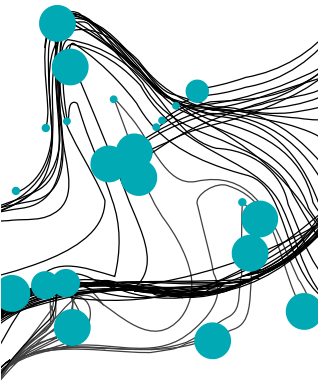
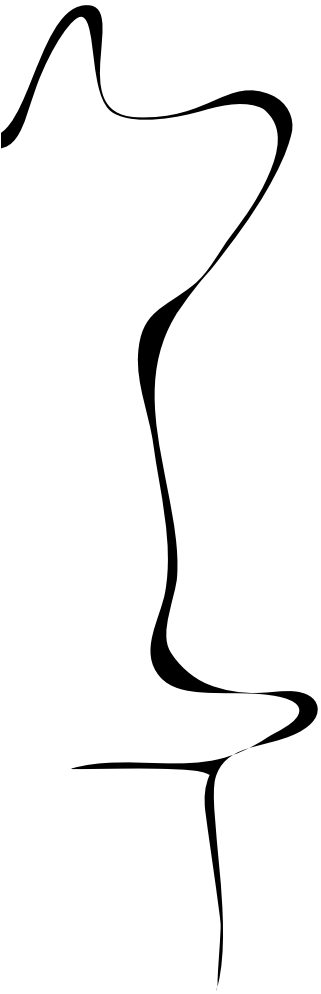


.54463

DMB

DATABASE MANAGEMENT
AND
BIOMETRICS



DETECTION OF AREAS OF INTEREST IN BENDING STRAIN DATA FOR PIPELINES THROUGH 1D OBJECT DETECTION

Wouter Couwenbergh

MASTER THESIS ASSIGNMENT

Committee:
Estefania Talavera Martínez
Dr.Ir. Luuk J. Spreeuwers
Dr Duc V. Le Viet Duc
Klaas Kole (ROSEN) External

November, 2022

2022DMB0009
Data Management and Biometrics
EEMathCS
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

UNIVERSITY OF TWENTE. | DIGITAL SOCIETY INSTITUTE

Acknowledgment

With this master thesis I will be finishing my studies at the University of Twente, which started almost five and half years ago. There are a couple of people I would like to thank for supporting me throughout this project. I want to start off by thanking ROSEN for the opportunity to write my thesis as part of my internship with them. More specifically I would like to thank my supervisors at ROSEN, Klaas Kole and Michiel Roeleveld, for their support and guidance, both at the office and online. Additionally, I would like to thank the ROSEN Enschede office for their support and feedback during my time there. Finally, I would like to thank my university supervisor, Estefania Talavera Martínez, and examiners, Dr.IR. Luuk Spreeuwers and Dr. Duc Le Viet Duc, for their feedback.

Abstract

Bending strain is a metric used to evaluate the build-up of stress within pipelines and subsequently their risk of breaking. Its analysis is a time-consuming and tedious process mostly involving manual evaluation which this paper aims to address. Bending strain data is 1D, similar to time series. However, in this domain detection and localization of areas of interest is still a relatively new field. U-Net has shown to work well on similar 1D data such as EEG and ECG data, but object detection algorithms have yet to be used. This paper will therefore explore the feasibility of applying a YOLO v4 based model to detect areas of interest within 1D bending strain data. The model's performance will be evaluated using a 1D U-Net as a baseline, as it has already been established to work well on 1D data. Though, to allow for direct comparison with YOLO, the segmentation map of U-Net will be converted into a set of bounding boxes.

The results show that U-Net outperforms YOLO in terms of detecting bends (with an average precision of 0.71 and 0.51 respectively), but that YOLO outperforms U-Net in detecting strain areas (with an average precision of 0.084 and 0.065 respectively). Moreover, while both models achieve comparable results (suggesting that YOLO performs on par with U-Net on 1D data), they are still found lacking in performance especially when detecting strain areas. In the best-case scenario, using an IoU threshold of 0.5, both models were able to attain an average precision of about 0.15, which is not sufficient to be used. Using the same threshold, however, both models were able to achieve an average precision of about 0.8 for bends which is a lot more promising. It was later found that some possible inconsistencies within the data and the labelling of said data might be the cause for this performance disparity. Future work using this data should therefore first aim to standardize the data and remove any inconsistencies. Thereafter, the focus of any future work should be on improving the detection performance of strain areas within bending strain data.

Table of Contents

1 Introduction

1.1 Background

Pipelines can fail due to a host of causes. One of the most hazardous ones is pipeline movement due to geotechnical reasons (e.g., landslides) and their partly unpredictable nature. When a pipeline moves too far away from its original location, stress can build up in the pipe causing it to break. To evaluate whether a pipeline is at risk of breaking, the so-called bending strain of a pipeline is monitored. This represents the elongation of the outer fibre of the pipe material with respect to a straight pipeline. The assumption is made here that a pipeline is straight in an 'as laid' condition and that bending occurs with respect to a neutral line through the centre of said pipeline. Bending strain is a metric derived from a combination of the trajectory of the pipeline, which is measured by an inline inspection pipeline tool, and the diameter of the pipeline (see formula 1) [1]. This trajectory is measured at a set interval throughout the pipeline, in this case every 0.1 meter, and results in a vertical and horizontal strain component (optionally also a total strain) at a given location in the pipeline.

$$\varepsilon_{Bending} = \frac{D}{2R_{curv}} \qquad (1)$$

Where D [m] is the pipeline diameter and R_{curv} [m] the radius of the pipeline bend.

Through the analysis of this data, several anomalies can be identified by domain experts. Among these anomalies, two are of special interest, namely the areas labelled as 'bend' and areas labelled as 'strain' (see Figure 1 for an example). Bends are areas where the pipe is experiencing bending strain caused primarily by the trajectory the pipe has been laid in, for example because the pipeline had to make a 90-degree turn. However, while strain could be experienced here, depending on the method taken to construct a bend, this is normally mostly plastic strain and is therefore deemed acceptable. Strain areas on the other hand are of special concern, as these are the areas where the pipeline has, for example, moved from its original location resulting in mostly elastic strain. It is this elastic strain that has to potential to damage a pipeline and cause it to break. Moreover, the circumstances that cause elastic strain are also the ones where there is the potential for them to worsen over time. For example, in the case of the ground being eroded away after heavy rainfall (see Figure 2). Strain areas are therefore the most important areas of interest to find and find correctly.

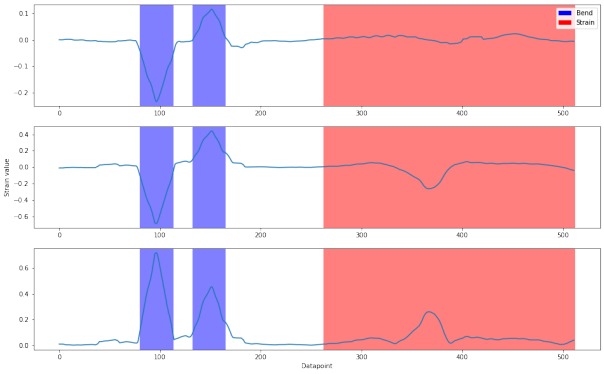


Figure 1: Bending strain plot, with in blue the bend and in red the strain areas. From top to bottom the graph shows horizontal, vertical, and total strain.

Figure 2: An example of a geotechnical reason causing pipeline movement. In this case ground subsidence is affecting the trajectory of a pipeline, shown circled in red, possibly leading to elastic strain (or a strain area) [1]

[1] Image provided by ROSEN

1.2 Motivation

The identification and classification of Bend and Strain areas of interest within bending strain data has up until now been a mostly manual process that is both tedious and a time-consuming process. There are some algorithms available, within ROSEN, to help detect bending strain areas of interest. However, these are relatively simple rule-based algorithms. Because of this, they cannot be relied upon to detect all the relevant areas and are therefore only rarely used. Currently, the algorithms can detect about 90 percent of all areas, but 80 percent of the areas identified are identified incorrectly (false calls). Moreover, taking into account the post processing required by the evaluators, it takes about 10 seconds to process a single feature or, on average, 4 hours to process an entire pipeline. Machine learning is starting to be applied to the problem as well, however it currently seems to be limited to classification. For example, a deep belief network has been applied on bending strain data to classify various pipeline features [2]. Automating the process of localizing so called areas of interest is therefore much a desired addition to the current workflow. Based on the current performance, any new method of detecting bending strain areas of interest should have a detection rate of at least 90 percent; a false call rate of less than 50 percent; and be able to process a single pipeline, on average, in at most an hour.

Bending strain data is similar to both time series as well as image data. Just as time and image data can be multi-channel, so can strain data. In this case, it contains either 2 or 3 channels, depending on whether the total strain is also included. It is comparable to time series data as it too is 1D data. However, unlike time series data, it is not measured at a set time but a set distance interval. It is therefore possibly more appropriate to say that it is like a 1D image. While it might be more comparable to a 1D image, both data domains can still provide valuable insight as to how to create an automatic bending strain detector. When considering the field of computer vision, there are two main methods by which objects are selected within an image: semantic segmentation and object detection by way of bounding boxes. Within the domain of time series data, on the other hand, extracting regions of interest is a relatively new area.

Until now, the analysis of time series data in terms of anomaly detection has mostly focussed on classification; for example to classify whether an atrial fibrillation is present in an ECG signal [3]. Moreover, this field is well established and has well put together datasets available for testing such as the UCR Time series archive [4] and extensive comparisons of various models using (part of) this dataset [5]. While classification might work well on bending strain data, it is too coarse of a method. This is because localization is an important aspect of labelling the data besides detecting whether a feature is present within a sample. Next to classification, semantic segmentation is starting to be applied on timeseries data as well, mostly in the medical field. Semantic segmentation is in essence a per pixel, or in this case per time step, classification of the input. It has thus far been applied to EEG data using a U-net to detect the different sleep stages [6]; to EEG data using both a U-Net and hybrid network to detect an individual's state of vigilance [7]; to ECG data to extract characteristic parts of the signal [8]. Semantic segmentation therefore looks like a promising option to detect bending strain.

However, whereas semantic segmentation is starting to see its adoption within the time series domain, object detection (by way of bounding box detection) does not. Until now bounding box detection does not seem to have been applied within the context of time series like data. Moreover, it can be argued that bounding box detection is an easier task than semantic segmentation as it involves less fine grain details to be resolved. This is due to bounding box detection not needing to classify every individual pixel (or datapoint). It would therefore be interesting to see whether there is any improvement to be gained as compared to using a semantic segmentation approach. Furthermore, since the desired output for bending strain labels is in the form of a start and end distance (essentially a 1D bounding box), which would require addition conversion step when using semantic segmentation, a model that outputs this format would be desirable. To that end,

this study will be implementing and comparing two popular models, one for semantic segmentation and one for object detection, and adapting them to work on 1D data. These two models will be based on U-Net [9] and YOLO v4 [10] respectively. To summarize, the contributions of this paper will be:

1.3 Contribution

1. The adaptation of YOLO v4 to the 1D domain.
2. A framework for the detection of areas of interest within pipeline bending strain data.
3. The detection of bending strain areas of interest using machine learning.
   3.1. Application of 1D YOLO and 1D U-Net on bending strain data.

1.4 Outline

The outline of this paper will look as follows: in chapter 3 the related works will be explored, chapter 4 will go into more detail about the proposed networks, chapter 5 will discuss the experimental setup, chapter 6 will discuss the results, chapter 7 will further discuss the results and answer any research questions that are left, chapter 8 will conclude this paper, and chapter 9 will talk about possible future work.

1.5 Research Questions

This section will introduce the research question that shape this thesis. These questions will either be answered in the related works section or through the results of (one of) the experiments.

1. How should a framework to extract areas of interest in bending strain data from multiple pipelines function?
2. How can machine learning be applied to extract areas of interest from bending strain data?
   2.1. How can U-Net best be configured to perform optimally on 1D bending strain data?
      2.1.1. How much performance can be gained by employing different loss functions when training U-Net on 1d bending strain data?
      2.1.2. How do different activation functions affect U-Net's performance?
   2.2. How well can YOLOv4 be adapted to predict bounding boxes in 1D?
      2.2.1. How does YOLOv4 perform compared to a 1D U-Net for area of interest detection?
      2.2.2. How does pre-training the backbone of YOLO affect the model's performance?
   2.3. How does the window size influence the models' performance?
   2.4. How does the amount of data influence the models' performance?
      2.4.1. How can the class imbalance best be handled?
      2.4.2. How will applying data augmentation for bending strain data affect the models' performance?

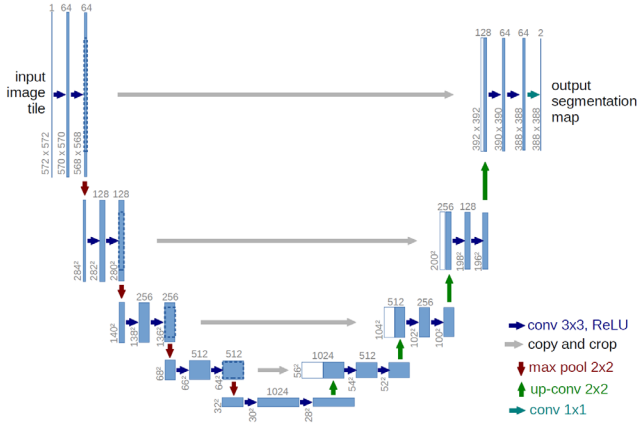


Figure 3: U-Net network structure. On the top left is the input to the network and on the top right the output. By default, U-Net takes in a grayscale image resulting in an input of 572x572x1. Blue boxes represent multi-channel feature maps and white represent copied feature maps. The feature maps on the right-hand side differ in x-y size due to the cropping applied by the overlap-tile strategy. [9]

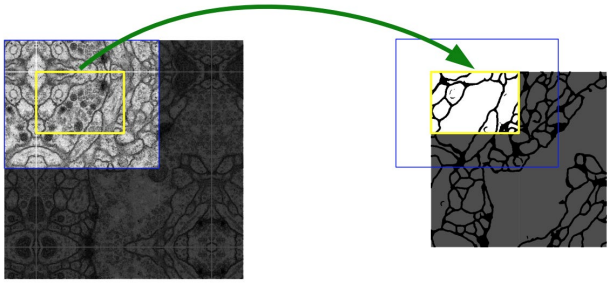


Figure 4: Overlap-tile strategy for dividing up large images into manageable sized pieces to be processed by the network. In yellow is the segmentation prediction of the network which requires the area in blue as input to the network. Missing data is extrapolated by mirroring. [9]

2 Related Works (RQ 2)

Classification has long been a staple for time series data going back to the early 2010s and has had time to mature [11]. However, the same cannot be said for the identification of regions of interest within time series data. As mentioned before, semantic segmentation is starting to see some adoption for time series data, mainly in the medical field. But object detection has been applied even less, if at all. The following sections will therefore give an overview of semantic segmentation and its application on time series (like) data, some background on object detection and what is currently still missing.

2.1 Semantic segmentation

Semantic segmentation is the task of classifying each datapoint in the input sample. In the case of images this would be pixel wise classification or in the case of time series data each datapoint in time gets assigned a class. Semantic segmentation has seen its start with image data and over the years has seen a host of different approaches to solving this challenge. These models range from fully convolutional networks and encoder-decoder networks to recurrent neural networks and attention-based models [12]. Some popular networks for semantic segmentation include Mask-RCNN [13], built using Faster-RCNN [14] object detector as a foundation; DeepLabv3 [15], developed by Google; and U-Net, originally developed for medical imaging. Of these models, DeepLabv3 most likely performs best but is also by far the biggest model. While there are a couple cases where U-Net and R-CNN perform on par or are close in performance [16], [17]; more often than not R-CNN outperforms U-Net [18], [19]. However, of these models U-Net has the strongest foundation within the domain of time series data [6]–[8], [20]–[23] and will therefore be chosen as the baseline to compare 1D YOLO-v4 against.

U-Net was first introduced in 2015 by Ronneberger et al. to segment neuronal structures in microscopy images [9]. It is a fully convolutional encoder-decoder network that incorporates skip connections between the encoder and decoder of the network. This is done to add back high-resolution features from the encoder to help with the localization of objects within the input. Its encoder-decoder like structure also inspired its name as the network itself forms U like shape (see Figure 3).

U-Net also made use of an overlap-tile strategy (see Figure 4). This involves splitting up the input image into multiple chunks that overlap each other, which are then each separately fed through the network. The output

of the network is essentially a crop of the input such that the overlap between the chunks is cut away. The overlap is to still provide the pixels along the edge of the chunk with the context they would have had, had it been one big image. In case a chunk contains an edge of the image itself, the overlap areas along the outside are extrapolated by mirroring along that edge. After each chunk has been seen by the network, the output segmentation maps are recombined to form the semantic segmentation of the input image. While this strategy shows some merit, especially for larger input images, it has not been used by the various time series implementation of U-Net.

When adapting U-Net to work on time series data, various strategies have been employed. Some take an almost vanilla U-Net, and only adjust the 2D layers to their corresponding 1D versions ([7], [22]) or adjust U-Net only slightly ([21]). These adjustments include, but are not limited to; changing the max pooling layers to average pooling, or exchanging ReLu activations with Leaky ReLu. Others add layers onto a base U-Net to achieve their goal ([6], [7]), expand on the idea of U-Net by incorporating features like Residual and Squeeze-Excitation blocks ([8]), or split the decoder up into multiple separate encoders which eventually merged to one decoder ([23]). In each case, U-Net is successfully adapted to enable semantic segmentation on time series data.

2.2 Object detection

Object detectors can generally be split into two categories, the two stage detectors and the one stage, or single shot, detectors. Two stage detectors, such as the Region-Based Convolution Neural Network (RCNN) family of detectors [14], [24], [25], solve the object detection task as a classification problem. As the name already implies, this is accomplished by splitting the task in two: a region proposal task and a classification task. In the first RCNN model [24], this is done by two separate systems. Selective search is used to generate region proposals and a convolutional neural network (or CNN) is used to classify these proposals. Since time series classification is an established field and some of the best performance has been achieved using CNNs [3], [11], [26], this opens the door for object detection in time series (like) data.

Later RCNN models improved by first no longer warping the proposals but taking a whole image as input and pooling the proposals with the network at a later stage in Fast RCNN [27]. This evolved into Faster RCNN [14], which extracts region proposals from the feature map of the CNN allowing the model to be trained end-to-end. When compared to other state of the art models, Faster RCNN still, after seven years, remains competitive in terms of accuracy. However, where it does lack in comparison to other models is inference speed. Of the models with real-time inference speeds, YOLOv4 a one stage detector comes closest in terms of performance. Where Faster R-CNN achieved an Average Precision at 0.5 overlap ($AP_{0.5}$) of 67.00% at 5 FPS, YOLOv4 achieved 64.90% at 31 FPS [28]. While speed is not the most pressing requirement in this case, having the option to run the model in real time would open the door for it to be used in other scenarios where speed is of concern.

2.2.1 YOLO

YOLOv4 is one of many versions of the YOLO family of models available, each adding new features and the latest developments to offer incremental improvements with respect to the previous version. Originally YOLO was started by Redmon et al who created YOLO [29], YOLO9000 or YOLOv2 [30] and YOLOv3 [31]. After this latest version, Redmon et al decided to stop working on object detection. Since then, YOLO has been picked up by other groups resulting in, among others, YOLOv4, which once again offered incremental improvements. The main idea behind YOLO remained pretty much constant however: You Only Look Once, predict the bounding boxes all at once instead of through multiple stages.

As opposed to the image classification approach of R-CNN, YOLO approaches object detection as a single regression problem. It predicts bounding boxes for a given image in one shot, going straight from the pixels to the bounding box coordinates. It does this using a fully convolutional network which can be separated into a

couple of sub-networks. These are the backbone, or feature extractor and is often a network also used in classification tasks; the neck, which is used to collect feature maps from the different scales within the backbone; and the head, which is responsible for predicting the classes and bounding boxes of an object (see Figure 5). Additional modules may also be added in between.

YOLO originally started out with a custom model, but from YOLOv2 onwards the idea of a base network or backbone has consistently been implemented. Throughout the versions these, of course, kept getting updated to the latest methods available. YOLOv2 started out with Darknet19, which contains 19 convolutional layers and 5 maxpooling layers. V3 upgraded this to Darknet53, 53 once again referring to the number of convolutional layers, which implemented the skip connections seen in ResNets. V4 eventually settled on the CSPDarknet53, which added additional skip connection to emulate Cross-Stage-Partial-connections. Next to upgrading the type of backbone used in the network, YOLO v4 also switched the activation function used throughout the network. Whereas YOLO up to v3 made use of leaky ReLu [32], v4 opted to use the mish activation [33]. Just like leaky ReLu, mish can be used to address the issue with ReLu [34] where its gradient is zero when the output is zero. It has shown to perform better when compared to the same network using (leaky) ReLu. Finally, to enhance the receptive field a spatial pyramid pooling layer is added on top of the backbone [35], which was first implemented in YOLO v3 [31]. SPP involves applying max pooling at various sizes to the output of the network and then recombining the results of the pooling operations. However, the original SPP recombined the output of the pooling layers into a one-dimensional vector which would not work in a fully convolutional network. To address this v3 instead concatenates the output of the layers.

The neck was first introduced in v3 to facilitate the prediction at multiple scales and uses a similar concept to feature pyramid networks or FPN [36]. It essentially converts the single output layer of the backbone into three differently sized outputs. It does this by extracting the feature map at several layers in the backbone and concatenating them with the layer extracted from further up in the backbone. To be able to do this, the feature map does need to be up sampled to ensure that both layers are of equal dimension. Since each of these layers is extracted from the backbone at a different level, each will be of a different resolution. This would be similar to down sampling an image and passing it through the backbone again to achieve an output of a lower/higher resolution. This is done to more easily detect both large and small bounding boxes. V4 improved upon this by using the path aggregation network or PANet [37] approach instead of the FPN approach, which adds another aggregation path this time traveling in the same direction as the backbone. This way the combined knowledge of the higher-level features and the lower-level features is passed back up to the higher-level features.

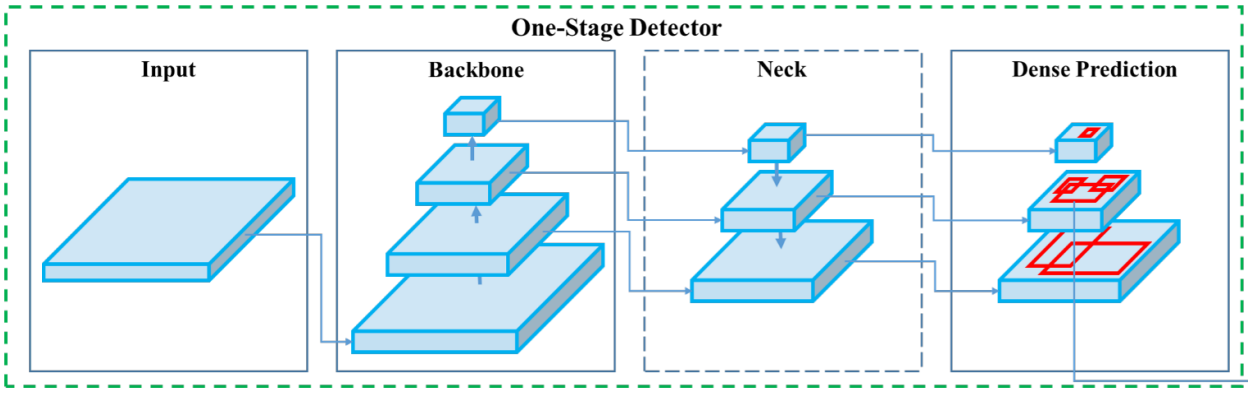


Figure 5: Abstract overview of the YOLOv4 network structure with from left to right the input, which in this case is a 2D image; the backbone, responsible for the feature extraction; the neck, which combines features across multiple scales; and the dense prediction, which results in the final set of bounding boxes. [10]

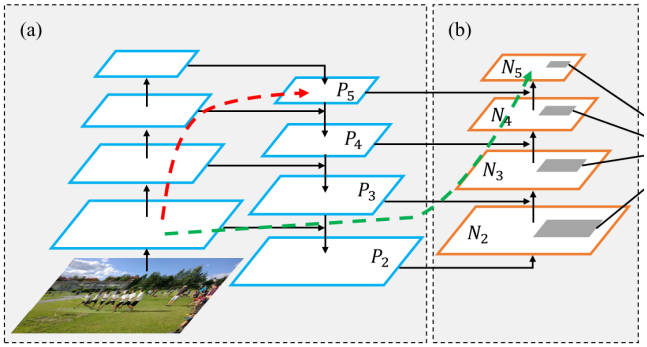


Figure 6: On the left the figure shows the FPN (a) and on the right the bottom-up path augmentation (b) resulting in the PANet neck used by YOLO v4. [37]

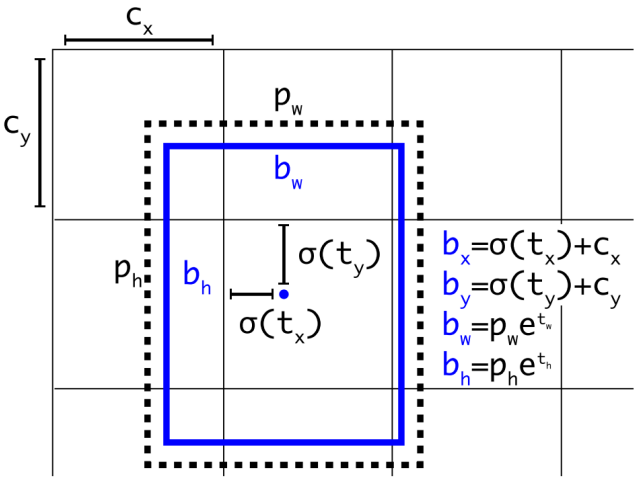


Figure 7: YOLO Anchor box, with t_x, t_y, t_w, and t_h the network's raw output, grid offset C_x and C_y and the anchor box's width and height p_w and p_h. Together these result in the final bounding box coordinates, b_x and b_y, and width and height, b_w and b_h. [30]

Finally, for each of the output layers of the neck (or in case of YOLO v2, the backbone) a set of bounding box prediction is made with respect to a grid across the feature map. This grid size is function of the input size of the network and the downscale factor. In case of YOLO v3 and v4 the downscale factor at these three outputs of the neck is 8x, 16x and 32x and with an input size of 512 (used by YOLO v4) this leads to grid sizes of 64, 32 and 16, respectively. Each grid cell has the option to predict up to a set number of bounding boxes, which in case of YOLO is often 3. However, instead of predicting the absolute coordinates of the bounding box, the difference with respect to an anchor box is predicted (first introduced in v2), which improved bounding box regression significantly (see Figure 7, for an illustration). These anchor boxes are selected beforehand by way of k-means clustering on the training bounding boxes. This is done to ensure that the anchor boxes are already close in size and aspect ratio to the ones that the model needs to predict. For the output of the model these coordinates relative to an anchor box will be converted back to coordinates relative to the size of the input image. The output bounding box prediction consists of the x and y coordinates, width and height and a confidence score (that there is indeed an object within the bounding box). Next to this, each box also has a one-hot encoding for classes. The final output of the model therefore results in the following shape for v3 and v4 per scale: N x N x [3 * (4 + 1 + C)] (where N is the grid size, 3 is for the prediction at the three scales, 4 the bounding box offsets and 1 the abjectness score and C the number of classes)

2.3 Missing from the state of the art

As mentioned before, time series like data is still a relatively new domain for semantic segmentation. This is even more the case for object detectors, which seem to not have been applied in this domain at all or at least even less often than semantic segmentation has. Moreover, most of the applications for time series semantic segmentation have been in the medical field and have yet to branch out significantly to other use cases. One of these cases is bending strain data, which does not seem to have had any segmentation or detection like model applied to it.

3 Proposed Method (RQ 1)

While YOLO-v4's input size can be set to any multiple of 32, it is still a fixed and relatively limited input size. A similar issue is present in U-Net. While it can scale a little more freely than YOLO can, its input size is also still limited. In their research Bochkovskiy et al. have shown that YOLO-v4 works best at an input size of 512 (by 512) [10]. This is, however, multiple orders of magnitude smaller than the amount of data points for even the

smallest pipeline. On average there are almost 650.000 data points per pipeline with the smallest already containing well over 50.000 samples. Even though research has shown that object detection is possible for high resolution images [38], the resolution of the pipelines would still be too big. Therefore, it was decided to use a sliding window approach [39] to feed the data to the network. This method would extract windows out of the data of a given length, set to the input size of the network, and, in this case, a given overlap which will initially be set to 50%. This also has the added benefit of increasing the number of samples the network can train on. To make sure no areas are omitted, the windows are generated with a 50 percent overlap and each of the same size as the input size of the network.

However, using a sliding window approach does mean that after having processed a pipeline the separate windows need to be recombined. Moreover, any overlapping areas will need to be merged or evaluated in some way. This would result in an additional post processing step, but this should have no impact on the performance of the model itself. The eventual process pipeline would resemble the process shown in Figure 8 and would in practice work as follows. When a new pipeline needs to be annotated it is first split into appropriately sized windows (according to the model's input size), each of the windows is then passed through the model which will output a bounding box for any strain area in the given window. Once all windows have been evaluated, the bounding boxes are transferred over from the window to the complete pipeline (their location would at first be relative to the window, not the whole pipeline), after which overlapping areas can be merged or pruned. The result would be a list of strain areas with respect to the complete pipeline.

When using U-Net one extra conversion does need to be made though. Where YOLO's output is bounding boxes, U-Net's output will be a segmentation map of the same size as the input data. Therefore, this output will first need to be converted into bounding boxes to be able to output a list of areas of interest and to allow for a side-by-side comparison between the two models. The models themselves will of course also require some adjustments to allow them to work with 1D bending strain data.

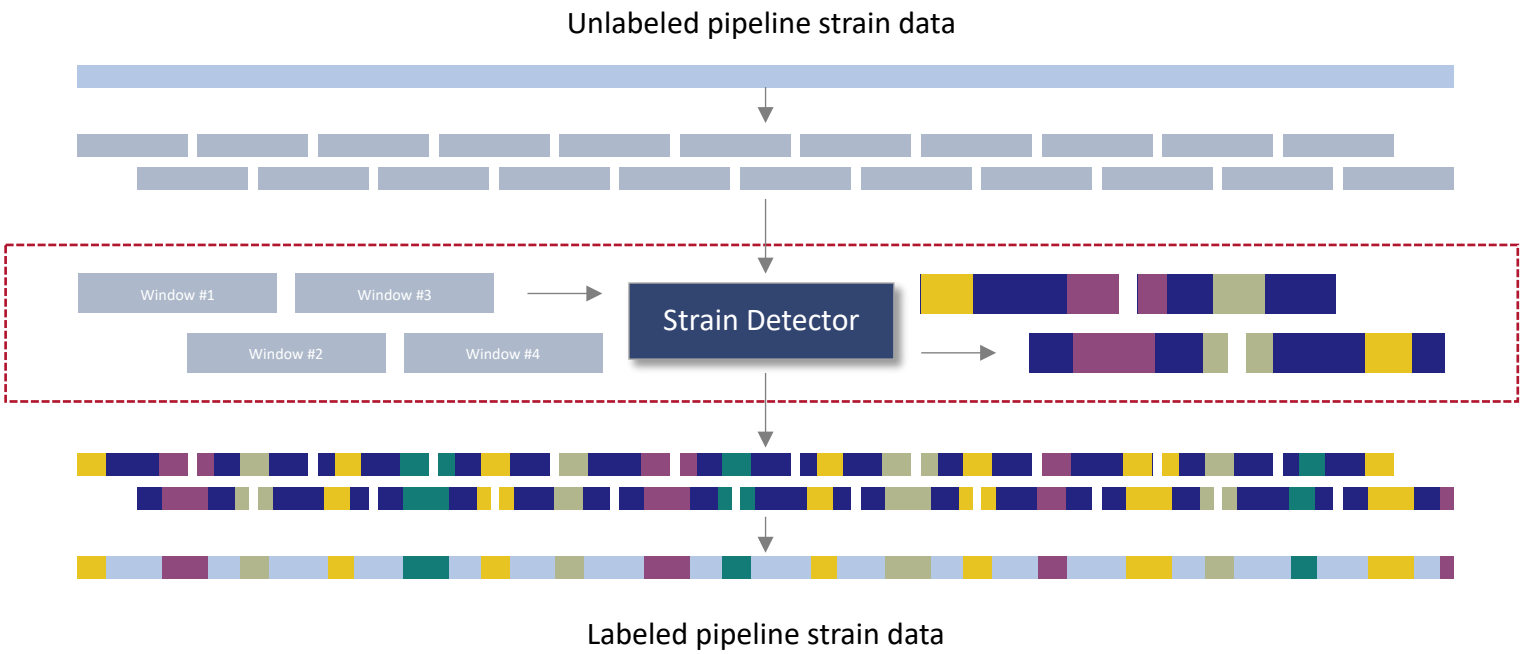


Figure 8: Given unlabeled pipeline bending strain data for a single pipeline, our proposed framework will work as follows. The bending strain data is first split into windows, each of which will be processed by the detector. After this the output of the detector for the individual windows is merged back into one list containing all the areas of interest for the given pipeline.

3.1 U-Net-1D

To create U-Net-1D all the 2D layers used in U-Net will be exchanged for their corresponding 1D versions. However, unlike the original U-Net, no cropping will be applied to the input of the network. The in- and output will therefore have the same size, though different number of channels. The default input size will be 512 with 3 channels, the output of the network will also be of size 512 though with the same number of channels as the number of classes, in this case 2. Moreover, just like the SigU-Net from [21], the max pooling operations will be replaced with average pooling as this was shown to yield a slightly better performance. Lastly, it will be evaluated if switching the ReLu activation function for the Mish activation [33], also employed by YOLO v4, will lead to any performance gains. This is done as Mish has shown to consistently perform better than ReLu (especially at higher number of layers).

See Appendix A *for an overview of the model structure*

3.2 YOLOv4-1D

To get the 1D version of YOLOv4, YOLOv4-1D, a couple adjustments will need to be made. The structure of the YOLOv4-1D model will mimic that of YOLOv4, but all its 2D layers will be switched out with 1D layers. Furthermore, the output of the model will also be adjusted to conform to the bounding boxes in the 1D domain, meaning that instead of an output of 5 (x, y, width, height, and confidence) + number of classes per bounding box, it will now be 3 (x, width, and confidence) + number of classes. Resulting in a final output size of N x [3 * (2 + 1 + C)] for each of the three scales; where N is the grid size, 3 the number of bounding boxes per grid cell, 2 the centre coordinate and width of the box, 1 the confidence score, and C the one-hot-encoding for the number of classes. The default input of the network will be a width of 512 with 3 channels.

See Appendix B *for an overview of the model structure*

4 Experiments

Evaluating and comparing the 1D versions of U-Net and YOLO will consist of two phases:

1) An ablation study on various model configurations and hyper parameters.
2) Evaluation and comparison of the optimal configuration of both models.

To that end, the data available will be split into three parts: train, validation, and test. This is done by first creating a train and test split of 90 and 10 percent, respectively. The train set is then used in k-fold cross validation to get the train and validation sets. In each case, these splits are made based on the specific pipeline the data came from instead of the individual samples. This is done to ensure that in case some pipelines data come from a slightly different domain, these domains are not mixed between training and evaluating the model. This will mean that there can be a slight variance in data samples per set, however this will be mitigated somewhat by randomly creating these sets.

Please note that at the time of publishing, it was no longer possible to extract the number of samples for each data split and the exact class ratio present in these splits. This was because the data itself was no longer available to the author.

4.1 Data

The data used has been provided by ROSEN. The manual evaluation of the data is a time intensive process but combining the data from existing datasets from various projects resulted in a substantial database of labelled data. A subset of this dataset will be used in this project, which at the time of writing this consists of 92 pipelines. Bending strain data is measured and calculated along two axes, horizontal and vertical. Each dataset provides these both and a total strain value for a given location (/distance travelled) in the pipeline. These values are given every 0.1 meters. The data consists of one table containing all the strain data and one containing all the labelled areas. The bending strain table contains, for a given distance, the total, vertical and horizontal strain values, and the label table contains the start, median and end distance for a given area as well as the area's length and class. For an overview of the data distribution see Table 1. For an example of what the data would look like when plotted, see Figure 9. In this figure the top graph represents the horizontal bending strain, the middle the vertical and the bottom the total bending strain.

Table 1: Classes present in ROSEN data

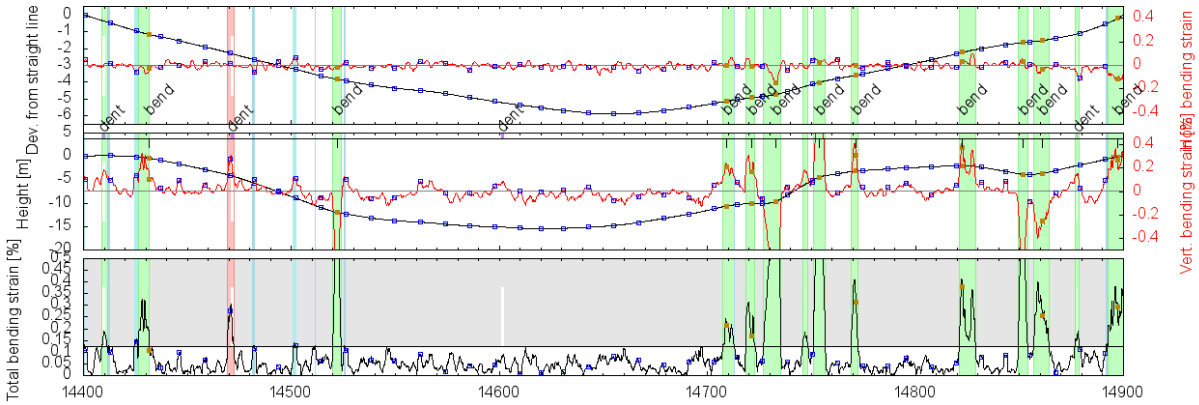


Figure 9: Bending strain snippet provide by ROSEN, with from top to bottom horizontal, vertical, and total bending strain. Along the x-axis is the distance along the pipeline (or the location within the pipeline) and along the y-axis the bending strain value.

Class	Description	Number of samples
1	Bend	208265
2	Strain	3034
3	Other	66251

4.2 Training

While YOLO is a so-called one-shot detector and predicts the bounding boxes all in one go and can thus be also be trained end-to-end, parts of the network are often pre-trained on a similar task before fine-tuning it on bounding box prediction. In the case of YOLO, its backbone would normally first be trained on image classification after which the whole network, now also including the neck and head, would be trained on bounding box prediction. Often, this stage is split in two: training and finetuning. During training, the backbone is frozen and will not be updated during backpropagation. When fine-tuning, the backbone layers will be unfrozen again and trained together with the neck and head. This is done to not lose the knowledge embedded in the backbone when starting with the training of the complete model as the task has now shifted from classification to object detection. Moreover, the layers of the neck and head that are added have not been trained at all yet, so they need to be brought to a comparable level of training as that of the backbone. Once the head and neck have been trained sufficiently, unfreezing the backbone should allow for some finetuning of the performance. To emulate this approach first only the backbone of the model will be trained on a synthetic dataset after which the whole network will be trained for bounding box prediction using the sliding

window mentioned before. The model will also be trained as is, without a separately trained backbone, to evaluate the impact a pre-trained backbone has, if any, on the model.

4.2.1 Data pre-processing

As mentioned before, a sliding window approach will be used to extract correctly sized input data for the models. However, after the windows have been generated the labels have to be formatted correctly as well. The first step is to convert the labels from their original reference frame to that of a single window, making sure the start and end distance of each area are with respect to the window and not the complete pipeline. Subsequently it was decided to leave out windows with no areas present for the train split. This is done to speed up the model training and evaluation during cross validation. The final test split however will contain these empty areas. Then, depending on whether they are used for YOLO or U-Net, they need to be converted slightly different. For YOLO, the start and end distance need to be converted to centre location and width. For U-Net, the labels need to be converted to a segmentation map of the same size as the input.

The synthetic dataset for training the backbone will be generated from the bending strain data. Each sample will contain data for only one area of interest, in this case bend, strain or nothing. However, since there is no concrete nothing class, the other classes that are not bend or strain will be combined to form a nothing class. To further generalize this nothing class, additional nothing areas will be generated from the data where no label is assigned. Though, something that must be taken into account during training is the fact that the strain class contains the lowest number of samples. Moreover, when the other classes are merged to form the 'nothing' class (and enhanced by more nothing areas), strain will be very underrepresented. This will pose an especially big challenge as strain is the most important area that the model should detect. To combat this imbalance, several methods will be evaluated. The first of which will involve weighing strain loss more heavily during training, the second will consist of under sampling the majority class (bend areas) and the last will consist of over sampling the minority class (strain areas), which both have been shown to increase performance [40].

4.2.2 Data Augmentation

Depending on the model being trained, various forms of data augmentation will be employed. When training the backbone, the augmentation will consist of shifting, rotating, scaling, and flipping the data. When training YOLO or U-Net the data will only consist of rotating and flipping the data. These augmentations are allowed under the assumption that the measurements do not contain any tool-measurement artifacts.

Shifting will involve placing the data of interest in a random location within the training sample. This is possible since most of the data is only a fraction of the input size of the backbone and therefore requires padding along its sides to ensure it has the right shape. By varying the bias this padding has, to the left or right of the data, the data can be placed throughout the input window.

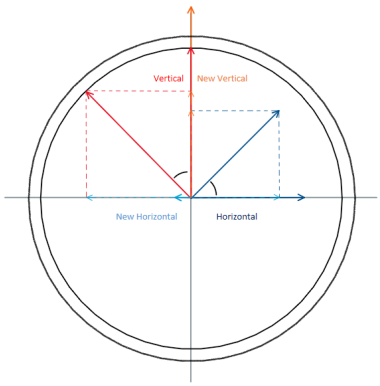


Figure 10: Illustration of rotating data. Blue is the Horizontal strain and Red the vertical. Orange is the new vertical strain and light blue the new horizontal

Rotating the data will involve spinning the pipeline along its axis, so to say. This is accomplished by taking the horizontal and vertical strain as a vector in space and rotating both by the same degrees, see Figure 10 for an illustration. Both the horizontal and vertical strain are rotated, the horizontal and vertical components of the rotated horizontal and vertical strain will then be combined to construct the new horizontal and vertical strain. This is done using formulas 2 and 3. The aim of this is to essentially simulate a bend or strain area occurring in another direction within the pipeline.

$$New\ Horizontal = \cos(\alpha) * Horizontal - \sin(\alpha) * Vertical \quad (2)$$

$$New\ Vertical = \sin(\alpha) * Horizontal + \cos(\alpha) * Vertical \quad (3)$$

Where α is the angle with which to rotate the data

Scaling will involve either up sampling or down sampling a given area. To ensure the data will still fit the input size of the backbone, only areas that are smaller than half the input size are up sampled and areas that are larger are down sampled. The rate of down sampling is random but limited to make ensure that the length of the data will not surpass that of the backbone's input.

Flipping, compared to the others is relatively straight forward. The whole list of datapoints will be flipped around meaning that what was previously the last sample will now be the first and vice versa.

4.2.3 Loss functions

This section will set out the loss function that will be used to train U-Net and YOLO. YOLO's loss function is quite set in stone, having had a few iterations to figure out what works best and tailored specifically to YOLO. U-Net on the other hand has a couple options available. Moreover, the data imbalance could potentially have a greater effect on U-Net than it does on YOLO. This is due to the relationship between fore- and background in semantic segmentation. There will be cases where predicting nothing will already lead to a reasonably accurate prediction since most of the segmentation map is supposed to contain nothing. Therefore U-Net essentially experiences a double class imbalance problem. To combat this and hopefully also the original class imbalance between bend and strain areas, multiple options for the U-Net's loss function will be explored.

4.2.3.1 U-Net

Binary Cross-entropy

By default, U-Net utilizes binary cross-entropy as its loss function, seen in formula 4, which will be used as a baseline to compare the other U-Net loss functions against.

$$\mathcal{L}_{CE} = -\frac{1}{N}\sum_{n=1}^{N}[y_n log\hat{y}_n + (1-y_n)\log(1-\hat{y}_n)] \quad (4)$$

Where y is the binary class label and $\hat{y}$ the prediction.

Focal loss

The first alternative loss function is focal loss [41], which is based on binary cross-entropy and weighs correct prediction less heavily. It takes two hyper-parameters, γ and α which set the degree to which well classified predictions weigh in the total loss and the trade-off between precision and recall respectively (see formula 5).

$$\mathcal{L}_{FL} = -\frac{1}{N}\sum_{n=1}^{N}[ay_n(1-\hat{y}_n)^{\gamma} log\hat{y}_n + (1-y_n)\hat{y}_n{}^{\gamma}\log(1-\hat{y}_n)] \quad (5)$$

With y as the binary class label and $\hat{y}$ the prediction made by the network. γ is the focussing parameter that determines the degree to which the well classified predictions weigh in the loss function. Setting this parameter to 0 turns the focal loss back into binary cross-entropy. Finally, α governs the trade-off between precision and recall and is by default set to 1 (which equals no weighing).

Jaccard loss

IoU based or Jaccard (distance) loss is a loss based in the concept of Intersection over Union (IoU) (see formula 6), a metric often employed to evaluate semantic segmentation models.

$$IoU = \frac{|A \cap B|}{|A \cup B|} \quad (6)$$

where A and B are finite sample sets, or in the case of object detection bounding boxes with for example A being the ground truth and B the predicted bounding box.

As shown in [42] the standard notation for IoU can be adapted to the continuous domain (and for semantic segmentation) resulting in formula 7.

$$\mathcal{L}_J = \frac{1}{N}\sum_{n=1}^{N}\left[1 - \frac{(y_n * \hat{y}_n) + \varepsilon}{(y_n + \hat{y}_n - y_n * y_n) + \varepsilon}\right] \quad (7)$$

With y is the binary class label and $\hat{y}$ the prediction. ε is used to prevent zero division.

Power Jaccard loss

Power Jaccard loss [43] can be seen as the focal loss approach but then for Jaccard loss. Depending on the parameter value chosen for p, incorrectly classified pixels can be weighed more heavily in the loss than correctly classified pixels.

$$\mathcal{L}_{PJ} = \frac{1}{N}\sum_{n=1}^{N}\left[1 - \frac{(y_n * \hat{y}_n) + \varepsilon}{({y_n}^p + {\hat{y}_n}^p - y_n * y_n) + \varepsilon}\right] \quad (8)$$

With y is the binary class label and $\hat{y}$ the prediction. P is the power parameter and caverns the degree to which incorrectly classified pixels more heavily in the loss. This values should always be between 1 and 2, where at 1 the power Jaccard loss acts just like the normal Jaccard loss. ε is used to prevent zero division.

Cosine similarity loss

Cosine similarity loss is like Jaccard in that it is a similarity measure between the prediction and true label, though slightly different. It can be seen in formula 9, which is the form as described by Tensorflow[2]:

$$\mathcal{L}_{cos} = -\sum[\|y_n\|_2 * \|\hat{y}_n\|_2] \quad (9)$$

Where y is the binary class label and $\hat{y}$ the prediction. Both are Tensors of shape [batch_size, d0 , … , dN]

4.2.3.2 YOLO

The loss used by YOLO v4 is constructed of 3 sub losses corresponding to the three predictions made by YOLO per bounding box, namely the bounding box coordinates, the confidence of the bounding box and its class. These are optimized by a GIoU loss, Log loss and binary cross entropy loss respectively.

GIoU

Generalized Intersection over Union (GIoU) [44] is an extension to the IoU metric. It addresses an issue present in the IoU metric (see formula 6) when trying to use it as a loss function for bounding boxes: when there is no overlap IoU will always be 0. This hampers the ability for it to be used in a loss function as it does not allow for the convergence of the predicted bounding box to the true bounding box. GIoU, on the other hand, does, allowing it to be used as a loss function (see formula 10).

$$GIoU = IoU - \frac{|C \setminus (A \cup B)|}{|C|} \quad (10)$$

where A and B are bounding boxes and C is the smallest enclosing convex object of A and B.

GIoU as a distance function, or alternatively the loss function, would subsequently result in formula 11.

$$\mathcal{L}_{GIoU} = 1 - GIoU \quad (11)$$

$$\mathcal{L}_{GIoU} = \frac{1}{N}\sum_{n=1}^{N} 1 - \frac{|y \cap \hat{y}|}{|y \cup \hat{y}|} - \frac{|C \setminus (y \cup \hat{y})|}{|C|} \quad (12)$$

Applying this to the 1D bounding boxes, this results in the following GIoU loss function (formula 13).

$$\mathcal{L}_{GIoU} = \frac{1}{N}\sum_{n=1}^{N} 1 - \frac{|\max(\min(X2_y, X2_{\hat{y}}) - \max(X1_y, X1_{\hat{y}}), 0)|}{|\max(X2_y, X2_{\hat{y}}) - \min(X1_y, X1_{\hat{y}})|} - \frac{|(\max(X2_y, X2_{\hat{y}}) - \min(X1_y, X1_{\hat{y}})) - \max(\min(X2_y, X2_{\hat{y}}) - \max(X1_y, X1_{\hat{y}}), 0)|}{|(\max(X2_y, X2_{\hat{y}}) - \min(X1_y, X1_{\hat{y}})|} \quad (13)$$

Where X1 and X2 represent the start and end distance of a given bounding box.

[2] tf.keras.losses.CosineSimilarity | TensorFlow v2.9.1

Confidence

$$\mathcal{L}_{Cf} = -\frac{1}{N}\sum_{n=1}^{N}[y_n log\hat{y}_n + (1-y_n)\log(1-\hat{y}_n)] \quad (14)$$

With y as the true confidence (either 0 or 1 depending on if there should be a prediction) and $\hat{y}$ the confidence of the prediction.

Class

$$\mathcal{L}_{C} = -\frac{1}{N}\sum_{n=1}^{N}\sum_{c\in classes}[C_n log\hat{C}_n + (1-C_n)\log(1-\hat{C}_n)] \quad (15)$$

With y as the binary class label and $\hat{y}$ the prediction.

Total

The total Loss for YOLO is simply the sum of the three separate losses, resulting in formula 16.

$$\mathcal{L}_{Total} = \mathcal{L}_{GIoU} + \mathcal{L}_{Cf} + \mathcal{L}_{C} \quad (16)$$

4.2.4 Training Parameters

Each model will be trained for a maximum of 100 epochs or until it starts to overfit using a batch size of 512. For the optimizer, an AMSgrad optimizer with a learning rate of 0.0001 will be used. Training will be done on an NVidia Tesla T4.

4.3 Validation

When evaluating both models, three metrics will be used: (mean) average precision (mAP), true positive rate (TPR) and false positive rate (FPR). These metrics, especially mAP, assume the output of the model to be a set of bounding boxes. However, this is not the case for U-Net which outputs a segmentation map. Luckily, this segmentation map can rather easily be converted to bounding boxes as the output is only 1D. Doing this allows both models to be compared using the same metrics.

To determine whether a bounding box is a true positive or false negative an IoU threshold is employed [45]. When the IoU between the predicted bounding box and the true bounding box exceeds this threshold, the prediction is seen as a true positive, otherwise it is seen as a false negative. To make this metric more robust, inspiration will be taken from the MS COCO primary challenge metric. This metric averages the score across a list of 10 IoU threshold ranging from 0.5 to 0.95 with a step size of 0.05 [46]. The same procedure will be applied for the mAP, TPR and FPR.

4.4 Cross Validation

To explore which configurations of both U-Net-1D and YOLOv4-1D perform best, an ablation study will be performed. This ablation study will be executed using k-fold cross validation with k=5. The models will be evaluated one variation at a time, leaving the others to their default value. Parameters that will be evaluated for both models are the input size, the number of input channels, the number of classes in the output, normalization of the data, data augmentation and the impact of under or over sampling of the majority and minority classes respectively. For U-Net specifically, different loss functions will be explored as will the inclusion of mish as an activation function instead of ReLu. For YOLO specifically a weight within the loss function towards strain classes will be explored as well as the effect a pre-train backbone will have on the model's performance. For an overview of the various parameters see Table 2 and Table 3, where the baseline values are made bold.

*Table 2: U-Net parameters (baseline is **bold**)*

Parameter	Value					
Input size	256		**512**		1024	
Loss function	Binary Focal Loss	**Binary Cross-entropy**	Binary Cross-entropy with Smoothing	Jaccard	Power Jaccard	Cosine Similarity
Activation Function	**ReLu**			Mish		
Class imbalance	**None**		Under sampling		Over sampling	
Channels	**Horizontal, Vertical & Total**			Horizontal & Vertical		
Normalization	None	**With respect to each pipe, normal distribution**				
Data augmentation	**None**	Flip		Rotate		Flip and rotate
Classes	**Strain & Bends**		Only strain		Only bends	
Amount of data	25%	50%		75%		**100%**

*Table 3: YOLO Parameters (baseline is **bold**)*

Parameter	Value				
Input size	256		**512**		1024
Class imbalance	**None**	Relative	Constant (10x)	Under sampling	Over sampling
Channels	**Horizontal, Vertical & Total**			Horizontal & Vertical	
Normalization	None	**With respect to each pipe, normal distribution**			
Data augmentation	**None**	Flip		Rotate	Flip and rotate
Backbone	**Not pre-trained**		Pre-trained frozen		Pre-trained & finetuned
Classes	**Strain & Bends**		Only strain		Only bends
Amount of data	25%	50%		75%	**100%**

4.5 Model Comparison

Once the final model configurations have been found through the ablation study, one final model for both YOLOv4-1D and U-Net-1D will be trained on the entire test set and evaluated on the remaining 10% test set. As both models are evaluated using the same metrics, they can then be compared against each other. Since U-Net has previously shown to work well on time series (like) data, it will be used as the baseline to compare YOLO against to see if it too can be said to perform well on time series (like) data.

5 Results

This section will discuss the results of the various experiments that have been run. First the outcome of the cross validation will be discussed per model. For a subset of the results see Table 4 for U-Net-1D and Table 5 for YOLOv4-1D. For a more complete overview of the results see *Appendix C: U-Net-1D performance* and *Appendix D: YOLOv4-1D performance*. When the results of the cross validation have been discussed, the two final configurations for both models, YOLO and U-Net, will be compared against each other.

5.1 U-Net-1D

Of the three different input sizes, an input size of 256 performed the worst, especially with respect to the strain areas. A window size of 1024 shows a slightly different picture, this time outperforming the default size of 512 ever so slightly. However, this is mostly due to an increase in performance for the bends as it too still performed worse on strain.

As for the data augmentation, none of the three approaches made much of an improvement, with flip performing overall worse, especially on strain areas; rotate performing just a little bit better on bend areas but worse on strain; and a combination of the two once again only slightly better on bends and worse on strain areas.

The different loss functions once again do not show one standout improvement in overall mAP performance. It is more a case of trade-offs being made in some cases while others perform worse across the board. This holds true for smoothed cross entropy, Jaccard, Jaccard Power and cosine similarity loss. These all performed worse with respect to the baseline of cross entropy loss. Focal loss, however, is another matter. Both the normal and smoothed variant of focal loss show a decrease in performance with respect to detecting bends, smoothed more so than normal, but a significant increase in performance when detecting strain, once again smoothed more so than normal. While there is none that performs outright best in all cases, both versions of focal loss would in this case be seen as a positive trade-off as strain is the more important metric in this case.

Under sampling the majority and over sampling the minority class both saw an improvement in strain specific mAP scores and a slight decrease in mAP score with respect to bends. However, of the two, under sampling the majority class showed the largest decrease in performance for the bend areas and the smallest increase in strain performance. This is logical as less of the data for bends has been seen by the network, whereas over sampling used all the data for bend areas and provided strain areas with (almost) equal weight during training. This results in over sampling achieving the highest strain performance of any configuration tasked with detecting both bends and strain areas and the highest overall mAP score.

Adding the mish activation or using less input channels did not show any improvement once again with respect to the baseline, where both had similar performance for detecting bends and both performed worse on detecting strain areas. Of the two, the two-channel network performed the worst on strain detection, with a more than 50 percent decrease in performance.

Not normalizing the data had no significant effect on the performance, with a slight increase in performance for bends and a slight decrease with respect to strain areas. However, weighed against possible training instabilities it is not worth using instead of the normalized input.

Detecting the areas individually instead of combined showed an overall slight increase in performance for bends and a major increase in performance of almost 250 percent for strain areas. While using two separate networks would increase the inference time, the increase in performance would suggest that this might be worth it, especially since U-Net is able to process more than 3000 windows per second.

The final best configuration for U-Net seems to be one involving over sampling the minority class and possibly the use of focal loss as the loss function. To evaluate whether this is indeed the case, this combination has also

Table 4: Results for U-Net cross-validation. The Results are given relative to the baseline, where each row represents a single change to the network. For example, input size 256 indicates that the difference between the baseline and this variation is the input size which has changed to 256 instead of 512. For each variation, the average mAP across the folds is reported (Total) as well as the AP for the individual classes (Bend and Strain).

	Total	Bend	Strain
Baseline	**0.3936**	**0.7545**	**0.0328**
Input size 256	-11.03%	-8.23%	-75.49%
Input size 1024	4.29%	5.30%	-19.02%
Flip	-0.41%	0.69%	-25.78%
Rotate	1.24%	1.66%	-8.43%
Flip & Rotate	-0.21%	0.87%	-25.20%
Cross-entropy Smooth	-6.09%	-3.01%	-77.09%
Focal	-9.66%	-15.44%	123.52%
Focal Smooth	-18.50%	-25.40%	140.65%
Cosine	-72.98%	-75.34%	-18.56%
Jaccard	-90.85%	-91.89%	-66.92%
Jaccard Power	-90.05%	-90.72%	-74.55%
Under sample	1.40%	-4.15%	129.38%
Over sample	7.06%	-1.69%	208.72%
mish	-0.97%	-0.10%	-20.96%
2 Channel	-1.14%	1.41%	-59.91%
Raw	2.86%	3.09%	-2.60%
Bends		1.61%	
Strain			241.48%
25 %	-15.19%	-11.96%	-89.71%
50 %	-1.40%	0.40%	-42.98%
75 %	-1.51%	0.46%	-46.97%
Oversample Focal	-0.97%	-9.20%	188.72%

Table 5: Results for YOLO cross-validation. The Results are given relative to the baseline, where each row represents a single change to the network. For example, input size 256 indicates that the difference between the baseline and this variation is the input size which has changed to 256 instead of 512. For each variation, the average mAP across the folds is reported (Total) as well as the AP for the individual classes (Bend and Strain).

	Total	Bend	Strain
Baseline	**0.2936**	**0.5569**	**0.0304**
Input size 256	-5.96%	-9.41%	57.26%
Input size 1024	-9.76%	-7.24%	-56.05%
Flip	3.95%	2.73%	26.42%
Rotate	0.38%	0.65%	-4.56%
Flip & Rotate	-5.21%	-6.28%	14.53%
weighted	-2.90%	-3.10%	0.63%
Weighted relative	1.52%	3.06%	-26.79%
Under sample	-14.29%	-17.68%	47.80%
Over sample	8.85%	-2.77%	222.18%
2 Channel	-2.46%	-4.44%	33.94%
Raw	-0.07%	-2.91%	52.02%
Pre-Trained	-19.83%	-18.58%	-42.69%
Pre-Trained finetuned	-19.53%	-19.24%	-24.97%
Bends		-11.74%	
Strain			143.07%
25%	-21.17%	-19.28%	-55.76%
50%	-10.57%	-10.02%	-20.63%
75%	-6.63%	-9.77%	50.94%
2 Ch Flip Over sample	14.93%	-1.95%	324.51%

been trained and evaluated using cross-validation. However, while it does perform better than the baseline in regards to the strain performance, it performs worse on every metric with respect to only over sampling the minority class. The final configuration for U-Net will therefore consist of an input size of 512, cross-entropy loss and oversampling the minority classe.

Finally, to answer the question of whether even more data would increase the performance, when looking at the performance achieved for bends using 25, 50 and 75 percent of the data, this seems to have reached a plateau. At a quarter of the data there was still a significant decrease in performance, however after the 50% threshold the performance for detecting bends has levelled out. Strain, on the other hand, is once again a different story. At 25 percent there was a significant decrease in performance of almost 90% which did not raise above 40% even at 70% of the data. Therefore, more data would, in the case of strain areas, most likely still lead to an increase in performance.

5.2 YOLOv4-1D

When comparing the three different window sizes for YOLO, the comparison seems almost flipped with respect to U-Net. In the case of YOLO both the input size of 256 and 1024 performed worse on bends, however this time the size of 256 performed best on strain (almost the complete opposite with respect to the performance gain). However, in both cases overall performance was either on par or slightly worse than the baseline of 512.

Looking at the impact data augmentation has on the model, it can be seen that rotating the data performed worst. It also performed worse than the baseline, at least with respect to strain areas. Bends performed either on par or slightly better than the baseline. As for flipping and a combination of flipping and rotating the data these both performed better on strain, with flipping performing overall better, seeing a slight increase in performance for bend areas and a decent improvement for strain areas.

Combatting the class imbalance found between the strain and bend areas through weighing each differently did not seem to have much of an improvement. When applying a constant weight to strain areas, bends overall performed marginally worse and strain only showed a very minor improvement in mAP. Of note is however the more significant increase in recall. When applying a weight relative to the number of samples of each class present, no improvement is made regarding strain areas and only a very minor increase in mAP for bends. Both under sampling the majority class and over sampling the minority class showed an improvement in mAP for strain areas, where over sampling showed the most improvement. Moreover, under sampling showed a significant decrease in performance with respect to Bends leading to a worse overall mAP score. This is logical, as less of the data containing bend areas has been seen by the network. Over sampling, on the other hand, showed only a minor decrease in mAP performance for Bends leading to an overall best score for total mAP.

When reducing the number of input channels, YOLO at first shows a picture similar to that of U-Net with an overall decrease in performance. However, whereas U-Net performed slightly better on bends and significantly worse on strain, YOLO performs slightly worse on bends and significantly better on strain.

Using raw input data as opposed to normalized input data did show a significant increase in strain performance this time with only a minor decrease in mAP performance for bends. However, it was also, naturally, more difficult to get this model to converge as the raw data led to more instabilities during training.

Almost entirely across the board, the pre-trained backbone showed worse performance than the baseline; both the initial and finetuned versions. The only exception would be the false positive rate, which in case of the finetuned model showed a small increase in performance for strain areas.

Training the model separately on strain and bend areas once again lead to an increase in performance for strain areas, though not as large as seen with U-Net. This time resulting in a little more than 140 percent

improvement in mAP. The model trained only on bends however performed overall worse as compared to the baseline, which is in contrast with a slight increase in mAP performance seen in U-Net for bends.

The final best configuration for YOLO seems to be a combination of the flip data augmentation, 2 channel input and the use of oversampling the data (and an input size of 512). To confirm this, this model was also trained using cross validation achieving performance in line with expectations. The model managed to achieve the highest mAP score for strain so far with only a slightly lower mAP score for bends than the baseline and slightly higher than the over sample version. While there are models that performed better on bends than this configuration (e.g. flip, rotate and weighted-relative), this trade-off is deemed worthwhile as the performance of detecting strain areas is most of concern. Making this the final model for YOLO.

Finally, having a look at the impact a reduced dataset has on YOLO's performance, a slightly less clear picture can be drawn. In general, there still seems to be a trend of increased performance with more data. The only outlier for this seems to be the strain performance for YOLO at 75 percent data, which managed to perform better than the baseline. Perhaps this is caused by the lines along which the data has been split into the reduced datasets, though this will need to be investigated further to give a definitive answer.

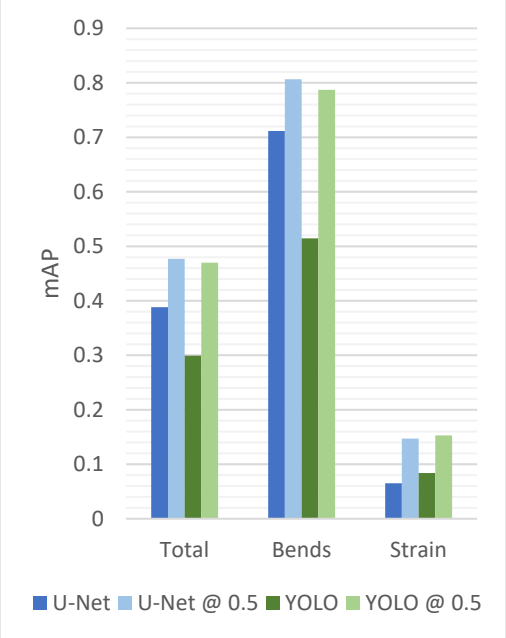


Figure 11: mAP results for U-Net vs YOLO, with in blue U-Net and in green YOLO. The dark blue and green are mAP averaged across multiple IoU thresholds and the lighter at an IoU of 0.5.

5.3 U-Net-1D vs YOLOv4-1D

Having trained the final model for both YOLOv4-1D and U-Net-1D on the complete train set and evaluated on the test set the results shown in Table 6, Table 7 and Figure 11 were obtained. Depending on the metric weighed most heavily either model can be said to perform best. When it comes to overall performance, U-Net seems to perform best. Especially when it comes to detecting Bends in which the performance disparity is the greatest between the two models. YOLO on the other hand has a slight edge in terms of detecting strain areas, which are the areas that are of biggest concern. While YOLO lags slightly in terms of TPR when it comes to strain areas, it achieves a better mAP score and FPR.

Table 6: Final score for U-Net-1D (using Oversampling) and YOLOv4-1D (using 2 channel input, flip data augmentation and oversampling) results

	mAP			**TPR**			**FPR**		
	Total	Bends	Strain	Total	Bends	Strain	Total	Bends	Strain
U-Net-1D	0.3883	0.7115	0.0650	0.7418	0.7499	0.2629	0.6836	0.1923	0.9969
YOLOv4-1D	0.2992	0.5145	0.0838	0.5998	0.6056	0.2437	0.4664	0.3637	0.9793

Table 7: Final score (at 0.5 IoU) for U-Net-1D (using Oversampling) and YOLOv4-1D (using 2 channel input, flip data augmentation and oversampling)

	mAP @ 0.5			**TPR @ 0.5**			**FPR @ 0.5**		
	Total	Bends	Strain	Total	Bends	Strain	Total	Bends	Strain
U-Net-1D	0.4769	0.8066	0.1472	0.8099	0.8160	0.4475	0.6545	0.1212	0.9948
YOLOv4-1D	0.4700	0.7871	0.1530	0.8020	0.8091	0.3668	0.2865	0.1499	0.9688

6 Discussion

6.1 Framework feasibility

Based on the performances achieved, it can be said that the framework proposed in this paper (see Figure 8) is indeed a feasible approach for area of interest detection in bending strain data. Furthermore, the setup of the current framework allows for relatively easy and straightforward integration in other software packages currently being used for bending strain analysis. Moreover, the framework has the adaptability to work with any network that outputs either a list of bounding boxes or a segmentation map for a given window.

6.2 Bending strain models

To extract areas of interest from bending strain data a 1D version of both YOLO and U-Net has been used. Both went through a set of experiments using cross-validation to evaluate the various possible setups to come to a final configuration that performs best.

6.2.1 Influence of window size (RQ 2.3)

Both models were trained using several different window sizes, with varying results. U-Net showed the clearest results, having a window size of 512 perform on par or better in each metric compared to a window size of 256 or 1024. The reason why especially strain areas performed worse for a window size of 256 could be because strain areas are varied more in size. While bend areas are all below about 50 samples in width, strain still has a reasonable number of areas sized 100 to 200 samples and a small subset of 200 or greater (see Figure 12). However, the windows are not created around the areas that might be present in them instead they are created one after each other. Therefore, the chance that a strain area is split across two windows is greater than a bend area is split between two areas. This would explain why strain areas performed much worse on a 256 sized window as compared to bend areas.

YOLO showed a somewhat stranger picture. It performed worse on every metric when using a window size of 256 or 1024 as compared to a window size of 512 except for the strain areas for the 256 sized windows. They showed an increase of almost 60 percent. At first this seems somewhat strange; however, a possible explanation might be found in YOLO's anchor boxes. The different sizes for the anchor boxes are determined beforehand using k-means on the sizes of the areas of interest present in the dataset. However, no distinction is made between the classes. Therefore, seeing as strain areas are a lot less prevalent, the sizes might be skewed somewhat to the sizes for bend areas. This could explain the increase in performance, as with a

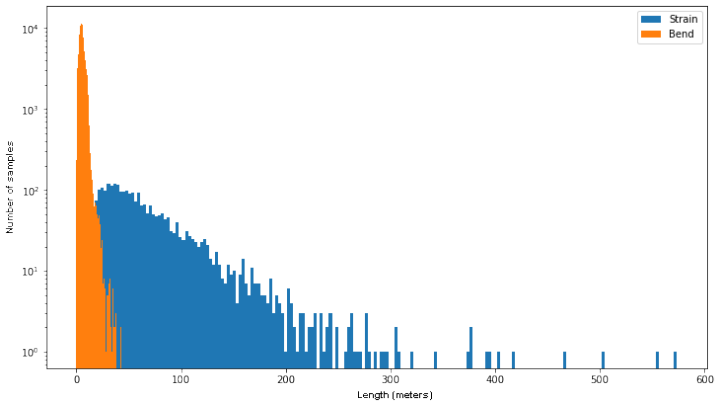


Figure 12: Number of areas per length in meters for both strain and bend areas of interest

window size of 256 the bigger strain areas have a higher chance of being split in two and thereby finding an anchor box closer in size than they would have with windows sized 512 or 1024. This would make it easier to learn to size these areas correctly as the difference to learn between the ground truth and anchor box is smaller.

6.2.2 Data quantity and quality (RQ 2.4.1)

As mentioned before, the data used is very imbalanced with about 70 times more bend areas than strain areas. The biggest struggle therefore has been to get the models to detect strain areas correctly. To help address this issue a couple of possible methods have been tried. For both models, under- and oversampling has been tried as well as splitting the detection task in two: training the model on either bends or strain. Next to this some model specific adaptations have been tried. For YOLO this involved weighing the minority class more heavily in the loss, either through a constant factor of 10x or relative to the number of strain areas in relation to bend areas in that batch. For U-Net this was one of the reasons to try out different loss functions and different configurations of loss functions (e.g. label smoothing).

6.2.2.1 Re-sampling and task separation

In both models, under- and oversampling saw an increase in strain performance with a minor decrease in performance for bends in the case of U-Net and oversampling with YOLO. Under sampling on YOLO on the other hand saw a more substantial decrease in bend performance. For both models oversampling looks like a good way to combat the class imbalance. As for splitting the detection task in two to be handled by two sperate models (of the same kind), this showed somewhat promising results. U-Net showed an increase in performance across the board, achieving the highest scores for strain areas and a minor increase in performance for bend areas as well. YOLO also improved significantly on strain areas, though less than the improvement achieved by oversampling. However, Bend areas showed a decrease in performance. This might be explained by the fact that YOLO has a separate loss for the localization, the confidence, and the class of the areas of interest. Thus, by training only on bend areas the more varied strain areas cannot help to improve the localization and confidence performance of bend areas.

6.2.2.2 Model specific adaptations (RQ 2.1.1)

Applying a weight to the minority class showed almost no improvement in strain detection or performed worse than the baseline without any weight. Using a constant weight saw a decrease in bend detection with a very minor increase in strain detection. Using a relative weight also did not show any significant improvement, leading to a minor increase in bend detection but an across-the-board worse performance for strain detection. All in all, weighing the minority class did not prove to be a possible solution for the imbalanced data. U-Net also got some modifications to its loss function, though this time in the form of new (variations of the) loss functions. This came in the form of focal loss, cosine similarity and Jaccard loss. Of these loss functions, the default cross-entropy loss and focal loss were also used in conjunction with label smoothing.

The only loss function that showed some improvement was the focal loss, which showed in both cases a significant increase in strain performance though it did trade in some bend performance. All the other loss functions, as well as the cross-entropy loss with label smoothing, showed a decrease in performance across the board.

6.2.2.3 Other causes for the performance disparity

On top of the class imbalance, another possible reason has been found that might explain the performance disparity between bend and strain areas. Namely the quality of the data, as towards the end of this research some inconsistencies were discovered in the data. The first one, which almost exclusively affects strain areas, is the fact that the threshold for considering something to be a strain area is not entirely consistent between the pipelines. Different clients might have different standards to which the pipelines should be evaluated (e.g. some might use the default 0.125 percent [1] as a threshold for strain areas, while others might require them to only be mentioned at a threshold of 0.15 percent). Moreover, the way in which the bending strain is originally generated includes taking the median average across a certain window size (typical window sizes range from 1 to 3 meters), which also seems to differ between pipelines. This can result in a 2x increase/decrease in bending strain depending on the window size used. However, it is not yet clear to what degree this differs between the bending strain data used in this paper, but it is something that should be investigated.

6.2.2.4 Data augmentation (RQ 2.4.2)

With the aim of improving performance, especially of the minority class, data augmentation has also been evaluated for YOLO and U-Net. This consisted of two methods: flipping and rotating the data. Both were expected to deliver some performance gains, especially flipping the data as the bending strain should be direction invariant. This was the case for YOLO, where flipping the data lead to a clear improvement in performance, especially for strain areas. Rotating did not show any improvement though. This could perhaps be due to the completely random nature of the rotating augmentation. The degrees with which the data was rotated was also randomized. As mentioned before, strain areas are generally classified as such if any of the channels exceeds a certain threshold. If rotated just right, this could lead to neither of the horizontal or vertical channels exceeding such a threshold increasing the difficulty of detecting strain areas. Just like with YOLO, U-Net too showed a decrease in performance when applying the rotating augmentation. However, it also showed a decrease in performance when flipping the data which was rather unexpected. Why this happened is still unclear.

6.2.2.5 *Amount of data (RQ 2.4)*

To assess the impact more data would have on the performance of both models, tests were also run where the models were trained on a subset of the data to simulate increasing the data available. Both models were trained using 25, 50 and 75 percent of the data (in addition to the 100 percent used to train the baseline). See Figure 13 for an illustration of these results. U-Net still showed minor performance gains with more data added, though this can mostly be attributed to an increase in strain performance. Performance on detecting bend areas does not seem to benefit from more data. YOLO showed a slightly different picture. Here overall performance kept seeing improvement, which can in part be attributed to a still increasing performance on bend areas. Strain areas were a little bit of an anomaly, where peak performance was achieved at 75 percent. Running the test again resulted in similar results. Why this happened is still unclear. However, both seem to suggest that more data can still lead to an increase in performance, either in detecting bend areas, strain areas or both.

6.2.3 U-Net-1D configuration (RQ 2.1, RQ 2.1.2)

Next to the various loss functions evaluated for U-Net, a different activation function has also been tried. This came in the form of the mish activation function used by YOLO v4. The hope was that it would also show some performance increase for U-Net, just like it had done for YOLO v4, however that was not the case. It performed on par in terms of bend performance but showed a decrease in strain performance. Combined with the outcome of the various loss functions the best configuration seemed to at first be a combination of focal loss and oversampling. However, training this version on cross validation as well showed an overall worse performance than the default loss function of cross-entropy loss and oversampling. The final best configuration for U-Net therefore turned out to be the overall default structure of U-Net, with the max pooling layers exchanged for average pooling layers and trained using the cross-entropy loss and oversampling of the minority class.

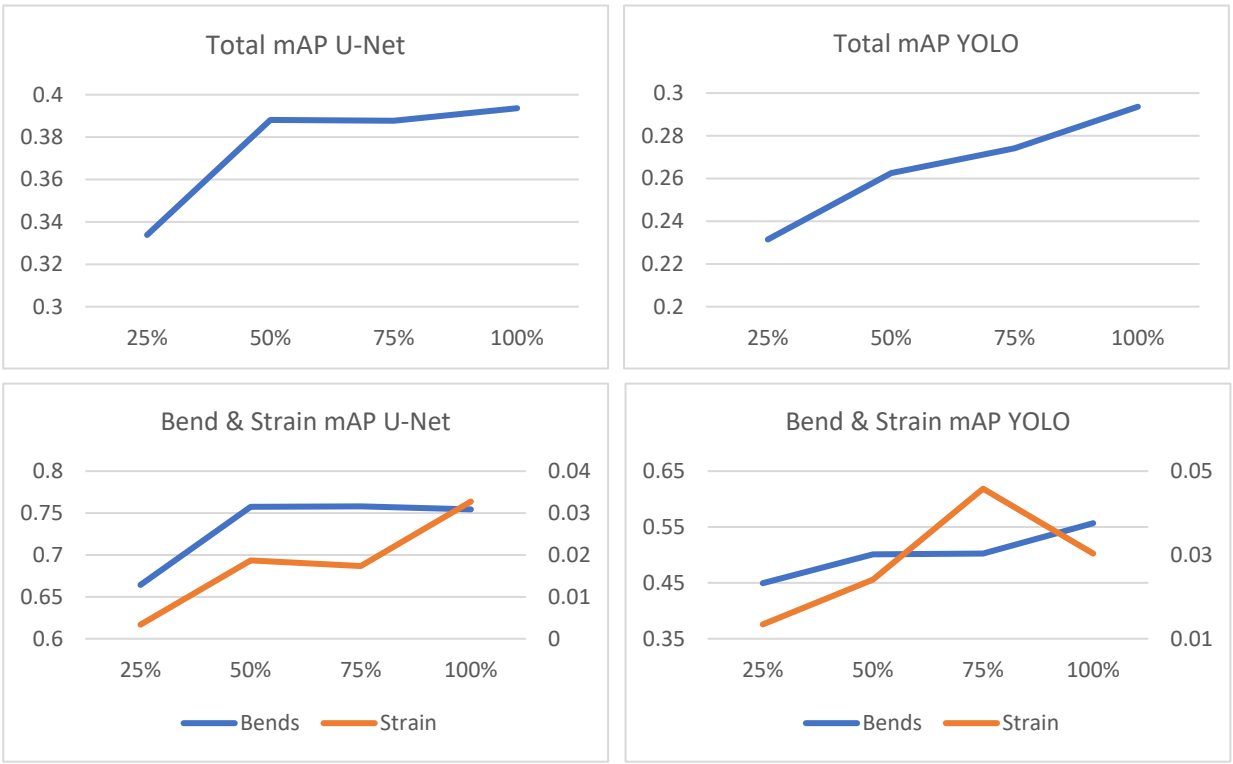


Figure 13: mAP performance for U-Net and YOLO at different subset sizes. The top two graphs show the total mAP performance for both models, the bottom two graphs show the mAP performance for Bend and Strain areas with on the left axis the scores for the bend areas and on the right axis the scores for the strain areas.

6.2.4 YOLOv4-1D configuration (RQ 2.2.2)

Concerning YOLO's setup, it was also evaluated what affect pre-training the backbone would have on the model's performance. It was found, however, that pre-training the backbone on a classification dataset generated from the same dataset that would later be used to train the complete model did not lead to any increase in performance. This might be due to the data used in pre-training not being able to introduce any new information. Perhaps taking a different time series classification task for pre-training will lead to an increase in performance. Together with the results found earlier, the optimal configuration for YOLO turns out to a model using only two input channels (horizontal and vertical), applying flip data augmentation, and making use of oversampling.

6.2.5 Model performance (RQ 2.2, RQ 2.2.1)

Both models have shown promising results in detecting bending strain areas of interest. In most metrics YOLO performs on par with U-Net showing the potential for an object detection model to be used on 1D data for area of interest detection. However, both models still have a way to go in terms of performance before they can be considered good enough for bending strain detection based on the requirements set out earlier. These requirements demand at least a 90 percent true positive rate and at most a 50 percent false positive rate. The performance for detecting bends does come quite close already, achieving 75 percent true positive rate and a 19 percent false positive rate using U-Net. When relaxing the IoU threshold slightly, this performance gets even closer to the bar set by the requirements, but even then, does not pass it yet. Strain areas on the other hand have quite a way to go, achieving only a 26 percent true positive and a 99.7 percent false positive rate in the case of U-Net and a 24 percent true positive and 97.9 false positive rate in the case of YOLO.

6.3 Limitations

While each model takes a different approach to the detection of areas of interest, segmentation versus bounding boxes, it is not the only difference between the two. When comparing network sizes YOLO is by far the biggest and most complex of the two. This is also reflected both in the training and inference times. Where U-Net on average takes 110 seconds per epoch, YOLO hovers around the 287 seconds, more than 2.5 times that of U-Net (at the time of publishing, the inference times could not be collected). Moreover, due to the higher model complexity YOLO requires more system memory to run. This would generally not be an issue for systems with a dedicated GPU, however it is not guaranteed that the evaluators that will might be using these models will have access to such a GPU (and a sufficiently powerful one). Therefore, there does not seem to be much benefit currently to using YOLO over U-Net, seeing as the detection performance is either close to equal or better when using U-Net. Moreover, U-Net is able to detect areas of interest in less time while requiring less system resources.

As mentioned before, the data collected for this research came from several sources and spans a range of different pipeline diameters. Since the data collection for this research is still in its early stages, all the data has been used interchangeably. However, there is the possibility that there is a significant difference in the data taken from different regions and different pipeline diameters. This could result in the models performing worse on a specific subset of the pipelines. When the models were being tested though, there was not yet enough data available to train them on a specific subset only. Moreover, in the time frame for this research it was no longer possible to evaluate whether the models do indeed perform significantly worse on a specific subset of the data.

7 Conclusion

In this paper both a U-Net and a YOLO based model have been adapted to work on 1D time series like data and implemented for automatic detection of bending strain areas of interest. This was accomplished by replacing the 2D convolution layers in both networks with their 1D counterparts. Next to this, U-Net also had

its max pooling layers replaced with average pooling layers. Finally, both networks had their output size adjusted to accommodate the 1D version of their outputs. For U-Net this results in an output of the same length as the input and the number of channels equal to the number of classes. For YOLO this essentially results in a reduced output size as it now only has to predict one coordinate (x) and one length (width) instead of two coordinates (x and y) and two lengths (width and height). The aim of this was to assess to what degree bending strain areas of interest could automatically be detected. An additional aim of developing both a 1D U-Net and a YOLO based model was to evaluate whether an object-detection based model, such as YOLO, could perform equally well on a 1D detection task as U-Net can. Since U-Net has previously shown to work well on similar tasks, it was used as a baseline to compare YOLO against.

Both models were successfully adapted to the 1D domain and trade blows when it comes to the various performance scores. Following this it can be concluded that YOLO would be a valid candidate for similar detection tasks on time series like data. However, neither network performed as desired when it comes to detecting bending strain areas. While bends are found reasonably successfully (with an average precision of 0.71 and 0.51 for U-Net and YOLO respectively), strain are more difficult to find (with an average precision of 0.065 and 0.083 for U-Net and YOLO respectively). If a slightly less stringent threshold is employed to determine if a prediction is a true or false positive, the results are a little more promising. With an IoU threshold of 0.5 both models achieve similar performance on bends (0.80 and 0.79 for U-Net and YOLO respectively) and strain areas (0.147 and 0.153 for U-Net and YOLO respectively). Bend performance is looking promising, especially combined with a greater than 80 percent recall rate for both models and a false call rate of 12 and 15 percent for U-Net and YOLO respectively. However even with the more relaxed threshold the strain performance of both models is still not good enough to be used and lags behind the bend performance significantly. This is likely to be (partially) due to a combination of the class imbalance and inconsistencies within the bending strain dataset.

8 Future work

While YOLOv4-1D seems to work well on time series like data when taking U-Net-1D as a baseline, this is only one case. Future work could therefore look at other domains and evaluate whether YOLO is a valid candidate in those domains as well. As for improvements to be made in this domain, the most obvious would be improving the models' performance on strain areas, for both YOLO and U-Net. Furthermore, it would also be useful to ascertain whether this deficiency is due to a fault in either of the models, their make-up, loss function or something else, or potentially the data itself.

One thing that will remain a challenge is the huge class imbalance present in the current dataset. It is expected that this will remain a concern once more data becomes available. Examining other methods to address this issue could therefore be helpful as well, as rebalancing the dataset showed an increase in performance in both U-Net and YOLO. A possible solution for this might be simulating the bending strain to generate more strain areas instead of or on top of rebalancing the dataset. On top of this, the current dataset should be examined further for the possible inconsistencies mentioned before and standards set for any new data that is to be added.

Another avenue to explore is the effect the different diameters of the pipelines have on the detection of bending strain areas of interest. Currently the dataset contains data from a number of different pipeline diameters all mixed together, which could be impacting the performance both positively and negatively. Finally, splitting up the task into separate single class prediction seems to have shown some merit, at least during cross validation and would also be worthwhile to investigate further.

9 References

[1] S. Berends, "CLASSIFICATION OF BENDING STRAIN AREAS," University of Twente, 2018.

[2] S. Liu, H. Wang, R. Li, and B. Ji, "A Novel Feature Identification Method of Pipeline In-Line Inspected Bending Strain Based on Optimized Deep Belief Network Model," *Energies*, vol. 15, no. 4, 2022, doi: 10.3390/en15041586.

[3] C. H. Hsieh, Y. S. Li, B. J. Hwang, and C. H. Hsiao, "Detection of atrial fibrillation using 1D convolutional neural network," *Sensors (Switzerland)*, vol. 20, no. 7, 2020, doi: 10.3390/s20072136.

[4] H. A. Dau *et al.*, "The UCR time series archive," *IEEE/CAA J. Autom. Sin.*, vol. 6, no. 6, pp. 1293–1305, 2019, doi: 10.1109/JAS.2019.1911747.

[5] Z. Wang, W. Yan, and T. Oates, "Time series classification from scratch with deep neural networks: A strong baseline," in *2017 International Joint Conference on Neural Networks (IJCNN)*, May 2017, vol. 57, no. 7, pp. 1578–1585, doi: 10.1109/IJCNN.2017.7966039.

[6] M. Perslev, M. H. Jensen, S. Darkner, P. J. Jennum, and C. Igel, "U-Time: A fully convolutional network for time series segmentation applied to sleep staging," *Adv. Neural Inf. Process. Syst.*, vol. 32, no. NeurIPS 2019, pp. 1–12, 2019.

[7] S. Khessiba, A. G. Blaiech, K. Ben Khalifa, A. Ben Abdallah, and M. H. Bedoui, "Innovative deep learning models for EEG-based vigilance detection," *Neural Comput. Appl.*, vol. 33, no. 12, pp. 6921–6937, 2021, doi: 10.1007/s00521-020-05467-5.

[8] K. Duraj, N. Piaseczna, P. Kostka, and E. Tkacz, "Semantic Segmentation of 12-Lead ECG Using 1D Residual U-Net with Squeeze-Excitation Blocks," *Appl. Sci.*, vol. 12, no. 7, pp. 1–13, 2022, doi: 10.3390/app12073332.

[9] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," *IEEE Access*, vol. 9, pp. 16591–16603, May 2015, doi: https://doi.org/10.48550/arXiv.1505.04597.

[10] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "YOLOv4: Optimal Speed and Accuracy of Object Detection," 2020, [Online]. Available: http://arxiv.org/abs/2004.10934.

[11] H. Ismail Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P. A. Muller, "Deep learning for time series classification: a review," *Data Min. Knowl. Discov.*, vol. 33, no. 4, pp. 917–963, 2019, doi: 10.1007/s10618-019-00619-1.

[12] S. Minaee, Y. Boykov, F. Porikli, A. Plaza, N. Kehtarnavaz, and D. Terzopoulos, "Image Segmentation Using Deep Learning : A Survey," pp. 1–22.

[13] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 2, pp. 386–397, 2020, doi: 10.1109/TPAMI.2018.2844175.

[14] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *Total Perform. Scorec.*, pp. 159–183, Jun. 2015, doi: 10.4324/9780080519340-12.

[15] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking Atrous Convolution for Semantic Image Segmentation," 2017, [Online]. Available: http://arxiv.org/abs/1706.05587.

[16] Prof.Dr.E.A.Zanaty, Prof.Dr.Mahmoud M.Abdel-Aty, and Khalid abdel-wahab ali, "Comparing U-Net convolutional network with mask R-CNN in Nuclei Segmentation," *IJCSNS Int. J. Comput. Sci. Netw. Secur.*, vol. 22, no. 3, Sep. 2022, doi: https://doi.org/10.22937/IJCSNS.2022.22.3.35.

[17] T. T. P. Quoc, T. T. Linh, and T. N. T. Minh, “Comparing U-Net Convolutional Network with Mask R-CNN in Agricultural Area Segmentation on Satellite Images,” *Proc. - 2020 7th NAFOSTED Conf. Inf. Comput. Sci. NICS 2020*, pp. 124–129, 2020, doi: 10.1109/NICS51282.2020.9335856.

[18] E. Alfaro, X. B. Fonseca, E. M. Albornoz, C. E. Martinez, and S. C. Ramrez, “A Brief Analysis of U-Net and Mask R-CNN for Skin Lesion Segmentation,” *IWOBI 2019 - IEEE Int. Work Conf. Bioinspired Intell. Proc.*, pp. 123–126, 2019, doi: 10.1109/IWOBI47054.2019.9114436.

[19] T. Zhao, Y. Yang, H. Niu, Y. Chen, and D. Wang, “Comparing U-Net convolutional networks with fully convolutional networks in the performances of pomegranate tree canopy segmentation,” no. October, p. 64, 2018, doi: 10.1117/12.2325570.

[20] M. Perslev, S. Darkner, L. Kempfner, M. Nikolic, P. J. Jennum, and C. Igel, “U-Sleep: resilient high-frequency sleep staging,” *npj Digit. Med.*, vol. 4, no. 1, pp. 1–12, 2021, doi: 10.1038/s41746-021-00440-5.

[21] J. M. Wu, Y. C. Liu, and D. T. H. Chang, “SigUNet: Signal peptide recognition based on semantic segmentation,” *BMC Bioinformatics*, vol. 20, no. Suppl 24, pp. 1–14, 2019, doi: 10.1186/s12859-019-3245-z.

[22] Z. Shang, Y. Xia, L. Chen, and L. Sun, “Damping ratio identification using attenuation responses extracted by time series semantic segmentation,” *Mech. Syst. Signal Process.*, vol. 180, no. August 2021, p. 109287, 2022, doi: 10.1016/j.ymssp.2022.109287.

[23] T. Wen and R. Keyes, “Time Series Anomaly Detection Using Convolutional Neural Networks and Transfer Learning,” 2019, [Online]. Available: http://arxiv.org/abs/1905.13628.

[24] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 580–587, 2014, doi: 10.1109/CVPR.2014.81.

[25] R. Girshick, “Fast R-CNN,” *Proc. IEEE Int. Conf. Comput. Vis.*, vol. 2015 Inter, pp. 1440–1448, Apr. 2015, doi: 10.1109/ICCV.2015.169.

[26] Z. Wang, W. Yan, and T. Oates, “Time Series Classification from Scratch with Deep Neural Networks : A Strong Baseline,” pp. 1578–1585, 2017.

[27] P. Adarsh, P. Rathi, and M. Kumar, “YOLO v3-Tiny: Object Detection and Recognition using one stage improved model,” *2020 6th Int. Conf. Adv. Comput. Commun. Syst. ICACCS 2020*, pp. 687–694, 2020, doi: 10.1109/ICACCS48705.2020.9074315.

[28] S. S. A. Zaidi, M. S. Ansari, A. Aslam, N. Kanwal, M. Asghar, and B. Lee, “A Survey of Modern Deep Learning based Object Detection Models,” pp. 1–18, 2021, doi: 10.1016/j.dsp.2022.103514.

[29] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016-Decem, pp. 779–788, 2016, doi: 10.1109/CVPR.2016.91.

[30] J. Redmon and A. Farhadi, “YOLO9000: Better, faster, stronger,” *Proc. - 30th IEEE Conf. Comput. Vis. Pattern Recognition, CVPR 2017*, vol. 2017-Janua, no. April, pp. 6517–6525, Dec. 2017, doi: 10.1109/CVPR.2017.690.

[31] J. Redmon and A. Farhadi, “YOLOv3: An Incremental Improvement,” 2018, [Online]. Available: http://arxiv.org/abs/1804.02767.

[32] A. L. Maas, A. Y. Hannun, and A. Y. Ng, “Rectifier nonlinearities improve neural network acoustic models,” *ICML Work. Deep Learn. Audio, Speech Lang. Process.*, vol. 28, 2013.

[33] D. Misra, “Mish: A Self Regularized Non-Monotonic Activation Function,” 2019, [Online]. Available: http://arxiv.org/abs/1908.08681.

[34] V. Nair and G. E. Hinton, “Rectified linear units improve restricted boltzmann machines,” *ICML’10 Proc. 27th Int. Conf. Int. Conf. Mach. Learn.*, pp. 807–814, 2010, doi: 10.5555/3104322.3104425.

[35] K. He, X. Zhang, S. Ren, and J. Sun, “Spatial pyramid pooling in deep convolutional networks for visual recognition,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 8691 LNCS, no. PART 3, pp. 346–361, 2014, doi: 10.1007/978-3-319-10578-9_23.

[36] Y. Zhao, R. Han, and Y. Rao, “A new feature pyramid network for object detection,” *Proc. - 2019 Int. Conf. Virtual Real. Intell. Syst. ICVRIS 2019*, pp. 428–431, 2019, doi: 10.1109/ICVRIS.2019.00110.

[37] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia, “Path Aggregation Network for Instance Segmentation,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 8759–8768, 2018, doi: 10.1109/CVPR.2018.00913.

[38] Z. Liu, G. Gao, L. Sun, and Z. Fang, “HRDNet: High-resolution Detection Network for Small Objects,” *Proc. - IEEE Int. Conf. Multimed. Expo*, pp. 1–6, Jun. 2020, doi: 10.1109/ICME51207.2021.9428241.

[39] U. Rebbapragada, P. Protopapas, C. E. Brodley, and C. Alcock, “Finding anomalous periodic time series : An application to catalogs of periodic variable stars,” *Mach. Learn.*, vol. 74, no. 3, pp. 281–313, 2009, doi: 10.1007/s10994-008-5093-3.

[40] R. Batuwita and V. Palade, “Efficient resampling methods for training support vector machines with imbalanced datasets,” *Proc. Int. Jt. Conf. Neural Networks*, 2010, doi: 10.1109/IJCNN.2010.5596787.

[41] T. Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar, “Focal Loss for Dense Object Detection,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 2, pp. 318–327, 2020, doi: 10.1109/TPAMI.2018.2858826.

[42] M. A. Rahman and Y. Wang, “Optimizing Intersection-Over-Union in Deep,” *Int. Symp. Vis. Comput.*, pp. 234–244, 2016, doi: 10.1007/978-3-319-50835-1.

[43] D. Duque-Arias *et al.*, “On power jaccard losses for semantic segmentation,” *VISIGRAPP 2021 - Proc. 16th Int. Jt. Conf. Comput. Vision, Imaging Comput. Graph. Theory Appl.*, vol. 5, no. Visigrapp, pp. 561–568, 2021, doi: 10.5220/0010304005610568.

[44] H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. Reid, and S. Savarese, “Generalized intersection over union: A metric and a loss for bounding box regression,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2019-June, pp. 658–666, 2019, doi: 10.1109/CVPR.2019.00075.

[45] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, “The pascal visual object classes (VOC) challenge,” *Int. J. Comput. Vis.*, vol. 88, no. 2, pp. 303–338, 2010, doi: 10.1007/s11263-009-0275-4.

[46] “Microsoft COCO detection evaluation metrics.” https://cocodataset.org/#detection-eval.

10 Appendix A: U-Net-1D model architecture

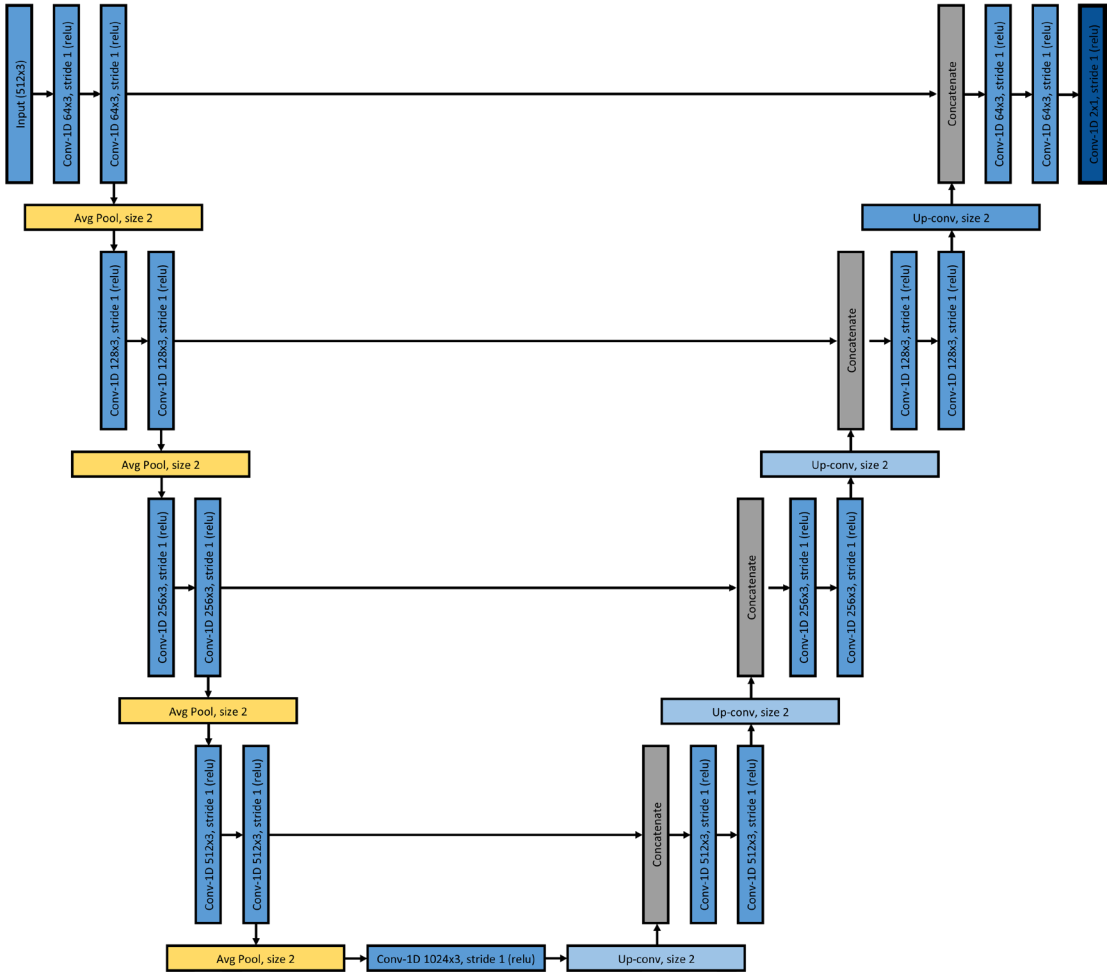


Figure 14: U-Net-1D Model architecture. The model can be split into two, the encoder and the decoder. The encoder can be found on the left-hand side and the decoder on the right. The input and output of the model are of the same width, 512 in this case, but differ in number of channels. The number of output channels is defined by the number of classes to be predicted, which in this case is 2.

11 Appendix B: YOLOv4-1D model architecture

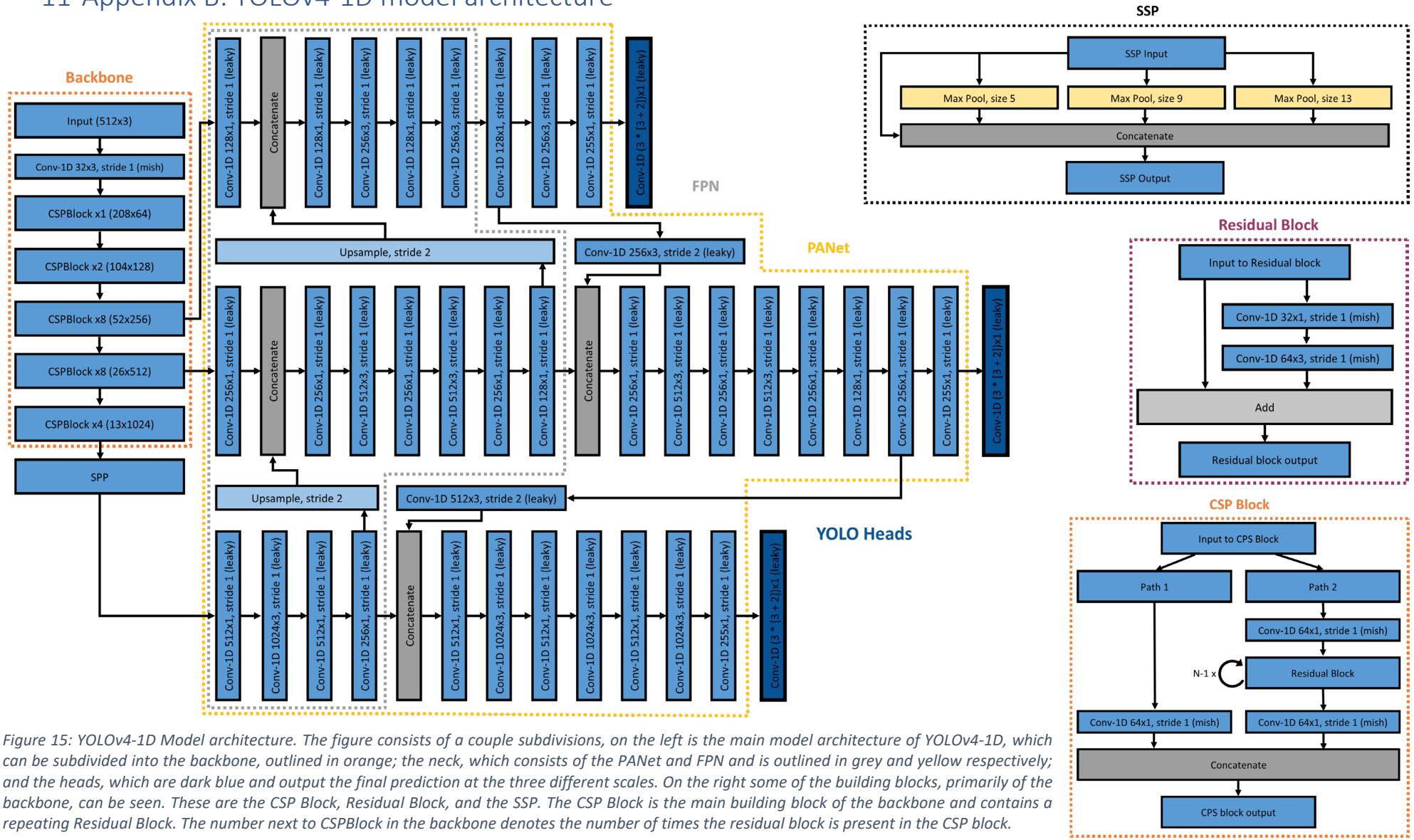


Figure 15: YOLOv4-1D Model architecture. The figure consists of a couple subdivisions, on the left is the main model architecture of YOLOv4-1D, which can be subdivided into the backbone, outlined in orange; the neck, which consists of the PANet and FPN and is outlined in grey and yellow respectively; and the heads, which are dark blue and output the final prediction at the three different scales. On the right some of the building blocks, primarily of the backbone, can be seen. These are the CSP Block, Residual Block, and the SSP. The CSP Block is the main building block of the backbone and contains a repeating Residual Block. The number next to CSPBlock in the backbone denotes the number of times the residual block is present in the CSP block.

12 Appendix C: U-Net-1D performance

12.1 mAP performance

	Class Average			Bends			Strain		
	mAP @ 0.5	mAP @ 0.75	mAP*	mAP @ 0.5	mAP @ 0.75	mAP*	mAP @ 0.5	mAP @ 0.75	mAP*
Baseline	**0.47077**	**0.41190**	**0.39362**	**0.85686**	**0.80251**	**0.75449**	**0.08469**	**0.02129**	**0.03275**
Input size 256	0.43392	0.38009	0.35021	0.84186	0.75706	0.69239	0.02597	0.00311	0.00803
Input size 1024	0.46434	0.42215	0.41050	0.85967	0.82732	0.79447	0.06901	0.01698	0.02652
Flip augmentation	0.46161	0.41203	0.39199	0.86067	0.80795	0.75968	0.06255	0.01611	0.02431
Rotate augmentation	0.47092	0.41705	0.39850	0.86508	0.81379	0.76700	0.07675	0.02032	0.02999
Flip & Rotate augmentation	0.46211	0.41007	0.39279	0.85634	0.80581	0.76108	0.06787	0.01433	0.02450
CrossentropySmooth	0.43231	0.39226	0.36966	0.84387	0.78034	0.73181	0.02075	0.00419	0.00750
Focal	0.47468	0.37062	0.35559	0.79898	0.67977	0.63797	0.15038	0.06146	0.07321
FocalSmooth	0.45911	0.33288	0.32082	0.75580	0.59909	0.56282	0.16242	0.06666	0.07881
CosineSimilarity	0.22861	0.08897	0.10635	0.39166	0.16005	0.18604	0.06556	0.01789	0.02667
Jaccard	0.09312	0.02328	0.03601	0.15194	0.04164	0.06119	0.03430	0.00492	0.01083
JaccardPower	0.10252	0.02337	0.03916	0.18065	0.04245	0.06999	0.02439	0.00428	0.00834
Undersample	0.49524	0.41832	0.39914	0.83331	0.77464	0.72315	0.15718	0.06201	0.07513
Oversample	0.51277	0.43486	0.42142	0.82605	0.78184	0.74174	0.19950	0.08788	0.10111
mish	0.46031	0.40775	0.38979	0.84913	0.79967	0.75370	0.07149	0.01583	0.02589
2 Channel	0.45117	0.41017	0.38912	0.86262	0.81364	0.76510	0.03972	0.00669	0.01313
Raw	0.48330	0.42540	0.40486	0.87709	0.83294	0.77782	0.08952	0.01786	0.03190
Bends				0.86526	0.81305	0.76661			
Strain							0.24410	0.09095	0.11184
25 %	0.40379	0.35843	0.33382	0.79761	0.71494	0.66426	0.00997	0.00192	0.00337
50 %	0.45429	0.40669	0.38811	0.85613	0.80193	0.75754	0.05245	0.01144	0.01868
75 %	0.45127	0.40660	0.38768	0.85186	0.80321	0.75799	0.05068	0.01000	0.01737
Oversample Focal	0.51005	0.40739	0.38981	0.82184	0.73439	0.68506	0.19826	0.08039	0.09456

12.2 TPR and FPR performance

	Class Average		Bends		Strain		Class Average		Bends		Strain	
	TPR @ 0.5	TPR*	TPR @ 0.5	TPR*	TPR @ 0.5	TPR*	FPR @ 0.5	FPR*	FPR @ 0.5	FPR*	FPR @ 0.5	FPR*
Baseline	0.85445	0.78017	0.87153	0.79926	0.27729	0.13872	0.48285	0.52564	0.26023	0.31911	0.98477	0.99236
Input size 256	0.84049	0.73045	0.85743	0.74630	0.09156	0.03438	0.36935	0.45293	0.28740	0.38087	0.98679	0.99528
Input size 1024	0.85221	0.80524	0.87338	0.82917	0.27928	0.14386	0.47178	0.50074	0.25765	0.29513	0.98149	0.99044
Flip augmentation	0.85369	0.78299	0.87562	0.80583	0.21817	0.10991	0.44130	0.48632	0.26195	0.31936	0.98338	0.99188
Rotate augmentation	0.86107	0.78977	0.87955	0.81001	0.25441	0.12544	0.47188	0.51404	0.26375	0.32041	0.98438	0.99222
Flip & Rotate augmentation	0.85086	0.78253	0.87003	0.80335	0.23990	0.11560	0.42151	0.46630	0.22624	0.28394	0.98152	0.99113
CrossentropySmooth	0.83634	0.76077	0.85905	0.78283	0.10173	0.04738	0.36685	0.42153	0.26858	0.42024	0.98733	0.99581
Focal	0.81699	0.70118	0.83139	0.71725	0.43705	0.26075	0.77676	0.80751	0.47603	0.54634	0.99363	0.99622
FocalSmooth	0.79044	0.64364	0.80195	0.65637	0.48231	0.29236	0.89575	0.91486	0.62547	0.69152	0.99697	0.99821
CosineSimilarity	0.51608	0.31336	0.52522	0.31980	0.30878	0.16886	0.83946	0.90345	0.71171	0.82681	0.99286	0.99622
Jaccard	0.32436	0.16610	0.32682	0.16772	0.22007	0.09880	0.91126	0.95745	0.72852	0.86968	0.99345	0.99704
JaccardPower	0.374763	0.187626	0.381430	0.191016	0.184681	0.090782	0.931871	0.965765	0.769158	0.887026	0.994538	0.997480
Undersample	0.83703	0.76239	0.84724	0.77545	0.44699	0.26371	0.70387	0.72987	0.19853	0.26624	0.99396	0.99647
Oversample	0.82657	0.76522	0.83459	0.77607	0.48909	0.29678	0.64634	0.67215	0.11404	0.17558	0.99258	0.99571
mish	0.84454	0.77686	0.86170	0.79546	0.22387	0.10489	0.34923	0.39934	0.17511	0.23649	0.97848	0.98981
2 Channel	0.85571	0.78664	0.87759	0.80904	0.15420	0.06809	0.36798	0.41841	0.22158	0.28180	0.98262	0.99238
Raw	0.86668	0.79752	0.88344	0.81568	0.23836	0.11258	0.26673	0.32504	0.11380	0.18193	0.97072	0.98627
Bends			0.87889	0.81010					0.24051	0.29906		
Strain					0.26344	0.26344					0.86104	0.92350
25 %	0.79071	0.69764	0.81628	0.72095	0.05127	0.02199	0.27361	0.35400	0.23890	0.32319	0.97446	0.99001
50 %	0.85004	0.77813	0.87070	0.79983	0.18233	0.08475	0.41308	0.45968	0.26517	0.32246	0.98536	0.99349
75 %	0.84619	0.78022	0.86374	0.79900	0.19008	0.08596	0.41425	0.45969	0.21913	0.27755	0.98456	0.99283
Oversample Focal	0.82835	0.73545	0.84112	0.75088	0.46887	0.27406	0.73281	0.76337	0.29480	0.36908	0.99063	0.99475

13 Appendix D: YOLOv4-1D performance

13.1 mAP performance

	Class Average			Bends			Strain		
	mAP @ 0.5	mAP @ 0.75	mAP*	mAP @ 0.5	mAP @ 0.75	mAP*	mAP @ 0.5	mAP @ 0.75	mAP*
Baseline	**0.44924**	**0.32545**	**0.29360**	**0.84060**	**0.62243**	**0.55685**	**0.05787**	**0.02847**	**0.03035**
Input size 256	0.45236	0.28470	0.27609	0.81440	0.52451	0.50445	0.09032	0.04490	0.04774
Input size 1024	0.42806	0.28548	0.26495	0.82105	0.56265	0.51655	0.03507	0.00830	0.01334
Flip augmentation	0.45885	0.34154	0.30521	0.84645	0.64710	0.57204	0.07125	0.03598	0.03837
Rotate augmentation	0.44736	0.32864	0.29473	0.83914	0.62975	0.56048	0.05559	0.02754	0.02897
Flip & Rotate augmentation	0.44428	0.29701	0.27831	0.81685	0.56403	0.52186	0.07170	0.02999	0.03476
Weighted	0.45003	0.31022	0.28507	0.83578	0.59334	0.53960	0.06427	0.02711	0.03054
Weighted relative	0.44289	0.33608	0.29805	0.84370	0.65071	0.57388	0.04209	0.02145	0.02222
Undersample	0.46404	0.23719	0.25164	0.82261	0.43887	0.45841	0.10547	0.03552	0.04486
Oversample	0.49823	0.34826	0.31960	0.81724	0.60408	0.54140	0.17922	0.09244	0.09780
2 Channel	0.45208	0.30781	0.28638	0.83303	0.57329	0.53211	0.07113	0.04233	0.04066
Raw	0.47451	0.30664	0.29341	0.86510	0.56661	0.54067	0.08391	0.04668	0.04615
Pre-Trained	0.41672	0.24139	0.23539	0.79699	0.46724	0.45338	0.03645	0.01554	0.01740
Pre-Trained finetuned	0.41042	0.23661	0.23625	0.76693	0.45598	0.44973	0.05391	0.01724	0.02277
Bends				0.82180	0.51792	0.49146			
Strain							0.16569	0.05944	0.07378
25%	0.42340	0.21905	0.23145	0.81022	0.43084	0.44947	0.03657	0.00726	0.01343
50%	0.42900	0.27626	0.26257	0.80528	0.53277	0.50104	0.05273	0.01975	0.02409
75%	0.46010	0.27769	0.27413	0.82655	0.51465	0.50244	0.09365	0.04073	0.04582
2-Ch Flip Oversample	0.50920	0.36915	0.33744	0.80276	0.61485	0.54601	0.21564	0.12345	0.12886

13.2 TPR and FPR performance

	Class Average		Bends		Strain		Class Average		Bends		Strain	
	TPR @ 0.5	TPR*	TPR @ 0.5	TPR*	TPR @ 0.5	TPR*	FPR @ 0.5	FPR*	FPR @ 0.5	FPR*	FPR @ 0.5	FPR*
Baseline	0.83771	0.61432	0.86255	0.63316	0.15267	0.09453	0.37458	0.54080	0.33346	0.50999	0.92465	0.95305
Input size 256	0.82536	0.58115	0.84007	0.59205	0.23066	0.14701	0.40293	0.57961	0.36569	0.55316	0.93053	0.95501
Input size 1024	0.81403	0.57271	0.84694	0.59695	0.12772	0.06429	0.37581	0.56181	0.32983	0.52905	0.91015	0.95475
Flip augmentation	0.84196	0.62784	0.86495	0.64550	0.18201	0.11682	0.33438	0.50265	0.29562	0.47304	0.91209	0.94333
Rotate augmentation	0.83604	0.61563	0.85828	0.63254	0.14192	0.08780	0.32763	0.50261	0.29561	0.47892	0.91253	0.94492
Flip & Rotate augmentation	0.81649	0.58173	0.83785	0.59795	0.22183	0.12586	0.40214	0.56696	0.26981	0.48419	0.93861	0.95962
Weighted	0.83614	0.59880	0.85787	0.61525	0.19067	0.11352	0.37687	0.55308	0.32391	0.51481	0.94347	0.96630
Weighted relative	0.83830	0.62502	0.86176	0.64293	0.11240	0.07059	0.30555	0.48013	0.27470	0.45731	0.88956	0.92893
Undersample	0.83453	0.54017	0.84897	0.55070	0.31809	0.16711	0.52237	0.69400	0.33830	0.57472	0.98190	0.99065
Oversample	0.82185	0.60936	0.83152	0.61749	0.39264	0.25201	0.29781	0.47529	0.16930	0.37945	0.95627	0.97152
2 Channel	0.83436	0.59182	0.85907	0.60969	0.15464	0.10129	0.36582	0.54925	0.34556	0.53467	0.88704	0.92566
Raw	0.85634	0.60132	0.87897	0.61755	0.16214	0.10534	0.21193	0.44755	0.19844	0.43791	0.80990	0.87625
Pre-Trained	0.80829	0.52543	0.83414	0.54336	0.10512	0.06245	0.46126	0.63940	0.42197	0.61168	0.95867	0.97579
Pre-Trained finetuned	0.77919	0.52252	0.80123	0.53815	0.18492	0.10117	0.43195	0.62221	0.36888	0.57750	0.96428	0.97972
Bends			0.84653	0.57468					0.34440	0.55400		
Strain					0.18016	0.18016					0.81373	0.89563
25%	0.81817	0.52428	0.85107	0.54601	0.09752	0.05061	0.54425	0.70327	0.53764	0.69915	0.75534	0.89956
50%	0.81801	0.56740	0.84217	0.58475	0.16297	0.09242	0.46484	0.63582	0.43192	0.61224	0.94985	0.97303
75%	0.83440	0.57663	0.85138	0.58893	0.25096	0.15172	0.38342	0.57069	0.32592	0.53075	0.94049	0.96389
2-Ch Flip Oversample	0.80896	0.61363	0.81639	0.62006	0.46025	0.31605	0.27510	0.45083	0.11354	0.32765	0.95670	0.97024