Automated Sewer Inspection Using Multiclass Convolutional Neural Networks

Abhishank Gaba
Department of Mechatronics Engineering
University of Waterloo
Waterloo, Ontario
agaba@edu.uwaterloo.ca
Ju An Park
Department of Civil and Environmental Engineering
University of Waterloo
Waterloo, Ontario
j246park@edu.uwaterloo.ca

April 24, 2019

Abstract

A deep multi-class convolutional neural network is used to detect critical points in pipes; specifically joints, connections, and manholes. An accuracy of 99.7% is achieved using a dataset composed of 7 pipe videos of 2 concrete storm, 1 PVC storm, 2 concrete sanitary and 2 PVC sanitary sewer pipes. Three hyperparameters are varied: learning rate, batch size, and class weights assess their impact on training. In sum, the high accuracy implies that the model may be overfitted to the given dataset, despite the data augmentations, 5 fold validations, and dropouts used.

1 Introduction

The demand for standardizing inspection of urban linear sewer infrastructure in southern Ontario is quickly increasing due to rapid aging of storm and sanitary sewer pipes in urban areas. The current pipe inspection industry utilizes outdated technology and inefficient processes that cannot keep up with the rapidly increasing demand. Parts of the pipe inspection processes involve a large amount of time, money, and human resources, which introduce the possibility for human error. For example, visual inspection is typically done on CCTV videos of the insides of pipes. In other words, pipe inspectors are required to watch pipe videos and record any anomalies within the pipe. Pipe inspectors are typically tasked with reviewing hundreds of pipe videos within a week, which creates a high likelihood that the inspector will make an error or fail to identify a pipe defect due to fatigue or a lapse in concentration.

The National Association of Sewer Service Companies (NASSCO) sets and regulates industry standards for the assessment and maintenance of underground structures. In it's official Pipe Condition Assessment Using CCTV Performance Specification Guideline, NASSCO clearly states that critical points in pipeline structures, specifically joints, connections, manholes, and other defects, must be carefully and thoroughly inspected by inspectors. This in turn means that current crawlers must drastically slow down at each of these points and perform a full 360 degree rotation [1]. This act is not always performed by inspectors at every critical point.

Shown in Figure 1, is the pipe inspection process currently used by most pipe inspection firms, and consultants.

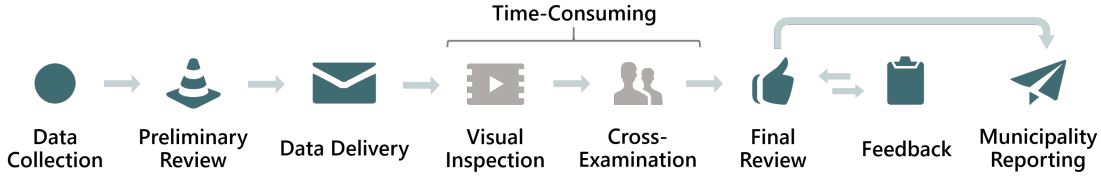

Figure 1: Current Pipe Inspection Process Overview

While revolutionary technologies, such as autonomous drones and artificial intelligence have been implemented in other industries, they have not yet entered into the pipe inspection industry.

Autonomous, waterproof drones are being used for the mapping of underground mines [2]. Edge computing and the introduction of 5G networks have also made it possible to process data close to where it is collected, reducing the latency involved with data transmission and processing. While there is huge value in cloud computing, edge computing has been revolutionary for remote locations (caves, and pipe networks) with limited accessibility and no network.

Traditional machine learning techniques are more likely to be overfitted to their training dataset. The recent emergence of deep learning has made it possible to learn and generalize to any task with a network of neurons. The idea of deep learning is that any task can be broken down into a summation of simple concepts. In terms of computer vision and object detection, this means any picture can be broken down into several recognition phases. The first phase might identify light and dark areas, second would identify the lines, and third would identify shapes. The phases would then be combined to recognize objects in a picture. Furthermore, deep learning algorithms are able to process enormous amounts of data, and achieve a high level of generalization which was impossible before.

While amazing individually, the integration of these technologies allows for the potential to create autonomous drones that can inspect sewer lines from manhole to manhole and identify problem defects. The first part in this process is the detection of high stress areas such as joints, connections, and manholes. The second part deals with performing certain procedures to perform detailed inspection of these areas. This paper implements t-Distributed Stochastic Neighbor Embedding (t-SNE) for data visualization and multiclass Convolutional Neural Networks (CNN) to achieve a high accuracy for the detection of critical points in a pipeline.

2 Background Review

For sewer pipe inspection, there are various literature works in the field of feature extraction technology. The most notable sensors are visual based, such as CCTV and 180 degree/360 degree cameras. Some research has recently been done on using machine intelligence perform the automated detection of pipe defects.

2.1 Using Automatic Anomaly Detection to Identify Faults in Sewers

Using CCTV images as a dataset, Myrans et al. [3] proposes the use of one-class support vector machines (OCSVM) and GIST feature descriptors for binary anomaly detection. The OCSVM uses a hypersphere instead of a linear hyperplane to distinguish anomalies from normal pipe sections. OCSVM is compared with Random Forest (RF) classifiers and Support Vector Machines (SVM).

The dataset consists of a total of approximately 8000 images and a 3-minute continuous CCTV video. The pipe sizes range from 150 mm to 1500 mm. The pipe shapes range from circular to egg to horseshoe. The pipe composition type ranges from vitrified clay to PVC to brick.

The general methodology of this study is split into 4 major stages. The first stage involves pre-processing the images, which includes resizing and converting to grayscale. The second stage focuses on feature extraction, which is done using a GIST descriptor, a type of Gabor filter. A GIST descriptor is used because it is faster than both histogram of oriented gradients (HOG) and scale-invariant feature transform (SIFT). The third stage involves training a OCSVM, RF, and SVM model for binary anomaly detection. The fourth stage applies prediction smoothing, using a Hidden Markov Model, followed by Order Oblivious Filtering with a 1 second buffer. The models are trained using a 25 fold cross validation.

On the still image test set, the OCSVM results in an accuracy of 75%. However these results are worse when compared to traditional support vector machines, which yield 79% on the still image test set. Ultimately, the RF performs the best, achieving an accuracy of 83%. When applied to the sample inspection video, which are 3 minutes long with 6 unique faults at varying locations, the one-class support vector machine yields an accuracy of 85%, while random forests yield an accuracy of 64%, and support vector machines yield an accuracy of 52%.

At the end of the research, the author himself notes that there are a large number of misclassifications that are unique to each classifier (RF, SVM, and OCSVM). To counteract these faults and increase accuracy, he proposes formulating a method of stacking the misclassifications.

While this study is well thought out in its experimental procedure, it leaves questions unanswered. The models are not trained for each unique material type. Different material types have different points of interest in the hyperplane, thus a specific coordinate may be an anomaly for one material type but completely normal for another. However using the current model, both material types would be treated equally, thus leading to misclassification. Another critique of this paper is its lack of feature equalization of sparse anomalies. A simple solution to this problem is to use image transformations to increase their size. Lastly, this study could have used a bagging approach, where a weighted average of outputs from multiple models is taken.

2.2 Automated Defect Detection for Sewer Pipeline Inspection and Condition Assessment

Guo et al. [4] uses a difference-of-images based detection approach to detect critical points: cracks, corrosions, connections, and joints. The underlying assumption is that pipe scenes with anomalies are visually different from a normal piping.

The dataset is extracted from RedZone Robotics, Pittsburgh, Pennsylvania. It contains a total of 103 images of a single 196 feet long stormwater pipe. Out of the 103 images, 51 contain defects.

The general methodology involves registration, histogram equalization, filtering, denoising, and change detection. Registration is performed by determining a suitable reference frame for the target image. Histogram equalization is performed in order to increase contrast by varying the pixel intensities. Change detection is performed through the use of vector subtraction between the image of interest and the reference image. The approach achieves an overall accuracy of 84.5%

2.3 Automated Defect Detection in Sewer Closed Circuit Television Images using Histograms of Oriented Gradients and Support Vector Machine

Halfawy et al. uses a HOG feature input into a SVM classifier to build a binary pipe defect detector, and tests the performance by detecting root intrusions.

The data is composed of root intrusion images from various cities in Regina and Calgary. Number of training images is not specified. The general methodology involves identifying the region of interest, and feeding it into an SVM classifier. This application process has a series of substeps. First, the coloured image is converted to binary values, then a median filter is applied, followed by a morphological closing operation. The next step is to apply a morphological opening operation using a rectangular structuring element. The regions of interests are then converted to HOG features, which are finally fed into the SVM classifier.

The resultant accuracy is 82% for linear SVM and 91% for Radio Basis Function SVM The single critique of this paper is that it fails to mention the accuracy of the ROI detector, which may have a significant impact on the results.

2.4 Evaluation of artificial intelligence tool performance and uncertainty for predicting sewer structural condition

Sousa et al. [6] focuses on quantifying the uncertainty associated with the performance of both artificial neural networks (ANN) and SVM. The data is composed of images taken of a total of 745 sewers in Sanest, near Lisbon. The images mainly consist of PVC, vitrified clay, HDPE pipes, and a few images of concrete pipes.

The methodology uses shallow artificial neural networks in order to train and test the dataset. When using ANN, there is a 68% correlation on training data and a 78% correlation on test data

(with a standard deviation of 5%). When using SVM, a correlation of 67.5% is found for training data, and a 74.8% correlation is found for test data.

Some critiques of this paper is that it only uses 1 hidden layer for the ANN, thus there is a high likelihood that the accuracy would increase if more hidden layers are used. The paper also doesn't mention using any pre-processing techniques.

2.5 Automated defect classification in sewer closed circuit television inspections using deep convolutional neural networks

Kumar et al. [7] uses deep CNNs to detect multiple defects in sewer CCTV images. Shrinath developed a prototype system in order to identify structural defects such as root intrusions, surface cracks, and external deposits.

The dataset consists of 12,000 images from over 200 different pipelines with the diameter being 8 to 10 inches. These material also range from VC, concrete cylinder pipes (PCCP), and ductile iron pipes (DIPs). Data is separated into 8 classes: root intrusions, deposits, cracks, infiltrations, debris, connections, material changes, and undamaged pipe sections. Shrinath et Al. also performs the data augmentation to increase the dataset by a factor of 1000 by adding motion blurs, random flips, brightness changes, and contrast changes.

The paper uses individual CNNs to detect and perform binary classification. Red-Green-Blue channel (RGB) images are used to train the CNNs, using a resolution of 256 by 256. In totality, the CNN is trained using 7500 training images, 2500 validation images, and 2000 testing images. A 5 fold cross validation is used. The model consists of 2 convolutional layers, 2 pooling layers, along with 2 fully connected layers.

The results are measured against 3 parameters: accuracy (86.2%), precision (87.7%), and recall (90.6%). The scores clearly indicate that this paper is a great example of how powerful CNNs can be for scene detection. One critique is that this paper fails to comment on the distribution of the given data. Furthermore, the paper fails to indicate whether testing is done using actual videos. While this neural network works well given a set of images, there is no guarantee that it will perform adequately given real time videos.

2.6 Automated detection of sewer pipe defects in closed-circuit television images using deep learning techniques

Cheng et. al [8] uses a faster Recurrent Convolutional Neural Network (RCNN) in combination with a region proposal network (RPN), in order to detect various abnormalities. The original dataset consists of 1200 images. The data is then augmented to 3000 images.

The general methodology of this paper includes collecting the image, augmenting and annotating the image, designing and training the model, and finally testing and evaluating the model using a MAP metric, where 75% of the data is allocated to training, 10% of the data is allocated to validations, and 15% of the data is allocated to testing.

The resulting accuracy is approximately 83%. Though the results are above par, there are some glaring flaws with this paper. For example, pipe defects are highly irregular in shape, thus using a rectangular bounding box may not be a reasonable implementation for real life scenarios. Rather, pixel based detection given a decent set of data may yield better results in the real world.

2.7 Automated diagnosis of sewer pipe defects based on machine learning approaches

Yang et Al. [9] is novel in his approach of using a combination of BPN, RBM, and SVMs in order to classify defects. The dataset consists of 291 images, where 107 images contain an open joint, 112 contain a crack, 16 contain a broken pipe, and 56 contain a fracture.

The general methodology of this paper is to first acquire the images, then extract features based on entropy, correlation & cluster tendency, and perform pattern recognition. Unfortunately, the first time use of BPN, RBN, and SVMs to classify defects results in an accuracy of 50%. The biggest critique is that it is a paper with a poor result with too small a training dataset.

2.8 Automated defect detection tool for closed circuit television (cctv) inspected sewer pipelines

Hawari et al. [10] solves the problem of detecting cracks, deposits, and gaps using image processing techniques. The dataset contains 640 images of VC pipes, approximately 200mm in size. 5% of the given dataset (32 images) are used to validate the data. The general methodology consists of using image processing techniques to detect cracks. For example, morphological image processing is used to detect cracks, a gabor filter is used to detect settled deposits, and active contour segmentation is used to detect ovality. The paper is able to correctly identify 74% of the total cracks, 53% of the total settled deposits, and 65% of the total displaced joints.

Though this paper uses well established and advanced image processing techniques, they still offer poor generalizability. This paper is published in 2018, its methodology has been well explored.

2.9 NB-CNN: Deep Learning-Based Crack Detection Using Convolutional Neural Network and Naïve Bayes Data Fusion

Chen et al. [11] uses CNN in conjunction with the naïve bayes method to implement crack detection. He aggregates information extracted from each video frame to enhance the performance of the classifier.

The data is extracted from 20 videos of 304 stainless steel rectangular (267mm by 267mm) pipes . The video resolution is 720 by 540 pixels, and 358,740 frames are extracted. 5,326 cracked patches are annotated. Data augmentation, specifically changes in brightness and angles, is then performed. The resultant data is composed of 147,344 cracked and 149,460 non-cracked patches.

The general methodology: a moving rectangular box is used to scan the frame and detect cracks using a CNN. The video's motion is then estimated, and the detected boxes are grouped according to their unique crack type. The categorization of cracks is done using spatial and temporal image

registration. The final step uses Naive Bayes classifier to determine whether the image contains a crack. The paper reports an astounding accuracy of 96.8%.

2.10 Real-Time Defect Detection in Sewer Closed Circuit Television Inspection Videos

Moradi et Al. [12] uses Hidden Markov Models to classify and localize defects. The dataset is a set of 35 videos of Circular Concrete pipes, 610mm in diameter from the City of Laval, Quebec, Canada. The resolution of these videos is 640 by 480 pixels.

The general methodology consists of first gray scaling and resizing the videos to 320 by 280 pixels. Noise reduction is then performed on these videos using gaussian convolution. This research results in an accuracy of 82.5%. While the accuracy score is adequate, the paper is not worth reading. From the literary point of view, the paper has several spelling mistakes, is incoherent and hard to follow.

3 Background

3.1 Framework for Automated CCTV Sewer Inspection

The proposed framework is composed of two parts: 1) the precise identification of pipe features, and 2) the implementation of an inspection algorithm in a robotic crawler to enable automatic manhole to manhole inspection. The scope for this paper deals with implementation and validation of a novel computer vision technique to accurately detect of pipe joints, connections, and manholes. The use of categorical CNNs and its metaparameters: batch size, learning rate, and class weights are explored.

3.2 Convolutional Neural Network

CNNs work by extracting features in each of their layers. The network is made up of 3 parts: convolution, pooling, and flattening.

Convolution extracts features from input and preserves the spatial relationship between the pixels. It learns features by splitting the input data into small squares, and outputting a feature map. The rectified linear (ReLu) function is used after every convolution operation.

Pooling is performed to down sample the data and reduce the dimensionality of each feature map while preserving the most important features. Max pooling is used to take the largest element within a neighborhood window.

Flattening is then used to convert the matrix into a linear array that will be input into the nodes of the neural network.

The add and dense operations are then performed to fully connect the convolutional network to a neural network and then compiling the results.

3.3 t-SNE

t-SNE is a technique used to project high dimensional data into lower dimensions through the use of local relationships between data points. t-SNE is often used to to visualize high dimensional data. This key difference between t-SNE and principal component analysis (PCA) is that gives it the ability to be able to work with non-linear structures [13].

Furthermore, t-SNE, for higher dimensions, uses a gaussian distribution to build the probability distribution that describes the relation between data points. In lower dimensions, t-SNE implements the student t-distribution model. The t-distribution eliminates the curse of dimensionality, where points gather up in one area when they are taken from a higher dimensions to lower ones[13].

t-SNE minimizes the error using the gradient descent technique. It uses a non-convex cost function, thus introducing the risk that it may become trapped in a local minima while trying to find the absolute minimum, however it uses several techniques to lower this inherent risk [13].

4 Results and Discussion

4.1 Dataset Description

Typically, pipes vary widely by material, diameter, age, use, and contain a variety of features and defects. Common materials found today range from concrete, PVC, VC, lead, and cast-iron, while diameter can range from 6-inches to 30 feet. Age of a pipe can vary significantly, anywhere from a few months old to over 100 years old. Most pipeline inspection videos contain joints and manholes, which connect one segment to another and show the end of a pipe where inspectors can enter. Some pipelines have lateral connections, which connect smaller pipes to larger pipes. Storm sewers redirect precipitation to nearby waterways while sanitary sewers transport sewage from buildings to the wastewater treatment plant. There are a variety of pipe characteristics at play that determine the complexity of the problem.

The dataset consists of 7 pipeline inspection videos that inspect a single pipe each, from the Town of Collingwood. The data is 1.5 gigabytes in size. The videos consist of 2 concrete storm, 2 concrete sanitary, 1 PVC storm, and 2 PVC sanitary sewer pipes.

The age of all the pipes are not specified, but are likely less than 40 years old. The diameter of 5 pipes are 1 to 2 feet in diameter, and the 2 pipes around 5 to 10 feet. Figure 2 shows 3 different materials of pipes and their respective features of interest.

Shown in the first row of figure 2, the feature is a joint, which is a point where 2 seperate pipe segments are joined. A joint is circular in shape, and looks like a literal disconnection within the pipeline. It is important to note that the diameter of the pipeline throughout the joint mechanism remains constant. It is visually defined as a dark ring surrounded by light pipe colours. The contrast on either side of the joint is distinct.

In the second row of figure 2, the second feature is a connection, which a small pipeline connected to the circumference of a much larger pipeline. Visually it looks like a small hole punched on the side of a bottle.

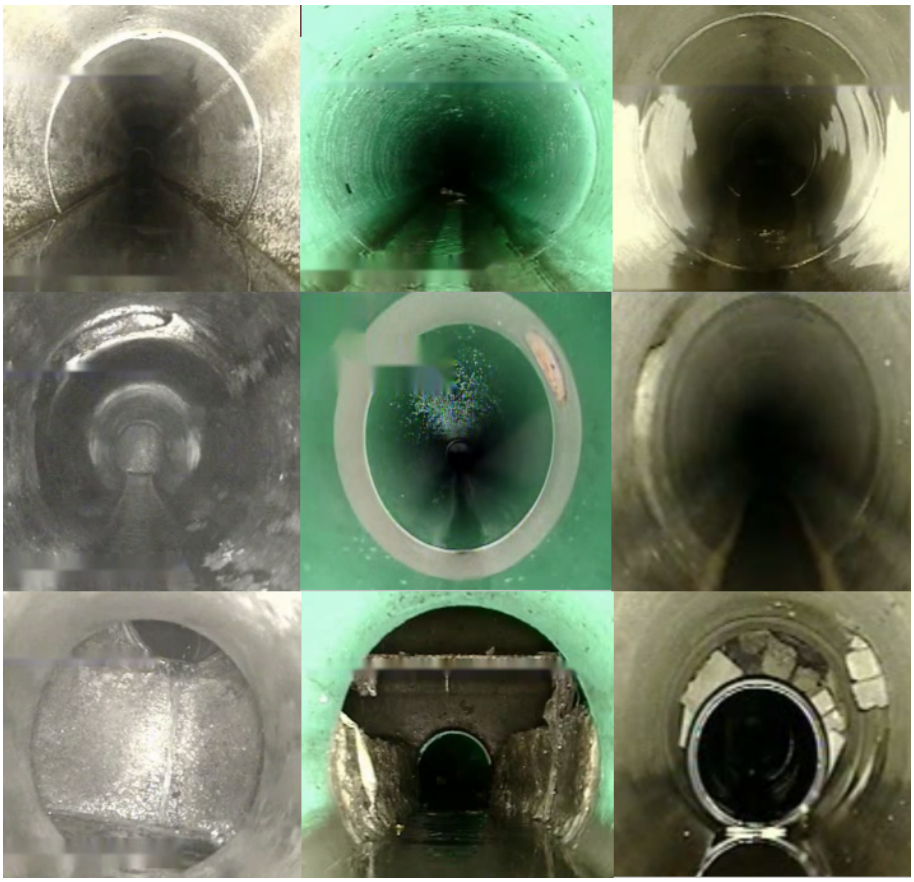

Figure 2: (from left column to right column) Sanitary Concrete, Sanitary PVC, and Storm Concrete Pipe, (from top row to bottom row) Joint, Connection, and Manhole

In the third row of figure 2, the third feature is a manhole, which is a vertical hole connected to the pipe at the beginning and end of a pipeline. A manhole is where humans can enter in order to inspect the pipeline. A manhole too is circular in shape. It is visually distinguishable from a joint as the manhole marks the end of a pipe.

4.2 Pipe Feature Types in Pipelines

Figure 3 shows a typical image of a pipeline. It is important to note that multiple pipe features can be seen in the same image at varying distances.

In order to properly classify these features, clear class definitions are required. As the scene in the video progresses as the drone moves forward, pipe features get closer (larger) and eventually disappear. Just prior to disappearance, the features are classified.

There are times when an operator stops and performs a 360 degree inspection on a joint. In this case, these scenes are considered to be a joint. When an inspector zooms in towards a connection, these scenes are classified as connections. These class definitions ensure that there is only one defect is classified and acted upon despite the presence of multiple stress points in the frame.

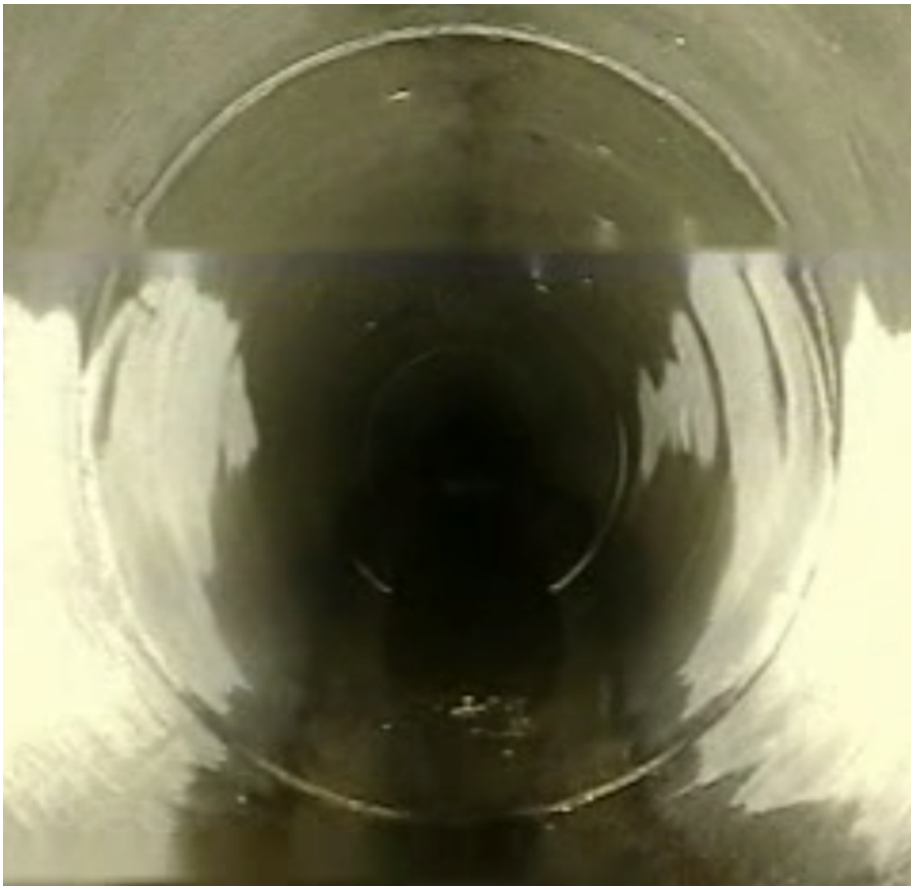

Figure 3: Pipe Scene with 2 Different Joints at Different Locations

4.3 Dataset Visualization, Class Distribution, and Data Augmentation

4.3.1 Dataset Visualization

The total number of extracted frames are 92,105. However, the original dataset contains significant image repetition as a result of the inspector trying to navigate the pipeline as well as rechecking areas. After filtering the repetitive video frames, the new total image count is 46,303. Out of these 46,303 images, 6207 images are annotated as joints, 864 as connections, 666 as manholes.

t-SNE has been used to visualize the high dimensional dataset. From 46,303 images, 5000 are randomly selected using a uniform distribution. The t-SNE analysis is completed using a perplexity of 30, the barnes hut method, with a upper limit of 6000 iterations.

The t-SNE plot shows that there is a lot of overlap between joints and normal pipes, and all the classes are found neighbouring close to one another. There are several distinct individual clusters, mainly manholes and connections. This is also likely due to the effect of sampling from video sequences, where one frame is highly similar to its preceding and following frames. Also, the plot shows that different materials do not overlap with one another and form distinct clusters.

4.3.2 Class Imbalance

Figure 4: 2D Plot of the t-SNE Analysis for the Filtered Dataset. (Images are color coded where blue = Normal Scenes, Green = Connections, Magenta = Manhole, and Red = Joints)

Figure 4 shows that there is a clear data imbalance. The majority are of normal pipes and joints. This distribution is expected because a there is only 1 manholes per pipeline, several connections per video, but joints are present every couple of meters.

Of the entire dataset, 13.4% are joints, 1.9% are connections, and 1.4% are manholes. This very biased distribution towards joints heavily skews the classifier towards the individual accuracy level of detecting joints, and nearly negates the impact of the other accuracies. The dataset can be significantly downsampled while still valid. In summary, to reduce the effects of class imbalance, 2 methods are employed: 1) downsampling of classes with a high number of images, and 2) calcu-

lating a proportionality factor that emphasizes the loss contributed by sparse classes. Joints and normal scenes have been downsampled from 6,207 and 38,566 images to 2000 images each. The downsampled training dataset is 249 megabytes, with a total of 5530 images.

This changes the proportions of the dataset, where normal pipes, connections, joints, and manholes have a proportion of 36.2%, 15.6%, 36.2%, and 12.0% respectively. This drastically improves the class imbalance issue. Afterwards, the proportionality factor, using the largest class as a base, is calculated as follows.

$$Proportionality\,factor:\ p = \%\,of\,normal\,pipes\,\%\,of\,feautres \tag{1}$$

$$p\_normal\_scenes = 36.2\%/36.2\% = 1$$

$$p\_joint = 36.2\%/36.2\% = 1$$

$$p\_connection = 36.2\%/15.6\% = 2.32$$

$$p\_manholes = 36.2\%/12.0\% = 3.02$$

Following downsampling, a revised t-SNE plot is generated to visualize the training set

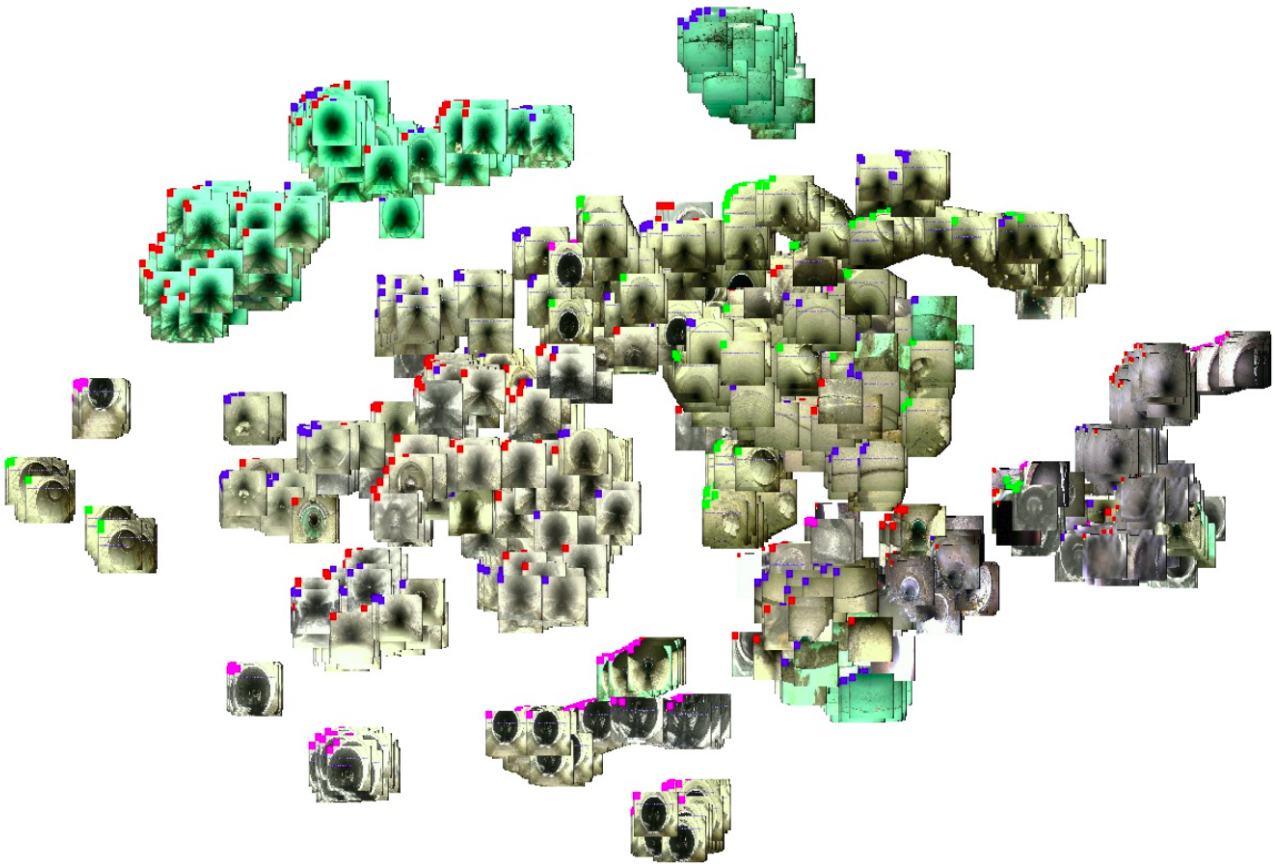

Figure 5: 2D 2D Plot of the t-SNE Analysis for the Downsampled Dataset. (Images are color coded where blue = Normal Scenes, Green = Connections, Magenta = Manhole, and Red = Joints)

While there is still some overlap between clusters due to the nature of time-series visual data, figure 5 shows much more distinct clustering of classes than Figure 4.

4.3.3 Data Augmentation

Since the video maintains a relatively consistent perspective (no significant rotations and skews), data augmentation is done to further generalize the model and prevent overfitting. The idea behind data augmentation is to make minor alterations to the existing dataset. Minor changes such as flips, translations, and rotations causes the neural network to perceive the altered data as completely new images. Therefore, the training dataset is duplicated with random vertical flips and 10 degree rotations.

4.4 Training Results

4.4.1 Network Structure

Looking at the actual implementation, a multiclass model is used to classify the data. This means that defects are classified into one of three classes.

A Keras sequential model is created. The number of filters that the convolutional layer will learn are increased from one layer to another. Layers closer to the input have fewer filters, learning fewer features while layers deeper into the network and closer to the output increase in both filter and feature size.

A padding parameter of 'same' is chosen in order to keep the spatial dimensions of the input volume. An activation function of 'ReLu' is chosen because it is robust, eliminating the issue of vanishing gradient.

Epoch is defined as the number of times the data is iterated, yielding individual accuracies. A higher epoch typically increases the accuracy. However due to the large size of data, 20 epochs are chosen in order to keep the runtime within an hour.

Batch size determines how many samples are processed at a time. A batch size of 16 is used as it takes the first 16 samples and then updates the neural network based on those 16 samples. It then takes the next 16 samples and updates the neural network on that dataset, and so on. A relatively low value used for batch size not only conserves memory and doesn't overload the GPU. This is because the network weights and parameters are updated after every batch. The tradeoff of using a low batch size is that the gradient becomes less accurate and has greater fluctuations when compared to the true gradient.

The decision of using a smaller batch and epoch size is made due to memory and time constraints. Given a longer time frame to optimize this detection model, both parameters would be significantly increased.

The images are set to a resolution of 299x299, as this is sufficiently accurate when compared to the original video dataset.

The data is fit using the sigmoid activation function with the loss model being categorical cross entropy and the metric being accuracy. Several other loss functions, such as minimum squared error, were tried but they resulted in a lower accuracy. Dropout is also implemented to decrease overtraining and generalize the model.

A stochastic gradient descent (SGD) optimizer with a learning rate of 1X10-4, a momentum of 0.9, and decay rate of 1X10-6 is used to train the model. There is a built-in function for SGD in the keras python package. A learning rate of 0.0001 is used. A higher learning rate might lead to a faster convergence, but it would also lead to big jumps between datapoints, thus risking not finding convergence at all. Too low a learning rate would be more consistent in its changes, but it would take too long to drive up the accuracy and drive down the error. The training pseudocode is as follows:

Algorithm 1 Model Training

Require: epoch = N

```
for epoch → 1 to N do
    for each batch ∈ Batches do
        Get predictions
        Generate Loss
        Backpropagate
    end for
end for
return Model
```

In section 4.4.3, Parametric Analyses of Multiclass Classifier, effect of learning rates, batch sizes, and class weights is explored.

4.4.2 Training Method and Results

Since the dataset used to fit the model is large (249 megabytes), k-fold cross-validation, with k equal to 5, is used to split the dataset into training and testing portions. The objective of cross validation is to evaluate the capabilities of a machine learning model by testing it with unseen data[14].

K-fold cross validation uses "k" as a single parameter to split the total dataset into 2 parts. The first part being the training dataset, which the model uses to learn and optimize its weights. The second part is the test set, which is used to evaluate how accurately the model performs on unseen data. This second part simulates real world data which the model has absolutely no knowledge of or experience with [14].

K-fold validation works by performing the following general algorithm. It starts off by randomly shuffling the original dataset. It then splits the shuffled set into k groups. A series of sequential steps are then performed on each group [14].

The (k) group is first labeled as a 'test data' set. The other (k-1) groups are augmented abd labeled as training. The model is then trained on the the augmented training set, and evaluated on the test data set. The resultant score of this mini procedure is recorded, and the model weights are reset to unknown [14]. The resultant scores of each 'k' iterations are averaged to determine the resultant skill of the model.

As previously mentioned, k is chosen to be 5. Several values of k were experimented with. The value of k directly impacts the model's skill estimate, which is defined by the amount of bias and variance in the predictions. In the first iteration, the k is set to a default value of 10 because generally this value results in a low bias and average variance. Though the resultant skill estimate is good, the cross validation takes an unacceptably long time to complete. Several other values are tried, before finally landing on a value of 5 which stikes a good balance between a high enough skill estimate (low bias and low variance) along with fast run time.

The validation accuracies of the 5-fold cross validation, as well as the confusion matrix, accuracy, and validation plot of one of the folds are as follows:

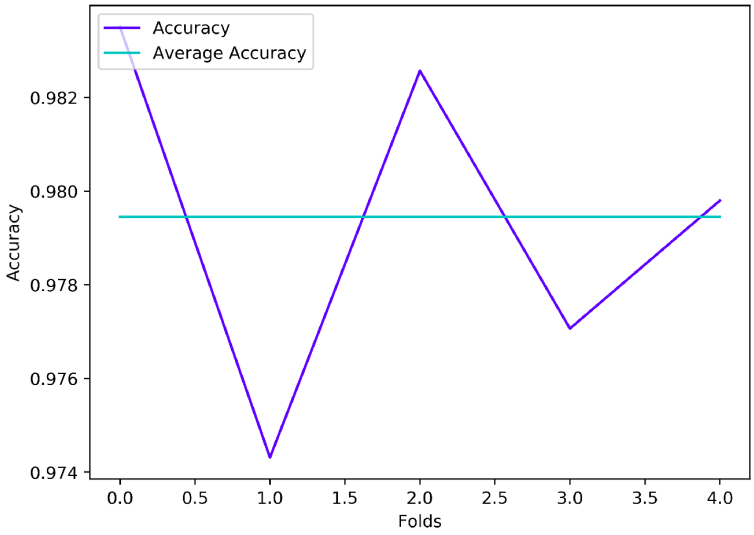

Figure 6: 5-fold Validation Accuracy Results

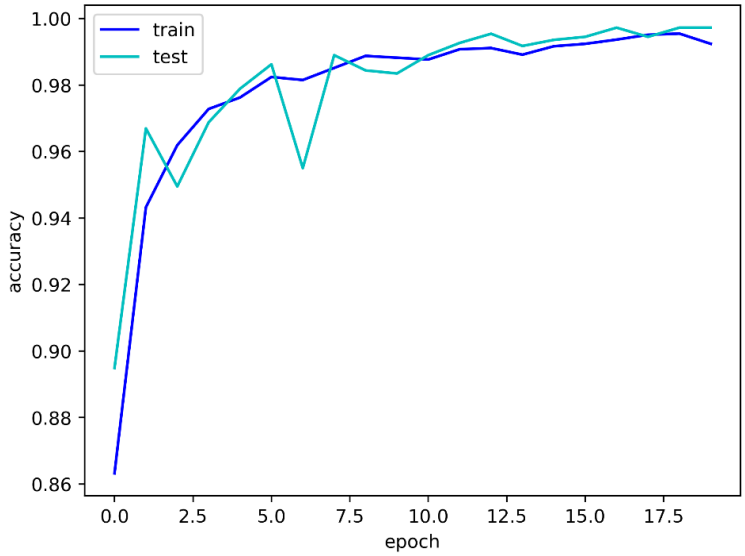

Figure 7: Accuracy Plot

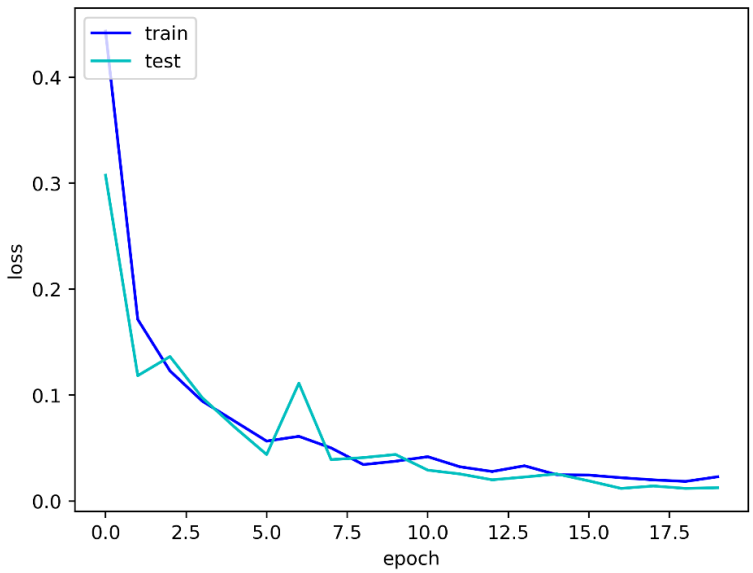

Figure 8: Loss Plot

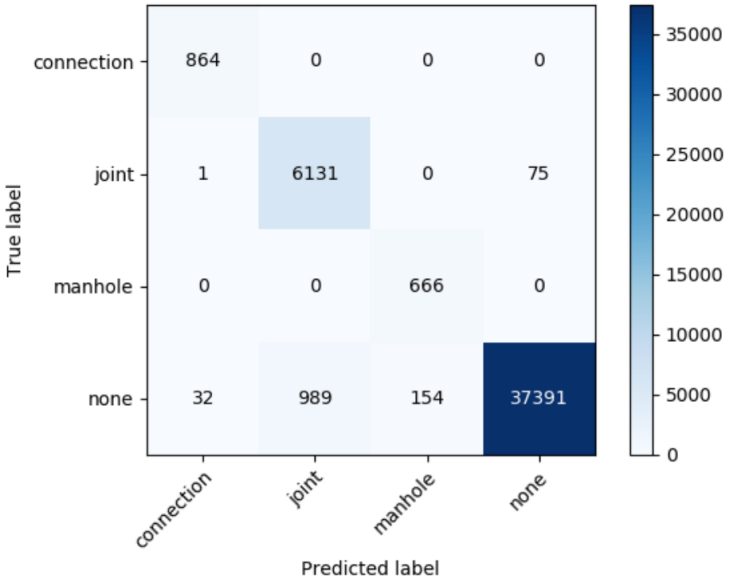

Figure 9: Confusion Matrix

	precision	recall	f1-score	support
connection	0.96	1.00	0.98	864
joint	0.86	0.99	0.92	6207
manhole	0.81	1.00	0.90	666
none	1.00	0.97	0.98	38566
micro avg	0.97	0.97	0.97	46303
macro avg	0.91	0.99	0.95	46303
weighted avg	0.98	0.97	0.97	46303

Figure 10: Classification Report

The accuracy and loss plots show that there is little to no over-fitting, and follows a smooth transition into convergence. The validation accuracy exceeds 99%. The confusion matrix is generated by running predictions on the entire filtered dataset containing 46,303 images. Note, all the following confusion matrices will be generated using the same dataset.

Analyzing the confusion matrix leads to some concerns. Though the overall validation accuracy is 99.7%, the precision of joints is 86%, and the precision for manholes is 81%. It is peculiar that there is a significant difference in values between accuracy and precision. While accuracy is defined by the number of correct classifications/total number of detections, precision is the total number of true positives/the sum of true positives and false positives. This model is over detecting due to the fact that there are many normal images that resemble joints and manholes. Partial defects in the frame, low resolution, and low lighting may contribute to the false positive detections. This may be corrected by increasing the sensitivity of each classifier. However in the grand scheme of things, the number of correct classifications far outweighs the number of false positives, therefore the accuracy still comes out to be high.

4.4.3 Case Study of Misdetections

4.4.4 Learning Rate Parametric Analysis

First the learning rate parameter is altered to the following values: 0.00001, 0.0001, and 0.001, and the resultant accuracy plot and loss plot are shown below.

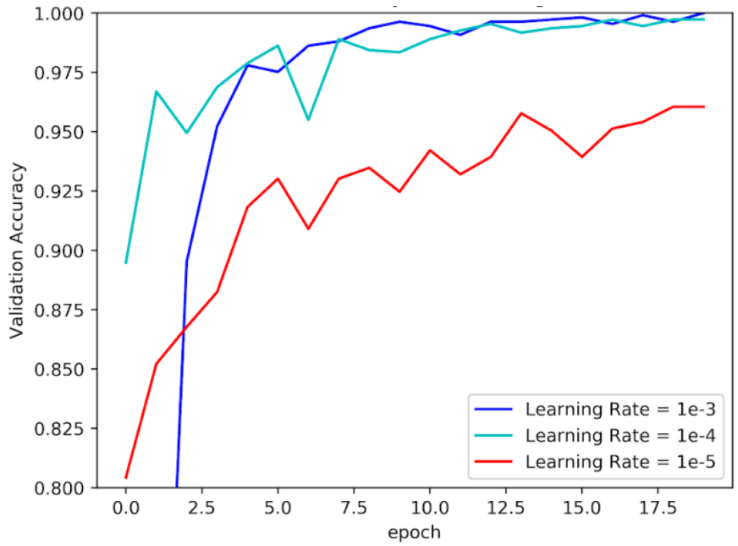

Figure 11: Learning Rate Parameter Analysis Accuracy Plot

The first point to note is the learning rates' impact on overall change. Smaller learning rates make minute changes per epoch iteration, thus their overall impact on driving the accuracy up and driving the error down is quite low. As the learning rate value grows, the changes made per epoch also grow, thus resulting in a greater impact on both accuracy and error. Therefore, the two higher learning rates (0.0001 and 0.001) result in the lowest final error and highest final accuracy.

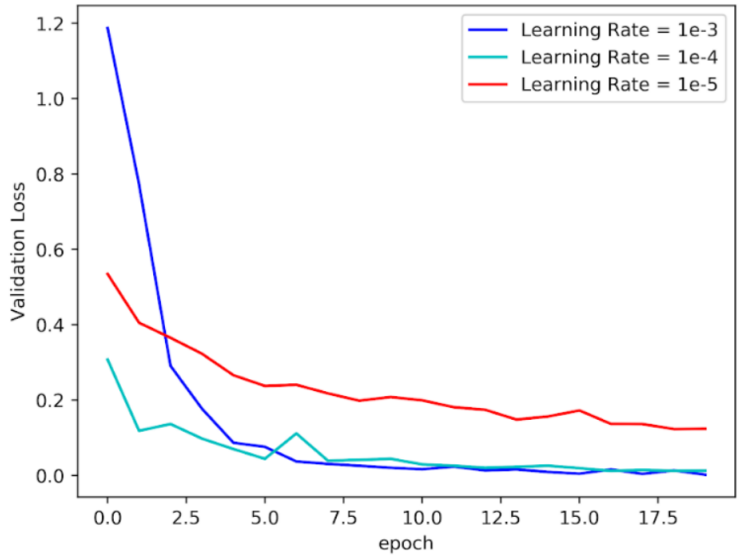

Figure 12: Learning Rate Parameter Analysis Loss Plot

There is no major difference between the resultant accuracies of both 0.001 and 0.0001 because the model is given a large number of epochs to allow the model to converge, therefore both values are adequate for use. However, the highest learning rate value sees the most variation in accuracy. This is fine for the current dataset because the change in error and accuracy is almost zero in the later epochs. If the dataset is increased by a significant amount while keeping the number of epochs constant, the accuracy would not converge, therefore higher learning rates would have a greater likelihood of deviating further from the optimum accuracy.

4.4.5 Batch Size Parametric Analysis

Batch sizes of 2, 8, and 16 are tried. The smallest batch size sees the most variation in accuracy and error due to the frequent weight updates and it being more sensitive to individual data points. These two factors also lead to a lower overall accuracy and higher error.

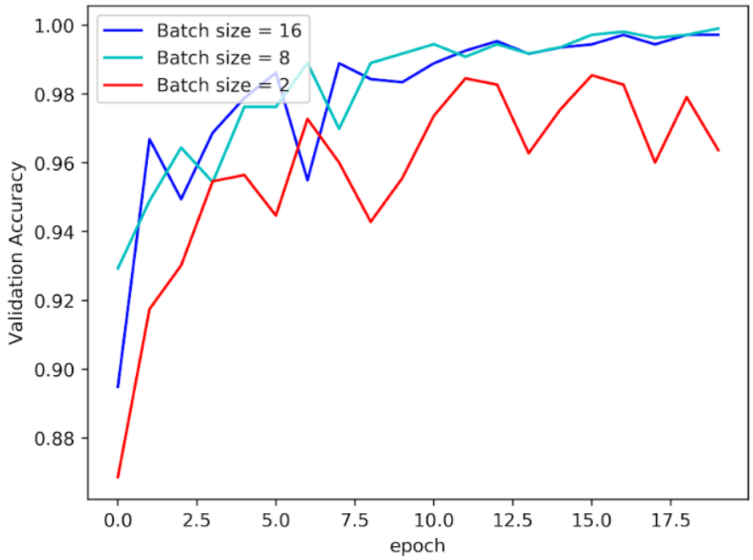

Figure 13: Batch Size Parameter Analysis Accuracy Plot

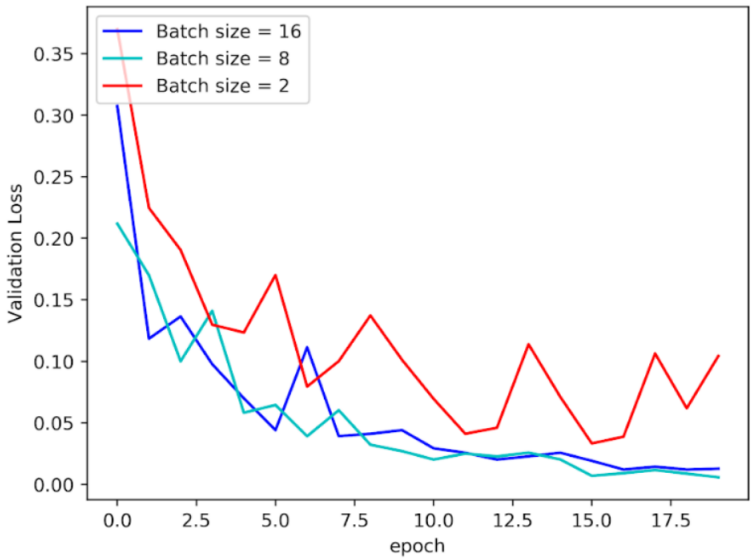

Figure 14: Batch Size Parameter Analysis Loss Plot

Both batch sizes of 8 and 16 lead to similar overall accuracies and errors. Their accuracies also follow a very similar trend. This is most likely because the averaged values of 8 datapoints is very close to the averaged values of 16 data points. Therefore both these sizes would be good choice.

It is interesting to note that a batch size of 16 shows larger fluctuations per epoch when compared to 8. This is probably just due to how this dataset is shuffled per epoch, and shouldn't be used as a hard rule. Taking this fact into account, a batch size of 16 is chosen, because it is the safest bet when minimizing this model's sensitivity to outliers.

4.4.6 Class Weights Parametric Analysis

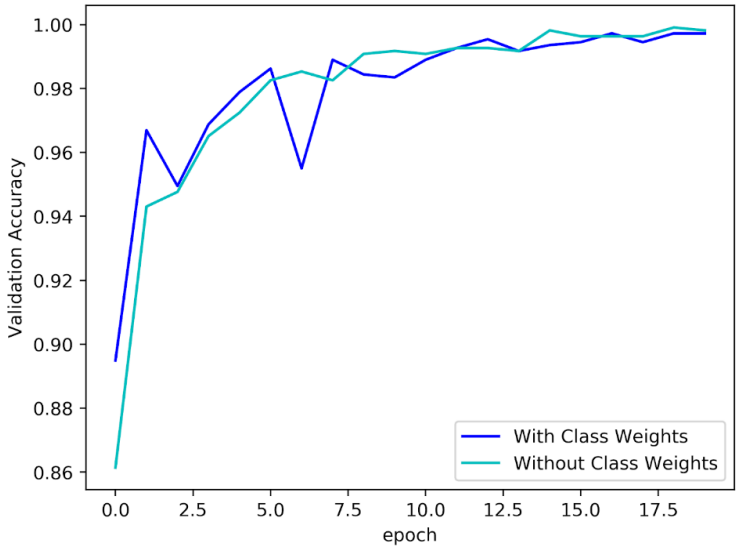

Figure 15: Class Weights Parameter Analysis Accuracy Plot

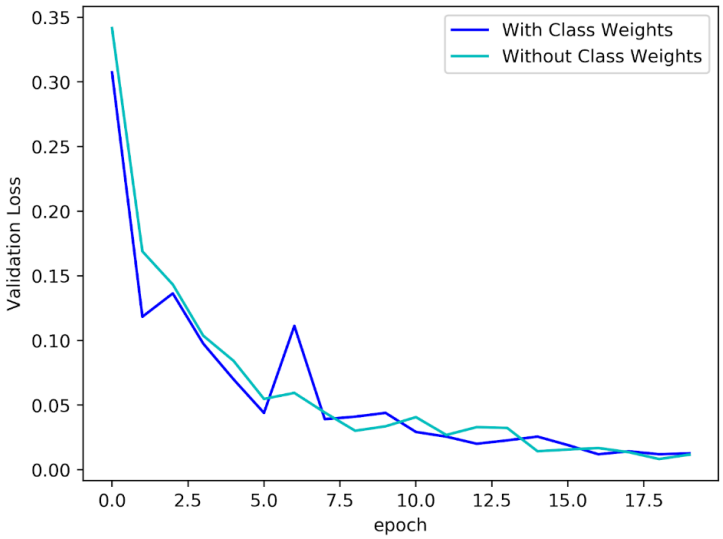

Figure 16: Class Weights Parameter Analysis Loss Plot

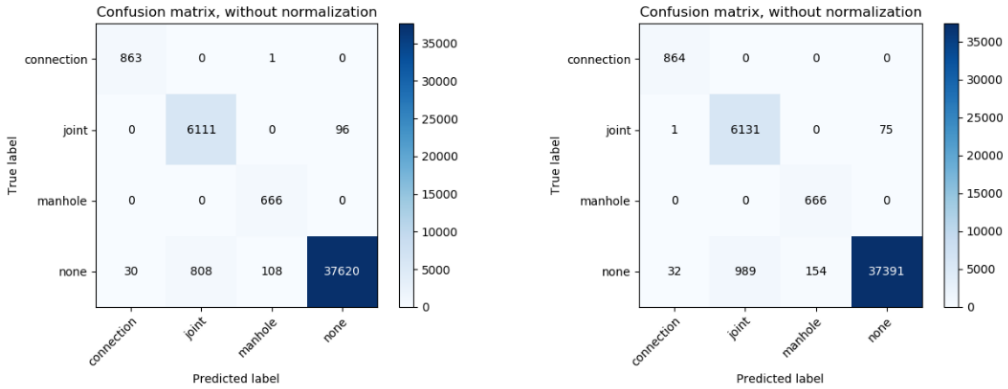

Figure 17: Confusion Matrix (Left) without Class Weights, (Right) with Class Weights

Despite adjusting the weights in order to mitigate the effects of the class imbalance, the class weighting does not seem to make a drastic difference on the resultant accuracy and loss plots, and the confusion matrix. This is likely due to the effect of downsampling the dataset to a point where the effect of class imbalance is small.

When comparing the confusion matrices of both weighted and unweighted classes, almost all the values are identical. One significant change occurs in the manhole row and column, where the number of false positives increase by 30% from unweighted to weighted. False positives are defined as the number of false values that are misclassified as true. This makes sense because by applying a large proportionality factor to manholes, its region size expands, thus increasing the likelihood that a larger number of datapoints are classified as manholes.

It is intriguing that the number of false positives of the joint also increase by 18% from unweighted to weighted even though the weight of the joints did not change (proportionality factor remains 1). It could be that by reducing the power of the normal pipe weight, it increases the likelihood that a certain portion of normal pipes will be misclassified as something else. Since the next heaviest weighting is joints, they find a perfect place to land.

4.5 Comparison of Results to Literature

Although there are no official benchmarks for pipeline defect detection in the field of civil, there are several papers that have revolutionized the use of artificial intelligence in pipelines. A key article, named "Automated defect classification in sewer closed circuit television inspections using deep convolutional neural networks", is mentioned in the literature review above. Shrinath et Al. use of deep convoluted neural networks to detect multiple defects in sewer images taken CCTV results in an accuracy of 86.2

Though, numerically, the model proposed in this paper far exceeds the accuracy of the benchmark, it is very likely that this model is overfitted to the pipe characteristics of this specific dataset. Given more time, additional measures should be taken to achieve a more realistic accuracy. These measures would range from changes in implementation (better data augmentations, more dropouts, higher k fold validations) to a higher variance in test sets (different resolution videos with variance in defect shapes, materials, and sizes).

A forte of this model is that it is trained using frames extracted from real time videos rather than still images (used by the benchmark).

5 Conclusion

Based on a resultant accuracy of 99.7%, CNNs may have the potential to accurately detect critical stress points in pipelines. However this model needs to be tested on hundreds of live pipeline videos to verify this assumption. This research is limited by its variance of two parameters: learning rate and batch size with 3 values per analysis, however if this research proves viable, current model parameters must be expanded and optimized for.

A small part of the work done here has been submitted to ENVE401 as part of Final Year Design Requirements for Ju An Park (20577312)

References

[1] Diego Calderón, T. D. (2014, October). *Pipe Condition Assessment Using CCTV Performance Specification Guideline.* Retrieved from https://www.nassco.org/sites/default/files/SPECIFICATION%20GUIDELINE%20-%20CCTV%2015Dec2014-2018logo.pdf: https://www.nassco.org/sites/default/files/SPECIFICATION%20GUIDELINE%20-%20CCTV%2015Dec2014-2018logo.pdf

[2] Szentpeteri, Krisztian & Reza Setiwan, Tatzky & ismanto, Arief. (2016). Drones (UAVs)in mining and Exploration. An application example: Pit Mapping and Geological Modelling..

[3] Myrans, J. K., Kapelan, Z., & Everson, R. (2018). Using Automatic Anomaly Detection to Identify Faults in Sewers. *1st International WDSA / CCWI 2018 Joint Conference, Kingston, Ontario, Canada* (p. 8). Devon: University of Exeter.

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

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

[6] Sousa, V., Matos, J. P., & Matias, N. (2014). Evaluation of artificial intelligence tool performance and uncertainty for predicting sewer structural condition. *Automation in Construction, 44*, 84-91. doi:10.1016/j.autcon.2014.04.004

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

[8] Cheng, J. C., & Wang, M. (2018). Automated detection of sewer pipe defects in closed-circuit television images using deep learning techniques. *Automation in Construction, 95*, 155-171. doi:10.1016/j.autcon.2018.08.006

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

[10] Hawari, A., Alamin, M., Alkadour, F., Elmasry, M., & Zayed, T. (2018). Automated defect detection tool for closed circuit television (cctv) inspected sewer pipelines. *Automation in Construction, 89*, 99-109. doi:10.1016/j.autcon.2018.01.004

[11] Chen, F., & Jahanshahi, M. R. (2018). NB-CNN: Deep Learning-Based Crack Detection Using Convolutional Neural Network and Naïve Bayes Data Fusion. *IEEE Transactions on Industrial Electronics, 65*(5), 4392-4400. doi:10.1109/tie.2017.2764844

[12] Moradi, S., & Zayed, T. (2017). Real-Time Defect Detection in Sewer Closed Circuit Television Inspection Videos. *Pipelines 2017*. doi:10.1061/9780784480885.027

[13] Keitakurita, A. (2018, September 15). Paper Dissected: "Visualizing Data using t-SNE" Explained. Retrieved April 23, 2019, from http://mlexplained.com/2018/09/14/paper-dissected-visualizing-data-using-t-sne-explained/
[14] A Gentle Introduction to k-fold Cross-Validation. (2018, May 21). Retrieved April 24, 2019, from https://machinelearningmastery.com/k-fold-cross-validation/

Code for SYDE522 Paper

Used python 3.6

Running TSNE on the raw dataset

```
from sklearn.manifold import TSNE
from glob import glob
import numpy as np
import pandas as pd
import cv2
import matplotlib.pyplot as plt
from matplotlib.offsetbox import OffsetImage, AnnotationBbox
import pickle
import ast

def parse_tuple(string):
    try:
        s = ast.literal_eval(str(string))
        if type(s) == tuple:
            return s
        return
    except:
        return

# Subset of images for clustering
n_imgs = 5000
clust_imgs = []
df = pd.read_csv("../labels.csv", index_col=0, dtype={'filename':str, 'tags': str,
'index': int})
df['tags'] = df['tags'].apply(parse_tuple)
image_paths = df.sample(n_imgs)

# Import mlb
with open('mlb.pkl', 'rb') as f:
    mlb = pickle.load(f)

# Image preprocessing
for i in image_paths.filename.values:
    img = plt.imread("../raw_images/" + i)
    img = cv2.resize(img, (100, 100), cv2.INTER_LINEAR).astype('float')
#     img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY).astype('float')
    img = cv2.normalize(img, None, 0.0, 1.0, cv2.NORM_MINMAX)
    img = img.reshape(1, -1)
    clust_imgs.append(img)

# Convert into a Numpy array
img_mat = np.vstack(clust_imgs)
# Number of images, (100pix by 100pix) by 4 bands
print(img_mat.shape)
```

```
# Fit a t-SNE manifold to the subset of images
tsne = TSNE(
    n_components=2,
    init='random',  # pca
    perplexity=30,
    random_state=101,
    method='barnes_hut',
    n_iter=6000,
    verbose=2
).fit_transform(img_mat)

# Plot the subset of images in a two dimensional representation
def imscatter(x, y, images, ax=None, zoom=0.1):
    ax = plt.gca()
    images = [OffsetImage(image, zoom=zoom) for image in images]
    artists = []
    for x0, y0, im0 in zip(x, y, images):
        ab = AnnotationBbox(im0, (x0, y0), xycoords='data', frameon=False)
        artists.append(ax.add_artist(ab))
    ax.update_datalim(np.column_stack([x, y]))
    ax.autoscale()


plt.figure(figsize=(20, 20))
images = [plt.imread("../raw_images/"+image_paths.filename.values[i]) for i in
range(n_imgs)]


for i in range(n_imgs):
    tg = df[df.filename == image_paths.filename.values[i]].tags.values[0]
    if sum(mlb.transform([tg])[0]) == 0:
        images[i] = cv2.rectangle(images[i], (0, 0), (50, 50), (255, 0, 0), -1)
    elif mlb.transform([tg])[0][0] and sum(mlb.transform([tg])[0]) == 1:
        images[i] = cv2.rectangle(images[i], (0, 0), (50, 50), (0, 255, 0), -1)
    elif mlb.transform([tg])[0][1] and sum(mlb.transform([tg])[0]) == 1:
        images[i] = cv2.rectangle(images[i], (0, 0), (50, 50), (0, 0, 255), -1)
    elif mlb.transform([tg])[0][2] and sum(mlb.transform([tg])[0]) == 1:
        images[i] = cv2.rectangle(images[i], (0, 0), (50, 50), (255, 0, 255), -1)
    elif mlb.transform([tg])[0][0] and mlb.transform([tg])[0][1]:
        images[i] = cv2.rectangle(images[i], (0, 0), (50, 50), (0, 0, 255), -1)
    elif mlb.transform([tg])[0][0] and mlb.transform([tg])[0][2]:
        images[i] = cv2.rectangle(images[i], (0, 0), (50, 50), (0, 255, 0), -1)
    else:
        raise ValueError("Error")

images = [cv2.resize(image, (300, 300), cv2.INTER_LINEAR).astype('int64') for image in
images]

imscatter(tsne[0:n_imgs,0], tsne[0:n_imgs,1], images)
plt.savefig("tSNE.jpg")
```

Running TSNE on the filtered dataset

```
from sklearn.manifold import TSNE
from glob import glob
import numpy as np
import pandas as pd
import cv2
import matplotlib.pyplot as plt
from matplotlib.offsetbox import OffsetImage, AnnotationBbox
import pickle
import ast


def parse_tuple(string):
    try:
        s = ast.literal_eval(str(string))
        if type(s) == tuple:
            return s
        return
    except:
        return


# Subset of images for clustering
clust_imgs = []
df = pd.read_csv("../labels.csv", index_col=0, dtype={'filename':str, 'tags': str,
'index': int})
df['tags'] = df['tags'].apply(parse_tuple)
images = glob('../train_val_images_multiclass/*/*')
n_imgs = len(images)
chosen_idx = [np.random.choice(len(images), n_imgs, replace=False)]
types = np.array([image.split('/')[-2] for image in images])[chosen_idx]
images = np.array([image.split('/')[-1] for image in images])
image_paths = images[chosen_idx]
# Import mlb
with open('mlb.pkl', 'rb') as f:
    mlb = pickle.load(f)

# Image preprocessing
for i in image_paths:
    img = plt.imread("../raw_images/" + i)
    img = cv2.resize(img, (100, 100), cv2.INTER_LINEAR).astype('float')
#     img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY).astype('float')
    img = cv2.normalize(img, None, 0.0, 1.0, cv2.NORM_MINMAX)
    img = img.reshape(1, -1)
    clust_imgs.append(img)

# Convert into a Numpy array
img_mat = np.vstack(clust_imgs)
# Number of images, (100pix by 100pix) by 4 bands
print(img_mat.shape)

# Fit a t-SNE manifold to the subset of images
tsne = TSNE(
    n_components=2,
```

```
    init='random',  # pca
    perplexity=30,
    random_state=101,
    method='barnes_hut',
    n_iter=6000,
    verbose=2
).fit_transform(img_mat)

# Plot the subset of images in a two dimensional representation
def imscatter(x, y, images, ax=None, zoom=0.1):
    ax = plt.gca()
    images = [OffsetImage(image, zoom=zoom) for image in images]
    artists = []
    for x0, y0, im0 in zip(x, y, images):
        ab = AnnotationBbox(im0, (x0, y0), xycoords='data', frameon=False)
        artists.append(ax.add_artist(ab))
    ax.update_datalim(np.column_stack([x, y]))
    ax.autoscale()


plt.figure(figsize=(20, 20))
images = [plt.imread("../raw_images/"+image_paths[i]) for i in range(n_imgs)]


for i in range(n_imgs):
    tg = df[df.filename == image_paths[i]].tags.values[0]
    if types[i] == 'none':  # none
        images[i] = cv2.rectangle(images[i], (0, 0), (50, 50), (255, 0, 0), -1)
    elif types[i] == 'connect':  # connection
        images[i] = cv2.rectangle(images[i], (0, 0), (50, 50), (0, 255, 0), -1)
    elif types[i] == 'joint':  # joint
        images[i] = cv2.rectangle(images[i], (0, 0), (50, 50), (0, 0, 255), -1)
    elif types[i] == 'manhole':  # manhole
        images[i] = cv2.rectangle(images[i], (0, 0), (50, 50), (255, 0, 255), -1)
    else:
        raise ValueError("Error")

images = [cv2.resize(image, (300, 300), cv2.INTER_LINEAR).astype('int64') for image in
images]

imscatter(tsne[0:n_imgs, 0], tsne[0:n_imgs, 1], images)
plt.savefig("tSNE.jpg")
```

Training model for parametrization analysis

```
from keras import applications
from keras.applications.xception import preprocess_input
from keras.layers import Flatten, Dense, GlobalAveragePooling2D
from keras.models import Sequential, Model, load_model
from keras.preprocessing.image import ImageDataGenerator, img_to_array, load_img
from keras.optimizers import SGD, RMSprop
from keras.callbacks import ModelCheckpoint
from sklearn.metrics import classification_report, confusion_matrix
```

```
import pickle
import numpy as np
import matplotlib.pyplot as plt
import collections
import os

# parameters
num_classes = 4
postfix = 'BS2'
epochs = 20
batch_size = 2
img_width, img_height = 299, 299

# read the file names of training and validating images
data_dir = '../train_val_images_multiclass'
validation_split = 0.2

top_weights_path_check = os.path.join(os.getcwd(), '{}.h5'.format(postfix))

# build the Xception network
base_model = applications.Xception(input_shape=(img_width, img_height, 3),
weights='imagenet',
                                   include_top=False)  # Top Model Block
x = Flatten(name='flatten')(base_model.output)
# x = Dense(1024, activation='relu', name='fc1')(x)
# x = Dense(1024, activation='relu', name='fc2')(x)
predictions = Dense(num_classes, activation='softmax', name='predictions')(x)
new_model = Model(base_model.input, predictions)

# first: train only the top layers (which were randomly initialized)
# i.e. freeze all layers of the based model that is already pre-trained.
# for layer in base_model.layers:
#    layer.trainable = False

new_model.summary()


def get_class_weights(y, smooth_factor=0):
    counter = collections.Counter(y)

    if smooth_factor > 0:
        p = max(counter.values()) * smooth_factor
        for k in counter.keys():
            counter[k] += p

    majority = max(counter.values())

    return {cls: float(majority) / count for cls, count in counter.items()}


# prepare data augmentation configuration
# https://stackoverflow.com/questions/42443936/keras-split-train-test-set-when-using-
imagedatagenerator
```

```
train_datagen = ImageDataGenerator(
    preprocessing_function=preprocess_input,
    horizontal_flip=True,
    rotation_range=10,
    validation_split=validation_split)

train_generator = train_datagen.flow_from_directory(
    data_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='categorical',  shuffle=True)

validation_generator = train_datagen.flow_from_directory(
    data_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='categorical', shuffle=False,
    subset='validation')

sgd = SGD(lr=0.0001, decay=1e-6, momentum=0.9, nesterov=True)
new_model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])

callbacks_list = [ModelCheckpoint(top_weights_path_check, monitor='val_acc', verbose=1,
save_best_only=True)]

class_weight = get_class_weights(train_generator.classes)

H = new_model.fit_generator(
    train_generator,
    epochs=epochs,
    # class_weight=class_weight,
    validation_data=validation_generator,
    callbacks=callbacks_list,
    verbose=1,
    steps_per_epoch=train_generator.samples // batch_size,
    validation_steps=validation_generator.samples // batch_size
)

new_model.save(top_weights_path_check)

with open('train_hist_{}.pickle'.format(postfix), 'wb') as file_pi:
    pickle.dump(H.history, file_pi)

# Confution Matrix and Classification Report
print(H.history.keys())
# summarize history for accuracy
plt.figure(dpi=400)
plt.plot(H.history['acc'],'b')
plt.plot(H.history['val_acc'],'c')
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
```

```
plt.savefig('accuracy_{}'.format(postfix))
plt.show()
# summarize history for loss
plt.figure(dpi=400)
plt.plot(H.history['loss'],'b')
plt.plot(H.history['val_loss'],'c')
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.savefig('loss{}'.format(postfix))
plt.show()
```

Training model using K-Fold Validation

```
from keras import applications
from keras.applications.xception import preprocess_input
from keras.layers import Flatten, Dense, GlobalAveragePooling2D
from keras.models import Sequential, Model, load_model
from keras.preprocessing.image import ImageDataGenerator, img_to_array, load_img
from keras.optimizers import SGD, RMSprop
from keras.callbacks import ModelCheckpoint
from sklearn.metrics import classification_report, confusion_matrix
import pickle
import numpy as np
import matplotlib.pyplot as plt
import collections
import os
from glob import glob
import numpy as np
import os
import pandas as pd
import shutil
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix


def train_model(i):
    # parameters
    num_classes = 4
    postfix = 'KFold_{}'.format(i)
    epochs = 20
    batch_size = 16
    img_width, img_height = 299, 299

    # read the file names of training and validating images
    train_data_dir = 'data/training'
    test_data_dir = 'data/test'

    top_weights_path_check = os.path.join(os.getcwd(), '{}.h5'.format(postfix))

    # build the Xception network
```

```
    base_model = applications.Xception(input_shape=(img_width, img_height, 3),
weights='imagenet',
                                       include_top=False)  # Top Model Block
    x = Flatten(name='flatten')(base_model.output)
    predictions = Dense(num_classes, activation='softmax', name='predictions')(x)
    new_model = Model(base_model.input, predictions)

    # first: train only the top layers (which were randomly initialized)
    # i.e. freeze all layers of the based model that is already pre-trained.
    # for layer in base_model.layers:
    #    layer.trainable = False

    new_model.summary()

    def get_class_weights(y, smooth_factor=0):
        counter = collections.Counter(y)

        if smooth_factor > 0:
            p = max(counter.values()) * smooth_factor
            for k in counter.keys():
                counter[k] += p

        majority = max(counter.values())

        return {cls: float(majority) / count for cls, count in counter.items()}

    # prepare data augmentation configuration
    # https://stackoverflow.com/questions/42443936/keras-split-train-test-set-when-using-
imagedatagenerator
    train_datagen = ImageDataGenerator(
        preprocessing_function=preprocess_input,
        horizontal_flip=True,
        rotation_range=10)

    train_generator = train_datagen.flow_from_directory(
        train_data_dir,
        target_size=(img_height, img_width),
        batch_size=batch_size,
        class_mode='categorical', shuffle=True)

    validation_generator = train_datagen.flow_from_directory(
        test_data_dir,
        target_size=(img_height, img_width),
        batch_size=batch_size,
        class_mode='categorical', shuffle=False)

    sgd = SGD(lr=0.0001, decay=1e-6, momentum=0.9, nesterov=True)
    new_model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=
['accuracy'])

    callbacks_list = [ModelCheckpoint(top_weights_path_check, monitor='val_acc',
verbose=1, save_best_only=True)]
```

```
    class_weight = get_class_weights(train_generator.classes)

    H = new_model.fit_generator(
        train_generator,
        epochs=epochs,
        class_weight=class_weight,
        validation_data=validation_generator,
        callbacks=callbacks_list,
        verbose=1,
        steps_per_epoch=train_generator.samples // batch_size,
        validation_steps=validation_generator.samples // batch_size
    )

    new_model.save(top_weights_path_check)

    with open('train_hist_{}.pickle'.format(postfix), 'wb') as file_pi:
        pickle.dump(H.history, file_pi)

    # Confution Matrix and Classification Report
    print(H.history.keys())
    # summarize history for accuracy
    plt.figure(dpi=400)
    plt.plot(H.history['acc'], 'b')
    plt.plot(H.history['val_acc'], 'c')
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.savefig('accuracy_{}'.format(postfix))
    plt.show()
    # summarize history for loss
    plt.figure(dpi=400)
    plt.plot(H.history['loss'], 'b')
    plt.plot(H.history['val_loss'], 'c')
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.savefig('loss{}'.format(postfix))
    plt.show()
    return H.history['val_acc'][-1], H.history['val_loss'][-1]

# used to copy files according to each fold
def copy_images(df, directory):
    destination_directory = "data/" + directory
    print("copying {} files to {}...".format(directory, destination_directory))

    # remove all files from previous fold
    if os.path.exists(destination_directory):
        shutil.rmtree(destination_directory)

    # create folder for files from this fold
    if not os.path.exists(destination_directory):
```

```
        os.makedirs(destination_directory)

    # create subfolders for each class
    for c in set(list(df['class'])):
        if not os.path.exists(destination_directory + '/' + c):
            os.makedirs(destination_directory + '/' + c)

    # copy files for this fold from a directory holding all the files
    for i, row in df.iterrows():
        try:
            # this is the path to all of your images kept together in a separate folder
            path_from = "{}"
            path_to = "{}/{}".format(destination_directory, row['class'])

            # move from folder keeping all files to training, test, or validation folder
(the "directory" argument)
            shutil.copy(path_from.format(row['filename']), path_to)
        except Exception:
            print("Error when copying {}:".format(row['filename']))


# dataframe containing the filenames of the images (e.g., GUID filenames) and the classes
pths = glob('../train_val_images_multiclass/*/*')
# pths = np.array(glob('../train_val_images_multiclass/*/*'))
# pths = list(pths[np.random.choice(len(pths), 500, replace=False)])
df_y = pd.DataFrame([pth.split('/')[-2] for pth in pths], columns=['class'])
df_x = pd.DataFrame(pths, columns=['filename'])

n_folds = 5
skf = StratifiedKFold(n_splits=n_folds)
total_actual = []
total_predicted = []
total_val_accuracy = []
total_val_loss = []
total_test_accuracy = []

for i, (train_index, test_index) in enumerate(skf.split(df_x, df_y)):
    x_train, x_test = df_x.iloc[train_index], df_x.iloc[test_index]
    y_train, y_test = df_y.iloc[train_index], df_y.iloc[test_index]

    train = pd.concat([x_train, y_train], axis=1)
    test = pd.concat([x_test, y_test], axis = 1)

    # copy the images according to the fold
    copy_images(train, 'training')
    copy_images(test, 'test')

    print('**** Running fold '+ str(i))

    # here you call a function to create and train your model, returning validation
accuracy and validation loss
    val_accuracy, val_loss = train_model(i)
```

```
    # append validation accuracy and loss for average calculation later on
    total_val_accuracy.append(val_accuracy)
    total_val_loss.append(val_loss)

# summarize history for kfold
plt.figure(dpi=400)
plt.plot(total_val_accuracy, 'b')
plt.plot([sum(total_val_accuracy)/len(total_val_accuracy)]*len(total_val_accuracy), 'c')
plt.title('{}-Fold Validation Accuracy'.format(n_folds))
plt.ylabel('Accuracy')
plt.xlabel('Folds')
plt.legend(['Accuracy', 'Average Accuracy'], loc='upper left')
plt.savefig('{}_folds'.format(n_folds))
plt.show()

print(classification_report(total_actual, total_predicted))
print(confusion_matrix(total_actual, total_predicted))
print("Validation accuracy on each fold:")
print(total_val_accuracy)
print("Mean validation accuracy: {}%".format(np.mean(total_val_accuracy) * 100))

print("Validation loss on each fold:")
print(total_val_loss)
print("Mean validation loss: {}".format(np.mean(total_val_loss)))

print("Test accuracy on each fold:")
print(total_test_accuracy)
print("Mean test accuracy: {}%".format(np.mean(total_test_accuracy) * 100))
```

Running model on the filtered dataset to generate predictions

```
from glob import glob
import cv2, os, shutil
import numpy as np
import pandas as pd
from keras.models import load_model
from keras.applications.xception import preprocess_input
from tqdm import tqdm
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.utils.multiclass import unique_labels
from copy import deepcopy
import matplotlib.pyplot as plt
import json

def plot_confusion_matrix(y_true, y_pred, classes,
                          normalize=False,
                          title=None,
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
```

```
    if not title:
        if normalize:
            title = 'Normalized confusion matrix'
        else:
            title = 'Confusion matrix, without normalization'

    # Compute confusion matrix
    cm = confusion_matrix(y_true, y_pred)
    # Only use the labels that appear in the data
    classes = classes[unique_labels(y_true, y_pred)]
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    fig, ax = plt.subplots()
    im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
    ax.figure.colorbar(im, ax=ax)
    # We want to show all ticks...
    ax.set(xticks=np.arange(cm.shape[1]),
           yticks=np.arange(cm.shape[0]),
           # ... and label them with the respective list entries
           xticklabels=classes, yticklabels=classes,
           title=title,
           ylabel='True label',
           xlabel='Predicted label')

    # Rotate the tick labels and set their alignment.
    plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
             rotation_mode="anchor")

    # Loop over data dimensions and create text annotations.
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, format(cm[i, j], fmt),
                    ha="center", va="center",
                    color="white" if cm[i, j] > thresh else "black")
    fig.tight_layout()
    if normalize:
        plt.savefig('confusion_matrix_normalized')
    else:
        plt.savefig('confusion_matrix')
    return ax

# Retrieve data (Videos) list
images = glob('../train_val_images/*/*')
# video_idx = round(random.random() * len(videos))
# print("Video: {}".format(videos[int(video_idx)]))
```

```
# Model Parameters
top_weights_path_check = os.path.join(os.getcwd(), 'BS2.h5')
xception_height = 299
xception_width = 299

d = {'connect': 0, 'joint': 1, 'manhole': 2, 'none': 3}
target_names = ['connection', 'joint', 'manhole', 'none']

# Load Model
new_model = load_model(top_weights_path_check)

result = []
# images = np.random.choice(images, 1000)

# Run through and predict each video
for image in tqdm(images):
    # Video parameters
    tag = image.split('/')[-2]


    # generate predictions
    img = plt.imread(image)
    # Pre-process image
    tmp = cv2.resize(img, dsize=(xception_height, xception_width),
interpolation=cv2.INTER_LINEAR).astype(np.float64)
    tmp = np.expand_dims(tmp, axis=0)
    tmp = preprocess_input(tmp)
    # Predict and store results
    prediction = np.argmax(new_model.predict(tmp))
    # Based on prediction, overlay green circle (non-joint) or red circle (joint)
    if prediction == 0:
        result.append([image.split('/')[-1], 0, d[tag]])
    elif prediction == 1:
        result.append([image.split('/')[-1], 1, d[tag]])
    elif prediction == 2:
        result.append([image.split('/')[-1], 2, d[tag]])
    elif prediction == 3:
        result.append([image.split('/')[-1], 3, d[tag]])
    else:
        raise ValueError("Something's gone wrong")

# Output predictions to csv and images
predictions = pd.DataFrame(result, columns=['filename', 'predicted', 'actual'])
predictions.to_csv('predictions.csv')
cr = classification_report(predictions.actual.values, predictions.predicted.values,
target_names=target_names)
with open('classification_report', 'w') as f:
    json.dump(cr, f)

print(cr)

# Plot non-normalized confusion matrix
```

```
plot_confusion_matrix(predictions.actual.values, predictions.predicted.values,
classes=np.array(target_names), title='Confusion matrix, without normalization')
plot_confusion_matrix(predictions.actual.values, predictions.predicted.values,
classes=np.array(target_names), title='Confusion matrix, without normalization',
normalize=True)
```

Moving Classes into their respective folders

```
import pandas as pd
import os
from sklearn.preprocessing import MultiLabelBinarizer
import ast
from shutil import copyfile
import pickle
def parse_tuple(string):
    try:
        s = ast.literal_eval(str(string))
        if type(s) == tuple:
            return s
        return
    except:
        return

# Read data
df = pd.read_csv("../labels.csv", index_col=0, dtype={'filename':str, 'tags': str,
'index': int})
df['tags'] = df['tags'].apply(parse_tuple)

mlb = MultiLabelBinarizer()
mlb.fit(df['tags'].values.tolist())

with open('mlb.pkl', 'wb') as f:
    pickle.dump(mlb, f)

if not os.path.isdir('../train_val_images/{}'.format('none')):
    os.mkdir('../train_val_images/{}'.format('none'))
for c in mlb.classes_:
    if not os.path.isdir('../train_val_images/{}'.format(c)):
        os.mkdir('../train_val_images/{}'.format(c))

# Start moving images
for pth, l in zip(df['filename'].values, df['tags'].values):
    if sum(mlb.transform([l])[0]) == 0:
        copyfile("../raw_images/{}".format(pth),
"../train_val_images/none/{}".format(pth))
    elif mlb.transform([l])[0][0] and sum(mlb.transform([l])[0]) == 1:
        copyfile("../raw_images/{}".format(pth),
"../train_val_images/{}/{}".format(mlb.classes_[0], pth))
    elif mlb.transform([l])[0][1] and sum(mlb.transform([l])[0]) == 1:
        copyfile("../raw_images/{}".format(pth),
"../train_val_images/{}/{}".format(mlb.classes_[1], pth))
    elif mlb.transform([l])[0][2] and sum(mlb.transform([l])[0]) == 1:
```

```
        copyfile("../raw_images/{}".format(pth),
"../train_val_images/{}/{}".format(mlb.classes_[2], pth))
    elif mlb.transform([l])[0][0] and mlb.transform([l])[0][1]:
        copyfile("../raw_images/{}".format(pth),
"../train_val_images/{}/{}".format(mlb.classes_[1], pth))
    elif mlb.transform([l])[0][0] and mlb.transform([l])[0][2]:
        copyfile("../raw_images/{}".format(pth),
"../train_val_images/{}/{}".format(mlb.classes_[0], pth))
    else:
        raise ValueError("Error")
```

Filtering the Normal Scenes Data

```
from glob import glob
from shutil import move
import numpy as np

n_imgs = 32000
files = glob("../train_val_images_connect/none/*")
chosen_idx = [np.random.choice(len(files), n_imgs, replace=False)]
files = np.array(files)[chosen_idx]

for f in files:
    move(f, "../train_val_images_connect_extra/none/")
```

Downsizing Dataset

```
from glob import glob
from os import remove
import numpy as np

n_imgs = 1000
files = glob("../train_val_images_multiclass/joint/*")
chosen_idx = [np.random.choice(len(files), n_imgs, replace=False)]
files = np.array(files)[chosen_idx]

for f in files:
    remove(f)
```