the use of CNN-based models to detect various issues with 3d prints like stringing [34] or other printout issues [35].

Deep learning finds its use in other 3d printing processes. In, [36] a CNN based flaw detector is employed, by analyzing a thermal image from laser-powder bed fusion (LPB-F) process. The network has accuracy of 97%. However, the model is trained to detect flaws specific to the LPB-F process, being radically different from the FDM process, used in this project.

Classical computer vision is being utilized as well. Study [37] compares a 3d-scanned model of a resulting print against a CAD model, by computing the distance between scanned vertex and the ground truth vertex, yielding possible deviations in the printout surface. The method is able to classify if a underfill/underextrusion or overfill/overextrusion surface defect is being detected.

In [38], the project compares workpiece simulation against a picture from calibrated camera. The system continuously captures the work piece as it is being printed and compares it against the simulated image. Both the capture and simulation is being segmented, so the shape, size and infill pattern can be compared. In [34], an attempt to detect honeycomb infill defects, by creating a dataset that contains both defected and flawless top-down pictures of infill, then a CNN-based classifier is being trained that classifies if the infill has flaws or not on layer-by-layer basis. In [35], a dataset of 1166 images in the training set and 498 images in the test set, captured by a CCD camera, have been created. The processed pictures in the study are in grayscale, which decreases the image size, at the expense of losing color information, which may be useful. Additionally, the study labeled their data as 0 and 1, where 0 means that the print is without defects and 1 meaning that the print contains defects.

An attention-based method have been attempted to solve the problem of flaw detection in 3D printing. The method involves the use of attention mechanism to predict printing parameters on the fly. The model is able to assign one of three labels ("low", "good", "high") to four of 3d printing parameters - flow rate, lateral speed (X and Y axes print speed), z offset and the hotend/printing nozzle temp and is able to adjust the parameters in GCode on-the-fly. The created dataset contains 1.2m images of the nozzle region, which showcases how the plastic is being extruded. Each snapshot is automatically labeled by the current printing parameters: actual and target temperatures, flow rate, lateral speed and the z offset parameters [39]. This approach, hewer does only predict and estimate the ideal parameters, it does not perform the detection of a flaw, but actively tries to prevent it from happening, by ensuring that ideal print parameters are being used.

Image quality metrics based methods are being used as a way to asses if a 3d print is of a good quality. In [40], the paper describes the application of the SSIM (Structural Similarity), CW-SSIM (Complex-Wavelet SSIM) and STSIM (Structural Texture Similarity) metrics in the context of 3d printing quality assessment, by assuming that a high-quality print will have a homogeneous surface, meaning that parts of the print will be self-similar. Each print picture (taken by a high-resolution camera or scanned using a flat-bed scanner) is divided into 4 and 16 blocks. For each block the metrics have been calculated, yielding a 4x4 or 16x16 matrix of scores. The study concluded that the use of SSIM metric on its own is not sufficient, but STSIM use for scanned images allows to distinguish between a good quality print and a low quality one, if a division into 16 blocks is assumed.

2.6.2 Open source and commercial solutions

Commercial solutions and open source solutions dedicated for 3d printing flaws are in active development.
First solution found is QuinlyVision by 3dQue. According to the company's documentation, it's an object detection model which is able to detect 5 FDM 3D printing flaws: Spaghetti, Underextrusion, Warping, Stringing and first layer adhesion issues [41]. The software seem to be able to run on a Raspberry Pi 4, meaning that either the used model is quantized or runs on the cloud, as the mentioned SBC (single board computer) lacks a CUDA-enabled GPU [42]. The company does not disclose what model was used or what and how big dataset was being used to train their model. QuinlyVision does not perform severity assessment of the detected flaw, however it seems to be planned as a future software upgrade [41].

A similar solution developed by PrintPal, called PrintWatch have been found. It's a solution that integrates with OctoPrint and Klipper print host software, enabling users to detect 3d printing defects. The company does not mention what defects are being used or what model is being utilized [43]. The model seem to be executed on the cloud, as the examples provided by PrintPal require an API key and the software distributed by the company are plugins to aforementioned print host software. Similarly to QuinlyVision, PrintWatch performs only flaw detection.

Roboflow is a dataset and model training software platform. It enables its users to create and train various computer vision models. The platform supports a variety of deep learning models, including 19 object detection algorithms (different flavours of YOLO and other multi-modal models like GPT-4 or LLaVA), 7 classification models (ViT, YOLO-based classification, OpenAI CLIP and Efficient net), 6 instance segmentation models, 1 semantic segmentation model and 2 keypoint detection models. On Roboflow many users train their own models, but the trained models are limited to the datasets created by the said user.

2.6.3 Summary

The presented scientific research and solutions focus on defect detection, without assessing how severe a defect is, which in production at a scale may be crucial. Only one of the presented studies focus on severity assessment and is based on the self-attention mechanism, which is using a massive (over 1 million samples) dataset and is not feasible for manual data collection. Open-source and commercial solutions do provide defect detection, but also lack severity assessment, which may lead unnecessary job cancellation, increasing energy and material costs.

2.7 Datasets

Studies mentioned in 2.6 are using data collected by the researchers themselves. Not many of the studies used any public datasets, nor published the created datasets, posing a challenge - how to obtain the data necessary for a 3d printing defect object detection model training?

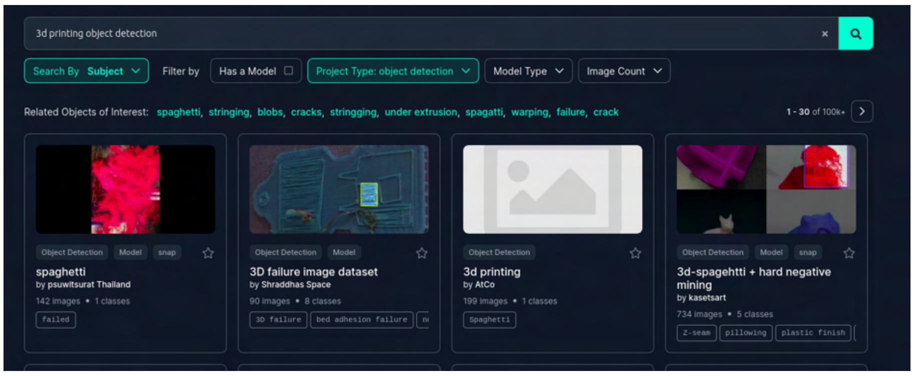
2.7.1 Object detection public datasets

Object detection is a popular problem in deep learning. Many public datasets exist to help researchers train and evaluate their findings against other works. There are many datasets, but the most known are [44]: MS COCO [45], Pascal VOC [46], ImageNet [47]. However, none of the mentioned datasets include 3d printing flaws, the datasets include "common" real-life objects. For example Pascal VOC dataset includes 20 classes like "car", "cat" or "motorbike", but none of the classes refer to 3d printing at all.

2.7.2 3d printing datasets

Manual labelling process could be used to solve the issue, however manual data labelling is a time-intensive process, despite the fact that many software distribution exist to streamline and speed up the process [48] [49]. Studies use varied sizes of their datasets, as mentioned in 2.6, however it seems that a bigger dataset is being favoured [50] [34] [36].

The Roboflow platform mentioned in 2.6 allows users to publish their own datasets, serving as a place for computer vision experimentaion platform. Datasets on Roboflow are searchable on the platform, allowing to filter by the dataset name and labels contained within the search query. Query "3d printing" returns a great amount (more than 100k) of user-submitted datasets.



3d printing object detection

Search By Subject · Filter by · Has a Model · Project Type: object detection · Model Type · Image Count

Related Objects of Interest: spaghetti, stringing, blobs, cracks, stringging, under extrusion, spagatti, warping, failure, crack

1 - 30 of 100k+

Object Detection · Model · snap
spaghetti
by psuwitsurat Thailand
142 images • 1 classes
failed

Object Detection · Model
3D failure image dataset
by Shraddhas Space
90 images • 8 classes
3D failure · bed adhesion failure

Object Detection
3d printing
by AtCo
199 images • 1 classes
Spaghetti

Object Detection · Model · snap
3d-spagehtti + hard negative mining
by kasetsart
734 images • 5 classes
Z-seam · pillowing · plastic finish

Figure 2.16: Query results in on the Roboflow platform.

Datasets submitted on the platform vary in size, class count and class names, requiring further post-processing to unify similar class names ("stringing" vs "stringging") and to merge the selected datasets together. Furthermore, it's necessary to assess if the images within the selected datasets are of good quality and the labels are correct, as some of the defects may be incorrectly labeled. ("stringing" vs "spaghetti" vs "poor bridging")
Each dataset can be exported to one of most popular labeling formats associated with the most popular object detection datasets like COCO or Pascal VOC [51], simplifying data merge process.

2.8 Analysis discussion

Commonly occurring 3d printing defects, along with a review of current both commercial solutions and scientific approaches to solve the problem of 3d printing flaw detection and quality assessment have been presented.

Presence of user-submitted labeled datasets solves the issue of data availability, as the datasets published on the Roboflow platform are with the CC-BY 4.0 licence, allowing the use of the datasets, assuming correct attribution of the dataset authors. The data available on Roboflow can be exported to one of the most-commonly used data formats, further simplifying the dataset creation process.

Since the amount of 3d printing defects is limited and their characteristics are consistent and don't vary too much between prints, the use of more advanced object detection frameworks like Open Vocabulary Object Detection or Zero-Shot object detection may not be justified, because for Open Vocabulary framework, a dataset needs to be captioned properly, so the language model will be able to infer labels for a defect. The presence of already labeled datasets on Roboflow, enables easier training and evaluation of a resulting model, relieving from the need of manual data labeling of the defects.

It seems that little work was done towards the severity assessment of 3d printing defects (stringing defect in particular), as only one study [39] seems to achieve a similar outcome, by assigning labels to the toolpaths as they are being laid out. However, the labels are not continuous, which may limit the usability of the solution. Commercial solutions mentioned in 2.6.2 seem not to attempt flaw severity assessment, but may be included in the future updates.

Severity assessment of print is necessitated by the material costs and energy costs of the print process. A print that wasn't stopped in time and had severe enough defects will lead to material and energy waste, creating a need to not only detect defects, but also assess how severe the defect is.

2.9 Problem statement

Considering the problem analysis, the object detection models, methodologies and data presence, the following project statement have been formed:
"Given the current object detection approaches and the dataset availability, how can one create a defect object detection and severity assessment model in the context of FDM 3d printing, to aid 3d print job supervision?"

Chapter 3

Technical Analysis

This chapter aims to describe and analyse what methods can be used to create a defect detection and severity assessment model.

3.0.1 Deep Learning

Deep learning is a subset of machine learning, based on Artificial Neural Networks. In deep learning a *model*, based on a multi-layer architecture is used to extract progressively higher-level features from the submitted data. Depending on the problem's domain, different features learned. The features are being extracted automatically, without previous feature extraction; the model itself performs feature extraction and uses them for downstream tasks, like classification or object detection [52]. In the context of computer vision, a earlier layer may identify edges, corners or lines, whereas deeper layers will detect more complex features [53]. The task of feature extraction may be performed by so-called Convolutional Neural Networks (CNNs) or Vision Transformers (ViT).

3.0.1.1 Convolutional Neural Networks

Convolutional Neural Networks are a type of deep learning that utilize image filters to learn and extract features from an image. The filter is being slid along input, producing a matrix of values, called a *feature map* [54]. Such a feature map is being used as input to the next layer, allowing the network to learn increasingly complex features [54]. A filter applies the following operation:

$$g(x,y) = \sum_{j=-R}^{R} \sum_{j=R}^{R} h(i,j) \cdot f(x+i, y+j) \tag{3.1}$$

Figure 3.1: The definition of output pixel as a result of applied filter.

In equation 3.1 output pixel $g(x, y)$, where x and y are image positions within the sliding window, i and j are coefficient positions within the filter, the pixel $g(x, y)$ is calculated as dot product of surrounding pixels $f(x, y)$ and filter coefficients $h(i, j)$. R is the filter's kernel size. The resulting layer is being fed into an activation function which helps the network learn non-linear features [54].
Various types of pooling layers are being utilized as well, to perform feature selection and reduce feature maps' dimensions, before feeding the outputs to the next layer. For instance MaxPooling selects the highest within a sliding window of a feature map. On the other hand AveragePooling averages the values within a window [55]. Feature maps generated by the CNN layers are used in downstream tasks, like classification, object detection, instance segmentation and other.

3.0.1.2 Object detection

Object detection is a task of finding and localizing an object within a scene. Previously mentioned CNNs are usually used to perform the task, however ViT-based networks seem to be used as well [56].

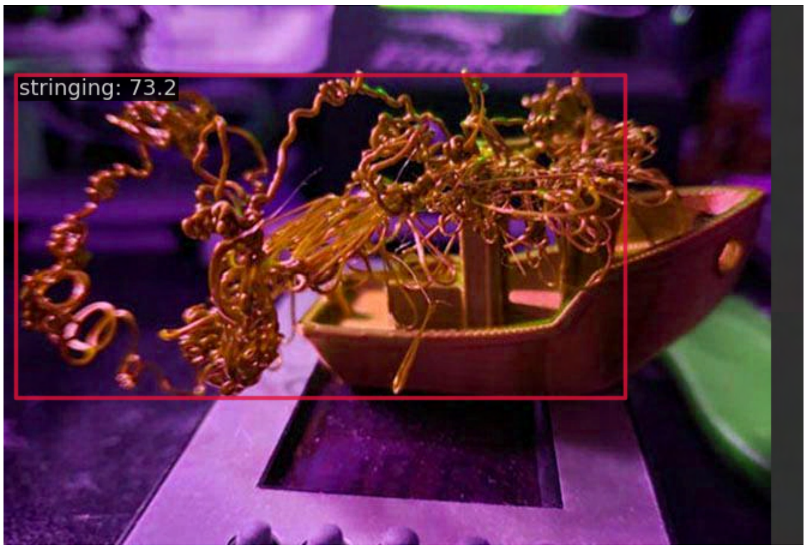


Figure 3.2: Example detection after applying an object detection model on an image.

Figure 3.2 presents an image with a localized 3d printing defect on it. The detection includes a bounding box which localizes the defect, the defect classification (or rather defect's type) and the class confidence, which informs the user on how confident the model is.

Object detection models can be divided into three categories:

- Single-stage detectors, which locate and categorize the object in "one pass" [44].
- Two-stage detectors, which divide the object detection task into two sub-tasks: object localization and object classification. [44].
- Open vocabulary object detection, a novel object detection method, aiming to detect objects beyond detected classes using captions [57].

Each object detection model include a *backbone* network, which performs the feature extraction process and based on the generated feature maps perform the detection [44].

3.0.2 Object detection model types

A **single-stage detector** locates and categorizes the object simultaneously, without dividing it to two portions [44]. Such a network directly map the image's pixels to a bounding box and a class prediction. Single-stage models are faster than two-stage models, at the expense of being less accurate [44].

In a **two-stage detector**, first the network generates region proposals, where the object is localized and then the region is being classified. Such a detector has higher accuracy, over the single-stage detector, at the expense of being slower [44].

Open vocabulary object detection is a new object detection framework, in which a model takes an input image and detects any object within given target vocabulary. To train such a model, a image-caption dataset is needed, which will cover a large variety of words but also a smaller dataset containing localized object annotations from a set of base classes. Target classes are not known during training and can be any subset of the entire input caption vocabulary. The combination of language models and CNN-based models improves resulting model's performance [58].

3.0.3 Object detection model evaluation

Each object detection has to be evaluated to asses how well the model performs. A set of datasets and evaluation metrics have been developed to facilitate it.

3.0.3.1 Detection classifications

A detection can be classified as one of four cases [44]:

- True Positive (TP) - the prediction have been made and is correct.
- True Negative (TN) - the prediction have not been made and there isn't a detection to be made.
- False Positive (FP) - the prediction have been made but is incorrect.
- False Negative (FN) - the prediction have not been made, but there is a detection to be made.

3.0.3.2 Intersection over Union

Intersection over Union (IoU) is a metric that measures the overlap between a detected bounding box and the ground truth bounding box [44].

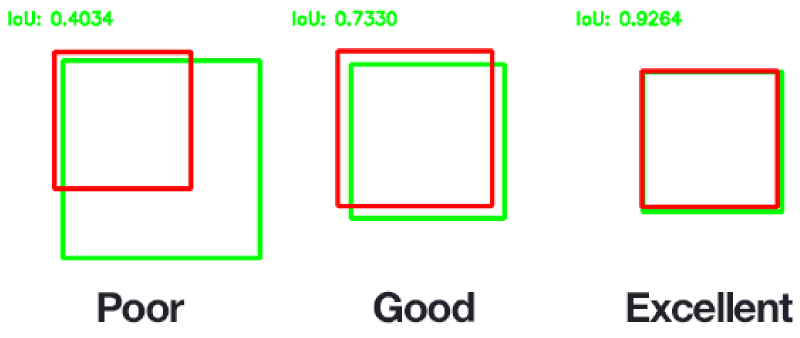


Figure 3.3: An image illustration how IoU may be interpreted. Source: By Adrian Rosebrock - http://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=57718559

Its values range from 0 to 1, where 0 means no overlap and 1 meaning that the detection box and the ground truth box are identical in size and position [44], as demonstrated on figure 3.3. A threshold of 0.5 or more may be established to assess if the detection is good enough to consider [44].

IoU is defined with the following equation:

$$IoU = \frac{|GT \cap Detection|}{|GT \cup Detection|} \tag{3.2}$$

Figure 3.4: The definition of IoU metric

In simple terms - IoU can be understood as the division of the ground truth bounding box and the detection box over the total area of the ground truth bounding box and the detection box.

3.0.3.3 Accuracy

Accuracy describes model's performance across all classes. It is computed as the ratio of the total number of samples classified correctly vs the total sample count [44].

Accuracy is defined by the following formula:

$$Accuracy_c = \frac{TP_c + TN_c}{TP_c + FP_c + TN_c + FN_c} \tag{3.3}$$

Figure 3.5: The definition of Accuracy metric

3.0.3.4 Precision

Precision says how much positive identifications were actually correct. It's a ratio between the number of accurately identified samples to the total count of positive samples [44]:

$$Accuracy_c = \frac{TP_c + TN_c}{TP_c + FP_c + TN_c + FN_c} \tag{3.4}$$

Figure 3.6: The definition of Accuracy metric

3.0.3.5 Recall

Recall indicates how many of actual positive classification were identified correctly. It is a proportion of correctly identified samples to the total number of actual positive samples [44]:

$$Recall_c = \frac{TP_c}{TP_c + FN_c} \tag{3.5}$$

Figure 3.7: The definition of Accuracy metric

3.0.3.6 Mean average precision

The mean average precision (mAP) is calculated by taking the average over all objects categories, evaluating the performance of object detectors [44]:

$$AP_c = \frac{1}{j}\sum_{c=1}^{j} Precision_c$$
$$mAP = \frac{1}{j}\sum_{c=1}^{j} AP_c \tag{3.6}$$

Figure 3.8: The definition of Accuracy metric

In the formula 3.8, j is the number of all categories.

3.0.4 Object detection evaluation datasets

Multiple datasets have been created for the task of evaluating an object detection model. Most of the well-known datasets include many kinds of common objects real-life objects and are used by many researchers as a benchmark to compare and evaluate their model's performance against other works.

3.0.4.1 PASCAL VOC

The PASCAL Visual Object Classes (VOC) is a dataset created for the needs of the VOC challenge. The newest iteration of the dataset (and the challenge) includes 20 classes of the most-common objects like "cat" or "bottle" across 11k images [46]. PASCAL VOC uses AP@0.5 as its evaluation metric.

3.0.4.2 MS COCO

The MS Common Object in Context (COCO) is another common choice for an evaluation dataset. The newest version of the dataset contains over 123 thousand of images divided 80 classes, like "person", "horse" or "zebra" [45]. MS COCO uses Average Precision as one of its metrics, with different IoU thresholds and scales:

1. Average Precision (AP) - average precision of a class:
    - AP - AP at IoU from .75 to 0.95 with a 0.05 step.
    - $AP^{IoU=.50}$ - AP at IoU greater than 0.50.
    - $AP^{IoU=.75}$ - AP at IoU greater than 0.75.
2. AP Across Scales - AP across different area of the ground truth detections:
    - AP^{small} - AP for objects with area less than 32^2.
    - AP^{medium} - AP for objects with area more than 32^2 and less than 96^2.

- AP$^{\text{large}}$ - AP for objects with area more than 96^2.

3.1 Selected models

Two models were picked to accomplish the task of stringing defect detection with severity assessment.

3.1.1 MobileNetV3

MobileNetV3 is a backbone network used in object detection, classification and semantic segmentation, optimized for edge and mobile applications [59] [60], where a GPU or other processing accelerators are unavailable for use. Here, it will achieve the task of severity classification. The architecture is comprised of so-called *depth-wise separable convolutions* (introduced by MobileNetV1), which separate feature map generation and spatial filtering mechanism, increasing layer's efficiency [61], *linear bottlenecks and inverted residual structures* (added in MobileNetV2), which allow for a compact representation of higher-dimensional feature space, thus increasing the expressiveness of non-linear per-channel transformations [62].
MobileNetV3 further enhances the architecture, by introduction of the *swish* activation layer, which overcomes sigmoids' inaccuracies in fixed-point arithmetic.
Other modifications include latency reduction and high-dimensional feature preservation by moving the *inverted bottleneck structure* past the final AvgPool layer, resulting in increased efficiency [59]. Thanks to the use of aforementioned *swish* activation function, the number of image filters could been reduced to 16 from 32, while maintaining the same accuracy, saving more computational power [59].

The models' performance have been measured on a Google Pixel [63] smartphone and presents as following:

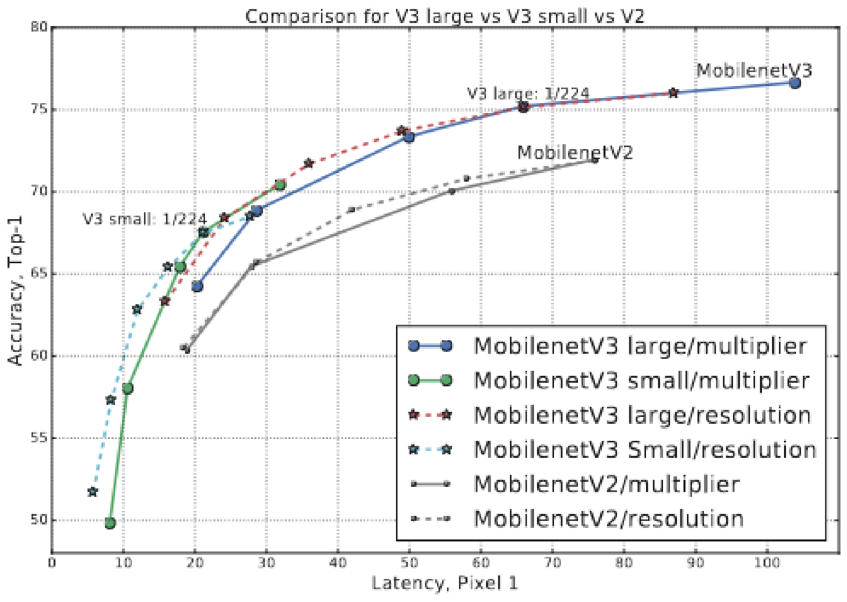


Figure 3.9: MobileNetV3 comparison of Top-1 accuracy vs latency at different internal resolutions

Figure 3.9 show that the MobileNetV3 is fast enough to operate on a smartphone, without any additional accelerators, while maintaining decent classification accuracy.

3.1.2 YOLOv8

YOLOv8 is the 8th iteration of popular single-stage object detection model architecture. The model is composed 3 blocks [64]:

1. Backbone network - the foundation of the model, responsible for feature extraction from the input image, based on CSPDarknet53 network.
2. Neck - based on Path Aggregation Network, used to improve information flow across different resolutions, enhancing multi-scale feature extraction.
3. Head - responsible for processing feature maps, producing detections and classifications.

3.1.2.1 CSPDarknet53 backbone

CSPDarknet53 is the backbone network used by YOLOv8 based on Darknet53 CNN, comprised of several convolutional layers and residual layers, outperforming similar architectures like Resnet-101 and Resnet-152 [65].

CSPDarknet53 extends on Darknet53 by integrating the Cross Stage Partial Network (CSPNet). It works by splitting the feature maps in a dense block into two, in which one of the split feature maps will go directly to the output and the other feature map will go through another dense block. That way the gradient information is not duplicated, like it's the case in DenseNet [66], increasing the efficiency of the architecture.

3.1.2.2 Path Aggregation Network

Path Aggregation Network (PANet) involves the use of a Feature Pyramid Network (FPN), bottom's-up path augmentation and adaptive feature pooling in order to improve networks' performance.

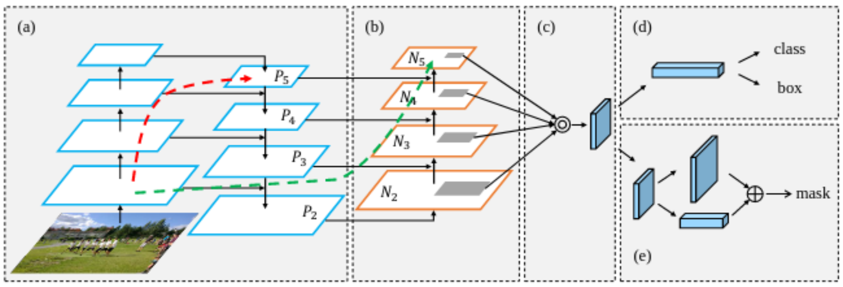


Figure 3.10: PANet architecture visualized. (a) - Feature Pyramid Network, (b) - Bottom-up path augmentation, (c) - Adaptive feature pooling, (d) - Box and class detection head, (e) - semantic segmentation mask generation head. [67]

Feature Pyramid Network (FPN) is a type of top-down architecture, in which multi-scale features are being detected via convolution at different scales. However, PANet enhances localization of the feature hierarchy by propagating stronger responses of lower layers, as high-responses to edge features or instance parts are a string indicator to accurately localize instances. A "shortcut" is being added, that consists of 10 layers is being added (green arrow on figure 3.10)[67].
Augmented bottom-up structure follows FPN to define layers, which produce feature maps with the same spatial size in the same network stage.
Adaptive feature pooling pools features from all levels of the FPN and fuses them for a prediction. Each proposal is clustered into four classes, based on the levels from the FPN, improving prediction accuracy [67].

3.2 Performance requirements

3.2.1 Model inference time

Solutions mentioned in 2.6.2, try to achieve real-time model performance, which may not be necessary, depending on the printout size and 3d printer's lateral speed 2.2.1. Small models will print faster, larger models will take more time to finish.
To estimate the average print time and derive model inference time requirements, a set of test objects have been sliced, varied in size and print parameters, sliced in the Ultimaker Cura software. Print time estimates are directly derived from the generated GCode, based on the print movements and programmed print speed. The slicer software have been configured with the Creality CR-20 printing profile and a nozzle diameter of 0.4mm.

The following test object have been processed:

1. *3DBenchy* by *Creative Tools* - one of the most commonly used test prints used for benchmarking and testing 3D printers [68].
2. **MINI* All In One 3D printer test* - a test object aiming to test all of the printer's capabilities to print. The test contains a lot of small details and separated structures, provoking more travel movements [69].
3. *XYZ 20mm Calibration Cube* by *iDig3Dprinting* - a simple test object aiming to test machine's dimensional accuracy. The object contains a lot of infill, allowing to test out the influence of the infill type and percentage vs print time [70].

Each object is placed directly in the middle, the default purge line is being kept. The following table describes what parameters have been configured in the slicer software:

Parameter Name [paremeter unit]	**Parameter Values**
Layer Height [mm]	0.3, 0.2, 0.16, 0.12
Infill Density [%]	5, 20, 40, 50, 80, 100
Print Speed [mm/s]	10, 40, 80, 100, 200, 300, 500
Support Generation [bool]	True/False

Table 3.1: Table presenting tested parameters.

For each time estimation, median layer time, average layer time, layer's time mode have been calculated. Additionally, the FPS values of object detection models have been considered.

3.2.1.1 Print time estimation results

Result Name [paremeter unit]	Result Value
Median layer time [secs]	39
Average layer time [secs]	46
Mode layer time [secs]	15

Table 3.2: Table presenting print time estimation results.

The table 3.2 represent the print time estimation results. The lowest time was achieved was with the mode time, equaling to 15 seconds per layer. Having a low inference time means, that the model will be able to observe most of the layers during the printing process, ensuring that the printed object is inspected closely. To further examine if the mode layer time, as the model's inference time requirement is a reasonable value, it is necessary to see how some of object detection models perform on benchmark datasets. A performance comparison is made in [44] of a couple of object detection models:

Model Name and backbone network	FPS value	Frame time [sec]
Fast RCNN + VGG16	0.03	33.(3)
Faster RCNN + VGG16	5	0.2
SSD300 + VGG16	46	0.021
SSD512 + VGG16	19	0.05
YOLOv5 650 + CSPDarknet	140	0.007

Table 3.3: Object detection models performance comparison made on COCO 2015 and 2017 datasets [44].

Models presented in 3.3 show that at the slowest model (FastRCNN) takes on average 33 seconds to generate predictions, however other single-shot models take significantly less time to generate predictions, generally taking less than 1 second to make predictions, meaning that single-shot models presented in the comparison [44] will be able to make multiple predictions per one layer. Of course, some shorter layers may be missed, if the printed object is extremely small and the mode layer time will be considered as the inference time requirement.

3.2.2 Target hardware platform

Mentioned in section 2.6.2 OctoPrint is a popular printer control software, targeting popular Raspberry Pi 4 education computer. Raspberry Pi 4 proven to be a good choice for the maker community, offering a good balance of performance and a great

variety of I/O options, enabling a broad range of interfacing options with the external world [42] [71].
Because OctoPrint (via the OctoPi software distribution) offers out-of-the-box experience on the Raspberry Pi 4, it is often used as the go-to solution to control a 3D printer. A camera can be attached as well, transforming Raspberry Pi 4 into a full-featured 3D printer controller.
A potential solution has to be fast and efficient enough to run on the Raspberry Pi 4, without interrupting the printing process, as the commands have to be streamed to the printer via the USB port [72]. Raspberry Pi 4 offers the following hardware specifications [42]:

Parameter name	Value
CPU	64-bit Quad Core ARM Cortex A7 @ 1.8 GHz
RAM	4GB of LPDDR4-2400MHz SDRAM
GPU	Broadcom VideoCore IV

Table 3.4: Raspberry Pi 4 hardware specification important to the project.

The GPU presented in table 3.4 does not support for CUDA or ROCm software development platforms, stripping the computer from deep learning acceleration, without the use of external hardware like the Google Coral TPU. OpenCL is supported by the computer, hovewer OpenCL support for inference is inconclusive for the Raspberry Pi 4 platform, leaving CPU to do all of the calculations [73].
In that case, the model has to fit in the computers' RAM and run fast enough and infer on layer-by-layer basis. To make sure that OctoPrint has enough of RAM for its operation (without the use of swap, as the SD cards are a slow mass-storage medium), it seems that reserving half of specified RAM amount (2 GB) a reasonable option.

3.2.3 Model accuracy

The COCO and the Pascal VOC publish the datasets challenge results online, aiding with model accuracy mAP score derivation. Since Pascal VOC [46] dataset considers samples that have more than 0.5 IoU as True Positive, the same score is considered in the case of the COCO dataset. For each dataset, the average, the mean and the mode values have been calculated and present as following, based on the published results from the challenges:

Dataset Name	**mAP @ 0.5 average**	**mAP @ 0.5 median**	**mAP @ 0.5 mode**
Pascal VOC 2012	0.61	0.71	0.81
COCO	0.59	0.62	0.66

Table 3.5: Table presenting calculated statistics of mAP @ 0.5 scores.

The mode of Pascal VOC seems to be quite high, comparing to the COCO dataset. One of possible causes is that, the COCO dataset is a much larger dataset (886k instances vs 19k instances). Considering the lower mode value (0.66), it seems reasonable to select it as the accuracy requirement, as it was the most commonly submitted score in the COCO dataset challenge.

3.3 Model functionality

3.3.1 Model severity assessment

Section 2.5 established 4 severity labels, essentially reducing the severity assessment problem to a classification problem. In a 3d print smaller defect can occur, which may have less of an influence over whole print job and will not need any post-processing to full-fill its purpose. Small defects, like isolated blobs and zits or single stringing strands should be classified as "low" severity, as they have little influence on the overall print. In cases, where the defect occurs consistently but does not affect the object's functional usability should be "low" severity as well. The decision whether to stop the print may lie on the machine's operator, if a "low" severity defect will occur.

3.3.2 Model severity assessment metrics

It is crucial to correctly assess defect's severity, but also to ensure that false positive rate is as low, as it is possible. Not only the accuracy has crucial impact on model's performance, but also False Positive and False Negative rate. If a less-severe defect is being classified as a higher-severity one, then it may lead to unjustified job stop, for example if a defect had a the ground truth class assignment as "medium" severity and was classified as "total", then the print might be stopped without a reason. Similarly, if defect's severity is underestimated, then a print may continue, when it was necessary to stop it, to avoid material and energy losses. In the study [39] the highest accuracy achieved was 85.3% across all four parameters examined in the study, increasing the number to 90% seem like a reasonable idea for a proof of concept project.
Another case to consider is detected defect count across multiple frames. A detection could be intermittent; it could happen in only one frame, meaning that a single invalid detection and severity classification could stop 3d print without any apparent reason. For that reason consider severity assessment over several layers and if the single layer time is long enough - multiple frames. That way it will be possible to limit the amount of false positives and false negatives.

In subsection 3.2.1.1 it was established, that the most common value layer time among tested models was 15 seconds. In those 15 seconds, a defect may start appearing, which may lead to different severity assessments across multiple layers. It may be necessary for a severity assessment to be consistent over the course of multiple layers to be correct. In how many layers a defect will fully emerge is dependent on the layer height and on the defect type.

3.4 Test requirement list

3.4.1 Performance requirements

To summarize, the following requirement list have been formed:

- The model should occupy no more than 2 GB of RAM during inference, to prevent the swap space from being used.
- The model inference time should not exceed the mode layer time, being **15 seconds**.

3.4.2 Functional requirements

3.4.2.1 Model accuracy

- The model should reach at least 0.66 COCO mAP @ 0.5 on the stringing defect on the dataset test split.
- The model should be able to reach at least 90% of accuracy in the testing across all four defect classes during a test print, across multiple layers.

Chapter 4

Design

This chapter aims to describe the solutions' and severity assesment score design.

4.1 System design

The overall system will consist of several components:

1. 3D printer controller, connected to the 3D printer via USB, sending GCode commands and recording the printing process.
2. 3D printer itself, executing commands sent by the controller, resulting in a printout.
3. Deep learning models residing on the 3d printer controller, perfroming the task of stringing defect detection and severity assesment.

Overall system schematic is shown on the following diagram:

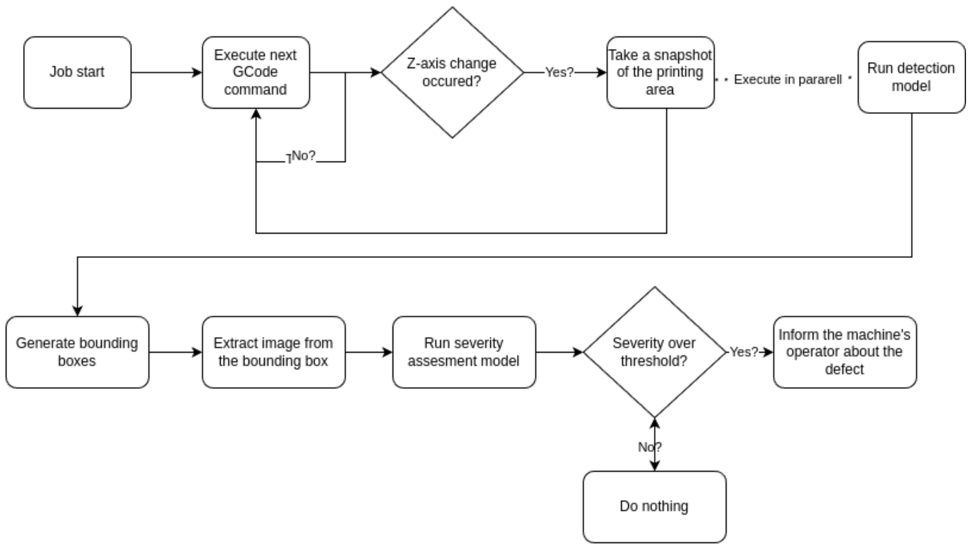
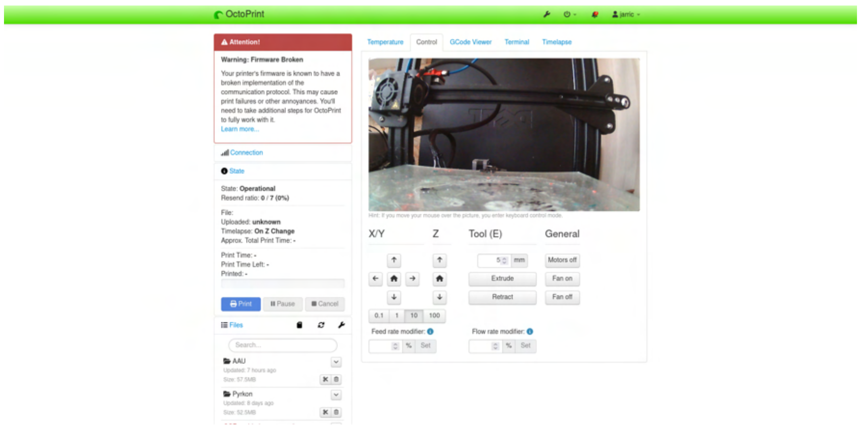

Job start
Execute next GCode command
Z-axis change occured?
Yes?
No?
Take a snapshot of the printing area
Execute in pararell
Run detection model
Generate bounding boxes
Extract image from the bounding box
Run severity assesment model
Severity over threshold?
Yes?
Inform the machine's operator about the defect
No?
Do nothing

Figure 4.1: System's design diagram

Figure 4.1 represents the system's design. The 3d printer controller sends the commands to the machine in a streaming fashion, once the controller receives a confirmation from the machine, next command is being sent. When a Z-axis movement command is being sent, a snapshot is being taken with the connected camera to it, resulting in a timelapse of the 3d printing process. A live stream from the printing process is being captured as well. The snapshot is forwarded to the deep learning models, running in parallel to the printing process, making sure that the models running will not disturb the printing process. If a detection is made, the detected defect is then forwarded to the severity assessment model, resulting in a severity class. If the class is severe enough, a notification is sent to the machines' operator.

4.1.1 Hardware

Raspberry Pi 4 described in section 3.2.2 is an ideal hardware platform to host the controller, due to its low cost, popularity and relatively good performance. The controller will be connected via USB serial connection to the 3D printer, sending GCode commands to it. The computer is powered via a 5V USB-C power supply and will have OctoPi Linux distribution installed on it, providing a ready-to-use environment for testing. Sliced 3D models will be uploaded to the computer via a web-browser.



Warning: Firmware Broken

Your printer's firmware is known to have a broken implementation of the communication protocol. This may cause print failures or other annoyances. You'll need to take additional steps for OctoPrint to fully work with it.
Learn more...

State: **Operational**
Resend ratio: **0 / 7 (0%)**

File:
Uploaded: **unknown**
Timelapse: **On Z Change**
Approx. Total Print Time: -

Print Time: -
Print Time Left: -
Printed: -

AAU
Updated: 7 hours ago
Size: 57.5MB

Pyrkon
Updated: 8 days ago
Size: 52.5MB

Hint: If you move your mouse over the picture, you enter keyboard control mode.

Figure 4.2: OctoPrint GUI with visible view on the printing area.

Figure 4.2 presents OctoPrints' GUI visible in a web-browser. The camera connected to the controller points directly to the printing area, at the front of Y-axis and has the following specifications:

Parameter name	**Value**
Manufacturer	Imilab
Maximum resolution	1920x1080
Framerate	30FPS
Connection interface	USB 2.0

Table 4.1: Specifications of USB webcam connected to the 3D printer controller

4.1.2 Severity score design

3D printing faults defect analysis conducted in 2.5 could be quantified to simple labels, essentially converting the problem of severity assesment to a classification problem, simplifying data labeling process. Based on analysis four labels are proposed:

1. Label "Low" - the defect is visible, but has little-to-none influence on item usability and/or aesthetics. Simple post-processing steps may be needed to further improve the item.

2. Label "Medium" - the defect moderately affects items' usability and/or aesthetics. More post-processing or item alteration may be needed for item to fully fulfill its purpose.

3. Label "High" - the defect severely affects items' usability and/or aesthetics. The item has to severely modified and post-processed to be usable.

4. Label "Total" - the defect is completely destroys items' usability and aesthetics. No post-processing or item alteration will make the print usable

However, stringing defect is a visual defect and rarely results in print failure nor influences object's functionality (assuming a correct post-processing process). Having that in mind the following labels have been defined, to better reflect stringing defect's nature:

1. Label "Low" - single strands of extruded plastic are visible on the resulting printout.
2. Label "Medium" - multiple strands of extruded plastic in multiple places are visible on the resulting printout. Some oozed plastic blobs may be visible on the dragged strands.
3. Label "High" - strands of extruded plastic in multiple places dominate the printout. Oozed plastic blobs may be visible on the dragged strands of extruded plastic.

Instead of discrete labels the severity score could be continuous, making the problem of severity assessment a regression problem. The severity of the defect could be modeled after the density of plastic strands within the detected defects' bounding box, creating a score from 0-3, where 0 is no stringing within the bounding box and 3 is the most severe case of stringing.

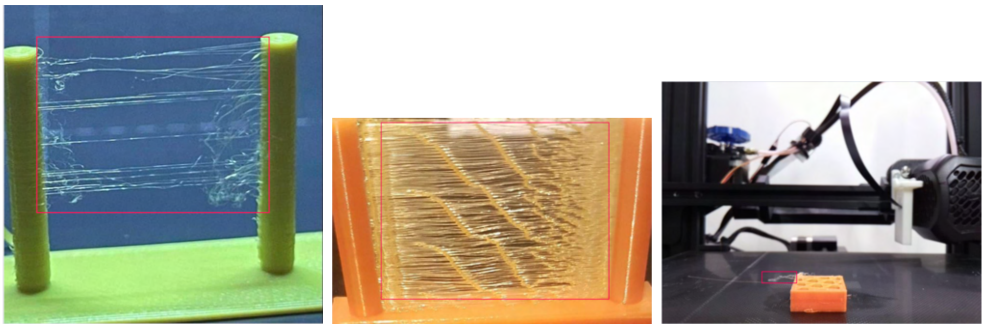


Figure 4.3: Examples of assigned severity regression scores, taken from the created dataset.

Figure 4.3 present different stringing defect severities, scored after their densities within the bounding box. The severity scores would be, from the left: 2 (due to visible, connected plastic strands between two points), 3 (due to high density of the defect, forming thicker blobs), 0.5 (due to little density of the plastic and being localized to one point within the printed object)

Chapter 5

Implementation

5.1 Dataset Creation

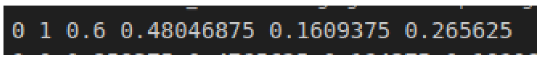
5.1.1 Stringing defect dataset

A stringing defect dataset have been created on the Roboflow platform, by combining datasets shared by other users on the said platform. The following datasets have been combined:

- "3D Printing Error" by AtCo [74] - dataset containing 315 stringing defect samples.
- "Defect 3d Print" by rmuti [75] - dataset containing 848 stringing defect samples.
- "3D printing defects" by HCMUT [76] - dataset containing 626 stringing defect samples.
- "3dprintfail_v2" by MoschA [77] - dataset containing 26 stringing defect samples.

The datasets contained samples from other 3d printing defects, those labels and samples have been removed, to simplify and speed up the training process.
In the resulting dataset, the samples have been resized to 640x640 (which is the size required by the selected object detection model) and auto-oriented to match the orientation of the bounding box labels.

Each label in the dataset have been labeled using Ultralytics [78] YOLOv8 labeling format with the addition of severity class to a bounding box:



```
0 1 0.6 0.48046875 0.1609375 0.265625
```

Figure 5.1: Example label from the created dataset

Figure 5.1 representing an example label, is formatted in the following manner: *<defect_class> <severity_class> <normalized_bbox_center_x> <normalized_bbox_center_y> <normalized_bbox_width> <normalized_bbox_height>*

Such a label can be directly consumed by the YOLOv8 model or be easily converted to other formats on the Roboflow platform or be converted to other formats by the torchvision [60] package.

5.1.2 Severity assessment label creation

The total amount of bounding boxes for the dataset exceed 3000. Manually labelling such an amount of bounding boxes will take a significant amount of time. To automate the process, a small subset (around 300 samples, picked at random) of samples have been labelled and then a small bootstrap model have been trained.
Based on the comparison [60] prepared by the torchvision project, MobileNetV3 classification model have been selected, as the bootstrap model. The model will assist the user with labelling, producing suggestions, that after user acceptance are being used. Wrong labels were corrected on-spot. After reaching 500 of labeled samples, the rest of bounding boxes were labeled by the model without user supervision, to save more time.

5.2 Software implementation

The solution have been implemented using the Python programming language, using torch with torchvision [60], ultralytics YOLOv8 implementation [78], albumentations image augmentation library [79] and torchmetrics metric calculation library [80]. Currently, training and evaluation code is implemented, however due to time constraints, operator notification system is left unimplemented as it would require integration with OctoPrint. The system tests rely on the OctoPrint timelapse system.

5.3 Training

Both models have to be trained on the stringing defect dataset described in section 5.1, using the models described in section 3.1.

5.3.1 Severity assessment bootstrap model training

MobileNetV3 model have been chosen to assist and then automatically create necessary labels for the severity assessment process. Torchvision [60] provides the model definition. The data have been augmented with two sets of data augmentations in effort improve models' performance, provided by library albumentations [79]:

1. Set #1 - vertical and horizontal image flips, image contrast and brightness transform, random pixel dropout, random 90 degrees rotation
2. Set #2 - random 90 degrees rotation, image defocus, random pixel dropout, vertical flip

Both sets have been applied to the samples, creating two more samples out of one in effort to prevent overfitting from happening.

The chosen hyperparameters are:

Hyperparameter Name	Hyperparameter Value
Batch size	8
Learning Rate	0.0001
Epoch count	55

Table 5.1: Table containing selected severity assessment model hyperparameters

Training and validation plots present as following:

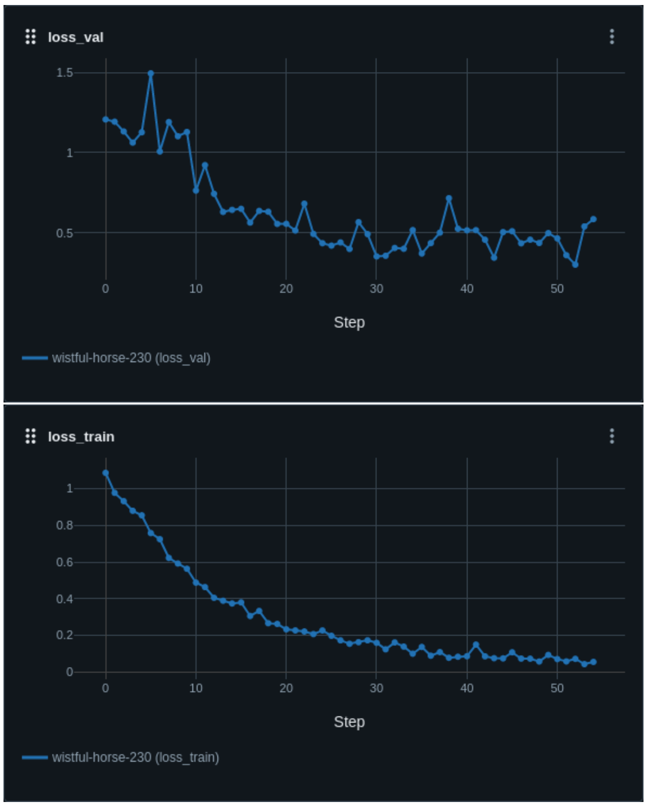


Figure 5.2: Training and validation loss plots

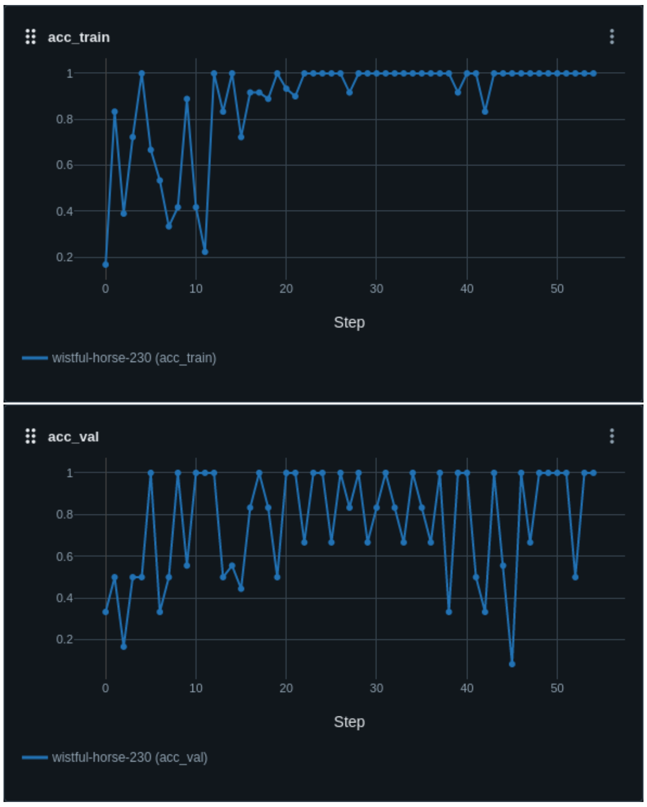


Figure 5.3: Training and validation accuracy plots

The training and validation loss plots seem to decrease during the training process, indicating that the model is indeed learning the selected 300 samples. However, the validation accuracy seem to be unstable, showing that potentially there may be difficult samples to classify. The training started at training loss of 1.08 and decreased to loss value of 0.05. Validation loss started at 1.2 and decreased down to 0.58. Training accuracy started at 0 and increased to 1, validation accuracy started at 0.33 and increased to 1.

5.3.2 Severity assessment model training

MobileNetV3 model have been chosen to classify the severity. The implementation of the model have been used provided by torchvision [60] package. The same data augmentations have been used, like in bootstrap model training. Similarly, both sets of augmentations have been applied to the dataset, creating more samples for the training purposes.

During training the model kept overfitting to the train dataset, due to that multiple iterations of hyperparameters have been tested, ending up with the following hyperparameter set:

Hyperparameter Name	Hyperparameter Value
Batch size	8
Learning Rate	0.0001
Epoch count	20
Weight decay rate	6e-1

Table 5.2: Table containing selected severity assessment model hyperparameters

MLFlow was used to monitor training progress, torchmetrics [80] package was used to calculate nessesary metrics. Training, validation and accuracy plots present as following:

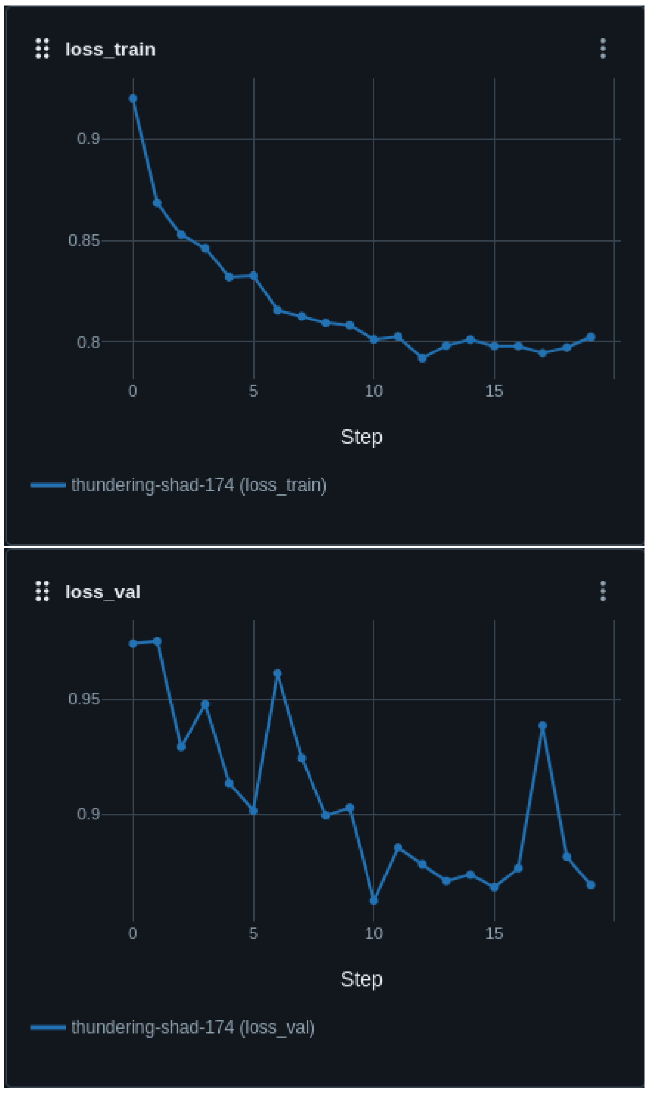


Figure 5.4: Loss plots during severity model training

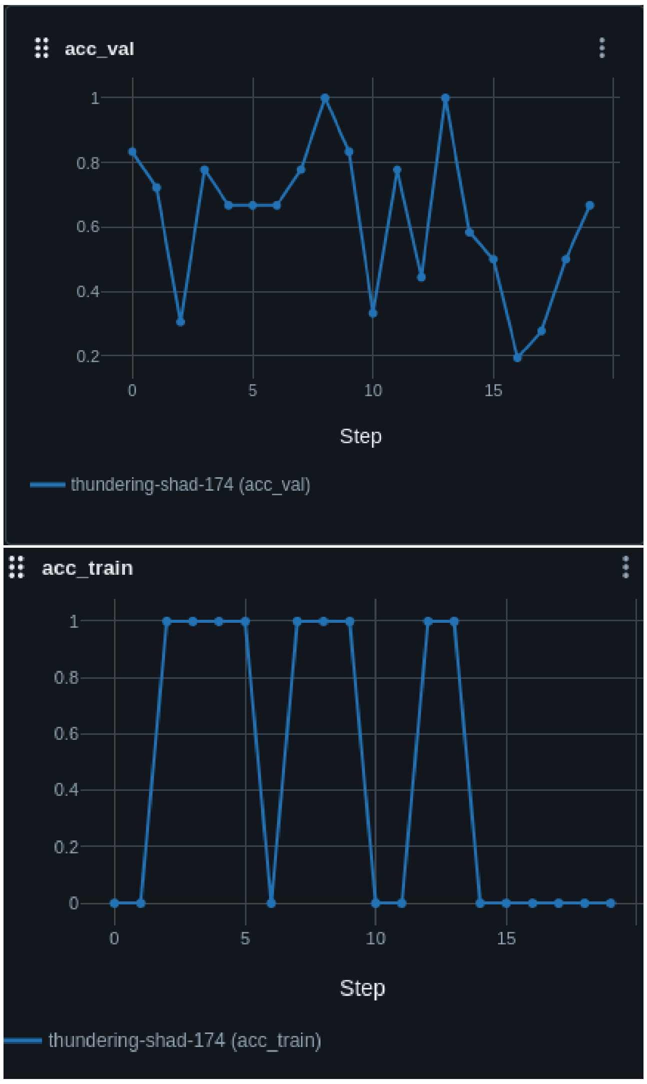


Figure 5.5: Accuracy plots during severity model training

Figures 5.4 and 5.5 show that there may be a problem with the severity dataset. Train loss and validation losses decrease as the training progresses, however the validation loss graph has sudden increases in the loss values, whereas train loss plot doesn't, indicating that there may be a problem with training and the dataset. Accuracy plots further reinforce the concern, that there may be a problem with the severity assessment dataset [81], because the accuracy does not increase, as the training process progresses.

5.3.3 Object detection model training

The selected implementation of YOLOv8 created by ultralyrics [78], due to the fact that their implementation includes everything needed to train, evaluate and test the model, hiding a lot of the boilerplate. The selected variant for training is YOLOv8s. The data have been augmented with the default set of augmentations defined by ultralytics:

1. HSV space augmentation - hue, saturation and value changes
2. Image transformations - translate, rescale, image flip left-to-right.
3. Random erasure of image data from the image

The hyperparameters were kept default, presenting as following:

Hyperparameter Name	**Hyperparameter Value**
Batch size	16
Learning Rate	0.01
Epoch count	500
Weight decay rate	0.0005
Loss functions weights	box: 7.5, cls: 0.5, dfl: 1.5

Table 5.3: Table containing selected stringing defect detection model

The training statistics plots present as following:

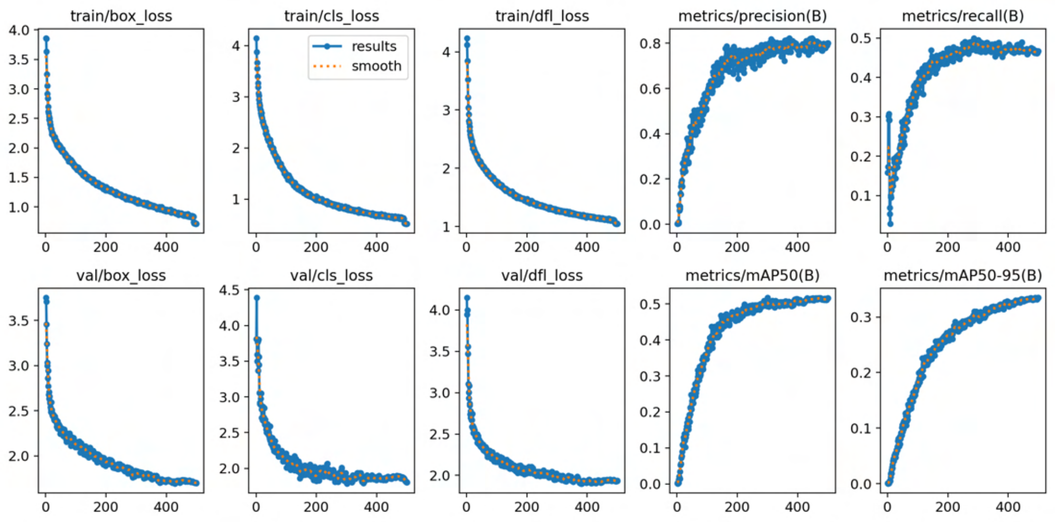


Figure 5.6: Training statistic plots of the stringing defect detection model

Figure 5.6 indicate that the model fits the dataset quite well. Both training and validation losses decrease consistently as the training process progresses. The mAP plots, precision and recall metrics increase as the model is able to learn the data.

The following table present statistics calculated on the training split:

Metric name	**Metric value**
Starting box loss	3.86
Ending box loss	0.71
Starting class loss	4.15
Ending class loss	0.52
Starting DFL loss	4.23
Ending DFL loss	1.046

Table 5.4: Object detection model training statistics

The following table present statistics calculated on the validation split:

Metric name	**Metric value**
Starting box loss	3.753
Ending box loss	1.7
Starting class loss	3.18
Ending class loss	1.8
Starting DFL loss	4.15
Ending DFL loss	1.93
Precision	0.8
Recall	0.46
mAP @ 50	0.51

Table 5.5: Object detection model training statistics

Tables 5.4 and 5.5 further reinforce the fact, that the model successfully trained the dataset.

Chapter 6

Experiments

This chapter aims to describe the conducted experiments on the projects' dataset,

6.1 Evaluation on test split

This section aims to evaluate whether the solution is able to work with out-of-distribution data.

6.1.0.1 Evaluation results

During inference evaluation mAP metrics have been calculated, defined in accordance to COCO mAP.

Metric name	Metric value
mAP @ 0.75 - 0.95	0.373
mAP @ 0.5	**0.62**
mAP @ 0.75	0.39

Table 6.1: Table presenting mAP metric results on the test set.

Table 3.5 presents mAP metrics for the test split. Metric that is taken into account in requirements 3.4 is mAP @ 0.5. The metrics' value is close to the requirements' value. The value achieved on the test set is approximately 6% off the target requirement.

6.1.1 Detection examples

The trained model was directly run on the dataset's test set. All preprocessing steps are handled by the Ultralytics library itself [82]. The test set contains 69 labeled images of varying resolutions.

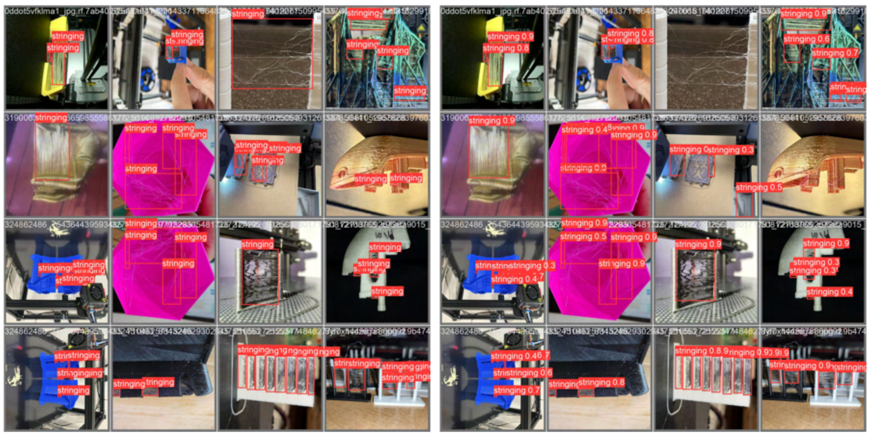


Figure 6.1: Visualization showing some of the ground truth images, along with ground truth bounding boxes and predictions inferred by the model

The trained model was able to make predictions on the test set, as demonstrated on figure 6.1.

However, in some cases the predictions are missed. The first image example with missing prediction is:

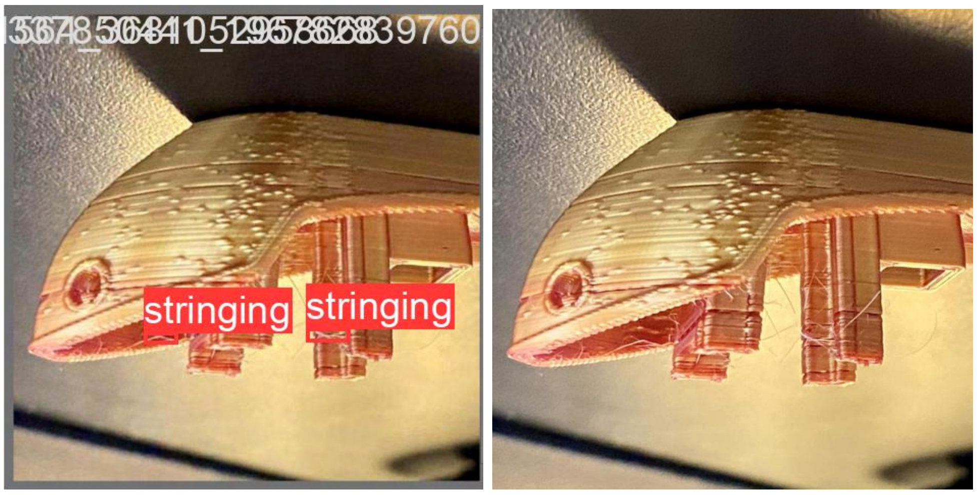


Figure 6.2: Missing detection example. On the left the ground truth image is shown, on the right image with missing predictions is shown.

On the figure 6.2 the model missed predictions, due to fact that the labeled bounding boxes are small and the strands of the dragged material are fine and blend-into the background. The image also seem to use a light that is close in color to the printed object's color. The solution for such a case would be to label whole cluster of dragged material as one defect and use a different light color during image capture, so the material strands do not blend into the background.

The next case is of missing predictions is:

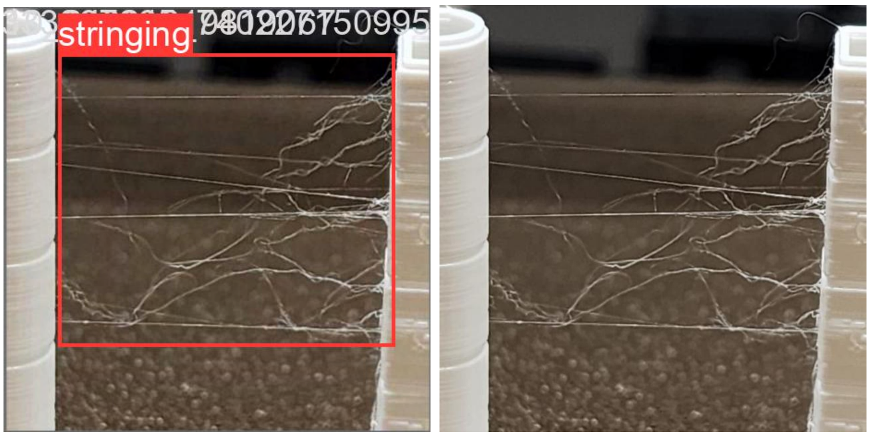


Figure 6.3: Missing detection example. On the left the ground truth image is shown, on the right image with missing predictions is shown.

Figure 6.3 presents missed prediction for a stringing defect. The strands are not dense, and may be too fine and underrepresented in the dataset to be detected.

Another case represents the situation when the labels rendered by ultralytics is inconsistent with the ground truth rendered on Roboflow:

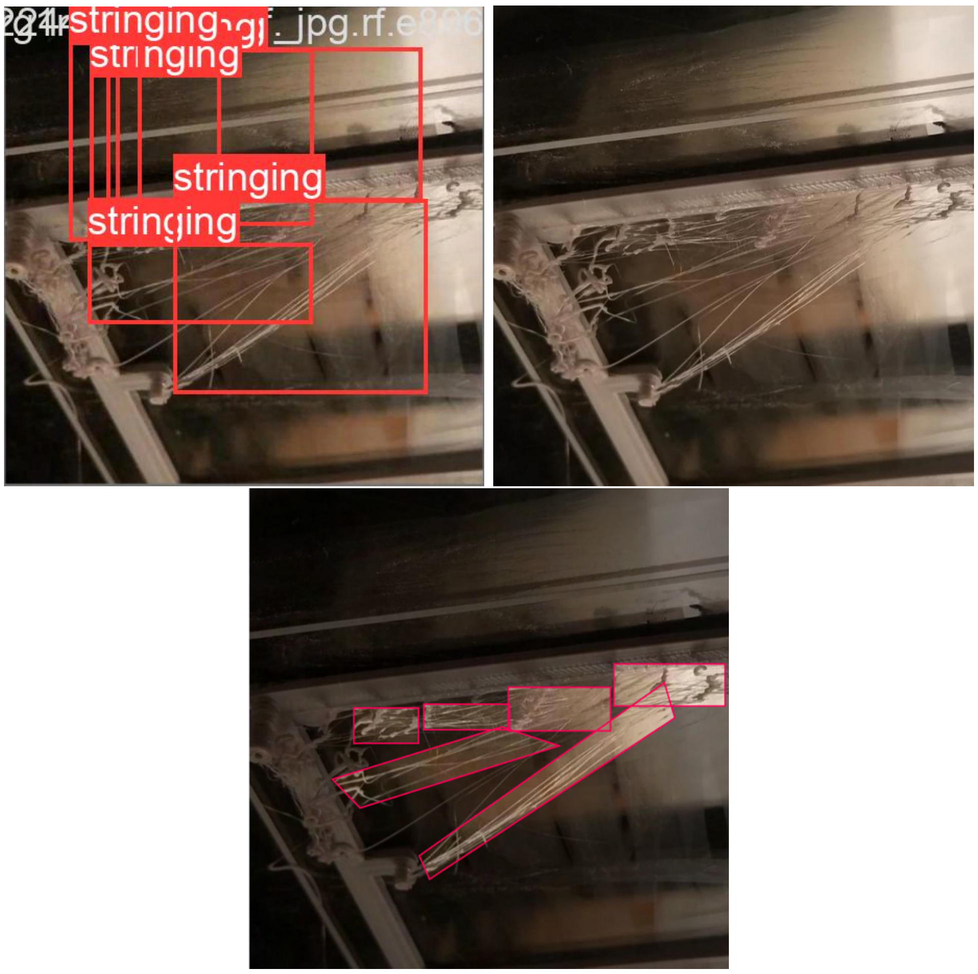


Figure 6.4: A case of inconsistent ground truth representation. On the left - ground truth rendered by ultralytics, middle - missing predictions, right - ground truth rendered by Roboflow

On the figure 6.4 labels between ultralytics and Roboflow are inconsistent, which could lead to a missing prediction. More investigation is needed to verify, where the problem occurs - during export from Roboflow or during label processing in ultralytics.

6.1.2 Severity assessment model

Similarly to defect detection model, the severity assessment (classification) model was run on the test set, containing 258 samples, labeled first by a human with a bootstrap model assistance, then with the model itself, as described in section 5.1.2.

The model reached **60%** accuracy on the test split of the dataset, not satisfying the accuracy requirement, meaning that the model is not accurate enough, to be considered "good". To investigate why that's the case, samples from the test split have been drawn, to see how the bootstrap model has labeled the dataset.

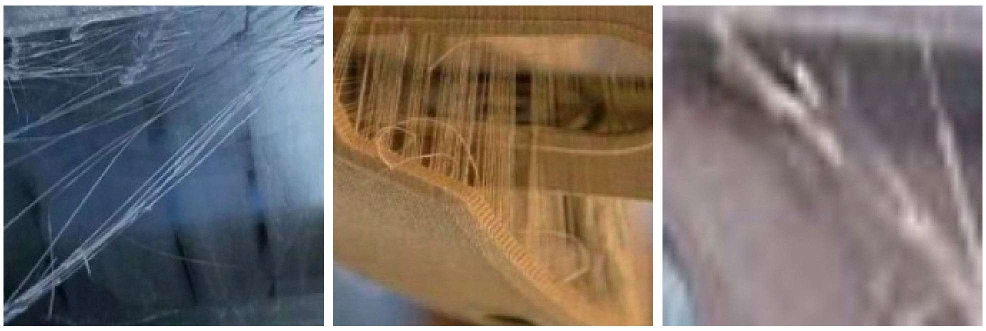


Figure 6.5: Incorrectly labeled images, in which the label the assigned label is "high".

Figure 6.5 represents image examples, that have been mislabeled, but classified as true positive. The definition of "high" severity states, that the stringing defect should've contain blobs in the dragged plastic strands, which are missing on presented samples. The correct classes for the presented images would be, starting from left "low", "medium" and "low".

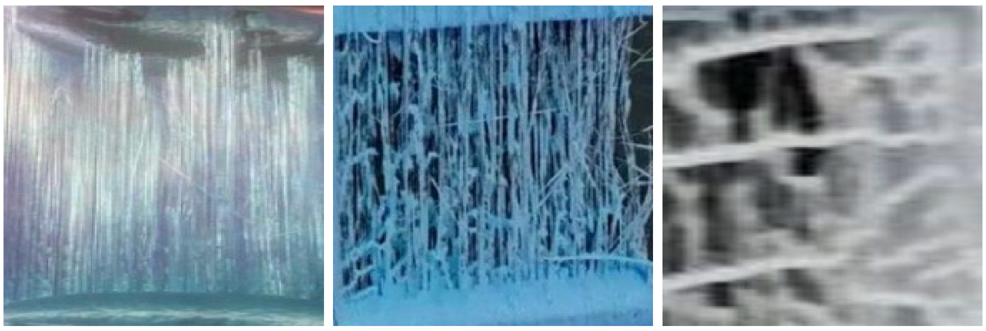


Figure 6.6: Correct classification examples for class "high"

On the other hand figure 6.6 present correctly classified and labeled samples from the test set, showing that many samples are mislabeled by the bootstrap model. In order to improve the accuracy better user supervision over the bootstrap model labelling process has to be employed.

A confusion matrix have been created, to illustrate how the model performed severity classifications:

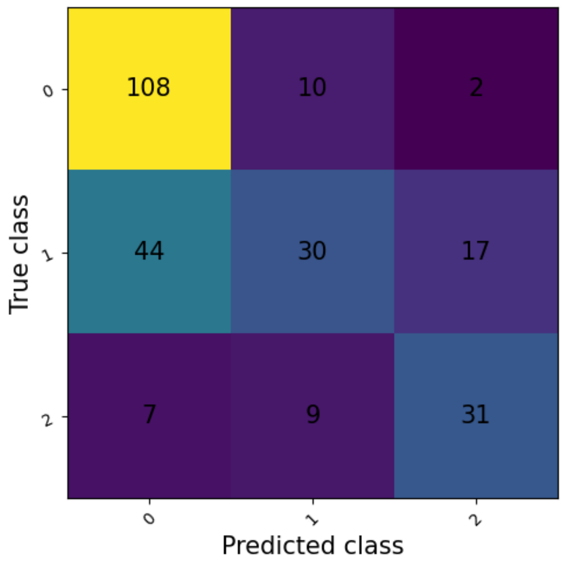


Figure 6.7: Confusion matrix created based on the classifications from the test split. Class assignments: 0 - "low", 1 - "medium", 2 - "high"

The confusion matrix displayed on figure 6.7 contains samples with data augmentation. The test split (and possibly whole dataset) is quite unbalanced, favouring class "low" and class "medium".

The severity assessment model classified most of "low" and "high" samples correctly. The "medium" class seems to be the most difficult to classify. To produce better results, it's likely that the dataset has to be re-balanced, re-labelled and model has to be retrained. Another option would be to remove class "medium" and redistribute the samples between labels "low" and "high".

6.2 Print timelapse test

Models have been tested on a 3D printing timelapse captured via the OctoPrint software [72] in order to check whether the model performs well in a real-life application.

6.2.1 Testbed setup

Several test prints have been produced with retraction setting turned off, in order to provoke the stringing defect to occur.

OctoPrint have been configured to capture a frame on Z-axis movement, after layer change, simulating layer-by-layer frame capture. Device on which OctoPrint will run is previously mentioned Raspberry Pi 4 single board computer with active cooling, running at stock CPU clock frequency.

The machine used for test is the Creality CR-20 FDM 3D printer, with the following print parameters:

1. Nozzle type and size - brass 0.5mm nozzle.
2. Layer height - 0.16mm, yielding
3. Retraction - off.
4. Z-axis hop - off.
5. Infill density and type - 20% and cubic infill pattern.
6. Filament type - Fiberology Gray Easy PLA.
7. Printing temperatures - nozzle: 215°C, printing bed: 50°C.
8. Support generation - off.
9. Print speed - 85 mm/s.

"Stringing test" by Na-temps [83] 3D object have been printed for the test purpose being positioned at the bed center, with a wooden background present. The room, in which printer was situated was filled with natural light, which intensity depended on the time of day. Turning off retraction produces stringing defect severity classified as "high".

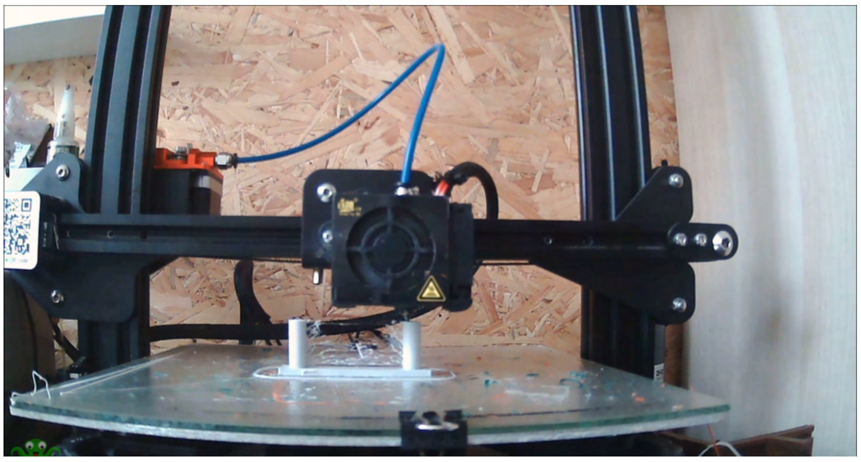


Figure 6.8: Sample screenshot from one of recorded timelapses.

Figure 6.8 presents a timelapse screenshot with the printed object and visible stringing defect. The camera directly points at the printed object, being positioned at the front of the 3D printer.

The printed object consists of a base and two cylinders, which causes the 3D printer to axes perform so-called travel moves, which will produce stringing, if retraction is turned off. The test object is presented on the following render:

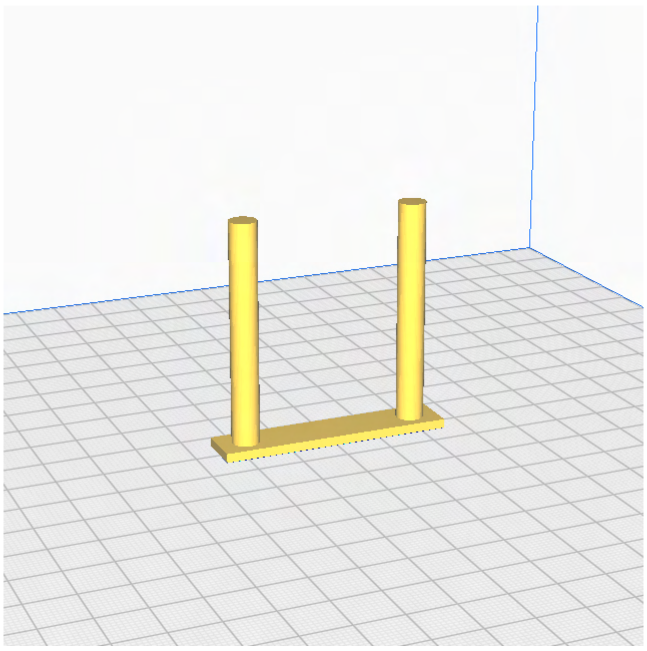


Figure 6.9: Test object render in Ultimaker Cura.

6.2.2 Testing process

A test Python script have been developed that takes a frame from the timelapse. The bounding boxes detected by the defect detection model (if any defect was detected) are then fed into severity assessment model, to ultimately produce a bounding box containing information about what defect have been detected and how severe the defect is. One severity ("high") have been labeled on the image, as the retraction setting produces a consistent defect on the resulting printout. Defect bounding boxes have been labeled as well. Timelapse had 250 samples, beginning 25 frames are not containing the defect, due to object's base being printed first.

6.2.3 Testing results

Defect detection model detected defects in 45 frames out of 225. If a detection was made, the IoU of the detection was high. Severity of the detected defects have been classified correctly in most of the detected cases. Missed detections are counted as if their IoU was 0 and no correct classifications was made.

The results present as following, if missed detections are excluded:

Metric name	**Metric value**
Average IoU	0.74
Average processing time per frame	1817 ms
Severity assessment classification accuracy	75%
Severity assessment precision	0.5
Severity assessment recall	0.36
Detection precision	0.964
Detection recall	0.8
Detection mAP @ 50	0.891
Memory Usage	565MB

Table 6.2: Table presenting timelapse test results, excluding missing detections.

However the results are much worse, if missed detections are included:

Metric name	**Metric value**
Average IoU	0.14
Average processing time per frame	1659 ms
Severity assessment classification accuracy	16%
Severity assessment precision	0.048
Severity assessment recall	0.24
Detection precision	0.422
Detection recall	0.292
Detection mAP @ 50	0.28
Memory Usage	565MB

Table 6.3: Table presenting timelapse test results, including missing detections.

Tables 6.2 and 6.3 show that if a detection is made, then the detection is accurate. However, detections are inconsistent, missing the defect in most of the captured frames.

6.2.3.1 Detection examples

The model seem to be able to make a good detection, scoring high average IoU.

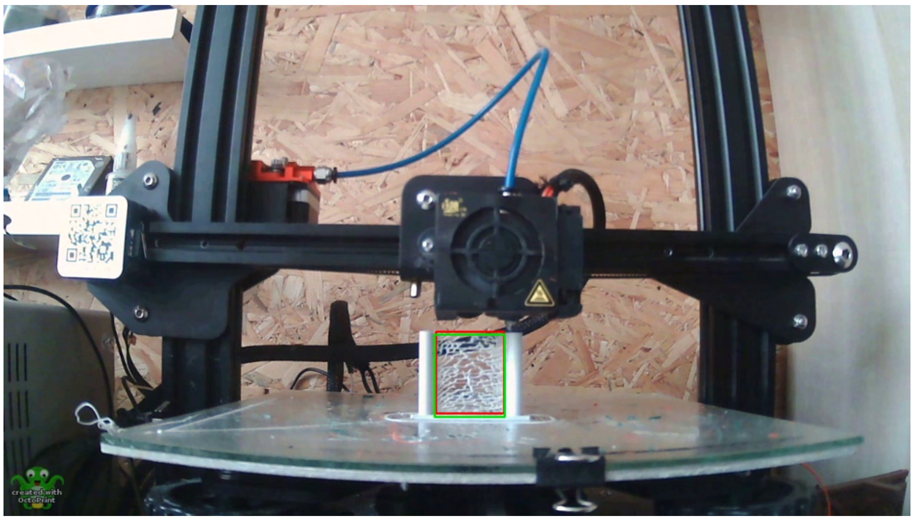


Figure 6.10: Example of a good detection. Green bounding box - the ground truth, red bounding box - defect detection.

On figure 6.10 the model was able to match closely the ground truth, which applies to the rest of detections. However, sometimes the model missed the defect and detected an unrelated object in the scene:

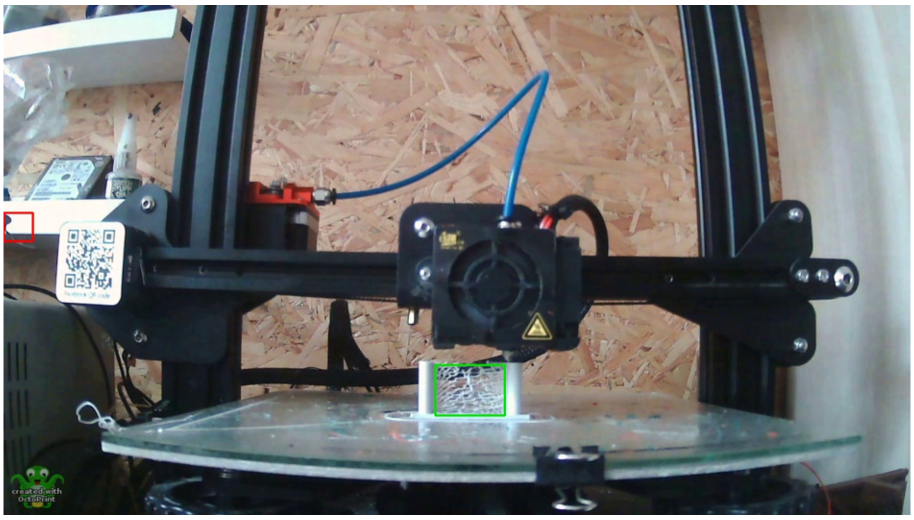


Figure 6.11: Example of a bad detection. Green bounding box - the ground truth, red bounding box - defect detection.

Figure 6.11 presents, that the model has missed the defect on the printed object. What's interesting, such an invalid detection happens only half-way of the printing process.

The model to seem to miss the beginning phases of the defect, but makes detections near the end of print job:

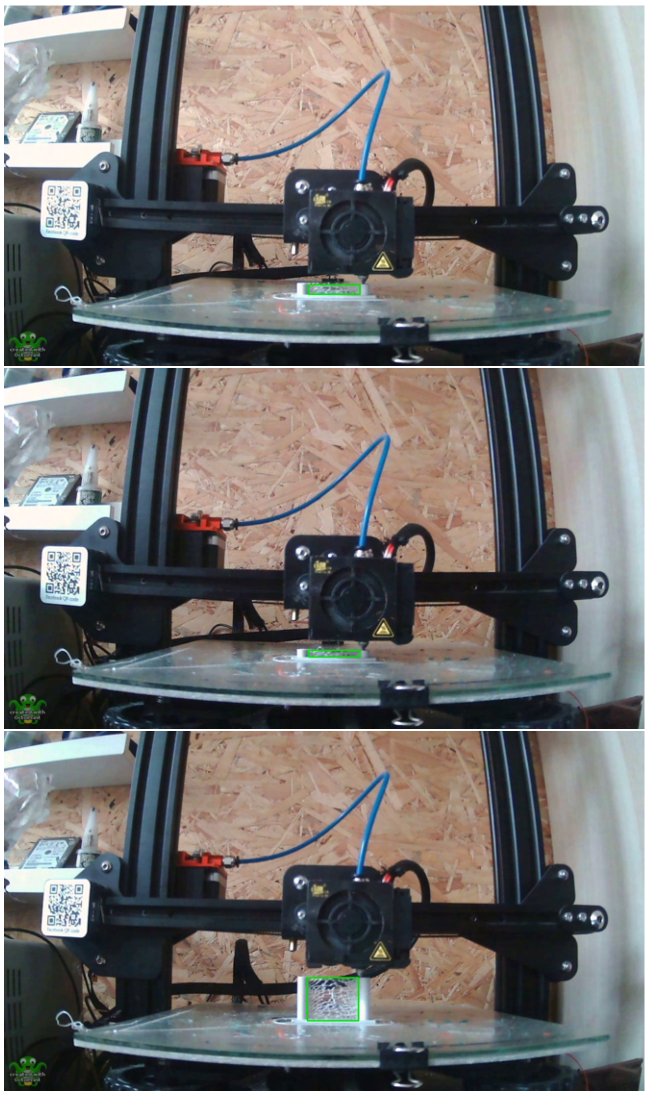


Figure 6.12: Beginning phases of the print job with missing detections. Green box - the ground truth.

Figure 6.12 present how the model missed detections at the beginning of the print job. The reason of the missed detections may be due to the fact, that a webcam has a worse quality sensor, comparing to smartphones and cameras [84]. The issue could be resolved by recording 3D printing process using a variety of webcams, labeling the samples and including them in the dataset. Additionally, most of the samples in the dataset "fill-up" the frame, whereas in the samples from the timelapse don't.

Better camera positioning could help resolve the issue.

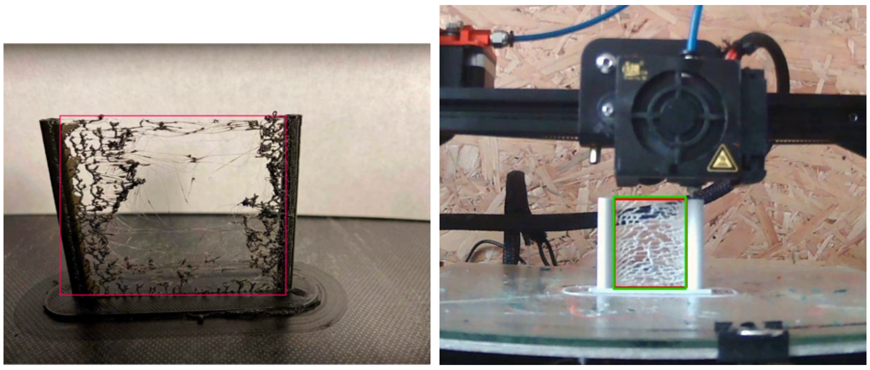


Figure 6.13: Comparison of two samples from the dataset (left) and the timelapse (right).

Dataset sample presented on the figure 6.13 subjectively has a better quality, in comparison with the timesample frame. The image from dataset is sharper, whereas the image captured by the webcam seems blurry.
Furthermore, the created dataset has 1600 images, which in comparison with major object detection datasets like MS COCO [45] (330k images) or PACAL VOC (9963 images) [46] seem small.

Additionally, the presented confusion matrix in 6.7 may explain why the severity assessment model made correct predictions, if a detection was made. Most of "high" labeled samples have been classified correctly during validation, which is the case here. The print settings created defect with "high" severity, which the severity assessment model was able to classify correctly and explain high accuracy, if missed detections are ignored.

Chapter 7

Results and discussion

This chapter aims to discuss the results and requirements fulfillment.

7.1 Requirements

The project was able to fulfill some of the requirements, if tested just on the datasets' test split. However, the results are much worse when the model is tested on a timelapse recording, recorded during printing process.

7.1.1 Performance requirements

Model inference time

The projects' inference time was fast enough to fulfill the layer time requirement. The average time of 1800 milliseconds is low enough, to perform multiple inferences during one layer, if a live capture setup is desired. Such a low processing time leaves a lot of room for potential improvements regarding model selection and dataset size.

Model memory size

The project occupied 565 MB of tested devices' RAM, which is 1/4th of maximum 2GB, set by the requirement, leaving more than enough space for OctoPrint to function on the device.

7.1.2 Functional requirements

Defect detection accuracy

During validation on the test set, the model reached 0.62 mAP @ 0.5, which is approximately 6% off the target requirement. It does not mean that the project is not able to make predictions on the test set, it's just not accurate enough. Lowering this requirement to the level of 0.62 from 0.66 wouldn't influence the results too much.

However, during the test on the timelapse, the project was struggling with making consistent detections and severity predictions, due to poor generalization, towards out-of-distribution data. If a detection was finally made, then the detection was accurate, scoring 0.89 mAP @ 50. More work is needed, if the project is ought to work in a real-life application, but the results show some promise, that it is possible.

Severity assessment accuracy

The project reached 60% accuracy on the test set, which is far below the requirement. The possible causes are poor automatic labeling of the severity dataset, which could be reflected in the training and accuracy plots during training. Proper labelling is needed, to see whether the selected classification model is able to perform the task of severity assessment.

7.1.3 Requirements table

Based on the comments made in the previous section, a table have been composed, to summarize the requirements is fulfilled or not.

Requirement name	**Success?**
Processing time	Yes
Memory usage	Yes
Defect detection accuracy	Yes, on test set, if the requirement is relaxed. No, on a real-life capture.
Defect severity assessment	No

Table 7.1: Requirements summary

2 of 4 requirements are passed. The model is small and fast enough to run on a Raspberry Pi 4, without interrupting the printing process and will be able to make multiple prediction attempts per one layer.
If the accuracy requirement is relaxed, then the detection model is accurate enough

on the test set, however the model is not making consistent detections on real-life capture, meaning that more work is needed on the detection part. The last requirement, the severity assessment requirement isn't fullfilled, due to model being not accurate enough and due possible issues with the severity assessment dataset. More work is needed on the dataset, in order for the solution to be functional, hovewer it may just the matter of relabelling the severity assessment dataset.

Chapter 8

Conclusion

This thesis tried to solve the following problem statement:

Given the current object detection approaches and the dataset availability, how can one create a stringing defect object detection and severity assessment model in the context of FDM 3d printing, to aid 3d print job supervision?

The current state of project failed to solve the problem of stringing defect detection and severity assessment, due to problems with the defect detection on a real-life image capture. The project as it is now makes inconsistent detection (albeit accurate, if a detection is made), which is unacceptable in a real-life application, as well severity assessment model is underperforming to be considered "good" to be useful.

Bibliography

[1] N. Shahrubudin, T. Lee, and R. Ramlan, "An overview on 3d printing technology: Technological, materials, and applications," *Procedia Manufacturing*, vol. 35, pp. 1286–1296, 2019, The 2nd International Conference on Sustainable Materials Processing and Manufacturing, SMPM 2019, 8-10 March 2019, Sun City, South Africa, ISSN: 2351-9789. DOI: https://doi.org/10.1016/j.promfg.2019.06.089. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2351978919308169.

[2] R. Contributors. [Online]. Available: https://reprap.org/wiki/RepRap.

[3] T. Rayna and L. Striukova, "From rapid prototyping to home fabrication: How 3d printing is changing business model innovation," *Technological Forecasting and Social Change*, vol. 102, pp. 214–224, 2016, ISSN: 0040-1625. DOI: https://doi.org/10.1016/j.techfore.2015.07.023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0040162515002425.

[4] Wikipedia contributors, *Fused filament fabrication — Wikipedia, the free encyclopedia*, https://en.wikipedia.org/w/index.php?title=Fused_filament_fabrication&oldid=1206857766, [Online; accessed 27-February-2024], 2024.

[5] All3dp, *3d printing materials – the ultimate guide*, https://all3dp.com/1/3d-printing-materials-guide-3d-printer-material/, [Online; accessed 27-February-2024], 2024.

[6] Wikipedia contributors, *3d printing — Wikipedia, the free encyclopedia*, https://en.wikipedia.org/w/index.php?title=3D_printing&oldid=1203041433, [Online; accessed 27-February-2024], 2024.

[7] Solidator, *Msla - what are the advantages of msla over sla?* 2023. [Online]. Available: https://solidator.com/en/msla-sla-comparison/.

[8] S. 3D, *Print quality troubleshooting guide*, https://www.simplify3d.com/resources/print-quality-troubleshooting/, [Online; accessed 27-February-2024], 2024.

[9] P. 3D, *Everything about nozzles with a different diameter*, https://blog.prusa3d.com/everything-about-nozzles-with-a-different-diameter_8344/, [Online; accessed 27-February-2024], 2024.

[10] All3dp, *3d printing infill: The basics for perfect results*, https://all3dp.com/2/infill-3d-printing-what-it-means-and-how-to-use-it/, [Online; accessed 27-February-2024], 2024.

[11] Marlin, *G-code index*, https://marlinfw.org/meta/gcode/, [Online; accessed 27-February-2024], 2024.

[12] teachingtechyt, *Teaching tech 3d printer calibration*, https://teachingtechyt.github.io/calibration.html, [Online; accessed 28-February-2024], 2024.

[13] *How to calibrate your 3d printer: A step-by-step guide*, 2023. [Online]. Available: https://all3dp.com/2/how-to-calibrate-a-3d-printer-simply-explained/.

[14] J. Ramian, J. Ramian, and D. Dziob, "Thermal deformations of thermoplast during 3d printing: Warping in the case of abs," *Materials*, vol. 14, no. 22, 2021, ISSN: 1996-1944. DOI: 10.3390/ma14227070. [Online]. Available: https://www.mdpi.com/1996-1944/14/22/7070.

[15] BNC3D, *Printing issues related to filament affected by humidity*, https://support.bcn3d.com/knowledge/humid-filament-bcn3d, [Online; accessed 27-February-2024], 2024.

[16] M. K. Mitchell and D. E. Hirt, "Degradation of pla fibers at elevated temperature and humidity," *Polymer Engineering & Science*, vol. 55, no. 7, pp. 1652–1660, 2015. DOI: https://doi.org/10.1002/pen.24003. eprint: https://4spepublications.onlinelibrary.wiley.com/doi/pdf/10.1002/pen.24003. [Online]. Available: https://4spepublications.onlinelibrary.wiley.com/doi/abs/10.1002/pen.24003.

[17] All3dp, *3d printing materials – the ultimate guide*, https://all3dp.com/2/3d-print-stringing-easy-ways-to-prevent-it/, [Online; accessed 27-February-2024], 2024.

[18] All3dp, *Humidity affects 3d printer filament*, https://www.printsolid.com.au/knowledge-base/humidity-affects-3d-printer-filament/, [Online; accessed 27-February-2024], 2024.

[19] T. 3d, *How bad is wet filament really?* https://toms3d.org/2021/11/23/how-bad-is-wet-filament-really/, [Online; accessed 27-February-2024], 2024.

[20] A. 3D, *3d print stringing: Common causes and quick fixes*, https://www.ankermake.com/eu-en/blogs/guides/3d-print-stringing, [Online; accessed 27-February-2024], 2024.

[21] P. 3D, *Layer separation and splitting fdm*, https://help.prusa3d.com/article/layer-separation-and-splitting-fdm_1806, [Online; accessed 28-February-2024], 2024.

[22] P. 3D, *Spaghetti monster*, https://help.prusa3d.com/article/spaghetti-monster_1999, [Online; accessed 28-February-2024], 2024.

[23] [Online]. Available: https://www.creality3dofficial.com/products/ender-3-v3-se-3d-printer.

[24] [Online]. Available: https://www.dst.dk/en/Statistik/emner/miljoe-og-energi/energiforbrug-og-energipriser/energipriser.

[25] Wild-Meteor. [Online]. Available: https://www.reddit.com/r/FixMyPrint/comments/13coi1s/warping_is_getting_out_of_hand_help/.

[26] Regular$_{Bike}$1437. [Online]. Available: https://www.reddit.com/r/FixMyPrint/comments/1aogjfp/bottom_of_print_warping/.

[27] 94YPe. [Online]. Available: https://www.reddit.com/r/FixMyPrint/comments/kv7n90/need_tips_on_reducing_warping_in_pla_prints/.

[28] MonaghanRed. [Online]. Available: https://www.reddit.com/r/FixMyPrint/comments/1b3x1kk/bloody_zits/.

[29] xcjacob30. [Online]. Available: https://www.reddit.com/r/FixMyPrint/comments/i5zien/zits_on_print/.

[30] Angle$_{Smithereen}$. [Online]. Available: https://www.reddit.com/r/FixMyPrint/comments/10omyp7/zits_in_surface_of_print_with_silk_filament/#lightbox.

[31] 23-15-12-06. [Online]. Available: https://www.reddit.com/r/FixMyPrint/comments/17xz9wk/neptune_4_pro_midprint_spaghetti/.

[32] S.-K. H. C. e. a. Kumar R., "Development and comparison of machine-learning algorithms for anomaly detection in 3d printing using vibration data," 2023. DOI: https://doi-org.zorac.aub.aau.dk/10.1007/s40964-023-00472-1.

[33] J. Sendorek, T. Szydlo, M. Windak, and R. Brzoza-Woch, *Dataset for anomalies detection in 3d printing*, 2020. arXiv: 2004.08817 [cs.OH].

[34] K. Paraskevoudis, P. Karayannis, and E. P. Koumoulos, "Real-time 3d printing remote defect detection (stringing) with computer vision and artificial intelligence," *Processes*, vol. 8, no. 11, 2020, ISSN: 2227-9717. DOI: 10.3390/pr8111464. [Online]. Available: https://www.mdpi.com/2227-9717/8/11/1464.

[35] Z.-Z. Y. Y. e. a. Zhang H., "Printing defect detecting by machine vision with convolutional neural network," 2022. DOI: https://doi-org.zorac.aub.aau.dk/10.1007/s40799-022-00577-2.

[36] T.-J. B. R. e. a. Baumgartl H., “A deep learning-based model for defect detection in laser-powder bed fusion using in-situ thermographic monitoring,” 2020.

[37] B. Ye, K.-J. Kim, and E. P. Sacks, “Global and local defect detection for 3d printout surface based on geometric shape comparison,” *Precision Engineering*, vol. 82, pp. 324–337, 2023, ISSN: 0141-6359. DOI: `https://doi.org/10.1016/j.precisioneng.2023.04.005`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S014163592300079X`.

[38] J. Li, W. Quan, L.-K. Shark, and H. Laurence Brooks, “A vision-based monitoring system for quality assessment of fused filament fabrication (fff) 3d printing,” in *Proceedings of the 2022 5th International Conference on Image and Graphics Processing*, ser. ICIGP ’22, Beijing, China: Association for Computing Machinery, 2022, 242–250, ISBN: 9781450395465. DOI: `10.1145/3512388.3512424`. [Online]. Available: `https://doi-org.zorac.aub.aau.dk/10.1145/3512388.3512424`.

[39] “Generalisable 3d printing error detection and correction via multi-head neural networks,” *Nature*, DOI: `https://doi.org/10.1038/s41467-022-31985-y`. [Online]. Available: `https://www.nature.com/articles/s41467-022-31985-y`.

[40] K. Okarma, J. Fastowicz, and M. Tecław, “Application of structural similarity based metrics for quality assessment of 3d prints,” in *Computer Vision and Graphics*, L. J. Chmielewski, A. Datta, R. Kozera, and K. Wojciechowski, Eds., Cham: Springer International Publishing, 2016, pp. 244–252, ISBN: 978-3-319-46418-3.

[41] 3D Que, *Quinlyvision ai failure detection*, [Online; accessed 01.03.2024], 2024. [Online]. Available: `https://docs.3dque.com/docs/quick-start-tutorials/quinly-vision`.

[42] Raspberry Pi Foundation, *Raspberry pi 4 tech specs*, [Online; accessed 01.03.2024], 2024. [Online]. Available: `https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/`.

[43] PrintPal, *Printpal.io printwatch documentation*, [Online; accessed 01.03.2024], 2024. [Online]. Available: `https://docs.printpal.io/`.

[44] R. Kaur and S. Singh, “A comprehensive review of object detection with deep learning,” *Digital Signal Processing*, vol. 132, p. 103 812, 2023, ISSN: 1051-2004. DOI: `https://doi.org/10.1016/j.dsp.2022.103812`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S1051200422004298`.

[45] T. Lin, M. Maire, S. J. Belongie, *et al.*, "Microsoft COCO: common objects in context," *CoRR*, vol. abs/1405.0312, 2014. arXiv: `1405.0312`. [Online]. Available: `http://arxiv.org/abs/1405.0312`.

[46] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, *The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results*, http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[47] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255. DOI: `10.1109/CVPR.2009.5206848`.

[48] M. Tkachenko, M. Malyuk, A. Holmanyuk, and N. Liubimov, *Label Studio: Data labeling software*, Open source software available from https://github.com/heartexlabs/label-studio, 2020-2022. [Online]. Available: `https://github.com/heartexlabs/label-studio`.

[49] Roboflow, *Top computer vision models*, [Online; accessed 04.03.2024], 2024. [Online]. Available: `https://roboflow.com/models/classification/`.

[50] [Online]. Available: `https://www.ibm.com/docs/en/mas-cd/maximo-vi/continuous-delivery?topic=sets-data-set-considerations`.

[51] *Computer vision annotation formats.* [Online]. Available: `https://roboflow.com/formats`.

[52] Wikipedia contributors, *Deep learning — Wikipedia, the free encyclopedia*, [Online; accessed 28-February-2024], 2024. [Online]. Available: `https://en.wikipedia.org/w/index.php?title=Deep_learning&oldid=1209134089`.

[53] Distill contributors, *Feature visualisation*, [Online; accessed 29-February-2024], 2017. [Online]. Available: `https://distill.pub/2017/feature-visualization/`.

[54] R. Szeliski, *Computer Vision: Algorithms and Applications*, 1st. Berlin, Heidelberg: Springer-Verlag, 2010, ISBN: 1848829345.

[55] J. Brownlee, *A gentle introduction to pooling layers for convolutional neural networks*, 2019. [Online]. Available: `https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/`.

[56] J. Beal, E. Kim, E. Tzeng, D. H. Park, A. Zhai, and D. Kislyuk, "Toward transformer-based object detection," *CoRR*, vol. abs/2012.09958, 2020. arXiv: `2012.09958`. [Online]. Available: `https://arxiv.org/abs/2012.09958`.

[57] T. Cheng, L. Song, Y. Ge, W. Liu, X. Wang, and Y. Shan, *Yolo-world: Real-time open-vocabulary object detection*, 2024. arXiv: `2401.17270 [cs.CV]`.

[58] A. Zareian, K. D. Rosa, D. H. Hu, and S.-F. Chang, *Open-vocabulary object detection using captions*, 2021. arXiv: `2011.10678 [cs.CV]`.

[59] A. Howard, M. Sandler, G. Chu, *et al.*, "Searching for mobilenetv3," *CoRR*, vol. abs/1905.02244, 2019. arXiv: `1905.02244`. [Online]. Available: `http://arxiv.org/abs/1905.02244`.

[60] A. Paszke, S. Gross, F. Massa, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

[61] A. G. Howard, M. Zhu, B. Chen, *et al.*, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, vol. abs/1704.04861, 2017. arXiv: `1704.04861`. [Online]. Available: `http://arxiv.org/abs/1704.04861`.

[62] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation," *CoRR*, vol. abs/1801.04381, 2018. arXiv: `1801.04381`. [Online]. Available: `http://arxiv.org/abs/1801.04381`.

[63] en-US, May 2024. [Online]. Available: `https://www.gsmarena.com/google_pixel-8346.php`.

[64] J. Torres and J. T. J. Austen, *Yolov8 architecture: A deep dive into its architecture*, 2024. [Online]. Available: `https://yolov8.org/yolov8-architecture/`.

[65] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *CoRR*, vol. abs/1804.02767, 2018. arXiv: `1804.02767`. [Online]. Available: `http://arxiv.org/abs/1804.02767`.

[66] A. Bochkovskiy, C. Wang, and H. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," *CoRR*, vol. abs/2004.10934, 2020. arXiv: `2004.10934`. [Online]. Available: `https://arxiv.org/abs/2004.10934`.

[67] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia, "Path aggregation network for instance segmentation," *CoRR*, vol. abs/1803.01534, 2018. arXiv: `1803.01534`. [Online]. Available: `http://arxiv.org/abs/1803.01534`.

[68] C. Tools, *3dbenchy - the jolly 3d printing torture-test by creativetools.se.* [Online]. Available: `https://www.thingiverse.com/thing:763622`.

[69] majda107, **mini* all in one 3d printer test.* [Online]. Available: `https://www.thingiverse.com/thing:763622`.

[70] iDig3Dprinting, *Xyz 20mm calibration cube.* [Online]. Available: `https://www.thingiverse.com/thing:1278865`.

[71] [Online]. Available: `https://www.raspberrypi.com/documentation/computers/raspberry-pi.html`.

[72] G. Häußge, *Octoprint.org.* [Online]. Available: https://octoprint.org/.

[73] A. Marcantoni, *Is there a way to get gpu acceleration on raspberry pi 4 for deep learning?* [Online]. Available: https://stackoverflow.com/questions/62202550/is-there-a-way-to-get-gpu-acceleration-on-raspberry-pi-4-for-deep-learning.

[74] AtCo, *3d printing error dataset*, https://universe.roboflow.com/atco/3d-printing-error, Open Source Dataset, visited on 2024-03-06, 2023. [Online]. Available: https://universe.roboflow.com/atco/3d-printing-error.

[75] rmuti, *Defect*$_3$*d*$_p$*rintdataset*, https://universe.roboflow.com/rmuti-lpzwc/defect_3d_print, Open Source Dataset, visited on 2024-05-21, 2024. [Online]. Available: https://universe.roboflow.com/rmuti-lpzwc/defect_3d_print.

[76] HCMUT, *3d printing defects dataset*, https://universe.roboflow.com/hcmut-yxyhm/3d-printing-defects, Open Source Dataset, visited on 2024-05-21, 2023. [Online]. Available: https://universe.roboflow.com/hcmut-yxyhm/3d-printing-defects.

[77] MoschA, *3dprintfail*$_v$*2dataset*, https://universe.roboflow.com/moscha-hpkr8/3dprintfail_v2, Open Source Dataset, visited on 2024-05-21, 2023. [Online]. Available: https://universe.roboflow.com/moscha-hpkr8/3dprintfail_v2.

[78] G. Jocher, A. Chaurasia, and J. Qiu, *Ultralytics yolo*, Jan. 2023. [Online]. Available: https://ultralytics.com.

[79] A. Buslaev, V. I. Iglovikov, E. Khvedchenya, A. Parinov, M. Druzhinin, and A. A. Kalinin, "Albumentations: Fast and flexible image augmentations," *Information*, vol. 11, no. 2, 2020, ISSN: 2078-2489. DOI: 10.3390/info11020125. [Online]. Available: https://www.mdpi.com/2078-2489/11/2/125.

[80] N. S. Detlefsen, J. Borovec, J. Schock, *et al.*, "Torchmetrics - measuring reproducibility in pytorch," *Journal of Open Source Software*, vol. 7, no. 70, p. 4101, 2022. DOI: 10.21105/joss.04101. [Online]. Available: https://doi.org/10.21105/joss.04101.

[81] J. Brownlee, *How to use learning curves to diagnose machine learning model performance*, 2019. [Online]. Available: https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/.

[82] Ultralytics, *How to do image preprocessing by opencv in yolov8? · issue 2580 · ultralytics/ultralytics.* [Online]. Available: https://github.com/ultralytics/ultralytics/issues/2580.

[83] [Online]. Available: https://www.printables.com/pl/model/841839-stringing-test/files.

[84] A. Davies, *Smartphone vs digital camera: Which is better?* 2024. [Online]. Available: `https://amateurphotographer.com/buying-advice/smartphone-vs-digital-camera-which-is-better/`.