Capstone Final Report

Ninja Turtles
Kyle Bowman, Muskaan Narula, Carrie Choi, Junlin Huang

Client: Sam White

December 8th, 2021

Throughout this semester, our team has continued working with our client, Sam White at the Western Virginia Water Authority, to implement an efficient way to automate defect detection in sewage pipes. The client's goal with our project is to decrease the amount of manual labor it takes to go through the large amount of sewer defect data and create a model that could automatically detect the sewage defects. Our client provided us with over 140 videos that contained sewage defects. We manually labeled all frames with defects present. By using machine learning techniques, we were able to get a successful model to complete our client's goals. We identified YOLO as the stronger model choice over Faster-RCNN and recommend training on a grayscale dataset rather than a colored one. We outline a system architecture on how this model should be used and provide many ideas for future improvements such as utilizing a massive public sewer defect dataset containing over 1.3 million images and using data augmentation to provide a broader range of scenarios to train the model on.

1 Problem Statement

The objective of our project is to evaluate a variety of machine learning algorithms and determine their viability for helping identify sewer system defects. These algorithms should be able to identify defects such as roots and root balls. Our client, The Western Virginia Water Authority (WVWA), is swamped with thousands of hours of sewer videos to review. Sewer pipes that are not maintained in a timely manner are prone to failure which can harm our environment, negatively impact the health of citizens, and cost a lot of money to repair. Our project would help alleviate this by automating some of the process. With this project, we sought to answer questions such as: which algorithm is the best to use for object detection in sewer systems, how do different ways of labeling data impact the performance of these models, and whether these models could potentially be viable for detecting sewer defects.

2 Ethical Considerations

Ethical problems to consider when building our model include potentially outdated data as well as the chance of financial loss due to false detection. The data we were using to train our model consists of camera footage from the sewage pipes over a long period of time. However, as these sewage pipes are being redone and updated, there is a chance that our model would no longer be fit enough to accurately detect all the damages in the pipes. Therefore, it was important for us to constantly update and train the data in order to accurately detect sewage defects and avoid model obsolescence. Our model could also lead to ethical issues such as financial loss for the company. Although our model is aimed to have a high accuracy rate, there is still a chance for inaccuracy in the model that needs to be considered. When building our model, we are consistently training our algorithm to detect different shapes and sizes of sewage defects. Once these defects are detected, our model would essentially flag the pipe as a root, root ball, etc. However, in some videos the lighting makes it hard to see objects and our model could fail to identify a defect. Failing to flag a major defect in a sewer pipe could lead to imminent collapse that would have been avoided had a human reviewed the footage. Therefore, it was crucial for us to train our model with data that included a variety of sizes and shapes so that our model would be able to determine defects as accurately as possible.

3 Literature Review

In his article, "Automated condition assessment of buried sewer pipeline using Computer Vision Techniques", Sunil Sinha was solving a similar problem as ours - his goal was to detect cracks in pipes given large amounts of CCTV footage of pipe walkthroughs.[1] Sinha discusses techniques that were used in detecting cracks.[1] This is critical to our project as detecting and flagging pipe cracks is within our project scope. Learning about some of the machine vision techniques he used will help us formulate our solution as well. Sinha .[1]arted by breaking the footage into individual frames.[1] Instead of analyzing video, he focused on a few frames.[1] He started by turning the image into grayscale.[1] In his experience the color in the image was not useful, and resulted in extra computation for the computer.[1] Then he aimed to morph the image to make the crack distinguishable from the random background patterns, pipe joints as well as pipe laterals.[1] Sinha discusses various equations to make this decision for every image.[1] Looking at his morphing techniques could help us define how we detect pipe cracks. We could also explore how to apply this logic for other iden-

tification too, for example, root balls.

Similar to Sinha's article, in Xianfei Yin, in his paper called "Automation for Sewer Pipe Assessment", discusses ways to improve the automation of the sewer pipe assessment process and proposes different methods to see what is most efficient. Yin suggests two main methods for this project. The first method is to generate the sewer pipe conditions and any other necessary information via text in excel form.[2] The second method is to create a user-friendly software with all the developed functions.[2] The CCTV videos would be processed by a defect detector called YOLOv3 which has been trained with 4,065 different defects and 3,664 images.

He also focused on building a novel video interpreter (VIASP) that would take the CCTV videos and return an output via text that states all the defects in sewer pipes.[2] They used this program in order to exclude the noise and merge frames that show the same defect.[2] These gaps (the frames that have no defect) could potentially increase the level of difficulty when interpreting through the CCTV videos. Rakiba Rayhana's paper, "Automated Vision Systems for Condition Assessment of Sewer and Water Pipelines", expresses similar problems to the previous articles. But Rayhana uses different application approaches for her solution. She classified the typical failures in sewer and water systems with different failure types, which includes types like sewer pipeline blockage, root intrusion, open-joints and cracks, along with water pipeline leakage, dis-junctions, corrosion and fractures.[3] The failures that are mentioned in this paper are similar to what we are working on with our project.

Rakiba Rayhana suggests six techniques that can be used to detect the pipeline failures, including CCTV technique, Sewer Scanner and Evaluation Technology (SSET), Laser-Based Scanning Technique, Zoom Camera Technique, Digital Scanning Technique, and Electro Scanning Technique.[3] The CCTV technique that was mentioned in the paper fits the client since the client is going to provide CCTV footage as our raw data.

Different from Sinha and Yin's article, Abbhishek Gupta's article, "Deep learning for object detection and scene perception in self-driving cars", analyzes a variety of deep learning models and compares their effectiveness in object detection and scene perception for self-driving cars.[3] The problem of tracking moving objects across many different frames of a video from the point of view of a moving camera is almost exactly the same problem that we are aiming to solve. He analyzed several deep learning models including Autoencoder (AE). Of these, some offer benefits such as the ability to learn from unlabeled data or analyze spatial data.[4] He identified AE as invariant to image transforms, which would be useful for identifying objects while the camera in the sewer is turned at different angles.

4 Project Criteria

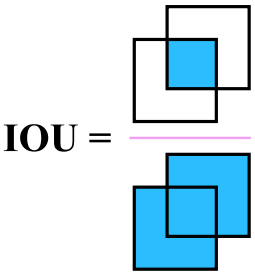
When implementing our model, there were many necessary criteria to consider. Our solution needed to have a high accuracy rate, be intuitive for the end user, and our code needed to be easily readable for non technical users. The most important criteria was to shorten the time it took to detect the sewer defects and have the least amount of manual work possible. Our project was greatly influenced on the quality of our data frames which determined how accurately our model was able to detect the roots and root balls in the camera footage. Additionally, it was important for us to have a variety of defect shapes and sizes in our training data in hopes to increase the accuracy rate.

5 Selected Solutions

One of the primary deliverables planned for our client was an algorithm that could detect defects in sewer pipes using only video footage from a Closed Circuit Television (CCTV) camera. With a variety of defects often present within a single frame of video, an object detection algorithm was a natural fit. We chose to analyze the effectiveness of two different object detection algorithms with a focus on specifically detecting roots. There are a variety of algorithms available for detecting objects in images, however most involve the use of Convolutional Neural Networks (CNNs). We were primarily interested in identifying the performance and accuracy tradeoffs between two different models: Faster-RCNN and YOLO.

Faster-RCNN is a model capable of running in real-time that improved upon regular RCNN by adding a feature proposal network (FPN) that efficiently gets predictions for where objects may be in an image[6]. Detectron2 is a python library built by facebook that can perform Faster-RCNN object detection. We chose this as one of our solutions. The YOLO model format works on 24 distinct convolutional layers and is able to run much faster than traditional object detection frameworks[5]. The YOLO model format is implemented in the Darknet python library and we chose it as another one of our solutions.

A variety of metrics exist for object detection. We will be utilizing the standard COCO metrics throughout this paper. In brief, if one has a predicted bounding box and a true bounding box, calculating the intersection area over the union area (IOU) of these boxes will give a metric for how close they are to each other[7]. This is pictured below. Before checking these distances, prediction boxes with less than a certain confidence threshold are typically thrown out.



After determining the IOU for all predicted bounding boxes, only the highest for each intersection is kept and each intersection is then determined to be a true positive or false positive based on a threshold. Typically this threshold for the IOU is around 0.5, however other numbers are sometimes used. After determining the number of true positives and false positives, we can calculate the precision of the model for each class:

$$\textbf{precision} = \frac{TP}{TP+FP}$$

Another metric to consider is the recall, otherwise know as the true positive rate.

$$\textbf{recall} = \frac{TP}{TP+FN}$$

With a hypothetical perfect model, both precision and recall are 1. In reality, depending on the original confidence threshold chosen, you will get a trade-off between the precision and the recall. A high confidence threshold will make your model precise in its predictions, but it will miss many detections and thus have a low recall. To make comparing different models easier, the precision is typically averaged over all values of recall to compute the average precision (AP)[7]. Depending on what you pick as the IOU threshold for true positives, you will get a different AP. In most COCO dataset evaluations, AP50 and AP75 are computed. The symbol AP is left alone to mean the average of AP50 - AP95 with increments of 5 percent.

6 Results

6.1 Text Extraction

In the preprocessing stage of our project, we planned to extract the text overlays from the camera footage given to us. Accessing this text would give us useful information about each frame in the video, such as the total distance the camera has traveled in the pipe, the date, and the time. Additionally, our client's database is organized based on distances in the pipe, not on video times. So, extracting the distance information would allow us to automatically select key frames in videos for us to perform analysis on.

A survey of various optical character recognition (OCR) softwares led us to find Google's Tesseract OCR. It is surprisingly easy to set up and use through python. Tesseract works best on black and white images where the text is pure white and the background is pure black (a binary image), so further preprocessing on our videos was necessary. Prior to receiving larger datasets, we assumed converting a video to white text and black background would be easy because the bright white text stands out so clearly on the dark sewer pipes. See Figure 1 for an example of this. Turning this into a binary image is as simple as creating a threshold for the brightness value of each pixel. Anything greater than that threshold becomes pure white and anything less becomes pure black. The Python library OpenCV provides built-in functionality to perform this simple thresholding. In Figure 1, we can see the output from performing basic thresholding using a brightness threshold of 200. Feeding Figure 1 into the Tesseract OCR yields the exact text string shown in the image. However, once we received our dataset, we realized that this method would not be enough. Some sewer videos have significantly different lighting and some have backgrounds brighter than the text itself. The text is still visible in these cases due to the black outline on the text. However, extracting it would take more effort. Applying a sharpening filter to the image made the text borders more defined (See Figure 2). Another method of binarizing an image is using adaptive thresholding. Instead of simply applying a global threshold to each pixel, adaptive thresholding calculates the threshold uniquely for every pixel depending on how bright its neighbors are within a certain window. There are multiple variations, but we looked at two. Mean thresholding simply calculates the mean of the neighboring pixels' brightness to determine the threshold. Gaussian thresholding weights the nearby pixel's brightness depending on distance. We applied this and were able to mostly isolate the text borders. See Figure 5.4. Despite the text being quite readable, Tesseract OCR is only trained to identify solid text, not borders of the text. Additionally, we ran into other problems where the text was in different formats, locations, and colors depending on the video we looked at. Sometimes the text was white with a border, while other times it was black with no border. Due to all these differences and the difficulty of retraining Tesseract, it seems to us that a generalized text extractor for the sewer videos is out of our reach.

6.2 Data Analysis

Some data was also provided by our client regarding root-ball defects found during inspections done since the 1960s. This data set included information such as the age of the pipes, the amount of repairs done, and even the material the pipes were made of etc. We did some analysis of this data to determine any significant trends or observations that could be made. The goal of this was to explore pipe defects beyond detection, and consider factors that could affect the existence of the defects as well.

Figure 1: Simple Thresholding (Brightness > 200)

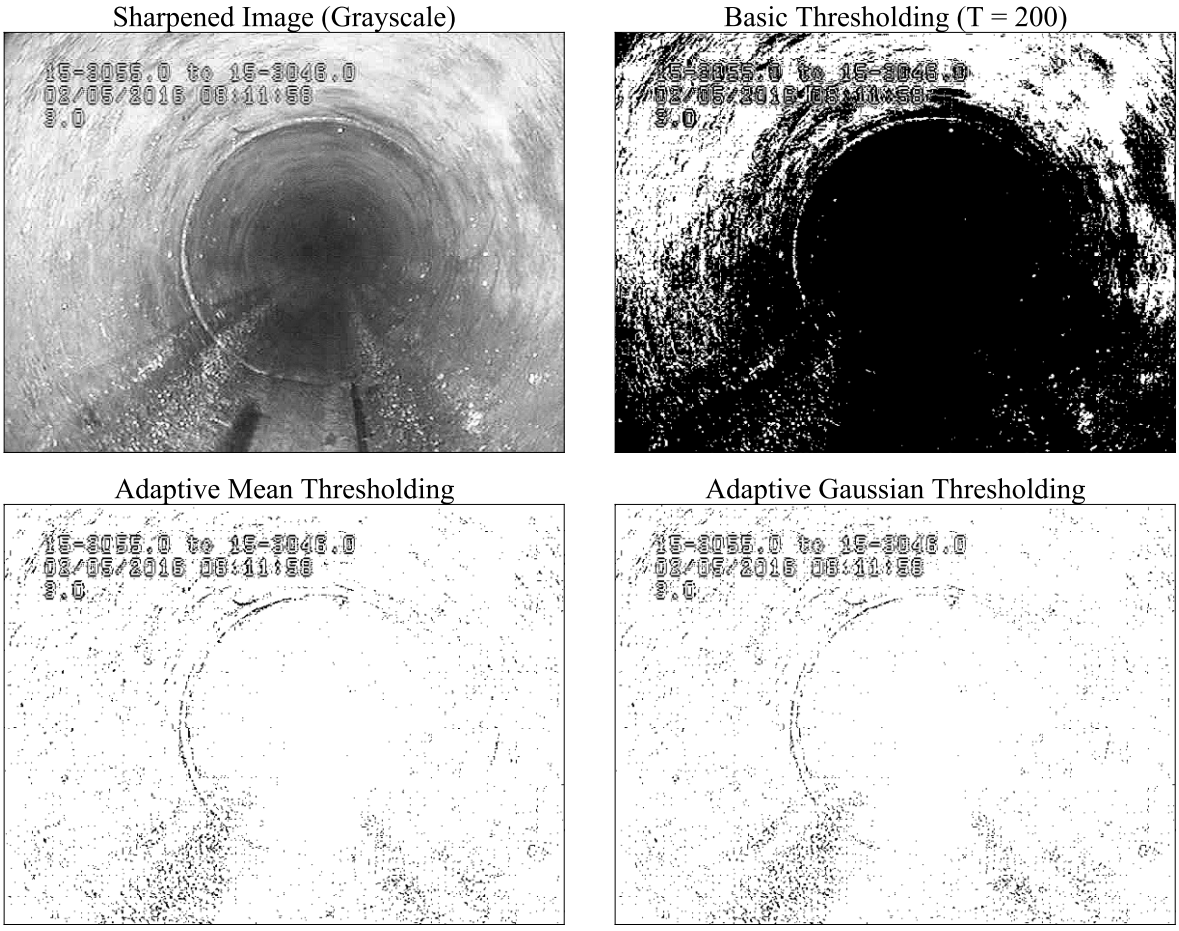


Figure 2: Adaptive Thresholding on a Difficult Image

First the data-set was re-grouped from being grouped by inspection to being grouped by each pipe, so information like whether the pipe flowed upstream or downstream, material of the pipe was grouped likewise. This allowed us to look at each pipe individually as well.

The first characteristics of pipes considered was the whether it flows upstream or downstream.This was compared to the number of repairs required by the pipe itself. However, the direction of flow was not statistically significant in the number or repairs needed.

Another metric that was available in the data was the material of each pipe. The relationship of this with the number of repairs needed by the pipe was assessed. This did turn out to be a significant factor. As in, the type of material the pipe was made of, affected the number of repairs, and therefore roots, the pipe had.

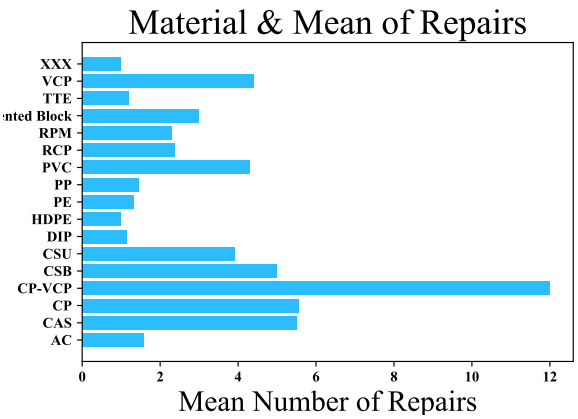


Figure 3: Mean Number of Repairs by Material

The figure above shows that CP-VCP pipes have a higher number of repairs needed. While this does not automatically mean that CP-VCP is a bad material, it could indicate a point of exploration. Our first guess was that CP-VCP pipes are older, and perhaps would need more repairs overall.

Also looking at the distribution of the types of root damage, it is evident that "Root Fine Joint" is the most common type of root-ball. This makes sense as a Root Fine Joint root-ball is a small root growth appearing from between a joint that connects two pipe segments.

6.3 Image Labeling

A crucial component of training an object detection model is having a large dataset of images with pre-labeled bounding boxes for objects in the images. In our case, this would mean hundreds of images of sewer pipes with boxes around defects such as roots. Our client provided us with 97 gigabytes of sewer footage. This equates to over 44 hours of footage over 140 different videos. Additionally, we were given an excel spreadsheet with the defects each video contained and the distance in the pipe to find the defect. Unfortunately, there was no timestamp for where to find the defects in the videos. And since we were unable to utilize automatic text extraction to match the frames to distance, we had to manually search all 140 videos. We split up the data into fourths and each of us manually searched for defects in the videos. When we found a defect, we would save the image into our dataset. Doing this, we accumulated a dataset of 561 images with roots in them. We then began the process of labeling these images.

For labeling, we originally utilized a python library called LabelImg. While this library was simple and easy to use, it didn't compare to the features of the browser-based Roboflow. In Roboflow, we are able to label images, automatically split the dataset into training/validation/testing, export to several different formats, and even augment our dataset with various transformations such as rotations or flips. The detectron model never saw these images during the training process. While the model did well in this example, not

all testing images look this good. There are a sleuth of false positives and negatives throughout the array of testing images. However, some bounding boxes seem like they are on top of the root but maybe are slightly off or have too big of an area covered or too small. It is useful to define a uniform way to define false positives and negatives.

6.4 Detectron2

For training detectron2, we split our dataset into a subset of testing, validation, and testing data on a 60/20/20 percent ratio. We trained detectron2 with a variety of configurations and compared the results on our testing data. The figure below shows an example of some visualized output after training.

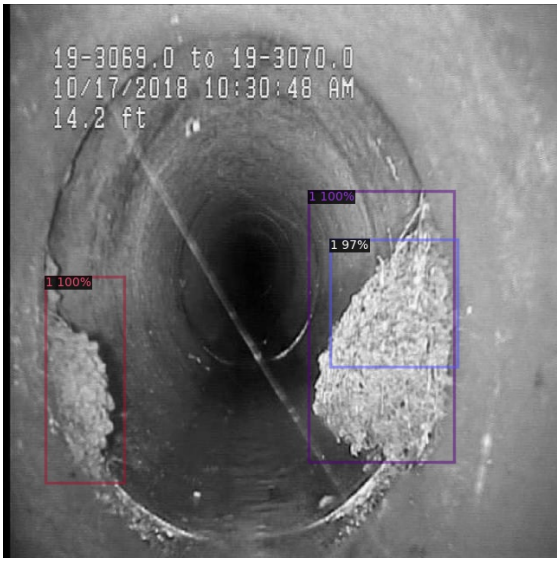


Figure 4: Predicted Bounding Boxes

When training our model for Detectron2, we wanted to compare many different model configurations across a variety of different datasets. The plan was to test out datasets with different labeling and class separations. For example, in our current dataset we only identify roots. However, many roots are full of dirt and form a large clump that is referred to as a root ball. An example is provided below. Labeling these separately could potentially offer an improvement to the model's performance. However, due to time constraints we were unable to relabel our data and test that hypothesis.

Another data difference we considered was color vs black and white. Due to the variety of colors present for different lighting conditions and pipe materials, without a large dataset, we would argue that training on color will hurt performance because there are too many variations and the model will fit more to certain color ranges rather than fit to the shape and texture a root has. We were able to test color vs black and white with detectron and compared metrics. Additionally, we added an augmented dataset that contained both colored and black and white images.

For testing different model configurations, there are a variety of defaults available with Detectron2. You can find them available on Detectron's model zoo github page. Each of these configurations has varying training speeds, inference speeds, and average precision values. We wanted to compare these and see if these speeds and precisions matched up well for sewer data rather than the pedestrian and car data they were tested on. Running each model on the ARC cluster took several hours with some of the longest runs taking up to 15 hours. Some errors occurred during training for many of the models we planned to experiment with. Due to time constraints, we were unable to debug all of these and some models are only partially trained. We chose not to include the results of running those severely undertrained models as they add little to the discussion. The model we had consistently working was the FPN-50-3x. We tested it on colored, grayscale, and mixed data for 90,000 training iterations. Our resulting AP values on the validation dataset are compiled in the table below:

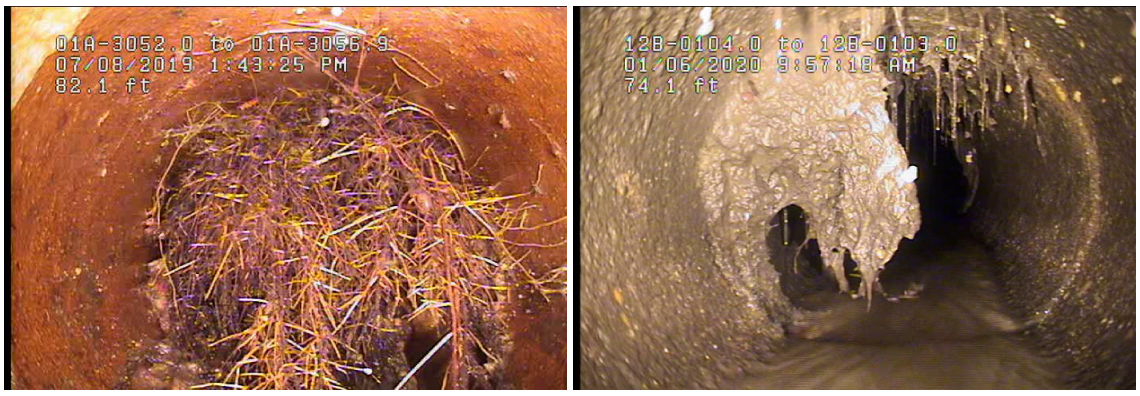


Figure 5: Stringy Roots vs Muddy Roots

Model Config	Dataset	AP	AP50	AP75
FPN-50-3x	Color	9.2	34.0	1.7
FPN-50-3x	B/W	11.4	36.6	3.2
FPN-50-3x	Mixed	10.7	38.1	3.2

Our model APs were much lower than we had originally anticipated for Detectron2. On their website, their baselines provide APs up to the high 40s and even into the 50s. However, the models have a reasonable AP for an IOU threshold of 50 percent. We explored a variety of options for improvement and we have some ideas about why the overall AP values might be so low. Our dataset is fairly limited in size. Datasets available like the COCO challenge datasets contain a few hundred thousand images. Having less than 500 could be impacting performance of the model.

Another thing we considered as a potential for decreasing our accuracy was over-fitting. Too many training iterations can cause the model to overfit the training data and out of sample predictions begin to suffer. A strategy often employed is to include a validation dataset that you use to fine tune the model. When the validation accuracy begins to suffer, you determine that further training is overfitting the model. By plotting the validation and training loss together, we are able to see this effect. In this example, by around iteration 5000 our validation loss has already diverged away and is steadily increasing. At the point of the lowest validation loss, we saved the model to a separate file and computed the metrics on that model too. Peculiarly, the AP is worse for the model we saved at the lowest validation loss.

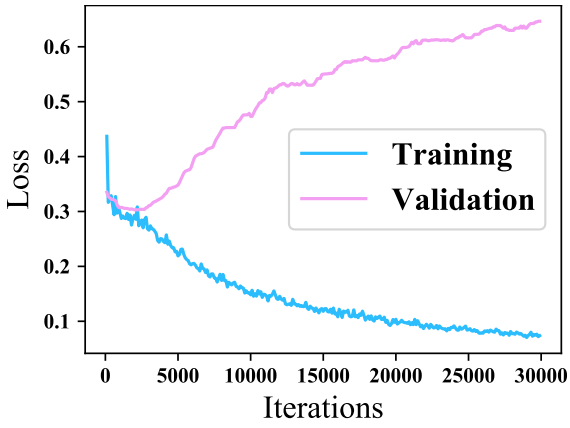
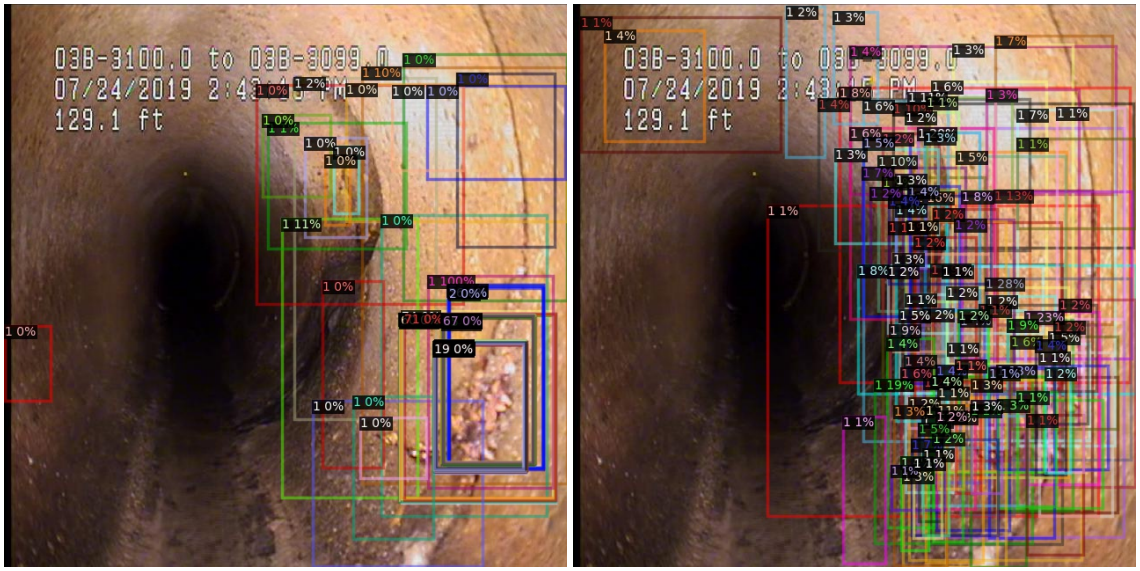


Figure 6: Train vs. Validation Loss

Interpreting this result is difficult and may take further analysis, but we have some ideas. For the rest of this paragraph we will refer to the lowest validation loss model as the "best model" and the fully trained model as the "final model". A component of the loss function for Faster-RCNN uses the negative of log of the probability p of a label u. $\mathscr{L}_{cls}(p,u) = -log(p_u)$. Thus, if there are predicted boxes with probabilities near zero, it will cause the validation loss to inflate a lot. However, these



(a) Final Model (b) Best Model

Figure 7: Near-Zero Probabilities in Final Model

probabilities near zero will have almost no effect on the average precision because they will be cut off by almost all confidence thresholds. When visualizing the final model, we find that there are many probabilities of near zero. The best model produced low probabilities, but most were above 3 or 4 percent. We are working with the inverse log, so minor changes to probability near zero produce large changes to the loss.

6.5 YOLOv5

In order for our client to have a selection with different techniques that could detect pipe failures from CCTV camera footage, we included YOLOv5 as another choice of model after we have considerable results from detectron2. By comparing the two available techniques, it makes it possible for our client to choose the best one that fits their system in future development.

For the training process of YOLOv5, we used same dataset that we used for Detectron2, with the YOLOv5 format instead of the COCO format that Detectron2 was using. We trained YOLOv5 on two different versions of the dataset, one is with all colored images, and another version is with 25 percent grey scale images. The result in the figure below is showing a 76 percent certainty of the model on this testing image. There were two model weights we considered using for YOLOv5. At the begining of the trainning process of YOLOv5, we first trained our model on the smaller model weights to test the best split of the dataset into

Figure 8: Result from all colored dataset

train, validation and test portion. Next step was to train the model on the larger model weights that were expected to have a higher mean average precision on validation and a higher overall precision.

Model	Dataset	AP0.5	AP0.95	MP
YOLOv5x	Color	61	24	68
YOLOv5x	Mixed	59	24	71

While training the model on the larger model weights, we also trained the model on the two different versions of the dataset mentioned above. The table above shows the results we have running the model on the two different dataset, even though the mixed black / white / color dataset is showing a higher mean precision, the all colored dataset shows a higher mean average precision of 0.61, which was giving us better results on the testing dataset.

We also examined the validation and training loss over iterations as we did for Detectron2. YOLOv5 validation loss also starts around a certain number of iterations, which caused overfitting. However, when we tried to stop the iterations before the validation loss increased, the model was not trained enough and was not able to label any roots in the images from the testing dataset. We could have the model trained enough to detect some roots if we stop the training process a few iterations after the validation loss increases, but it is not certain that the model would be good enough to detect as many roots as possible. One possible solution is that we increase the size of our dataset, including more than 10,000 images for the root class to avoid early validation loss.

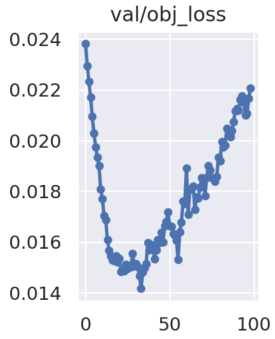


Figure 9: YOLOv5 Validation loss

6.6 System Architecture

An important consideration of the project was ensuring the scalability of our solution. Our model should be able to be used by our client to cleanly detect root-ball defects quickly over a large amount of footage.

To ensure this, we created a method for our client to easily use our model. We have developed a script with allows our client to run our model on any pipe footage they may have. The script will create a csv file containing data that helps flag where any defects detected by our model. The flagged frames are able provided to the client for further viewing.

The footage is stored by the name of the pipe segment, making it easy to identify. This is then used to create a folder where a frame by frame break down of the footage is stored. These frames will then be passed on to our model for root-ball detection. Then, those that get flagged with a high enough score are added to a csv file for further viewing. All the non-flagged frames are also deleted.

With is set up, operators only view frames of the footage that are flagged with defects, reducing their time significantly. It also allows for a large scale running of our model. Running the script on a large set of videos will return data sorted by pipe. This organization will also allow operators to check which pipes' flagged root-balls raise concern

quickly.

7 Limitations

While our results indicate that an object detection model is well suited for identifying root defects, there are some limitations. One such limitation is speed. Both Detectron2 and YOLO claim to be fast enough for real-time analysis. This definition definitely depends on the architecture you are running on. Using a CPU instead of a GPU will yield far slower results. To get through our 80 image testing set, it typically took around 10 minutes when running on a single CPU core. When you are working with thousands of hours of footage, this could become very slow very quickly. This should be taken into consideration when running videos through the system. A scalable architecture should be implemented to give evaluation tasks enough computational power.

No matter how good our model is trained on some dataset, it will never be perfect. This inherently leaves open the risk of errors and these errors will be amplified over time if the model is not re-trained on newer pipe systems. The model should not be viewed as a replacement for expert technicians who analyze footage all the time. Instead, it should be seen as a tool to aid them in doing their job faster and more effectively. For such a model to continue to be viable long-term, technicians need to be intimately involved with the process.

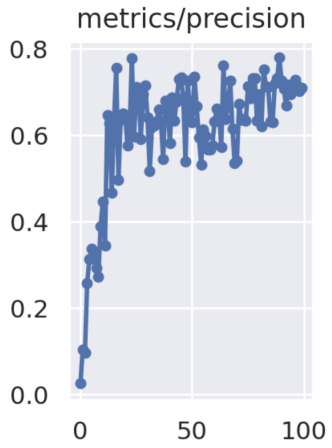


Figure 10: Trained on black, white, and color

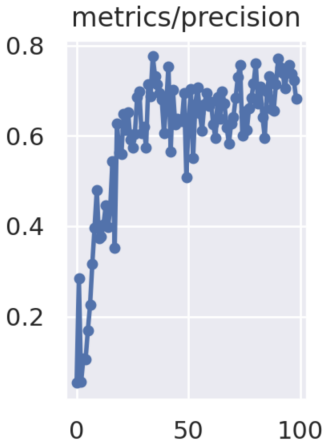


Figure 11: Trained only on color

8 Interpretation Results For Client

In order to run our scripts, you may need to modify things depending on your system. Both Detectron2 and YOLO require a graphics card to perform training. You will need to install a few python software packages such as PyTorch for Detectron2. Another option is simply using Google Collab which already has the installations laid out for you in a helpful notebook format. While Google Collab is

convenient, other considerations such as bandwidth usage and security may play into your decision to use it.

For collecting data, you may want to consider starting with a pre-made dataset. We only now discovered the Sewer-ML dataset which contains over 1.3 million images with multiple classifications that were labeled by professionals in three different companies. Using this could certainly improve your accuracy compared to our measly 500 labeled images.

The object detection models developed in this paper are an interesting proof-of-concept for detecting roots in sewer videos. However, on their own, they still do not solve the original problem posed. There are thousands of hours of sewer footage to go through and we want automation to help alleviate the load. Simply piping the videos into Detectron2 or YOLO will give us a list of bounding boxes and probability scores associated with them. This could work great for determining around what timestamp in the videos roots pop up. In areas with high probability, it is likely there is a root present. This could easily be mapped to the location of the video. However, it does nothing to give an overall score of how bad the root growth is. Another model could be applied after the roots are detected in the original video. Perhaps this model would give a score for how much the root is blocking the pipe based on how large it is and how much it is covered in debris. This idea could also be extended to cracks and other sewer defects. With many defects loaded into the system, these scores could be used to generate a report of top-priority maintenance jobs. If this is done through a dashboard, it could be interactive and display the most pertinent defect images when you click on the pipe segment.

9 Team Roles

Kyle Bowman

Technical Contribution: Kyle led the team's research efforts in Detectron2 and implemented the code on the Virginia Tech ARC compute cluster. Additionally, Kyle labeled a portion of the data for training/testing data.

Non-Technical Contribution:
Kyle contributed to the tech memos, led the development of the elevator pitch and presented it. Kyle also led the planning of the Tools and Techniques presentation.

Muskaan Narula

Technical Contribution: Muskaan researched MOT for potential applications, labeled data for training, performed data analysis for the root data-set, as well as developed script for maintaining system architecture for application.

Non-Technical Contribution: Muskaan contributed to the tech memos, and participated and led the planning of the Midterm presentation.

Carrie Choi

Technical Contribution: Carrie labeled data for testing, created visualizations for loss data, performed analysis and visualization of model output to help flag pipes appropriately.

Non-Technical Contribution: Carrie contributed to the tech memos, planned and presented Midterm presentation, and will present part of the Final Presentation.

Junlin Huang

Technical Contribution: Junlin led the team's research efforts in YOLOv5 and implemented the training process on Google Colab, and labeled a portion of data for training/testing data.

Non-Technical Contribution:Junlin contributed to the tech memos, participated in planning and presented the Tools and Technique presentation.

10 Future Work

While the object detection models we trained are a promsiing proof of concept, there is substantial room for improvement. For starters, extending the dataset to work with not only roots, but cracks, joint separations, and other defects would be an important step to fully automating the video review process. Additionally, we believe that adding other defects could potentially improve the accuracy for roots by providing images with no roots to the model. One problem we saw was the model detected cracks as roots because they sometimes have a similar structure and color difference that roots have. Updating the model to use multiple classes would allow it to learn on the negatives of a root as well and mitigate this problem.

A closer look at data labeling may also prove helpful to improving the models. We tried our best to be consistent with how we labeled each root by working together to eyeball what looked like the best bounding boxes, but a more strict definition may help. Often times the roots are merged together and it is difficult to define what is one versus two root objects in a single image. Something we were unable to explore was instance segmentation, which highlights the region in the image containing the object, rather than simply drawing a box around it. This could potentially improve precision, however it would likely be harder to implement and label data for.

One potential improvement that would likely be the least amount of work while offering big rewards is data augmentation. This is the process of expanding the overall training dataset by augmenting it with variations of the dataset. For example, one could flip, rotate, scale, or crop the images. This can make different angles you would n

11 Bibliography

[1] Sinha, S. K. (2001). Automated condition assessment of buried sewer pipeline using Computer Vision Techniques. Pipelines 2001. https://doi.org/10.1061/40574(2001)18

[2] Yin, X., Ma, T., Bouferguene, A., Al-Hussein, M. (2021). Automation for Sewer Pipe Assessment: CCTV video interpretation algorithm and Sewer Pipe Video Assessment (SPVA) system development. Automation in Construction, 125, 103622. https://doi.org/10.1016/j.autcon.2021.103622

[3] Rayhana, Rakiba, et al. "Automated Vision Systems for Condition Assessment of Sewer and Water Pipelines." Ieee Transactions on Automation Science and Engineering, vol. 1-18, 2020, pp. 1–18., doi:10.1109/TASE.2020.3022402.

[4] Gupta, A., Anpalagan, A., Guan, L., Khwaja, A. S. (2021). Deep learning for object detection and scene perception in self-driving cars: Survey, Challenges, and open issues. Array, 10, 100057. https://doi.org/10.1016/j.array.2021.100057

[5] C, Deepika. (2020). An Overview of You Only Look Once: Unified, Real-Time Object Detection. International Journal for Research in Applied Science and Engineering Technology. 8. 607-609. 10.22214/ijraset.2020.6098.

[6] Ren, Shaoqing He, Kaiming Girshick, Ross Sun, Jian. (2015). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. IEEE Transactions on Pattern Analysis and Machine Intelligence. 39. 10.1109/TPAMI.2016.2577031.

[7] : Padilla, R.; Passos, W.L.; Dias, T.L.B.; Netto, S.L., da Silva, E.A.B. A Comparative Analysis of Object Detection Metrics with a Companion Open-Source Toolkit. Electronics 2021, 10, 279. https://doi.org/10.3390/ electronics10030279