Infrastructure Surface Crack Detection

Fall 2022 - CS7641 ML - Group Project Midpoint Report

Team: Botao Li, Inshira Seshie, Kaiqin Bian, Lawrence Bradley, Qinghao Zeng

Github: https://github.gatech.edu/qzeng41/CS7641_group_34

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. Methods:

We used supervised learning on our labeled dataset for crack detection on concrete surface images. We used pytorch's neural net module and created a simple classification model based on image features obtained from PCA.

For our next steps, we will also implement an unsupervised learning algorithm to gauge the severity of any cracks detected in the image. Clustering is based on image features of detected cracks (such as width and length), which enables efficient defect reporting to project and maintenance supervisors with detailed descriptions of the structural flaw and its severity.

So far during our project we have made use of the following packages:

- Numpy for linear algebra.
- Pandas for data processing of csv files.
- Os for operating the system to open, read, and write files.
- Pytorch for neural networks
- OpenCV-python library, cv2 package for computer vision methods.
- Sklearn for PCA and performance metrics
- Seaborn for data visualization
- Matplotlib for data visualization

3.1 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.

3.2 Data Visualization and Preprocessing:

Following manual inspection, we determined that our main dataset “Structal Defects Network 2018” has been cleaned enough, as there were no incorrected, 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.

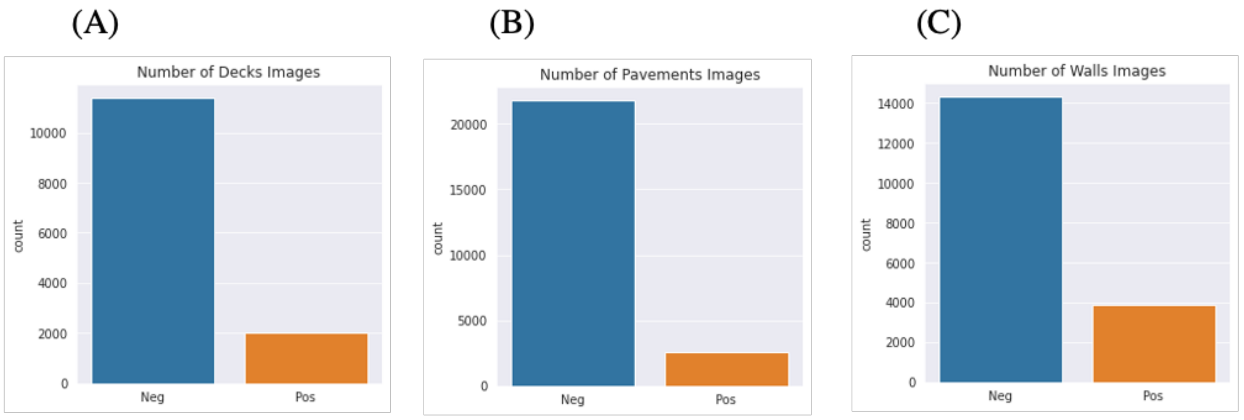
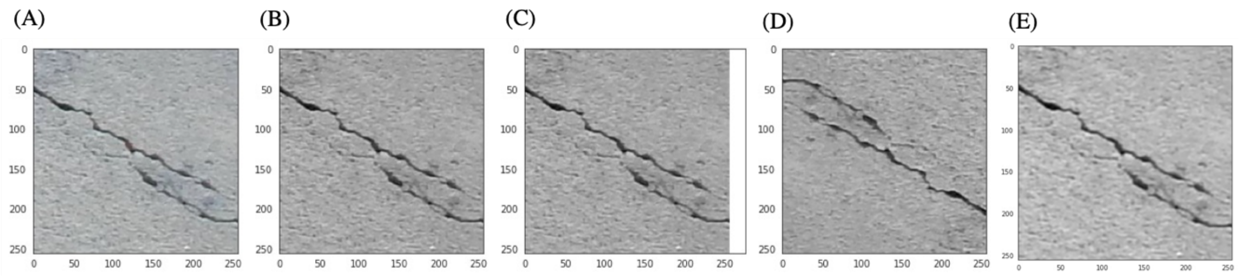


Figure 1. Distribution of non-cracked (Neg) and cracked (Pos) images in original datasets ((A) dataset_Decks, (B) dataset_Pavements, (C) dataset_Walls).

Because our data's distribution is not balanced between the classes, we decided that we should take steps to increase the size of our cracked/positive dataset. We chose to implement some functions to introduce noise to our data for two purposes. First, so that we could artificially extend the size of our dataset. Second, to make our model more robust at dealing with noise by obscuring the true data.

During pre-processing, as shown in Figure 2 (A)-(B), we converted the original image into grayscale to remove unreliable variables such as structure color, time of day, and shadowing. As shown in Figure 2 (C)-(E), added noise functions included white edges, flipping, and blurring. These were added into random selected images in each sub-datasets. The distribution of non-cracked (Neg) and cracked (Pos) images with noises are shown in Figure 3.



.

Figure 2. Illustration of preprocessing (features reduction and noises addition).
(A) Original image (B) Gray image (C) White edge added image (D) Flipped image (E) Blurred image.

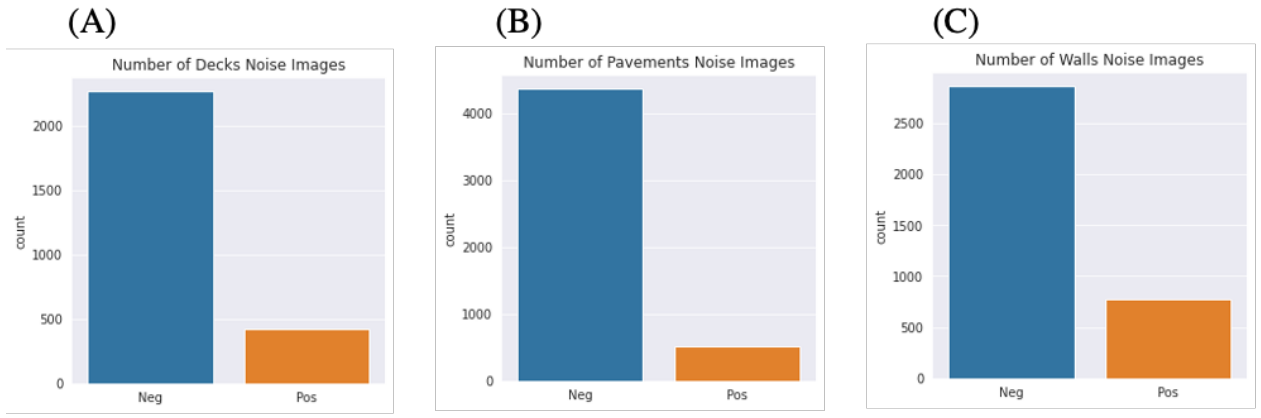
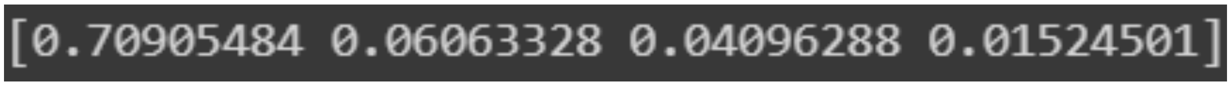


Figure 3. Distribution of non-cracked (Neg) and cracked (Pos) images in noise dataset (only noise images involved) ((A) dataset_Decks_noise, (B) dataset_Pavements_noise, (C) dataset_Walls_noise).

Noise images (5% adding white edge, 5% flipping, 10% blurring) were added into datasets without replacing original images, which also helps to reduce the possibility of overfitting to the dataset, and then the data was separated to 80% train and 20% test.

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 we were feeding into the model (256 x 256 grayscale pixels per sample). Therefore we chose to perform principal component analysis to reduce the number of features introduced to our model. We kept the first 4 components, for a total retained variance of 0.826. The retained variance matrix for these components is shown in figure 4.

Due to the size of our dataset, we were unable to perform PCA on the full 256x256 images. Attempting to do so caused our Google colab environment to crash by exceeding the RAM limit, so we resized the samples to 128x128 first. We also operated only on the Decks dataset for testing for a similar reason. In the future we hope to resolve this issue to allow our model to run on the full dataset.

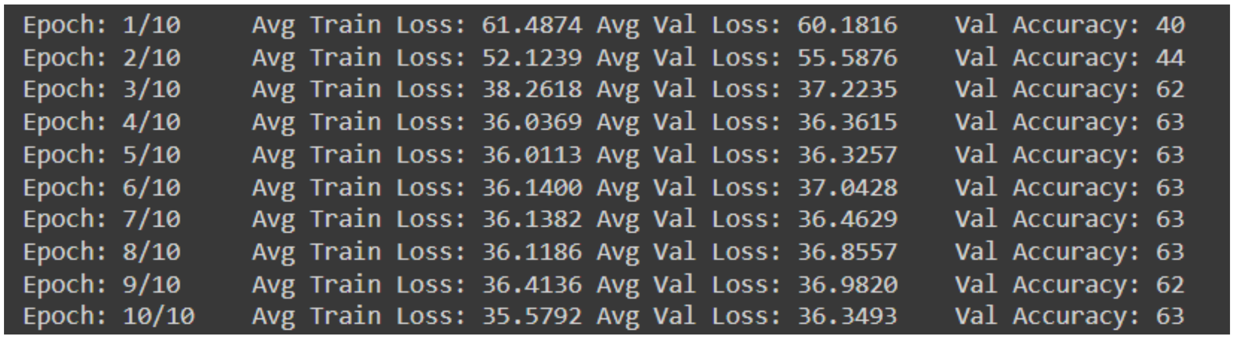

```
[0.70905484 0.06063328 0.04096288 0.01524501]
```
Figure 4. Variance retained by each component after PCA.

3.3 Model architecture

Our pytorch neural net model is 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 3.2), we did not explore more complicated model architectures, though we plan to do so in the future.

Our first linear layer had an input dimension of 4 and an output dimension of 12, which fed into two more 12-node linear layers and finally produced a single-value output for our sigmoid function. We trained for 10 epochs with a learning rate of 0.1 during our brief evaluation period. In the future we will train the model for a significantly longer amount of time, and consider the application of more complicated layer types.

We used the Adagrad optimizer and the binary cross entropy loss function. Our initial training outputs are shown in figure 5. Note that the accuracy is actually a very poor measure of performance for our dataset due to the skewed nature of our class distribution in our data. Although we prepared a suite of noise functions to augment our dataset, our dataset still contained a bias towards one class, making it artificially easier to reach higher accuracy scores.

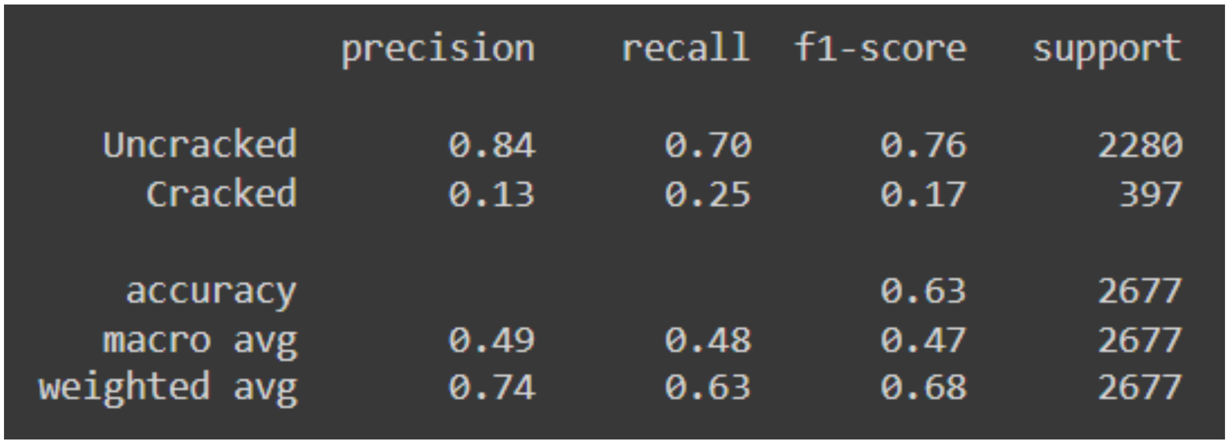


```
Epoch: 1/10     Avg Train Loss: 61.4874 Avg Val Loss: 60.1816    Val Accuracy: 40
Epoch: 2/10     Avg Train Loss: 52.1239 Avg Val Loss: 55.5876    Val Accuracy: 44
Epoch: 3/10     Avg Train Loss: 38.2618 Avg Val Loss: 37.2235    Val Accuracy: 62
Epoch: 4/10     Avg Train Loss: 36.0369 Avg Val Loss: 36.3615    Val Accuracy: 63
Epoch: 5/10     Avg Train Loss: 36.0113 Avg Val Loss: 36.3257    Val Accuracy: 63
Epoch: 6/10     Avg Train Loss: 36.1400 Avg Val Loss: 37.0428    Val Accuracy: 63
Epoch: 7/10     Avg Train Loss: 36.1382 Avg Val Loss: 36.4629    Val Accuracy: 63
Epoch: 8/10     Avg Train Loss: 36.1186 Avg Val Loss: 36.8557    Val Accuracy: 63
Epoch: 9/10     Avg Train Loss: 36.4136 Avg Val Loss: 36.9820    Val Accuracy: 62
Epoch: 10/10    Avg Train Loss: 35.5792 Avg Val Loss: 36.3493    Val Accuracy: 63
```

Figure 5. Loss and accuracy from initial testing

4. Results:

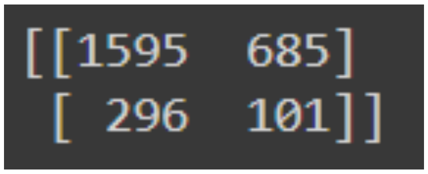
We made use of general metrics such as Accuracy, F1, Recall and Precision which are very useful for observing results from Classification. However, while processing the data we observed that we had a much larger amount of non-cracked images than cracked ranging to over 8 times as much. We believed this 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 6 below.



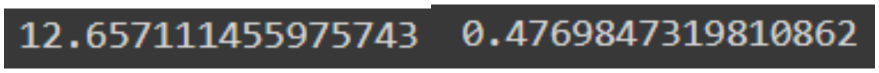
```
              precision    recall  f1-score   support

   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 6. Classification report from initial model testing

As shown above and discussed previously our model is remarkably more adept at identifying the uncracked surfaces simply because of the overwhelming sample ratio of our dataset. We intend to address this problem in the future by creating a more balanced representation of the two classes in our data. Below are also the confusion matrix, log loss, ROC graph, and ROC_AUC score.



```
[[1595  685]
 [ 296  101]]
```



```
12.657111455975743  0.4769847319810862
```

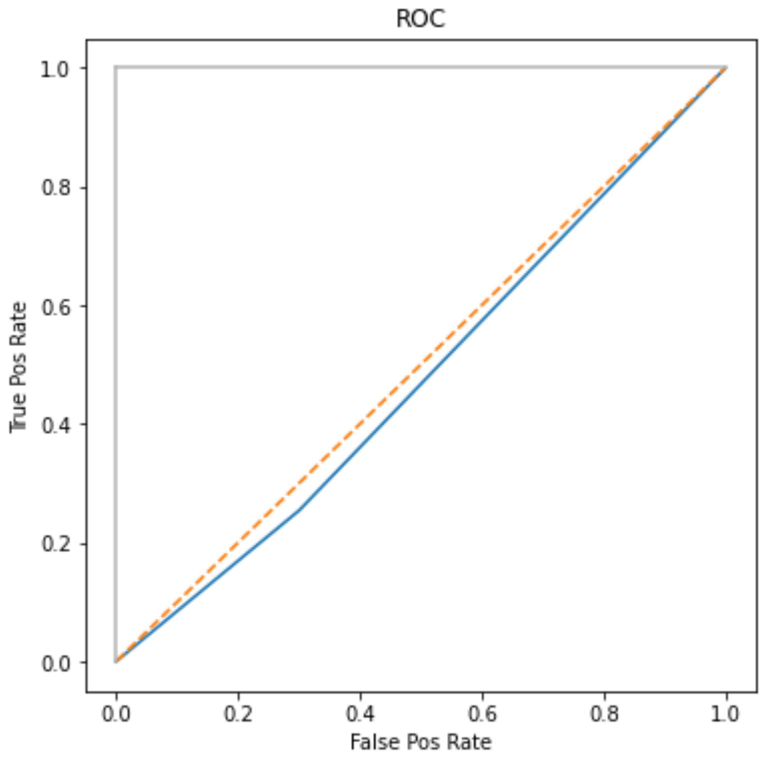


Figure 7. Top to bottom, left to right: confusion matrix, log loss, ROC_AUC score, ROC graph

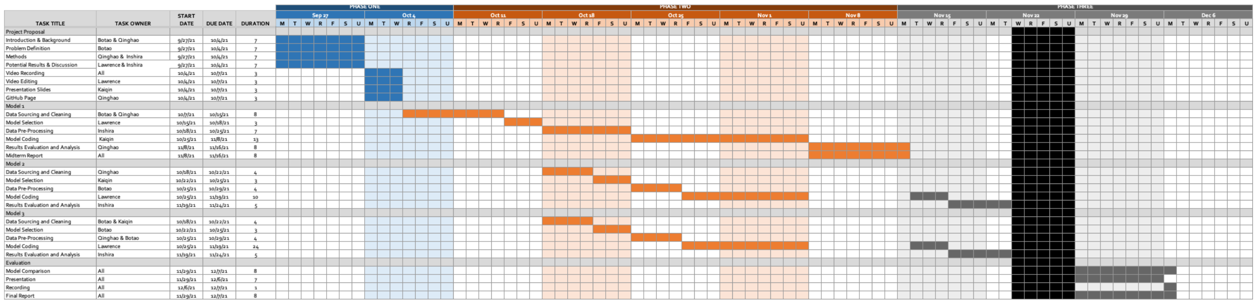
With the results from the Classification Report we are able to understand the reasoning behind the large log loss value which indicates the likelihood of our model to produce the incorrect prediction the higher the value is. In order to improve our overall accuracy within the model and lead it to be able to effectively identify cracked surfaces are next goal is to equalize the amount of cracked and non-cracked images used to train the data. Adding additional noise especially to the cracked dataset and reducing the amount of non-cracked images used would allow us create a model that is less efficient in identifying non-cracked surfaces but greatly increase its ability to identify cracked surfaces.

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:



TASK TITLE	TASK OWNER	START DATE	DUE DATE	DURATION
Project Proposal				
Introduction & Background	Botao & Qinghao	9/27/21	10/4/21	7
Problem Definition	Botao	9/27/21	10/4/21	7
Methods	Qinghao & Inshira	9/27/21	10/4/21	7
Potential Results & Discussion	Lawrence & Inshira	9/27/21	10/4/21	7
Video Recording	All	10/4/21	10/7/21	3
Video Editing	Lawrence	10/4/21	10/7/21	3
Presentation Slides	Kaiqin	10/4/21	10/7/21	3
GitHub Page	Qinghao	10/4/21	10/7/21	3
Model 1				
Data Sourcing and Cleaning	Botao & Qinghao	10/7/21	10/15/21	8
Model Selection	Lawrence	10/15/21	10/18/21	3
Data Pre-Processing	Inshira	10/18/21	10/25/21	7
Model Coding	Kaiqin	10/25/21	11/8/21	13
Results Evaluation and Analysis	Qinghao	11/8/21	11/16/21	8
Midterm Report	All	11/8/21	11/16/21	8
Model 2				
Data Sourcing and Cleaning	Qinghao	10/18/21	10/22/21	4
Model Selection	Kaiqin	10/22/21	10/25/21	3
Data Pre-Processing	Botao	10/25/21	10/29/21	4
Model Coding	Lawrence	10/25/21	11/19/21	10
Results Evaluation and Analysis	Inshira	11/19/21	11/24/21	5
Model 3				
Data Sourcing and Cleaning	Botao & Kaiqin	10/18/21	10/22/21	4
Model Selection	Botao	10/22/21	10/25/21	3
Data Pre-Processing	Qinghao & Botao	10/25/21	10/29/21	4
Model Coding	Lawrence	10/25/21	11/19/21	24
Results Evaluation and Analysis	Inshira	11/19/21	11/24/21	5
Evaluation				
Model Comparison	All	11/29/21	12/7/21	8
Presentation	All	11/29/21	12/6/21	7
Recording	All	12/6/21	12/7/21	1
Final Report	All	11/29/21	12/7/21	8

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