DETECTING BUILDING DEFECTS

A MINI PROJECT REPORT

Submitted by

BL.EN.U4CSE16014	ARVIND SUDHEER
BL.EN.U4CSE16106	RAMSHANKAR YADHUNATH
BL.EN.U4CSE16112	SRIVENKATA SRIKANTH

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE ENGINEERING

AMRITA SCHOOL OF ENGINEERING, BANGALORE

AMRITA VISHWA VIDYAPEETHAM

BANGALORE 560 035

JANUARY-2020

AMRITA VISHWA VIDYAPEETHAM
AMRITA SCHOOL OF ENGINEERING, BANGALORE, 560035

BONAFIDE CERTIFICATE

This is to certify that the project report entitled **“DETECTING BUILDING DAMAGE”** submitted by

ARVIND SUDHEER	BL.EN.U4CSE16014
RAMSHANKAR YADHUNATH	BL.EN.U4CSE16106
SRIVENKATA SRIKANTH	BL.EN.U4CSE16126

in partial fulfillment of the requirements for the award of the **Degree Bachelor of Technology** in **COMPUTER SCIENCE ENGINEERING** is a bonafide record of the work carried out under my(our) guidance and supervision at Amrita School of Engineering, Bangalore.

Jyotsna C
Assistant Professor
Computer Science and Engineering

Dr. Amudha J
Chairperson
Computer Science and Engineering

This project report was evaluated by us on …………………...(Date)

EXAMINER 1 EXAMINER 2

Acknowledgement

The “Detecting Building Defects” was an amazing project to work on as it helped us learn a lot in the field of computer vision. We were able to transcend the boundaries of convention and think outside the box and come up with novel approaches to solving our problems. There were several times that our approaches failed, but each time was a new learning experience. We thank our **HoD, Dr. Amudha J** for her valuable inputs and suggestions in the course of our project right from the initial stages till the final stage. We would like to thank our guide **Mrs. Jyotsana C** for her constant support throughout the project. Her inputs in using various Digital Image Processing techniques were fulcrum points in our project. We would also like to extend our gratitude to **Digiledge**-Startup at Amrita TBI for giving us the idea of “Building Damage Detection”.

We are also extremely thankful to the **evaluation panel** for their inputs during the zeroth presentation of our project.

Abstract

Buildings are structures built with tremendous investment of time, money and emotion. Therefore, every stakeholder involved in the process starting from construction companies to the tenants, wants to make sure that a structure is built well and that it can serve its purpose without any safety hazards. The safety of buildings is dependent on several factors such as foundational stability, structural integrity, construction material used, climatic conditions etc. While a lot of these factors can only be evaluated by a civil engineering expert, there are factors like detecting visible structural damage that might cause a severe investment of time via manual inspection. Therefore, in this project we have made an attempt to detect and identify visually discernible defects in buildings, thus reducing the need for manual inspection. Such a method also introduces objectivity in evaluation of defects.

Keywords : *building defect, image processing, CNN*

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF NOMENCLATURE AND ABBREVIATIONS USED

IP/DIP : Image Processing/Digital Image Processing

UAV : Unmanned Aerial Vehicle

CNN : Convolutional Neural Network

R-CNN : Recurrent CNN

GPU : Graphical Processing Unit

1. INTRODUCTION

1.1 Definition - About the Domain of Study

One of the most important aspects of building construction and maintenance is the ability to assess the safety of the resulting structure. While there are several works such as [1] which focus on the multiple aspects that need to be considered for constructing safe structures, others such as [2] focus on specific considerations of structural safety. Another focal point of building construction and maintenance is the aesthetic appeal of the construction. A building with an unsafe structure or a flawed appearance is most likely to be overlooked by potential dealers and buyers. However, there can be a variety of factors that engender this loss of structural integrity of buildings such as climatic effects, geographical effects, construction materials, faulty designs etc. [3]. But, irrespective of the cause it all finally degrades the overall value of a building.

A defect in a product is a general term for an anomaly that severely undermines the value of the product. When such “defects” occur in buildings, apart from the fact that they degrade the value of the building, it is to also be noted that reworking on these defects is a cost-intensive and time-intensive process if not detected early [4]. Recent past has witnessed several cases of building collapses that arose due to lack of timely inspection and identification of potential structural faults with buildings.

Based on visibility, defects in any building can be of two major types :

- Defects discernible to the naked eye (External)
- Defects not discernible to the naked eye (Internal)

In this project, our focus has been on the externally discernible defects on a building.

Based on the dependence on structure, building defects can be classified as :

- Structural Defects (Affects structural elements)
- Non-structural Defects (Affects non-structural elements)

While structural defects affect the overall stability of the building (roofing, external walls etc.), non-structural defects relate to the aesthetic appearance of buildings (paint, plastering etc.). [5]

Some of the defects that are discernible to the human eye, according to [5] are as follows :

- ***Cracks*** - Often an effect of ageing of buildings, can be very dangerous if not dealt with at an early stage (Fig 1.1)

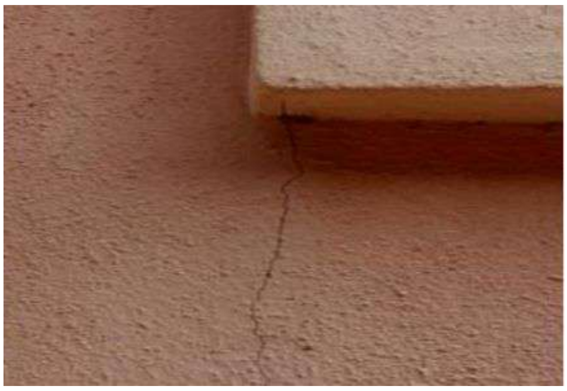


Fig 1.1

- ***Peeling of Paint*** - Occurs in areas that are exposed to excessive rain and great dampness, spoils the look of a structure (Fig 1.2)

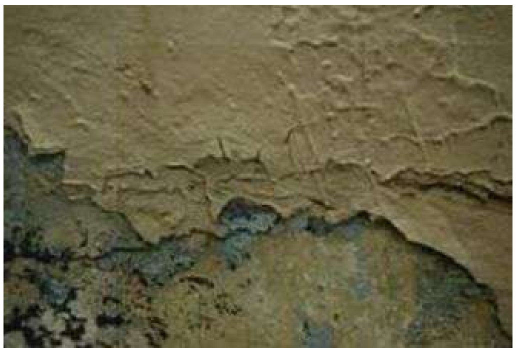


Fig 1.2

- ***Rising Dampness*** - Onset of unwanted and excessive moisture or water into a building, causes severe damage to brickwork (Fig 1.3)

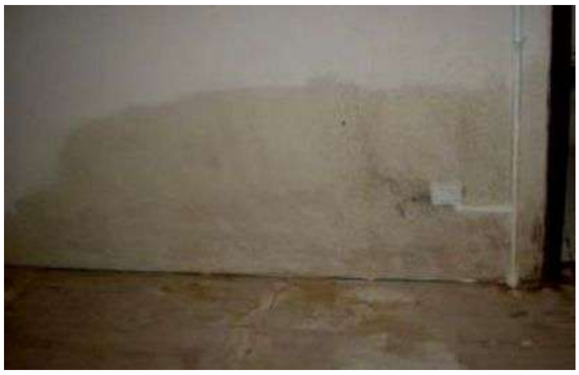


Fig 1.3

- ***Timber Decay*** - Decay of structures made of wood due to moisture, chemicals or insect activity (Fig 1.4)

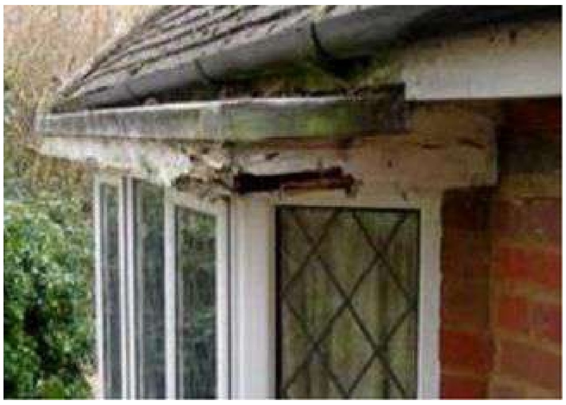


Fig 1.4

- ***Fungal Attack*** - Unwanted green growth on walls, distorts appearance (Fig 1.5)

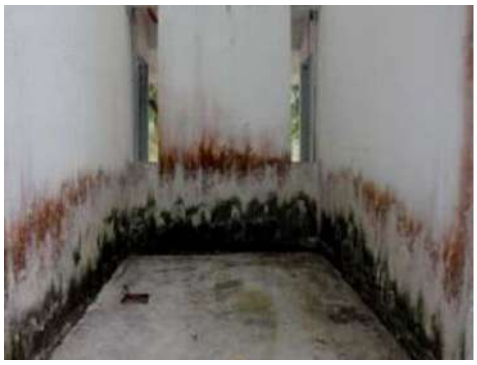


Fig 1.5

- ***Defective Plastering*** - Defects that are concerned with falling of plasters, holes in plaster walls etc. (Fig 1.6)

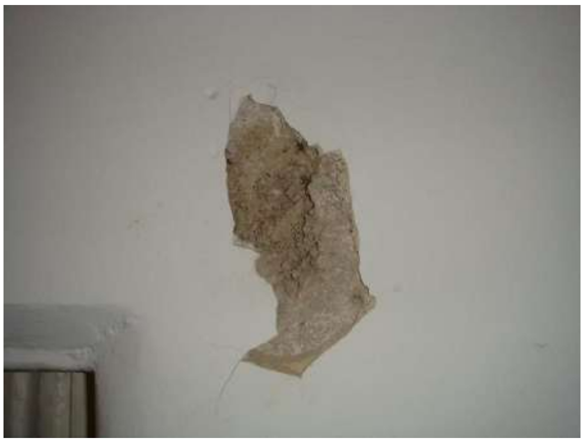


Fig 1.6

- ***Roof Sagging*** - Sagging roofs are dangerous and need to be identified early to prevent roof collapse (Fig 1.7)

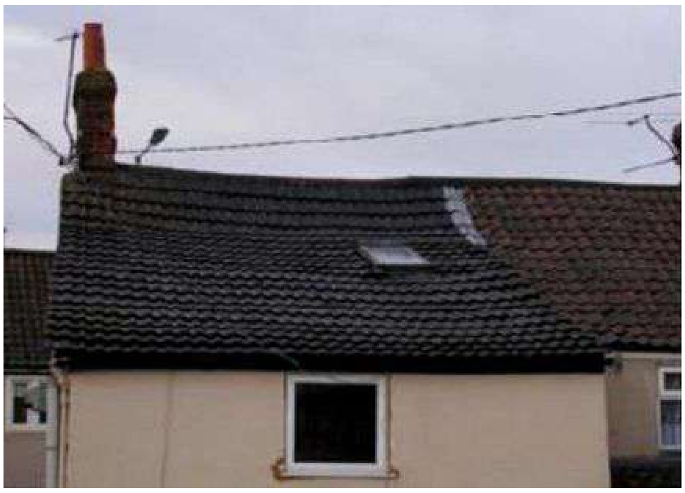


Fig 1.7

1.2 Motivation

Defects affect several stakeholders including tenants, construction companies and government agencies. Consumers or tenants will be the most affected by a faulty building as it is their lives that are under jeopardy if their property is faulty. It is to also be considered that tenants make a large share of emotional investment in the houses that they buy and damage to such property will lead to a loss of both their monetary as well as emotional investment. Construction companies, on the other hand, need to evaluate their projects and rid them off defects(if found) in order to maintain their credibility in the industry. The role of government agencies as stakeholders can’t be overlooked as the final responsibility of the lives of its people lie in the hands of those with administrative power. Therefore, these agencies too are in need of sophisticated techniques to evaluate structures and issue certificates of authorization(given the safety standards are met).

Over the years, there have been several attempts such as the one in [6] to assess building defects and evaluate structural conditions. Such analysis gives a great outlook on the factors that need to be considered and also allows for timely repairs, thus avoiding catastrophic occurrences in the future. But, the problem with most methods in use at present is that they involve a lot of manual labour and manual surveying. Therefore, there is a need for an imperative to design a system that would be autonomous in its operation of being able to detect defects in buildings and raise early alarms. It is also important that such a system should be robust, fast and convenient to use.

Tenants will need a Safe Home
Construction Companies will want to uphold Quality
Government agencies will need to Evaluate

Fig 1.8 : Motivation for the Project

1.3 Problem Statement

To build a model that will detect external damage to a building, assess the intensity of the defect and inform stakeholders, thus reducing the need for manual inspection.

1.4 The Approach

With the problem statement defined, the next step was to identify the approach to be used. The scope of this project deals with the use of *Computer Vision* to detect damages on external surfaces of buildings.

1.5 The Limitations of Data

Since there was not too much work being performed in this domain, it was difficult for us to collect large amounts of data for our use. There was a dearth of both the quantity and quality of data for us to work with.
Therefore, we had to rely on external data acquisition for our project. We have collected the data for the project from :

- A pre-defined crack images dataset
- Data Labelling Sites(Dataturks)
- Google Images
- Handheld Devices

2. LITERATURE SURVEY

The use of computer vision in building damage detection has been mostly concerned around the detection and identification of cracks on walls. This is due to the ability to identify cracks on a surface with relative ease than the other defects on walls. Also, public data sources for cracks are relatively accessible when compared to the extreme lack of public datasets for other wall defects.
It is also to be noted that there has been more work performed in detecting damage in buildings post-calamity, when opposed to the idea of detecting issues with structures before they lead to great damages.

2.1 Building Damage Detection Post-Calamity

Papers such as the one by Fujita et. al [7] focus on detecting damages on buildings from aerial images post-tsunami using CNN. They have created a benchmark AIST building change detection dataset that can be used to predict if a given building has been washed away or not. Their solution in the paper has achieved an accuracy ranging from 94-96%.

Li et. al [8] in their work detected damaged buildings using high resolution satellite imagery. They used OCSVM (one-class classifier) that helped make classifications with the help of only training samples that showed building damage. The best results were obtained at 82.33% overall accuracy and a kappa coefficient of 60.09%.

2.2 Wall Crack Detection

Lee et. al [9] proposes an IP inspired method for crack detection and crack width estimation. Images of cracks on concrete walls were procured by the authors using their own image acquisition setup. Cracks on walls were identified using IP and non-crack shapes were segregated based on shape analysis. Post identifying the cracks, the width was calculated using skeletonization and boundary detection techniques.

Hoang [10] in his work attempts to classify the types of cracks that can occur on a wall using steerable filters and integral projections. The dataset was acquired by the author through a self-created setup and there were about 500 images that were used. Support Vector Machines were used to classify a given image as belonging to the classes of longitudinal crack, transverse crack, diagonal crack, spall damage and an intact wall. The best accuracy for this approach was 85.33%.

Mohan et. al [11] performs a formidable review and analysis in their work where the idea is to identify the research challenges and achievements in using image processing for crack detection by reviewing 50 different papers on the same. The paper categorizes work in the domain based on the data set used, the objectives of the study and the error levels of detection. The most commonly employed IP techniques for crack detection according to this work are morphological approaches, digital image correlations, wavelet transforms and thresholding approaches.

2.3 Integrating Hardware Support

There have also been attempts to integrate hardware support such as the use of a UAV for real-time analysis of structures. This is a more realistic scenario and has been the focus of the work by Kim et. al [12]. Using a UAV, 384 images were collected of a bridge and these were evaluated using CNNs and R-CNNs in order to inspect the bridge for structural faults.

2.4 What do these works lack in ?

As is evident from the paper reviews, the discipline of surveying structural strength and stability through IP is not devoid of work, however most work is only centered strongly around the detection and identification of cracks on walls. Also, most work performed depicts results on specific datasets that have images which are mostly taken from the same image acquisition setup with unchangeable parameters like lighting, focal length of the camera etc. While this is a great approach to train and evaluate novel techniques and algorithms, they hardly model a real world scenario. It is not necessary that all the images that are collected have to be of the same origin and share the same characteristics.

Through our project, we make an attempt at creating a model that would be able to detect and identify defects on the external surfaces of walls even with the existence of several uncontrollable, external parameters.

3. REQUIREMENT SPECIFICATION

3.1 Data Characteristics

The following were our sources of data for the project :

3.1.1 CNN Dataset

Dataset generated by Ozgenel et. al [13] that contains 40,000 images of 227 x 227 pixels with RGB Channels. There are 20,000 images with cracks and with no cracks. The dataset was collected from METU Campus Buildings.

3.1.2 Dataturks Dataset

Dataturks is a Data Labelling and Annotation tool. We were able to collect 1428 images of concrete walls (with and without cracks). The link is available as on [14]. However, there were several images that were mislabelled or had too much noise (such as wires and other occlusions). Therefore, we have used only about 400 images of this data. The process of data collection via dataturks is depicted in (Fig 3.1).

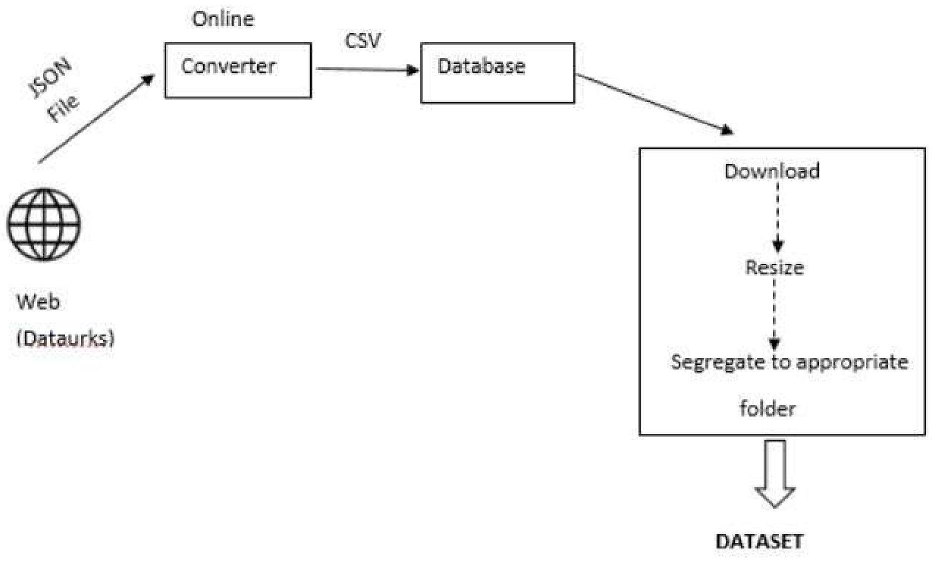


Fig 3.1

3.1.3 Google Images

We also downloaded custom images from Google Images using a custom python package [15]. The images in this dataset were taken from all over the internet and encompassed all kinds of images taken in all kinds of setups.

3.1.4 Handheld Device Collected Data

The images under this bracket include those taken using a Redmi Note 8 (48 MP camera) phone. A total of 42 images are included in this dataset (which were further segmented to 4004 images of 227x227 pixels each). The images are of cracks on old dilapidated structures and plaster defects on house walls.

3.2 Software Requirements

3.2.1 Spyder IDE

All the IP related code was written on Spyder [16]. Spyder stands for "Scientific Python Development Environment" and helps both scientists as well as developers alike.

The main Python libraries that were used were cv2 [17], numpy [18], matplotlib [19] and tkinter [20].

3.2.2 Jupyter Notebook

Jupyter notebooks [21] were used for running the deep learning-based approaches as well as conducting analysis on empirical methods.

3.2.3 Kaggle

Kaggle [22] was used as a platform that provided us with the GPU computational power for our CNN based architecture.

4. PROJECT THEORY

In this section, the background theory for the techniques that we have used in our project are discussed. In our project, we are attempting to find methods to detect and evaluate the presence of 3 major defects in buildings :

- Cracks on Walls
- Damping effects on Walls
- Plaster falling off Walls

4.1 Wall Crack Detection

Cracks are the most common type of building defects and they can occur due to several reasons that include moisture movement, corrosion of reinforcement, poor construction practices etc. [23]. Using an automated method of identifying and detecting cracks helps introduce objectivity into the process of crack detection, a feature that is lacking if manual inspection is used.

In the project, we have worked with 2 approaches to detect cracks in a building, both with their own strengths and weaknesses. In the following sections, the report will explain each of these approaches in greater depth.

4.1.1 The Pure Digital Image Processing (DIP) approach

This approach was used in order to employ traditional IP techniques for crack detection in an image. It involved the use of several basic techniques such as those explained below :

4.1.1.a) Grayscale Conversion

This is the process of converting an RGB image to a form where each pixel only provides information about the intensity of light. Since, walls of buildings can be any color, we had to perform grayscale conversion as a preliminary step to generalize our algorithms.

4.1.1.b) Thresholding

Thresholding is the process of converting an image into a form such that each pixel in the resulting image can have only one of the two possible values (white or black). It is a type of segmentation. In our project, we worked on 3 types of thresholding - Global, Adaptive and Otsu. We used thresholding to segment the cracks and noise in the images from the background.

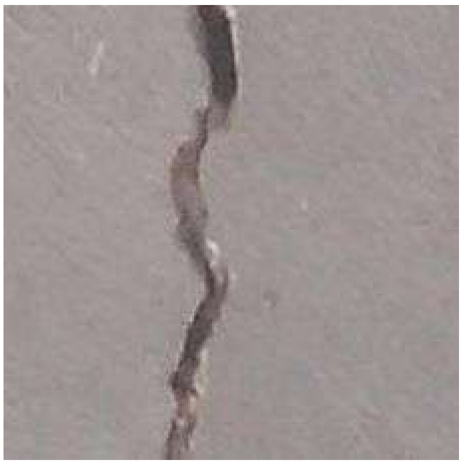


Fig 4.1 a) Original Image

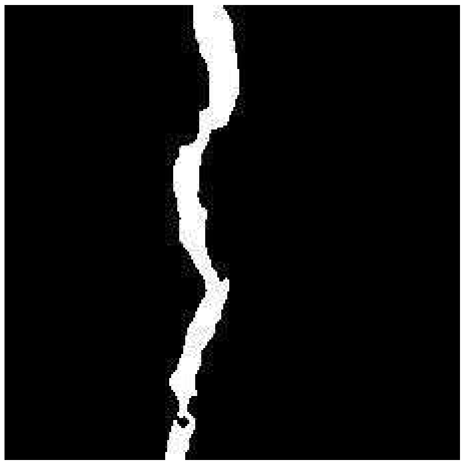


Fig 4.1 b) After Thresholding

4.1.1.c) Finding Contours

A contour is a region of continuous intensity of the same level. They are useful for object detection and in our project, we have used contours to localize and detect cracks(fig 4.2). *(Based on the assumption that most cracks have continuous edges. Even if they don't, we have tried preprocessing the images so that the boundaries of these cracks are of same intensity and so form a continuous contour)*

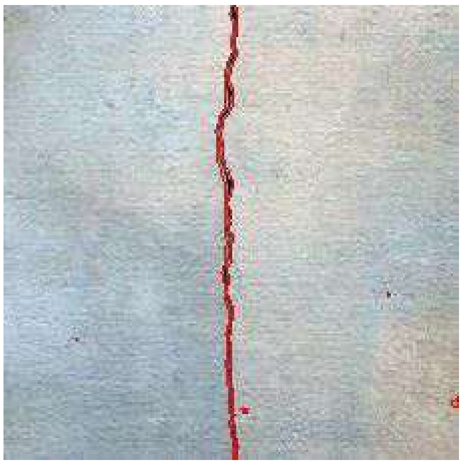


Fig 4.2 a)

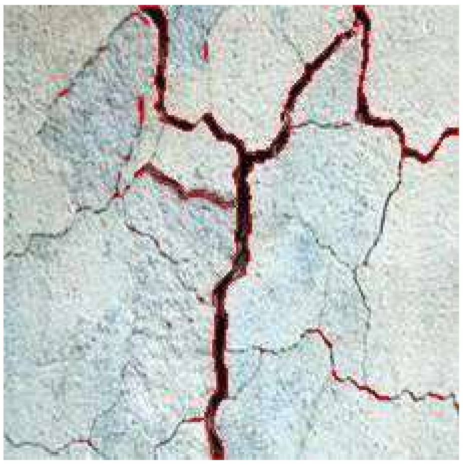


Fig 4.2 b)

4.1.1.d) Pixel Ratios

The white pixel-black pixel ratio is a simple metric that has been used in this rudimentary DIP-based approach that has been working with surprising accuracy.

$$white - to - black\ pixel\ ratio_{I} = \frac{(number\ of\ white\ pixels\ in\ I)}{(number\ of\ black\ pixels\ in\ I)} \qquad \textbf{- Equation 1}$$

4.1.2 The Deep Learning-based approach

A more sophisticated approach used was deep-learning. A Convolutional Neural Network [24] was used to classify a given image as having a crack or not.

The following explanations have been referenced from [25].

A Convolutional Neural Network (CNN) is a deep-learning approach (fig 4.3) that uses the concept of learning weights for classifying images. In a nutshell, the CNN is able to identify features in an image with as less preprocessing as required, extract these features, dimensionally reduce input to contain computational complexity and finally make classifications based on the extracted features.

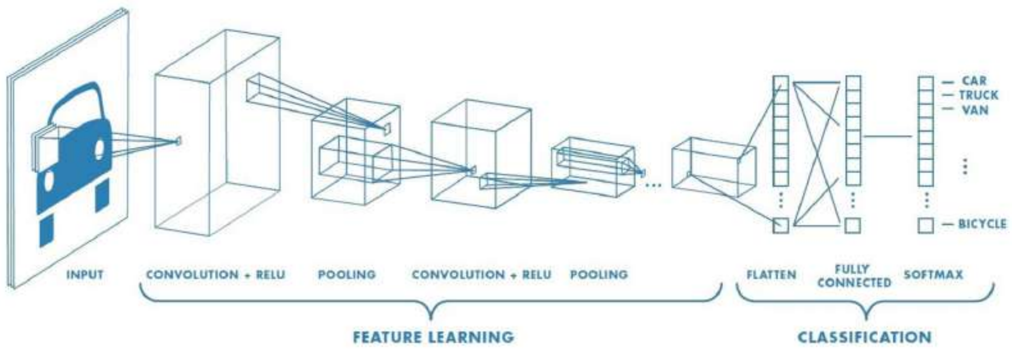


Fig 4.3

Convolutions are responsible for extracting features from the image. The initial layers of a CNN extract low-level features such as edges and as the number of layers increase, more complex features are extracted.

An ***activation function*** is used to transform the summed weights of a given node in a neural network to an output that will be passed onto the next layer. In our project, we have used ReLU (fig 4.4) as an activation function.

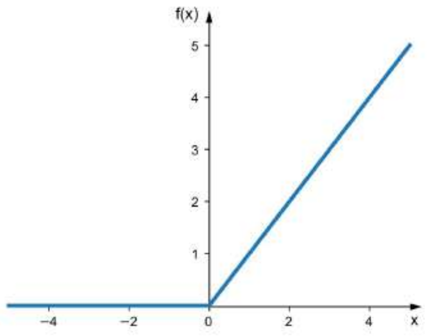


Fig 4.4

Pooling performs the task of reducing the dimensions of an input and is helpful for reducing the computational complexity involved. There are 2 types of pooling(fig 4.5) :

- Max Pooling : Picks the largest value in a window of size N x N
- Average Pooling : Picks the average of all values in a window of size N x N

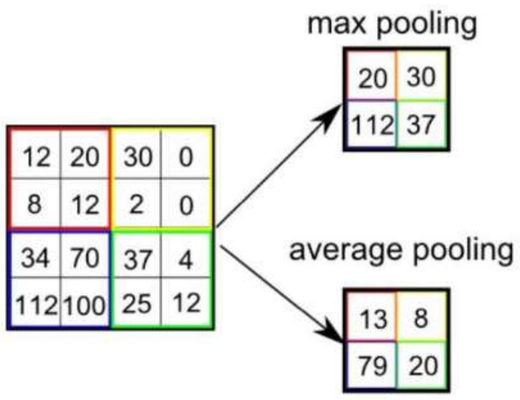


Fig 4.5

While max pooling discards noise in an image, average pooling only suppresses it. Since our images of concrete walls tend to have a high amount of noise, we have used Max Pooling.

Flattening simply converts our image to a column vector.

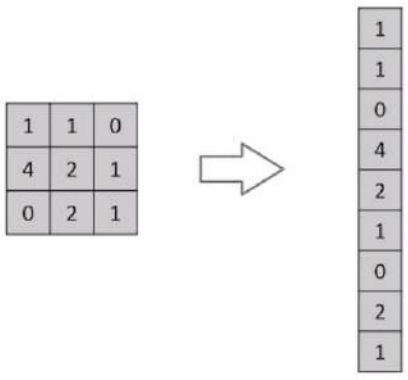


Fig 4.6

A ***fully connected layer*** learns non-linear combinations or functions in the given space. The ***Softmax function*** is just a classification technique that assigns a label to an image based on the probabilities of it belonging to a specific class. The class with the highest probability is assigned as a label to the input image.

4.2 Identifying Regions of Damping

Regions of Damping (ROD) are areas on walls that have been affected by moisture or unwanted water content. Damping leads to further problems such as fungal growth, peeling of paint layers, development of surface cracks etc.
An ROD is usually darker than the regions that have not yet been affected. So, if we are able to locate multiple regions of intensity in a wall, it can mean that there is most likely a chance for the wall to be affected by damping.

ROD can be any of the following :

- Moisture Content in wall (2 separate regions)
- Peeling of Paint
- Fungal or moss growth

Gabor Filtering has been used to identify ROD.

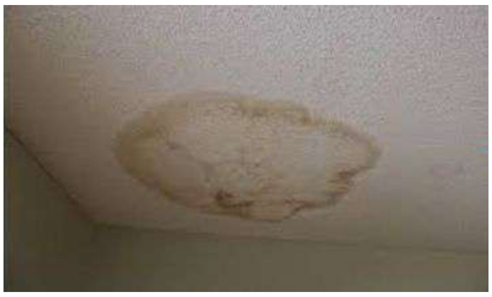


Fig 4.7 a)

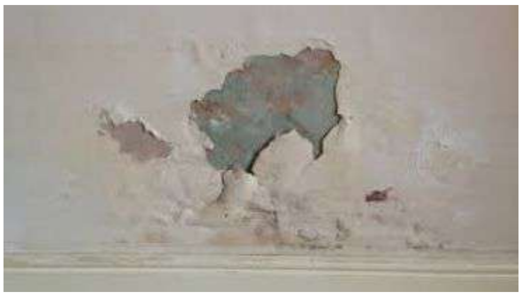


Fig 4.7 b)

4.3 Identifying Regions of Defective Plastering

Regions of Defective Plastering (RODP) are areas on walls where the plaster has fallen off due to both natural as well as man-made causes. RODP like ROD is a defect that deeply affects the aesthetic look of a building and therefore, needs to be fixed. RODP are of varied shapes and most of the time, they can be approximated to an elliptical geometry. Using this characteristic of RODP and gabor filtering , we have tried to detect RODP in an image of a wall.

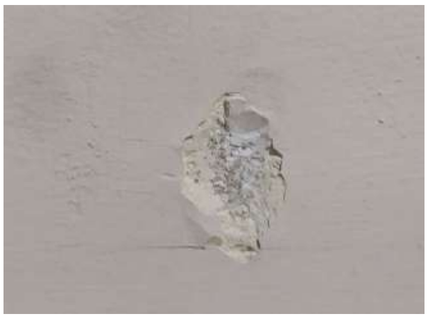


Fig 4.8 a)

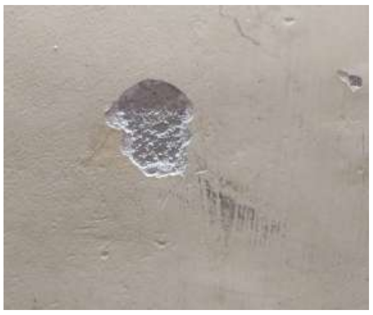


Fig 4.8 b)

5. DETAILED DESIGN AND IMPLEMENTATION

The project has been implemented in 3 phases :

- Crack Detection Phase
- ROD Detection Phase
- RODP Detection Phase

Each phase has been implemented separately independent of each other. The 3 different phases have been compared and contrasted in Table [5.1]

Table 5.1

Name of Phase	**Crack Detection**	**ROD Detection**	**RODP Detection**
Availability of Data	Available	Scarce	Scarce
Approach Used	1. DIP 2. Deep Learning	DIP	DIP
Performance	High Accuracy	Rudimentary	Rudimentary
Limitations	1. Shadow 2. Excessive Noise (Prevalent in unpainted, concrete walls) 3. Does not work for Brick Walls	1. Data Scarcity	1. Data Scarcity

5.1 Crack Detection Phase

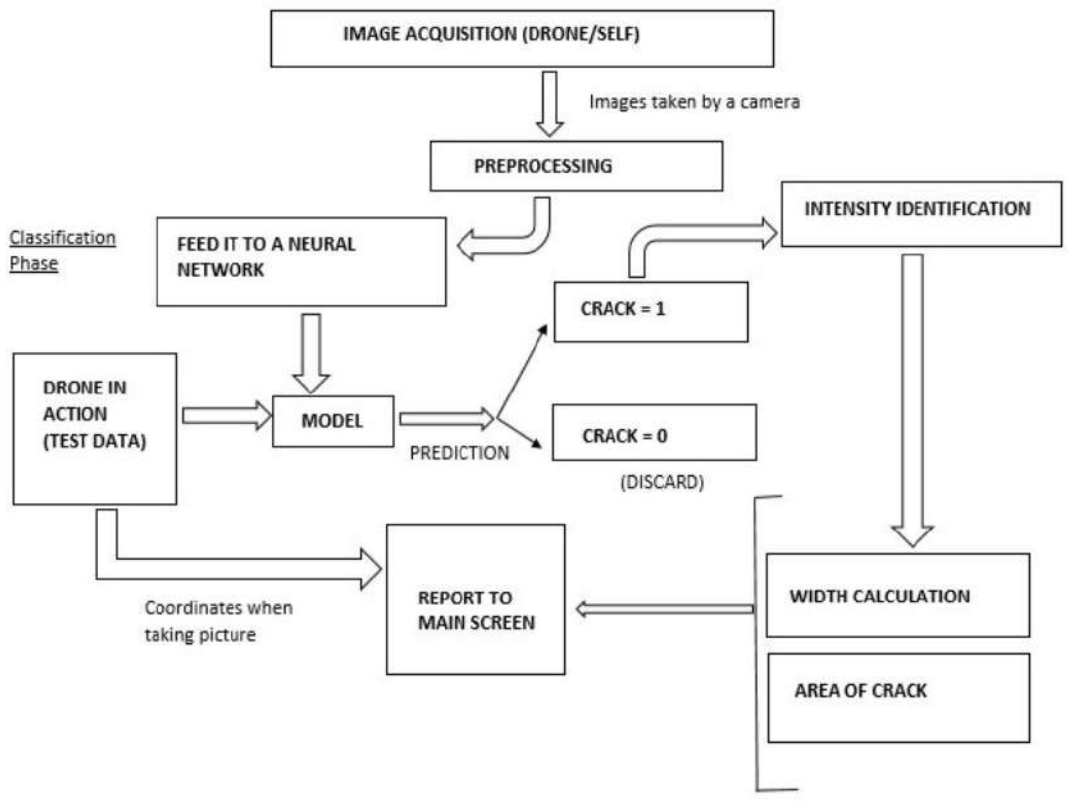


Fig 5.1

(Fig 5.1) depicts the Proposed Architecture for the System. A few components such as drone integration and intensity identification will be dealt with in the future.

5.1.1 The Pure DIP Approach (Empirical)

In this approach, we have used pure DIP techniques to make classifications. Every image is pre-processed and classified as either having a crack or no crack based on a simple metric (Equation 1). The system design for this approach is depicted in (Fig 5.2). There are 3 parts of this approach and each is discussed beneath.

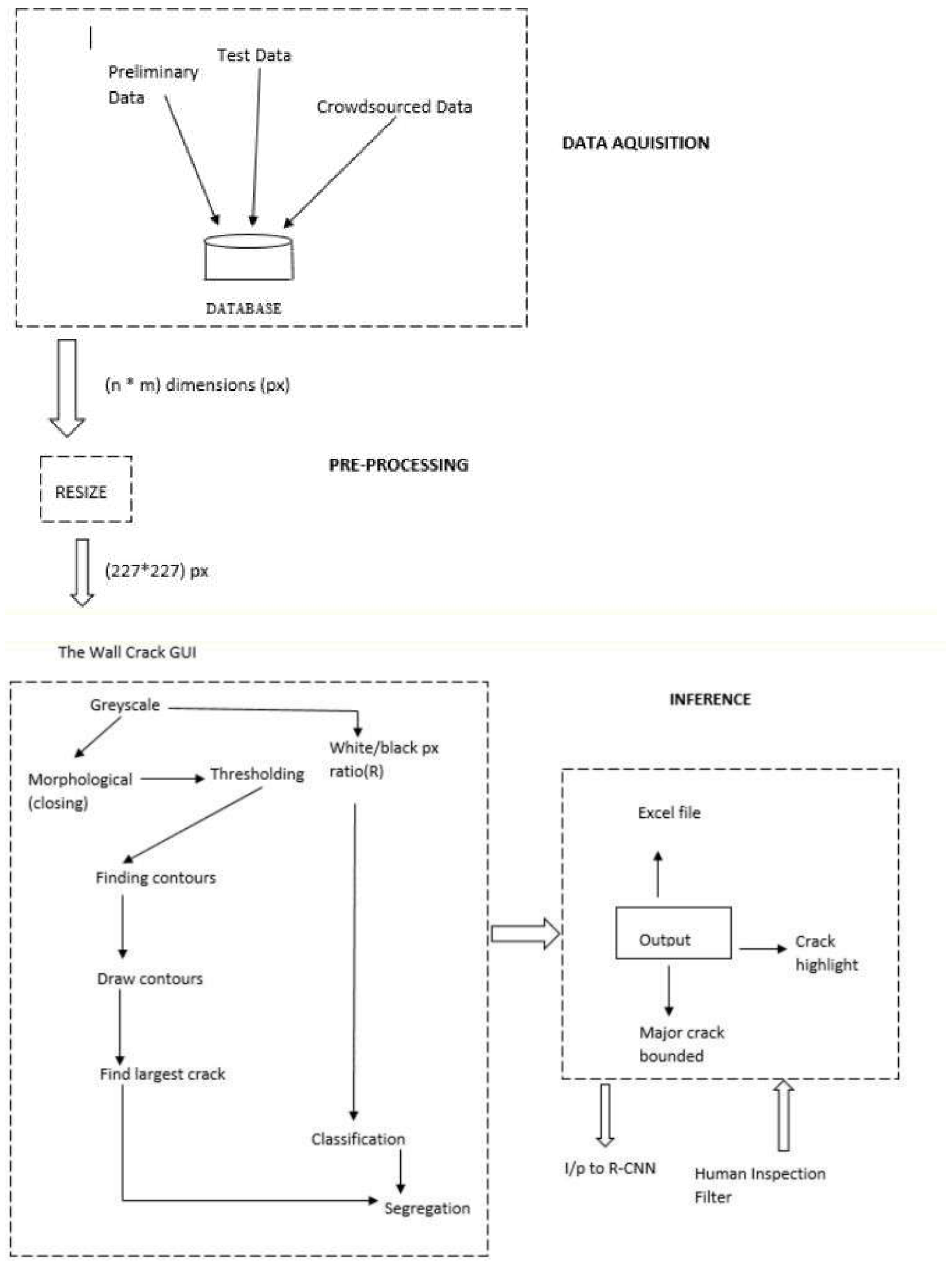

Preliminary Data
Test Data
Crowdsourced Data
DATABASE
DATA AQUISITION
(n * m) dimensions (px)
RESIZE
PRE-PROCESSING
(227*227) px
The Wall Crack GUI
Greyscale
White/black px ratio(R)
Morphological (closing)
Thresholding
Finding contours
Draw contours
Find largest crack
Classification
Segregation
INFERENCE
Excel file
Output
Crack highlight
Major crack bounded
I/p to R-CNN
Human Inspection Filter

Fig 5.2

5.1.1.a) Data Acquisition

The data we got came from various sources. All these images were stored and resized into 227 x 227 pixels. The resizing was performed to maintain uniformity in our input size for all approaches.

5.1.1.b) Pre-processing and Classification

- The image is converted to grayscale so that walls of all colours can be surveyed
- The Morphological method of closing is used to remove aberrant noisy pixels
- Thresholding(Otsu Thresholding was used) is performed as an attempt to segment the crack from the rest of the image. The crack is given white, while the background is given black. (The reason for doing so is to evaluate the crack detection problem as an object detection problem)
- Contours are drawn on the original image
- If there are multiple contours in an image, the contour that depicts the largest crack (contour with the largest area) is found out and a bounding box is drawn around it
- From the grayscale image and thresholding step, a white-to-black pixel ratio is computed
- Based on this ratio, a given image is classified as to having a crack or no crack
- Outputs are segregated

5.1.1.c) Inference

Three outputs are generated, namely an excel file (like in table 5.2), a folder with all images with the cracks highlighted and a folder with the images with cracks after drawing the bounding boxes. These can be further inputs to an R-CNN in the future. Also, it is important that there might be a need for a human inspection filter to identify the images that have been wrongly classified as having a crack.

Table 5.2

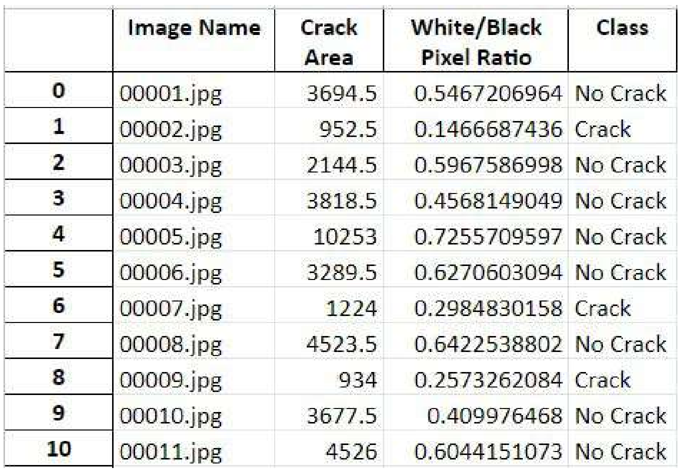


	Image Name	Crack Area	White/Black Pixel Ratio	Class
0	00001.jpg	3694.5	0.5467206964	No Crack
1	00002.jpg	952.5	0.1466687436	Crack
2	00003.jpg	2144.5	0.5967586998	No Crack
3	00004.jpg	3818.5	0.4568149049	No Crack
4	00005.jpg	10253	0.7255709597	No Crack
5	00006.jpg	3289.5	0.6270603094	No Crack
6	00007.jpg	1224	0.2984830158	Crack
7	00008.jpg	4523.5	0.6422538802	No Crack
8	00009.jpg	934	0.2573262084	Crack
9	00010.jpg	3677.5	0.409976468	No Crack
10	00011.jpg	4526	0.6044151073	No Crack

5.1.1.d) How are we making the classification without any kind of model training?

After computing the white-to-black pixel ratios on 72 images from the dataturks dataset (3.1.2), we observed a distinct pattern in these ratios. We plotted a scatterplot with the actual class (crack or no crack) of an image on the X-axis and the ratio on the Y-axis. (Fig 5.4) depicts that plot.

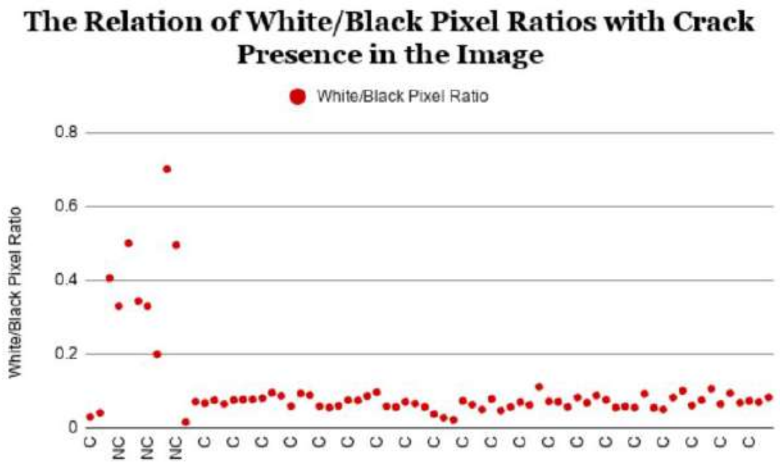


Fig 5.4 (All images not depicted in X-axis)

We could notice a general trend that showed how images with cracks had a lower white-to-black pixel ratio when compared to those without cracks. As the size of every image was uniform, this meant that *"There are more white pixels than black pixels after thresholding in images without a crack than those images where there is a crack".* Since we were not convinced, we took a few more samples from the dataset and iterated our process. But, the results were still like it was before. There was a general trend for higher white-to-black pixel ratios in images that didn't have a crack when compared to those that did. So, we introduced a ***classification threshold (CT)*** (not to be confused with "threshold", "threshold" refers to Otsu or Adaptive thresholding).

"If an image X had a white-to-black-pixel ratio > CT, X is classified as an image without a crack. Else, X has a crack. The CT we set was 0.3 and it was only based on the study of the first 72 images"

5.1.1.e) But, why does this method even work? How is a simple threshold performing so well?

The thresholding technique that we are using is an Otsu threshold. In otsu's method, a threshold is decided for an image X based on all the pixels in X. So, if there are darker pixels in an image X, then the value of this threshold will be higher than its value when there are lighter pixels. So, let's think of it like this -

- When there ***is a crack*** in an image, the threshold will be high (Because a crack is dark, it's usually almost black in most cases) - This causes a higher threshold to be set and not many pixels can qualify and that leads to only the "region around and on the crack" to be segmented well as a white region. **[This causes a low white-to-black pixel ratio]**
- When there ***is no crack*** in the image, the threshold will be low - This causes a better number of pixels to qualify and that leads to a lot of noise in the image to be segmented out as white regions. **[This causes a higher white-to-black pixel ratio]**

So, this is the reason why we are able to get not-too-bad results with this preliminary technique.

5.1.1.f) Experimental Results

Table 5.3 (Empirical Method Performance - Otsu Threshold)

Dataset	Crack Images	No Crack Images	Accuracy	False Negatives	False Positives
CNN Dataset	500	500	90.2%	1.79%	17.8%
Dataturks Dataset	63	9	97.22%	0%	22.2%
Google Images	27	5	100%	0%	0%

Below are a few examples of where our method performed well and where it did not.

Fig 5.4 a), b) and c) are examples where the approach performed well.

Fig 5.5 a), b) and c) are examples where the approach failed.

In fig 5.4 a) and b), our approach has perfectly detected the crack and classified them as cracks. In fig 5.4 c), our approach carefully bounds the largest crack in a frame with multiple cracks.

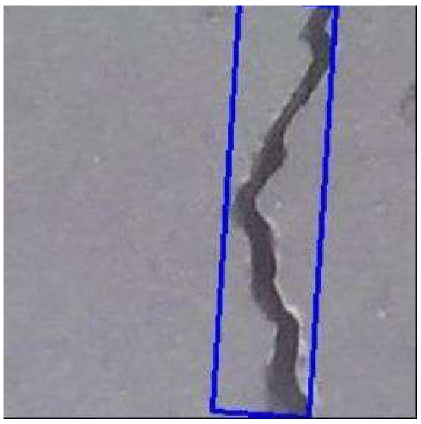


Fig 5.4 a)

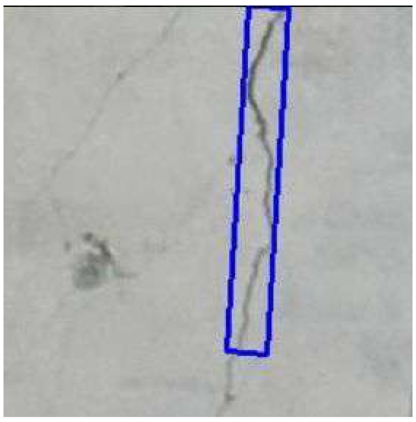


Fig 5.4 b)

Fig 5.4 c)

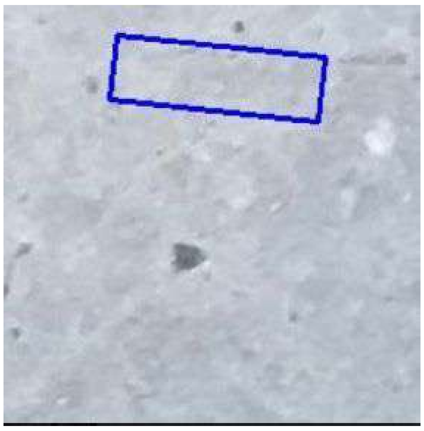


Fig 5.5 a)

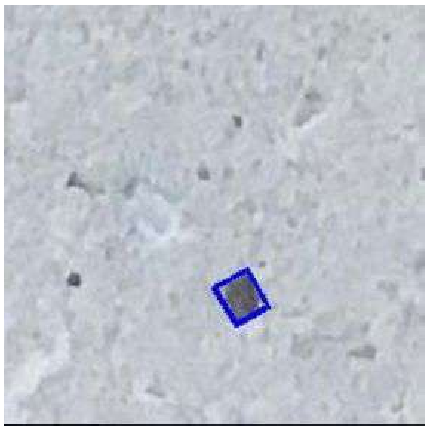


Fig 5.5 b)

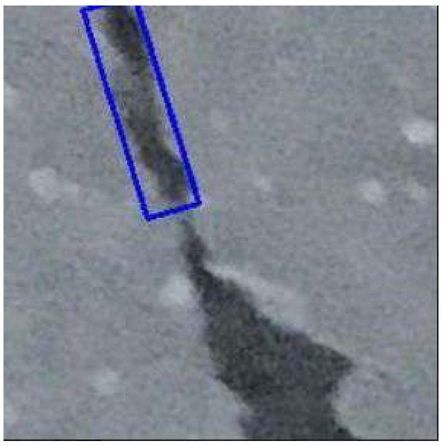


Fig 5.5 c)

In fig 5.5 a), a region that has no crack has been bounded. In fig 5.5 b), noise has been bounded and in fig 5.5 c), the larger region of the crack has not been bounded due to the poor image.

5.1.1.g) An alternative to Otsu thresholding

It is also to be noted that the Otsu threshold has a major drawback. It does not give good results when there is a shadow cast on the image or if the image has a washed out appearance. In such a case, adaptive thresholding is what makes more sense. Adaptive thresholding uses local thresholding and performs its operations on each local neighbourhood. It works really well in images that have shadows.

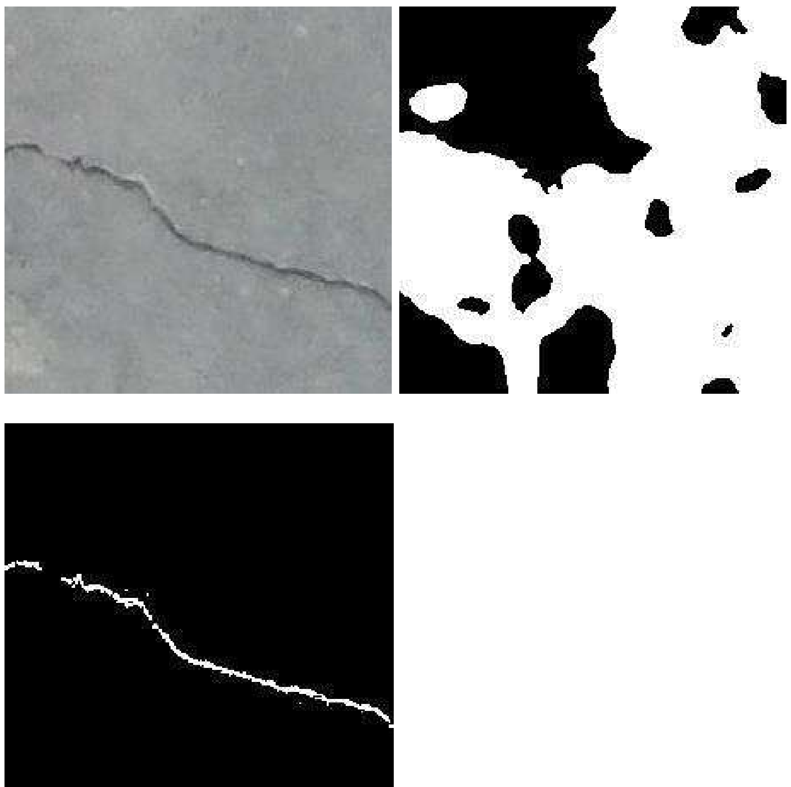


Fig 5.6 a), b), c) [Clockwise Order]

Fig 5.6 a) is the original image. Fig 5.6 b) and c) are the Otsu thresholded and adaptive thresholded images respectively. The difference is large!

So, we modified our approach to use adaptive thresholding instead of Otsu's method. In this case, we noticed a new pattern.

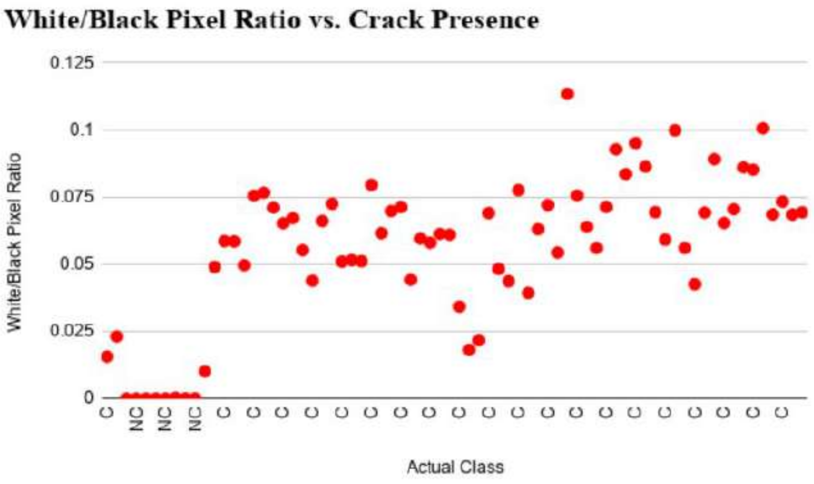


Fig 5.7

The white-to-black pixel ratio was really really low for images with no cracks. This definitely makes sense as the adaptive thresholding technique was thresholding to detect the cracks and not the noise too unlike Otsu's method. So, *"If an image X had a white-to-black-pixel ratio < CT, X is classified as an image with a crack. Else, X does not have a crack. The CT we set was 0.01 and it was only based on the study of the first 72 images"*

Table 5.4 (Empirical Method Performance - Adaptive Threshold)

Dataset	**Crack Images**	**No Crack Images**	**Accuracy**	**False Negatives**	**False Positives**
CNN Dataset	500	500	86.9%	0%	26.2%
Dataturks Dataset	63	9	98.41%	1.59%	11%
Google Images	27	5	100%	0%	0%

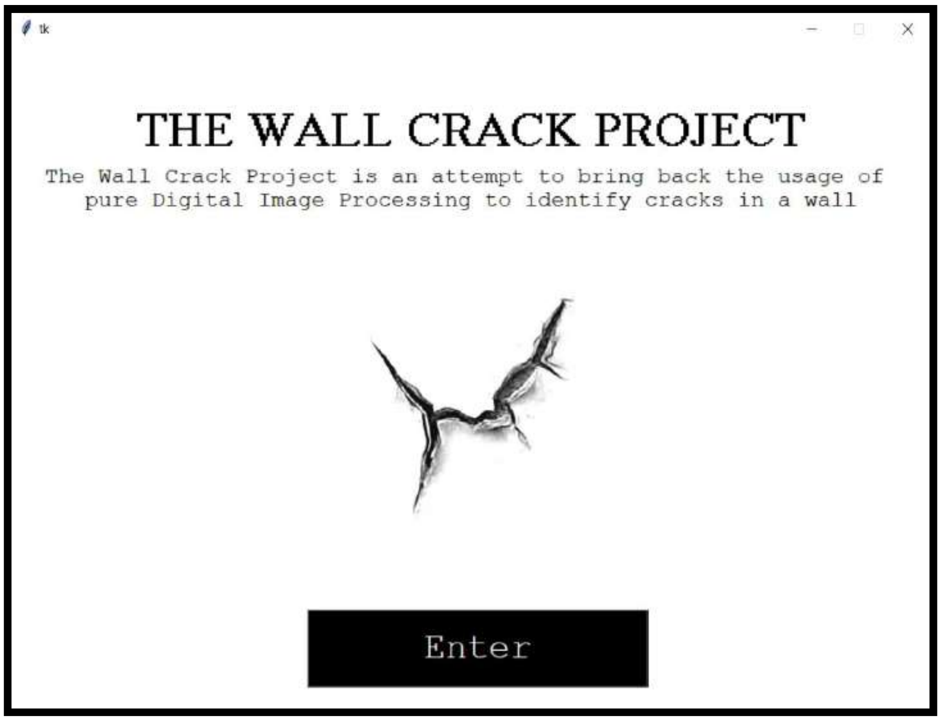
5.1.1.h) Limitations to the Approach

- The threshold method has worked really well for all the multiple variations of images we have tried. But there is still a need to see how this works in a production system.
- Accuracy is not as high as the CNN method (explained next)
- Can’t work if there is noise in the images like doors, hinges, wires etc.
- Does not work for brick walls

5.1.1.i) The Pros of the Approach

- No need for model training
- Simple to understand and apply
- Faster to compute than CNN (This approach takes around 0.2 seconds per image)

5.1.1.j) The GUI Implementation


THE WALL CRACK PROJECT

The Wall Crack Project is an attempt to bring back the usage of pure Digital Image Processing to identify cracks in a wall

Fig 5.8 a)

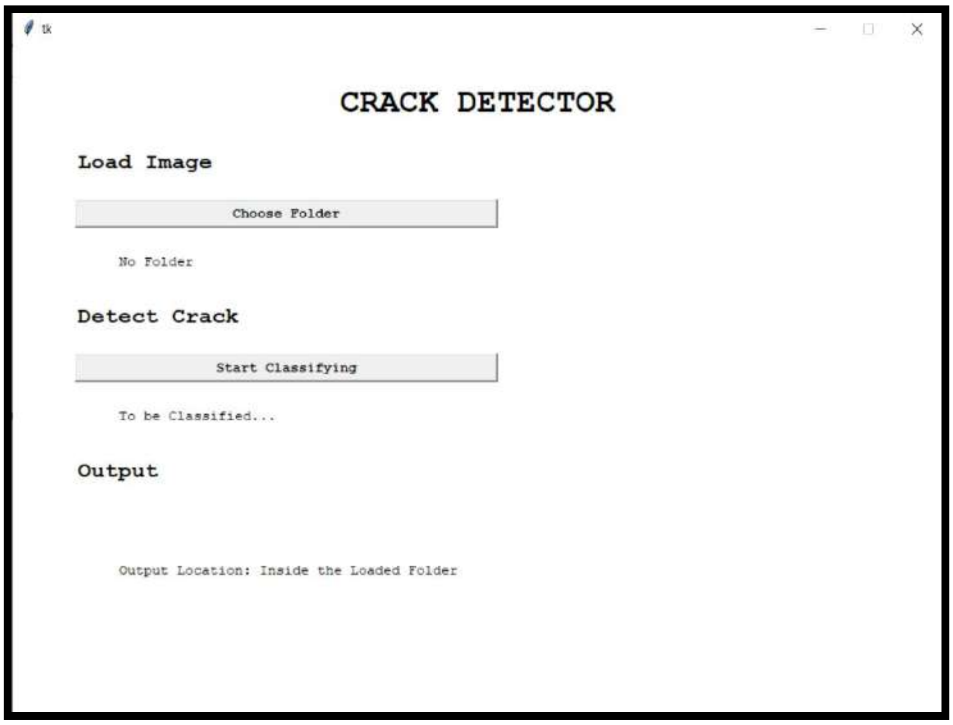


Fig 5.8 b)

Fig 5.8 a) and b) depict the basic GUI that was created as rudimentary software for performing classifications using the DIP based approach.

Tkinter was used to make the GUI.

5.1.2 The Deep learning-based Approach

In this approach we have used a CNN architecture (fig 5.9) to classify images.

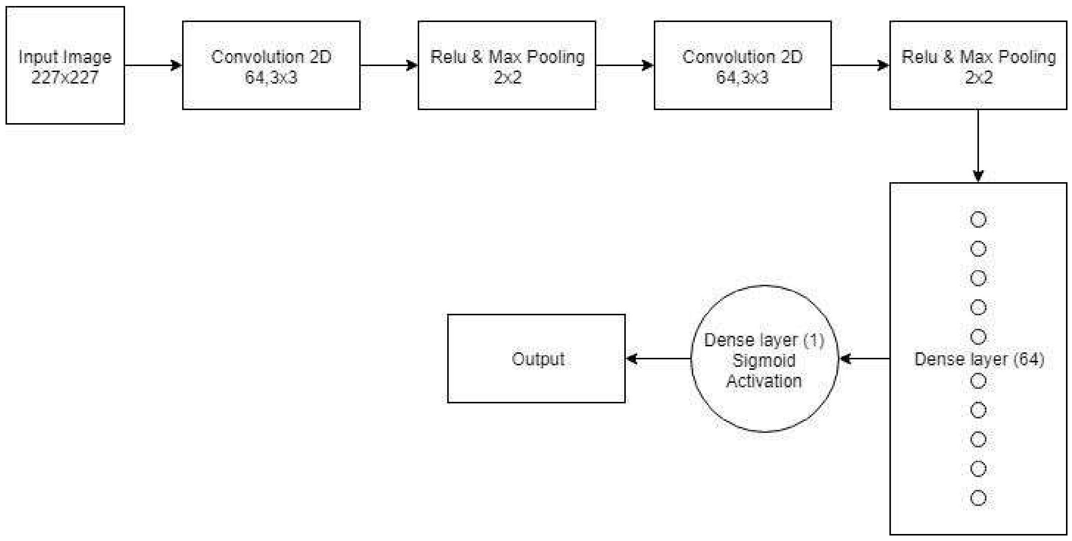


Fig 5.9

5.1.2.a) Training Data Specifics

Origin : CNN Dataset (3.1.1)
Number of Images : 10,000
Dimension of Images : 227 x 227
Number of Epochs : 5
Validation Split : 10%

5.1.2.b) Architecture

1. Convolutional layer - 3x3 (Activation function: Relu)
2. Pooling layer - 2x2
3. Convolutional layer - 3x3 (Activation function: Relu)
4. Pooling layer - 2x2
5. Flatten Function
6. Dense layer - 64
7. Dense layer - 1
8. Activation function: Sigmoid
9. Optimizer used- Adam

Table [5.5] shows the results of our CNN architecture where we have provided images as they are i.e without any kind of preprocessing. In table [5.6], the results are depicted for the case when the input images were pre-processed before being fed into the CNN.

5.1.2.c) Evaluation and Results

Table 5.5 (CNN Performance in Image Classification - No preprocessing)

Dataset	Number of Images	Type of Images	Accuracy
Validation Set	1000	Concrete Walls	99.1%
Google Images	31	Concrete Walls	100%
Handheld Device	197	All kinds	33.5%

Table 5.6 (CNN Performance in Image Classification - With preprocessing)

Dataset	Number of Images	Type of Images	Accuracy
Validation Set	1000	Concrete Walls	99.89%
Google Images	31	Concrete Walls	60%
Handheld Device	197	All kinds	49%

5.1.2.d) What was the preprocessing performed ?

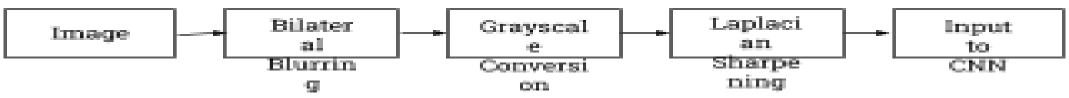


Fig 5.10

There were certain images that were not clear and therefore we tried to preprocess those images and pass them as input for the CNN. Bilateral Filtering is a smoothing technique that helps blur out the noise. Bilateral filtering was preferred over conventional techniques like gaussian smoothing and median filtering as bilateral filtering performs better around the edges when compared to the other two.
Laplacian sharpening was used to distinctively highlight the crack from its surroundings.

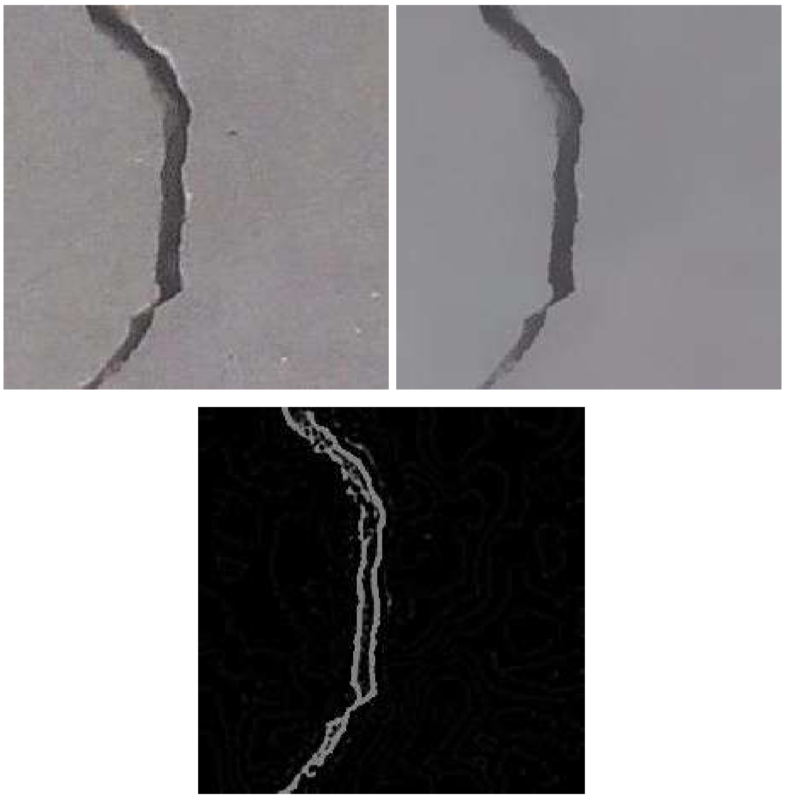


Fig 5.11 a), b), c) [Clockwise from top left]

Fig 5.11 a) depicts the original image. Fig 5.11 b) shows the blurred image and fig 5.11 c) shows the sharpened image

While the preprocessing helped increase the accuracy significantly with the validation set, it dropped CNN's accuracy by 40% when used on a dataset of images from the web.

Images from the handheld device have not been classified well by both CNN approaches.

5.1.2.e) Limitations to the Approach

- Requires a large number of training data
- High time complexity
- Does not work for brick walls
- Does not do well when posed with input that is completely deviant from what it has seen before

5.1.2.f) The Pros of the Approach

- Very High Accuracy (even without any kind of preprocessing)
- Unlike the empirical approach, the deep learning approach is definite to work on a larger spectrum of data

5.1.3 The Pure Empirical DIP-based approach beats the CNN

When both the approaches were compared on the handheld device images dataset, the results showed an empirical DIP approach climbing way higher than the invincible CNN when it came to classification of cracks.
The sample dataset used had 118 images for testing (109 cracks and 9 no-cracks). We called this dataset ***"The Decider"***. All the images were captured using a mobile phone. Each was 227x227 pixels and about 6 KB in size. These 118 images were picked randomly from the 4004 images of handheld device data.

NOTE : The dataset did not contain images with occlusions or noise like wires, doors, windows, hinges, plants etc.

Table 5.7 (Evaluation of both approaches on the Decider)

Approach	Accuracy
CNN - Without preprocessing	33.5%
CNN - With preprocessing	49%
DIP-based - Otsu	93.22%
DIP-based - Adaptive	85.59%

[Table 5.7] describes a clear victory for the DIP-based approach over CNN.

5.2 Region of Damping (ROD) Detection Phase

The work done in ROD is not in as much depth as crack detection, and this was majorly because we had severe data shortage. Yet, with the available data, we have been able to formulate a reasonable approach.

We have used Gabor filters as gabor filters help identify texture. And when there is a damping defect, the problem often is a new change in texture of the surface. We have made use of this very cause and worked on it.

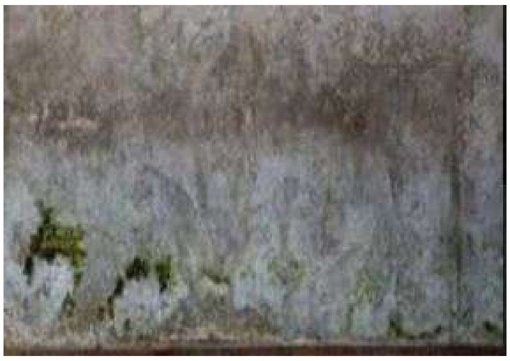


Fig 5.12 a) Original Image

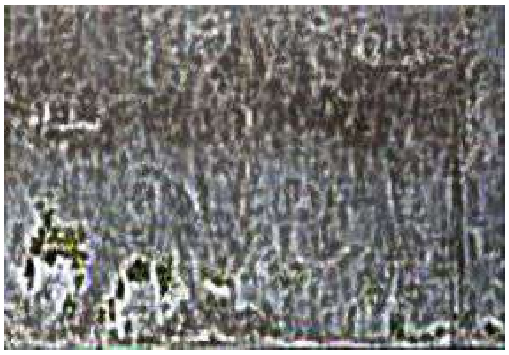


Fig 5.12 b) After Gabor Filtering

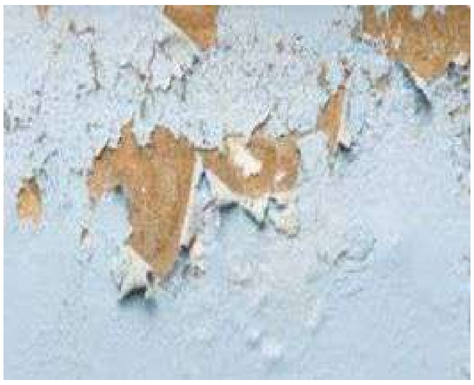


Fig 5.12 a) Original Image

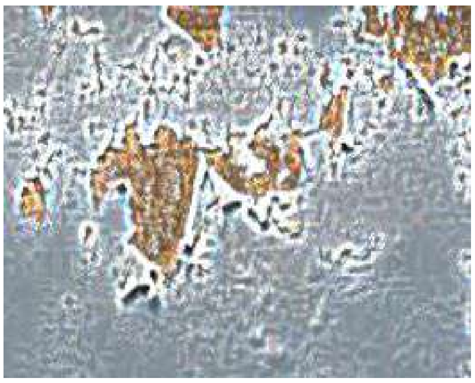


Fig 5.12 b) After Gabor Filtering

5.3 Region of Defective Plastering (RODP) Detection Phase

The work done in RODP has also been affected significantly by lack of data. Yet, we have tried to work with this constraint and have discovered that the gabor filter can help identify this too.

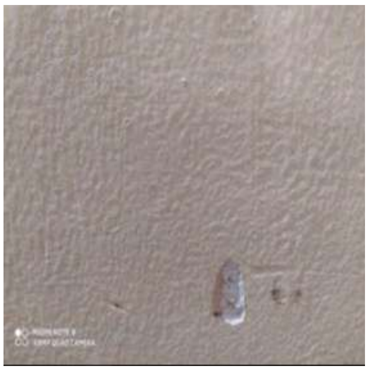


Fig 5.13 a) Defected Image

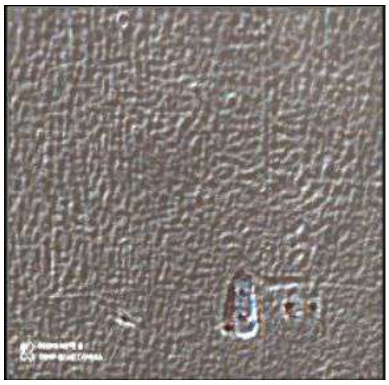


Fig 5.13 a) Defect Highlighted

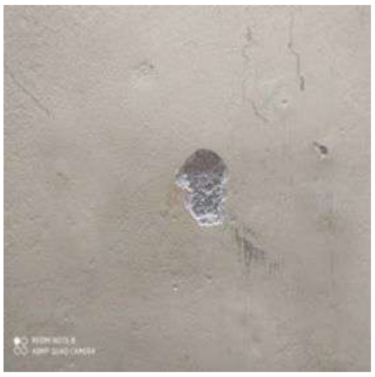


Fig 5.14 a) Defected Image

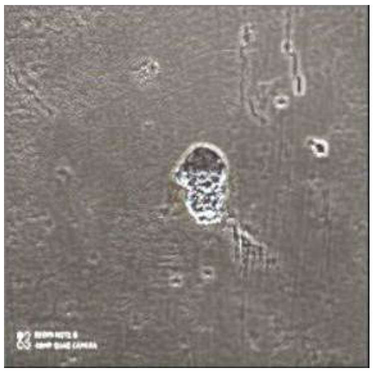


Fig 5.14 a) Defect Highlighted

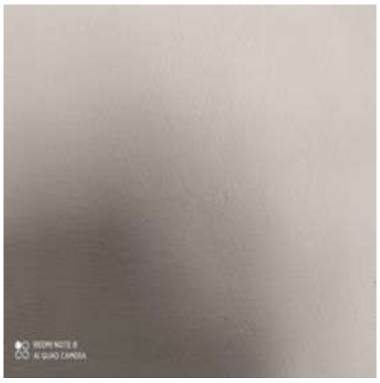


Fig 5.15 a) No-defect Image

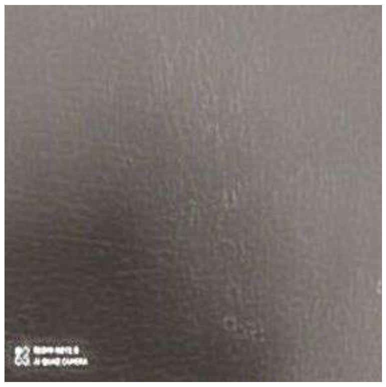


Fig 5.15 a) Nothing is highlighted

Fig 5.16 a) No-defect Image

Fig 5.16 a) Nothing is highlighted

The output of the gabor filter can be fed into a CNN (if we have loads of data) and we can use that approach to see if we can classify images as having an RODP or not.
Furthermore, “shape analysis” can be used to analyse the shape of the highlighted defect. And this can be used to calculate area. The approach in the empirical DIP-based method for crack detection will work well here too.

NOTE : We are using a gabor filter because in that way we will be able to use our method on any wall of any colour.

6. CODE

6.1 Crack Detection Code

6.1.1 Empirical Approach

```
"""
Task :
> Images are to be classified as cracks or no cracks (based on pure
image processing)
> Output an excel file to understand white/black pixel ratios

Metrics :
> Classification Accuracy
> False Positives
> False Negatives
"""

# Importing libraries
----------------------------------------------

from tkinter import *
import tkinter as tk
from tkinter import font  as tkfont
from tkinter import filedialog
from PIL import Image, ImageTk

import os
import sys
import cv2
import numpy as np
import matplotlib.pyplot as plt

import pandas as pd
import xlsxwriter
```

```
# Code for the GUI
-------------------------------------------------

class CrackApp(tk.Tk):

    def __init__(self, *args, **kwargs):
        tk.Tk.__init__(self, *args, **kwargs)
        '''Font of Title in Start Page'''
        self.title_font = tkfont.Font(family='Helvetica', size=48,
weight="bold")

        # Container into which the frames go

        container = tk.Frame(self)
        container.pack(side="top", fill="both", expand=True)
        container.grid_rowconfigure(0, weight=1)
        container.grid_columnconfigure(0, weight=1)

        self.frames={}
        for F in (HomePage, CrackDetector):
            page_name = F.__name__
            frame = F(parent=container, controller=self)
            self.frames[page_name] = frame

            # put all of the pages in the same location;
            # the one on the top of the stacking order
            # will be the one that is visible.
            frame.grid(row=0, column=0, sticky="nsew")

        # Start from the Home Page
        self.show_frame("HomePage")

    def show_frame(self,page_name):
        '''Show the Page for the given frame'''
        frame = self.frames[page_name]
        frame.tkraise()

''' The Structure of the Home Page '''
class HomePage(tk.Frame):

    def __init__(self, parent, controller):

        tk.Frame.__init__(self, parent, bg='#ffffff') # setting
background to white
        self.controller = controller

        title = Label(self, text="\nTHE WALL CRACK PROJECT",
font=('ms serif',36,"bold"), bg="#ffffff")
        title.pack()

        subtitle = Label(self, text="The Wall Crack Project is an
attempt to bring back the usage of \n pure Digital Image
Processing"
                     " to identify cracks in a wall",
font=('courier new',16), bg="#ffffff")
        subtitle.pack()

        icon = PhotoImage(file="images\home_logo.png")
        image_label = Label(self, image=icon, bg="#ffffff")
        image_label.image = icon
        image_label.place(x=330, y=220)



        enter = Button(self, text='Enter', font=('courier new',
26),
                              bg='black',
                              fg='white', padx=5, pady=5, width=15,
                              command=lambda:
controller.show_frame("CrackDetector"))

        enter.place(x=290,y=550)
```

```
''' The Structure of the Crack Detector Page '''
class CrackDetector(tk.Frame):

    def __init__(self, parent, controller):

        tk.Frame.__init__(self, parent, bg='#ffffff') # setting 
background to white

        # Initial values
        self.controller = controller
        self.folder=None

        # Loads the folder with images
        def load_folder():
            self.folder = filedialog.askdirectory()
            print(self.folder+" has been loaded")
            load_info["text"]="Folder has been loaded"  # changing 
the status

        # Function to find the largest contour area
        def findGreatestContour(contours):
            areas = []
            largest_area = 0
            largest_contour_index = -1
            i = 0
            total_contours = len(contours)
            while (i < total_contours ):
                area = cv2.contourArea(contours[i])
                areas.append(area)
                if(area > largest_area):
                    largest_area = area
                    largest_contour_index = i
                i+=1
            return largest_area, largest_contour_index, areas
        # Function to find and return pre-processed images from an
input image
        def image_derivatives(img):
            # Convert to gray
            imgray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

            # Morphological Processing - Erosion
            se =
cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(15,15))
            dilation = cv2.dilate(imgray,se,iterations = 2)
            erosion = cv2.erode(dilation,se,iterations = 2)
            final_img = erosion-imgray

            # Otsu's Thresholding
            ''' Global threshold is used to binarize the image;
works on individual images and is much more efficient than local
binarization
            '''
            blur = cv2.GaussianBlur(final_img,(5,5),0)
            theta,otsu =
cv2.threshold(blur,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
            binarized = otsu # In OpenCV object detection,
object=white & background=black

            # Finding Contours
            ''' A contour is a curve joining all the continuous
points(along the boundary), having same color/intensity '''
            image, contours, hierarchy =
cv2.findContours(binarized,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)

            # Making a copy as drawContours manipulates the
original
            img_copy = img.copy()

            # Drawing Contours
            crack_highlight = cv2.drawContours(img, contours, -1,
(255,0,0), 1) # The last param is the width of the contour (adjust
it if you need)

            # Finding largest crack
            largest_area, largest_contour_index, areas =
findGreatestContour(contours)
            cnt = contours[largest_contour_index]
            rect = cv2.minAreaRect(cnt)
            box = cv2.boxPoints(rect)
            box = np.int0(box)
            bounding_box =
cv2.drawContours(img_copy,[box],0,(0,0,255),2)

            # returns the final image with the thresholded image
and the cracks highlighted image
            return (binarized, crack_highlight, bounding_box,
largest_area)

        # Function to return white/black pixel ratio
        def white_black_ratio(img):
            pixel_vals={"white":0, "black":0}
            for i in range(227):
                for j in range(227):
                    if(img[i][j]==0):
                        pixel_vals["black"]+=1
                    else:
                        pixel_vals["white"]+=1
            # return white/black pixel ratio
            return (pixel_vals["white"]/pixel_vals["black"])

        # Classify as crack or no-crack
        def crack_or_no_crack(img):
            if(white_black_ratio(img)<0.3):
                return "Crack"
            else:
                return "No Crack"

        # Crack Detector Function
        def crack_detector():
            dirs = os.listdir(self.folder)

            # Creating the dictionary for the output excel file
            image_names = []
            crack_areas = []
            white_by_black_ratios = []
            image_class = []

            # Creating the new output folders
            crack_highlight_dir = self.folder + "/crack_highlight"
            try:
                # Create target Directory
                os.mkdir(crack_highlight_dir)
                print("Directory " , crack_highlight_dir ,  "
Created ")
            except FileExistsError:
                print("Directory " , crack_highlight_dir ,  "
already exists")

            major_crack_dir = self.folder + "/major_crack"
            try:
                # Create target Directory
                os.mkdir(major_crack_dir)
                print("Directory " , major_crack_dir ,  " Created
")
            except FileExistsError:
                print("Directory " , major_crack_dir ,  " already
exists")

            # Making the DIP based classification
            try:
                for item in dirs:
                    full_path = self.folder+'/'+item
                    # append image name
                    image_names.append(item)
                    print(full_path)
                    if(os.path.isfile(full_path)):
                        img = cv2.imread(full_path)
                        thresholded, highlight, major_crack,
major_crack_area = image_derivatives(img)
                        high = Image.fromarray(highlight)
                        major = Image.fromarray(major_crack)
                        # append image area
                        crack_areas.append(major_crack_area)
                        # append black/white pixel ratio

white_by_black_ratios.append(white_black_ratio(thresholded))
                        # append classification of image

image_class.append(crack_or_no_crack(thresholded))

                        # store the cracks' highlighted versions

if(crack_or_no_crack(thresholded)=="Crack"):
                            high.save(crack_highlight_dir+'/'+item)
                            major.save(major_crack_dir+'/'+item)
            except:
                print("Folder has not been loaded!")

            # Creating the excel file output
            data = {"Image Name":image_names, "Crack
Area":crack_areas, "White/Black Pixel Ratio":white_by_black_ratios,
                    "Class":image_class}
            df = pd.DataFrame(data)
            # Create a Pandas Excel writer using XlsxWriter as the
engine.
```

```
            writer =
pd.ExcelWriter(self.folder+'/'+'Test_output.xlsx',
engine='xlsxwriter')
            # Convert the dataframe to an XlsxWriter Excel object.
            df.to_excel(writer, sheet_name='Output_sheet')
            # Close the Pandas Excel writer and output the Excel
file.
            writer.save()

            class_info["text"]="Classification Complete"  #
changing the status

# ----------- Outlining a few things -------------

        title = Label(self, text="\nCRACK DETECTOR", font=('courier
new',24,"bold"), bg="#ffffff")
        title.pack()

        # Load the folder of images
        load_txt = Label(self, text="Load Image", font=('courier
new',16,"bold"), bg="#ffffff")
        load_txt.place(x=60, y=100)
        load_select = Button(self, text="Choose Folder",
font=('courier new',10,"bold"), width=50, command=load_folder)
        load_select.place(x=60, y=150)

        # Folder Status
        load_info = Label(self, text="No Folder", font=('courier
new', 10), bg="#ffffff")
        load_info.place(x=100, y=200)

        # Detect the Crack
        detect_txt = Label(self, text="Detect Crack",
font=('courier new',16,"bold"), bg="#ffffff")
        detect_txt.place(x=60, y=250)
        detect_select = Button(self, text="Start Classifying",
font=('courier new',10,"bold"), width=50, command=crack_detector)
        detect_select.place(x=60, y=300)

        # Classification Status
        class_info = Label(self, text="To be Classified...",
font=('courier new', 10), bg="#ffffff")
        class_info.place(x=100, y=350)

        # Output File
        output_file = Label(self, text="Output", font=('courier
new',16,"bold"), bg="#ffffff")
        output_file.place(x=60, y=400)
        '''
        output_download = Button(self, text="Download Excel File",
font=('courier new',10,"bold"), width=50)
        output_download.place(x=60, y=450)
        '''

        # Output Info
        output_info = Label(self, text="Output Location: Inside the
Loaded Folder", font=('courier new', 10), bg="#ffffff")
        output_info.place(x=100, y=500)

# Run the GUI
------------------------------------------------------
if __name__ == "__main__":
    app = CrackApp()
    # to set the dimensions of the window
    app.geometry("900x650")
    app.resizable(width=False, height=False)
    app.configure(background='white')
    app.mainloop()
```

6.1.2 Deep Learning Approach

```
"""
Task :
> Images are to be classified as cracks or no cracks (based on CNN
architecture)

Metrics :
> Classification Accuracy
"""

import pickle
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation,
Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D

pickle_in = open("X.pickle","rb")
X = pickle.load(pickle_in)

pickle_in = open("y.pickle","rb")
y = pickle.load(pickle_in)
X = X/255.0
#print(X.shape)
X=X.reshape([10000, 227, 227,1])
#print(X.shape)

# The CNN
import os
checkpoint_path = "cp.ckpt"
checkpoint_dir = os.path.dirname(checkpoint_path)

# Create checkpoint callback
cp_callback = tf.keras.callbacks.ModelCheckpoint(checkpoint_path,
save_weights_only=True,                                verbose=1)
model = Sequential()
```

```
model.add(Conv2D(64,(3,3),input_shape=(227,227,1)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64,(3,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())

model.add(Dense(64))

model.add(Dense(1))
model.add(Activation('sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
print(X.shape)
model.fit(X, y, batch_size=32, epochs=1, validation_split=0.1)

# Testing
import cv2
custtest = []
path = "customtest/"
def create_training_data():
    #path = os.path.join(DATADIR,category)
    print(path)
    print(os.listdir(path))
    for img in os.listdir(path):
        img_array = cv2.imread(os.path.join(path,img)
,cv2.IMREAD_GRAYSCALE)
        custtest.append([img_array])

create_training_data()

print(len(custtest))
#custtest[2]
```

```
import numpy as np

X = np.array(custtest)
X = X/255.0
X1 = X.reshape([120,1,227, 227,1])
#X1[1
n=0
tot = 120
cnt = 0
print("Cracked: 0")
print("Not Cracked: 1")
for i in X1:
    n+=1
    if model.predict(i)[0]>0.5:
        cnt += 1  #Number of non cracked concrete images
    print("Image number",n,model.predict(i))
acc = (cnt)/tot
print("Accuracy ", acc)
```

6.2 ROD Detection Code and RODP Detection Code (Gabor filter)

```
"""
Created on Fri Dec 27 11:27:57 2019

Gabor Filter Code
"""

import cv2
import numpy as np
import matplotlib.pyplot as plt

def build_filters():
 filters = []
 ksize = 31
 for theta in np.arange(0, np.pi, np.pi / 16):
     kern = cv2.getGaborKernel((ksize, ksize), 3.0, theta, 8.0,
30.0, 0, ktype=cv2.CV_32F)
     kern /= 1.5*kern.sum()
     filters.append(kern)
 return filters

def process(img, filters):
 accum = np.zeros_like(img)
 for kern in filters:
     fimg = cv2.filter2D(img, cv2.CV_8UC3, kern)
     np.maximum(accum, fimg, accum)
 return accum

if _name_ == '_main_':
 import sys

 try:
     img_fn = sys.argv[1]
 except:
     img_fn = "C:\\Users\\Ramshankar Yadhunath\\Desktop\\Detecting
Building Defects\\Paint  Peeling (From home)\\Paint  Peeling (From
home)\\Defect\\6.jpg"

 img = cv2.imread(img_fn)
 if img is None:
     print ('Failed to load image file:'), img_fn
     sys.exit(1)

 filters = build_filters()

 res1 = process(img, filters)
 cv2.imshow('result', res1)
 cv2.waitKey(0)
 cv2.destroyAllWindows()
```

```
#im = cv2.imread("C:\\Users\\Ramshankar
Yadhunath\\Desktop\\Detecting Building Defects\\Concrete Crack
Images for
Classification\\TheWallCrack_GUI_Test_Set_Positive\\00096.jpg")
#plt.imshow(im, cmap="gray")
```

7. CONCLUSION AND FUTURE ENHANCEMENT

The “Building Defect Detection” project has been a good learning opportunity for the team. In our work, we have used multiple techniques to solve our problem and have looked beyond traditional methods for identifying visible defects in buildings. Our work has taken into consideration an empirical approach for detecting cracks which has proven to work with reasonable accuracy across images taken by various setups. We have also ventured into detecting other defects in an image though there is a lot more work required to convert these to production ready modules.

7.1 Future Scope of Work

The following are expected to be included in the future scope of work :

- Use one-shot learning or zero-shot learning to be able to classify defects easily without having too much training data
- Automate the ability of finding the threshold in the empirical method for crack detection
- Find better ways to classify cracks based on their intensity
- Collect and create a dataset of images that have other defects on buildings as there are almost no datasets like this. Almost all only have cracks.
- It can be noticed how pure DIP has produced some wonderful results. So, trying out more techniques such as sharpening before thresholding and other related techniques will definitely aid in better results. Maybe even produce a model that outperforms CNN!
- Integrating our model with hardware like a UAV will be an important step to be undertaken so that we can automate the whole process of detecting building damage

8. REFERENCES

1. Pēteris Druķis, Līga Gaile, Leonīds Pakrastiņš, “Inspection of Public Buildings Based on Risk Assessment”, Procedia Engineering, Volume 172, 2017, Pages 247-255, ISSN 1877-7058, https://doi.org/10.1016/j.proeng.2017.02.106. (http://www.sciencedirect.com/science/article/pii/S1877705817306124)
2. Nama, Pooja, et al. “Study on causes of cracks & its preventive measures in concrete structures.” International Journal of Engineering Research and Applications 5.5 (2015): 119-123.
3. Ahzahar, N., et al. "A study of contribution factors to building failures and defects in construction industry." Procedia Engineering 20 (2011): 249-255.
4. Sommerville, James. "Defects and rework in new build: an analysis of the phenomenon and drivers." Structural Survey 25.5 (2007): 391-407.
5. Bakri, Nurul Nadia Omar, and Md Azree Othuman Mydin. "General building defects: causes, symptoms and remedial work." European Journal of Technology and Design 1 (2014): 4-17.
6. Georgiou, J. (2010), "Verification of a building defect classification system for housing", Structural Survey, Vol. 28 No. 5, pp. 370-383, https://doi.org/10.1108/02630801011089164
7. Fujita, Aito, et al. "Damage detection from aerial images via convolutional neural networks." 2017 Fifteenth IAPR International Conference on Machine Vision Applications (MVA). IEEE, 2017.
8. Li, Peijun, Haiqing Xu, and Jiancong Guo. "Urban building damage detection from very high resolution imagery using OCSVM and spatial features." International Journal of Remote Sensing 31.13 (2010): 3393-3409.
9. Lee, Bang Yeon, et al. "A technique based on image processing for measuring cracks in the surface of concrete structures." Proceedings of the 19th International Conference on Structural Mechanics in Reactor Technology (SMiRT-19). 2007.
10. Hoang, Nhat-Duc. "Image Processing-Based Recognition of Wall Defects Using Machine Learning Approaches and Steerable Filters." Computational intelligence and neuroscience 2018 (2018).
11. Mohan, Arun, and Sumathi Poobal. "Crack detection using image processing: A critical review and analysis." Alexandria Engineering Journal 57.2 (2018): 787-798.

12. Kim, In-Ho, et al. "Application of crack identification techniques for an aging concrete bridge inspection using an unmanned aerial vehicle." Sensors 18.6 (2018): 1881.
13. Özgenel, Çağlar Fırat (2019), “Concrete Crack Images for Classification”, Mendeley Data, v2, Available Online : http://dx.doi.org/10.17632/5y9wdsg2zt.2
14. Available Online : https://dataturks.com/projects/miaozh17/Crack%20Classification
15. Available Online : https://github.com/hardikvasa/google-images-download
16. Available Online : https://www.spyder-ide.org/
17. Available Online : https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_tutorials.html
18. Available Online : https://numpy.org/index.html
19. Available Online : https://matplotlib.org/
20. Available Online : https://wiki.python.org/moin/TkInter
21. Available Online : https://jupyter.org/
22. Available Online : https://www.kaggle.com/
23. Thagunna, Grishma. "Building cracks–causes and remedies." 3rd World Conference on Applied Sciences, Engineering & Technology at Basha Research Centre. 2014.
24. LeCun, Yann, et al. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE* 86.11 (1998): 2278-2324.
25. Available Online : https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53