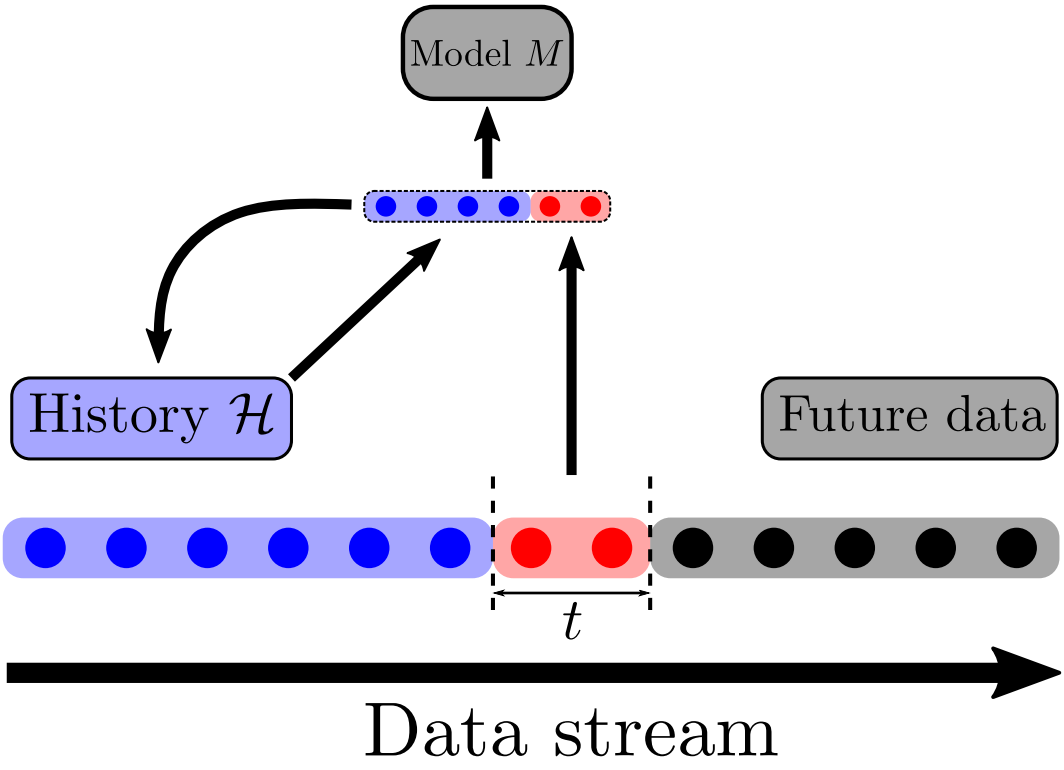
CHALMERS
UNIVERSITY OF TECHNOLOGY



Domain Validation and Continuous Improvement of Deep Learning Models

Applied to Smart Manufacturing Defect Detection Tasks

Master's thesis in Complex Adaptive Systems

Arash Darakhsh

Department of Physics
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2023

Master's thesis 2023

Domain Validation and Continuous Improvement of Deep Learning Models

Applied to Smart Manufacturing Defect Detection Tasks

Arash Darakhsh

CHALMERS

Department of Physics
Chalmers University of Technology
Gothenburg, Sweden 2023

Domain Validation and Continuous Improvement of Deep Learning Models Applied to Smart Manufacturing Defect Detection Tasks
Arash Darakhsh

Supervisors: Philipp Westphal & Edvard Svenman, GKN Aerospace
Examiner: Kristian Gustafsson, Department of Physics

Master's Thesis 2023
Department of Physics
Chalmers University of Technology
SE-412 96 Gothenburg

Cover: Schematic of proactive training, a method that continuously improves deep machine learning models using a combination of new and historical data.

Typeset in LaTeX
Gothenburg, Sweden 2023

Domain Validation and Continuous Improvement of Deep Learning Models
Applied to Smart Manufacturing Defect Detection Tasks
Arash Darakhsh
Department of Physics, Complex Adaptive Systems
Chalmers University of Technology

Abstract

With more industries shifting over to an industry 4.0 setting, many product manufacturing processes and related tasks are becoming more and more automated. Detection of defects in aerospace components, which is extremely crucial for aviation safety, is normally performed by trained operators. The operators apply techniques of non-destructive testing to acquire inspection data, which they then evaluate. In many of these inspection tasks, digital images are a fundamental part of the inspection process, as in X-ray inspection of welds. For these types, efforts have been made over the last years to increase the level of automation, thus decreasing the burden on the operators. Deep neural network models aiding or even completely replacing operator judgement are a main subject of interest. Before these models can be deployed into industry they need to be thoroughly tested and validated.

In this thesis, a U-Net model is used as an example for two frequently reoccurring problems that appear in industrial smart manufacturing. First, we investigate whether data from one inspection task can be used to predict the relationship between model performance and data set size for intra-domain data sets. In this sense, one can use information from one data set to estimate the needed sample size required to train a model for novel inspection tasks. Thereby, more precise estimations about the necessary amount of data for training models in development environments that also perform well when confronted with industrial data streams once deployed should be possible. The second problem considers methods for continuously improving deployed models as more data is made available. Here full retraining, online learning and proactive training, a recent retraining method described in Prapas et al, Datenbank-Spektrum 21, pp. 203-212 (2021) [1], are compared. The best strategy for continuously improving and maintaining industrial deep learning models is researched.

The obtained results suggest that the predictive capabilities from one data set do not carry over well to other data sets, even though belonging to the same domain of problems. The mean average F_1 error between the two considered data sets was measured to 0.11. Despite this, alternative estimation methods, solely using curve fitting, showed promising results. The findings from the second task favoured standard methods of scheduled retraining for the considered U-Net, although more extensive simulations are needed before ruling out the alternative methods considered in the report. In conclusion, the acquired results in this thesis support the progress of automation of industrial inspections tasks by providing methods and guidelines for industrial deep learning models.

Acknowledgements

First and foremost I would like to give a huge thanks to Edvard Svenman and Philipp Westphal for choosing me for this thesis and making it possible in the first place.

Danke Philipp for giving me the experience of working in a place where I could finally apply my knowledge and education. The consistent help and supervision has been invaluable in the making of the thesis.

Thank you Edvard for making me feel welcome at GKN and accompanying me during lunch with the rest of the GKN crew. I enjoyed discussing topics related to both the aerospace industry and diverse topics.

I would also like to show my great appreciation to Nikhil Masinete, one of the most generous and respectful people I have meet in my life. I am grateful for your help, input and time on certain decisions. I had great pleasure discussing various machine learning related topics and exchanging ideas for our respective work.

Also a big thanks to the other members of GKN that I interacted with who were nothing but nice and welcoming.

Thank you Kristian Gustafsson for accepting to be my examiner. Your guidance, help and quick response all throughout the thesis project, allowed me to focus and direct my attention to the actual thesis work instead of worrying about the specificities of the master thesis.

Finally, I would like to thank the university of Chalmers, their faculty and especially my professors for an incredible 5 year educational journey. Additionally I would like to show my gratitude to my closest classmates Jack Sandberg, Erik βrusewitz and Erik Bivrin for their company as making this journey on my own would have been impossible.

Arash Darakhsh, Gothenburg, 2023-06-19

Contents

1

Introduction

Industry 4.0, a new era of industry, has emerged from the recent technological advancements where improvements in machine learning (ML) has been a big enabler for its adoption. Industry 4.0 is the digitization of the manufacturing sector mainly driven by data, AI, sensors and robotics [2]. *Smart manufacturing*, a subset of industry 4.0, refers to the digitization of industry where big data, analytics and machine learning in combination with the Internet of Things are adopted in several applications related to product manufacturing [3]. Machine learning models have shown exceptional results in a broad set of smart manufacturing tasks such as inspection [4][5][6], quality control [7][8], anomaly detection [9][10] and production monitoring [11]. The benefits of moving manufacturing plants to an industry 4.0 setting, have shown great improvements in cost, labour, product quality and consumer customization [12] [13] . These benefits come with several problems and obstacles. To introduce ML models into the manufacturing pipeline, requires extensive testing for safety, performance and maintainability to be confident that its deployment yields benefits. These aforementioned obstacles have shown to be hard to overcome in practice since a majority of ML projects fail to deploy into industry in the first place due to bias in data and developers lack of understanding of algorithms [14][15].

The manufacturing of aerospace components have in recent years switched to more sustainable manufacturing methods. Instead of scrapping entire pieces and rebuilding them from scratch, additive manufacturing (AM), a process where subsequential layers of material are added to aerospace components using lasers and weld tools, has seen more use. Additive manufacturing methods make it possible to repair partially destroyed components. One of the leading companies in the field of AM is GKN Aerospace Sweden which is an aerospace supplier located in Trollhättan focused on sustainable and safe aviation [16]. A significant portion of the inspection process for detecting defects in manufactured aerospace components produces image data that is manually inspected by an operator. Defects in components can vary broadly and can be categorized into groups such as pores in welds, scratches, cracks, dents and buckling. Even detecting critical wear in the used tools is of interest, since this can lead to defects. All these defects have different levels of severity where some defects can immediately fail a component for production while other, less severe defects, can be repaired. The inspection process for detecting defects is both time consuming and prone to human error. Thus there is a great interest in implementing models and tools that either assist the operator in finding defects or that completely automate the defect detection task.

One of the most critical requirements for training robust and high-performing machine learning models is a large data set to capture information and statistical properties about the desired data set. Approximating how much data is needed a priori to training an ML model on a novel task, to reach a specified level of performance, is difficult. Areas in aerospace manufacturing are usually limited to small amounts of data samples dictated by production rates. Usually, to create larger data sets, data is joined together from different institutions, but this comes with its own set of problems such as meta data friction, scanning methods and skews in variable ranges [17]. Joining together data sets from different institutions is especially difficult for aerospace manufacturers where most of the data are kept classified since these companies potentially have ties to military applications and thus are subject to special regulations, export laws, company secrets and customer privacy. This usually leads to the data being collected entirely internal to the facilities. Another difficulty with data availability in the manufacturing field, comes from the fact that manufacturing is a continuously forward moving process of manufacturing and shipping components. This makes it difficult to retroactively scan already shipped components to gather image data for model training if no proper database infrastructure is set up.

The life cycle of ML models does not necessarily end once it has finished training and been deployed into use. In many of the assistive tools found in manufacturing, the ML models are deployed with very limited amount of training data as there is no substantial data set from the beginning. This leads to underwhelming models. As more data becomes available through production, one needs to retrain the model with more data to improve it. In many of the applications of smart manufacturing, the assumption that the data maintains an approximately constant distribution with time is inaccurate. Several factors in the manufacturing pipeline can be attributed to changes in the model's input data distribution. Changes in material supplier, instrument calibrations and deterioration in measurement tools all give rise to changes in the distribution. These changes are commonly called concept drift and to maintain an ML model that performs well after its deployment, there is a need to retrain it once the characteristics of the input data have sufficiently shifted from the original training data. An extensive amount of research has been done in detecting concept drift in data. Methods for monitoring input data, output data and analyzing how changes in the ML pipeline affect the downstream processes are mainly used for detecting concept drift. However, little research has been done on the topic of continually updating machine learning models. Questions such as how to continually improve ML models become relevant in settings where concept drift is present and has been detected.

1.1 Aim of thesis

This thesis work is split up into two main parts. The first part investigates approximation methods for estimating a relationship between the size of training data and model performance to see how these generalizations transfer to data sets for intra-domain tasks. For the second part of the thesis, a framework for deployed models is constructed to continuously improve the model in production. The framework

consists of a retraining system.

1.1.1 Task 1 - Domain validation

Task 1 can be summarized as a *domain validation* task. In this task we investigate if it is possible to estimate how much data is required to gather to train a model whose test performance will be representative of an industrial setting. In addition, the results will be validated to a similar data set within the same problem domain. It will always be true that more data will yield better results, however, for certain inspection tasks where data is limited, a rough estimate of a representative data size is valuable, since gathering data is costly and time consuming. From this, we derive the following problem formulation:
Under which conditions is it possible to use the training statistics derived from a data set A to predict the performance of a model trained on different data set B that belongs to the same domain of problem (intra-domain)?

1.1.2 Task 2 - Continuous improvement

Task 2 can be summarized as a *continuous improvement* task. This part of the thesis is centered around investigating how an ML model, that has been deployed in an industrial setting, can be continuously improved using new data. The continuous improvement protocol is implemented as a retraining algorithm that determines how to update the model when enough new data has become available. The article by Prapas et al. Ref. [1] proposes a new method for updating deployed models. Their method will be used as a reference for the retraining algorithm and built upon with modified sampling methods. Partial results are compared to the ones presented in [1].

1.2 Scope and limitation

To achieve the aim of the thesis project within the set 20 work weeks, certain limitations are introduced to keep the scope of the project focused on solving task 1 and task 2.

1. The main architectural shape of the neural network will be kept constant and no investigation will be made in experimenting how well different architectures perform on this specific inspection task.

2. The data used for training the network is obtained from measurements done in the production floor. The digital images are then pre-processed with proprietary image enhancement filters and then uploaded to an internal server. The work will be limited to not handling anything related to gathering the data and instead an already created data archive will be used.

1.3 Thesis outline

The remaining parts of the thesis is structured as follows. Chapter 2 introduces the reader to a theoretical background that is sufficient to understand the problem area. Furthermore, the chapter consists of a presentation of the fundamentals of the smart manufacturing process. Chapter 3 describes the methodology for setting up the experiments for solving task 1 and task 2. Chapter 4 presents the obtained results from the experiments and numerical simulations. Finally, chapter 5 discusses the results, potential improvements, future work and lastly concludes the thesis.

2

Theory

This chapter introduces the reader to the relevant theory that is required to understand the contents of the thesis. A large portion of fundamental knowledge of ML will be omitted as it is assumed that the reader has a baseline knowledge of elementary statistics and basic ML concepts. Readers unfamiliar with machine learning are advised to read up on machine learning fundamentals given in [18] and [19]. Here, focus is instead directed towards introducing the reader to the industrial manufacturing setting.

Machine learning can be categorized into supervised learning, unsupervised learning, semi-supervised learning and reinforcement learning. This thesis is restricted to the field of supervised learning only. Thus some concepts presented in this chapter are not defined nor can be generalized to the other fields of ML.

Section 2.1 familiarizes the reader to the product manufacturing process and development of new industrial technology. Section 2.2 describes how data is partitioned into different sets used for the training and evaluation of ML models. Section 2.3 describes sources of error and how these relate to model complexity and data sizes. Section 2.4 introduces methods of estimating data sizes for training ML models. Section 2.5 provides a detailed description of the architecture of the U-Net model used in this thesis. Section 2.6 defines standard performance metrics in ML and how these are generalized to the domain of defect detection. Lastly, in section 2.7 a recent retraining method, called *proactive training*, is summarized from the article by Prapas et al. Ref. [1].

2.1 Product manufacturing and inspection

This section introduces the reader to important concepts in product manufacturing. Here, the setting for industrial smart manufacturing is described to give the reader a perspective from where the problems, such as those mentioned in Section 1.1, may appear in industry.

Products that are manufactured can be subject to several inspections throughout the manufacturing process. This inspection, can be to check material properties or the success of an assembly method [20]. The shift in product manufacturing caused by the movement to an industry 4.0 standard has made more tasks automated or assisted by ML models. This transition has occurred during recent years with the

rapid advancement of deep learning models. This has lead to a lot of industries facing issues with data scarcity since no prior effort have been done into building infrastructure for archiving historical data. The process of implementing and deploying an ML model into the manufacturing pipeline can be broken down into several steps. Development timeline models can be used to guide development of technology. In the early stage of the development timeline is where the initial idea is presented to replace, add or modify a part to the current production pipeline. Already at this stage, questions such as development time, development cost, maintenance cost, return of investment and efficacy need to be considered before the development can start. For ML models, it is also especially important to know if machine learning is even suitable to solve that task to begin with. If an ML model is deemed as a reasonable solution, a follow-up question becomes how big of a data sample is required to train a model representative enough that it will not underperform once deployed in an industry-setting. Prior to the deployment step, it is necessary to perform an evaluation and validation of the model's performance. For inspection tasks such as defect detection, it is of higher priority to detect larger defects that can be fatal to the component's life-cycle. Therefore, in addition to standard performance metrics such as accuracy, error-rate and F1-score, probability of detection (POD) curves are necessary to establish a relation between the size of a defect and the model's probability to detect it.

2.2 Data splits and training

Machine learning models, whether it be classical statistical models such as support vector machines or deep neural networks, are parameterized using weights that are determined numerically during their training process. Here we describe different methods of splitting data in supervised learning.

The available data D is usually split into three subsets; training D_{tr}, validation D_{val} and test set D_{te}. This split method is usually referred to as holdout validation [21]. Some authors sometimes refer to the validation set as the development set but in this thesis the term validation set is strictly used. The training subset is used to update the model's weights directly through a learning algorithm. For deep neural networks, the learning algorithm is Backpropagation [22], while for less complex ML models such as linear regression, matrix inversion is sufficient to obtain the model weights. Using the training set to evaluate the performance of the network introduces a bias since the model is optimized with respect to the data set. Thus an evaluation of the training set will be a biased estimation of the model's generalization strength on unseen data samples.

The validation set is used to optimize the network's hyperparameters. Hyperparameters include choices such as the architecture of the network (number of layers and size of each layer), batch size of the data, the parameters of the optimizer and any parameter that is not part of the network's weights. Similarly, evaluating the model on the validation set introduces a bias since the entire network's hyperparameters have been optimized with respect to the validation data.

The test set is set aside during the entire training process and is used to evaluate the model's final performance on an unbiased data set.

It is worth noting that the different data samples can move from the test set to the validation set and similarly from the validation set to the training set without introducing bias issues. For example, a model can predict on a data point to test its performance and then use that data point to update the model's weights. In settings where data is continually generated, for example, deployed machine learning models that predict on incoming data or domains where data is composed of a continual stream, prequential evaluation can be used to continually evaluate the model's performance on more recent data. In prequential evaluation, every training sample is first used to test the model and then train on it [23]. Using this evaluation method, one can establish a cumulative or running average of the model performance instead of just evaluating it on a held-out static test set from the training process.

2.3 Types of model error

The theory presented in this section is sourced from chapter 5 of *Deep learning* by Goodfellow et al. [19].

The main goal of an ML task is to learn and generalize the relevant data for the respective task. The data can be anything from digital images, numerical measurements, time series, audio recordings to text documents. The underlying process from where the data is generated from is called the *data generating process*, and its distribution is normally represented by p_{data}.

ML models achieve high levels of generalization by seeking to maximize the accuracy of the model or correspondingly minimize its error. Here we analyze the concepts using error but the corresponding interpretation exists for accuracy or general performance metrics. The error can be categorized and broken down into different terms that can be related to the properties of the data and the model. As mentioned in the previous section, three important data sets (train, validation and test) are used in holdout validation from where one can define train error e_{tr}, validation error e_{val} and test error e_{te} as the model's error on the respective data set. For a trained network these errors are typically related as

$$e_{te} \geq e_{val} \geq e_{tr}. \tag{2.1}$$

Since the model weights are optimized with respect to the train set, e_{tr} will greatly underestimate the model's true error. Similarly, the hyperparameters are optimized with respect to the validation set so e_{val} will slightly underestimate the true error. For this reason, the model's generalization error is approximated by the error on the test set e_{te} to avoid any bias.

The train and test error will vary depending on the model complexity. Model complexity is a vaguely defined metric that measures the size of the *hypothesis space*, which is the vector space of all the possible solutions that the ML algorithm can converge to. For example, the *Vapnik-Chervonenkis dimension* (VC-dimension) is a

complexity measure used for binary classification tasks [24]. However, this metric is not usually used to quantify the complexity of neural networks despite there existing bounds for very specific network configurations [25]. For deep neural networks, the more common complexity measure, and the one prominently found in literature, is the total number of trainable parameters in the network. One considerable downside of this complexity metric is that the parameter count fails to include the activation functions and the size of the network's individual layers. In general, models with too low of a complexity than what is required to fully capture the properties of the data tend to showcase high training errors e_{tr}. This is because the hypothesis space is too simple for the model to learn any complicated representation of the data. This concept is called *underfitting* and an example is trying to fit a first degree polynomial to data generated by a second first-degree polynomial. Correspondingly, models trained with a very large complexity tend to instead have very low train error e_{tr} since the large hypothesis space enables the model to learn overly complex representations of the training data. However, this often leads to the unwanted effect of *overfitting* where the difference between train and test error

$$e_{gap} = e_{te} - e_{tr} \geq 0 \tag{2.2}$$

becomes very large. Thus one of the many challenges with training an ML model is finding the optimal model complexity for the given task to not showcase any underfitting or overfitting. The optimal model complexity will also depend on the size of the available training data. In the extreme case of small sizes of training data, the optimal complexity will be relatively small since it is hard to learn complex representations from a low number of data points due to data uncertainty and high noise. As the size of the training set approaches infinity the optimal complexity will plateau and stabilize around a value that is sufficient for the model to learn the representations from the data distribution p_{data}.

Finding upper bounds for the error difference e_{gap} is of interest. Generally one can define an upper bound M to the error gap e_{gap} such that

$$e_{gap} \leq M, \tag{2.3}$$

where M increases with higher model complexity and shrinks with an increasing number of training instances [26]–[28]. However, these bounds are rarely used in practice due to the fact that it is hard to define model complexity for deep neural networks and the bounds are not generalizable to many deep learning tasks. This makes it difficult to make theoretical estimates of model error on novel tasks and one instead has to resort to numerical simulations and data estimation methods.

2.4 Data estimation methods

Since the field of machine learning is very diverse and covers many different problems, topics and domains, the task of estimating how much data is needed prior to training a model, to achieve sufficiently good performance, is difficult. The usual method for getting precise estimates of samples sizes, is to look at prior research that has

already performed a similar train-test pipeline and use their recorded values as a starting point. However, for a novel tasks this is not possible and one needs to resort to alternative estimation methods. Methods that estimate the size of the required data are called *sample size determination methods* (SSDMs) and are categorized into model-based and curve-fitting based approaches [29].

Model-based approaches estimate the sample size based on the characteristics of the model, the features of the data and minimum allowed classification error. For example, these estimation methods can range from rule of thumbs, such as one needs 'X' number of samples for each feature in a regression problem (these rules are normally derived empirically) to derive conditions for the magnitude of the sample size for shallow feed-forward neural networks [30], [31]. However, these methods are restricted to special network architectures, which are typically shallow and do not generalize well to different problem domains making them unsuitable for estimating data sizes for large-scale industrial models.

Curve fitting SSDMs instead simulate the training and testing process for increasing sizes of samples and fit a so-called learning curve (LC) from the recorded test accuracies. These methods are based on the assumption that the performance of a network increases monotonically with increasing training data and the relation has been shown to follow an inverse power law [32].

Figueroa et al. used a curve-fitting approach to predict the required sample size to achieve a classification performance f [33]. The study investigated model accuracy as a function of sample size given by:

$$f(x; a, b, c) = (1 - a) - b \cdot x^c \tag{2.4}$$

where x is sample size, and the parameters a, b, c are minimum error, learning rate and decay rate, respectively. Due to higher training variance when training models on smaller data sizes, the curve fit is weighted, where points corresponding to larger sample sizes receive a larger weight (priority). Fitting a curve according to Equation 2.4 from empirical data can then be done by using weighted least square optimization.

A huge downside with curve fit based SSDM methods is that one needs to gather a sufficiently large size of data to be able to run the simulation and achieve a high resolution in the curve fit, which to some degree defeats the purpose of estimating a sample size to begin with. To overcome this problem, one could instead use the LC extrapolated from one type of data to get a sample size estimate of how much data you need from a less available data set. This method assumes that the LC is approximately identical for the two types of data, which is generally not true. However, it is reasonable to expect that the LC for intra-domain data would be similar if all other factors are kept constant (model architecture, variable ranges etc.). Unfortunately, there has been a very limited amount of research studying the validity of this assumption. This lack of research is most likely attributed to the popularity in transfer learning, a method where pre-trained networks are fine-tuned to case-specific domains, which enables ML models to easily adapt to other problem

domains with substantially smaller data sets compared to training a model from scratch [34].

2.5 U-Net

The U-Net is a deep convolutional neural network (CNN) architecture developed and published by Ronneberger et al. [35]. More standard CNN architectures, such as AlexNet [36], that are structured using a convolutional and classifier block, sequentially increase the number of feature maps from the input image through a series of convolutional layers. These convolutional layers are often combined with nonlinear activation functions, pooling layers and batch normalization layers to stabilize the gradients of the network and to extract the most important features. After the convolutional block, typical CNN structures have a classifier block where the extracted feature maps from the image are passed through a series of linear layers that map to some output representation. For typical classification tasks, the output is a probability vector p representing the network's prediction for class i with prediction probability $p_i,\ i = 1, \ldots, |C|$.

The U-Net model, in comparison, as described in its name, has a U-shaped architecture where the input image is first downscaled in downsampling blocks and then upscaled in upscaling blocks, see Figure 2.1 for a schematic of the the U-Net architecture. The left section of the U-Net is often referred to as the contracting path and the right section is similarly referred to as the expanding path. The respective outputs of the downsampling block in the contracting path are concatenated, through a residual connection, with the outputs of the upscaling block with the same corresponding shape in the expansive path. The U-Net takes as input an image I with shape (C, H, W) and outputs a probability map $\mathcal{P}$ with shape (C', H', W'), where C, H, W is the input image's color channel, height and width respectively and C', H', W' is the output probability map's class probabilities, height and width respectively. Depending on padding and kernel size for the convolutional layers and or if center cropping is used in the residual connections, the shape of the feature map in the network's output will be either the same as the input ($H = H', W = W'$) or smaller ($H > H', W > W'$). In the former case each pixel in the input image has a 1:1 relation to the output pixel, i.e the model's classification of pixel $I_{i,j}$ is given by class $c_{i,j} = \text{argmax}(\mathcal{P}_{i,j})$. In the later case, the pixels in the output $\mathcal{P}$ correspond to the central pixels in the input image I, meaning for the border pixels, there doesn't exist any relation to the output map and therefore these pixels are not classified.

For multi-class segmentation, the input image's pixels are segmented and classified into $C' > 1$ different classes. The U-Net structure has been shown to perform very well in segmentation tasks where the network will label a class c to each pixel in the input image. The original article showed this by segmenting biomedical images into different cell structures [35]. U-Net models have also shown promising results for detecting defects in binary segmentation tasks where the model classifies pixels as either defects ($c = 1$) or background ($c = 0$) [37]. In the specific case of binary segmentation, the output dimension C' is set to 1 and the classification of a pixel is

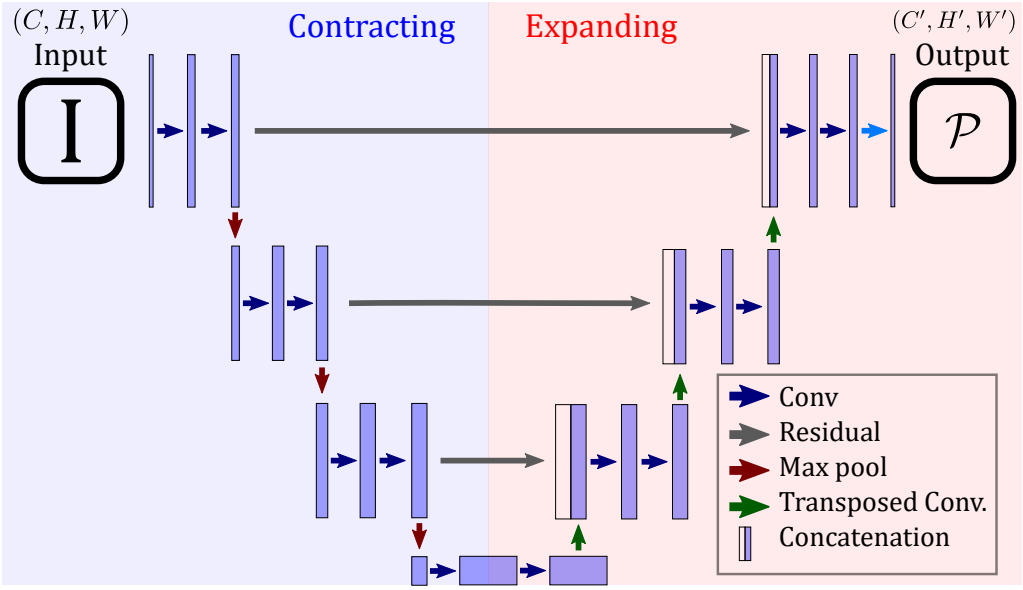


Figure 2.1: U-Net architecture. The input image I passes through several convolutional and max pooling layers (contracting path). The downsampled input is then upscaled through a series of transposed convolutional layers (expanding path). Residual connections take the inputs in the contracting path and concatenate it with the corresponding feature map size in the expanding path. Figure is replicated with partial inspiration from the original U-Net found in Ref. [35].

defined as

$$c_{i,j} = \begin{cases} 0, & p_{i,j} \leq \theta_t \\ 1, & p_{i,j} > \theta_t \end{cases}$$

where $p_{i,j} \in \mathcal{P}$ is the pixel probability in row i, column j in $\mathcal{P}$ and $\theta_t \in (0, 1)$ is a defined threshold.

2.6 Performance metrics

Performance metrics are essential to validate an ML model's generalization performance. The most prominently used performance metrics for classification tasks are precision P, recall R, F1-score F_1 and accuracy A. These metrics are applicable for multi-class classification but are usually applied in binary classification tasks where the classes are either *positive* or *negative*. For binary classification, there are 4 possible combinations of prediction and target class, termed as *true positive* (TP), *false negative* (FN), *false positive* (FP) and *true negative* (TN). These different combinations can be summarized in a confusion matrix, see Figure 2.2. From the confusion matrix precision P, recall R, F1-score F_1 and accuracy A can be defined as

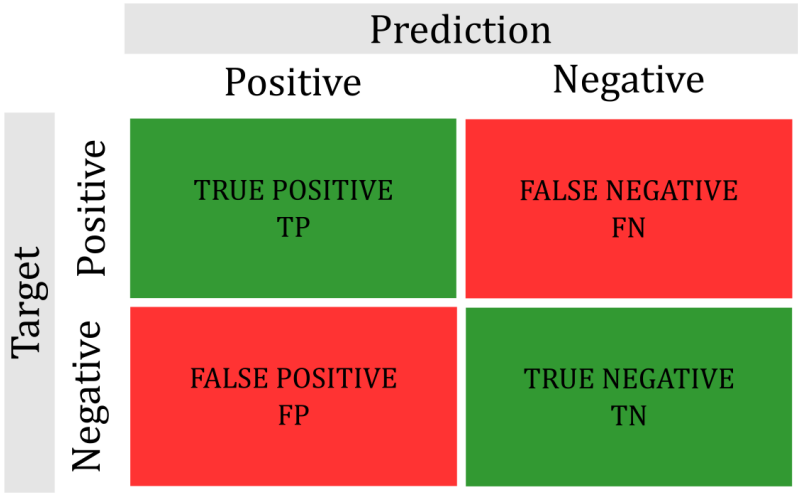


Figure 2.2: Binary confusion matrix with the 4 different binary classification combinations

$$P = \frac{\text{True positive}}{\text{True positive} + \text{False positive}} = \frac{TP}{TP + FP} \tag{2.5}$$

$$R = \frac{\text{True positive}}{\text{True positive} + \text{False negative}} = \frac{TP}{TP + FN} \tag{2.6}$$

$$F_1 = 2\frac{PR}{P + R} \tag{2.7}$$

$$A = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.8}$$

In the domain of defect detection, the positive class ($c = 1$) is usually labeled as defects and the negative class ($c = 0$) is considered the background or non-defects. Using these conventions the model's predictions can be visualized as a binary image. These performance metrics can be viewed from the perspective of individual defects (groups of defect pixels) or on a pixel-by-pixel basis. In the pixel case, each pixel is assigned a class c and is compared pixel-wise to the target mask. When instead considering a connected group of pixels, which we will refer to as defect *contours*, the comparison is based on if the prediction contour and the target contour overlap or not (one singular pixel overlap would classify as a TP). However, one might observe that the notion of *true negatives* does not make sense in this regard since the black background does not make up contours. The difference in the different perspectives is demonstrated in Figure 2.3.

2.6.1 Probability of detection

Probability of detection (POD) is a widely used measurement metric in non-destructive testing to assess the detection capabilities of inspection methods. In inspection tasks, such as defect detection for aerospace components, there is a much higher importance to detect larger defects compared to smaller defects as the larger ones can

(a) Prediction (b) Target

Figure 2.3: The binary prediction of a U-Net model (left) after a threshold has been applied to its output $\mathcal{P}$ and its respective target mask (right) . White pixels are classified as defects ($c = 1$) and black pixels are classified as background ($c = 0$). On a pixel basis there are 3 TP, 4 FN, 5 FP and 13 TN predictions. But if instead once considers defect contours, there are 2 TP, 1 FP and 1 FN.

potentially cause catastrophic failures once installed in vehicles. The POD curve describes the probability that an inspection will detect a defect characterized by a size a given by the relation

$$\mathrm{POD}(a) = g(a;\theta) \tag{2.9}$$

where $\mathrm{POD}(a)$ is the probability of detecting a defect of size a and $g(a;\theta)$ is a link function parameterized by θ. From prior research, the probit and logit link functions are preferred to describe the POD relation for real-world non-destructive testing inspection data [38]. For the logit link function, the POD can be described as

$$\mathrm{POD}(a) = \frac{\exp\left(\alpha + \beta \ln a\right)}{1 + \exp\left(\alpha + \beta \ln a\right)} \tag{2.10}$$

where $\theta = [\alpha, \beta]^\top$ are the curve parameters. POD methods can be branched into two main types, '$\hat{a}$ versus a' and 'hit/miss'. The $\hat{a}$ versus a method, where the inspection process generates a continuous output signal, is mostly used in, but not limited to, Eddy current inspections [39], [40]. In comparison hit/miss methods are more concrete, each defect size a has a binary measure, either being a hit (1) or a miss (0). The method for constructing POD curves is standardized in the MIL-HDBK-1823A handbook Ref. [20]. A link function g is chosen for the data and its respective parameters θ are estimated through maximum likelihood estimation (MLE) [41], [42], [43]. When using the logit link function as shown in Equation 2.10, this problem reduces to a standard binary logistic regression task. Furthermore, confidence levels can be achieved through log likelihood ratios [44], however one is mostly interested in the lower bound for the POD estimations when the number of inspection samples is very small. The most interesting points on POD curves that appear in industrial settings are a_{90} and $a_{90/95}$ which are the sizes a for which the defect is detected with 90% probability for the regular and 95% confidence level POD curves respectively, see Figure 2.4 for an example of a POD curve.

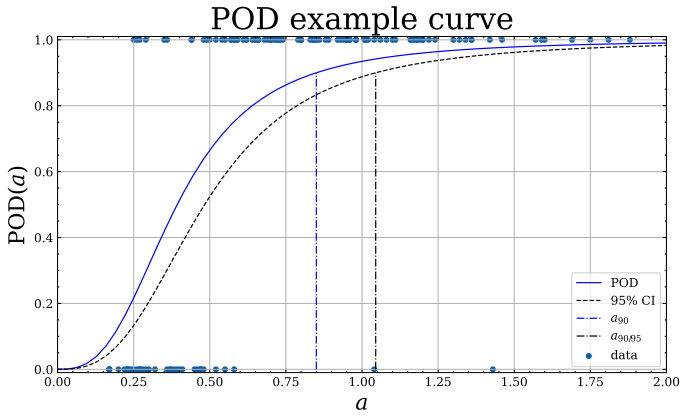


Figure 2.4: POD curve with its respective 95% estimation lower confidence level. The 90% defect detection sizes are highlighted at a_{90} and $a_{90/95}$ for the respective curves. The data is gathered from an operator manually inspecting images of crack defects and recording the sizes for which that are detected (hit) and which that are not (miss).

2.7 Improvement methods for deployed machine learning models

The research done on retraining methods for ML models once they have been deployed is very limited and the most common method is to completely retrain the model once more data has become available. The biggest downside to complete retraining from scratch is that for larger models, the computational cost and training time introduces a big problem for applications where the the input data is continually changing and deployed models can become stale. An alternative method to full retraining (FR) is *online learning* (OL) where the weights of a deployed ML model are updated whenever enough new data has been gathered to construct a new training batch. Prapas et al. [1], [45] propose a new method that lies somewhere in between full retraining and online learning called *proactive training* (PT). Proactive training is a continuous training method for ML models. Training data that the model has already been trained on is part of a historical data set while newly arrived data that is unseen to the network is kept in a separate set. Proactive training is similar to OL where the model's weights are updated in continuous batches instead of full retrains. The only difference between OL and proactive training is that the new batches in OL is made out of only new data while for proactive training, the constructed batches are made from a mix of the historical data and the new unseen data. Training batches in PT are characterized by a batch size B and trigger size t. The formation of a new batch is triggered when t new data points have been sampled or observed, where $(B - t)$ data points are sampled randomly from the historical set and combined with the B new data points to form a batch b. After the constructed batch has been used for a training iteration, the t new data points are moved to the historical set. For $t = B$ proactive training becomes identical to OL. By controlling the trigger size t, proactive training iterations are triggered $\frac{\text{batch size}}{\text{trigger size}} = \frac{B}{t}$ more times compared to OL.

Prapas et al. do not specify how historical data is sampled to form new batches. Investigating different sampling methods could be of interest since there still exists a temporal order in the historical data points that can affect the training performance. Examples of ways to weight the historical data points is uniformly, exponentially decreasing with time, frequency of times the data point has been used in the training or taking the last $(B - t)$ data points.

3

Method

This chapter outlines the methodology for solving the tasks established in Section 1.1. First, in Section 3.1 the different data sets are introduced. Then in Section 3.2 the architecture of the U-Net used in this thesis is presented followed up by how the model is trained in Section 3.3. The methodology and set of hyperparameters used to solve task 1 and task 2 are described in Sections 3.4 and 3.5 respectively.

3.1 Data set

The data used in this thesis is image data from two different inspection tasks that are acquisitioned internally in GKN's facilities. The two data sets are called X-ray and automated visual inspection (AVI). The original images from both data sets are several thousand pixels in both height and width. These large images are then cropped into smaller 512×512 squared images suitable for the U-Net model.

The X-ray data consists of scanned laser welds where imperfect welds give rise to air bubbles and pores (defects), see Figure 3.1 for a full X-ray image and Figure 3.2 for smaller crops with their respective target mask. The X-ray data set consists of a total of 37309 cropped images that are collected over the span of several years, from 2017 to 2022, see Table 3.1 for a year by year split. The X-ray images are encoded in an unsigned 16-bit grey scale. Higher values (brighter pixels) correspond to low-density regions, i.e air pockets and pores where the X-ray light does not penetrate as much material while lower values (darker pixels) correspond to high-density regions indicative of nominal welds.

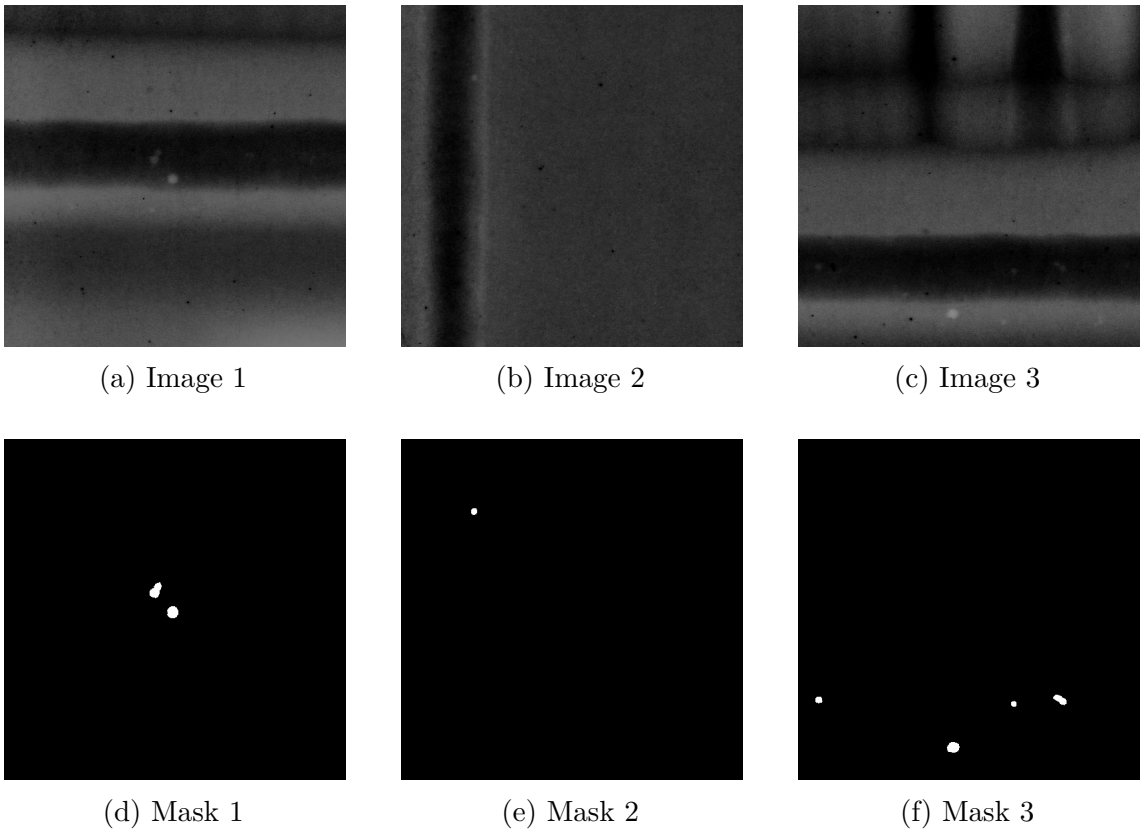
Table 3.1: Year by year split of the X-ray image crops.

Year	Images
2017	8884
2018	16946
2019	2597
2020	524
2021	5220
2022	3138
Total	37309

Figure 3.1: Full 5861×6840 X-ray image taken of a weld segment. A 512×512 sliding window traverses the full images and extracts image crops that are fed to the U-Net model. The weld regions are highlighted in the red bounding boxes. Small white speckles along the welds correspond to pores, indicative of defects that can potentially fail a component.

The AVI data are acquired from a camera sweeping over and scanning the surface of metal components. There are multiple types of defects that can appear on the surface such as scratches, dents, fingerprints, discolorations and cracks. Here we only consider images of dents as their labeled target masks are the most similar in shape to the ones in the X-ray data, see Figure 3.3 for smaller crops with their respective target mask. The AVI images are encoded in unsigned 8-bit RGB. The AVI images have no year attribute as they are generated through a proprietary augmentation process. The AVI dataset consists of 27000 cropped images.

The cropped images, from both data sets, are preprocessed by converting their data type to float32 and normalizing the pixel values in the range $[0, 1]$, whereas for the AVI images, each color channel is normalized individually. The corresponding



(a) Image 1 (b) Image 2 (c) Image 3

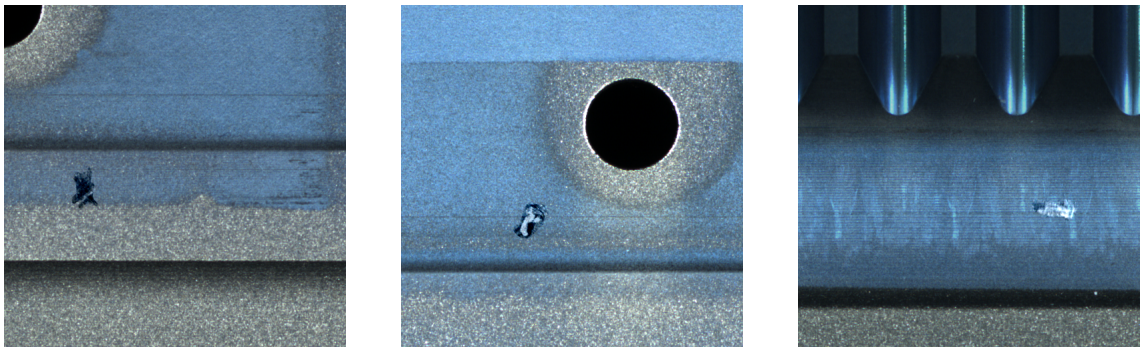
(d) Mask 1 (e) Mask 2 (f) Mask 3

Figure 3.2: 512×512 X-ray crops and their respective binary target mask. Defective pores are appear as white circular bright spots along the black weld.

binary masks to the cropped images are encoded in unsigned 8-bit to save memory. However, during training, each batch is converted to float32 just before calculating the loss.

3.2 U-Net architecture

The deep learning model used for this thesis is a U-Net model. The specific U-Net architecture is inspired by GKN's previously implemented models. The main differences are in the deep learning framework used (this model is implemented in Pytorch while GKN's is implemented in Tensorflow) and that 2D batch normalization layers are added to this model. The batch normalization layers are added after each of the convolutional blocks to maintain the magnitude of the input as it propagates through the network, see Appendix A.1 for a detailed description of the U-Net's complete layer architecture. The complexity of the model is controlled through a parameter $v \in \{1, 2, 3\} \subset \mathbb{N}$ that linearly scales the number of all the feature maps with the factor v. The number of trainable model parameters as a function of v is shown in Table 3.2.



(a) Image 1 (b) Image 2 (c) Image 3

(d) Mask 1 (e) Mask 2 (f) Mask 3

Figure 3.3: 512x512 AVI crops and their respective binary target mask.

Table 3.2: Number of trainable model weights as a function of complexity parameter v.

Model complexity v	# weights (million)
1	2
2	8
3	17

3.3 Training U-Net model

Since this thesis requires training a U-Net model a repeated number of times, a brief summary of the training process will be explained in this section. The training process is given in Algorithm 1. First, a U-Net model M is instantiated with complexity given by the v parameter. This is followed by loading in the data D and sectioning it into train D_{tr}, validation D_{val} and test D_{te} subsets. The model's weights w are updated incrementally by passing batches b of the training data through the network and then updating the weights according to the gradients g. The gradients are calculated by applying the model output $\mathcal{P}$ and target mask $\mathcal{T}$ to the loss function L and then calculating the gradient with respect to the model weight $\frac{\partial L}{\partial w}$. The loss

function used, is a weighted binary cross entropy (WBCE) loss defined as

$$L(w) = \frac{1}{N} \sum_{p_{i,j} \in \mathcal{P}} w_+ t_{i,j} \log(p_{i,j}) + (1 - t_{i,j}) \log(1 - p_{i,j})$$

where w_+ is the penalty factor for the positive class, $p_{i,j}$ is the pixel probability of pixel (i, j) and $t_{i,j}$ is its respective target. The positive penalty factor w_+ is chosen as the ratio of white pixels to black pixels in the entire train set. One epoch is defined as one complete pass of the entire training data. After each epoch, a validation score is measured using the validation set. Because of the large network, one epoch can take several minutes of computational time. To prevent wasteful compute, an early stoppage criteria is triggered whenever N_p consecutive epochs yield worse validation performance compared to the validation performance from the current best epoch. The value N_p is usually called the *patience*. Once the model has finished training, it is evaluated on the test set to measure its generalization performance.

Algorithm 1: Batched training of U-Net model

```
Instantiate U-Net model M with complexity v
Load in all data D
Partition the dataset D according a random generated 80/10/10 split to define
  a train set D_tr, validation set D_val and test set D_te
Section D_tr into batches b of batch size B
for epoch e ∈ {1, 2, ..., E_train} do
    for batch b in D_tr do
        P ← M(b): Pass batched input images through model M(b) to obtain
          batched output probability map P
        L ← L(P, T): Calculate the loss L by comparing output P with target
          mask T
        g ← ∂L/∂w: Calculate batch gradients g by applying Backpropagation to
          the loss function L
        w ← optimizer(w, g): Update the model's weights w according to the
          optimizer function
    Calculate validation metrics using D_val
    if early stoppage criteria met then
        stop training
    else
        continue
Evaluate the trained model's generalization performance using the test set D_te
```

3.4 Task 1 - Domain validation

Prior to running the longer numerical simulations, some of the model's hyperparameters need to be manually tweaked. Since the model outputs a probability map $\mathcal{P}$ for each image, one needs to determine a suitable threshold value θ_t to differentiate

between black pixels ($c = 0$) and white pixels ($c = 1$). Normally for balanced binary data, a threshold value of $\theta_t = 0.5$ will, in general yield the best performance. However, for heavily imbalanced data sets, as in this case, performing threshold tuning is essential for achieving optimal performance. As mentioned in Section 2.6 it is possible to define performance metrics based on pixels and contours. In defect detection tasks, one is more interested in actually finding the defect compared to predicting the exact shape of it. Therefore, the most truthful and representative metric that one would want to maximize in an industrial setting, is given by the contour F1-score denoted as F_{1C}. The standard method for tuning a threshold θ_t for binary classification tasks, requires one to simulate a receiver operating characteristic (ROC) curve [46]. However, as mentioned in Section 2.6 one cannot define true negatives for contours, and instead needs to resort to a precision-recall curve (PRC). The optimal threshold θ_t in a PRC is given by the point corresponding to the highest F_{1C} score.

To establish a PRC curve, a U-Net model with complexity parameter $v = 1$ is trained using X-ray images from the years 2017 to 2021 as described in Section 3.3. Since, at this stage no suitable threshold θ_t had been tuned, one could not optimize against contour metrics. Instead, validation loss was used as a optimization metric for the early stopping criteria. The trained model is then evaluated using a subset consisting of 300 random samples from the year 2022, measuring contour precision P_C and contour recall R_C. The test evaluation is repeated several times while varying the threshold in the interval $\theta_t \in (0, 1)$. In addition to varying the threshold, a minimum allowable contour size s_c was imposed where contours made up of fewer pixels than s_c were discarded and replaced with black pixels. The main idea with introducing a minimum limit s_c is to rule out any potential one-off pixels that are too small to physically translate to a defect, as the model tends in some cases to assign singular pixels a very high prediction probability $p_{i,j}$. These singular pixels can thus skew the performance metrics since a single pixel technically is a contour. This evaluation process was repeated for a model with complexity parameter $v = 2$ as well to investigate whether its respective PRC would look any different and if it would result in a different optimal threshold θ_t.

For task 1, the main goal is to validate and estimate the learning curves for a U-Net model trained on two intra-domain data sets, in our case, the X-ray and AVI data sets. In addition, comparisons between evaluations on varying sizes of test sets will be done, from which one can establish a estimate of representative test sizes. The main algorithm for task 1 is presented in Algorithm 2 and the hyperparameters used in Table 3.3. From the entire data set, a static industrial set D_i of size $|D|/k$ ($k = 11$) is set aside right after loading in the data D, while the rest of the data D_r is used for the remaining process. Here the industrial set is used to approximate a large enough data set to represent what one can expect once the model is deployed in industry. The industrial set remains fixed throughout the entire simulation. The main loop of the code, incrementally trains a U-Net model with larger sizes of available training data by increasing the split $s \in S = \{s_1, s_2, \ldots, s_M\}$. The available data D_s for each split s is randomly sampled from the rest data D_r. The split data D_s is then partitioned into a standard train set D_{tr}, validation set

D_{val} and test D_{te} (hence why we need another name for the industrial set) according to a 80/10/10 divide. As mentioned in Section 2.3, one would expect the optimal complexity of the model to depend on the size of available training data. Therefore a hyperparameter optimization (HPO) simulation is done by training a model for E_{hpo} epochs with different complexity parameter v and batch size B. The configuration of hyperparameters corresponding to the highest achived validation F_{1C} during the HPO training (the one with E_{HPO} epochs), is selected for the specific split. Since training neural networks is an inherently stochastic process, the training is repeated $N = 5$ times for each split to establish a mean and variance of the training process. After each model training, the U-Net is evaluated on both the test set D_{te} and industrial set D_i to record potential differences originating from the limited test set. Capturing the discrepancy between the test and industrial set and when this approaches zero is of interest to determine a lower bound for a representative test size.

A modified version of Algorithm 2 was simulated where the HPO step was omitted (skipping line 6 in Algorithm 2) and the hyperparameter configuration of the U-Net model was instead fixed with complexity parameter $v = 1$ and batch size $B = 8$. The reason for skipping the HPO step was due to preliminary results indicating that the model showed optimal performance for complexity $v = 1$ regardless of the size of the training data.

Algorithm 2: Task 1 main loop

```
1  Load in all data D
2  Set aside |D|/k samples from D for an industrial set D_i and let the remaining
     data define a rest set D_r
3  for split s in S = {s_1, s_2, ..., s_M} do
4      Randomly sample s|D_r| data points into a reduced split data set D_s
5      Use a random 80/10/10 split on D_s to create a train set D_tr, validation set
         D_v and test set D_te partition
6      Perform HPO using D_tr to train and D_v to validate
7      for i in 1,...,N do
8          Train a model from scratch using the best found hyperparameters
9          Evaluate trained model on test set D_te and industrial set D_i
10         Log evaluation metrics
```

Table 3.3: Task 1 hyperparameters. These parameters are kept fixed throughout task 1 while complexity parameter v and batch size B are determined from the HPO of each split s.

Optimizer	Adam
Learning rate α	1e-4
Loss function	WBCE
Positive class loss weight w_+	183.3
Training epochs E_{train}	50
HPO training epochs E_{hpo}	30
Patience N_p	8
Threshold θ_t	0.95

To establish a learning curve for the X-ray data an inverse power law curve is fitted given by Equation 2.4, by training a U-Net with varying sizes of data. As previously mentioned in Section 2.6, the metric of interest in defect detection is contour F_{1C}. The logged metrics from Algorithm 2 will be used to fit a learning curve on the form:

$$F_{1C}(x) = (1 - a) - b \cdot x^c \tag{3.1}$$

where x is sample size and a, b, c are parameters estimated from the curve fit algorithm. The solver for the curve fit is weighted ordinary least squares (WOLS) where the residual of each sample j is weighted with a factor w_j. We investigate 3 different weight methods; $w_j = 1, \forall j$ (ordinary least squares), $w_j = \frac{j}{N_{fit}}$ as described in the methodology of [33] where N_{fit} is the number of points used to fit the curve and $w_j = \frac{1}{\sigma_j}$ where σ_j is the standard deviation of $F_{1C}(x_j)$ calculated from the N repeated trainings for each split s.

3.5 Task 2 - Continuous improvement

For task 2 the main goal is to investigate different methods for continuously improving an imperfect machine learning model that has been deployed in a production setup and investigate how to improve it as fast as possible. The underlying problem assumption is that an imperfect ML model has been deployed into an industrial setting and is continually predicting queries on incoming image data from production. This data is then assumed to be annotated making it possible to use it for future training. From this standpoint the performance and the learning speed of different retraining methods will be studied as an artificial timeline of incoming future data is simulated and used to update the model. The timeline is created by using the archive of X-ray data from the years 2017 to 2021 where the first 1000 images from 2017 are used to train a baseline model. The data from the year 2022 is used as a final held-out test set once the data has gone through the entire timeline to compare the methods at the end.

The retraining methods that will be investigated can be categorized into three groups

1. Fully retraining (FR) the ML model from scratch once C_{FR} new data samples arrive. The new model is trained using the old data combined with the C_{FR} new data samples. For FR, a portion of the data needs to used for the validation set D_{val} to monitor the training. In this task we use a 80/20 split for train and validation set respectively.

2. Online learning (OL) where we continually update the deployed model with batches b using only new incoming data. Each data sample is only used once to update the model and then discarded.

3. Proactive training (PT) as described in Section 2.7. For the PT, we investigate different sampling methods and how these affect training performance, see Figure 3.4 for how each batch is sampled in PT. The following sampling methods will be used when sampling from the historical data set $\mathcal{H}$:

   (a) **R**andom uniform distribution: $p_i = \frac{1}{|\mathcal{H}|},\ i \in \mathcal{H}$

   (b) **F**requency based: $p_i = \frac{1}{f_i f_{\mathcal{H}}},\ i \in \mathcal{H},\ f_{\mathcal{H}} = \sum_{i \in \mathcal{H}} 1/f_i$

   (c) **B**oltzmann distribution: $p_i = \exp(iC_B)/Z,\ i \in \mathcal{H},\ Z = \sum_{i \in \mathcal{H}} \exp(iC_B)$

   (d) Uniform sliding **W**indow : $p_i = \frac{1}{|W|},\ i \in W,\ W \subset \mathcal{H}$

   (e) **L**ast $(B - t)$ samples in history set $\mathcal{H}$.

   where p_i is the sampling probability for data point $i \in \mathcal{H}$, f_i is the number of times sample i has been used to update the model, C_B is the Boltzmann factor and W is a sliding window consisting of the last $|W| = N_W \cdot B$ $(N_W \in \mathbb{N})$ samples in the history set $\mathcal{H}$. We introduce the naming conventions PT-R, PT-F, PT-B, PT-W and PT-L to refer to the different sampling methods mentioned above, in the respective order they are introduced. Furthermore, a specific PT method with a trigger size t is symbolized as PT-Xt where X is the sampling method, for example, PT-R4 is random uniform sampling with trigger size $t = 4$. To refer to the collection of PT methods with the same trigger size, we omit the X in the previous notion as PT-t.

As mentioned in Section 2.2, prequential evaluation is suitable in settings where data is continually generated in a streaming format. The X-ray data is loaded into memory according to the year attribute of the image crops and aligned as a stream of images. The deployed model is initially trained on a relatively small data set to achieve mediocre performance, leaving room for a large margin of improvement. This initial model will be used as a baseline. The subset of data used for the initial training makes up the history set $\mathcal{H}$. The model is then continuously improved through batch-wise updates as new data from the artificial stream becomes available, see Algorithm 3 for the case of PT. First, the data is evaluated on the current model and then used to update it. The value of the parameters used for the different retraining methods are listed in Table 3.4.

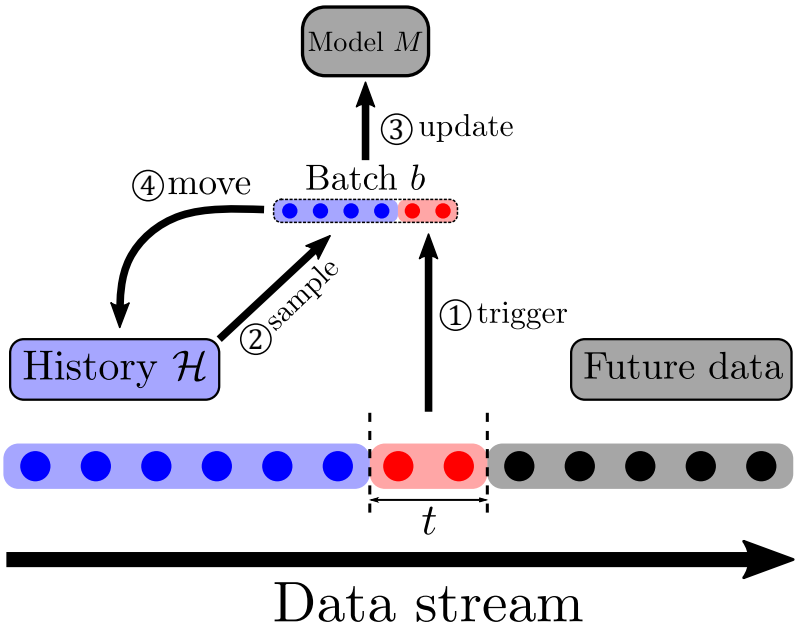


Figure 3.4: Proactive training. Once t new data points become available (red points), a new update is triggered (1). Sample $(B-t)$ data points from the history set $\mathcal{H}$ (blue points) according to some given sampling method (2). Update the model with the new batch b (3). Move the t new data points to the history set $\mathcal{H}$ (4).

Algorithm 3: Proactive training & prequential evaluation

```
1  Load in partially trained model M_0
2  while True do
3      if t new data samples become available then
4          Evaluate the model on the t new samples
5          Sample (B − t) unique data points from the historical set H according
             to p
6          Construct a batch b using the t new data points combined with the
             sampled data
7          Update the model using batch b according to lines 6-10 in Algorithm 1
             (Batched training of U-Net model)
8          Move the t new data samples to the historical set H
9      else
10         Continue
```

Table 3.4: Task 2 experiment parameters

Batch size B	16
Trigger size t	[4,8,12]
Full retrain trigger C_{FR}	7000
Window batch size count N_W	50
Boltzmann factor C_B	$\log(2)/(50B)$

4

Results

This chapter presents the results obtained from the experiments described in Chapter 3. The results for task 1 and task 2 are split up in Sections 4.1 and 4.2 respectively.

4.1 Task 1 - Domain validation

The intensities of the pixel probabilities $p_{i,j}$, as mentioned in Section 2.5, in the output probability map $\mathcal{P}$ of an example input is shown in Figure 4.1. The trained U-Net model converges to a solution (by solution we refer to the set of obtained model weights) that learns to assign high pixel probabilities $p_{i,j}$ to regions that are defective. From the pixel distribution it is evident why a naive threshold value of $\theta_t = 0.5$ will lead to a significant drop in prediction performance since pixel probabilities in the range $p_{ij} \in [0.5, 0.95]$ will be classified as white pixels (defects).

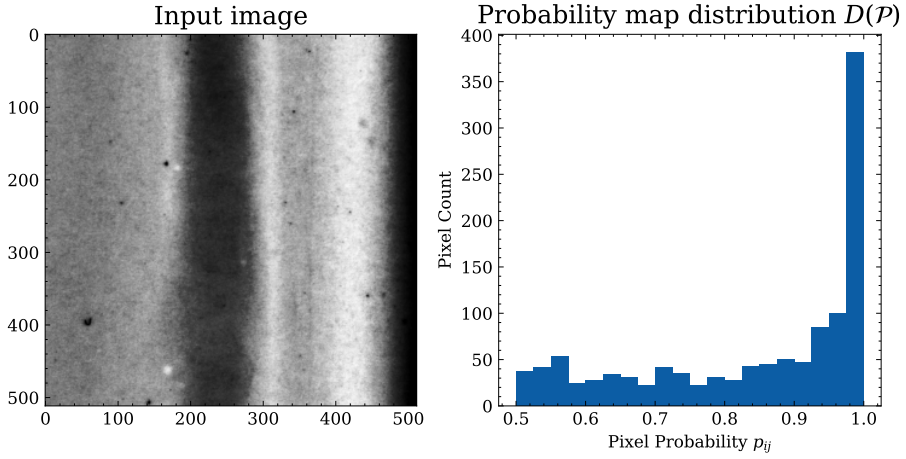
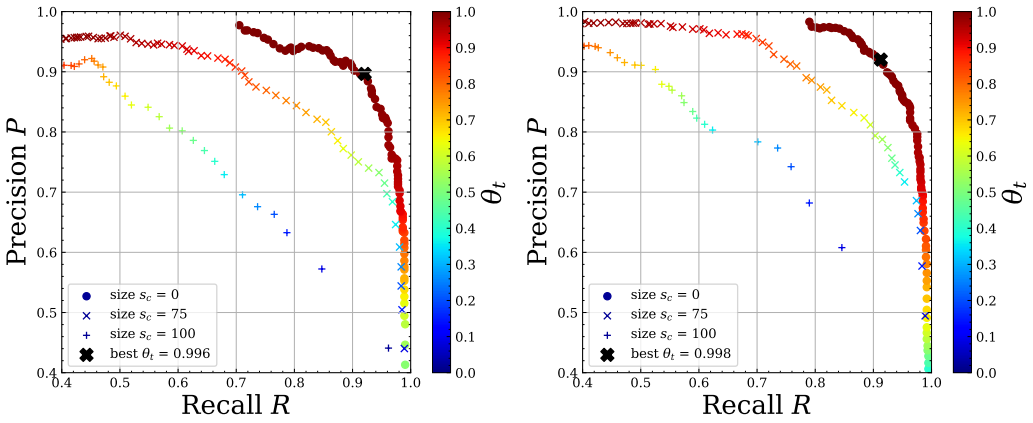


Figure 4.1: Example input image (left) and the distribution of pixel probabilities $p_{i,j}$ of the output probability map $\mathcal{P}$ from the U-Net model (right). Note that pixel values in the histogram are limited to the range $[0.5, 1.0]$.

The precision-recall curve for a U-Net model trained with complexity parameter $v = 1$ and $v = 2$ are shown in Figure 4.2. The threshold values θ_t are logarithmically spaced in the interval $\theta_t \in (0, 1)$, with a finer resolution for higher threshold values. Here the optimal threshold occurs for $\theta_t > 0.995$ for the two different model complexities. This was measured on a model trained on a large training set where

one can expect optimal training convergence. There is a risk that models trained on lower sizes of data will not converge to solutions that assign pixel probabilities above $p_{i,j} > 0.99$ for defects, thus a slightly more conservative threshold value of $\theta_t = 0.95$ was used for the remaining simulations.



(a) $v = 1$ (b) $v = 2$

Figure 4.2: Precision recall curves for model trained with complexity parameter $v = 1$ (left) and $v = 2$ (right). Minimum contour sizes $s_c = \{0, 75, 100\}$ are shown.

The split values of $S = \{.01, .02, .03, .05, .075, .1, .125, .15, .2, .25, .35, .5, .75, 1.0\}$ are, for both data sets, used to train the model with successively larger size of training data D_{tr}. Note that, since these data sets differ in size, the same split s corresponds to different train and test sizes for X-ray compared to AVI, see Table 4.1 for data set sizes as a function of split s for the different data sets. The best configuration from the HPO results for each split is presented in Table 4.2. The complexity parameter $v = 3$ is enabled only for splits $s > 0.5$ as it is only for higher training sizes one would expect it to outperform lower complexity models. The results indicate that the best hyperparameter configurations with complexity v and batch size B are $(v, B) = (1, 4)$ & $(1, 8)$. Complexity $v = 1$ with high batch sizes is favoured throughout the entire range. For a fixed complexity, the higher batch sizes $B = 4, 8$ are preferred while a batch size $B = 1$ results in subpar performance regardless of complexity v. The results from the HPO simulation justified skipping the HPO in Algorithm 2 and use a fixed hyperparamter configuration. Thus the simulation was done with a fixed configuration $(v, B) = (1, 8)$, once for the X-ray set and once for the AVI set. Thus, in total, 3 runs were performed; X-ray with HPO, X-ray without HPO and AVI without HPO.

Table 4.1: Data set sizes as a function of split $s \in S$ for X-ray and AVI.

Split s	Train size (X-ray)	Test size (X-ray)	Train size (AVI)	Test size (AVI)
0.01	272	34	197	24
0.02	544	67	393	49
0.03	815	101	590	73
0.05	1357	169	983	122
0.075	2036	254	1473	184
0.1	2714	339	1965	245
0.125	3392	424	2456	306
0.15	4071	508	2946	368
0.2	5428	678	3928	491
0.25	6784	848	4910	613
0.35	9498	1187	6874	859
0.5	13568	1695	9819	1227
0.75	20352	2543	14728	1841
1.0	27135	3391	19637	2454

Table 4.2: HPO results measuring contour F_{1C} for different hyperparameter configurations of (v, B) for each split $s \in S$. The best hyperparameter configuration for each split is bold faced.

v, B \ s	.01	.02	.03	.05	.075	.1	.125	.15	.2	.25	.35	.5	.75	1.0
(1,1)	.642	.605	.682	.735	.752	.778	.584	.727	.484	.463	.713	.555	.778	.636
(1,4)	**.730**	**.730**	.683	**.845**	**.831**	**.850**	.772	.805	.829	**.838**	.826	.855	.825	.855
(1,8)	.500	.562	.507	.766	.787	.839	**.834**	**.833**	**.844**	.818	**.865**	**.864**	.833	**.869**
(2,1)	.366	.449	.497	.485	.192	.049	.267	.296	.307	.036	.036	.189	.043	.033
(2,4)	.642	.691	.718	.796	.762	.841	.796	.799	.762	.732	.827	.756	.768	.704
(2,8)	.538	.628	**.718**	.810	.757	.805	.818	.819	.834	.807	.786	.790	**.843**	.746
(3,1)	–	–	–	–	–	–	–	–	–	–	–	.199	.030	.040
(3,4)	–	–	–	–	–	–	–	–	–	–	–	.739	.752	.464
(3,8)	–	–	–	–	–	–	–	–	–	–	–	.787	.792	.868

The metric of interest is contour F_{1C} since it is the most relevant to defect detection. Here we present the results for F_{1C} while the less relevant metrics such as pixel metrics and the other contour metrics are shown in Appendix A.2. The mean and one standard deviation of F_{1C} as a function of training size is shown in Figure 4.3 for the 3 respective runs. As expected, all 3 runs showcase increasing performance with increasing training data. Additionally the standard deviation decreases with larger training sizes indicating that there exists some instability in the training process for small train sizes that diminishes for larger sizes. Furthermore, the absolute difference between the evaluations on the test set and industrial set approaches 0 with increasing test size $|D_{te}|$, as shown in Figure 4.4. This suggests that evaluations on test sets with limited size, in this case below 300 test images leads to a deviation

above 0.02, will not be representative of industry. This type of uncertainty in model evaluation needs to be considered if one deals with a limited size of data.

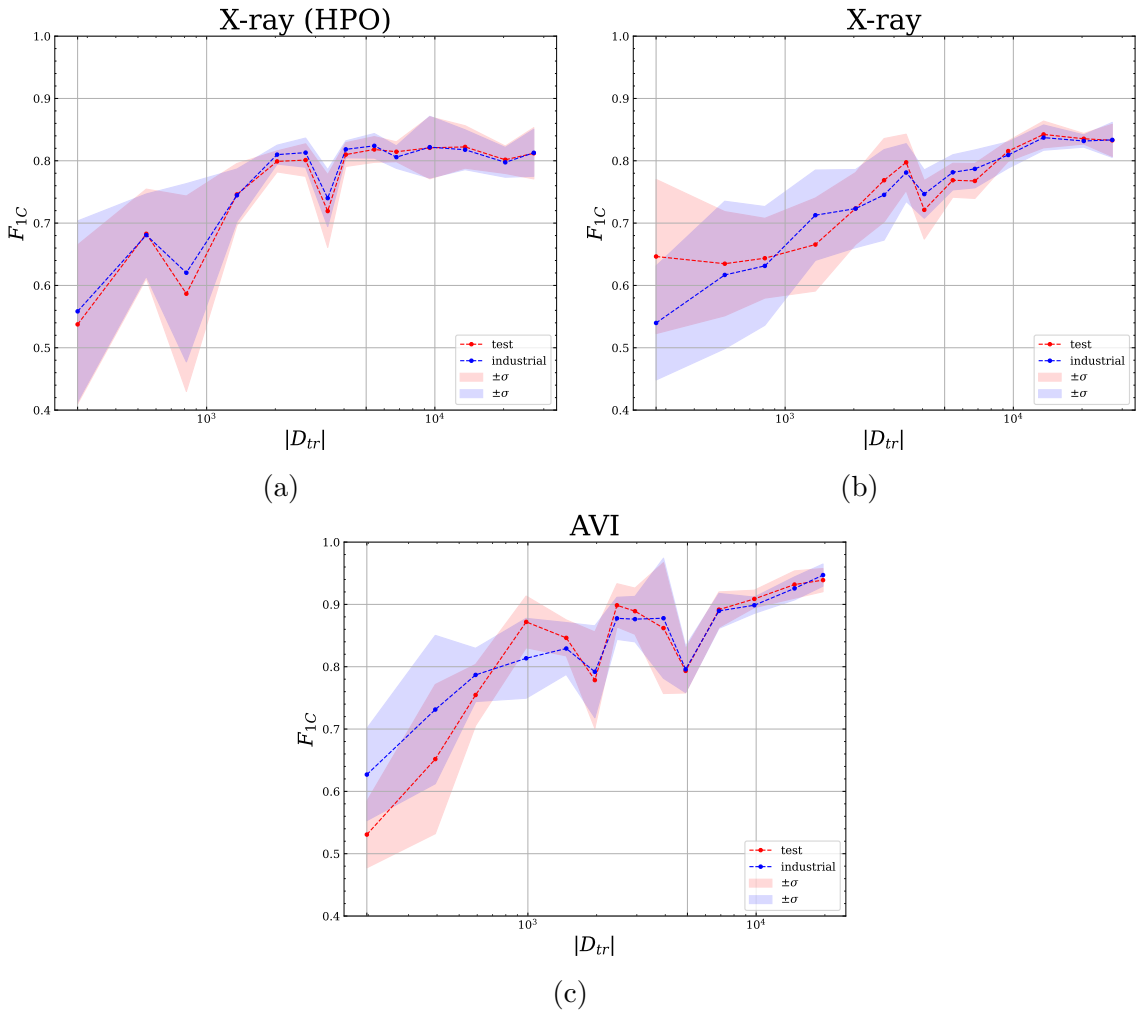


Figure 4.3: Contour F1-score F_{1C} for different runs. The test (red) curve is evaluated using the test set D_{te} for each split s while the industrial (blue) is evaluated on a static industrial set D_i.

The mean values evaluated on the industrial set are then used to fit a curve using 3 different weight functions as described in Sections 3.4, see Table 4.3 for mean average error (MAE) results and Appendix A.2.4 for plot of fitted curves. In a realistic scenario, one would be restricted to a limited amount of training data. For this reason, the first $j = 5$ points are used to fit the curve, and the remaining points are used for validation. The value $j = 5$ corresponds to having around 2.5-3k annotated crops, as given by Table 4.1, which is estimated to take around one work week to gather (gauged based on supervisors previous experience gathering data). The biggest differences between the fitting methods is shown for X-ray with HPO where the unweighted fit has the lowest fit and validation (MAE). For the non HPO runs the differences between the different methods become once again negligible, however the unweighted fit performs slightly better with lower validation and fit MAE.

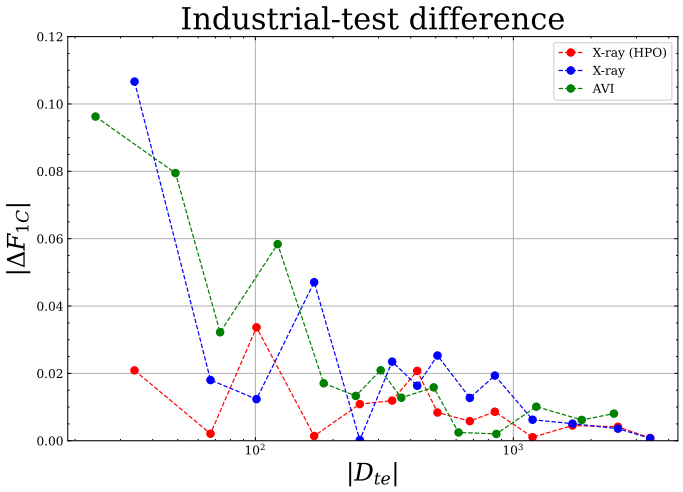


Figure 4.4: Difference in F_{1C} between the test evaluations on the varying test set D_{te} and fixed industrial set D_i.

Table 4.3: Comparing the different curve fitting methods: unweighted (1), σ-weights (2) and linear weights (3).

	X-ray (HPO)			X-ray			AVI		
Method	1	2	3	1	2	3	1	2	3
E_{fit}	**.030**	.031	.043	**.009**	.009	.012	**.004**	.006	.007
E_{val}	**.049**	.077	.070	**.013**	.014	.015	**.033**	.033	.035

Lastly, the curve fit using the measured points from the X-ray without HPO is used to predict the behaviour of the AVI data (intra-domain prediction) to assess how well two similar data sets will perform, see Figure 4.5 for comparison. Based on the previous results comparing the different curve fit methods, the unweighted fit was used to fit the X-ray curve. The deviation between the X-ray curve and AVI target is quite substantial measuring an average deviation of $E_{mae} = 0.11$ for F_{1C} prediction. This deviation is considered substantial as F_{1C} is limited in the interval $[0, 1]$.

Probability of detection curves were estimated using the test set D_{te} for different split values s, see Figure 4.6 for estimated POD and its 95% confidence level curve. The figures show that the critical values a_{90},$a_{90/95}$ and the their gap $|a_{90} - a_{90/95}|$ decreases with increasing split s. This is also shown in Figure 4.7 for all splits $s \in S$. The critical values decrease with higher s, since this correlates to models trained with more data resulting in better detection performance. The gap shrinks due to larger test size $|D_{te}|$ making the 95% confidence level curve approximately similar to the regular POD curve.

4.2 Task 2 - Continuous improvement

For the PT methods, regardless of sampling method, the trigger size t dictates the number of batch updates; $t = 4$ corresponds to 8292 updates, $t = 8$ to 4146 updates

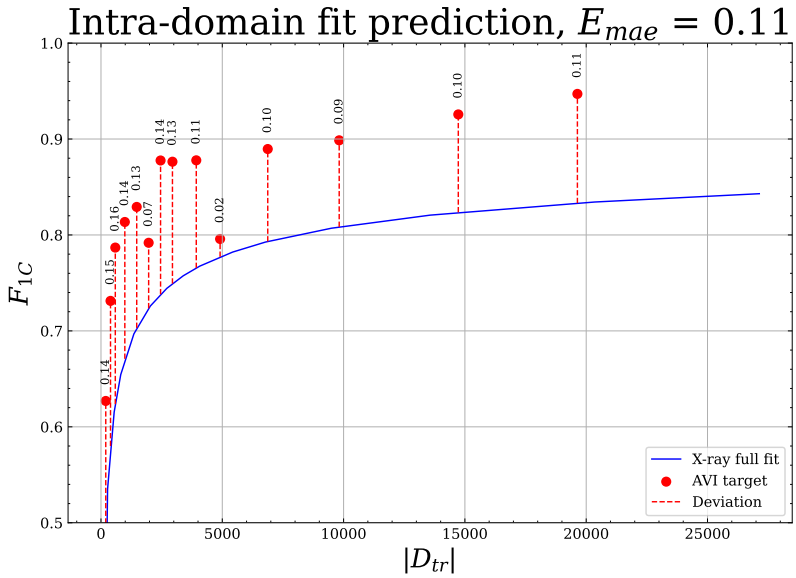


Figure 4.5: Domain prediction. Comparing the curve estimated from X-ray data to predict samples for AVI. The absolute deviation δ is shown for each point.

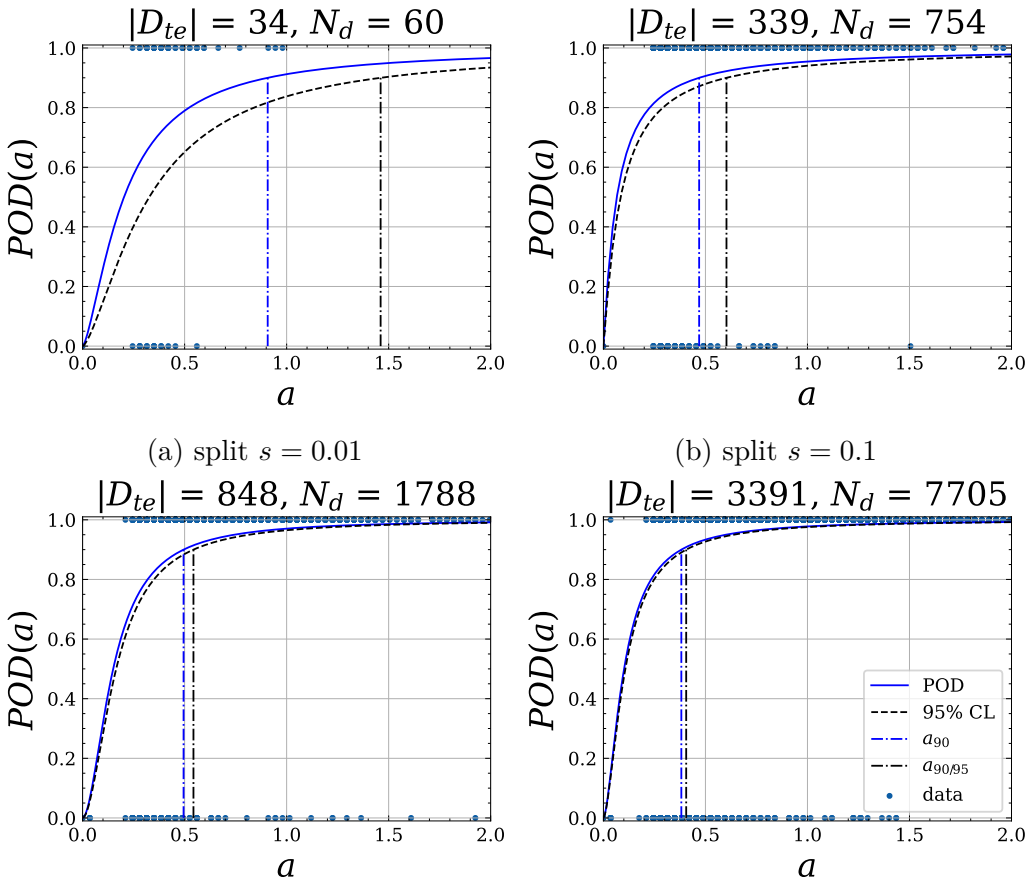
and $t = 12$ to 2764 updates, see Figure A.25 for detailed comparison of sampling distributions between the different PT methods.

The results of the FR algorithm are shown in Figure 4.8 measuring the cumulative F_{1C} mean and a moving mean with a window of the last $W = 500$ data points. The cumulative F_{1C} is fairly constant until the 7000th data point in the stream in which the first retrain is triggered. The initial retrain captures the largest increase in performance. Subsequent retrainings do not yield any significant change in F_{1C}. The model after the third retrain achieves the highest moving mean performance of $F_{1C} \approx 0.82$. The model from the last retrain, is trained using the most data, however, since the training process is stochastic it is possible to obtain a slightly worse model as is observed towards the end of the timeline.

The results for the PT methods, for $t = 4, 8, 12$, are presented in Figures 4.9a (PT-R), 4.9b (PT-F), 4.9c (PT-B), 4.9d (PT-W) and 4.9e (PT-L). For the proactive methods, the model improvement is gradual compared to FR, making it sufficient to show cumulative F_{1C}. For all methods the lower trigger size t results in better performance, except for PT-L where $t = 8$ for performed better than $t = 4$.

The comparison between FR, OL and PT-4 (PT with $t = 4$) is shown in Figure 4.10 for a cumulative F_{1C} mean. FR significantly outperforms all other methods. PT-R4, PT-F4, PT-B4 and PT-W4 have similar performance while PT-L4 and OL are the worst training methods.

After the continuous training has been finished, the trained models are evaluated on a held-out data set consisting of the images from 2022 which is not part of the timeline. The final F_{1C} evaluation is presented in Figure 4.11 for FR, OL and the PT-4. FR clearly leads to the best-performing model for the final evaluation while OL performs the worst. The significant differences between the PT methods indicate



(a) split $s = 0.01$

(b) split $s = 0.1$

(c) split $s = 0.25$

(d) split $s = 1.0$

Figure 4.6: POD curves for different splits s. The POD curves are estimated from $|D_{te}|$ test images which corresponds to a total of N_d defects.

that the sampling method has a significant impact on the continuous improvement even though all the models had an equal number of batch updates for the different PT methods. Of the 5 different sampling methods, frequency-based sampling (PT-F) performed the best and sampling from the latest point in the history set (PT-L) performed worse.

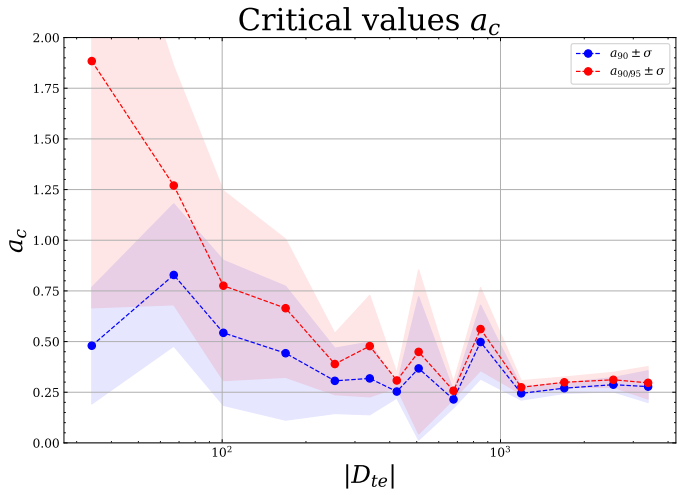
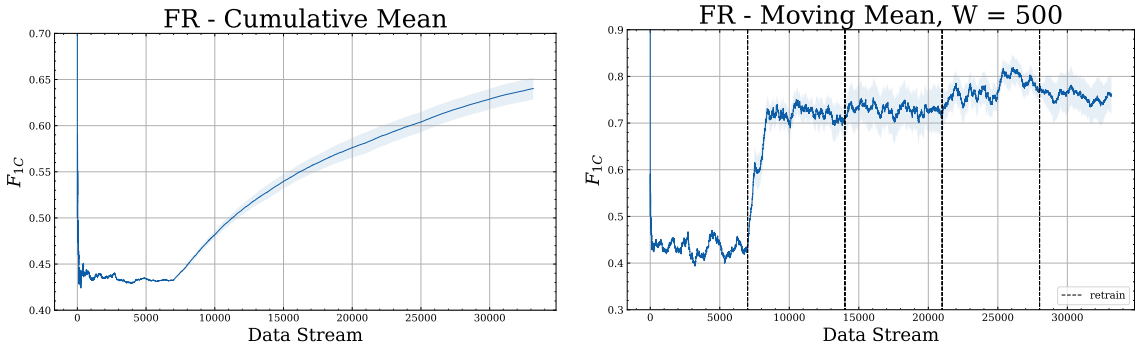


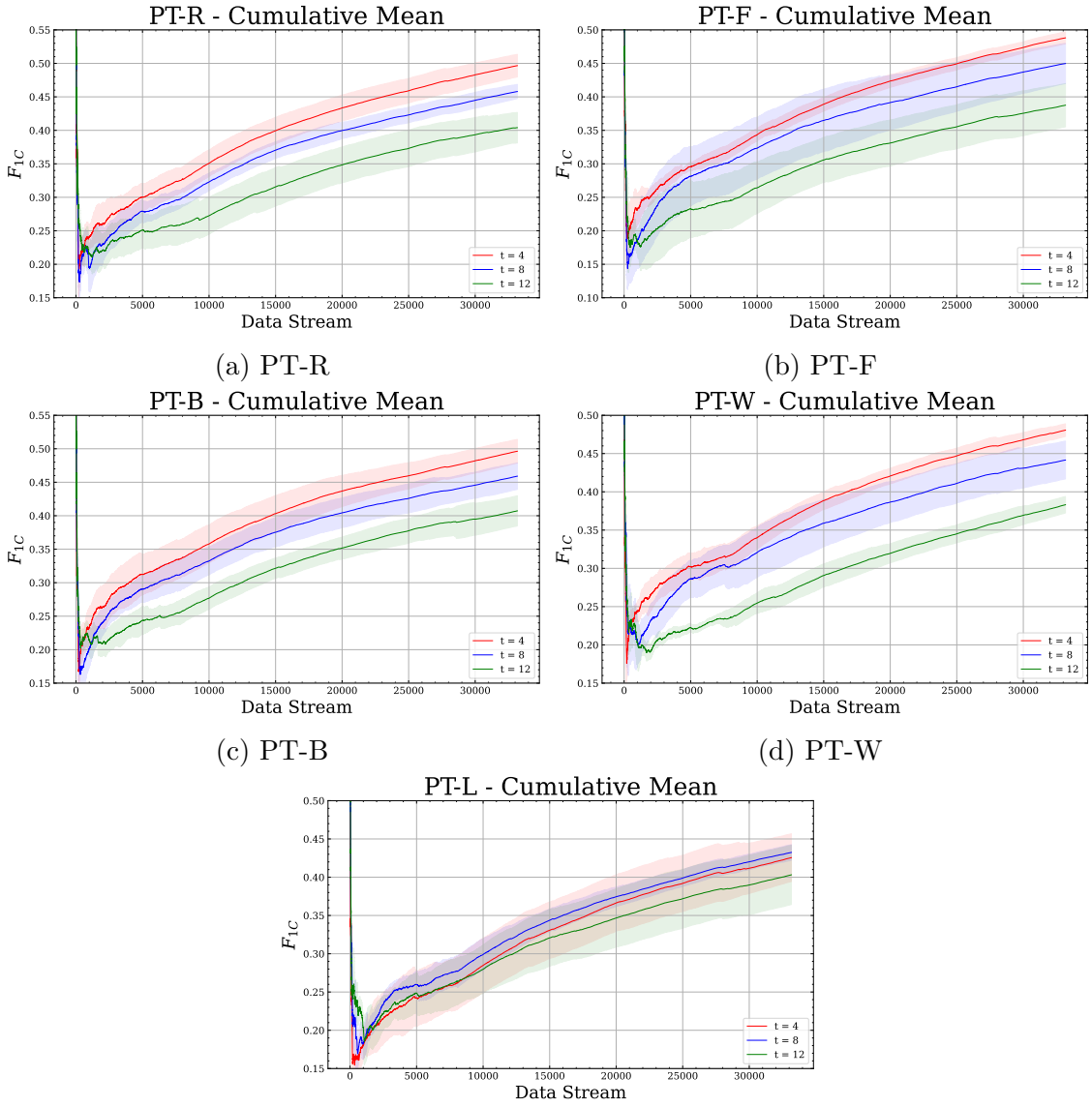
Figure 4.7: Critical values a_C as a function of test size. Standard deviation is calculated from the $N = 5$ repeated simulations.



(a) Cumulative F_{1C} mean of full retrain (FR) algorithm.

(b) Moving F_{1C} mean with window size $W = 500$.

Figure 4.8: Performance of FR measuring cumulative mean (left) and moving mean (right).



(a) PT-R

(b) PT-F

(c) PT-B

(d) PT-W

(e) PT-L

Figure 4.9: Cumulative F_{1C} mean of different PT methods.

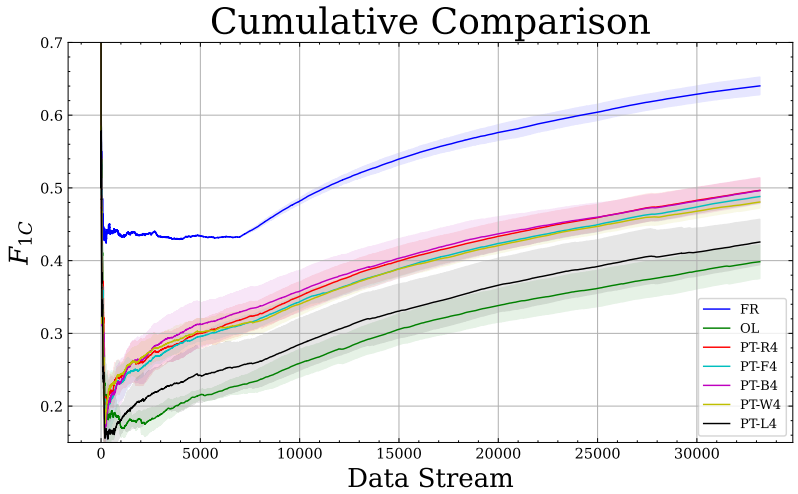


Figure 4.10: Comparison between FR, OL and PT-4 methods.

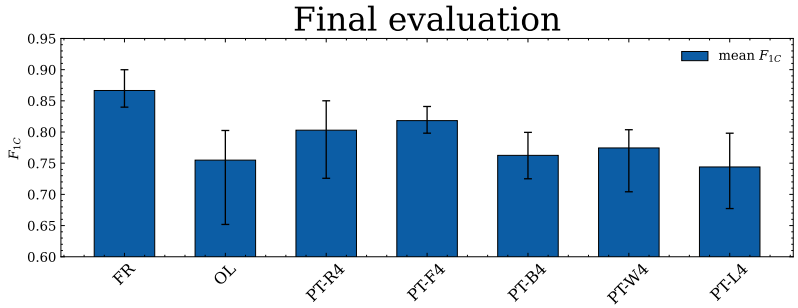


Figure 4.11: Mean F_{1C} over 5 repeated runs for FR, OL and PT-4. Error bar indicates max and min evaluation in the runs for each respective method.

5

Discussion

In this chapter, the overall findings of the thesis are discussed. First, general remarks that applied to the entire process of the thesis are discussed in Section 5.1. The discussion focused on the obtained results for task 1 and task 2 are presented in Sections 5.2 and 5.3 respectively. From there, an interpretation of how the obtained results can be applied to industries, such as GKN, is given in Section 5.5. Further work and relevant additions from where the thesis project can be branched out are discussed in Section 5.6. Lastly, the thesis is concluded in Section 5.7.

5.1 General remarks and drawbacks

Initially when experimenting with different configurations for the training process, alternative loss functions such as Jacard loss [47] and regular binary cross entropy (BCE) were tested. These two loss functions had the tendency to make the model mode collapse, i.e converge to suboptimal solutions that exploit certain properties of the data. When training a model using Jacard loss, the model converged on solutions that would predict a large portion of the output image as white (defects). Because the Jacard loss is a intersection-based loss function it makes sense that the model could be prone to predicting almost the entire image as white to achieve some intersection with the target mask. On the opposite end, training a model using regular BCE loss, resulted most of the time in a model that predicts a completely black image regardless of input. This is most likely due to the fact that there is an extreme imbalance between the white and black pixels. The solution was to use a weighted BCE loss that would assign a higher loss if the model miss-classifies white pixels compared to black pixels.

Another issue that was found during the training process was that the model had a tendency to *catastrophically forget* its learned representation which would cause a significant drop in validation F_{1C}, roughly by a factor of 2, compared to the previous epoch. This effect would mostly occur in the earlier epochs during training and it would take around 4-8 epochs for the model to recuperate back to its prior performance. Normally for supervised tasks, the performance metrics are approximately monotonically increasing for every epoch until they plateau. This issue was partially solved by adding check pointing at the end of every epoch where the weights of the best model throughout the training process are saved. Then, when the training is finished, the weights corresponding to the best model are loaded in.

5.2 Task 1 - Domain validation

From the HPO simulations, it is clear that a non-unit batch size ($B \neq 1$) leads to a better trained network. In comparison, the original U-Net paper [35] used batches consisting of a single image ($B = 1$) in combination with high momentum Stochastic Gradient Descent (SGD). Although not specified in their paper, one would assume this training setup was the most optimal for their model. The Adam optimizer used in this thesis, which is an adaptive version of SGD is not expected to show this type of discrepancy in optimal optimizer settings. Further investigation is necessary to determine what gives rise to this difference in preferred batch size B.

The generalization capabilities between intra-domain data sets are, from the obtained results, very poor as the predictions showcase significant deviations with a mean average error of $E_{mae} = 0.11$. From the available image data archived from GKN's inspection, the X-ray and AVI were found to be the most similar ones. This similarity was based on our personal eye judgement. It is entirely possible that these data sets differ quite significantly on a mathematical representation level which would explain the large difference between the estimated learning curves. Another reason for the deviation can be attributed to the quality of the data in the X-ray images, which is discussed further in Section 5.4.

The curve fitting method presented in [33] where later data points are weighted with higher weight, did not perform better than standard least square optimization when fitting a learning curve from the simulated data, as shown in [33]. The difference between the 3 methods was insignificant and all methods are considered valid for estimating the learning curves for both the X-ray and AVI data sets using only a very small initial gathered data set. The curve fit from the non HPO experiment had substantially lower validation error which is most likely due to stability from fixed hyperparameters.

5.3 Task 2 - Continuous improvement

From the simulated timeline in task 2, it is clear from the results that full retraining (FR) is the best method both based on the prequential evaluation and the final evaluation. One concerning effect that is shown in cumulative comparison figure (Figure 4.10), is how PT and OL instantly deviate from FR from the start when all methods are initialized using the baseline model. There is a possible reasons for this large gap in starting performance.

The most plausible reason is that the Adam optimizer is causing the initial dip in the performance of OL and PT methods. Adaptive optimizers used for training neural networks are technically not parameterless as they store statistics about the data set during training. This is completely fine for FR as the optimizer is initialized from scratch for every new retrain. However, for PT and OL, that use the initial baseline model and its corresponding optimizer will be affected by this since the optimizer's state parameters are computed from the initial training data of 1000 samples. Once new data is introduced to the training through the timeline, the mismatch between

the optimizer's state statistics and the new data leads to a temporary performance drop. One small detail to note is that although trigger size $t = 12$ resulted in the worse cumulative F_{1C}, it did have a slightly smaller initial dip. This is most likely due to the fact that the batches contain a higher percentage of new data, which makes the optimizer adapt faster to the change in training data.

Prapas et al. [1] examined a similar effect with an initial drop-off in performance which they also attributed to the optimizer function. They mention a possible solution called *warm starting* of the optimizer but do not clarify further what that specifically entails.

The trigger sizes t that were selected so each batch b would contain 25%, 50% and 75% new incoming data. One of the big reasons for the discrepancy between the FR and the other methods is most likely due to the considerable difference in the number of batch updates for the different methods. For FR the first retrain triggers a training from scratch using 8k samples of which 80% are used as the train set D_{tr} and the remaining 20% is used as the validation set D_{val}. Given a batch size of $B = 16$ and 50 training epochs, corresponds to $N = 50 \cdot 0.8 \cdot 8000/16 \approx 20k$ batch updates. Already at the first retrain, FR will train a model that is updated more than twice compared to the fully trained PT-4 methods. To make the comparison between FR and PT more fair one would need to consider smaller trigger sizes t. It is also possible to add resampling where the model is updated using N batches each time PT is triggered, by using the t new samples and repeatedly sampling the $(B - t)$ samples from the history set $\mathcal{H}$ for the N different batches.

These continuous improvement methods were investigated separately. There is some merit in combining FR with PT methods. A hybrid retraining method that continuously updates the model once t new samples arrive and performs a full retraining from scratch once C_{FR} new samples arrive. This method would bring benefits from both methods as FR utilizes the available data in the most optimal way and PT methods would give an additional performance boost during the time period between full retrainings. For this method to be valid one would need to find the exact cause that leads to a performance drop, because otherwise the model would deteriorate immediately after FR is triggered, which is unwanted in deployment.

5.4 Quality of data set

Some caution should be taken when interpreting the numerical results as the data sets used during the model training are far from perfect. The annotation of the X-ray data was done in assistance with an already existing U-Net model, by letting it predict on images and processing its outputs. The output of the pre-trained model was then manually refined by a human, through removing false positives and adding annotations for false negatives which then would constitute the finished target mask $\mathcal{T}$. This data gathering-process is thus subject to two sources of error, model error and human error. Unfortunately, when manually inspecting a handful of image predictions, it became apparent that a significant portion of the false positives were due to missing annotations instead of model error, see Figure 5.1 for an example of

this phenomena.

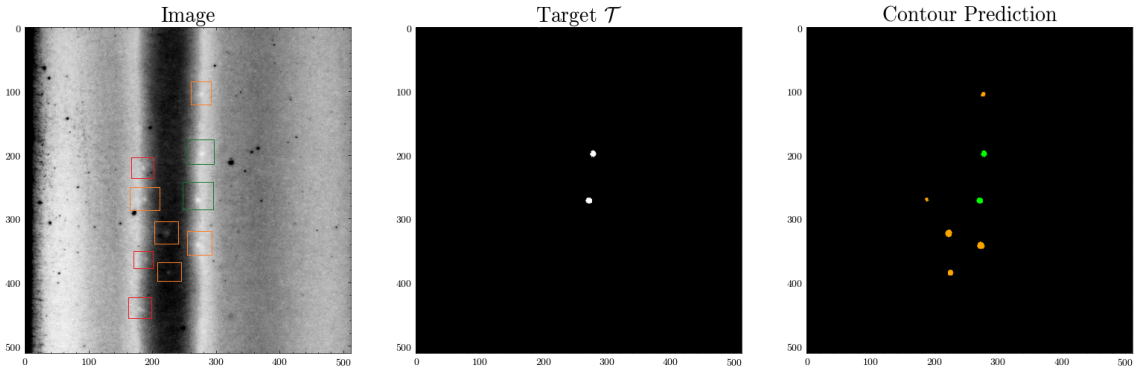


Figure 5.1: Example of input image (left) with its respective target task $\mathcal{T}$ (center). The defects are highlighted with bounding boxes where green, orange and red corresponding to true positives (TP), false positives (FP) and false negative (FN) respectively. The model's contour prediction (right) shows the overlap between the prediction and target mask. All the FP actually correspond to real defects in the input image but due to poor annotations, this leads to some predictions being incorrectly classified.

The U-Net could still confidently learn to find defects, which is expected since machine learning models are robust and can handle a certain amount of mislabeled data without losing significant generalization performance. This issue was not replicated for the AVI data as it is generated from an augmentation process and because each image in the AVI set only had one defect. However, for the X-ray metrics, one should assume that the evaluations are pessimistically biased due to inflated numbers of false positives. This reduced quality in data is believed to be partially answerable to the 0.11 MAE observed when comparing the learning curves for the two data sets.

5.5 Application of results to industry

Here, the results for both tasks are combined to formulate guidelines when developing a ML model for manufacturing inspection tasks.

Assume that in the future some new aerospace component is being added to the manufacturing line. The inspection of this component generates image data so there is a need to automate the defect detection process through a ML model. From this standpoint one is assigned the task of estimating how much data needs to be manually annotated and gathered before training a model. The estimated size will depend on some desired level of performance, which should be assessed during the initial stages of development. The assessment of performance should take into account the potential benefits that comes with automating, or assisting, the inspection process and the potential risks of the model missing certain defects.

The results from task 1 indicate that one can not naively use the learning curves

from prior gathered data, to make predictions for new inspection tasks. Instead it is recommended to fit an inverse power law curve to some desired performance metric f as this was shown to be very accurate for F_{1C}, which was the main considered metric used in this thesis. Unfortunately, to fit a curve one needs some initial data to train the model with varying sizes of training data. The size of the initial data set is assumed to be substantially smaller than the necessary training size needed to deploy the model into industry. In this thesis the initial size corresponds to approximately 2.5-3k cropped images (same split s was used for fit but due to X-ray and AVI data sets being different, the size for fit was slightly different). This data is split up intro train, validation and test data according to a 80/10/10 split.

From this initial data, one repeatedly trains a model by varying the available data similar to the method used in task 1. Since the model is trained on relatively small sizes of data, one can get away with a significantly finer split resolution, compared to those in task 1, without running into excessive computational times. This should lead to an even better curve fit as the split resolution in task 1 was limited to not make the computational time too long. From the fitted curve one can get an accurate estimate of how much training data is needed to achieve a certain performance.

Once deployed into industry, one needs to continuously improve the model using incoming data. From the estimated learning curve one can set an improvement criteria Δf. This criteria is triggered whenever Δx new data samples arrive such that

$$f^*(x + \Delta x) - f^*(x) \geq \Delta f$$

where x is the currently available training data and f^* is the estimated learning curve, see Figure 5.2 for illustration of when the retraining is triggered.

5.6 Further work

5.6.1 U-Net architecture

The original U-Net model [35] that was used to segment biomedical images, uses central cropping in its residual connection. This is due to the lack of padding in its convolutional layers. The model used in this thesis instead propagates the entire feature map and uses zero-padding in the convolutional layers to keep the shape of the feature maps fixed. This is more intuitive since there exists a 1:1 relation between pixels of the input image and the model's output map $\mathcal{P}$. Ronneberger et al. [35] never justify their choice of the unpadded convolutions, however, it can be argued that using zero padding can introduce numerical boundary effects on the border of feature maps [48]. Whether this can cause singular pixels near the border to be classified as defects or, conversely, prevent the model from detecting defects near the edge needs to be studied. This however, will not pose a major problem in practice as the image crops that are extracted from the full-sized X-ray image ($\approx 7000 \times 7000$ px) will have some overlap making any potential edge effects manageable.

The complexity of the model was dictated by the complexity parameter $v \in \mathbb{N}$ which

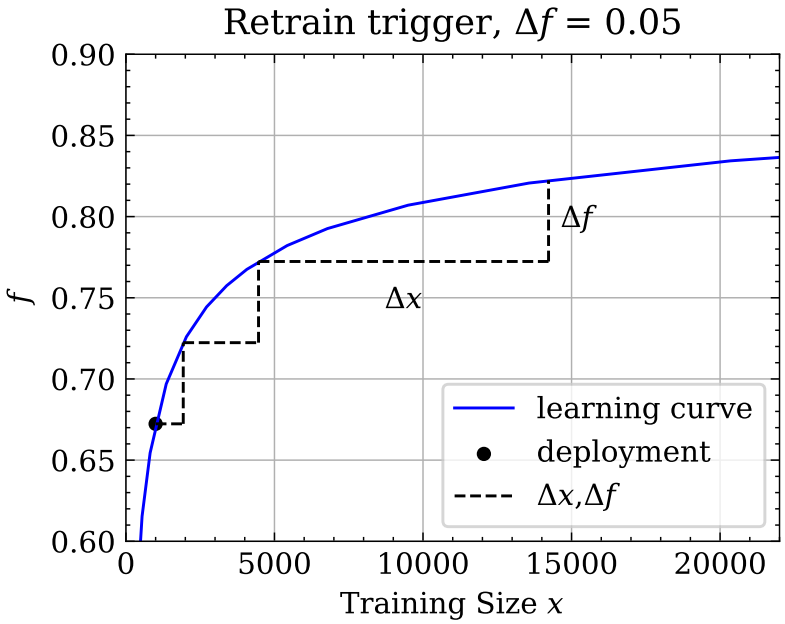


Figure 5.2: An initial model trained on $x = 1000$ training samples is deployed into industry. A full retrain (FR) is triggered whenever enough new data Δx arrives that according to the estimated curve fit will yield an performance increase of Δf. Here the trigger is set to $\Delta f = 0.05$, however, realistically it would be lower.

scaled the number of feature maps for each layer by a factor v. This resulted in a very coarse control of the model complexity, partly due to the fact that the architecture of the unscaled U-Net is already large and partly due to the fact that the number of model parameters grows as $\mathcal{O}(v^2)$. There is thus a need to investigate alternative U-Net architectures, cause it is not evident that the used model architecture is optimal for the designated task. This can be done in addition to changing the parameter v, by also changing the depth of the model by varying the number of convolutional layers.

5.6.2 Alternative network architectures

This thesis restricted the neural network model to a U-Net model. It was argued through [37] that U-Nets are suitable for defect detection tasks as they output a binary segmentation where the pixels are classified as either defect (white) or background (black). There are numerous different network architectures that have been applied and demonstrated promising for defect detection tasks.

You Only Look Once (YOLO) is a CNN model that has gained massive popularity due to its applications in real-time object detection [49]. Modifications to the original YOLOv1 model have led to the introduction of improved and case-specific YOLO models [50]. YOLO models, in comparison to the U-Net, predict bounding boxes around objects of interest. These bounding boxes are defined by the pixel coordinates (x, y) for the box's center and w, h as the respective width and height. One downside with bounding box predictions is that separate post-processing meth-

ods are required to determine the dimensions of the detected objects. Despite this, a YOLOv4 model has been used to detect defects from wire arc additive manufacturing on metal surfaces [51].

Region-based convolutional neural networks (R-CNN) are two-stage object detection models [52]. The first stage proposes candidate boxes in the input image and the second stage, consisting of a CNN, determines whether there exists and object in the candidate box. In comparison to YOLO-based models, R-CNNs have, in general higher detection accuracy but are less suitable for domains where real-time detection is necessary because R-CNN's complex pipeline results in relatively slow inference speeds. In [5], a faster R-CNN model is used in an industrial case study to detect defects on turbo blades in car engines, achieving a F1-score $F_1 = 0.76$.

Both the YOLO based and R-CNNs models give bounding boxes as output predictions from which one can establish precision, recall and in turn a F1-score F_1 comparable to the contour F1-score F_{1C} used to evaluate the U-Net. Whether these networks require more or less data to train a high-performing model is unclear and needs to be investigated using a similar methodology as described in this thesis to establish learning curves for the respective models.

5.6.3 Monitoring system and data drift detection

In addition to continuously improving the deployed model, a monitoring system is essential to detect deterioration in model performance or unexpected outlier behaviour. The deployed model will continually make predictions on incoming data.

The assumption that the properties of the incoming remains constant is often not true for real-world data and changes to the data generating process distribution p_{data} is referred to as data drift and can negatively effect the model [53], [54]. These changes can either be gradual, abrupt, or oscillating and some previous effort has been done to standardize the terminology related to concept drift [55]. Concept drift can lead to model staleness, meaning the data distribution p_{data} deviates from the original representations that the model learns during training. Standard methods of investigating concepts drift consists of measuring data distribution statistics of subsamples and compare it to subsamples from older data using similarity measures, such as distance metrics between distributions [56].

Alternatively to monitoring the input data, methods exist that monitor the model's performance during a window of recent time. Statistical process charts are a monitoring method that based on predefined rules will trigger an alert if the behaviour of the model's predictive performance changes drastically over a short span of time [57]. The triggered alert can then be used to assess whether the model needs to be retrained or to detect potential outlier behaviour in the inspection pipeline. Normal patterns of concern consists of observations that deviate from a moving mean.

In this thesis we ignored the potential existence of concept drift in our data sets, but due to X-ray data being gathered over the span of 6 years, it is thus possible that some detectable drift can be found in the data between years. One limiting

factor of the X-ray data is that the time property of each image is only by year and no finer time scale descriptions exists.

5.7 Conclusion

This thesis aimed at solving two partial problems that arise in smart manufacturing settings relevant to defect detection. The first task was to investigate the validity of using the performance characteristic, such as an empirically estimated learning curve, from one data set to predict the expected performance for another, yet similar data set, given that the remaining structure of the pipeline remains fixed. This assumption showed to be inaccurate when comparing the results from X-ray and using them to predict performance for the AVI data. Instead, it is advised to gather an initial data set from where one can repeatedly train a model with varying sizes of training data. These results can then be used to fit an inverse power law using standard curve fitting methods to predict the relation between expected model performance and training data (learning curve). From this learning curve it is possible to get an accurate estimate of training data.

For the second task, different methods of continuously improving a deployed ML model as new incoming data is generated was compared. The obtained results strongly indicate that full retraining (FR) significantly outperforms both online learning (OL) and proactive training (PT) based methods. Although one would need to investigate more aggressive PT methods with very low trigger sizes t before completely ruling out PT based methods.

Bibliography

[1] I. Prapas, B. Derakhshan, A. R. Mahdiraji, and V. Markl, "Continuous training and deployment of deep learning models," *Datenbank-Spektrum*, vol. 21, no. 3, pp. 203–212, 2021. DOI: 10.1007/s13222-021-00386-8.

[2] S. Vaidya, P. Ambad, and S. Bhosle, "Industry 4.0 a glimpse," *Procedia Manufacturing*, vol. 20, pp. 233–238, 2018, 2nd International Conference on Materials, Manufacturing and Design Engineering (iCMMD2017), 11-12 December 2017, MIT Aurangabad, Maharashtra, INDIA, ISSN: 2351-9789. DOI: https://doi.org/10.1016/j.promfg.2018.02.034. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2351978918300672.

[3] F. Bachinger, G. Kronberger, and M. Affenzeller, "Continuous improvement and adaptation of predictive models in smart manufacturing and model management," *IET Collaborative Intelligent Manufacturing*, vol. 3, no. 1, pp. 48–63, 2021. DOI: 10.1049/cim2.12009.

[4] J. Schmitt, J. Bönig, T. Borggräfe, G. Beitinger, and J. Deuse, "Predictive model-based quality inspection using machine learning and edge cloud computing," *Advanced Engineering Informatics*, vol. 45, p. 101 101, 2020. DOI: 10.1016/j.aei.2020.101101.

[5] Y. Wang, M. Liu, P. Zheng, H. Yang, and J. Zou, "A smart surface inspection system using faster r-cnn in cloud-edge computing environment," *Advanced Engineering Informatics*, vol. 43, p. 101 037, 2020. DOI: 10.1016/j.aei.2020.101037.

[6] J. Schmitt, J. Bönig, T. Borggräfe, G. Beitinger, and J. Deuse, "Predictive model-based quality inspection using machine learning and edge cloud computing," *Advanced Engineering Informatics*, vol. 45, p. 101 101, 2020. DOI: 10.1016/j.aei.2020.101101.

[7] T. Yang, T.-N. Tsai, and J. Yeh, "A neural network-based prediction model for fine pitch stencil-printing quality in surface mount assembly," *Engineering Applications of Artificial Intelligence*, vol. 18, no. 3, pp. 335–341, 2005. DOI: 10.1016/j.engappai.2004.09.004.

[8] G. Susto, S. Pampuri, A. Schirru, G. De Nicolao, and S. McLoone, "Automatic control and machine learning for semiconductor manufacturing: Review and challenges," English, 10th European Workshop on Advanced Control and Diagnosis 2012 ; Conference date: 08-10-2012 Through 09-10-2012, 2012.

[9] V. Natarajan, T.-Y. Hung, S. Vaikundam, and L.-T. Chia, "Convolutional networks for voting-based anomaly classification in metal surface inspection,"

2017 IEEE International Conference on Industrial Technology (ICIT), 2017. DOI: 10.1109/icit.2017.7915495.

[10] Z. Huang, J. Wu, and F. Xie, "Automatic recognition of surface defects for hot-rolled steel strip based on deep attention residual convolutional neural network," *Materials Letters*, vol. 293, p. 129 707, 2021. DOI: 10.1016/j.matlet.2021.129707.

[11] M. Kano and Y. Nakagawa, "Data-based process monitoring, process control, and quality improvement: Recent developments and applications in steel industry," *Computers amp; Chemical Engineering*, vol. 32, no. 1-2, pp. 12–24, 2008. DOI: 10.1016/j.compchemeng.2007.07.005.

[12] K. Moran, *Benefits of industry 4.0*, Sep. 2022. [Online]. Available: https://slcontrols.com/en/benefits-of-industry-4-0/.

[13] M. Sony, J. Antony, O. Mc Dermott, and J. A. Garza-Reyes, "An empirical examination of benefits, challenges, and critical success factors of industry 4.0 in manufacturing and service sector," *Technology in Society*, vol. 67, p. 101 754, 2021. DOI: 10.1016/j.techsoc.2021.101754.

[14] R. van der Meulen, *Gartner says nearly half of cios are planning to deploy artificial intelligence*, 2018. [Online]. Available: https://www.gartner.com/en/newsroom/press-releases/2018-02-13-gartner-says-nearly-half-of-cios-are-planning-to-deploy-artificial-intelligence.

[15] J. Weiner, *Why AI/Data Science Projects Fail: How to avoid project pitfalls*. Springer International Publishing, 2021.

[16] *About gkn aerospace: Gkn aerospace*. [Online]. Available: https://www.gknaerospace.com/en/about-gkn-aerospace/.

[17] P. N. Edwards, M. S. Mayernik, A. L. Batcheller, G. C. Bowker, and C. L. Borgman, "Science friction: Data, metadata, and collaboration," *Social Studies of Science*, vol. 41, no. 5, pp. 667–690, 2011. DOI: 10.1177/0306312711413314.

[18] S. Rogers and M. Girolami, *A first course in machine learning*. Chapman amp; Hall/CRC, 2020.

[19] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MITP, 2018.

[20] D. of Defense Handbook, *MIL-HDBK-1823: Nondestructive Evaluation System Reliability Assessment*, Apr. 2009.

[21] J. Awwalu and F. Ogwueleka, "On holdout and cross validation: A comparison between neural network and support vector machine," vol. 6, pp. 2394–9333, Apr. 2019.

[22] B. J. Wythoff, "Backpropagation neural networks: A tutorial," *Chemometrics and Intelligent Laboratory Systems*, vol. 18, no. 2, pp. 115–155, 1993, ISSN: 0169-7439. DOI: https://doi.org/10.1016/0169-7439(93)80052-J. [Online]. Available: https://www.sciencedirect.com/science/article/pii/016974399380052J.

[23] A. P. Dawid, "Present position and potential developments: Some personal views: Statistical theory: The prequential approach," *Journal of the Royal Statistical Society. Series A (General)*, vol. 147, no. 2, p. 278, 1984. DOI: 10.2307/2981683.

[24] "Results on learnability and the vapnik-chervonenkis dimension," *Information and Computation*, vol. 90, no. 1, pp. 33–49, 1991, ISSN: 0890-5401. DOI: https://doi.org/10.1016/0890-5401(91)90058-A.

[25] P. L. Bartlett, N. Harvey, C. Liaw, and A. Mehrabian, *Nearly-tight vc-dimension and pseudodimension bounds for piecewise linear neural networks*, 2017. arXiv: 1703.02930 [cs.LG].

[26] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth, "Learnability and the vapnik-chervonenkis dimension," *Journal of the ACM*, vol. 36, no. 4, pp. 929–965, 1989. DOI: 10.1145/76359.76371.

[27] V. Vapnik, *Estimation of dependences based on empirical data*. Springer-Verlag, 1982.

[28] V. N. Vapnik, *The nature of statistical learning theory*. Springer, 1995.

[29] I. Balki, A. Amirabadi, J. Levman, *et al.*, "Sample-size determination methodologies for machine learning in medical imaging research: A systematic review," *Canadian Association of Radiologists Journal*, vol. 70, no. 4, pp. 344–353, 2019. DOI: 10.1016/j.carj.2019.06.002.

[30] E. B. Baum and D. Haussler, "What size net gives valid generalization?" *Neural Computation*, vol. 1, no. 1, pp. 151–160, 1989. DOI: 10.1162/neco.1989.1.1.151.

[31] S. Haykin and S. Hakin, *The Neural Networks a comprehensive foundation*. 2nd ed., 1998.

[32] C. Cortes, L. D. Jackel, S. Solla, V. Vapnik, and J. Denker, "Learning curves: Asymptotic values and rate of convergence," in *Advances in Neural Information Processing Systems*, J. Cowan, G. Tesauro, and J. Alspector, Eds., vol. 6, Morgan-Kaufmann, 1993. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/1993/file/1aa48fc4880bb0c9b8a3bf979d3b917e-Paper.pdf.

[33] R. L. Figueroa, Q. Zeng-Treitler, S. Kandula, and L. H. Ngo, "Predicting sample size required for classification performance," *BMC Medical Informatics and Decision Making*, vol. 12, no. 1, 2012. DOI: 10.1186/1472-6947-12-8.

[34] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu, *A survey on deep transfer learning*, 2018. arXiv: 1808.01974 [cs.LG].

[35] O. Ronneberger, P. Fischer, and T. Brox, *U-net: Convolutional networks for biomedical image segmentation*, 2015. arXiv: 1505.04597 [cs.CV].

[36] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017. DOI: 10.1145/3065386.

[37] T. Bergs, C. Holst, P. Gupta, and T. Augspurger, "Digital image processing with deep learning for automated cutting tool wear detection," *Procedia Manufacturing*, vol. 48, pp. 947–958, 2020, 48th SME North American Manufacturing Research Conference, NAMRC 48, ISSN: 2351-9789. DOI: https://doi.org/10.1016/j.promfg.2020.05.134. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2351978920315869.

[38] A. P. Berens and P. W. Hovey, "Characterization of nde reliability," *Review of Progress in Quantitative Nondestructive Evaluation*, pp. 579–586, 1982. DOI: 10.1007/978-1-4684-4262-5_74.

[39] P. R. Underhill, T. W. Krause, D. O. Thompson, and D. E. Chimenti, "Measurement of uncertainty in eddy current bolt hole crack measurements for use in pod," *AIP Conference Proceedings*, 2010. DOI: 10.1063/1.3362341.

[40] M. R. Bato, A. Hor, A. Rautureau, and C. Bes, "Experimental and numerical methodology to obtain the probability of detection in eddy current ndt method," *NDT E International*, vol. 114, p. 102 300, 2020, ISSN: 0963-8695. DOI: https://doi.org/10.1016/j.ndteint.2020.102300. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0963869518301166.

[41] D. Forsyth and J. C. Aldrin, "Build your own pod: New and improved version (conference presentation)," *8th International Workshop on Reliability of NDT/NDE*, 2023. DOI: 10.1117/12.2657453.

[42] C. Henry, C. S. Kabban, M. Cherry, and R. Mooers, "Ranked set sampling applied to hit-miss probability of detection data: A case study," *AIP Conference Proceedings*, 2019. DOI: 10.1063/1.5099820.

[43] F. W. Spencer, D. O. Thompson, and D. E. Chimenti, "Nonparametric pod estimation for hitmiss data: A goodness of fit comparison for parametric models," *AIP Conference Proceedings*, 2011. DOI: 10.1063/1.3592115.

[44] F. W. Spencer, "The calculation and use of confidence bounds in pod models," *AIP Conference Proceedings*, 2007. DOI: 10.1063/1.2718181.

[45] B. Derakhshan, A. R. Mahdiraji, T. Rabl, and V. Markl, "Continuous deployment of machine learning pipelines.," in *EDBT*, 2019, pp. 397–408.

[46] K. Hajian-Tilaki, *Receiver operating characteristic (roc) curve analysis for medical diagnostic test evaluation*, 2013. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3755824/.

[47] J. Bertels, T. Eelbode, M. Berman, *et al.*, "Optimizing the dice score and jaccard index for medical image segmentation: Theory and practice," in *Lecture Notes in Computer Science*, Springer International Publishing, 2019, pp. 92–100. DOI: 10.1007/978-3-030-32245-8_11. [Online]. Available: https://doi.org/10.1007%2F978-3-030-32245-8_11.

[48] M. A. Islam, M. Kowal, S. Jia, K. G. Derpanis, and N. Bruce, *Boundary effects in {cnn}s: Feature or bug?* 2021. [Online]. Available: https://openreview.net/forum?id=M4qXqdw3xC.

[49] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, *You only look once: Unified, real-time object detection*, 2016. arXiv: 1506.02640 [cs.CV].

[50] J. Terven and D. Cordova-Esparza, *A comprehensive review of yolo: From yolov1 and beyond*, 2023. arXiv: 2304.00501 [cs.CV].

[51] W. Li, H. Zhang, G. Wang, *et al.*, "Deep learning based online metallic surface defect detection method for wire and arc additive manufacturing," *Robotics and Computer-Integrated Manufacturing*, vol. 80, p. 102 470, 2023, ISSN: 0736-5845. DOI: https://doi.org/10.1016/j.rcim.2022.102470. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0736584522001521.

[52] R. Girshick, J. Donahue, T. Darrell, and J. Malik, *Rich feature hierarchies for accurate object detection and semantic segmentation*, 2014. arXiv: 1311.2524 [cs.CV].

[53] J. Gama, I. liobait, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM Computing Surveys*, vol. 46, no. 4, pp. 1–37, 2014. DOI: 10.1145/2523813.

[54] S. M. Jameel, M. A. Hashmani, H. Alhussain, M. Rehman, and A. Budiman, "A critical review on adverse effects of concept drift over machine learning classification models," *International Journal of Advanced Computer Science and Applications*, vol. 11, no. 1, 2020. DOI: 10.14569/IJACSA.2020.0110127. [Online]. Available: http://dx.doi.org/10.14569/IJACSA.2020.0110127.

[55] G. I. Webb, R. Hyde, H. Cao, H. L. Nguyen, and F. Petitjean, "Characterizing concept drift," *Data Mining and Knowledge Discovery*, vol. 30, no. 4, pp. 964–994, Apr. 2016. DOI: 10.1007/s10618-015-0448-4. [Online]. Available: https://doi.org/10.1007%5C%2Fs10618-015-0448-4.

[56] B. Eck, D. Kabakci-Zorlu, Y. Chen, F. Savard, and X. Bao, "A monitoring framework for deployed machine learning models with supply chain examples," *2022 IEEE International Conference on Big Data (Big Data)*, 2022. DOI: 10.1109/bigdata55660.2022.10020394.

[57] M. H. Omar, "Statistical process control charts for measuring and monitoring temporal consistency of ratings," *Journal of Educational Measurement*, vol. 47, no. 1, pp. 18–35, 2010. DOI: 10.1111/j.1745-3984.2009.00097.x.

A

Appendix

A.1 U-net layer architecture

Layer architecture of the U-Net model. Output is generated using built in function *summary()* from *torchinfo* python package with input tensor shape $(1, 1, 512, 512)$ = (batch size, channels, height, width).

Layer (type:depth-idx)	Output Shape	Param #
Unet	[1, 1, 512, 512]	--
Conv2d: 1-1	[1, 16, 512, 512]	160
ReLU: 1-2	[1, 16, 512, 512]	--
Dropout: 1-3	[1, 16, 512, 512]	--
Conv2d: 1-4	[1, 16, 512, 512]	2,320
ReLU: 1-5	[1, 16, 512, 512]	--
BatchNorm2d: 1-6	[1, 16, 512, 512]	32
MaxPool2d: 1-7	[1, 16, 256, 256]	--
Conv2d: 1-8	[1, 32, 256, 256]	4,640
ReLU: 1-9	[1, 32, 256, 256]	--
Dropout: 1-10	[1, 32, 256, 256]	--
Conv2d: 1-11	[1, 32, 256, 256]	9,248
ReLU: 1-12	[1, 32, 256, 256]	--
BatchNorm2d: 1-13	[1, 32, 256, 256]	64
MaxPool2d: 1-14	[1, 32, 128, 128]	--
Conv2d: 1-15	[1, 64, 128, 128]	18,496
ReLU: 1-16	[1, 64, 128, 128]	--
Dropout: 1-17	[1, 64, 128, 128]	--
Conv2d: 1-18	[1, 64, 128, 128]	36,928
ReLU: 1-19	[1, 64, 128, 128]	--
BatchNorm2d: 1-20	[1, 64, 128, 128]	128
MaxPool2d: 1-21	[1, 64, 64, 64]	--
Conv2d: 1-22	[1, 128, 64, 64]	73,856
ReLU: 1-23	[1, 128, 64, 64]	--
Dropout: 1-24	[1, 128, 64, 64]	--
Conv2d: 1-25	[1, 128, 64, 64]	147,584

```
ReLU: 1-26                    [1, 128, 64, 64]          --
BatchNorm2d: 1-27             [1, 128, 64, 64]          256
MaxPool2d: 1-28               [1, 128, 32, 32]          --
Conv2d: 1-29                  [1, 256, 32, 32]          295,168
ReLU: 1-30                    [1, 256, 32, 32]          --
Dropout: 1-31                 [1, 256, 32, 32]          --
Conv2d: 1-32                  [1, 256, 32, 32]          590,080
ReLU: 1-33                    [1, 256, 32, 32]          --
BatchNorm2d: 1-34             [1, 256, 32, 32]          512
ConvTranspose2d: 1-35         [1, 128, 64, 64]          131,200
Conv2d: 1-36                  [1, 128, 64, 64]          295,040
ReLU: 1-37                    [1, 128, 64, 64]          --
Dropout: 1-38                 [1, 128, 64, 64]          --
Conv2d: 1-39                  [1, 128, 64, 64]          147,584
ReLU: 1-40                    [1, 128, 64, 64]          --
ConvTranspose2d: 1-41         [1, 64, 128, 128]         32,832
Conv2d: 1-42                  [1, 64, 128, 128]         73,792
ReLU: 1-43                    [1, 64, 128, 128]         --
Dropout: 1-44                 [1, 64, 128, 128]         --
Conv2d: 1-45                  [1, 64, 128, 128]         36,928
ReLU: 1-46                    [1, 64, 128, 128]         --
ConvTranspose2d: 1-47         [1, 32, 256, 256]         8,224
Conv2d: 1-48                  [1, 32, 256, 256]         18,464
ReLU: 1-49                    [1, 32, 256, 256]         --
Dropout: 1-50                 [1, 32, 256, 256]         --
Conv2d: 1-51                  [1, 32, 256, 256]         9,248
ReLU: 1-52                    [1, 32, 256, 256]         --
ConvTranspose2d: 1-53         [1, 16, 512, 512]         2,064
Conv2d: 1-54                  [1, 16, 512, 512]         4,624
ReLU: 1-55                    [1, 16, 512, 512]         --
Dropout: 1-56                 [1, 16, 512, 512]         --
Conv2d: 1-57                  [1, 16, 512, 512]         2,320
ReLU: 1-58                    [1, 16, 512, 512]         --
Conv2d: 1-59                  [1, 1, 512, 512]          17
==========================================
Total params: 1,941,809
Trainable params: 1,941,809
Non-trainable params: 0
Total mult-adds (G): 13.71
==========================================
Input size (MB): 1.05
Forward/backward pass size (MB): 385.88
Params size (MB): 7.77
Estimated Total Size (MB): 394.69
==========================================
```

A.2 Additional Performance metrics for task 1

As previously mentioned, the contour F1-score F_{1C} is the ideal metric to optimize and consider in an industrial setting which is presented in the report. Here we present the remaining evaluation metrics that were logged during task 1 for the interested reader. The following metrics are shown for each of the 3 different runs; pixel precision P, pixel recall R, pixel F_1, contour precision P_C, contour recall R_C and contour accuracy A_C defined as $A_C = \frac{TP}{TP+FP+FN}$ (remember TN are not defined for contours).

A.2.1 X-ray (HPO)

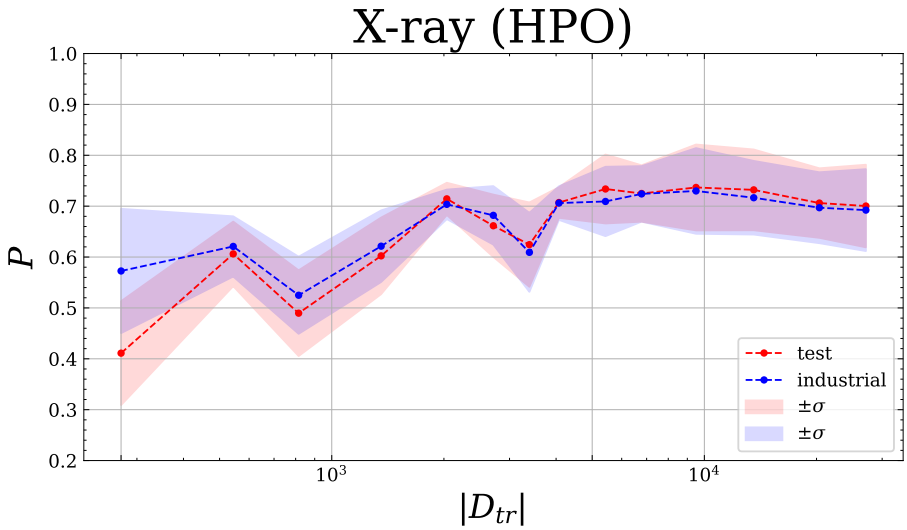


Figure A.1: X-ray (HPO) pixel precision P.

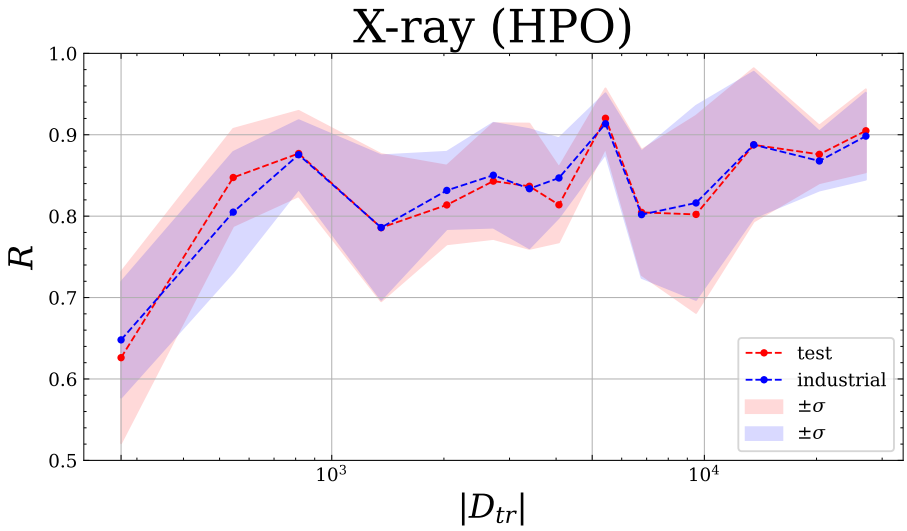


Figure A.2: X-ray (HPO) pixel recall R.

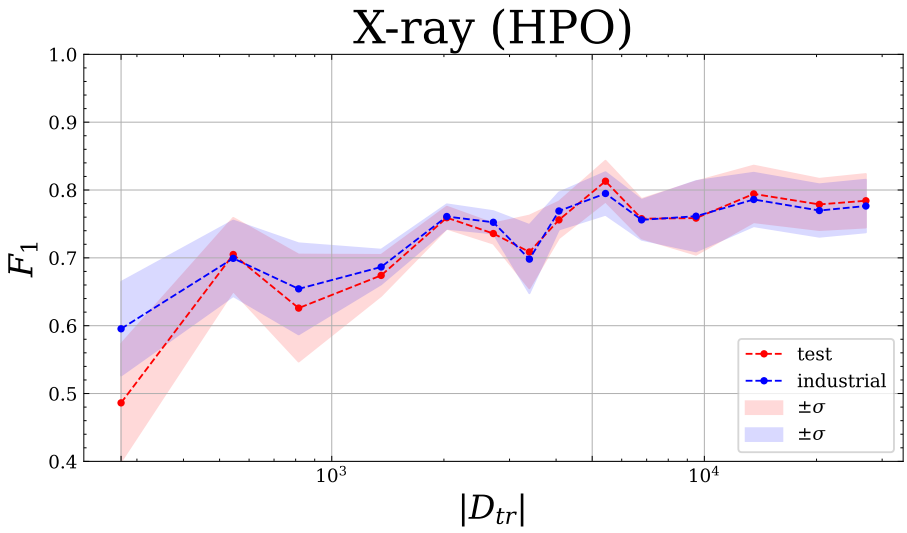


Figure A.3: X-ray (HPO) pixel F_1.

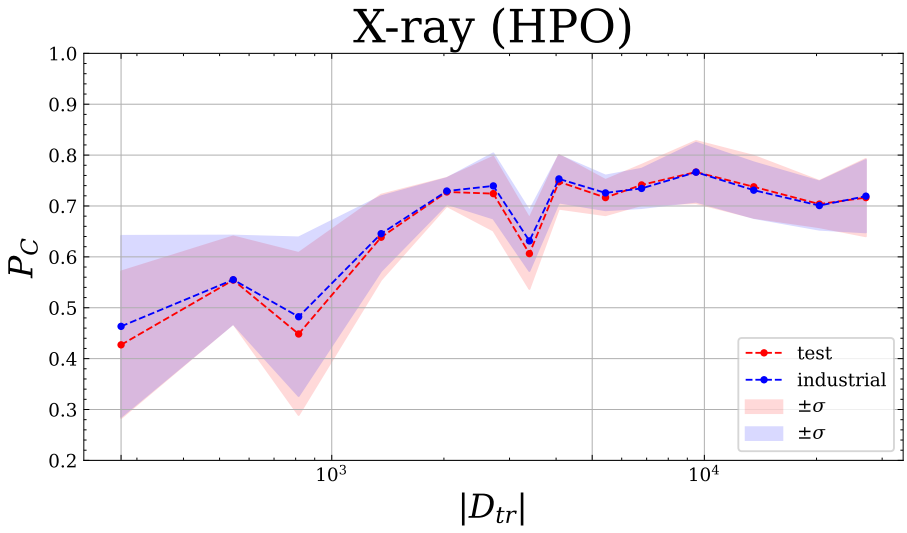


Figure A.4: X-ray (HPO) contour precision P_C.

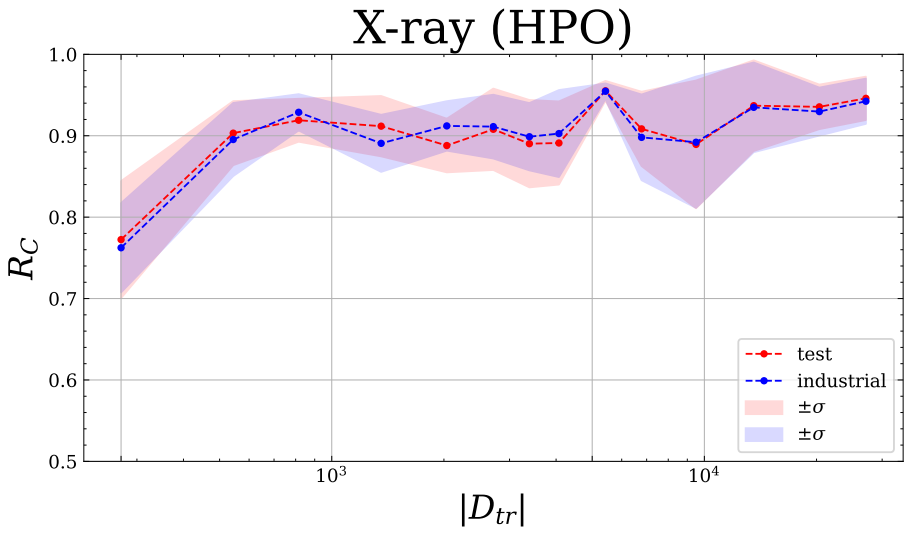


Figure A.5: X-ray (HPO) contour recall R_C.

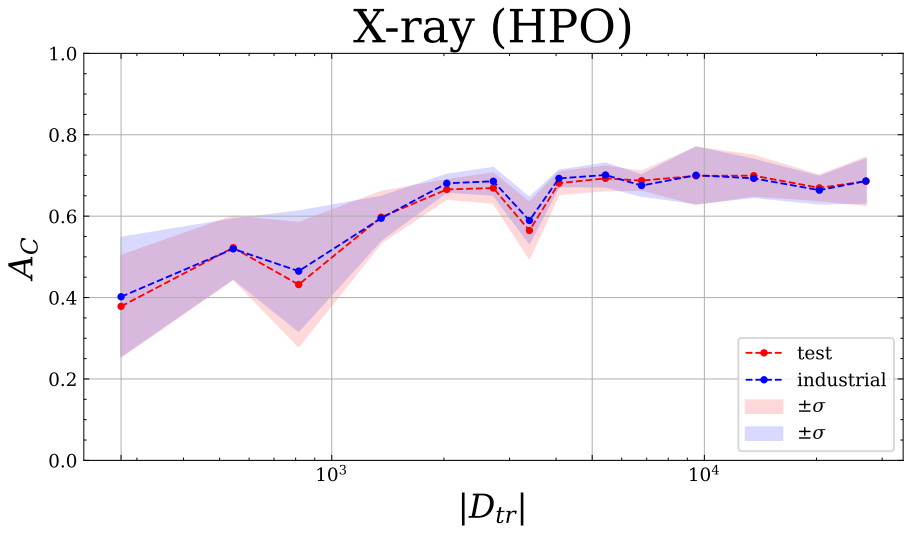


Figure A.6: X-ray (HPO) contour accuracy A_C.

A.2.2 X-ray

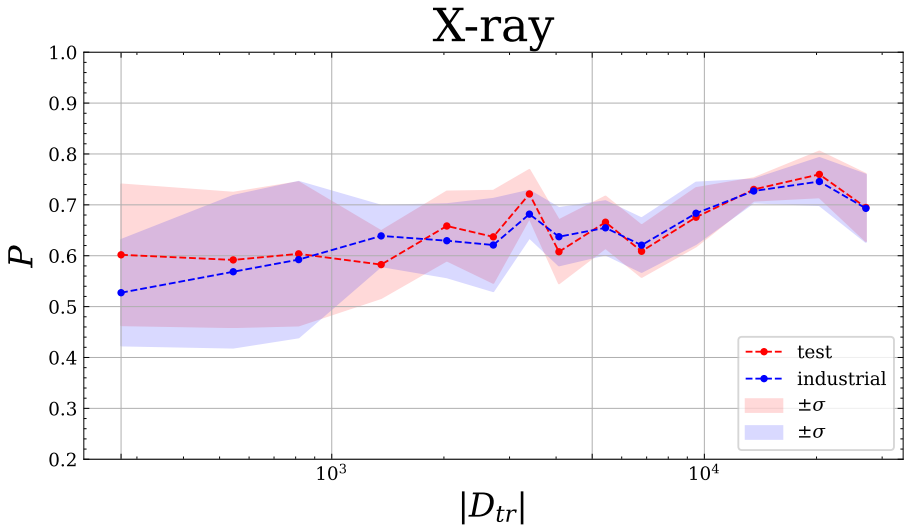


Figure A.7: X-ray pixel precision P.

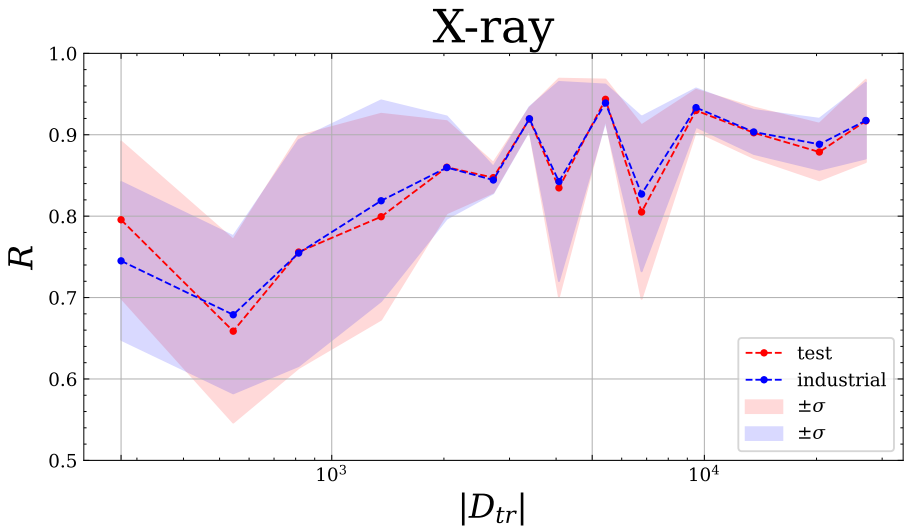


Figure A.8: X-ray pixel recall R.

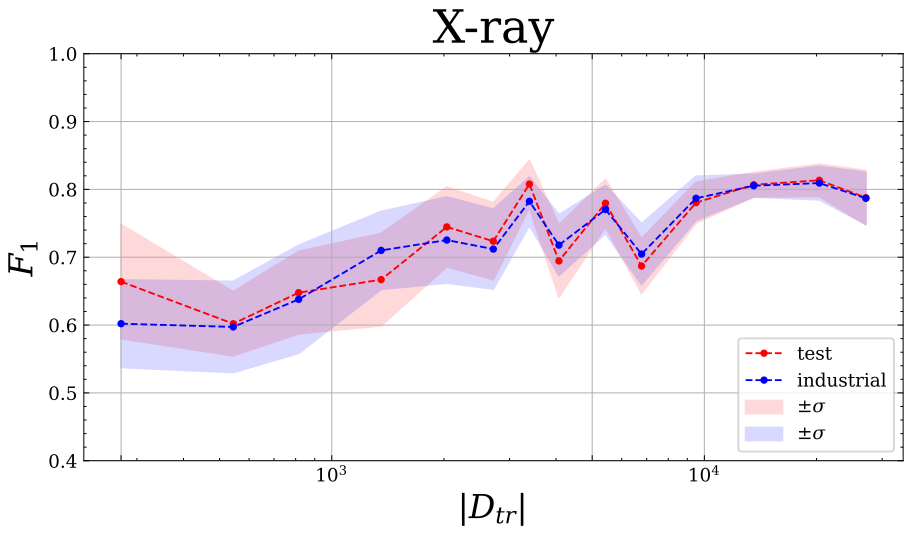


Figure A.9: X-ray pixel F_1.

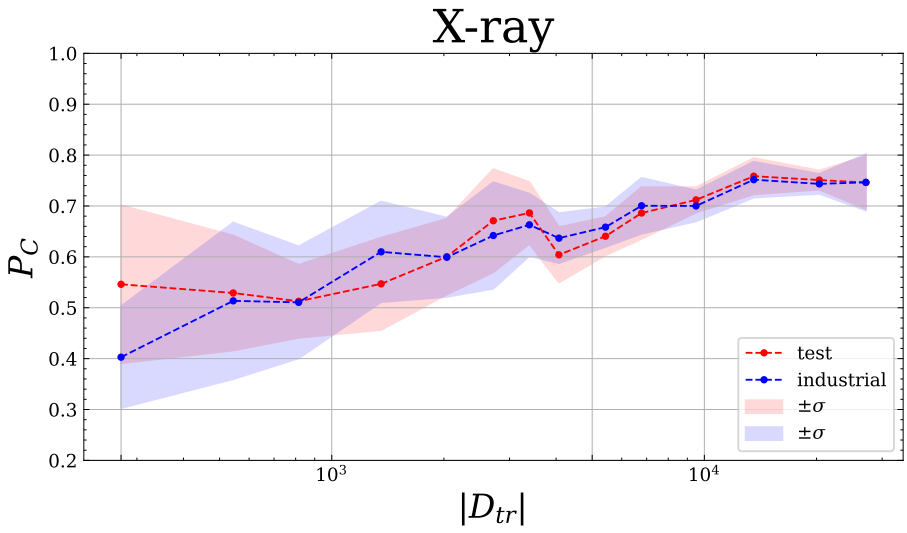


Figure A.10: X-ray contour precision P_C.

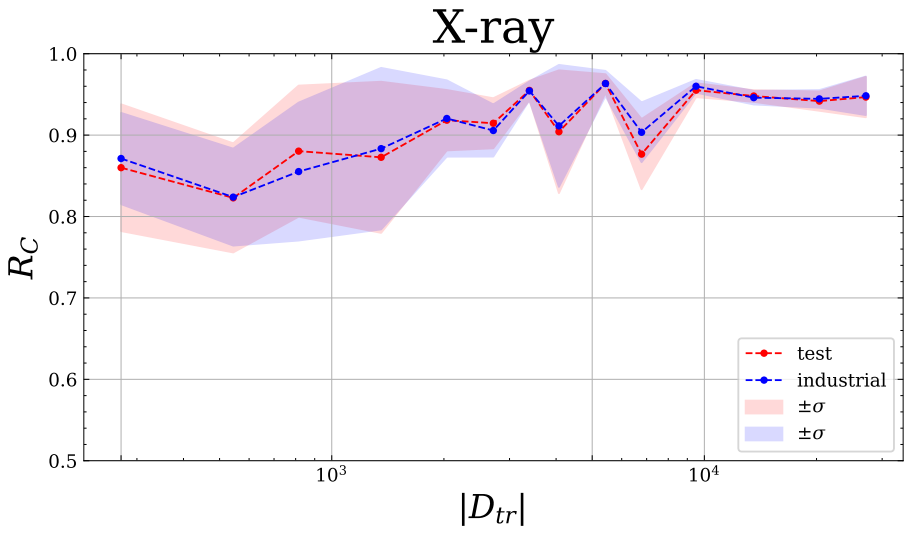


Figure A.11: X-ray contour recall R_C.

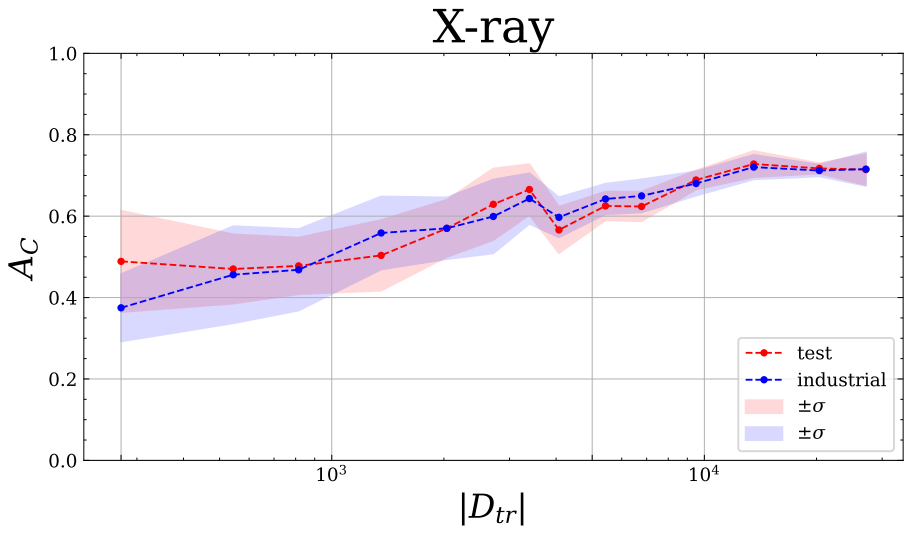


Figure A.12: X-ray contour accuracy A_C.

A.2.3 AVI

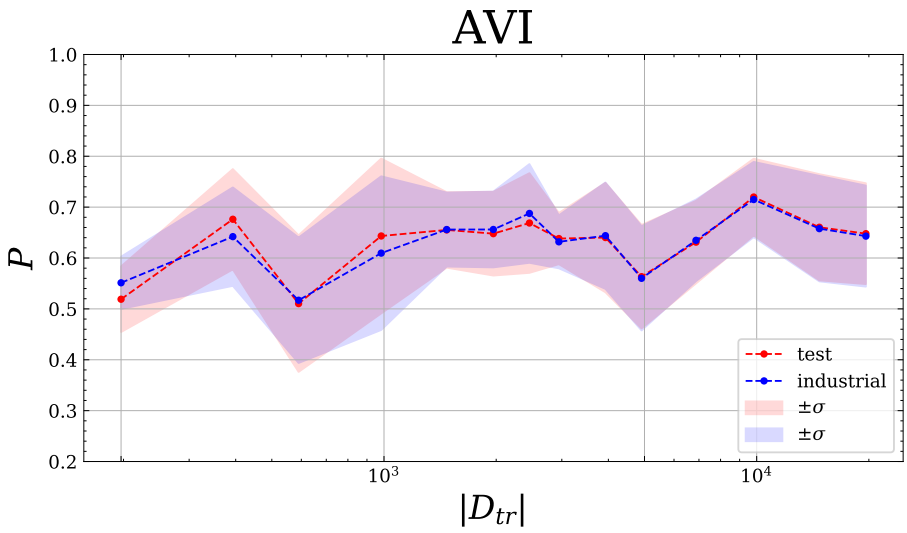


Figure A.13: AVI pixel precision P.

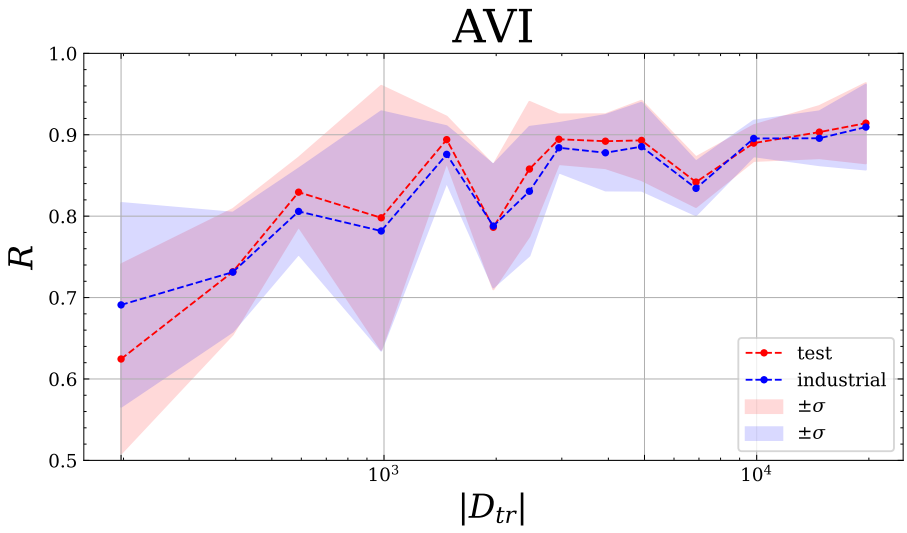


Figure A.14: AVI pixel recall R.

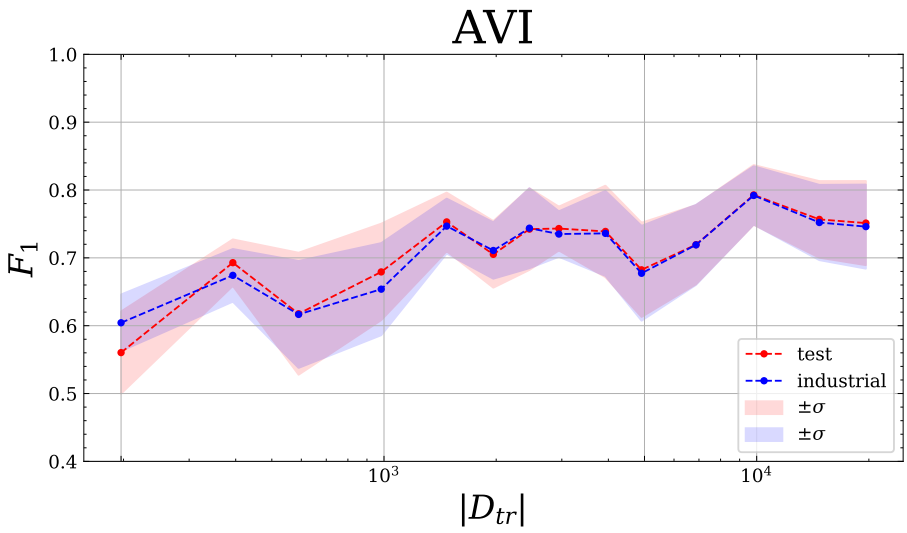


Figure A.15: AVI pixel F_1.

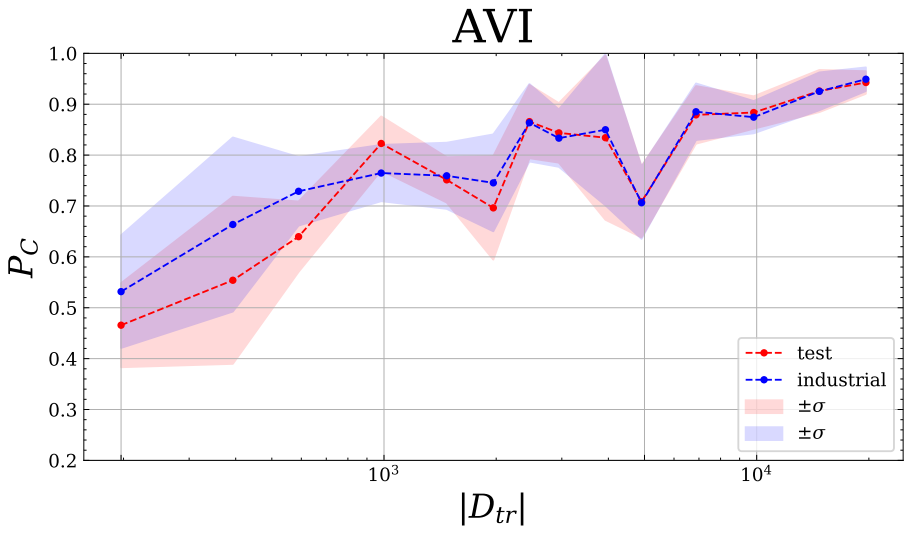


Figure A.16: AVI contour precision P_C.

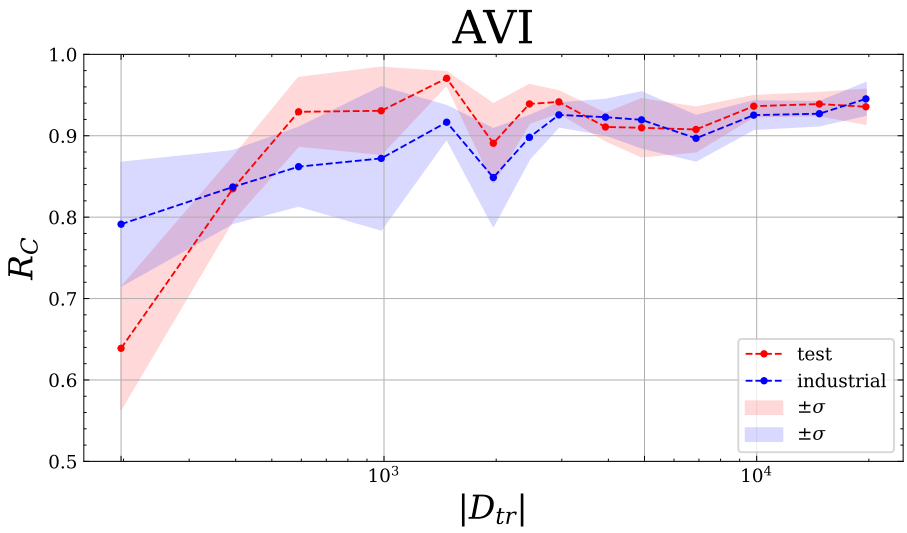


Figure A.17: AVI contour recall R_C.

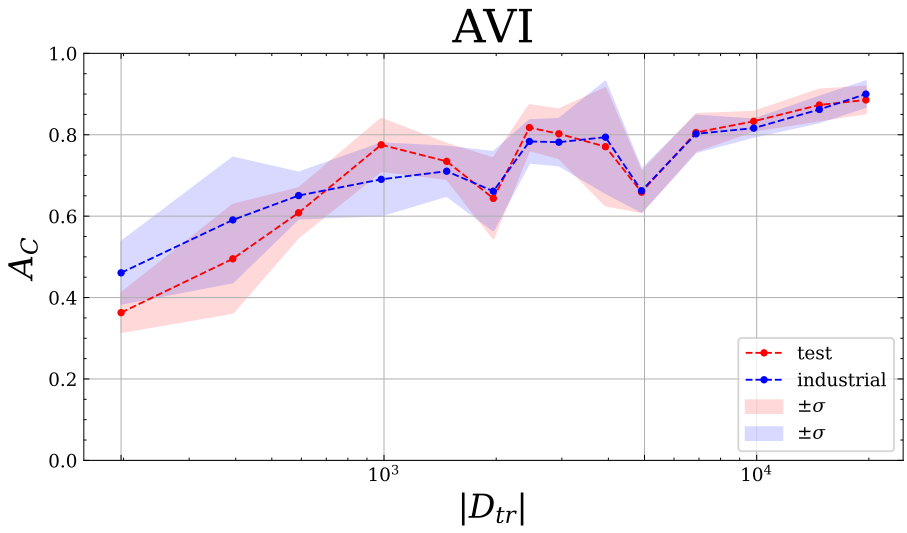


Figure A.18: AVI contour accuracy A_C.

A.2.4 Curve fits

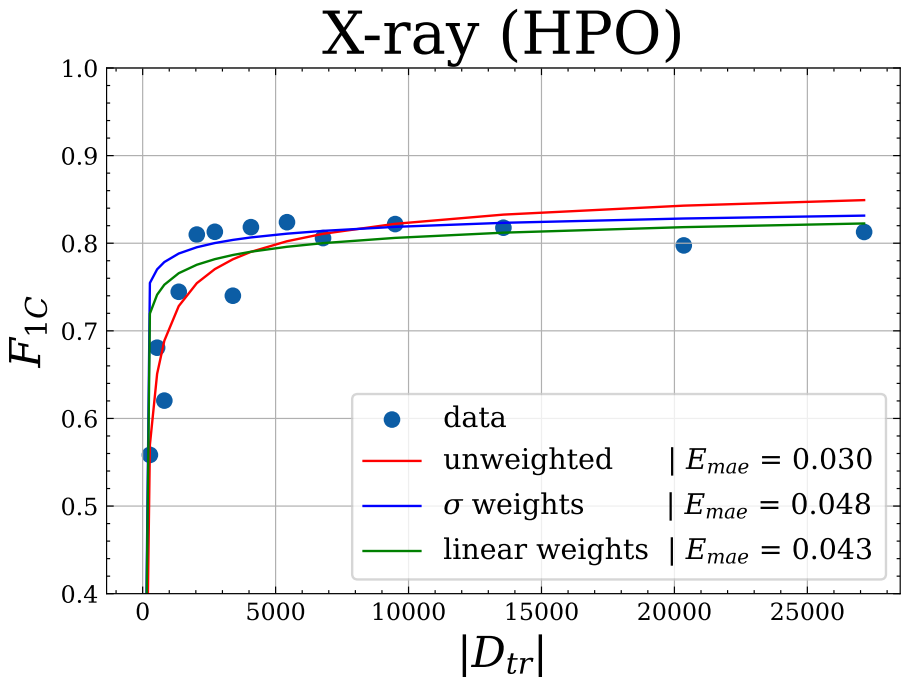


Figure A.19: X-ray (HPO)

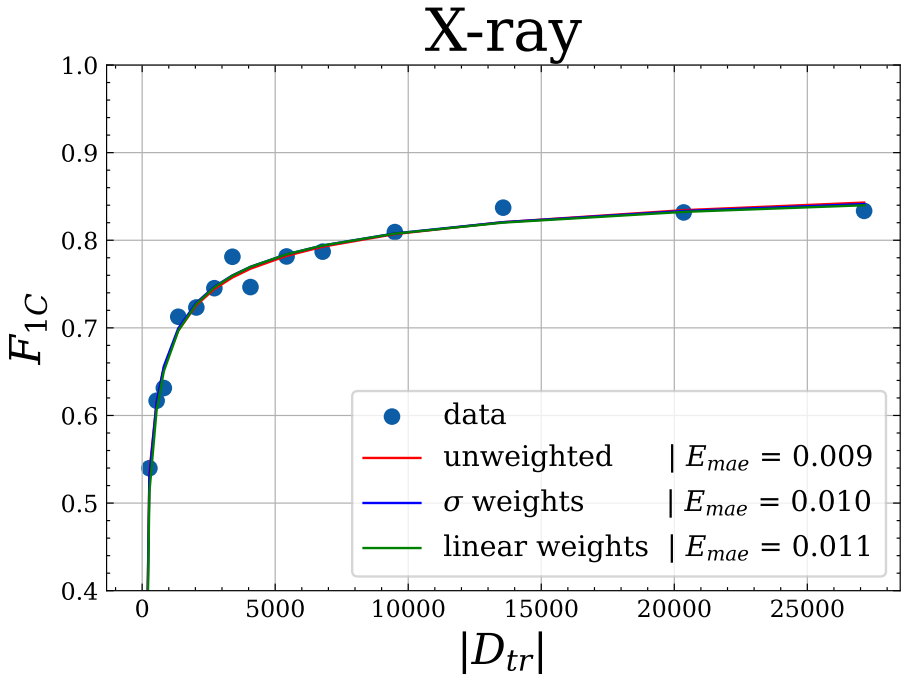


Figure A.20: X-ray

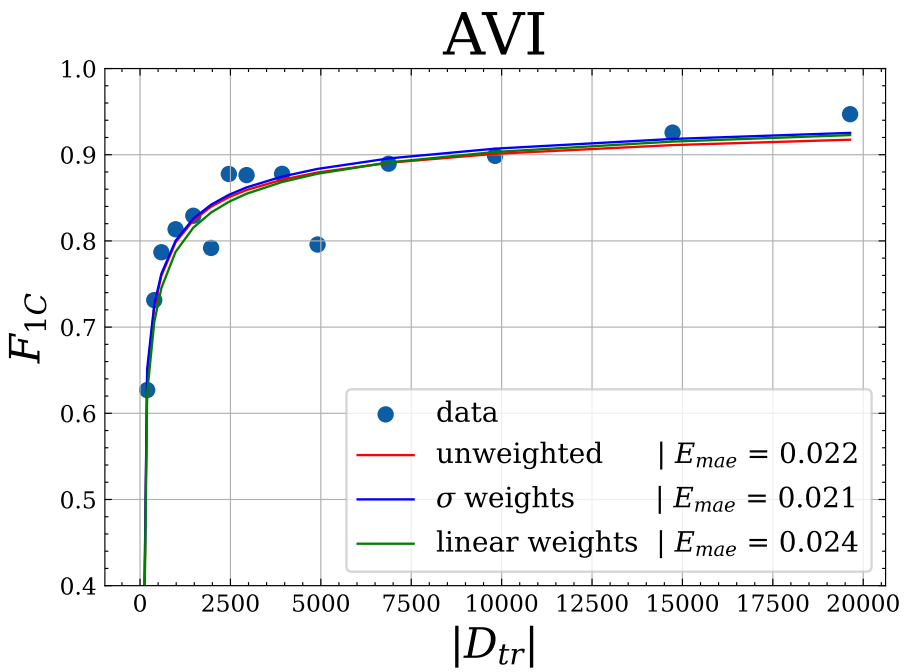


Figure A.21: AVI

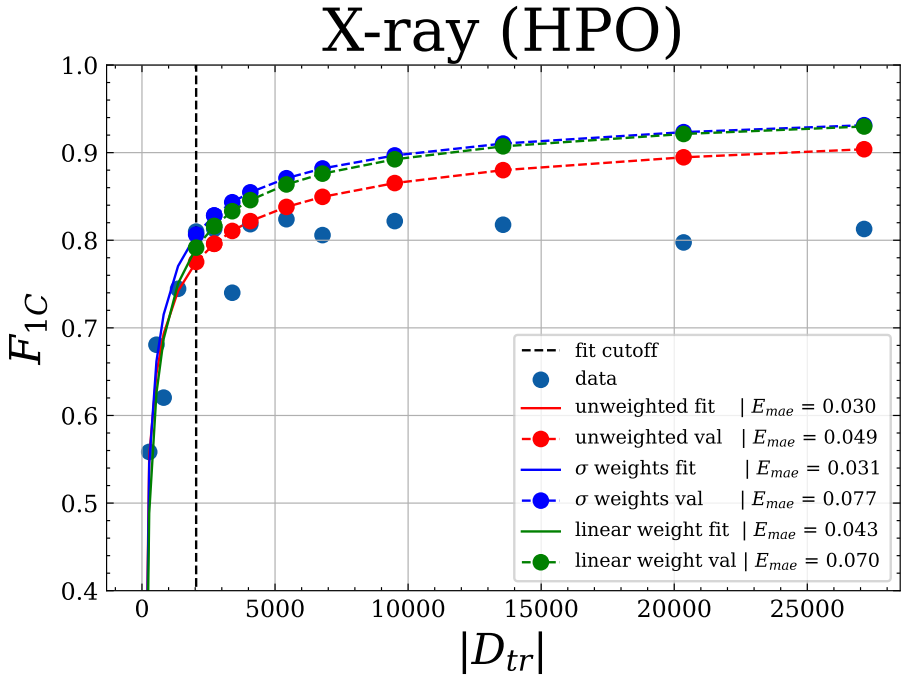


Figure A.22: X-ray (HPO)

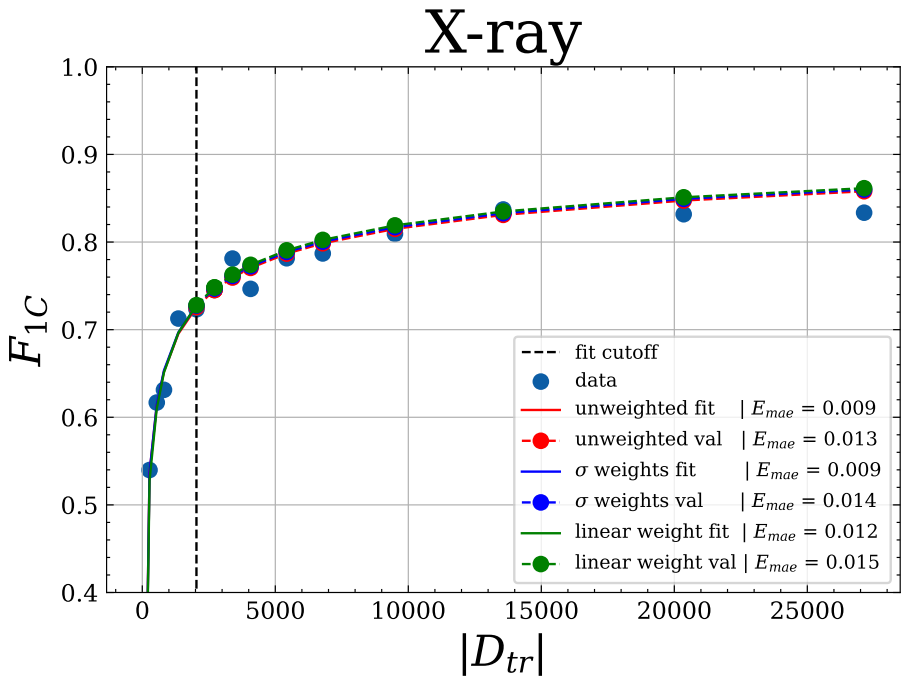


Figure A.23: X-ray

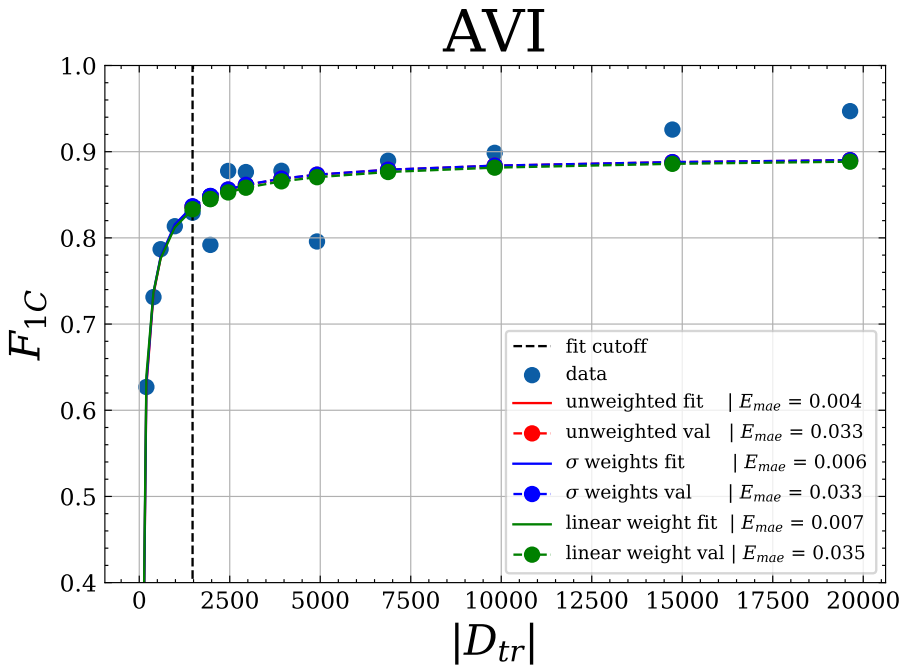


Figure A.24: AVI

A.3 Additional results for task 2

Here the sampling distribution is shown for the different proactive training (PT) methods.

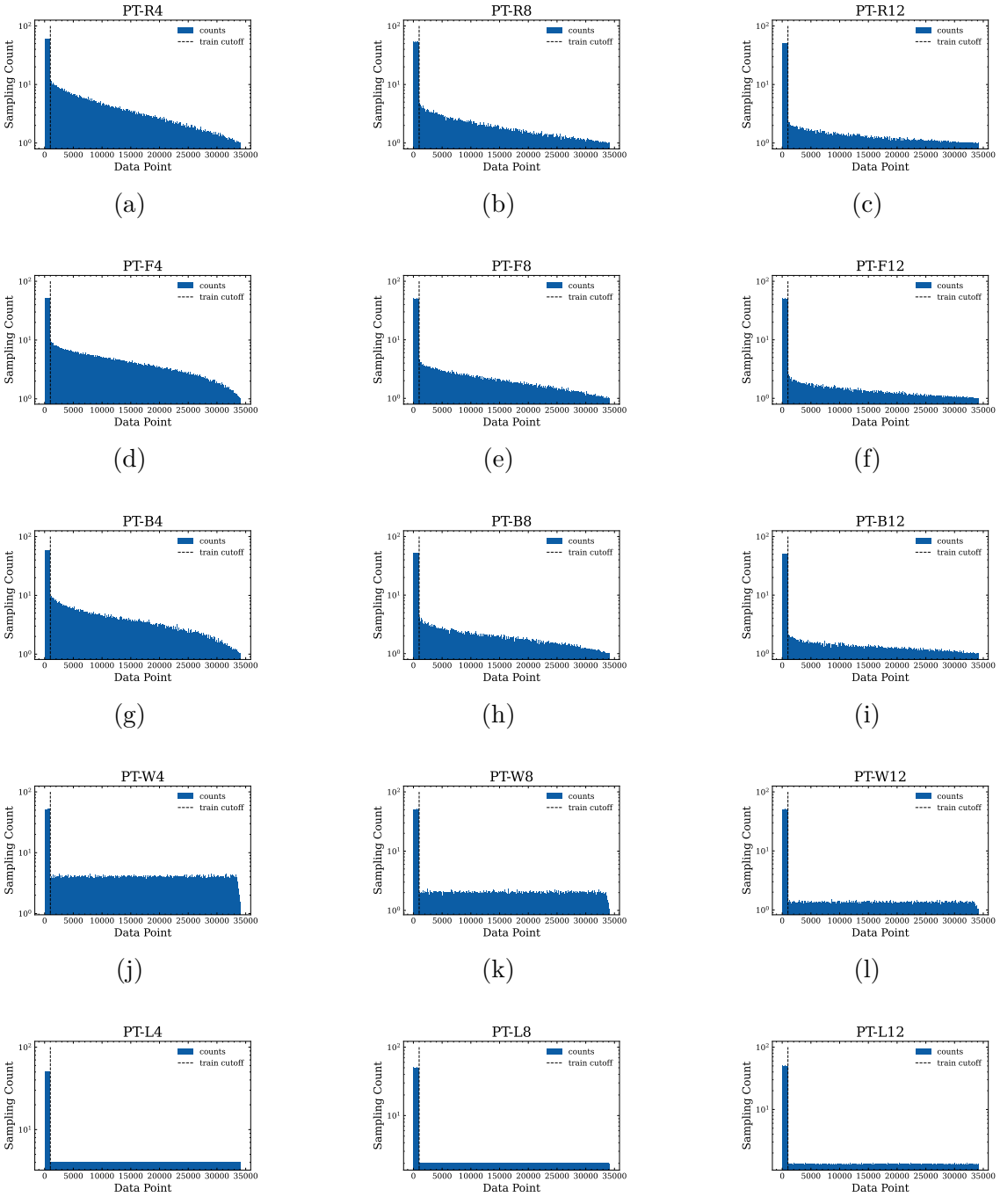


Figure A.25: Sampling distribution of data points for the different proactive retraining methods. Random sampling A.25a,A.25b,A.25c), frequency based sampling (A.25d, A.25e, A.25f), Boltzmann weighted (A.25g, A.25h, A.25i), window based (A.25j, A.25k, A.25l) and last (A.25m, A.25n, A.25o). The left peak corresponds to the data used for training the initial baseline model that later makes up the history set $\mathcal{H}$ at the start of the timeline.