National College of Ireland

Configuration Manual

MSc Research Project
MSCAI

Deep Kantilal Patel
Student ID: X23107260

School of Computing
National College of Ireland

Supervisor: SHERESH ZAHOOR

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: ……Deep Kantilal Patel……..……

Student ID: ……X23107260……..……

Programme: …MSCAI…… **Year:** …2023-24..

Module: …… MSc Research Project ……..……

Lecturer: … SHERESH ZAHOOR……..……

Submission Due Date: ……16 September 2024……..……

Project Title: …Construction Defect Detection using AI with Augmented Reality…

Word Count: ……1110…… **Page Count:** ……10……..…….……

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: …Deep Kantilal Patel……

Date: ……16 September 2024……

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Deep Kantilal Patel
Student ID: X23107260

1 Introduction

The detection of the building construction defects using AI techniques defines the implementation of the deep learning approach. This defines the implementation of the functional section which is applicable for the construction of the data models. The detection approach introduces the implementation of the augmented reality approach to detect the defects using 3D visualization. The image data is applicable to detect the images (Tan *et al.*, 2024). This provides information about the image processing to detect the defects in the system.

2 Project Overview

The overall project focuses on the detection approach to understand the defects present in the construction area. This project explores the usability of deep learning models such as CNN, RNN which are applied to evaluate parameters within the area. These models are crucial for evaluating parameters within the construction zones, enabling precise defect detection through their ability to process complex data and recognize patterns. The construction approach provides information about the handling of the data in an informative manner. The construction process determines the functional segments that are usable for the evaluation of the data parameters (Al-Sabbag *et al.*, 2022). The findings highlight the potential for integrating augmented reality with AI technology to enhance the evaluation process.

3 Hardware/Software Implementation

Hardware
Processor: Intel Core i3 or higher
GPU (optional, for fast model training): NVIDIA GTX 2050 Ti or Upper
RAM: 16 GB or more
Storage: 250 GB free space

Software
Jupyter Notebook: This software is applicable to run the Python code (Version: Latest)
Anaconda: This is the environment where Python coding can be implemented and also executed (Version: Latest)

4 Data Collection

The data is collected from a secondary resource, ‘Kaggle.com’. This defines the collection of the secondary data which contains various images. The data contains various folders which introduce various image data. Those data are applicable for the training of the model and also testing of the model.

5 Implementation of project

5.1 Libraries/Modules initialization

Imprort libraries

```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
import matplotlib.pyplot as plt
import os
```

```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator, img_to_array, load_img
import numpy as np
import os
import shutil
```

```
from imblearn.over_sampling import RandomOverSampler
```

Figure 1: Import of the libraries

(Source: Own-Evaluated)

The first section of the evaluation defines the initialization of the libraries and modules that are used for the execution of the overall process. This defines the initialization of those modules which are implemented for the testing process.

5.2 Data initialization and processing

Dataset path

```
base_dir = 'C:/Users/Deep/OneDrive/Desktop/MSC AI/Sem 2/Practicum/Final/project/Assest'
blister_dir = os.path.join(base_dir, 'C:/Users/Deep/OneDrive/Desktop/MSC AI/Sem 2/Practicum/Final/project/Assest/Blister')
cracks_dir = os.path.join(base_dir, 'C:/Users/Deep/OneDrive/Desktop/MSC AI/Sem 2/Practicum/Final/project/Assest/Cracks')
mold_dir = os.path.join(base_dir, 'C:/Users/Deep/OneDrive/Desktop/MSC AI/Sem 2/Practicum/Final/project/Assest/Mold')
peel_dir = os.path.join(base_dir, 'C:/Users/Deep/OneDrive/Desktop/MSC AI/Sem 2/Practicum/Final/project/Assest/Peel')
seepage_dir = os.path.join(base_dir, 'C:/Users/Deep/OneDrive/Desktop/MSC AI/Sem 2/Practicum/Final/project/Assest/Seepage')
```

Figure 2: Dataset Directories

(Source: Own-Evaluated)

The dataset path section is the initialization and processing approach of the data analysis. This structure organizes images into categories such as mold, peel, cracks, seepage and blister which are essential for setting up the data frames. These images are then used to evaluate and detect the specific factors related to defects in construction area.

5.3 Data preprocessing

```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    validation_split=0.2,  # split for training and validation
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

train_generator = train_datagen.flow_from_directory(
    base_dir,
    target_size=(150, 150),
    batch_size=20,
    class_mode='categorical',
    subset='training'
)

validation_generator = train_datagen.flow_from_directory(
    base_dir,
    target_size=(150, 150),
    batch_size=20,
    class_mode='categorical',
    subset='validation'
)
```

```
Found 16334 images belonging to 5 classes.
Found 4081 images belonging to 5 classes.
```

Figure 3: Train and validate image data count
(Source: Own-Evaluated)

The training and confirmation technique is enforced to train data to pinnacle and validate the data. This procedure also portrays the stage of the parameters for the structure of the models.

5.4 Model initialization and construction

```
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)),
    MaxPooling2D(2, 2),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(5, activation='softmax')  # 5 classes
])

model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Figure 4: Model setting
(Source: Own-Evaluated)

The model environment guidelines are established to set the parameters for how the model operates. This ensures that the guidelines are properly communicated and that the model's functionality, particularly in terms of detection is effective.

5.5 Model evaluation

```
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
```

```
history = model.fit(
    train_generator,
    epochs=100,
    validation_data=validation_generator
)
```

```
Epoch 1/100
511/511 ━━━━━━━━━━━━━━━━━━━━ 414s 802ms/step - accuracy: 0.9874 - loss: 0.1328 - val_accuracy: 0.9877 - val_loss: 0.1449
Epoch 2/100
511/511 ━━━━━━━━━━━━━━━━━━━━ 191s 373ms/step - accuracy: 0.9855 - loss: 0.1861 - val_accuracy: 0.9877 - val_loss: 0.0702
Epoch 3/100
511/511 ━━━━━━━━━━━━━━━━━━━━ 274s 536ms/step - accuracy: 0.9869 - loss: 0.1509 - val_accuracy: 0.9877 - val_loss: 0.1475
Epoch 4/100
511/511 ━━━━━━━━━━━━━━━━━━━━ 162s 316ms/step - accuracy: 0.9857 - loss: 0.1440 - val_accuracy: 0.9877 - val_loss: 0.0798
Epoch 5/100
511/511 ━━━━━━━━━━━━━━━━━━━━ 130s 255ms/step - accuracy: 0.9871 - loss: 0.0863 - val_accuracy: 0.9877 - val_loss: 0.0796
Epoch 6/100
511/511 ━━━━━━━━━━━━━━━━━━━━ 154s 301ms/step - accuracy: 0.9865 - loss: 0.0904 - val_accuracy: 0.9877 - val_loss: 0.0800
Epoch 7/100
511/511 ━━━━━━━━━━━━━━━━━━━━ 159s 309ms/step - accuracy: 0.9889 - loss: 0.0770 - val_accuracy: 0.9877 - val_loss: 0.0797
Epoch 8/100
511/511 ━━━━━━━━━━━━━━━━━━━━ 130s 254ms/step - accuracy: 0.9863 - loss: 0.0914 - val_accuracy: 0.9877 - val_loss: 0.0798
Epoch 9/100
511/511 ━━━━━━━━━━━━━━━━━━━━ 130s 254ms/step - accuracy: 0.9867 - loss: 0.0876 - val_accuracy: 0.9877 - val_loss: 0.0797
Epoch 10/100
```

Figure 5: Evaluation of the model

(Source: Own-Evaluated)

The evaluation process for the model involves assessing its performance over multiple epochs. Here the model is trained for 100 epochs to reach its peak performance and to measure its accuracy.

5.6 Training and validation accuracy

```
plt.figure(figsize=(12, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')
```

```
Text(0.5, 1.0, 'Training and Validation Accuracy')
```

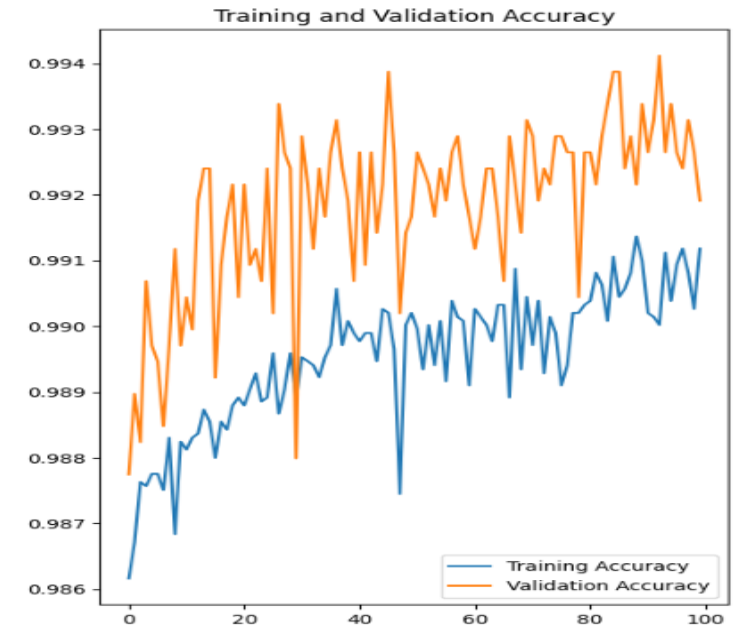

Figure 6: Training and validation accuracy plot

(Source: Own-Evaluated)

The plot of training and validation accuracy is illustrated in this section highlighting the practical and proof data parameters.

5.7 Original image dataset count

```
def count_images_in_classes(path):
    class_counts = {}
    for class_name in classes:
        class_path = os.path.join(path, class_name)
        num_images = len(os.listdir(class_path))
        class_counts[class_name] = num_images
    return class_counts


print("Original Dataset:")
original_counts = count_images_in_classes(dataset_path)
print(original_counts)


plt.figure(figsize=(10, 6))
plt.bar(original_counts.keys(), original_counts.values(), color='blue')
plt.title('Class Distribution of Images (Original Dataset)')
plt.xlabel('Class')
plt.ylabel('Number of Images')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

```
Original Dataset:
{'Blister': 46, 'Cracks': 20155, 'Mold': 144, 'Peel': 33, 'Seepage': 37}
```

Figure 7: Original Dataset Count

(Source: Own-Evaluated)

The dataset used in this project was collected from secondary data. This defect dataset consists of different class like Blister, Cracks, Mold, Peel and Seepage and all classes have unbalanced data.

5.8 Undersampling Matrix

```
def count_images_in_classes(path):
    class_counts = {}
    for class_name in classes:
        class_path = os.path.join(path, class_name)
        num_images = len(os.listdir(class_path))
        class_counts[class_name] = num_images
    return class_counts


print("Original Dataset:")
original_counts = count_images_in_classes(dataset_path)
print(original_counts)


print("\nUndersampled Dataset:")
undersampled_counts = count_images_in_classes(undersampled_path)
print(undersampled_counts)
```

```
Original Dataset:
{'Blister': 46, 'Cracks': 20155, 'Mold': 144, 'Peel': 33, 'Seepage': 37}

Undersampled Dataset:
{'Blister': 33, 'Cracks': 33, 'Mold': 33, 'Peel': 33, 'Seepage': 33}
```

Figure 8: Undersampling Dataset Count

(Source: Own-Evaluated)

Undersampling is used to remove sample from class having majority in sample. It will check image from all classes and remove extra image sample to make dataset balanced.

5.9 Oversampling Matrix

```python
def oversample_class(class_name, oversample_to=None):
    class_path = os.path.join(dataset_path, class_name)
    images = os.listdir(class_path)
    num_images = len(images)

    if oversample_to is None:
        oversample_to = max(num_images, 1000)

    # Create target directory if it doesn't exist
    target_path = os.path.join(oversampled_path, class_name)
    os.makedirs(target_path, exist_ok=True)

    # Calculate oversampling ratio
    ratio = oversample_to // num_images

    # Perform oversampling by replicating images
    for img_name in images:
        img_path = os.path.join(class_path, img_name)
        img = cv2.imread(img_path)

        for i in range(ratio):
            new_img_name = f"{os.path.splitext(img_name)[0]}_{i}.{os.path.splitext(img_name)[1]}"
            new_img_path = os.path.join(target_path, new_img_name)
            cv2.imwrite(new_img_path, img)
```

Figure 9: Oversampling Dataset Count

(Source: Own-Evaluated)

Oversampling is used to add more sample to the minority classes to make dataset balanced. As there is huge difference in dataset class oversample will perform to generate sample image in the minority class.

5.10 Comparison of dataset matrix

```python
# Plot class distribution for original dataset
plt.figure(figsize=(15, 6))
plt.subplot(1, 3, 1)
plt.bar(original_counts.keys(), original_counts.values(), color='blue')
plt.title('Original Dataset')
plt.xlabel('Class')
plt.ylabel('Number of Images')
plt.xticks(rotation=45)

# Plot class distribution for undersampled dataset
plt.subplot(1, 3, 2)
plt.bar(undersampled_counts.keys(), undersampled_counts.values(), color='green')
plt.title('Undersampled Dataset')
plt.xlabel('Class')
plt.ylabel('Number of Images')
plt.xticks(rotation=45)

# Plot class distribution for oversampled dataset
plt.subplot(1, 3, 3)
plt.bar(oversampled_counts.keys(), oversampled_counts.values(), color='red')
plt.title('Oversampled Dataset')
plt.xlabel('Class')
plt.ylabel('Number of Images')
plt.xticks(rotation=45)

plt.tight_layout()
plt.show()
```

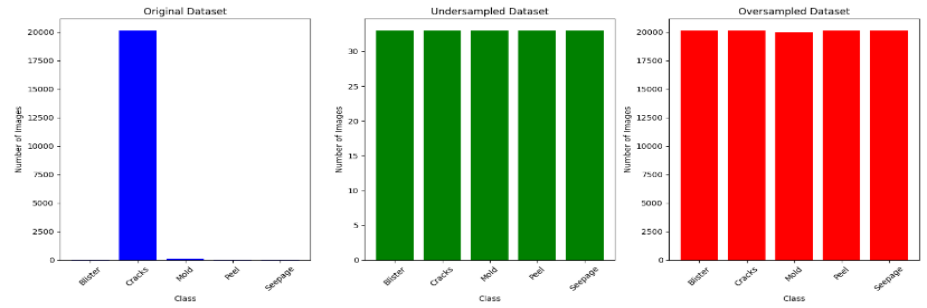

Figure 10: Comparison of all matrix

The comparison of all matrix is performed to get class distribution for original dataset, Oversampling dataset and Undersampling dataset.

5.11 Model parameter comparison

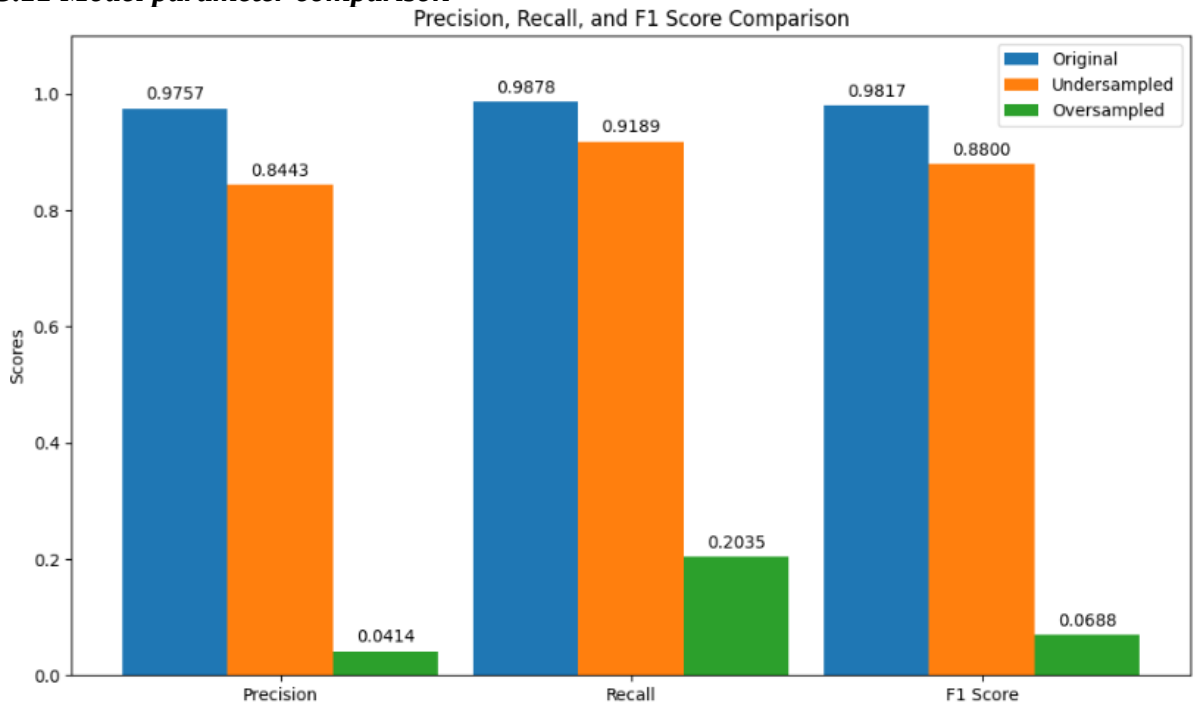

Figure 11: Comparison of F1, Recall, and Precision with respect to dataset types
(Source: Own-Evaluated)

The model characteristic comparison procedure is carried out to compare the parameters of different models. In this case, the performance metrics used are F1 Score, Recall, and Precision which are evaluated with respect to various dataset classifications such as original, oversampled and undersampled.

5.12 Training and Testing model

```
# Function to load and preprocess images
def load_data(dataset_path):
    X = []
    y = []

    for class_index, class_name in enumerate(classes):
        class_path = os.path.join(dataset_path, class_name)
        for img_name in os.listdir(class_path):
            img_path = os.path.join(class_path, img_name)
            img = cv2.imread(img_path, cv2.IMREAD_COLOR)
            img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  # Convert from BGR to RGB
            img = cv2.resize(img, (img_width, img_height))  # Resize to a standard size
            X.append(img)
            y.append(class_index)

    X = np.array(X)
    y = np.array(y)

    return X, y

# Load original dataset
X_orig, y_orig = load_data(dataset_path)

# Split original dataset into train and test sets
X_train_orig, X_test_orig, y_train_orig, y_test_orig = train_test_split(X_orig, y_orig, test_size=0.2, random_state=42)

# Load undersampled dataset (Assuming you have already undersampled your data)
undersampled_path = "C:/Users/Deep/OneDrive/Desktop/MSC AI/Sem 2/Practicum/Final/project/undersample"
X_under, y_under = load_data(undersampled_path)

# Split undersampled dataset into train and test sets
X_train_under, X_test_under, y_train_under, y_test_under = train_test_split(X_under, y_under, test_size=0.2, random_state=42)

# Load oversampled dataset (Assuming you have already oversampled your data)
oversampled_path = "C:/Users/Deep/OneDrive/Desktop/MSC AI/Sem 2/Practicum/Final/project/oversample"
X_over, y_over = load_data(oversampled_path)

# Split oversampled dataset into train and test sets
X_train_over, X_test_over, y_train_over, y_test_over = train_test_split(X_over, y_over, test_size=0.2, random_state=42)
```

Figure 12: Load, train and test image dataset

The load and preprocess functions will convert and standardize the image format and will split the dataset into training and testing sets.

```
# Function to create CNN model
def create_model(input_shape, num_classes):
    model = Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=input_shape),
        MaxPooling2D((2, 2)),
        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Flatten(),
        Dense(64, activation='relu'),
        Dropout(0.5),
        Dense(num_classes, activation='softmax')
    ])

    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

    return model
```

```
# Create and train model on original dataset
model_orig = create_model(input_shape=(img_width, img_height, 3), num_classes=len(classes))
model_orig.fit(X_train_orig, y_train_orig, epochs=2, validation_data=(X_test_orig, y_test_orig), verbose=0)

# Create and train model on undersampled dataset
model_under = create_model(input_shape=(img_width, img_height, 3), num_classes=len(classes))
model_under.fit(X_train_under, y_train_under, epochs=2, validation_data=(X_test_under, y_test_under), verbose=0)

# Create and train model on oversampled dataset
model_over = create_model(input_shape=(img_width, img_height, 3), num_classes=len(classes))
model_over.fit(X_train_over, y_train_over, epochs=2, validation_data=(X_test_over, y_test_over), verbose=0)
```

```
C:\Users\Deep\anaconda3\envs\Python\Lib\site-packages\keras\src\layers\convolutional\base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
<keras.src.callbacks.history.History at 0x284b19bd190>
```

Figure 13: Create and Train CNN model

(Source: Own-Evaluated)

The CNN model used consists of two convolutions layers for classification, flatten layer and dense layers. The model is compiled with Adam Optimizer and categorical cross-entropy loss.

5.13 Load, preprocess and detect defects in image using OpenCV

```
def load_images_from_folder(folder):
    images = []
    for filename in os.listdir(folder):
        img_path = os.path.join(folder, filename)
        if os.path.isfile(img_path):
            img = cv2.imread(img_path)
            if img is not None:
                images.append(img)
    return images
```

```
def preprocess_image(img):

    img = cv2.resize(img, (800, 600))

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    blurred = cv2.GaussianBlur(gray, (5, 5), 0)
    return blurred
```

```
def detect_defects(img):

    _, thresh = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)

    contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    img_with_contours = cv2.drawContours(img.copy(), contours, -1, (0, 255, 0), 3)
    return img_with_contours
```

Figure 14: Load, preprocess and detect defects in image using OpenCV

(Source: Own-Evaluated)

OpenCV is used for computer vision and image processing task to detect defects. OpenCV converts the original image into grayscale image. Grayscale based defect detection is widely used in quality control system where it will detect issues like cracks, peels, seepage, blister and mold.

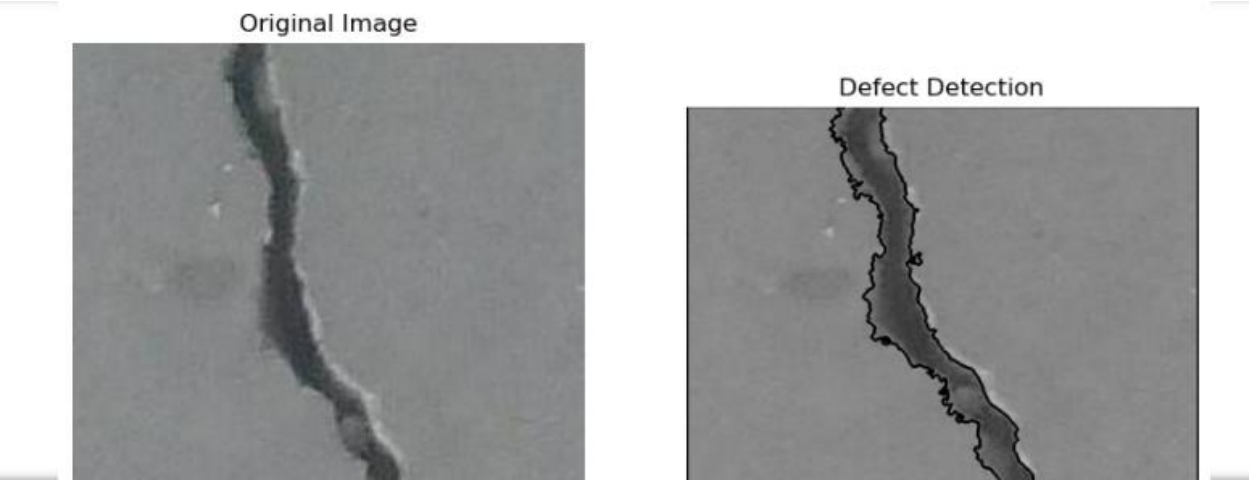

Figure 15: Original and defective data Determination 1

(Source: Own-Evaluated)

The cracks in the wall section are evaluated by the detection process using the original image evaluation approach. This defines the defection section from the original image data.

Figure 16: Original and defective data Determination 2

(Source: Own-Evaluated)

The upper-side corner wall defect detection approach is implemented by using the detection model. This evaluates the original image and detects the defect.

Figure 17: Original and defective data Determination 3

(Source: Own-Evaluated)

The lower-side corner of the room or wall is evaluated by using the original image data. The detected defect section is evaluated using the deep learning technique.

References

Tan, Y., Xu, W., Chen, P. and Zhang, S., (2024). Building defect inspection and data management using computer vision, augmented reality, and BIM technology. Automation in Construction, 160, p.105318.

Al-Sabbag, Z.A., Yeum, C.M. and Narasimhan, S., (2022). Interactive defect quantification through extended reality. Advanced Engineering Informatics, 51, p.101473.