A Defect Classication Methodology for Sewer Image Sets with Convolutional Neural Networks

Meijer, Dirk; Scholten, Lisa; Clemens, Francois; Knobbe, Arno

DOI
10.1016/j.autcon.2019.04.013

Publication date
2019

Document Version
Accepted author manuscript

Published in
Automation in Construction

A Defect Classification Methodology for Sewer Image Sets with Convolutional Neural Networks

Dirk Meijer[a,b,*], Lisa Scholten[b], Francois Clemens[b,c], Arno Knobbe[a]

[a] *Leiden University, Niels Bohrweg 1, 2333CA Leiden, The Netherlands*
[b] *Delft University of Technology, Stevinweg 1, 2628CN Delft, The Netherlands*
[c] *Deltares, Department of Hydraulic Engineering, P.O. box 177, 2600MH Delft, The Netherlands*

Abstract

Sewer pipes are commonly inspected *in situ* with CCTV equipment. The CCTV footage is then reviewed by human operators in order to classify defects in the pipes and make a recommendation on possible interventions. This process is both labor-intensive and error-prone. Other researchers have suggested machine learning techniques to (partially) automate the human review of this footage, but the automated classifiers are often validated in artifial testing setups, leading to biased results that do not translate directly to operational impact. In this work, we discuss suitable evaluation metrics for this specific classification task — most notably 'specificity at sensitivity' and 'precision at recall' — and the importance of using a validation setup that includes a realistic ratio of images with defects to images without defects, and a sufficiently large dataset. We also introduce *'leave-two-inspections-out'* cross validation, designed to eliminate a data leakage bias that would otherwise cause an overestimation of classifier performance. We designed a convolutional neural network (CNN) and applied this validation methodology to automatically detect the twelve most common defect types in a dataset of over 2 million CCTV images. With this dataset and our validation methodology, our CNN outperforms the state-of-the-art. Classification performance was highest for intruding and defective connections and lowest for porous pipes. While the CNN is not capable of fully automated classification at sufficient performance levels, we determined that if we augment the human operator with the CNN, this may reduce the required human labor by up to 60.5%.

Keywords: Automated Classification, CCTV Inspection, Convolutional Neural Networks, Image Processing, Sewer Asset Management, Classifier Validation

1. Introduction

Properly operating urban drainage systems are essential to ensure public health, safety, and productivity in cities, but not enough is known about the failure mechanisms that lead to decreased performance or loss of functionality [1]. To understand the condition of the system and to assess which assets need repair or rehabilitation, inspections are performed. The largest share of the operation and maintenance cost across the technical assets in the system is usually spent on the sewer pipes. For their inspection, CCTV inspection is commonly performed: a 'pipe inspection vehicle' is lowered into a manhole, where it records video footage which is reviewed by trained operators. The operators identify defects and possible indications of defects in the footage, and assign this a severity rating between 1 (no intervention necessary) and 5 (immediate intervention necessary).

One of the major shortcomings of this method is that these severity ratings and the defect identification prior to it are highly subjective, and have been shown to differ not only between operators, but also for the same operator at different time points [2, 3]. The SewerSense project [4] aims to solve this shortcoming by automating parts of the inspection process to, on the one hand objectify the inspection results, and on the other hand facilitate decision making in sewer asset management by providing more accurate information than an arbitrary urgency scale.

In this article, a possible method to automate the inspection process is demonstrated and shown to be viable. While the performance of the method is noteworthy, the authors consider the most important contribution of this paper not to be this method itself, but rather the methodology used to validate these results and assess their impact if used in practice.

1.1. Image Classification

Image classification is the primary way in which the SewerSense project attempts to address the automation of the inspection process. This classification assumes that we have *training data*, consisting of a set of images of CCTV

*Corresponding Author
Email addresses: meijerdwj@liacs.leidenuniv.nl (Dirk Meijer), l.scholten@tudelft.nl (Lisa Scholten), f.h.l.r.clemens@tudelft.nl (Francois Clemens), a.j.knobbe@liacs.leidenuniv.nl (Arno Knobbe)

footage, each of which has an assigned *label*, which indicates whether specific types of defects are present and visible in the image. The classifier infers a statistical relation between the images and the labels, which allows it to make predictions about the labels of images that we don't know the true labels for, such as recently recorded images that still require assessment.

Traditionally, the automated classification of images was done with extracted image features [5], which were known to capture information that is less visible in raw pixel values. Recently, this approach has been mostly replaced by convolutional neural networks (CNNs, explained in more detail is Appendix A) [6]. CNNs employ "end-to-end" learning: the raw pixel values are used as inputs, and the CNN learns the feature extractions as well as how these features relate to the labels. This allows for extracted image features that are more specialized to the task. There is one main downside to this approach: there are a lot more parameters to fit, as also the extracted features need to be inferred from the image data. Two resulting limitations are that a lot more data is required to fit all these parameters, and hyperparameter[1] optimization becomes more difficult as the hyperparameter search space (how many filters and of what shape) increases drastically compared to traditional methods.

The impact of the data availability problem can be lessened by using a network that has been pre-trained on a different set of images [7], but then we may also reduce the benefit that the CNN may have in training the convolutional filters specifically to the data and the task at hand. This approach is often favored over a random initialization of the network parameters to save time [8].

1.2. Classification Result Validation

To assess the performance of a trained classifier, we need a test set that is independent of the training set. To use (part of) the same training set as the test set introduces a bias and means we are not measuring how well the classifier performs, but only how well it can recognize before-seen data. Since two independent data sets may be difficult to come by, often a portion of the training set is set apart to be used as the test set [9]. The training and test set are not independent in such a scenario and likely contain the same sampling bias, but it is often the best we can do.

To assess the performance accurately, some variance in the samples in the training set is required, which means many samples are required, and a significant portion of the training set may have to be set apart. A significant reduction in size of the training set could itself impact the performance negatively, leading us to underestimate the actual performance of the classifier, due to lack of training data. An often used technique to circumvent this problem is k-fold cross validation [9]. Instead of setting apart a large portion of the training set, the training set is divided into *folds*, non-overlapping subsets. A single fold is used as the test set, and the remaining folds are used as the training set. This process is then repeated until each fold is used as the test set once, and the performance on each fold can be averaged to estimate the performance of the classifier. To practically eliminate the impact of having a reduced training set, we can use *leave-one-out* cross validation [9], where each fold is exactly 1 sample in size, and there are as many folds as there are samples in our training set.

Besides a test set, the performance metrics have to be defined. The most common performance metric used for classification is the *accuracy*, the percentage of correctly classified samples. However, the performance metric should be chosen based on the task at hand, and we feel that accuracy is not a good choice for this particular problem (further discussion on this in section 4). Most performance metrics can be thought of as some function of the false positive rate (FPR, or Type I error) and the false negative rate (FNR, or Type II error). A classifier can often be tuned after it has been trained, making it essentially a family of classifiers. In such cases the performance metric is also a function of this tuning, and it can be worthwhile to use metrics that are independent of which member of the family of classifiers is used. Examples of such metrics are the receiver operator characteristic, or the Pareto-boundary of any combination of metrics [9].

1.3. Related Work

Many researchers have already applied machine learning techniques to the task of automating sewer inspections. The validation of such methods is often of less note in such articles. As actual defect rates are often very low, around an order of magnitude of 1% of images of frames captured by CCTV — in our dataset we found 0.8% images with defects — it is curious that many authors will test their methods on artificial test sets in which 50% of images contain defects. We feel that such a result might be interesting in a vacuum, but unless the pipes later classified with the system have similar extreme defect rates, it gives little to no indication of the actual *'real-world performance'* of a classifier. A relevant selection of research is discussed in this section.

Chae and Abraham [10] use a (non-convolutional) neural network to learn various attributes in relation to the existence and severity of cracks from images of the inner surface of sewer pipes. Their neural network is trained on 20 images and tested on 13 images, so the actual applicability remains unclear.

Yang and Su [11] compare two SVM approaches and a neural network, trained on wavelet filter responses of images. The classifiers were only applied to images containing defects, and subsequently used to classify *what* defect was present in the image. This means no information is available regarding the false detections in images without defects.

[1] Parameters that the algorithm is initialized with, that are not fitted during the training phase.

Guo et al. [12] use image registration and the absolute pixelwise difference between images to classify image regions as defective or healthy. The method is tested on a dataset consisting of 51 images of defective pipes and 52 images of healthy pipes, with reported accuracy and false alarm rates.

Halfawy and Hengmeechai [13] present an algorithm for crack detection in CCTV inspections, based on a Sobel filter and morphological operations. As the model is based on expert knowledge, it does not require a large dataset to train, and it was tested on a dataset with 50 images containing cracks and 50 images not containing cracks.

Halfawy and Hengmeechai [14] improve on their previous work, now training an SVM with varying kernels with HoG features extracted from CCTV images, and report more meaningful performance metrics such as precision and AUROC. The experiments are still performed on a test set that consists of 50% images with defects, so it still tells us very little of real-world performance.

Kumar et al. [15] are one of the first to use convolutional neural networks to exploit end-to-end learning in sewer CCTV defect detection. They focus on three different defect types and train the network three times, once for the detection of each defect. They also report the precision as one of their performance measures and use a training set consisting of 12,000 images, but their test sets also consist of 50% images with defects, again limiting their obtained results to such an artificial scenario. In our work, we reimplemented their suggested convolutional neural network and performed tests on our dataset, which more accurately represents a real-world scenario.

Myrans et al. [16] trains an SVM and a random forest on extracted GIST features from CCTV images. They use 25-fold cross validation and provide the ROC curve along with the misclassification rates for various defect types, but work with a dataset that consists of approximately 37% images with defects, which is not representative of a realistic scenario. In [17] they combine both the SVM and the random forest on a dataset in which *"approximately half"* the images contained defects, and obtain results superior to either individual classifier. Again, the validation results are not representative of a real-world scenario because of the high prevalence of defects.

1.4. Contribution

Beyond the previous work by other authors, we have aimed to provide a mature, rigorous, and detailed methodological framework to assist future research. We summarize our contributions as follows:

- We have applied state-of-the-art machine learning techniques to the problem of sewer CCTV inspection classification.
- We have used an exceptionally large dataset of over 2.2 million images, taken at high resolution and without human interference.
- We have utilized a multi-label classification approach that allows different defect types to be detected in the same images by a *single classifier*, reducing training time and allowing features learnt for the classification of one class to be utilized in the classification of another class.
- We have introduced a methodological workflow for the classification of sewer CCTV images and validation of this classification, based on:
  - Specificity-at-sensitivity and precision-at-recall quality metrics
  - Leave-two-inspections-out Cross Validation and appropriate metric reduction across inspections, to prevent data leakage during cross validation
  - No rebalancing of the test set
  - No human interference in data collection, to prevent data leakage at time of data collection
- Classes were based on the widely used EN 13508-2 standard [18] and the performance metrics used translate directly to operational impact, meaning that our results should be readily interpretable for practitioners.

2. Data Exploration

A dataset has been kindly provided to us by Dutch sewer inspection company *vandervalk+degroot*. The data has two components: the images themselves, and the accompanying inspection reports. The data encompasses 30 inspections from 11 Dutch municipalities, for a total of 2,202,582 images from 3,350 different concrete pipes ranging in diameter between 300 mm and 1000 mm.

2.1. Image data

The images have been collected with the RapidView IBAK Panoramo® pipeline inspection system [19]. While the Panoramo software can be used to inspect the pipe in a virtual 3D environment, for this study we merely used the same 2D images used to create these 3D environments as input. The Panoramo system does not record video but rather still images with a strobe light, spaced 5 cm apart. This allows for improved image quality, without the need to stop the cart from moving. The system is equipped with a front-facing and back-facing camera, each with a 185° wide angle lens. The images from the back-facing camera are slightly occluded by parts of the cart and the chain that lowered it into the pipe, so our dataset contains only the images from the front-facing camera.

The images are 24-bit RGB, 1040×1040 pixels, JPEG images. No information was given about the compression level, but the images range from 19 KiB to 447 KiB. Four randomly selected sample images are shown in Figure 1.

An important feature of the images recorded by the Panoramo system is that the images are spatially aligned.

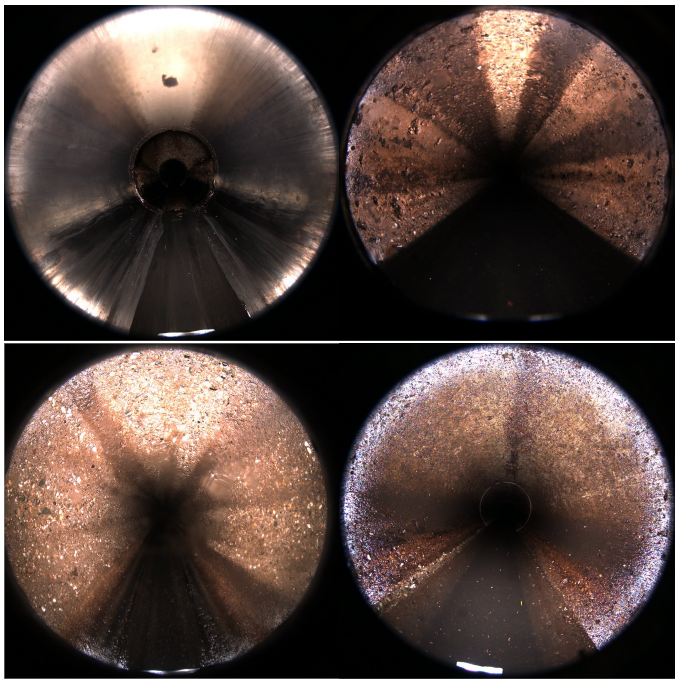

Figure 1: Randomly selected sample images from the dataset.

After the device is lowered into the sewer pipe, the operator aligns the camera with the center of the pipe before starting the recording. This allows the Panoramo software to stitch the images together into a three-dimensional, virtual environment, but it also allows us to consider the images to be of the same modality, allowing a 1-on-1 comparison between two images.

As the images from the Panoramo system are meant for offline processing, the operator does not pan, rotate, or zoom the camera during the recording, as they might with other CCTV feeds. This is a very important distinction, because we feel that being able to automatically classify images where a human operator has already isolated, centered, and zoomed in on the defects, as is apparently the case in some previous studies, does not achieve much in terms of "automating classification". We would not be able to tell if a classifier trained and tested on such data actually recognizes the defects, or rather recognizes the camera rotation, for example, which is indicative of the human inspector's attention. To take steps towards fully automated inspection, we should aim to classify images that were recorded without human intervention, other than starting the system.

2.2. Inspection Reports

Besides the images, each of the 30 inspections is accompanied by an inspection report, containing all points of interest along the pipes referenced according to the European standard coding norm EN 13508-2 [18], as annotated from visual review by human operators. An example of an entry from the tabular datafile that generated this report could be:

```
38.40m  BBAC2  @Blick=38.38;91;72;90;0;
```

This indicates that at 38.40 meters from the start of the pipe, a defect was found with main code `BBA` (roots), characterization `C` (complex mass of roots), and quantification `2` (pipe diameter reduced by $\leq 10\%$)

From these entries, we can assign contextual labels to the images. We have selected the twelve most commonly occurring defects (that are not simply expected landmarks such as pipe joints, an overview is shown in table 1) and matched these to specific images, using the location of the entries. Because we know the Panoramo system's images are spaced exactly 5 cm apart, it is a relatively simple task to determine which entries in the report should be visible in each image. It's important to note that the best we can aim for with such a dataset is to label the images as well as (and no better than) a human operator would.

The `@Blick` entry is added by the Panoramo software and can be used to recreate the exact view in the virtual environment the operator was looking at when this defect was recorded. In this research, the `@Blick` entry was not used.

In the end, we have a set of roughly 2.5 million images, and for each image a list of twelve boolean values (True/False), telling us whether or not that specific defect is present in the image. In the next section, we will go into detail on how this data is modelled.

3. Methodology

3.1. Classification

Supervised classification (henceforth simply classification) is a machine learning task where the goal is to infer a relationship between 'objects' and 'labels' by generalizing a *training set*, a collection of such objects for which the label is known. This generalization can then later be used to classify objects for which the label is not known.

We consider a dataset $\mathbf{X}$ consisting of N real-valued vectors of equal length:

$$\begin{aligned} &\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\} \\ &\mathbf{x}_i \in \mathbb{R}^d \qquad\qquad \forall i \in [1, N] \end{aligned}$$

Each of these vectors also has an associated label, $\mathbf{y}_i$, belonging to one of m distinct classes:

$$\begin{aligned} &\mathbf{Y} = \{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_N\} \\ &\mathbf{y}_i \in \{c_1, c_2, \ldots, c_m\} \qquad \forall i \in [1, N] \end{aligned}$$

In our case, each vector $\mathbf{x}_i$ is an image consisting of P pixels (treated as real values, for simplicity's sake, and flattened to a single dimension), and each label $\mathbf{y}_i$ is a boolean vector of length D, with one boolean value for each defect we distinguish, as noted in Section 2:

$$c_j \in \{+, -\}^D \qquad \forall j \in [1, m]$$

(where a positive label indicates presence and a negative label indicates absence of a particular defect in that particular image.) As such, each image has a true label,

a vector of boolean values, which reflects what defects are visible in that image.

The goal of classification is then to find a function F that relates the vectors to their label:

$$F(\mathbf{x}_i) = \mathbf{y}_i \qquad \forall i \in [1, N]$$

We can state that $F(\cdot)$ should be a function between the two domains:

$$F : \mathbb{R}^P \to \{+, -\}^D$$

In reality, we will have a different function, $f(\cdot)$, which does not return the real label, but rather an estimation, $\hat{y}$. Finding and improving this $f(\cdot)$ is usually done through a *loss function*, a function that is minimized when $\mathbf{y} = \hat{y}$, such as the Euclidean distance between the two, known in this context as the ℓ^2-norm:

$$L(\mathbf{y}, \hat{y}) = \|\mathbf{y} - \hat{y}\|^2$$

The classifier used in this research is a convolutional neural network, and $f(\cdot)$ is approximated through back-propagation, explained in more detail in Appendix A.

3.2. Loss Function for Multi-Label Classification

In a regular classification setting, we differentiate between different classes and each entry in the dataset is assigned to a single class. In the case of defect detection in sewers, this leads to a problem: several defects often co-occur. Infiltration, for example, almost always has a cause that is defined as a separate defect, such as a crack. This co-occurrence can be a result of the definitions used in the EN 13508-2 guidelines [18], or they might be an effect of cascading failures [20]. In our dataset there are 17,662 out of 2,202,582 images (0.802%) that contain defects, totalling 21,139 different defects, but 6,494 of these (30.7%) co-occur with another defect in the same image. When considering entire pipes, 2,512 out of 3,350 pipes (75.0%) contain defects, 6,918 defects types in different pipes are found in total, and 6,171 (89.2%) of these defect types are found co-occurring with other defect types in the same pipe.

We distinguish *multi-label* classification (multiple classes per object), as opposed to *multi-class* classification, as that might also refer to a non-binary classification case, i.e. an object has a single class, but there are more than two classes.

As a result of this, we have decided to label the images with a boolean vector, each of twelve boolean values, representing the presence or absence of a particular defect. This means that as most images do not contain a defect at all, these will have a vector of all negatives.

Obviously, not all misclassifications should be treated equally. If we correctly classify the presence or absence of eleven defects, but misclassify the presence or absence of the last defect, this is less severe than misclassifying multiple defects.

For each of the twelve defects we calculate an individual loss function, namely the cross-entropy [21] between the actual value for a defect, $\mathbf{y}_c$ (0 for absence, 1 for presence in the image), and the predicted value outputted by the network for that defect, $\hat{y}_c$ (a real value in the interval $[0, 1]$):

$$L(\mathbf{y}, \hat{y}) = -\sum_{c=1}^{12} \mathbf{y}_c \log \hat{y}_c$$

As written, only false negatives contribute to the cross-entropy loss, as $\mathbf{y}_c$ is zero for false positives. This means that we penalize the classifier for not detecting a defect, but not for detecting a defect where there is none. To make sure that the network does not simply output 1 for all defects, $\hat{y}$ is commonly normalized so that $\sum_c \hat{y}_c = 1$, which is called *soft-max* normalization [22]. Alternatively, it is also possible to account for false positives by adding contributions both for $\mathbf{y}$ and its complement:

$$L(\mathbf{y}, \hat{y}) = -\sum_{c=1}^{12} \mathbf{y}_c \log \hat{y}_c + (1 - \mathbf{y}_c) \log(1 - \hat{y}_c)$$

This is what we will use, as normalizing $\hat{y}$ does not make much sense when we expect defects to co-occur.

3.3. Class Imbalance & Oversampling

Our dataset consists of 3,350 pipes with a total of 2,202,582 images. Of the defects present in these images, we've selected the twelve most common to classify. Table 1 gives an impression of how common these defects are in the dataset.

Table 1: Defect types and occurrences

Defect Type \ **Total:**	**Pipes** 3,350	**Images** 2,202,582
Fissure	586	1,442
Surface Damage	1,242	2,507
Intruding Connection	375	1,004
Defective Connection	506	838
Intruding Sealing Material	74	173
Displaced Joint	1,509	4,988
Porous Pipe	117	187
Roots	273	629
Attached Deposits	183	338
Settled Deposits	164	219
Ingress of Soil	536	1,249
Infiltration	1,353	7,565

While every pipe contains at least one defect of some type in one of its images, only 17,663 images[2], roughly 0.8% of all images contain a defect. It should also be noted that the percentage of pipes that contain a specific defect

[2]This number is not equal to the sum of the numbers in the rightmost column of table 1, because defects often co-occur in the same image, and some images are counted multiple times if we sum the column.

is not the same as the percentage of images that contain that defect, as a pipe is said to contain a defect if at least one image from that pipe contains the defect.

The extreme class imbalance of images with and without defects in our dataset means that if we train a classifier without accounting for the imbalance, it will err on the side of false negatives, as these are simply more likely. If we make some number of misclassifications, we expect these to be distributed the same as the prior probabilities of the classes, simply put: our set has about 1% defects and 99% non-defects, so of the errors the classifier will make 1% will be a false positive and 99% will be a false negative. It can be easily seen that a false negative is more costly than a false positive (the former costs labor hours, the latter might pose a health hazard or incur additional costs e.g. through property damage or disruption of traffic). This has some important implications for the quality assessment of a classifier as well, which will be discussed in section 4.

Two choices have been made to adjust the classifier and make it more able to handle this imbalance: oversampling and a class-weighted loss function.

Oversampling is done from a more practical perspective, to train the CNN we have to load a *batch* of input images into memory and the backpropagation step happens for all images in the batch at the end of each batch. Because of computational limitations, we found that our experimental setup could handle batches of about 50 images at a time. This means that it is extremely likely for a batch not to contain any defects at all. The gradient of such a batch can not be used to accurately estimate the gradient of the entire training set [23]. To remedy this, each defect in the dataset is added not once but five times to the training set, to increase the odds of having at least one defect in every batch.

As oversampling the defects by a factor five brings the imbalance up from 0.8% to about 4% of the images in the training set containing a defect, we should still be wary of training a network with such an imbalance. To shift the error that the network makes more towards false positives than false negatives, we weight the cross-entropy loss function from equation (4.1) as follows:

$$L(\mathbf{y}, \hat{y}) = -\sum_{c=1}^{12} W \mathbf{y}_c \log \hat{y}_c + (1 - \mathbf{y}_c) \log(1 - \hat{y}_c)$$

Where W is a weight that represents how much more important false negatives are compared to false positives. If we consider a false negative to be 100 times more costly than a false positive, we should set W to 100.

3.4. Overfitting

Overfitting is what happens when the classifier is trained on the training set so well that it loses generalisability on other datasets. All data that has been sampled from real world measurements (such as photographs in our case) is expected to have some amount of noise in it. This means that any model or classifier that can describe this data — with the included noise — to a 100% accuracy has incorporated this noise in its model. The model's performance on a different dataset (with different noise, perhaps from different measuring instruments) will be worse than a classifier that learnt to model the data, but not the included noise [9].

This danger is exacerbated when the noise in the training set is systemic, for example through a sampling bias, as this becomes another pattern the classifier might detect and learn, when it is in fact noise that will not be present in future datasets that require classification. Neural networks are also more prone to overfitting than a lot of other classifiers, because of the large number of parameters that are subject to change when learning from the training data [24]. To prevent overfitting, we employ two methods: the use of a validation set and dropout.

The use of a validation set is the more general approach of the two. Instead of training on all the data in the training set, we keep a subset of the training set apart, which is not used to train on. Every so often during the training phase, we calculate the loss function on this validation set. At some point the classifier will start overfitting, meaning the loss function on the training set will keep decreasing, but the loss function on the validation will either stagnate or start increasing. At this point we choose to stop the training and take the classifier as trained up to that point as the final classifier. We have chosen to calculate the loss on the validation set after every epoch[3] and stop early if the loss on the validation increased significantly, or hasn't decreased for a few epochs in a row.

Dropout is a way to prevent overfitting specifically for neural networks. The idea is that to prevent a network that is too specifically catered to the input data, we should assure some stability with regards to small changes in the network structure. If the correct classification of a sample depends on a single specific path through the network, that would not be stable, as only one of the neurons in that path has to change some weights for the classification result to change. To force the network to not rely on a single path through the network, we randomly disable neurons in the dense layers during the training step, setting their output to zero. This forces the network to create a path to the correct classification result with the still enabled neurons. Since a different set of neurons will be deactivated for every batch of data, this increases the overall stability of the network by ensuring the correct result can be reached through different paths. As the dropout is disabled after training is complete, all the paths that lead to correct classification will work together, and small changes in any one of these paths should not change the end result.

[3] The number of batches it takes until the entire training set has been through the backpropagation step once.

4. Result Validation

Perhaps as important as obtaining a classifier that is well-trained on a domain, is knowing the classifier's boundaries. If a classifier predicts some defect, it is important to know how trustworthy this result is to put it into context before acting on it. To properly assess the quality of our classifier, we examine several different performance metrics to discuss their value, and we introduce a specific type of cross validation to unbias the results further.

4.1. Performance Metrics

As noted in section 1.3, a lot of previous work completely disregarded the class imbalance when training and testing their classifier, and instead opt for a more manageable 50% split. This approach has a major issue: the test results are not representative of a real-world scenario, and only indicative of the quality of the classifier in a general case, not for this specific classification scenario. Instead, we require the test set to have a realistic ratio of images with defects to images without defects, as this means our results should translate directly to the results we would obtain when applying our classifier on newly obtained data.

Commonly, the classification *accuracy* is used, which is the ratio of correctly classified samples out of all samples. It can be easily seen that this is not a very useful measure with our extremely imbalanced dataset. We could create a classifier that classifies every image as not having any defects, and it would have an accuracy of over 99%, giving the impression that it is a good classifier, when it is not.

Our classifier outputs real-valued predictions in the interval $[0, 1]$, while the actual labels are always either 0 or 1. This means we have some freedom in choosing a threshold θ, that separates a predicted 0 from a predicted 1. We write this as:

$$\hat{y}_c = \begin{cases} 0 & \text{if } f_c(x) < \theta \\ 1 & \text{if } f_c(x) \geq \theta \end{cases} \tag{4.1}$$

where $\hat{y}_c$ is the predicted label for defect c and $f_c(x)$ is the classifier's real-valued output for image x and defect c. Setting a specific threshold θ for the classifier's output results in each image being a true positive ($y_c = \hat{y}_c = 1$), a true negative ($y_c = \hat{y}_c = 0$), a false positive ($y_c = 0; \hat{y}_c = 1$), or a false negative ($y_c = 1; \hat{y} = 0$). TP, TN, FP, and FN are positive integer numbers, denoting the number of images ending up in that category.

We define the false positive rate (FPR), true negative rate (TPR, sometimes *specificity*), false negative rate (FNR), and true positive rate (TPR, sometimes *recall* or *sensitivity*) by dividing a quadrant in the confusion matrix with the row total:

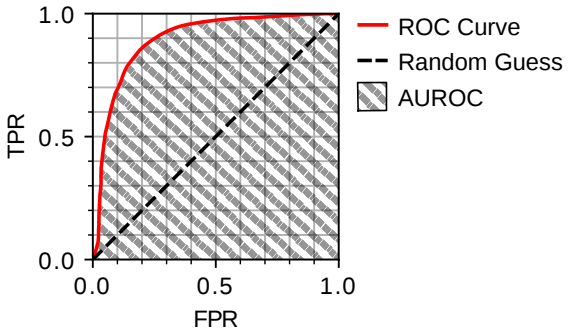

Figure 2: Example of a Receiver-Operator Characteristic (ROC) curve. Every point on the red line corresponds to a possible threshold θ, that defines the TPR and FPR. The shaded area shows the area under the ROC curve (AUROC), and the diagonal line is the ROC curve one would obtain by randomly guessing the label for each instance.

$$FPR = \frac{FP}{FP + TN} = 1 - TNR \tag{4.2}$$

$$TNR = \frac{TN}{FP + TN} = 1 - FPR \tag{4.3}$$

$$FNR = \frac{FN}{FN + TP} = 1 - TPR \tag{4.4}$$

$$TPR = \frac{TP}{FN + TP} = 1 - FNR \tag{4.5}$$

These rates hold some significance as they are equivalent to the chances that a classifier will make a certain error. For example, if FNR = 0.2, this means in practice that 20% of defects are missed by the classifier.

We have some freedom on how to choose θ, which gives us a way to balance the false negatives and false positives. Any increase in θ leads to an increase in FNR and a decrease in FPR (and vice versa). If we decide that a false negative is 20 times as costly as a false positive, we could set θ at an optimum such that $\frac{FPR}{FNR} = 20$. The trade-off leads to the construction of the *receiver operator characteristic* (ROC) [9], as shown in figure 2. Every value of θ corresponds to a point in the ROC curve, and it is common to use the area under the ROC curve (AUROC) as a measure of classifier performance that is independent of the threshold θ.

The reason accuracy is not a very useful measure for us, is that the FPR (and the equivalent TNR) are dominated by the TN term in this unbalanced classification scenario, which isn't very interesting, as it is easy to achieve a very high TN by classifying everything as negative. As the FPR is one of the dimensions of the ROC, this leads to the ROC only having limited usefulness. Instead of the FPR, we use *precision*, defined as:

$$Pr = \frac{TP}{TP + FP} \tag{4.6}$$

Both the FPR and the precision are measures of the number of type I errors a classifier makes. The difference is that the FPR is in comparison to the total images without

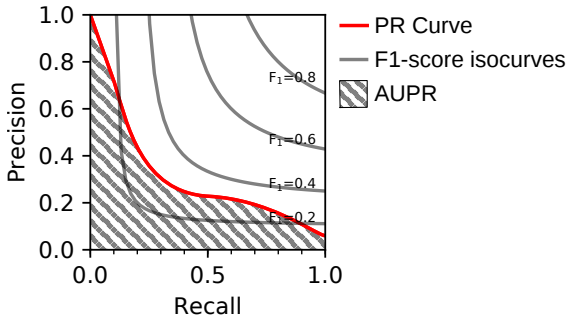

Figure 3: Example of a Precision-Recall (PR) curve. Every point on the red line corresponds to a possible threshold θ, that defines the precision and recall. The shaded area shows the area under the PR curve (AUPR), and the grey lines are curves with a constant F_1-score.

defects (*"How many of the images without defects did we label as defects?"*), whereas the precision is in comparison to the images that were labeled as defects (*"How many of the images labeled as defects are false alarms?"*). The former is heavily skewed towards the appearance of a good performance because of the prevalence of images without defects, but the latter does not have this issue.

If we now combine precision with recall (TPR), we can construct a curve analogous to the ROC curve, called the Precision Recall curve, or PR curve, as shown in figure 3. The area under the PR curve is more meaningful than the AUROC and also independent of θ. An example PR curve is shown in figure 3.

While the areas under these curves provide a metric independent of θ, they still take all levels of recall into account, whereas we are likely interested only in higher levels of recall. To this end, we introduce two more metrics: "specificity at sensitivity" and "precision at recall". Both these metrics do not require us to manually choose a value for θ, instead they dictate θ to be chosen such that recall is at a certain level, and report the specificity (TNR) or precision at this θ. This is the same as taking a point on the ROC and PR curves that corresponds to a particular value on the recall axis, and reading the point it corresponds to on the complimentary axis.

The authors feel that especially the specificity at sensitivity and precision at recall metrics are useful to put the results into real-world context: for public health reasons we might be restricted a minimum value for recall (as a lower value would allow too many defects to slip by unnoticed and increase the risk), and we simply want to know how efficient the system is *at least* at that level. For both of these metrics, we evaluate at the recall/sensitivity levels $[0.90; 0.95; 0.99]$, as we are mostly interested in high recall. An overview of the performance metric we are using is given in table 2.

4.2. Aggregating performance on pipe level

The previous section outlined performance metrics for classifying single images, but it is not an uncommon scenario to classify entire pipes as a whole for a certain defect. To achieve this, we aggregate the real and predicted labels on the images with some *aggregation rule*, and calculate the same metrics from table 2 on the aggregated labels.

Table 2: An overview of the performance metrics used

Metric	**Description**
AUROC	Area under the ROC curve
AUPR	Area under the PR curve
Specificity at Sensitivity	Percentage of non-defects classified as defects when we require a minimum percentage of defects to be detected
Precision at Recall	Percentage of false alarms among detections when we require a minimum percentage of defects to be detected

An obvious choice for an aggregation rule is the maximum: This would be analogous to determining whether any of the images are labeled as a defect, compared to whether any of the images actually contains a defect. Importantly, this aggregation rule does not depend on the size of the pipe, like the average value would. A downside to this rule is that we might actually be detecting a defect in an image where there is none and missing a defect that is in another image, but we still count this as a true positive, because we only care to know if we found the defect in the correct pipe.

Maximum aggregation performance metrics on pipe level will be reported alongside performance metrics for single images.

4.3. Leave-two-inspections-out Cross-validation

To accurately assess performance of a classifier on a dataset, we might use k-fold cross-validation [9]. The dataset is divided into *folds*, non-overlapping subsets that when put together reform the entire dataset. Instead splitting the data into a single training set and test set, we take one fold to be the testing set and train the classifier on the remaining $k-1$ folds. This process is repeated k times, until each fold has been used as the test set one time.

The folds are often divided either randomly or *stratified*, meaning that the classes are divided as equally as possible among the folds. Because of how our dataset was sampled, we expect some overlap in construction material and age within an inspection, which is often performed within a single geographical neighborhood. In this case a random or stratified split might lead to *data leakage*, information from outside the training set being implicitly part of the training set: two points in a single pipe might exhibit the same defect, as they are subject to the same condition, are of the same build material and year. But these factors also mean they might appear very similar. As a result, it might be so that our classifier is simply classifying the appearance of a pipe, and not the defects themselves. If we then use random or stratified splitting, we might overestimate the actual performance.

Instead, we introduce "leave-two-inspections-out" cross-validation. This is inspired by "leave-one-patient-out" cross-validation, sometimes used for medical data. Since the data is already categorized into 30 inspections, we use these same inspections as folds for cross-validation. We take 28 folds as the training set, 1 fold as the testing set, and 1 fold as a validation set, to prevent overfitting on the training set. These folds are rotated 30 times, until each fold has been used as the training set once, and we have a prediction for each image. This provides a more realistic scenario, where the classifier would be used to predict the presence of defects in a pipe it has never seen before.

A possible downside of this method is that for any given fold, we likely do not have every defect present in both the test and validation sets. Since there is no defect that appears in fewer than three inspections, at the very least every training set will contain every defect.

4.3.1. Averaging performance metrics across folds

While the leave-two-inspections-out cross-validation should prevent data leakage and give a more accurate performance indication, it also means we are training 30 different CNNs, and combining the performance results of these into a single metric is not straightforward. There is no guarantee that the trained networks have at all similar weights at any given point or that the outputs of the networks is similar. As noted in the previous section, the distribution of defects among folds can also be skewed, with some inspections containing a lot more or fewer defects than others.

As such, it does not make sense to average the metrics as calculated on the folds. We could set a single threshold θ for each defect and fold, but since the outputs of the different networks could behave very differently, this is also not desirable.

As we have argued that it is not unlikely for defect detection systems to be tuned to achieve some minimum sensitivity/recall, we have decided to construct the ROC and PR curves for each fold and each defect individually, and combine the curves for different folds by equating the sensitivity/recall axis, and combining the values on the complimentary axis. For the ROC curve, this is called 'horizontal averaging' [25], for the PR curve, we might call it 'vertical averaging', as the recall axis is the horizontal axis, but there is no previous use in literature that we know of.

It should also be noted that the averages for the specificity and precision are not calculated identically. Both are calculated with a weighted average, but the results for specificity in each fold should be weighted such that the combined result represents the specificity of the entire set, and the results for precision should be weighted such that the combined result represents the precision of the entire set. In practice, this means that the results are weighted with the relevant denominator from equation (4.3) or (4.6). As a result, a fold with no occurrences of a particular defect will have no impact on the combined specificity of that defect, and a fold with no *predicted* occurrences of a particular defect will have no impact on the combined precision of that defect.

5. Implementation Details

We have implemented two different CNNs, one designed by the authors for this task, and one reimplementation of the network used in [15] (with the first layer adapted to our image sizes). This is of course not an entirely fair comparison, as we failed to reproduce their entire pipeline, but instead only replicated the network itself, but it does put our performance into context.

The network topologies are shown in figures 4 and 5. The network topology of our proposed CNN was designed through informal experimentation with different layer sizes, filter sizes, and numbers of layers on a smaller subset of the dataset.

The CNNs were built and run with a TensorFlow (version 1.8.0) [26] and Python (version 3.4.8) [27], running on a Linux system with sixteen NVIDIA Tesla K80 GPUs and CUDA (version 9.2.148). Each network was trained using a single GPU, with several networks (one for each validation fold) being trained simultaneously on multiple GPUs. Training a single network took on average roughly five hours (per fold). Testing the different networks with each different testing fold took on average roughly 1 hour (for all folds).

Each of the networks was trained with a batch size of 50. After every 500 batches, the performance on the validation fold was assessed. The network stopped training when the AUROC on the validation fold had not increased for 25 consecutive assessments, or when the AUROC on the validation fold decreased by more than 1%, with a minimum of 1,000 batches processed.

6. Results

In this section we present the results achieved by our proposed CNN, as well as our reimplementation of the CNN proposed by [15], on the performance metrics outlined in section 4. We also provide an interpretation for these results and their practical meaning for operators.

Figures B.1, B.2, B.3, and B.4 show the ROC and PR curves for our proposed CNN for classification in images and entire pipes. For ease of reading, these figures have been moved to Appendix B.

Tables 3, 4, 5, and 6 show the specificity (TNR) and precision at recall (TPR) values of 0.90, 0.95, and 0.99, for our proposed CNN and our reimplementation of the CNN proposed by [15].

7. Discussion

Looking at tables 3, 4, 5, and 6, we see that in each of the shown scenarios, our proposed CNN either outperforms [15], or it does not perform significantly worse. Out

Table 3: TNR at specific TPR values when classifying single images. Numbers displayed in bold indicate that performance is significantly better than the performance achieved by the other network, as determined by a paired sample t-test at significance level $\alpha = 0.05$.

	TNR at 0.90 TPR		TNR at 0.95 TPR		TNR at 0.99 TPR	
Defect	This work	Kumar et al.	This work	Kumar et al.	This work	Kumar et al.
Fissure	**0.754**	0.375	**0.683**	0.290	**0.550**	0.208
Surface Damage	**0.702**	0.240	**0.548**	0.107	**0.291**	0.045
Intruding Connection	**0.916**	0.448	**0.809**	0.392	**0.741**	0.370
Defective Connection	**0.901**	0.553	**0.811**	0.460	**0.703**	0.372
Intruding Sealing Material	**0.780**	0.061	**0.731**	0.057	**0.706**	0.057
Displaced Joint	**0.691**	0.441	**0.532**	0.306	**0.262**	0.145
Porous Pipe	0.349	0.207	**0.322**	0.179	0.307	0.171
Roots	**0.728**	0.209	**0.633**	0.166	**0.561**	0.159
Attached Deposits	**0.388**	0.142	**0.313**	0.116	**0.281**	0.115
Settled Deposits	**0.510**	0.114	**0.459**	0.102	**0.442**	0.097
Ingress of Soil	**0.762**	0.322	**0.670**	0.237	**0.532**	0.180
Infiltration	**0.622**	0.220	**0.486**	0.160	**0.253**	0.092

Table 4: Precision at specific recall values when classifying single images. Numbers displayed in bold indicate that performance is significantly better than the performance achieved by the other network, as determined by a paired sample t-test at significance level $\alpha = 0.05$.

	Precision at 0.90 Recall		Precision at 0.95 Recall		Precision at 0.99 Recall	
Defect	This work	Kumar et al.	This work	Kumar et al.	This work	Kumar et al.
Fissure	**0.036**	0.006	**0.019**	0.004	**0.005**	0.003
Surface Damage	**0.011**	0.003	**0.005**	0.002	**0.003**	0.002
Intruding Connection	**0.071**	0.007	**0.011**	0.005	**0.006**	0.004
Defective Connection	**0.014**	0.002	**0.008**	0.002	**0.004**	0.001
Intruding Sealing Material	0.002	0.000	0.002	0.000	0.001	0.000
Displaced Joint	**0.015**	0.006	**0.010**	0.005	**0.005**	0.004
Porous Pipe	0.000	0.000	0.000	0.000	0.000	0.000
Roots	**0.003**	0.001	**0.002**	0.001	**0.002**	0.001
Attached Deposits	**0.001**	0.000	**0.001**	0.000	**0.001**	0.001
Settled Deposits	0.001	0.000	0.000	0.000	0.000	0.000
Ingress of Soil	**0.008**	0.002	**0.005**	0.002	**0.003**	0.002
Infiltration	**0.024**	0.012	**0.017**	0.012	**0.013**	0.012

Table 5: TNR at specific TPR values when classifying entire pipes. Numbers displayed in bold indicate that performance is significantly better than the performance achieved by the other network, as determined by a paired sample t-test at significance level $\alpha = 0.05$.

	TNR at 0.90 TPR		TNR at 0.95 TPR		TNR at 0.99 TPR	
Defect	This work	Kumar et al.	This work	Kumar et al.	This work	Kumar et al.
Fissure	**0.428**	0.357	**0.365**	0.279	0.306	0.261
Surface Damage	0.250	0.226	0.193	0.155	0.128	0.107
Intruding Connection	**0.414**	0.250	**0.371**	0.224	**0.354**	0.220
Defective Connection	**0.230**	0.186	0.186	0.147	0.172	0.132
Intruding Sealing Material	0.411	0.406	0.411	0.406	0.411	0.406
Displaced Joint	0.294	0.268	**0.219**	0.169	**0.123**	0.085
Porous Pipe	0.363	0.399	0.346	0.377	0.344	0.372
Roots	0.403	0.338	0.338	0.290	0.317	0.267
Attached Deposits	0.518	0.481	0.496	0.454	0.486	0.444
Settled Deposits	**0.386**	0.305	**0.364**	0.282	**0.363**	0.282
Ingress of Soil	0.285	0.315	0.237	0.286	0.209	0.245
Infiltration	0.284	0.298	0.218	0.207	0.141	0.133

Table 6: Precision at specific recall values when classifying entire pipes. Numbers displayed in bold indicate that performance is significantly better than the performance achieved by the other network, as determined by a paired sample t-test at significance level $\alpha = 0.05$.

	Precision at 0.90 Recall		Precision at 0.95 Recall		Precision at 0.99 Recall	
Defect	This work	Kumar et al.	This work	Kumar et al.	This work	Kumar et al.
Fissure	**0.414**	0.379	**0.393**	0.364	0.363	0.358
Surface Damage	0.582	0.589	0.573	0.570	0.557	0.552
Intruding Connection	**0.369**	0.333	**0.360**	0.324	**0.348**	0.315
Defective Connection	**0.294**	0.276	**0.291**	0.273	**0.285**	0.272
Intruding Sealing Material	0.068	0.062	0.068	0.062	0.068	0.062
Displaced Joint	0.600	0.602	0.585	0.582	0.565	0.561
Porous Pipe	0.130	0.137	0.129	0.136	0.129	0.135
Roots	**0.208**	0.189	0.185	0.184	0.176	0.175
Attached Deposits	0.190	0.196	0.185	0.182	0.183	0.179
Settled Deposits	0.125	0.118	0.117	0.108	0.117	0.109
Ingress of Soil	0.363	0.364	0.348	0.358	0.339	0.335
Infiltration	0.584	0.602	0.579	0.584	0.560	0.563

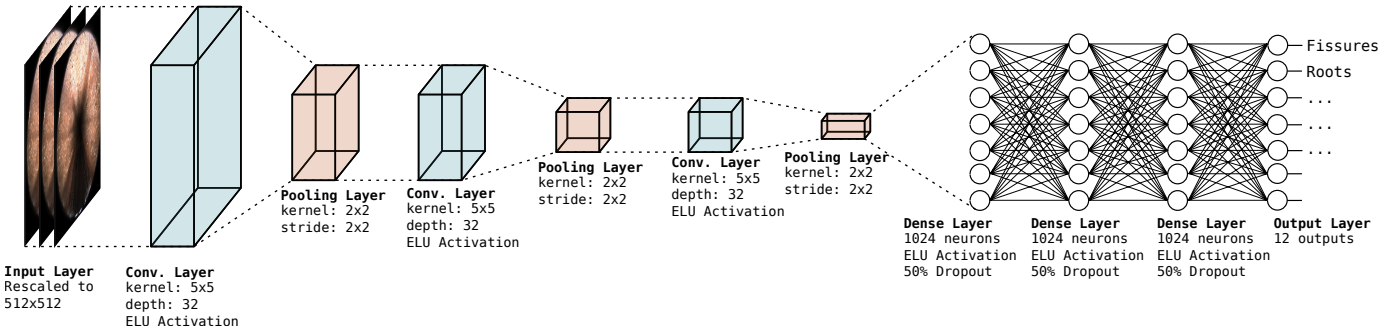

Figure 4: Network structure of our proposed CNN.

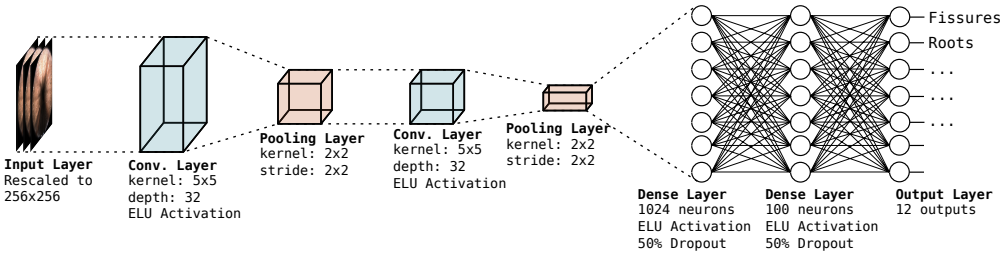

Figure 5: Network structure of the CNN proposed in Kumar et al. [15].

of 144 scenarios, our proposed network wins significantly 81 times. Additionally, it wins 44 times, but not by a significant margin, and loses 19 times, but never significantly.

When we take a closer look at the ROC and PR curves in figures B.1, B.2, B.3, and B.4 in Appendix B, there are a few observations to be made.

The ROC curves in figure B.1 generally look quite good, with the exception of those for porous pipes, attached deposits, and settled deposits. The class imbalance is quite important here: the AUROC does not take into account that a false positive and false negative are not comparable in this context. As noted earlier, we are mostly interested in the scenario with high true positive rates, as these are a requirement for any kind of automated sewer inspection, which means the top portion of each plot is more important than the bottom portion. It must also be noted that while both axes go from 0 to 1, the horizontal axis concerns many more images than the vertical axis does, because of the class imbalance. One interesting feature of these curves, is that they seem to have a 'plateau' near the top. This indicates that a specific threshold exists where it is no longer advantageous to further increase the threshold, as this will only increase the false positive rate, but not the true positive rate. The false positive rate at this interval (which is approximately equal to 1 minus the specificity at 99% recall, as noted in table 3) can be regarded as the best specificity we can achieve for a certain defect.

The PR curves in figure B.2 paint a different picture: the PR curves are mostly below an F_1-score of 0.2, seeming very unimpressive. Similar to the ROC curves, we are mostly interested in high recall, i.e. the rightmost portion of each plot. In this case, the precision seems to be quite low, but unlike the specificity, the precision axis *is* scaled with the prior probability of the defects. We will go into more detail on how to interpret these precision scores in the next section, but it should be noted that a small precision is expected when we have small prior probabilities.

When we observe the ROC and PR curves for classification of pipes, they paint a rather different image. The ROC curves in figure B.1 do not look very good, but it should be noted that the 'plateaus' are again present in a lot of the curves, which again indicates that there are some pipes which we are confidently sure *do not* contain defects. Rather interestingly, among the better ROC curves are those that underperformed on single-image classification: porous pipes, attached deposits, and settled deposits. This might indicate that some labels were missing in our test set: if a pipe has these defects at multiple locations but only a few locations were marked in the inspection report, we would overestimate the false positives our classifier finds in single-image classification, but we can be more sure when deciding whether a pipe has or does not have this defect.

The PR curves for the classification of pipes looks a lot better than that of single images, this is because the class imbalance is much less present on pipe-level. Still the worst results are obtained for classes that have a low prior on pipe-level (intruding sealing material, porous pipe, roots, attached deposits, settled deposits), as expected for the precision.

7.1. Result Interpretation

To put our findings into context, we will take a closer look at their impact on the day-to-day operation of sewer inspections aided by our automated system. When looking at our results superficially, they are easily misinterpreted. It is important to keep in mind that we are dealing with a very imbalanced dataset, which makes the precision the more interesting of these metrics (as described in section 4). Let's consider the class *Fissure* in more detail. From table 1 we can tell that approximately 0.065% of the images (1,442 out of 2,202,582 images in total) contain a fissure, which makes for a very unbalanced target. For fissures at 0.90 sensitivity we achieve a specificity of 0.754 and a precision of 0.036 (see tables 3 and 4, top left cell).

If we assume that fissures are randomly distributed, an unsupported operator would have to inspect 90% of all images ($0.9 \times 2{,}202{,}582 = 1{,}982{,}324$ images) to find 90% of the fissures. Our proposed classifier detects 90% of all images with fissures with a specificity (TNR) of 75.4%. This specificity of 75.4% indicates that, of all the images that do *not* contain fissures, we identify 75.4% as such, and the remaining 24.6% are suspected of containing fissures, meaning they still have to be inspected by an operator.

The precision indicates that 3.6% of all detections are true positives, while the remaining 96.4% are false alarms. To detect 90% of all fissures, an operator would have to inspect all detections the system made: $0.246 \times (2{,}202{,}582 - 1{,}442) + 0.9 \times 1{,}442 = 542{,}778$ images. In comparison to the situation without a classification system, this equates to a reduction of 72.6%. In an ideal situation, this means that the time an operator spends on inspecting fissures is reduced by almost a factor 4. Table 7 lists these reduction numbers (derived from tables 3 and 4) for all defect types considered. The reduction of 72.6% for Fissure appears as the top-left cell. The highest reduction (at 0.90 recall) is attained for Intruding Connection, with a 90.7% reduction (a factor 10). Not surprisingly, this defect types scores well both in the ROC as in the PR plots. It ranks 6th in terms of frequency of defect type, with 1,004 observed cases.

We can perform the same calculations with the results from classification on *entire pipes*, but the interpretation is a little less clear, as we cannot assume different pipes take the same amount of time for review; especially pipes with a lot of defects will be more labor-intensive to inspect. From table 1 we can tell that approximately 17.5% of pipes contain fissures (586 out of 3,350 pipes). Let us for this case assume 99% of all pipes containing fissures need to be detected, this means $0.99 \times 3{,}350 = 3{,}317$ pipes have to be inspected for fissures. Our classifier achieves 99% recall (TPR) with a specificity (TNR) of 30.6% (table 5) and a precision of 36.6% (table 6). By the same calculations as above, this means we now have to inspect $0.694 \times (3{,}350 - 586) + 0.99 \times 586 = 2499$ pipes. This is a reduction of 24.7%. Table 8 shows similar reductions for all the defect types, for pipes.

In both these tables, we see that intruding and defective connections are best classified by our CNN and have the largest reduction rate in images or pipes that still require human review, while porous pipes are the more difficult to classify and these have the lowest reduction rates.

Table 7: Reduction of *images* that need to be reviewed by a human after inspection with our classifier, expressed in percentage compared to images that would need to be inspected without our classifier.

	Recall/TPR		
Defect Type	**0.90**	**0.95**	**0.99**
Fissure	72.6%	66.6%	54.5%
Surface Damage	66.8%	52.4%	28.3%
Intruding Connection	90.7%	79.8%	73.8%
Defective Connection	89.0%	80.1%	70.0%
Intruding Sealing Material	75.5%	71.7%	70.3%
Displaced Joint	65.5%	50.7%	25.4%
Porous Pipe	27.7%	28.7%	30.0%
Roots	69.8%	61.4%	55.6%
Attached Deposits	32.0%	27.7%	27.4%
Settled Deposits	45.5%	43.1%	43.7%
Ingress of Soil	73.5%	65.3%	52.7%
Infiltration	57.8%	45.8%	24.4%

Table 8: Reduction of *pipes* that need to be reviewed by a human after inspection with our classifier, expressed in percentage compared to pipes that would need to be inspected without our classifier.

	Recall/TPR		
Defect Type	**0.90**	**0.95**	**0.99**
Fissure	30.1%	27.4%	24.7%
Surface Damage	10.5%	9.5%	7.5%
Intruding Connection	31.0%	30.0%	30.9%
Defective Connection	12.2%	12.1%	13.9%
Intruding Sealing Material	33.8%	37.2%	39.7%
Displaced Joint	11.9%	9.8%	6.3%
Porous Pipe	28.3%	30.1%	32.6%
Roots	30.9%	27.8%	28.5%
Attached Deposits	43.9%	44.3%	45.4%
Settled Deposits	30.3%	31.5%	33.9%
Ingress of Soil	17.2%	16.5%	16.9%
Infiltration	12.2%	10.5%	7.9%

Realistically, the defects are not randomly distributed throughout the image set and operators would not inspect single images, but rather a sequence of images with a clear spatial relationship (a 5 cm shift). This means that the reduction by a factor of 4 is almost certainly an overestimation. On the other hand, we know defects can often co-occur [20] and this estimation was only for fissures, which has one of the higher prior probabilities of the defects we consider. For defects with a lower prior probability, there is a larger potential for improvement.

It should also be noted that with the reported false negative probability of about 25% [2] in the labels of our data set, the actual precision and specificity are likely higher than we report. For any given defect, there is approximately a 1 in 4 change that the operator missed it and it was labeled in our dataset as not being a defect (whereas

Table 9: TNR and Precision at specific TPR/Recall values for binary classification on either single images or entire pipes.

Metric and Classification type	Recall/TPR 0.90	0.95	0.99
TNR for Images	0.649	0.452	0.180
TNR for Pipes	0.372	0.284	0.113
Precision for Images	0.021	0.014	0.009
Precision for Pipes	0.717	0.703	0.668

the probability of a false positive was estimated "in the order of a few percent"). The 1,442 images that are labeled as fissures, are possible only 75% of all images labeled as fissures, meaning there would be approximately 480 images among the images not labeled as fissures.

7.2. Combining Defect Outputs

Because of the co-occurrence of defects, it can be interesting to combine the classifier outputs for different defects into a single decision: *"Does this image/pipe need further (human) review?"*

As discussed in section 3, in our dataset 30.7% of defects in images co-occur with other defects in the same image and 89.2% of defects in pipes co-occur with other defects in the same pipe. To treat the problem as a binary classification problem, we simply take the maximum value of the true label over the classes (a 1 if at least one defect is present, a 0 if no defects are present), and the average of the predicted labels over the classes (a real-valued number between 0 and 1). This gives us the curves as shown in figure 6.

For classification on images, reducing this problem to a binary classification case does not improve things much. The overall result is approximately equal to the average of the classification results on individual classes. This is not unexpected, as the co-occurrence of defects in individual images is rare.

For classification on pipe level though, the results are more interesting than a simple averaging. The PR curve is strictly better than the PR curves of individual defects. The ROC curve at high TPR is slightly worse than some individual defects, but the overall AUROC is higher.

Table 9 shows the TNR and Precision at specific Recall/TPR values, for comparison with the multi-label classification results in tables 3, 4, 5, and 6. Using these values we can again calculate the reduction in images or pipes that require review to achieve a certain TPR/Recall, as shown in table 10.

The reductions on pipe level are quite low, this is because the transition to a binary classification scenario results in the class imbalance disappearing on pipe-level: 75.0% of pipes contain *at least one* defect, and fall into the positive class. This means a high precision is required, and while precision had increased by combining the defect types, the reduction has decreased.

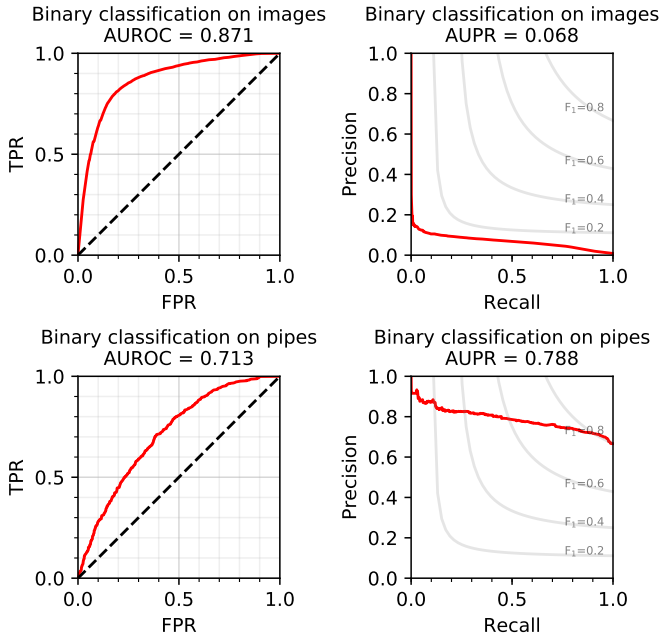

Figure 6: ROC and PR curves obtained when treating the problem as a binary classification problem, for image level or pipe level.

Table 10: Reduction of images or pipes that need to be inspected with our combined binary classifier, expressed in percentage compared to pipes that would need to be inspected without our combined binary classifier.

Classification Type	Recall/TPR 0.90	0.95	0.99
Images	60.5%	42.0%	17.0%
Pipes	7.6%	6.2%	2.6%

8. Conclusion

In this work we have approached the task of automated defect detection in sewer image sets as a supervised classification task. The focus has been on the validation methodology used to interpret the results achieved by a classifier. While we feel that there is a lot of potential for future improvement of classifiers trained for this task, with the data and computational resources available, the proposed convolutional neural network performed reasonably well.

While our proposed classifier does not perform well enough for fully autonomous classification, it can be used to significantly reduce the amount of images that require human review by eliminating images which are highly unlikely to contain defects according to the classifier. We estimate the amount of images that require human review can be reduced by 60.5%, given that detecting 90% of all defects is sufficient.

We compared the results of our proposed classifier to that of Kumar et al., [15] and our proposed classifier outperforms their proposed classifier, but we did not implement their classification pipeline beyond the network structure, such as for example, the oversampling outlined in

their work. Our dataset also differs significantly from theirs, as noted in section 2, no human inspector has changed the camera settings during the inspection, as is common with other CCTV inspection datasets.

The primary topic of this work was the validation methodology. We have discussed our reasons for choosing the "specificity at sensitivity" and "precision at recall" metrics for this specific task in section 4: these give us easily interpretable measures of the possible improved efficiency at realistic scenarios. We have also explained why "leave-two-inspections-out cross validation" is an appropriate way to prevent data leakage, and applied this technique in our experiments. These methods provide us with less biased and more easily interpretable results.

8.1. Future Work

Not all information in the inspection reports was used to its full potential and the authors feel that using the information on where in an image a defect is visible (with a classifier capable of processing this information of course) could lead to further performance improvement. Additionally, the use of other types of sensors, either already present on or easily added to the pipe inspection vehicle, may prove to be useful for further improvement.

Since we know there are undetected defects in our dataset [2], it would be an interesting experiment to see if a classifier trained on data where these are unlabeled, is still able to find these defects in its own training set. To achieve this, the false positive detections would have to be re-classified by a human operator. Hopefully, this would indicate that the classifier detected defects that we *thought* were false positives, but were in fact true positives. Unfortunately, this is beyond the scope of this work.

Acknowledgements

The authors would like to thank *vandervalk+degroot* for kindly providing us with the dataset.

This work is part of the Cooperation Programme TISCA (Technology Innovation for Sewer Condition Assessment) with project number 15343, which is (partly) financed by NWO domain TTW (the domain applied and Engineering Sciences of the Netherlands Organisation for Scientific Research), the RIONED Foundation, STOWA (Foundation for Applied Water Research) and the Knowledge Program Urban Drainage (KPUD).

References

[1] N. Stanić, J. G. Langeveld, F. H. Clemens, HAZard and OPerability (HAZOP) analysis for identification of information requirements for sewer asset management, Structure and Infrastructure Engineering 10 (11) (2014) 1345–1356, doi: 10.1080/15732479.2013.807845.

[2] J. Dirksen, F. Clemens, H. Korving, F. Cherqui, P. Le Gauffre, T. Ertl, H. Plihal, K. Müller, C. Snaterse, The consistency of visual sewer inspection data, Structure and Infrastructure Engineering 9 (3) (2013) 214–228, doi: 10.1080/15732479.2010.541265.

[3] R. Wirahadikusumah, D. Abraham, T. Iseley, Challenging issues in modeling deterioration of combined sewers, Journal of Infrastructure Systems 7 (2) (2001) 77–84, doi: 10.1061/(ASCE)1076-0342(2001)7:2(77).

[4] TISCA programme funded by NWO-TTW, SewerSense – Multi-Sensor Condition Assessment for Sewer Asset Management, URL `https://www.nwo.nl/onderzoek-en-resultaten/onderzoeksprojecten/i/77/27077.html`, accessed: 2019-03-25, 2016-2020.

[5] R. Szeliski, Computer Vision: Algorithms and Applications, Springer-Verlag, Berlin, Heidelberg, 1st edn., ISBN 1848829345, 9781848829343, 2010.

[6] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg, F. Li, ImageNet Large Scale Visual Recognition Challenge, Computing Research Repository abs/1409.0575, URL `http://arxiv.org/abs/1409.0575`, accessed: 2018-08-13.

[7] S. Hoo-Chang, H. R. Roth, M. Gao, L. Lu, Z. Xu, I. Nogues, J. Yao, D. Mollura, R. M. Summers, Deep convolutional neural networks for computer-aided detection: CNN architectures, dataset characteristics and transfer learning, IEEE transactions on medical imaging 35 (5) (2016) 1285, doi: 10.1109/TMI.2016.2528162.

[8] M. Oquab, L. Bottou, I. Laptev, J. Sivic, Learning and transferring mid-level image representations using convolutional neural networks, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 1717–1724, doi: 10.1109/CVPR.2014.222, 2014.

[9] C. M. Bishop, Pattern Recognition and Machine Learning (Information Science and Statistics), Springer-Verlag, Berlin, Heidelberg, ISBN 0387310738, 2006.

[10] M. J. Chae, D. M. Abraham, Neuro-fuzzy approaches for sanitary sewer pipeline condition assessment, Journal of Computing in Civil Engineering 15 (1) (2001) 4–14, doi: 10.1061/(ASCE)0887-3801(2001)15:1(4).

[11] M.-D. Yang, T.-C. Su, Automated diagnosis of sewer pipe defects based on machine learning approaches, Expert Systems with Applications 35 (3) (2008) 1327–1337, doi: 10.1016/j.eswa.2007.08.013.

[12] W. Guo, L. Soibelman, J. Garrett Jr, Automated defect detection for sewer pipeline inspection and condition assessment, Automation in Construction 18 (5) (2009) 587–596, doi: 10.1016/j.autcon.2008.12.003.

[13] M. R. Halfawy, J. Hengmeechai, Efficient algorithm for crack detection in sewer images from closed-circuit television inspections, Journal of Infrastructure Systems 20 (2) (2013) 04013014, doi:10.1061/(ASCE)IS.1943-555X.0000161.

[14] M. R. Halfawy, J. Hengmeechai, Automated defect detection in sewer closed circuit television images using histograms of oriented gradients and support vector machine, Automation in Construction 38 (2014) 1–13, doi:10.1016/j.autcon.2013.10.012.

[15] S. S. Kumar, D. M. Abraham, M. R. Jahanshahi, T. Iseley, J. Starr, Automated defect classification in sewer closed circuit television inspections using deep convolutional neural networks, Automation in Construction 91 (2018) 273–283, doi: 10.1016/j.autcon.2018.03.028.

[16] J. Myrans, R. Everson, Z. Kapelan, Automated detection of faults in sewers using CCTV image sequences, Automation in Construction 95 (2018) 64–71, doi: 10.1016/j.autcon.2018.08.005.

[17] J. Myrans, Z. Kapelan, R. Everson, Combining classifiers to detect faults in wastewater networks, Water Science and Technology 77 (9) (2018) 2184–2189, doi:10.2166/wst.2018.131.

[18] CEN, EN 13508-2: Condition of drain and sewer systems outside buildings, Part 2: Visual inspection coding system, European Norms, 2003.

[19] PANORAMO® 3D Optical Pipeline Scanner, `http://www.rapidview.com/panoramo_pipeline.html`, date of access: 2018-12-05, 2015.

[20] R. Sitzenfrei, M. Mair, M. Möderl, W. Rauch, Cascade vulnerability for risk analysis of water infrastructure, Water Science and

Technology 64 (9) (2011) 1885–1891, doi:10.2166/wst.2011.813.
[21] J. Shore, R. Johnson, Axiomatic derivation of the principle of maximum entropy and the principle of minimum cross-entropy, IEEE Transactions on Information Theory 26 (1) (1980) 26–37, doi:10.1109/TIT.1980.1056144.
[22] I. Goodfellow, Y. Bengio, A. Courville, Y. Bengio, Deep learning, vol. 1, MIT press Cambridge, ISBN 9780262035613, 2016.
[23] Y. Bengio, Practical recommendations for gradient-based training of deep architectures, in: Neural networks: Tricks of the trade, Springer, 437–478, doi:10.1007/978-3-642-35289-8_26, 2012.
[24] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting, The Journal of Machine Learning Research 15 (1) (2014) 1929–1958, ISSN 1532-4435, URL http://dl.acm.org/citation.cfm?id=2627435.2670313, accessed: 2019-03-25.
[25] L. A. C. Millard, M. Kull, P. A. Flach, Rate-Oriented Point-Wise Confidence Bounds for ROC Curves, in: T. Calders, F. Esposito, E. Hüllermeier, R. Meo (Eds.), Machine Learning and Knowledge Discovery in Databases, Springer Berlin Heidelberg, Berlin, Heidelberg, ISBN 978-3-662-44851-9, 404–421, 2014.
[26] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, URL https://www.tensorflow.org/, accessed: 2018-08-15, 2015.
[27] G. v. Rossum, Python tutorial, technical report CS-R9526, https://ir.cwi.nl/pub/5007/05007D.pdf, date of access: 2019-03-25, 1995.
[28] D. H. Wolpert, W. G. Macready, No free lunch theorems for optimization, IEEE Transactions on Evolutionary Computation 1 (1) (1997) 67–82, doi:10.1109/4235.585893.

Appendix A. Convolutional Neural Networks

Neural networks are a type of classification algorithm modelled after the way synapses in the human brain activate and pass information along [22]. Convolutional neural networks are a classification algorithm that has been show to work particularly well with image data [5]. This section will explain them into as much detail is required for the context of our work.

Appendix A.1. Rosenblatt's Perceptron

Neural networks are composed of neurons, smaller elements that perform a simple function. The network itself performs functions much more complicated than the neurons are capable of. The simplest neuron is Rosenblatt's perceptron [22]. A perceptron has inputs, weights, a bias, and an activation function. Each input is multiplied with its assigned weight, and all are added together with the bias. The single scalar resulting from this operation is passed through the activation function, which can be any function, but is usually a non-linear non-decreasing function [9]. (The reason it is non-linear is that if it were linear, the network itself would learn a linear function in the inputs, which means the network *would* be limited to a function no more complex than the neurons themselves.) Traditionally the step function was used, sigmoidal functions or piecewise linear functions are also common [22].

The perceptron itself is a neural network, and can be employed as a classifier as defined in section 3.1, by using it as a function to describe a relationship between data and labels. In short:

$$f(\mathbf{x}) = \hat{y} = \phi\left(w_0 + \sum_i x_i w_i\right) \tag{A.1}$$

where x_i are the inputs, w_i are the weights, w_0 is the bias, $\phi(\cdot)$ is the activation function, and $\hat{y}$ is the output.

The weights and bias are initialized to random values, which means its output $\hat{y}$ is initially unlikely to resemble its target y much at all. The perceptron is trained through an iterative process called *backpropagation*, which brings the output $\hat{y}$ closer to the target y with every iteration, by changing the weights and bias by slight increments.

We feed a single object data x into the perceptron, which returns output data $\hat{y}$. We then compare $\hat{y}$ to the expected output y with the loss function, and determine the derivative of the loss with regards to $\hat{y}$. We use the differentiation chain ruleto calculate the derivative of the loss with respect to each weight. A gradient step towards minimizing the loss is calculated by multiplying each weight's derivative by a (typically small) learning rate α and the weights are adjusted. This is called the *backpropagation* step [22].

After sufficiently many iterations of the backpropagation process, the perceptron will have converged to a function that minimizes the loss function for the data it was trained on.

To make a slightly more complex neural network, capable of learning more complex relations between input and output, we can chain multiple perceptrons together to create the multi-layer perceptron [22]. The idea is to have several *layers*, with each layer containing some number of perceptrons. The first layer has the data we want to learn from as its input, and each perceptron processes this data in parallel, each giving a single scalar as output. The perceptrons in the next layer then take these outputs from the first layer as inputs, applying their weights, bias, and activation function as if these were regular inputs, et cetera for the consequent layers. The final layer has as many neurons as the dimensionality of the output y, which we have called D in section 3. Such a network with a single direction between the layers (i.e. no loops) is called a *feed-forward* neural network. The backpropagation process still works the same way, except that the gradients require more complex calculations with each layer added. This way, adding additional layers allows us to model more complex functions by adding the same elementary neurons. An illustration is shown in figure A.1.

In this work we will be using the exponential linear unit, or ELU, as activation function for all neurons except in the output layer, defined as:

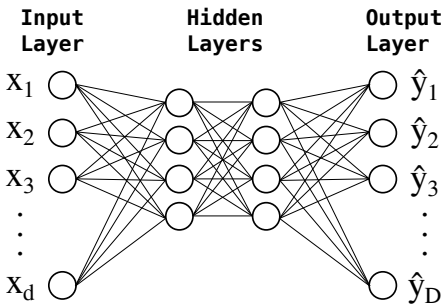

Figure A.1: An illustration of a multi-layer perceptron with 2 hidden layers.

$$\phi(x) = \begin{cases} e^x & \text{if } x \leq 0 \\ x + 1 & \text{if } x > 0 \end{cases} \tag{A.2}$$

The layers of neurons described in this section are called *dense* or *fully-connected* layers, as each neuron in a layer is connected to each neuron in the next layer. To construct a convolutional neural network, two additional layer types are required: the *convolutional* layer, a layer in which perceptrons have limited connections with the previous and next layers, and the *pooling* layer, a layer in which the neurons also have limited connections, but more importantly, the neurons perform a significantly different function from perceptrons.

Appendix A.2. Convolutional layers

In signal processing theory it is common to apply filters to signals through the use of a convolution operation. In the case of images, these filters can be used to smooth or sharpen certain patterns in images, like edges, corners, or textures. Filter are applied through the process of *convolution*, where a filter is moved across the image, and at each position the "overlap" between the image and the filter is calculated.

Discrete[4] convolution in two dimensions is a mathematical operation defined as:

$$I(x,y) * k(x,y) = \sum_{u=-\infty}^{\infty} \sum_{v=-\infty}^{\infty} I(u,v) \cdot k(x-u, y-v) \tag{A.3}$$

In practice we might smooth image $I(x, y)$ by convolving it with a Gaussian *kernel* $k(x, y)$. It should be noted that while convolution is defined on an infinite domain, in practice both the image and the kernel will be non-zero only within limited domains and the convolution can be performed by summing only over those domains.

We can simplify the perceptron somewhat to simulate this function. The advantage here is that we can use the backpropagation process to learn image filters, instead of having to design the filters based on what structure we expect the images to contain.

A simple convolutional layer consists of as many neurons as the previous layer in the network, but neurons are connected only to neurons in the previous layer that *surround* the neuron in the same location. In a one-dimensional setting, this would mean that neuron n_i in the convolutional layer receives as input the output from neurons n_{i-s} to n_{i+s} in the previous layer. We call $2s+1$ the *filter size*, as each neuron in the convolutional layer has $2s + 1$ inputs. In a higher-dimensional setting (such as our two-dimensional images), the neighborhood around neuron n_i that is connected to n_i in the next layer extends in all dimensions.

Additionally, each neuron in the convolutional layer has the same weights, just different connections. This is called *weight sharing* [22], and it results in an equivalent of the convolution operation performed by the network, as each neuron applies the same filter, but at a different location in the image.

Three additional hyperparameters are used to define a convolutional layer, the *stride*, the *depth*, and the *edge condition*.

The stride, or step size, incorporates a subsampling mechanism into the layer, if a stride of x is chosen in a one-dimensional setting, the convolutional layer contains N/x neurons, where N is the neurons in the previous layer. In other words, $(x-1)/x$ neurons are discarded. The reasons one might do this, is that if neither the kernel, nor the image, consist of white noise, adjacent samples in the result will be highly correlated, and we can reduce the neurons in the layer (and with it the computational requirements of training the network) without losing any information present in the input data. In higher-dimensional settings, the stride has as many dimensions because it can be defined for every dimension individually.[5]

The depth of a convolutional layer determines how many filters are applied in parallel. With a depth of 1, every neuron is just a perceptron with limited connections, as mentioned earlier in this section. However, when the depth increases, each neuron becomes a collection of these limited perceptrons, and the output of the neuron is simply a vector of these perceptrons' outputs.

Finally, the edge condition determines what happens at the edges of an image, when the input does not fully overlap with the filter size. For example, with a filter size of $2s+1$, the behavior for the first and last s samples of the input is poorly defined. It is an option to simply discard these cases, but this results in a layer size that is dependent on the filter size. Another option is to pad the input by half the filter size (rounded down) and then discard the

[4] As opposed to continuous convolution, which is defined for continuous signals. Our signals are images, consisting of pixels at discrete locations, and as such continuous convolution is beyond the scope of this article.

[5] In our case, the x and y coordinates of the images are of the same modality (a single pixel in one direction is the same real-world distance as a single pixel in another dimension), and it makes no sense to have different strides for the two dimensions.

violating cases, which does ensure the convolutional layer to be of the same size as the input layer. In this work we have chosen to apply zero-padding, but we believe that it does not matter too much, as our images are mostly black around the edges anyway.

Appendix A.3. Pooling layers

A pooling layer is similar in structure to a convolutional layer, but instead of consisting of limited perceptrons, it consists of neurons that perform a non-linear function. They are often placed immediately after a convolutional layer, to introduce a non-linearity that allows the network to learn more complex structures than it would with just convolutional and dense layers [22].

The most common example is the max-pooling layer. It has the same structure as the convolutional layer, taking as its input only the surrounding neurons from the previous layer, and has a filter size and stride as well. However, instead of multiplying the input by weights and summing, it simply outputs the maximum value of each depth slice from the inputs.

By taking the maximum value over a spatial window, we perform a dimensionality reduction (assuming a stride larger than 1). The reason the maximum value is used is that this works well with the convolution operation: convolution overlaps 2 signals (image and kernel in this case) and returns the inner product, so a high response at a location means that the image resembles the kernel at that location. By taking the maximum, we achieve a sense of how much that portion of the image resembles each kernel (each depth slice).

Appendix A.4. Network Design and Hyperparameter Optimization

Conventional wisdom in recent image processing techniques dictates a set of design patterns for convolutional neural network architectures that have been shown to work well. Specifically, the most common pattern is to interlace convolutional layers with max-pooling layers. The input is fed through some number of these interlaced layers successively before being passed into some number of dense layers, before being passed to the output.

The issue with designing and optimizing these networks is that the search space is infinite, and while there exists a lot of knowledge on what does and does not work for common datasets and tasks, still much research has to be done on *why* these are good design patterns [22].

Commonly, networks are used that have been shown to work well on difficult datasets (such as ImageNet), often pre-trained to reduce the time it takes to train on the new dataset, but the "no free lunch" theorem [28] dictates that there cannot be a single architecture that works best on all different tasks and datasets.

Appendix B. ROC and PR curves

Figures B.1, B.2, B.3, and B.4 show the ROC and PR curves for our proposed CNN, for classification on single images and entire pipes, with a plot for each defect type, and the areas under the curve (AUROC or AUPR) above each plot.

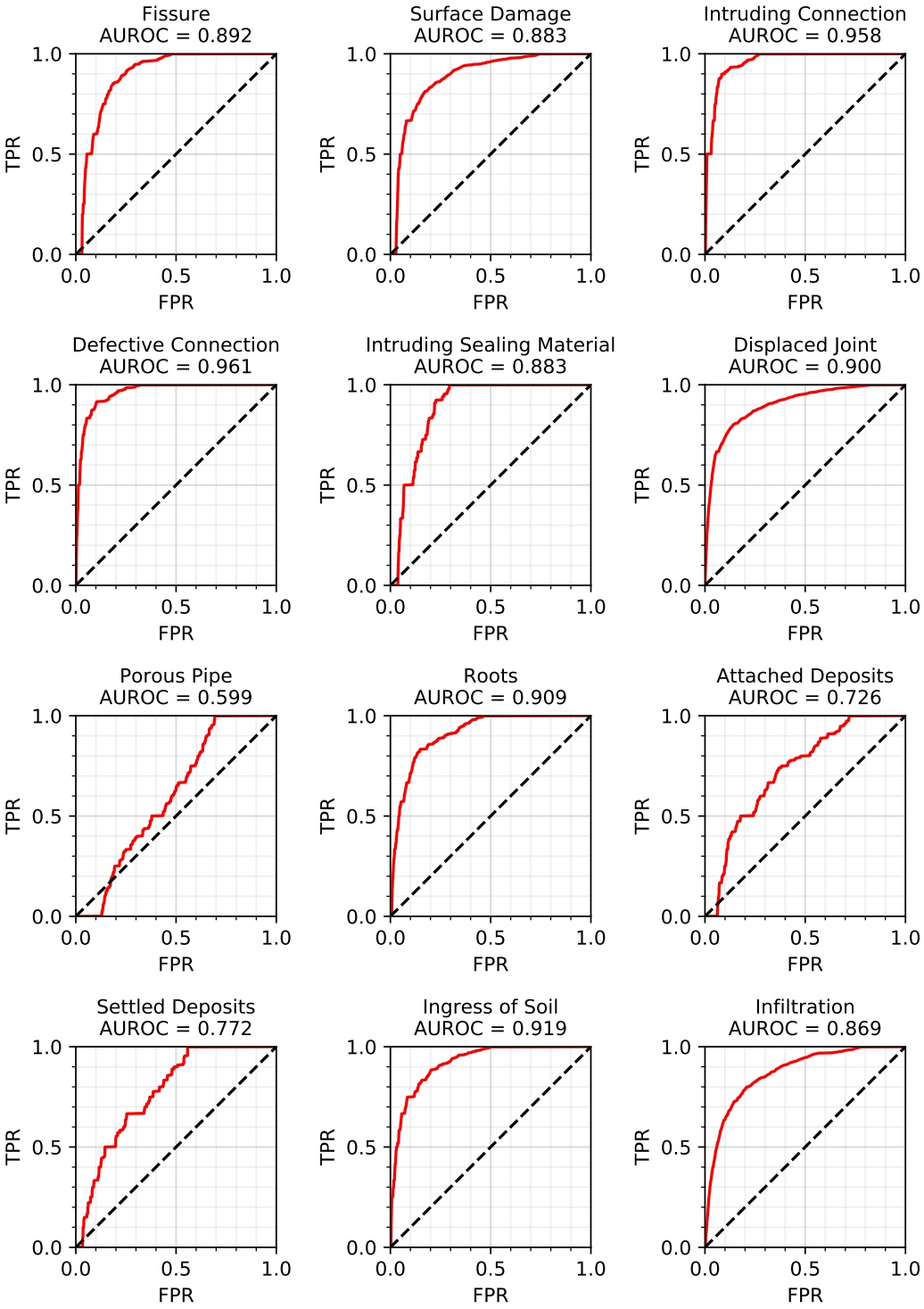

Figure B.1: ROC Curves for the proposed CNN when classifying single images.

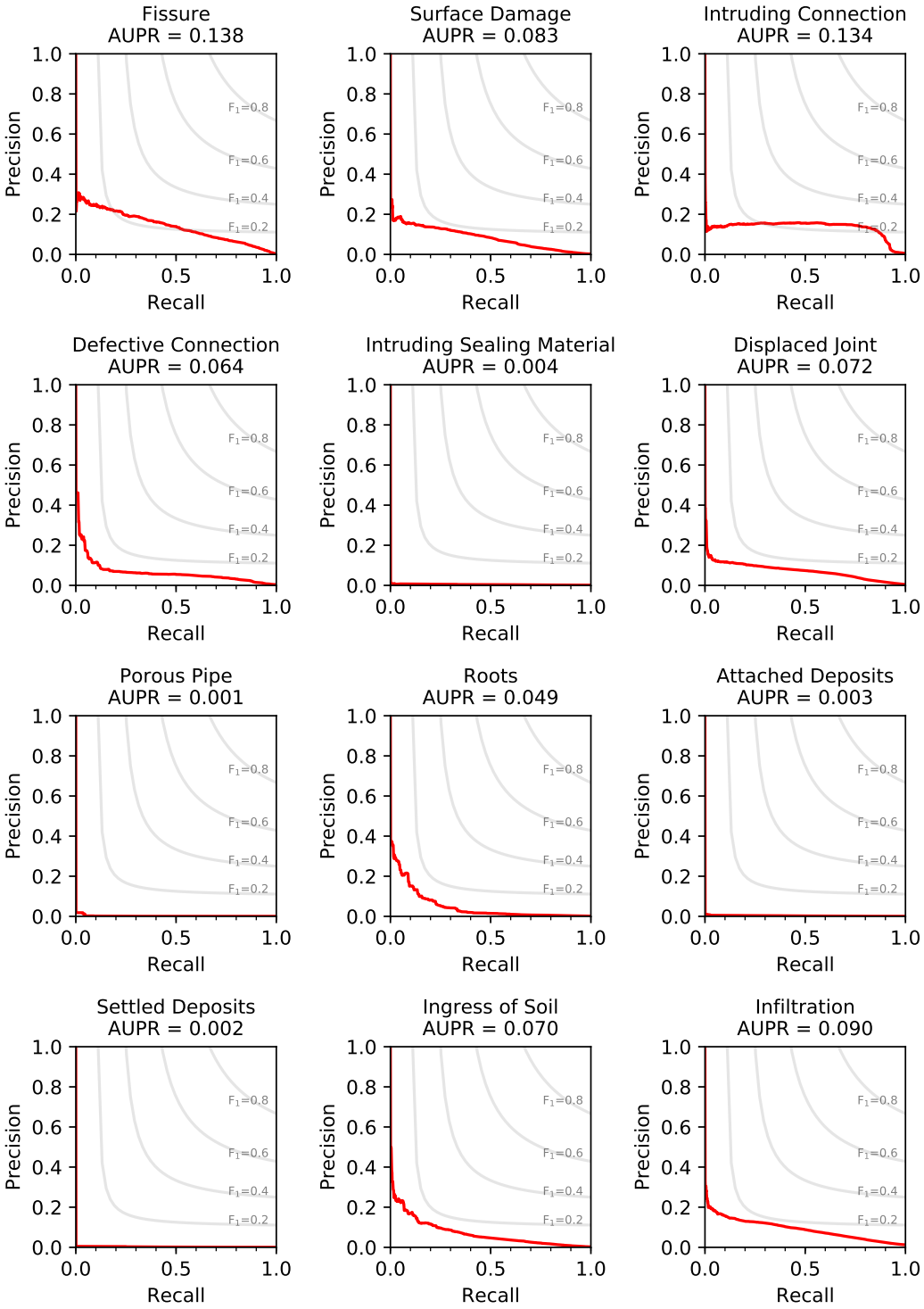

Figure B.2: Precision-Recall Curves for the proposed CNN when classifying single images.

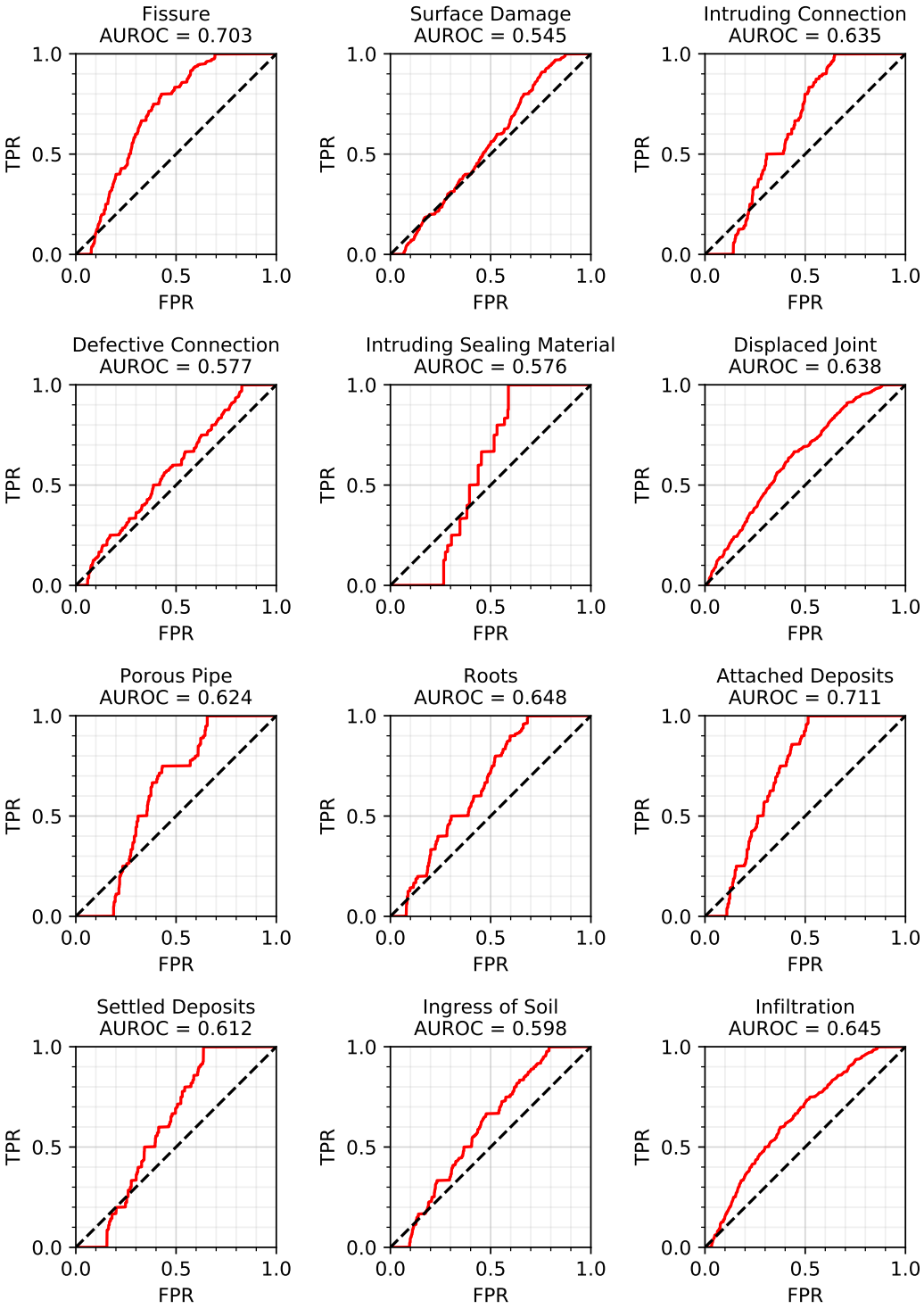

Figure B.3: ROC Curves for the proposed CNN when classifying entire pipes.

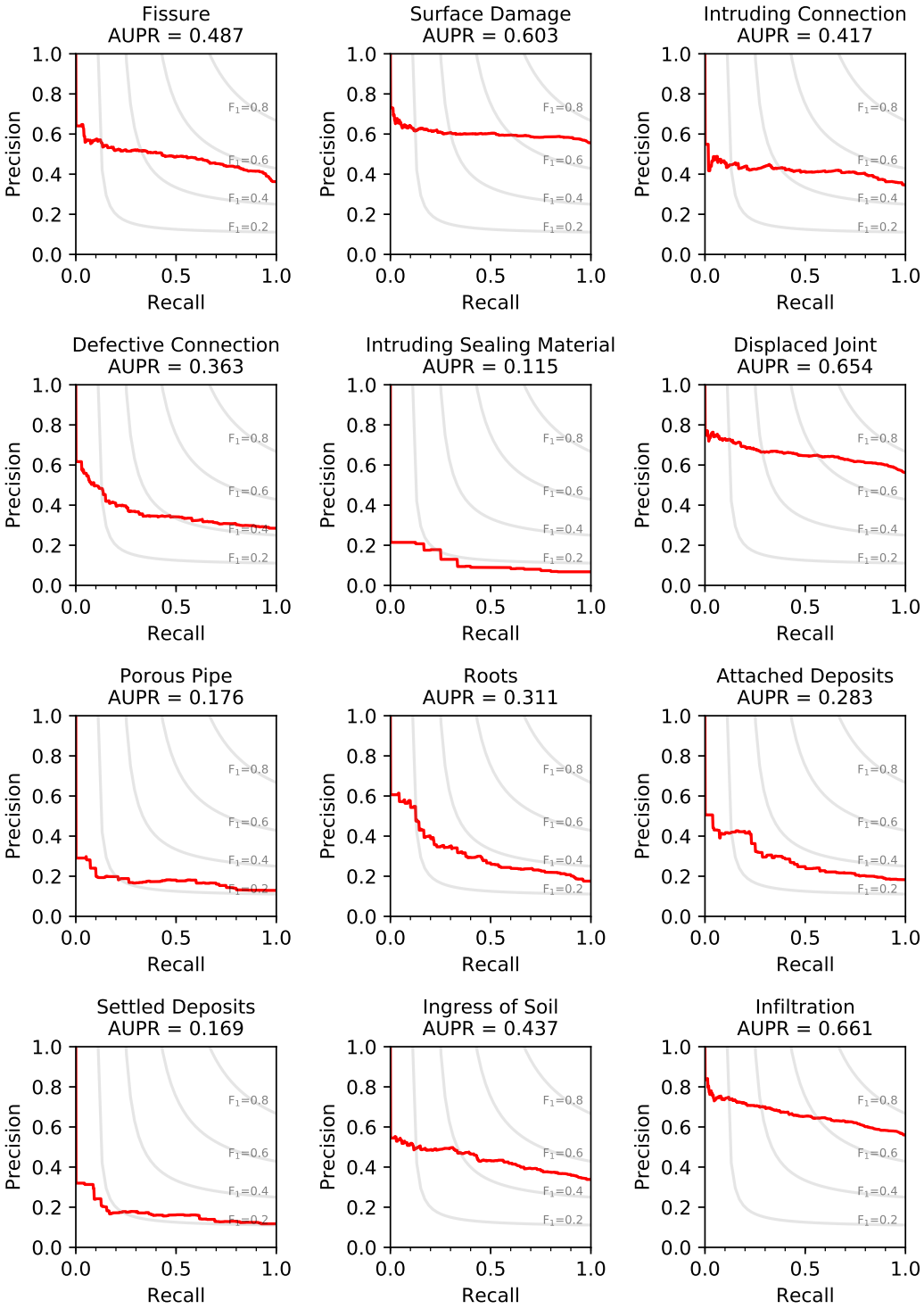

Figure B.4: Precision-Recall Curves for the proposed CNN when classifying entire pipes.