Infrastructure Surface Crack Detection

Fall 2022 – CS 7641 – Group Project Final Report
Team 34: Botao Li, Inshira Seshie, Kaiqin Bian, Lawrence Bradley, Qinghao Zeng
GitHub: https://github.com/Anderson-Zeng/CS-7641

1. Introduction/Background:

Defect detection constitutes a significant factor in environment and build maintenance, and it is a key process of Structural Health Monitoring (SHM) for infrastructures including buildings, bridges, and roads. Surface cracks, one of the most frequent defects, are superficial line-shaped damages, which in most cases initially appear as a minor gap on the surface. Without timely intervention and precaution, its size and severity could possibly extend beyond a superficial scope, causing grave structural damage (Zou, et al., 2019). This form of decay is preventable through the use of real-time crack detection and monitoring, which can ensure a development's structural health and improve its resilience against natural disasters.

Traditionally, crack detection is conducted by manual inspection by humans deployed to the site, which can be both time-consuming and labor-intensive (Munawar, et al., 2021). As a result, Computer Vision (CV) and Machine Learning (ML) algorithms have the potential to be utilized for assisting in improving crack detection accuracy and efficiency, thereby circumventing the need for human inspection. In this project, a combination of these approaches will be employed to design an automated surface crack detection model which will make use of an image dataset showing cracked and uncracked concrete material surfaces. The overall goal of this process is to not only detect these defects but also to evaluate their severity.

2. Problem definition:

Manual inspection is an inefficient use of time and manpower, and the quality and accuracy of inspection cannot be guaranteed. Our project aims at developing a new model to detect surface cracks and evaluate their severity based on deep neural networks (DNN). This application of CV-based strategies will replace manual crack detection methods with more effective and automated inspection procedures.

3. Data collection:

The main dataset we used is "Structural Defects Network 2018" from Kaggle consisting of 56,000 (256 x 256) images of cracked and non-cracked surfaces. We formatted each data point into a [N x 2] array: The first column is an array of (256 x 256) pixels forming the image, and the second column contains an array containing the Ground Truth of the 0 (negative) or 1 (positive). Using Numpy, Pandas, and OS, we extracted the dataset by the file path provided by Kaggle. These images have been grouped into 3 structure types: Decks, Pavements, and Walls with Cracked or Non-Cracked labels. Because our data comes from Kaggle, a reliable source for machine learning data sets, we determined that data cleaning would be unnecessary.

4. Methods:

We used supervised learning on our labeled dataset for crack detection on concrete surface images and unsupervised learning on datasets of cracked images to evaluate crack severity. For supervised

learning, we used PyTorch's neural net module and created a simple classification model. Originally, we based this model on image features obtained by performing PCA on the grayscale pixel information of resized 128x128 images. Later we found this approach to not be effective enough, so we pivoted to using Canny's edge detection function to create a 64x64 matrix of binary values indicating the presence or absence of edges and used these as our model inputs. For unsupervised learning, we used Gaussian Mixture Model (GMM) and Density-based spatial clustering of applications with noise (DBSCAN). The complete procedures are demonstrated below as 5 steps. Additional details of the methods used in each step are displayed in the sections below.

Step 1 - Visualize dataset and data information for pre-processing
Step 2 - Data pre-processing
    Scale down image size
    Apply edge detection
    Balance the numbers of data (cracked and uncracked images)
    Apply PCA to extract new features (removed, ineffective for our project)
    Split data into training set and validation set for training preparation
Step 3 - Model set up for supervised learning
    PyTorch's neural net module
    Classification model
Step 4 - Model set up for unsupervised learning
    GMM
    DBSCAN
Step 5 - Results and analysis for both models

4.1 Dataset Visualization

Following the manual inspection, we determined that our main dataset "Structural Defects Network 2018" has been cleaned enough, as there were no incorrect, corrupted, incorrectly formatted images, or incomplete data shown in the dataset. Thus, the data cleaning will not be processed here. For a better understanding of our dataset, we visualized our data before and after preprocessing. The distribution of non-cracked (Neg) and cracked (Pos) images in original datasets is shown in Figure 1.

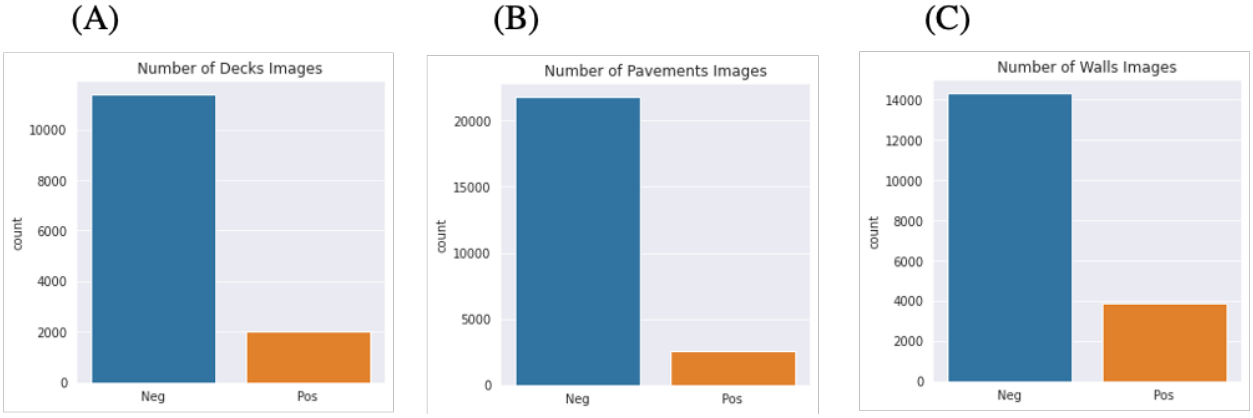


Figure 1. Distribution of non-cracked (Neg) and cracked (Pos) images in original datasets ((A) dataset_Decks, (B) dataset_Pavements, (C) dataset_Walls).

4.2 Dataset Preprocessing

As great unbalanced data distribution and low-noises dataset was observed, we chose to implement some functions to introduce noise to our data. Therefore, we could artificially extend the size of our dataset and balance the distribution of positive and negative data. Also, we can make our model more robust at dealing with noise by obscuring the true data. Data preprocessing on the raw dataset included (1) scaling images down by converting the images into gray scale and resizing them to have less pixels (256 x 256 to 64 x 64), (2) balancing numbers of each class (flip and rotate cracked images to increase dataset size), (3) resizing datasets and images, (4) applying edge detection, and (5) splitting dataset to training/test datasets.

4.2.1 Dataset mutation

During the mutation, initial images (Figure 2(A)) are first converted into grayscale to remove unreliable variables such as structure color, time of day, and shadowing. Then, as shown in Figure 2 (B-F), dataset balancing was conducted by applying 3 types of flips and 2 types of rotations to cracked sub datasets without replacing the original images. Other noise functions like white edges and blurring (Figure(G-H)) were also considered, but we opted to only use the flips and rotations. Further, the uncracked sub datasets remain unchanged because there were already a sufficient number of uncracked images.

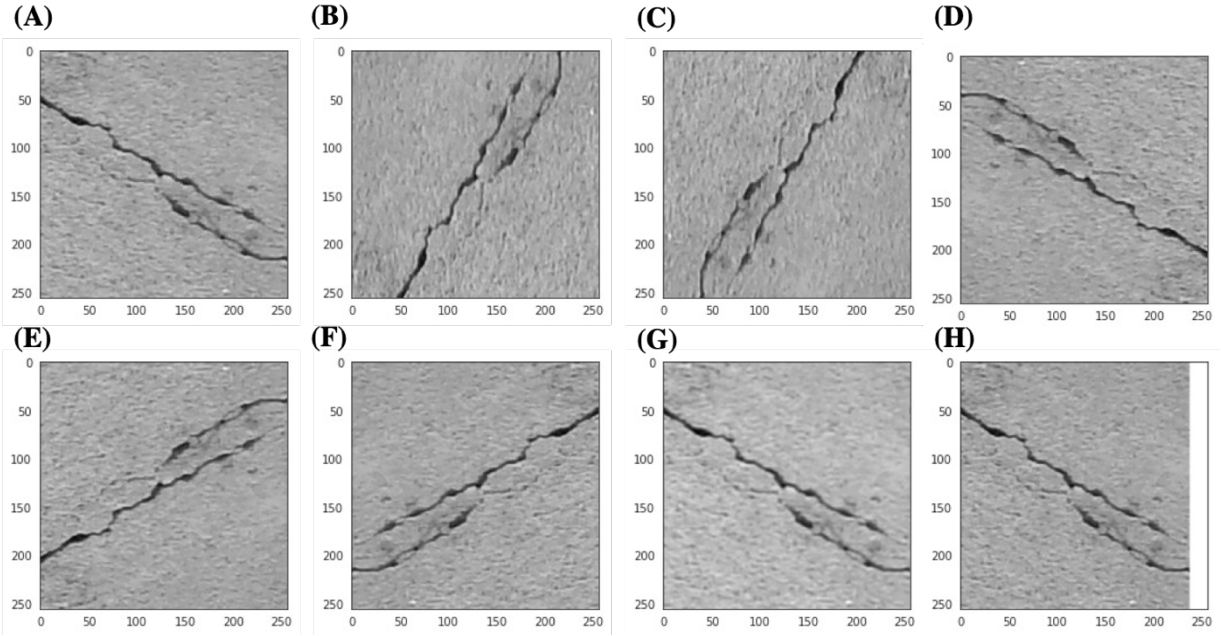


Figure 2. Illustration of preprocessing (features reduction and noise addition). (A) Original image (B) Rotated image_1 (D) Rotated image_2 (C) Flipped image_1 (D) Flipped image_2 (E) Flipped image_3 (F) Blurred image (G) White-edged image.

The distribution of dataset after data mutation is shown in Figure 3. Then we normalized each dataset (Decks, Pavements, and Walls) into 20,000 images (10,000 positive and 10,000 negative samples) and reduced the images (256 x 256 by default) into 64 x 64 to speed up the model. After mutation, we split the dataset into training and test datasets according to the ratio of 80% and 20%.

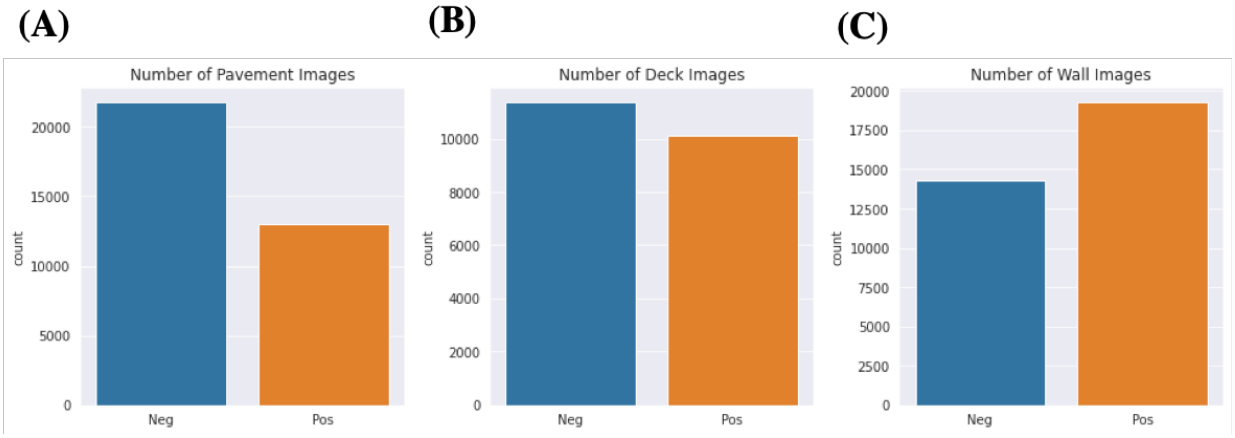


Figure 3. Distribution of mutated dataset (A) Pavement dataset, (B) Deck dataset, (C) Wall dataset, after applying flips/rotations but before discarding excess images.

4.2.2 Feature selection

During our model testing, we noticed that our model was experiencing difficulty with classification. We suspected this was due to the inordinate amount of features fed into the model (128 x 128 grayscale pixels per sample). Therefore, we chose to apply dimensionality reduction by principal component analysis. We kept the first 2 components for retained variances of 0.63558196 and 0.056407. The visualization of one training dataset with 2 PCA components is shown in Figure 4.

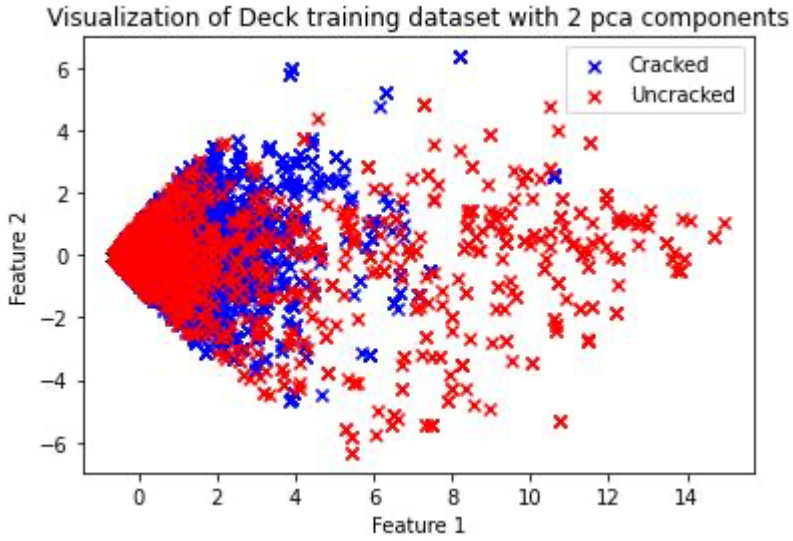


Figure 4. Visualization of Deck training dataset with 2 PCA components.

However, unfortunately using feature reduction using PCA did not help in the long run with increasing validation accuracy, which did not increase very far beyond 60% on the balanced dataset. To increase the model prediction accuracy, we further employed edge detection to make our model's features have more information. As shown in Figure 5, the images after edge detection only have white and edge pixels and identified the boundaries of cracks in the images which can help with data segmentation and data extraction. The intent was to reduce the amount of complexity which the model needed to learn in order to identify cracks in the input.

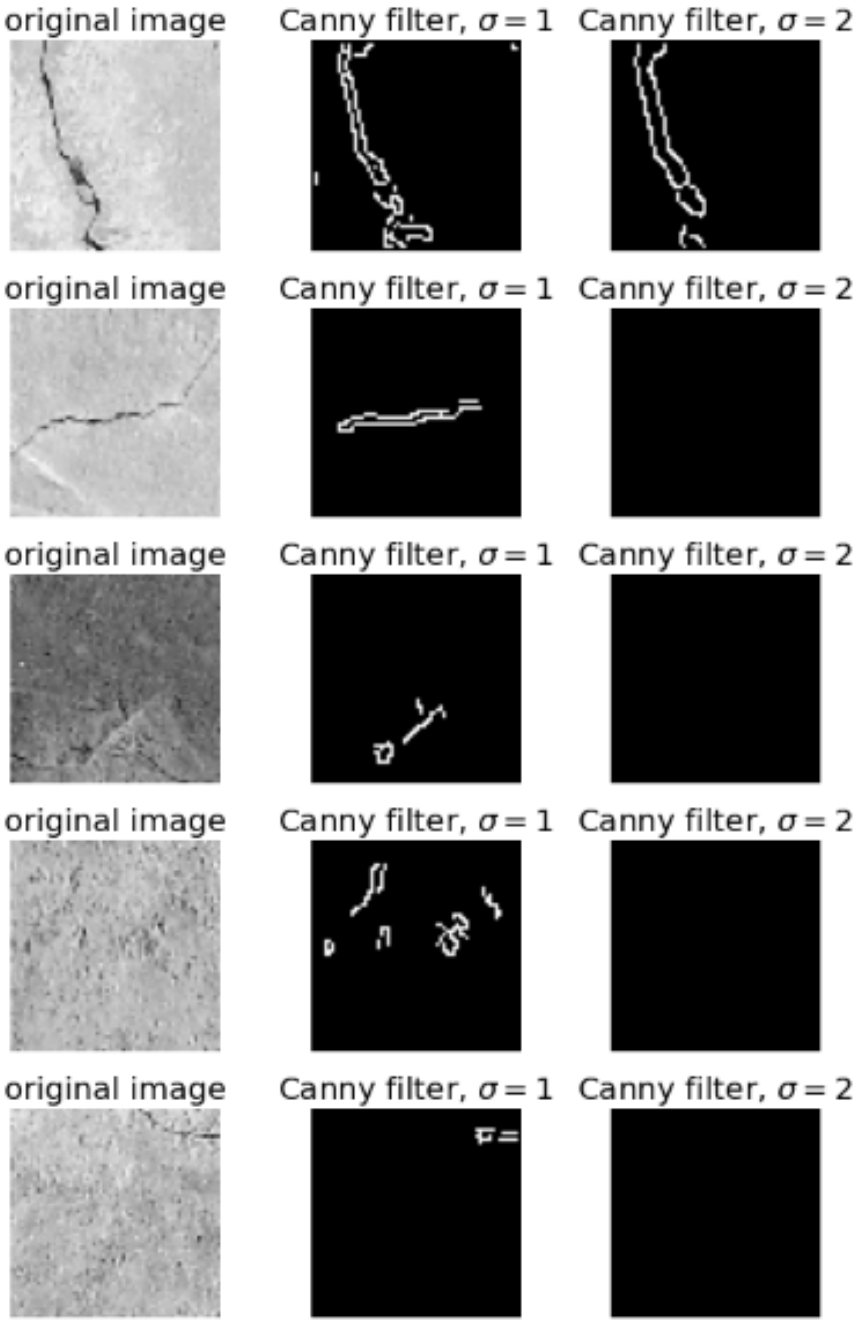


Figure 5. Example of images after edge detection for various deviation values of the Gaussian kernel.

4.3 Supervised learning

Our PyTorch neural net model was originally a simple classifier using a chain of three linear layers and a sigmoid output. Due to difficulties just getting the image data to be processed properly (described in more detail above, in 4.2), we did not explore more complicated model architectures, though we plan to do so in the future.

This architecture was modified somewhat after the input was changed from PCA values to the canny edge detection outputs. We extended the chain of linear layers to have 6 linear layers and

added Leaky ReLU activation functions to each, except for the final layer which used a sigmoid function. The linear layer has an input dimension equal to 64 * 64, our flattened canny input dimension, and fed into a buffer layer of dimension 20 which was designed to retain some of the initial complexity of the input layer. This fed into 3 hidden layers of dimension 12, and finally produced a 1-logit output for our class estimate. We trained this model for 5 epochs with a learning rate of 1e-4. We used an alpha value of 1e-2 for our Leaky ReLUs. Although training for more epochs was an option, the model began to overfit after only 5 due to the model's architecture, so we stopped at 5. We used the Adam optimizer and the binary cross entropy loss function, and the initial training outputs are shown in figure 6.

```
Epoch: 1/5      Avg Train Loss: 0.6705  Avg Val Loss: 0.6290     Train Accuracy: 60      Val Accuracy: 65
Epoch: 2/5      Avg Train Loss: 0.5891  Avg Val Loss: 0.6088     Train Accuracy: 67      Val Accuracy: 67
Epoch: 3/5      Avg Train Loss: 0.5355  Avg Val Loss: 0.5998     Train Accuracy: 71      Val Accuracy: 69
Epoch: 4/5      Avg Train Loss: 0.4848  Avg Val Loss: 0.6237     Train Accuracy: 73      Val Accuracy: 69
Epoch: 5/5      Avg Train Loss: 0.4527  Avg Val Loss: 0.6834     Train Accuracy: 74      Val Accuracy: 70
```

Figure 6. Loss and accuracy from initial testing. (A) Before data balancing, (B) After data balancing.

4.4 Unsupervised Learning

The unsupervised learning aims at clustering the cracked images in terms of the severity level. Two important features are extracted from images: the number of crack's edge pixels and the "raw model score" from our last layer of crack detection model (right before the sigmoid output). The raw model score was used to differentiate cracked with non-crack images in the model, representing the ability to detect cracks using computer vision. Only the cracked images were used for the severity clustering since this is a post-classification goal (severity of uncracked image would not make sense).

Initially, we ran the pre-defined function for collecting the number of crack's edge pixels, and the "raw model score". The distribution of normalized likelihood of these two features is displayed in Figure 7. GMM is one of the options that we tested for soft-clustering results. The higher predicted score from sigmoid function in supervised learning, the higher possibility of data points can be defined as cracked, allowing the "predicted score" to indicate different severity levels. Therefore, the output of sigmoid function in supervised learning ranging from 0 to 1 were chosen to group data points into 4 clusters. Original likelihood distribution with random mean, variance and weight is shown in Figure 6. Then, we repeated the expectation and maximization steps until there is little change in theta (Figure 7).

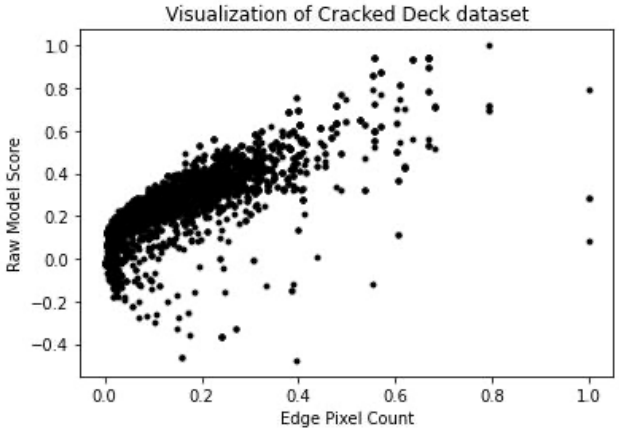


Figure 7. Normalized likelihood distribution.

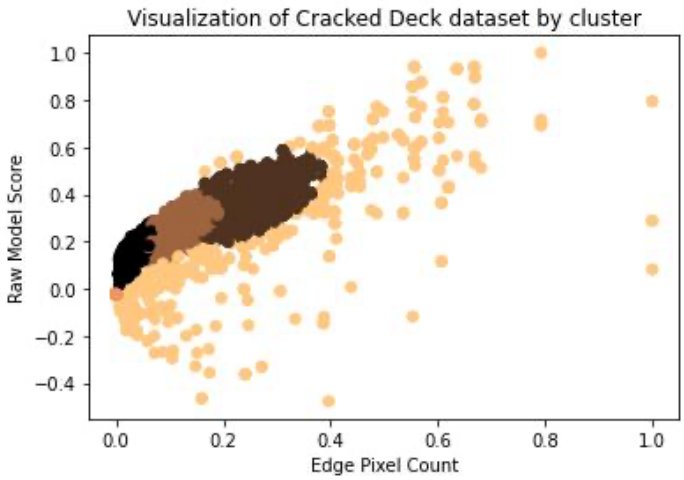


Figure 8. Cracked Decks Dataset clusters determined by GMM.

After GMM training, severity levels could not be distinguished due to the high density of the datapoints. Therefore, we conducted DBSCAN which can handle clusters of multiple sizes and structures and is not powerfully influenced by noise or outliers. Figure 9 shows the DBSCAN clustering result after training with different epsilon and minPoints with 4 clusters as generated results.

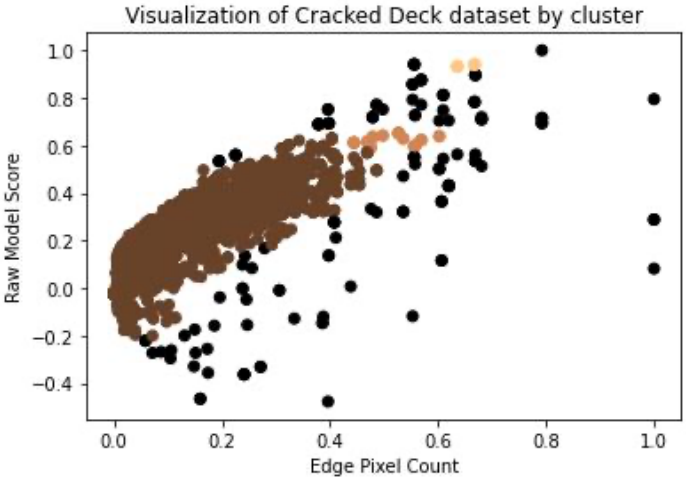


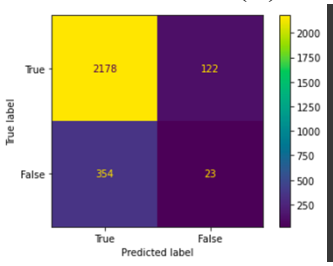
Figure 9. Different Cracked Decks Dataset clusters determined by DBSCAN.

5. Results and Discussion

5.1 Supervised

We made use of general metrics such as Accuracy, F1, Recall and Precision which are very useful for observing results from Classification. However, while processing the data we observed that we had a much larger amount of non-cracked images than cracked ranging to over 8 times as much. We believed this would skew the metrics by a significant amount, which prompted us to make use of the sklearn “Classification Report”. This provides relevant values per class as well as a weighted average of the accuracy, F1, recall and precision. A clear overview of the results can be seen in Figure 10.

(A) (B)



	precision	recall	f1-score	support
Uncracked	0.84	0.70	0.76	2280
Cracked	0.13	0.25	0.17	397
accuracy			0.63	2677
macro avg	0.49	0.48	0.47	2677
weighted avg	0.74	0.63	0.68	2677

Figure 10. (A) Confusion matrix. (B) Classification report from initial model testing

5.2 Unsupervised

To evaluate the clustering accuracy of GMM and DBSCAN, we calculated the models' Silhouette Score and Davies-Bouldin Index which reflect the compactness, connectedness, and separation of the cluster partitions. The results of the scores can be seen in Table 1 below. As DBSCAN has higher Silhouette Score and lower Davies-Bouldin Score, DBSCAN performed better than GMM.

Table 1. Internal performance measures of GMM and DBSCAN.

	GMM	DBSCAN
Silhouette Score	0.6211911931086308 (n=5)	0.625597181831163 (n=4)
Davies-Bouldin Score	2.396175298056523 (n=5)	1.428964905260969 (n=4)

6. Conclusion:

In this project, we implemented 2 separate parts: the first is a classification system that helps to detect cracks in different images, and the second is an evaluation system that predicts the severity of cracks in the cracked images. The detection of cracks performed well after balancing the training dataset. Difficulties arose in selecting data for the clustering model. The predicted score obtained from NN model, which is mainly related to the possibility of crack images, might not be solid for severity evaluation. Therefore, the output is not ideal enough due to the inaccurate input and the algorithm itself.

References:

1. Zou, Q., Zhang, Z., Li, Q., Qi, X., Wang, Q., & Wang, S. (2019). DeepCrack: Learning Hierarchical Convolutional Features for Crack Detection. IEEE Transactions on Image Processing, 28(3), 1498-1512. https://doi.org/10.1109/tip.2018.2878966.

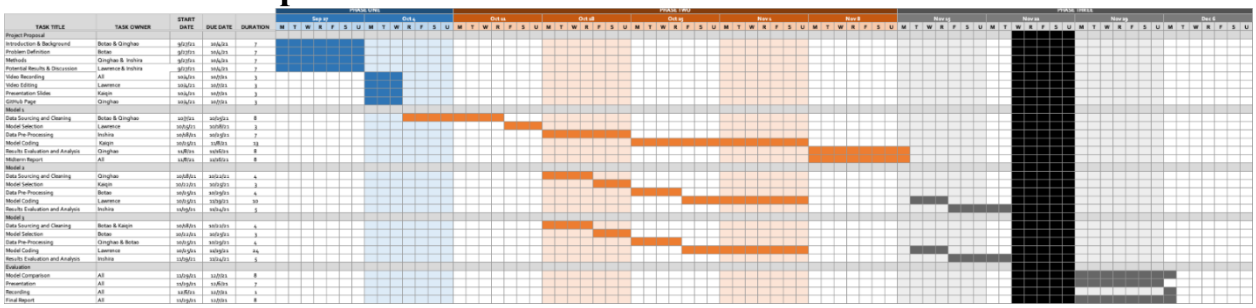
2. Munawar, H. S., Hammad, A. W. A., Haddad, A., Soares, C. A. P., & Waller, S. T. (2021). Image-Based Crack Detection Methods: A Review. Infrastructures, 6(8), 115. https://doi.org/10.3390/infrastructures6080115.

3. Mansuri, L. E., & Patel, D. A. (2021). Artificial Intelligence-based automatic visual inspection system for Built Heritage. *Smart and Sustainable Built Environment*. https://doi.org/10.1108/sasbe-09-2020-0139.

4. Mishra, M. (2021). Machine learning techniques for structural health monitoring of heritage buildings: A state-of-the-art review and case studies. *Journal of Cultural Heritage*, *47*, 227–245. https://doi.org/10.1016/j.culher.2020.09.005.

5. Lei Zhang , Fan Yang , Yimin Daniel Zhang, and Y. J. Z., Zhang, L., Yang, F., Zhang, Y. D., & Zhu, Y. J. (2016). Road Crack Detection Using Deep Convolutional Neural Network. In 2016 IEEE International Conference on Image Processing (ICIP). http://doi.org/10.1109/ICIP.2016.7533052.

Timeline and Responsibilities:



Contribution Table (Final Progress):

Botao	Data preprocessing, unsupervised learning
Inshira	Results analysis, final report and slides writing.
Kaiqin	Data visualization, final report and slide writing.
Lawrence	Feature reduction, model, supervised learning, notebook restructuring, final report.
Qinghao	Data preprocessing, unsupervised learning, notebook restructuring, Github page set up.

Contribution Table (Mid-term Progress):

Botao	Data loading, image resizing, adding three types noise into datasets, dataset and image visualization.
Inshira	Data metrics, results analysis, report writing
Kaiqin	Dataset and image visualization, adding noise functions, report writing
Lawrence	Model, train/test loops, notebook setup & restructuring, report
Qinghao	Data pre-processing part: data loading, image resizing, Github page update

Contribution Table (Project Proposal):

Botao	Literature reviews, intro/problem definition/potential results writing,
Inshira	GANTTs chart filling, assisted writing Methods/potential results section
Kaiqin	Literature reviews, slides making.
Lawrence	General edits for clarity, rewrote methods/potential results, video editing.
Qinghao	Methods and potential results writing, Github page set up