Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

SCHOOL OF COMPUTER SCIENCE ENGINEERING AND INFORMATION SYSTEMS

FALL-SEMESTER – 2024-25

Course code - SWE1011

Course Name – Soft Computing

Title: Surface Crack detection using CNN

Review – 3

TEAM MEMBERS

JYOTHI PRIYA N - 21MIS0417

VARUN KUMAR S - 21MIS0462

BENSON RANJITH – 21MIS0358

Under the guidance of

Prof. PRADEEPA. M

ABSTRACT:

This project proposes a soft computing-based approach for insulator crack detection in electrical power transmission systems. Leveraging fuzzy logic and neural networks, we preprocess and extract features from a dataset of insulator images with varying crack degrees. The model is trained and evaluated, with a focus on accuracy and precision. The project aims to enhance insulator monitoring, providing a robust crack detection system to prevent electrical failures. The outcomes contribute to intelligent systems for infrastructure maintenance, advancing the application of soft computing in the electrical domain.

PROBLEM STATEMENT:

The reliable operation of electrical power transmission systems is contingent upon the integrity of insulators. Over time, insulators are susceptible to cracks that, if undetected, can lead to catastrophic failures, service interruptions, and safety hazards. Current methods for insulator crack detection often rely on manual inspections, which are time consuming, labor-intensive, and prone to human error. This project addresses the need for an efficient and automated insulator crack detection system using soft computing techniques. The challenge lies in developing a model capable of accurately identifying and classifying cracks in insulator images, thereby facilitating proactive maintenance, and preventing potential electrical failures. The project seeks to overcome the limitations of existing detection methods by leveraging the capabilities of fuzzy logic and neural networks for robust feature extraction and intelligent decision-making in crack identification.

PROPOSED SOLUTION:

The proposed solution aims to enhance the reliability of electrical power transmission infrastructure by providing a timely and accurate means of detecting insulator cracks, minimizing downtime, and ensuring the uninterrupted flow of electrical power using soft computing techniques.

OBJECTIVES:

❖ Develop a CNN Architecture:

➢ Experiment with CNN architectures, considering layers, filters, and activation functions. Explore advanced CNN structures like ResNets or DenseNets for enhanced feature extraction.

➢ Implement dropout and batch normalization to improve the generalization of the CNN model.

❖ Preprocessing for Image Enhancement:

➢ Investigate advanced image enhancement techniques (e.g., histogram equalization) for insulator images. Apply image segmentation to isolate insulator regions, focusing on enhancing crack-related features.

➢ Implement noise reduction methods (e.g., Gaussian smoothing) to enhance the visibility of subtle cracks in images.

❖ Data Collection and Augmentation:

➢ Explore synthetic data generation (e.g., GANs) to diversify the dataset with realistic insulator crack variations. Address dataset imbalances using oversampling techniques or weighted loss functions.

➢ Investigate domain adaptation strategies to ensure the CNN's robustness across diverse environmental conditions.

❖ Transfer Learning with Pre-trained Models:

➢ Evaluate various pre-trained CNN models (e.g., VGG, Inception) for transfer learning effectiveness. Compare the impact of freezing layers versus fine-tuning the entire CNN architecture.

➢ Explore ensemble learning by combining predictions from different pre-trained models to enhance crack detection performance. Optimize CNN Hyperparameters.

SEVERAL SOFT COMPUTING TECHNIQUES APPLIED ON THE SYSTEM:

1. **Fuzzy Logic:** It is a mathematical framework that deals with reasoning and decision-making in situations where uncertainty and imprecision exist. It's particularly useful when dealing with vague or ambiguous information.

➢ In insulator crack detection, fuzzy logic can be employed to model the uncertainty associated with crack detection. Fuzzy sets can represent the degree of membership of an image region to the "cracked" or "non-cracked" categories, allowing for a more nuanced analysis.

2. **Neural Networks:** Neural networks are computational models inspired by the human brain. They consist of interconnected nodes (neurons) organized in layers and are capable of learning complex patterns from data.

➢ In insulator crack detection, Neural networks can be trained on your dataset to learn the intricate features associated with insulator cracks. Convolutional Neural Networks (CNNs) are particularly effective for image-related tasks, as they can automatically extract hierarchical features.

APPROPRIATE TECHNIQUE FOR INSULATOR CRACK DETECTION:

Convolutional Neural Networks (CNNs):

➢ Effective Feature Extraction: CNNs are specifically designed for image-related tasks, and they excel at automatically learning hierarchical features from visual data. In the context of insulator crack detection, where patterns and textures in images are crucial indicators, CNNs are well-suited for capturing these intricate features.

➢ State-of-the-Art Performance: CNNs have demonstrated state-of-the-art performance invarious computer vision tasks, including image classification and object detection. Their ability to capture complex relationships within image data aligns with the diverse patterns associated with different types and degrees of insulator cracks.

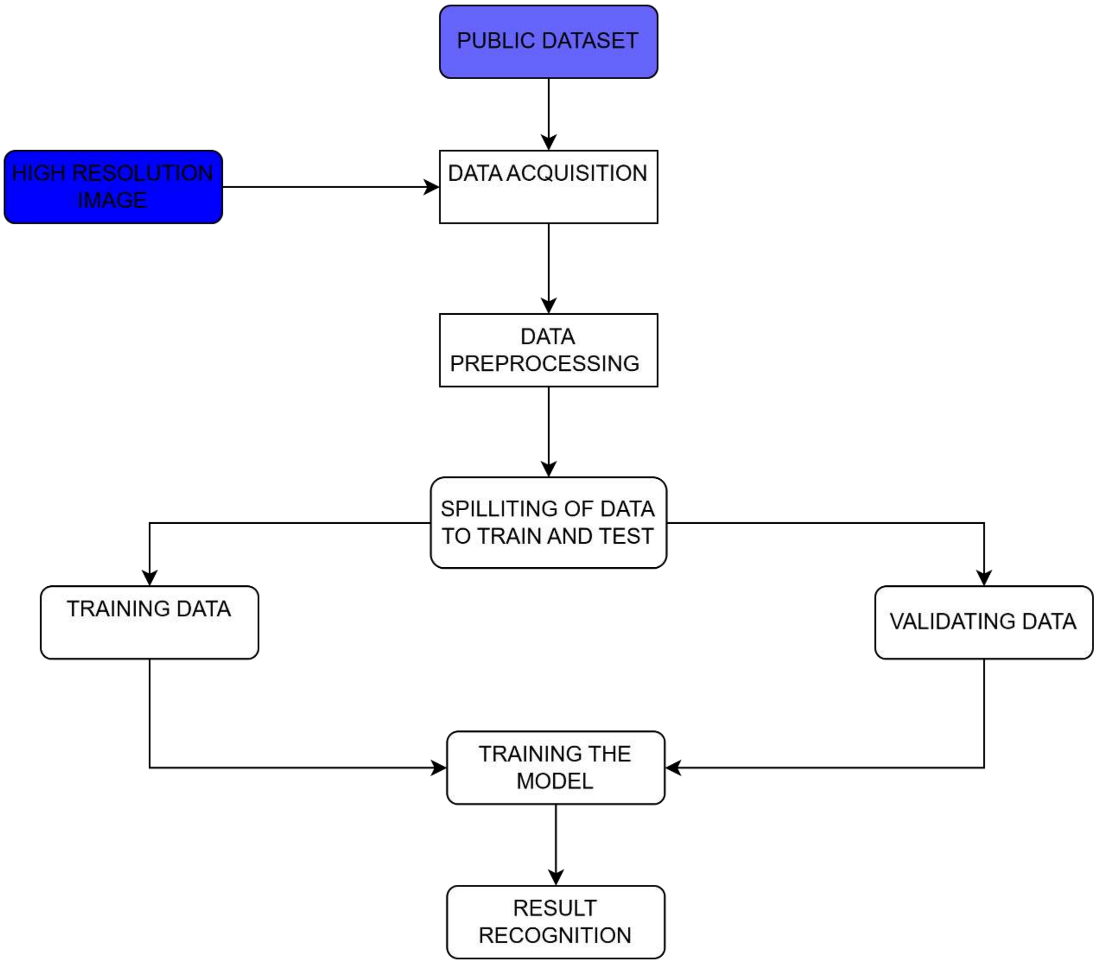
➢ Availability of Pre-trained Models: Pre-trained CNN models on large datasets (e.g., ImageNet) are readily available. Leveraging transfer learning by fine-tuning these models on your specific insulator crack dataset can significantly boost performance, especially when limited labeled data is available.

➢ Adaptability to Image Data: CNNs are inherently suitable for working with grid-like data, making them well-suited for image processing tasks. Insulator crack detection involves analyzing visual patterns in images, and CNNs are adept at capturing spatial dependencies within these images.

➢ Ease of Implementation: There are many deep learning frameworks (e.g., TensorFlow, PyTorch) that offer easy-to-use implementations of CNNs. This facilitates the development and experimentation process, allowing you to focus on optimizing and fine-tuning the model for your specific problem.

➢ Proven Success in Similar Tasks: CNNs have been successfully applied in various image recognition and detection tasks, showcasing their effectiveness in extracting relevant features from visual data. Their proven success in similar domains makes them a reliable choice for insulator crack detection.

CNN MODEL IN INSULATOR CRACK DETECTION:



DATASET :

https://www.kaggle.com/code/brsdincer/surface-crack-detection-end-to-end-process/notebook

LITERATURE SURVEY:

TITLE	AUTHORS	METHODOLOGY	ALGORITHM USED	ADVANTAGES
Surface Crack Detection Based on Image Stitching and Transfer Learning with Pretrained Convolutional Neural Network	Wu, Lijun, Xu Lin, Zhicong Chen, Peijie Lin, and Shuying Cheng	Deep learning approach utilizing BERT for sentiment analysis.	BERT (Bidirectional Encoder Representations from Transformers)	High accuracy in detecting surface cracks due to deep context capture.
Surface Crack Detection From Aerial Images Using a Deformable Convolutional Neural Network	Yu, Dawen, Shunping Ji, Xue Li, Zhaode Yuan, and Chaoyong Shen.	Utilizes deformable convolutions to handle irregular crack detection.	Crack-CADNet	Effective for handling complex, irregular crack patterns in natural terrains.
Surface Crack Detection Using Transfer Learning Based CNN Models	Ali, Sayyed Bashar, Reshul Wate, Sameer Kujur, Anurag Singh, and Santosh Kumar	Transfer learning-based CNNs (MobileNetV2, ResNet101, etc.)	**Transfer Learning-Based CNN Models**	High accuracy with MobileNetV2 model, reducing computation with pre-trained models
CNN-Based Pavement Defects Detection Using Grey and Depth Images	Li, Peigen, Bin Zhou, Chuan Wang, Guizhang Hu, Yong Yan, Rongxin Guo, and Haiting Xia	CNNs with attention mechanisms like SE-Net and CBAM for segmentation and identification.	Squeeze-and-Excitation Networks (SE-Net) and Convolutional Block Attention Module (CBAM)	Enhanced feature extraction and improved segmentation accuracy
Developing a New Deep Learning CNN Model to Detect and Classify Highway Cracks	Pre-trained CNN models and new CNN model tested with various optimization algorithms.	Pre-trained CNN models and new CNN model tested with various optimization algorithm	Custom CNN with Adam Optimizer	High accuracy of 97.62% using Adam, surpassing pre-trained models.

IMPLEMENTATION AND RESULTS:

Surface crack detection using CNN – Convolutional Neural Network

Mounting drive to google collab:

```
[ ] from google.colab import drive
    drive.mount('/content/drive')

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

Importing libraries and packages:

```
#GENERAL
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import random
#PATH PROCESS
import os
import os.path
from pathlib import Path
import glob
#IMAGE PROCESS
from PIL import Image
from keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import cv2
from keras.applications.vgg16 import preprocess_input, decode_predictions
from skimage.feature import hessian_matrix, hessian_matrix_eigvals
from scipy.ndimage import convolve
from skimage import data, io, filters
import skimage
from skimage.morphology import convex_hull_image, erosion
#SCALER & TRANSFORMATION
from sklearn.preprocessing import StandardScaler, MinMaxScaler, LabelEncoder
from keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from keras import regularizers
#ACCURACY CONTROL
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report, roc_auc_score, roc_curve, mean_squared_error, r2_score
from sklearn.model_selection import GridSearchCV, cross_val_score
#OPTIMIZER
from keras.optimizers import RMSprop, Adam, SGD
```

```
#MODEL LAYERS
from tensorflow.keras.models import Sequential
from keras.layers import (
    Dense, Dropout, Flatten, Conv2D, MaxPool2D, BatchNormalization, MaxPooling2D,
    Permute, TimeDistributed, Bidirectional, GRU, SimpleRNN, LSTM,
    GlobalAveragePooling2D, SeparableConv2D, ZeroPadding2D, Convolution2D
)
from keras import models, layers
import tensorflow as tf
from keras.applications import VGG16, VGG19, inception_v3
from keras import backend as K
from keras.utils import plot_model
from keras.models import load_model
#SKLEARN CLASSIFIER
from xgboost import XGBClassifier, XGBRegressor
from lightgbm import LGBMClassifier, LGBMRegressor
from catboost import CatBoostClassifier, CatBoostRegressor
from sklearn.linear_model import LogisticRegression, LinearRegression, Ridge, RidgeCV, Lasso, LassoCV, ElasticNet, ElasticNetCV
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor, GradientBoostingClassifier, GradientBoostingRegressor, BaggingRegressor
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
from sklearn.neural_network import MLPClassifier, MLPRegressor
from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor
from sklearn.cross_decomposition import PLSRegression
#IGNORING WARNINGS
from warnings import filterwarnings
filterwarnings("ignore", category=DeprecationWarning)
filterwarnings("ignore", category=FutureWarning)
filterwarnings("ignore", category=UserWarning)
```

Main path, label, transformation process:

```
[ ] Surface_Data = Path("/content/drive/MyDrive/Surface crack")

[ ] Surface_JPG_Path = list(Surface_Data.glob(r"*/*.jpg"))

[ ] Surface_Labels = list(map(lambda x: os.path.split(os.path.split(x)[0])[1],Surface_JPG_Path))

[ ] Surface_JPG_Path_Series = pd.Series(Surface_JPG_Path,name="JPG").astype(str)
    Surface_Labels_Series = pd.Series(Surface_Labels,name="CATEGORY")
```

Dataframe:

```
Main_Surface_Data = pd.concat([Surface_JPG_Path_Series,Surface_Labels_Series],axis=1)
print(Main_Surface_Data.head(-1))

                                                    JPG  CATEGORY
0     /content/drive/MyDrive/Surface crack/Negative/...  Negative
1     /content/drive/MyDrive/Surface crack/Negative/...  Negative
2     /content/drive/MyDrive/Surface crack/Negative/...  Negative
3     /content/drive/MyDrive/Surface crack/Negative/...  Negative
4     /content/drive/MyDrive/Surface crack/Negative/...  Negative
...                                                 ...       ...
2687  /content/drive/MyDrive/Surface crack/Positive/...  Positive
2688  /content/drive/MyDrive/Surface crack/Positive/...  Positive
2689  /content/drive/MyDrive/Surface crack/Positive/...  Positive
2690  /content/drive/MyDrive/Surface crack/Positive/...  Positive
2691  /content/drive/MyDrive/Surface crack/Positive/...  Positive

[2692 rows x 2 columns]
```

Shuffle:

```
[ ] Main_Surface_Data = Main_Surface_Data.sample(frac=1).reset_index(drop=True)
    print(Main_Surface_Data.head(-1))

                                                    JPG  CATEGORY
0     /content/drive/MyDrive/Surface crack/Negative/...  Negative
1     /content/drive/MyDrive/Surface crack/Negative/...  Negative
2     /content/drive/MyDrive/Surface crack/Positive/...  Positive
3     /content/drive/MyDrive/Surface crack/Negative/...  Negative
4     /content/drive/MyDrive/Surface crack/Negative/...  Negative
...                                                 ...       ...
2687  /content/drive/MyDrive/Surface crack/Negative/...  Negative
2688  /content/drive/MyDrive/Surface crack/Negative/...  Negative
2689  /content/drive/MyDrive/Surface crack/Negative/...  Negative
2690  /content/drive/MyDrive/Surface crack/Negative/...  Negative
2691  /content/drive/MyDrive/Surface crack/Negative/...  Negative

[2692 rows x 2 columns]
```

VISUALIZATION:

Labels:

```
[ ] plt.style.use("dark_background")
```

```
[ ] Positive_Surface = Main_Surface_Data[Main_Surface_Data["CATEGORY"] == "Positive"]
    Negative_Surface = Main_Surface_Data[Main_Surface_Data["CATEGORY"] == "Negative"]

    Positive_Surface = Positive_Surface.reset_index()
    Negative_Surface = Negative_Surface.reset_index()
```

Vision function:

```
[ ] def simple_vision(path):
        figure = plt.figure(figsize=(8,8))

        Reading_Img = cv2.imread(path)
        Reading_Img = cv2.cvtColor(Reading_Img,cv2.COLOR_BGR2RGB)

        plt.xlabel(Reading_Img.shape)
        plt.ylabel(Reading_Img.size)
        plt.imshow(Reading_Img)
```

```
def canny_vision(path):
    figure = plt.figure(figsize=(8,8))

    Reading_Img = cv2.imread(path)
    Reading_Img = cv2.cvtColor(Reading_Img,cv2.COLOR_BGR2RGB)
    Canny_Img = cv2.Canny(Reading_Img,90,100)

    plt.xlabel(Canny_Img.shape)
    plt.ylabel(Canny_Img.size)
    plt.imshow(Canny_Img)
```

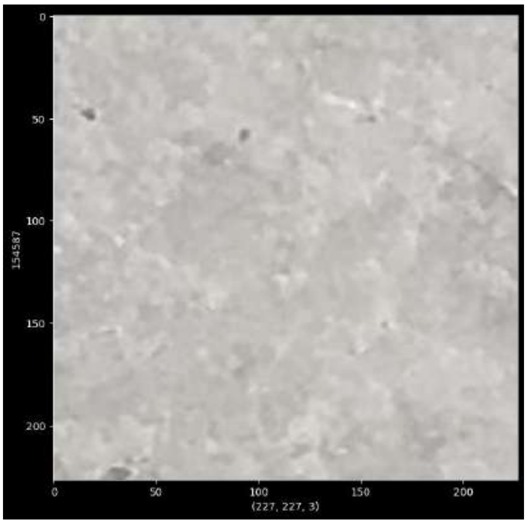
```
[ ] def threshold_vision(path):
        figure = plt.figure(figsize=(8,8))

        Reading_Img = cv2.imread(path)
        Reading_Img = cv2.cvtColor(Reading_Img,cv2.COLOR_BGR2RGB)
        _,Threshold_Img = cv2.threshold(Reading_Img,130,255,cv2.THRESH_BINARY_INV)

        plt.xlabel(Threshold_Img.shape)
        plt.ylabel(Threshold_Img.size)
        plt.imshow(Threshold_Img)
```

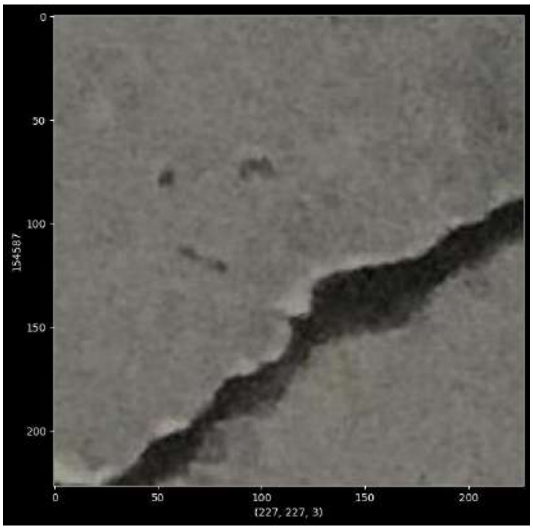
Simple vision :

```
[ ] def threshold_canny(path):
        figure = plt.figure(figsize=(8,8))

        Reading_Img = cv2.imread(path)
        Reading_Img = cv2.cvtColor(Reading_Img,cv2.COLOR_BGR2RGB)
        _,Threshold_Img = cv2.threshold(Reading_Img,130,255,cv2.THRESH_BINARY_INV)
        Canny_Img = cv2.Canny(Threshold_Img,90,100)

        plt.xlabel(Canny_Img.shape)
        plt.ylabel(Canny_Img.size)
        plt.imshow(Canny_Img)
```

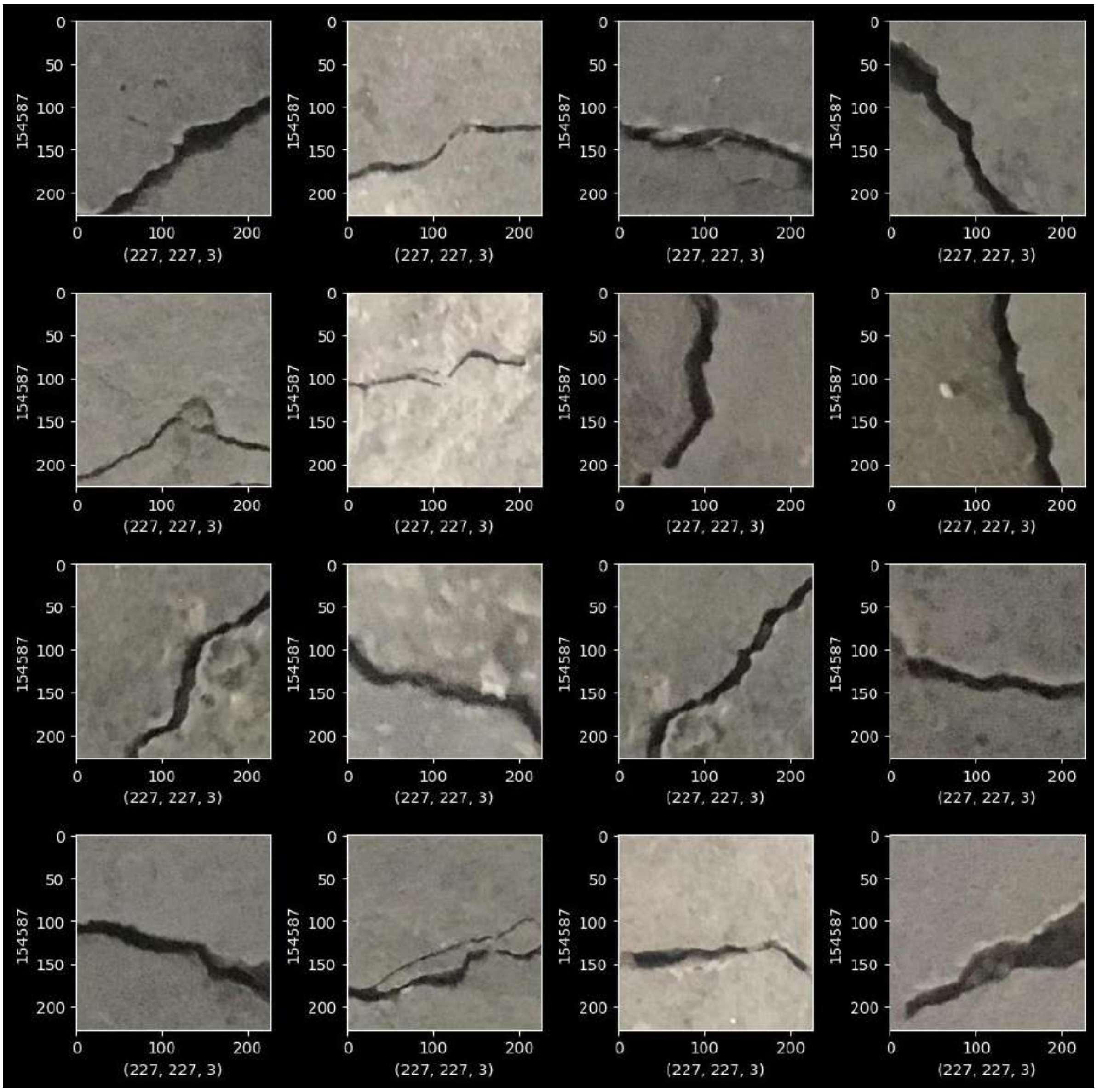
```
simple_vision(Main_Surface_Data["JPG"][4])
```

```
simple_vision(Main_Surface_Data["JPG"][2])
```

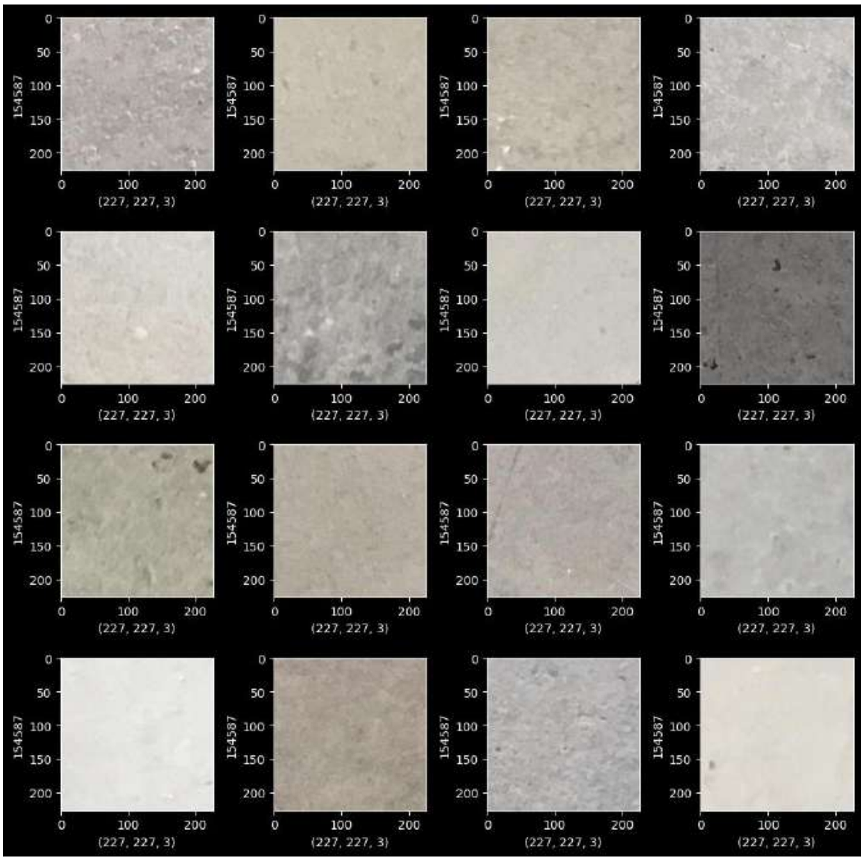





```
figure,axis = plt.subplots(4,4,figsize=(10,10))

for indexing,operations in enumerate(axis.flat):

    Reading_Img = cv2.imread(Positive_Surface["JPG"][indexing])
    Reading_Img = cv2.cvtColor(Reading_Img,cv2.COLOR_BGR2RGB)

    operations.set_xlabel(Reading_Img.shape)
    operations.set_ylabel(Reading_Img.size)
    operations.imshow(Reading_Img)

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



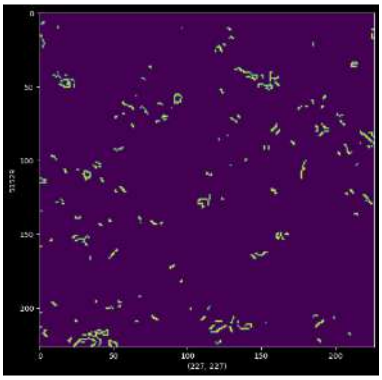

```python
figure,axis = plt.subplots(4,4,figsize=(10,10))

for indexing,operations in enumerate(axis.flat):

    Reading_Img = cv2.imread(Negative_Surface["JPG"][indexing])
    Reading_Img = cv2.cvtColor(Reading_Img,cv2.COLOR_BGR2RGB)

    operations.set_xlabel(Reading_Img.shape)
    operations.set_ylabel(Reading_Img.size)
    operations.imshow(Reading_Img)

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

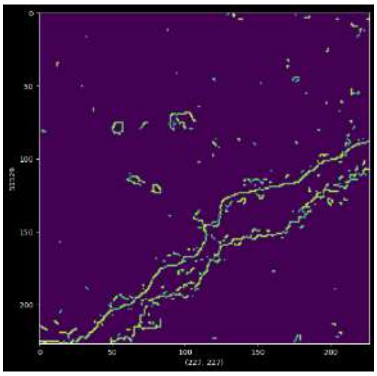


Canny Vision:

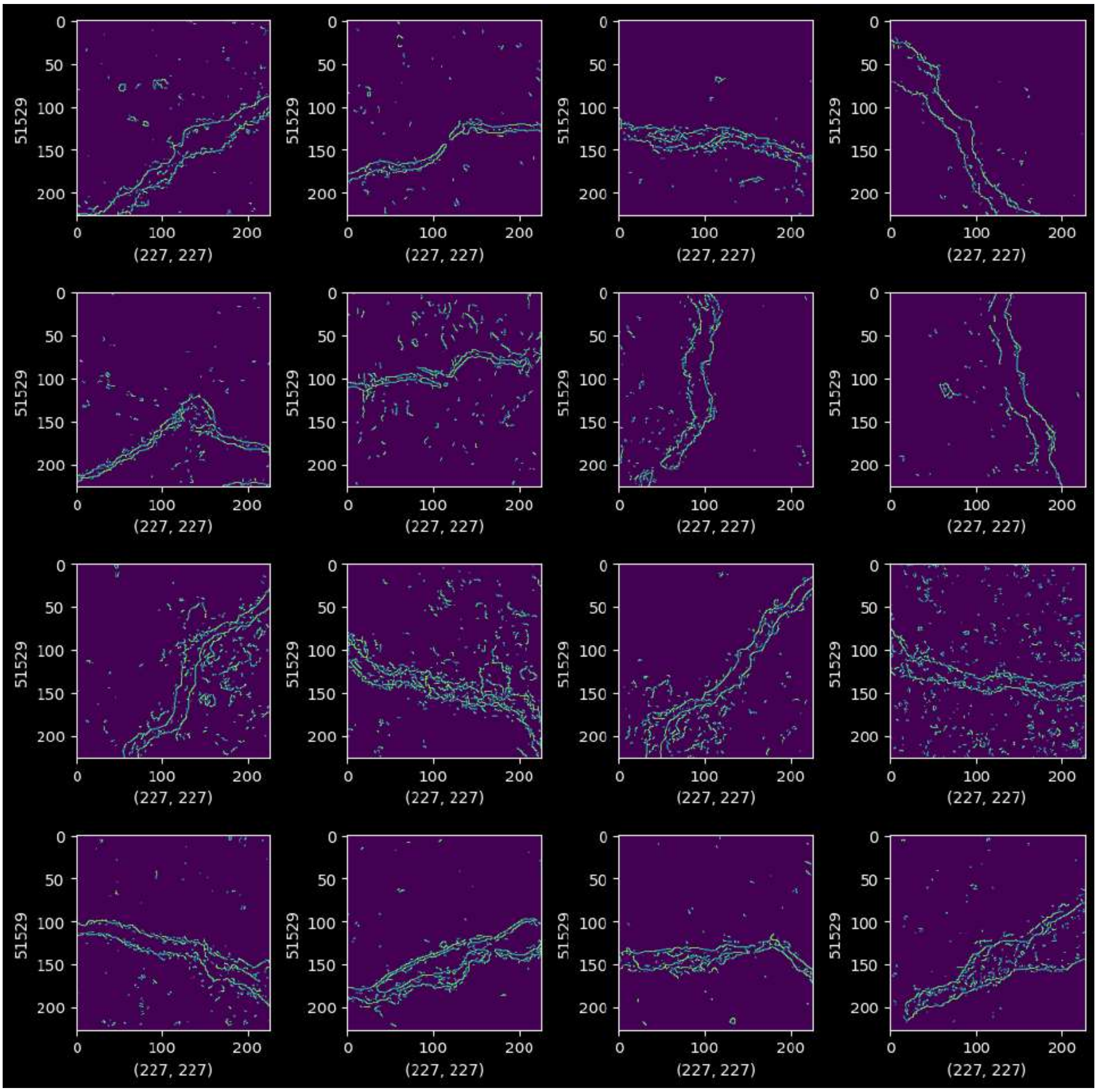
```
canny_vision(Main_Surface_Data["JPG"][2])
```

```
canny_vision(Main_Surface_Data["JPG"][4])
```

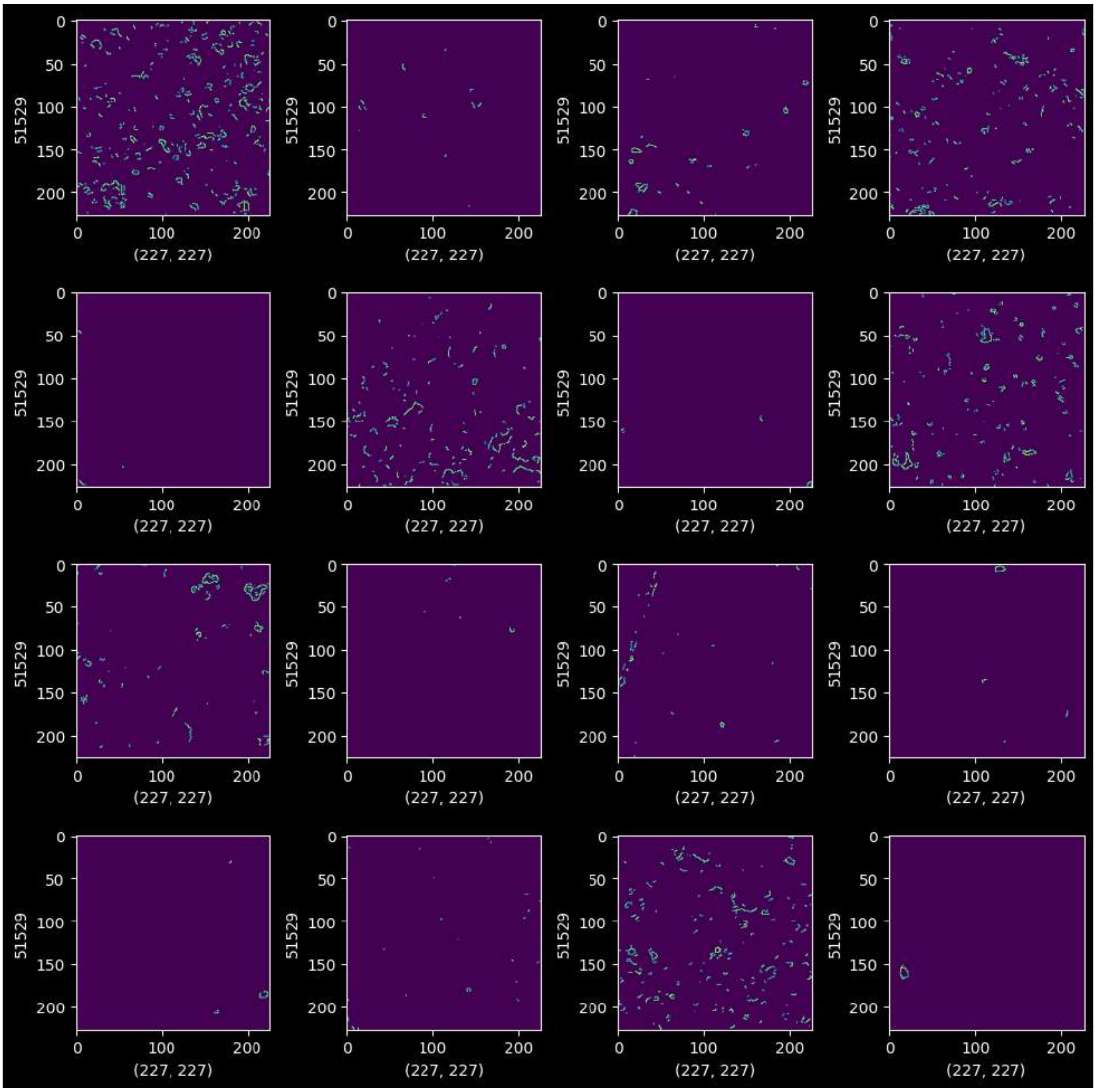




```python
figure,axis = plt.subplots(4,4,figsize=(10,10))

for indexing,operations in enumerate(axis.flat):

    Reading_Img = cv2.imread(Positive_Surface["JPG"][indexing])
    Reading_Img = cv2.cvtColor(Reading_Img,cv2.COLOR_BGR2RGB)

    Canny_Img = cv2.Canny(Reading_Img,90,100)

    operations.set_xlabel(Canny_Img.shape)
    operations.set_ylabel(Canny_Img.size)
    operations.imshow(Canny_Img)

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



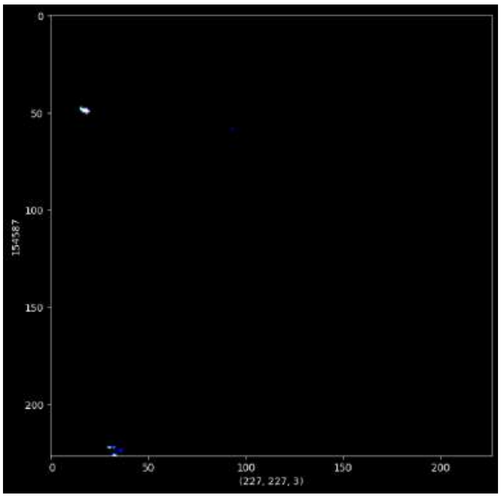
```
figure,axis = plt.subplots(4,4,figsize=(10,10))

for indexing,operations in enumerate(axis.flat):

    Reading_Img = cv2.imread(Negative_Surface["JPG"][indexing])
    Reading_Img = cv2.cvtColor(Reading_Img,cv2.COLOR_BGR2RGB)

    Canny_Img = cv2.Canny(Reading_Img,90,100)

    operations.set_xlabel(Canny_Img.shape)
    operations.set_ylabel(Canny_Img.size)
    operations.imshow(Canny_Img)

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

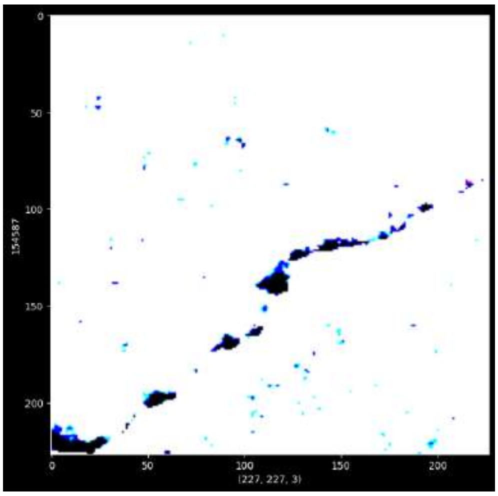


Threshold vision:

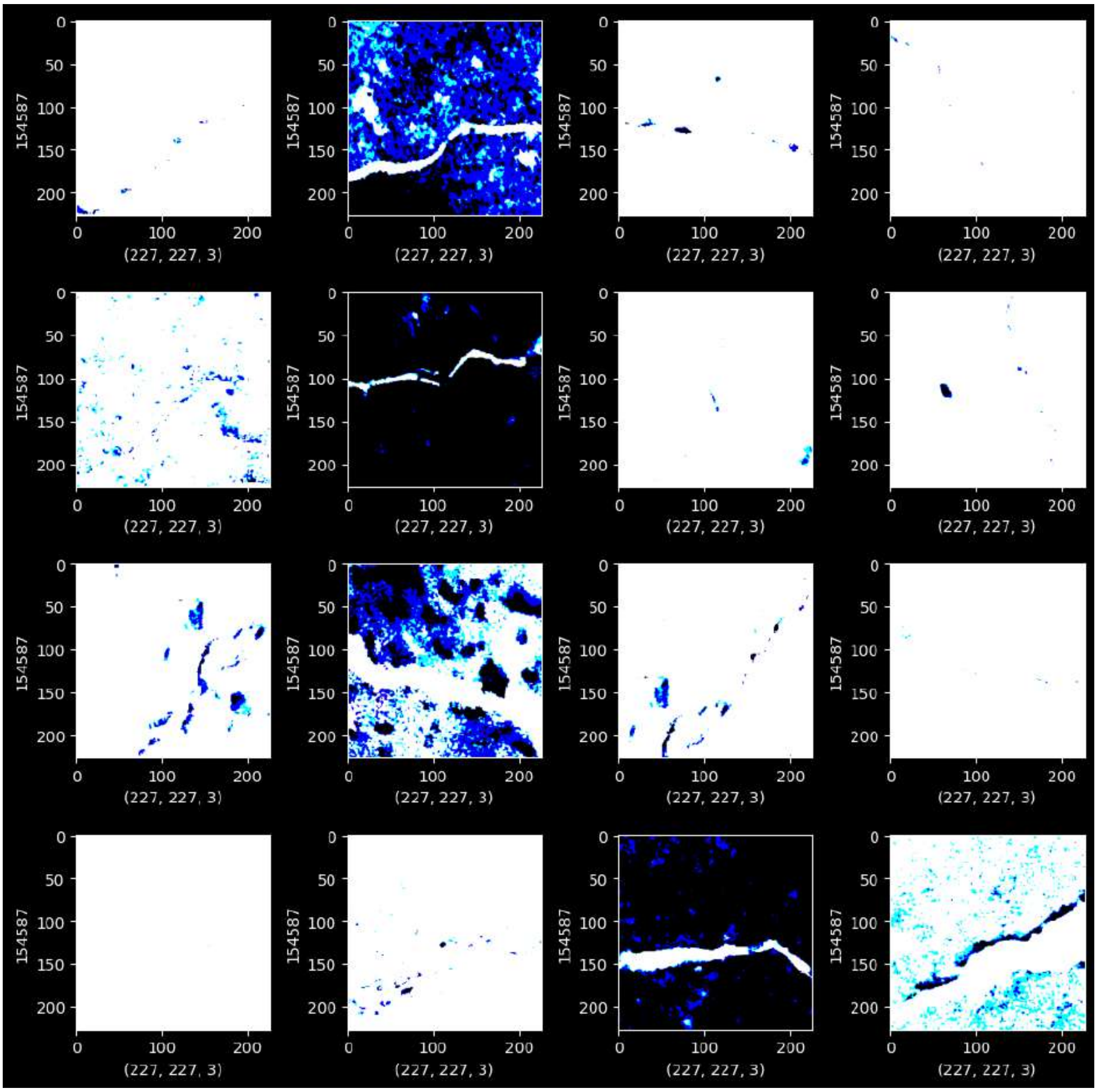
```
[ ]  threshold_vision(Main_Surface_Data["JPG"][4])
```

```
threshold_vision(Main_Surface_Data["JPG"][2])
```

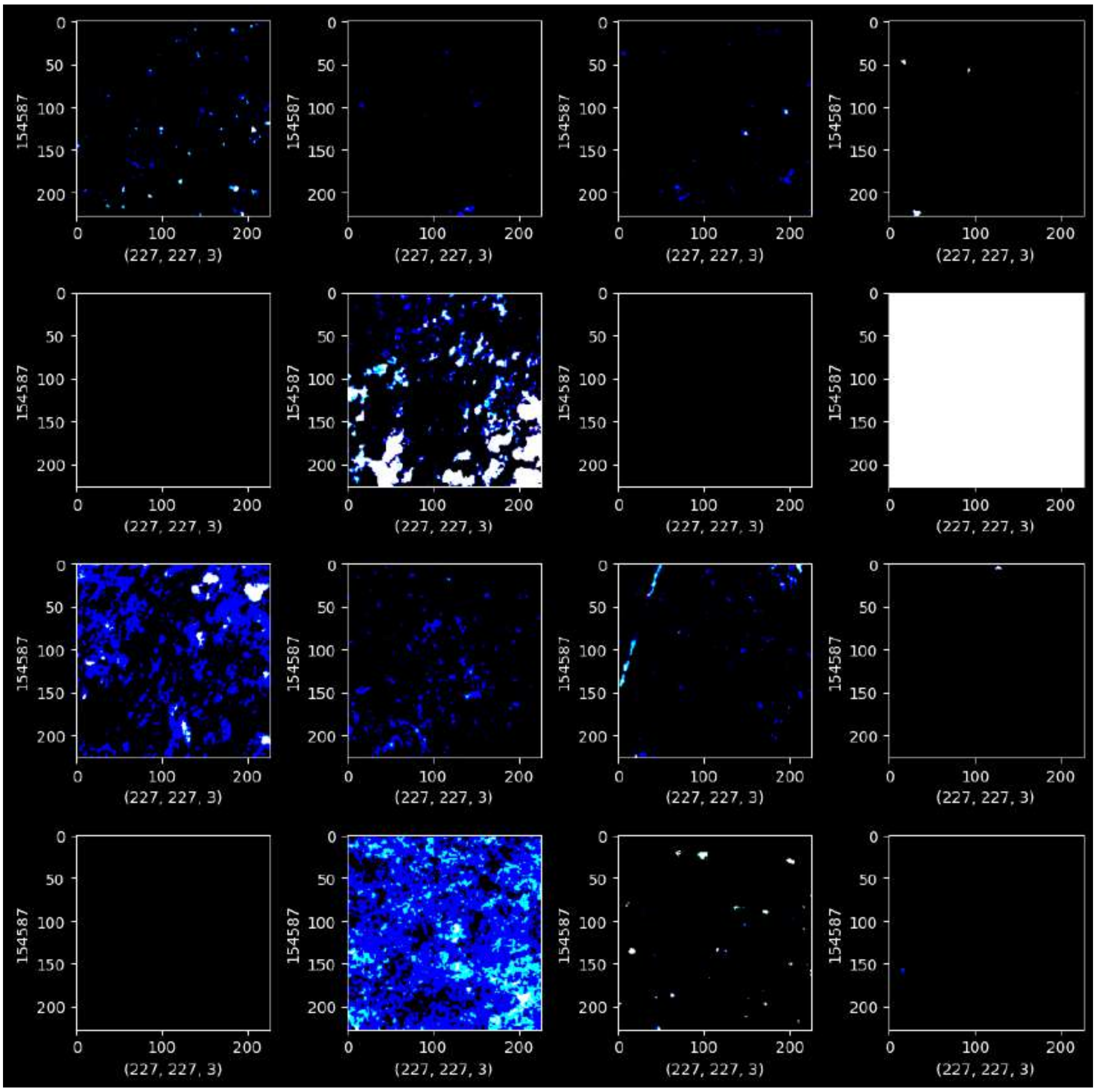





```
figure,axis = plt.subplots(4,4,figsize=(10,10))

for indexing,operations in enumerate(axis.flat):

    Reading_Img = cv2.imread(Positive_Surface["JPG"][indexing])
    Reading_Img = cv2.cvtColor(Reading_Img,cv2.COLOR_BGR2RGB)

    _,Threshold_Img = cv2.threshold(Reading_Img,150,255,cv2.THRESH_BINARY_INV)

    operations.set_xlabel(Threshold_Img.shape)
    operations.set_ylabel(Threshold_Img.size)
    operations.imshow(Threshold_Img)

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



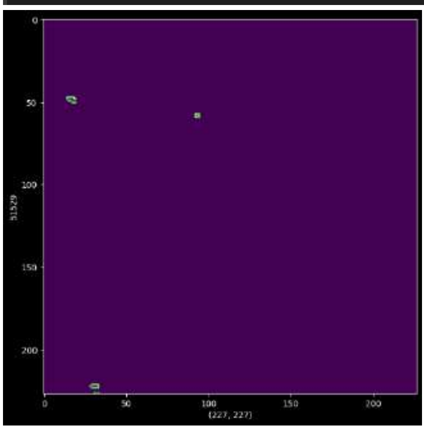
```
figure,axis = plt.subplots(4,4,figsize=(10,10))

for indexing,operations in enumerate(axis.flat):

    Reading_Img = cv2.imread(Negative_Surface["JPG"][indexing])
    Reading_Img = cv2.cvtColor(Reading_Img,cv2.COLOR_BGR2RGB)

    _,Threshold_Img = cv2.threshold(Reading_Img,150,255,cv2.THRESH_BINARY_INV)

    operations.set_xlabel(Threshold_Img.shape)
    operations.set_ylabel(Threshold_Img.size)
    operations.imshow(Threshold_Img)

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

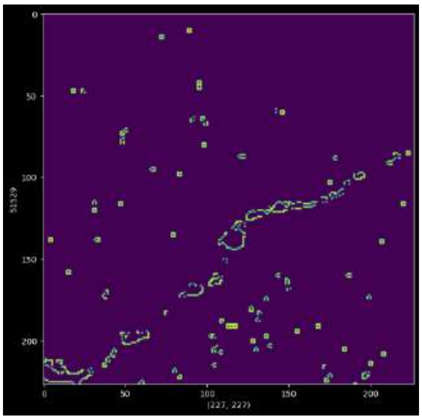


Threshold-Canny vision:

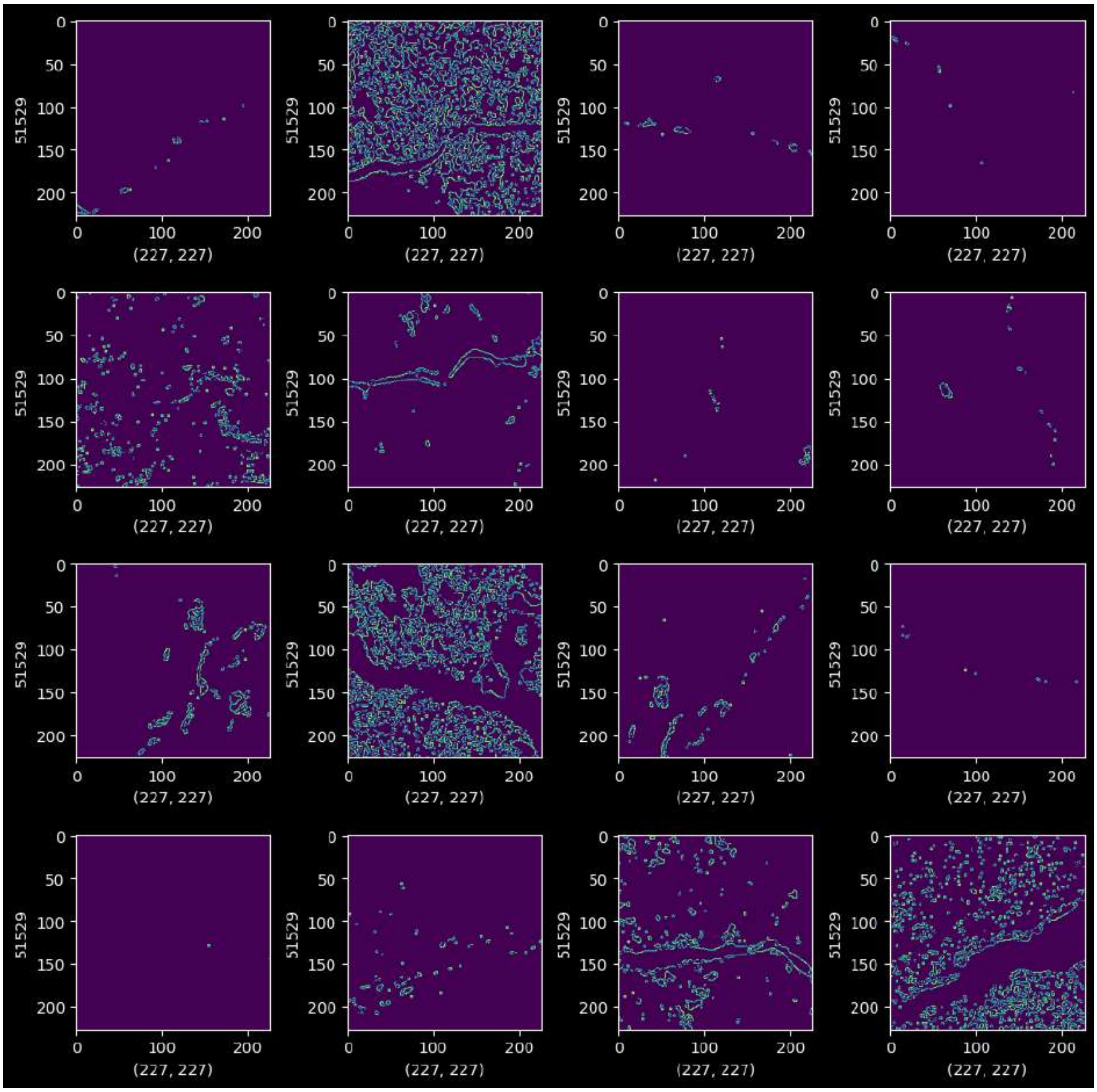
```
threshold_canny(Main_Surface_Data["JPG"][4])
```

```
threshold_canny(Main_Surface_Data["JPG"][2])
```

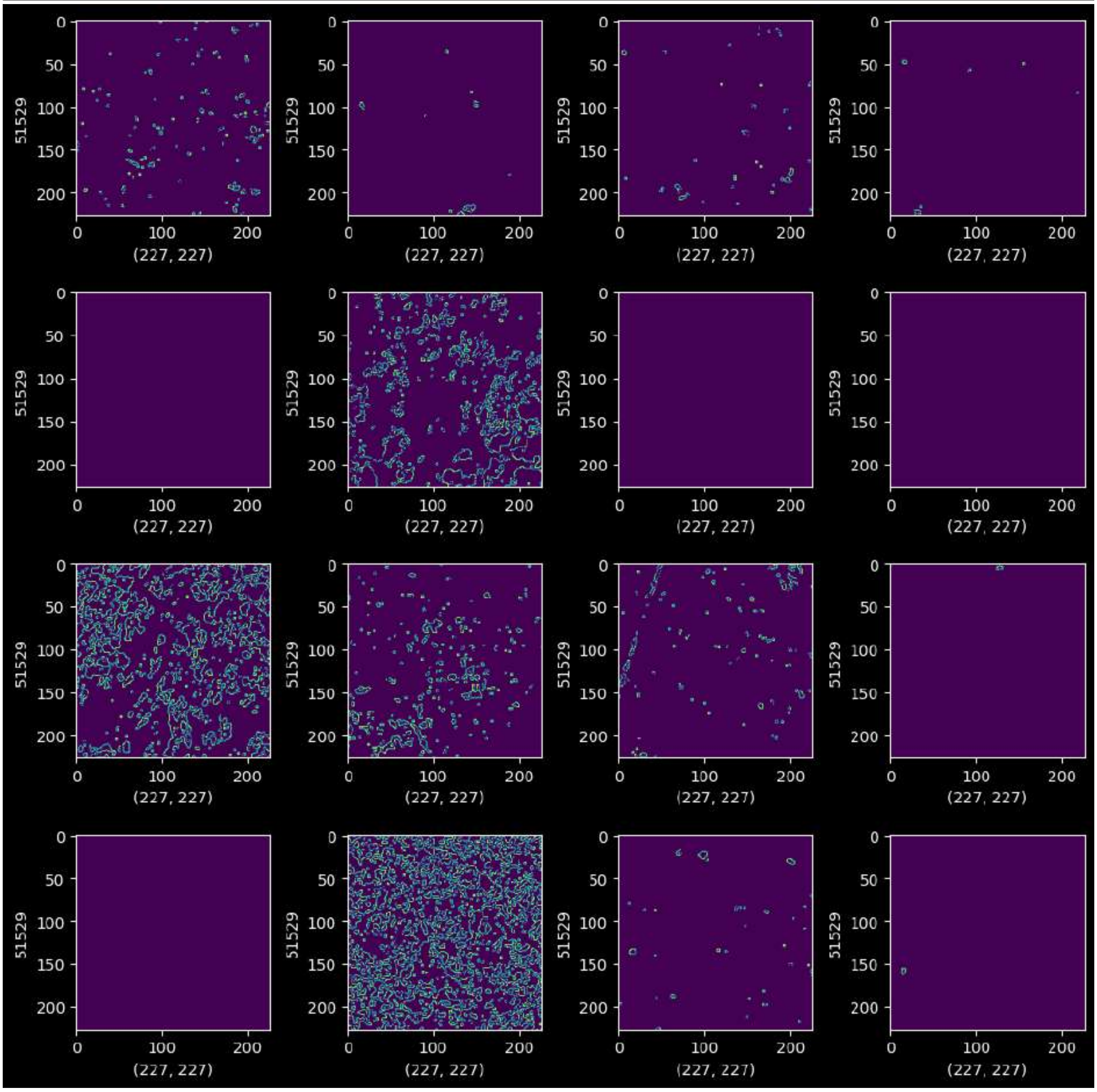




```
figure,axis = plt.subplots(4,4,figsize=(10,10))

for indexing,operations in enumerate(axis.flat):

    Reading_Img = cv2.imread(Positive_Surface["JPG"][indexing])
    Reading_Img = cv2.cvtColor(Reading_Img,cv2.COLOR_BGR2RGB)

    _,Threshold_Img = cv2.threshold(Reading_Img,150,255,cv2.THRESH_BINARY_INV)
    Canny_Img = cv2.Canny(Threshold_Img,90,100)

    operations.set_xlabel(Canny_Img.shape)
    operations.set_ylabel(Canny_Img.size)
    operations.imshow(Canny_Img)

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



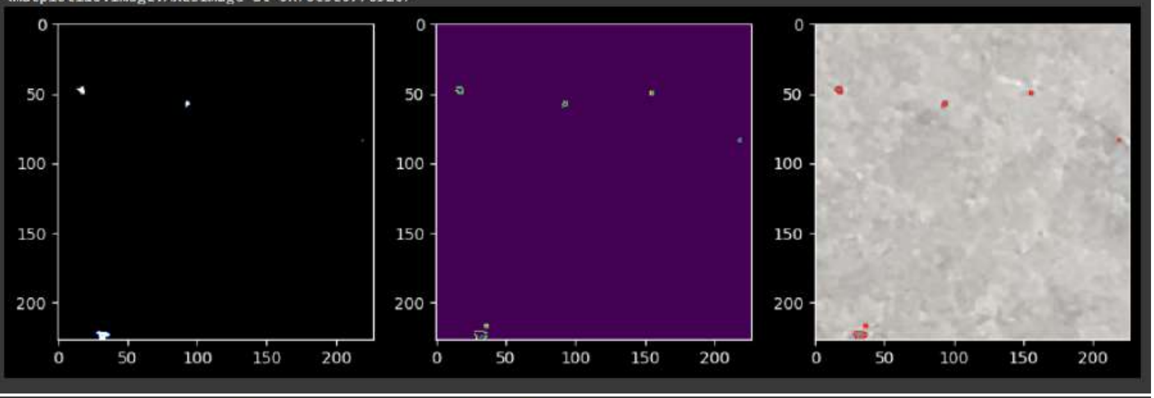
```python
figure,axis = plt.subplots(4,4,figsize=(10,10))

for indexing,operations in enumerate(axis.flat):

    Reading_Img = cv2.imread(Negative_Surface["JPG"][indexing])
    Reading_Img = cv2.cvtColor(Reading_Img,cv2.COLOR_BGR2RGB)

    _,Threshold_Img = cv2.threshold(Reading_Img,150,255,cv2.THRESH_BINARY_INV)
    Canny_Img = cv2.Canny(Threshold_Img,90,100)

    operations.set_xlabel(Canny_Img.shape)
    operations.set_ylabel(Canny_Img.size)
    operations.imshow(Canny_Img)

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



Draw contours:

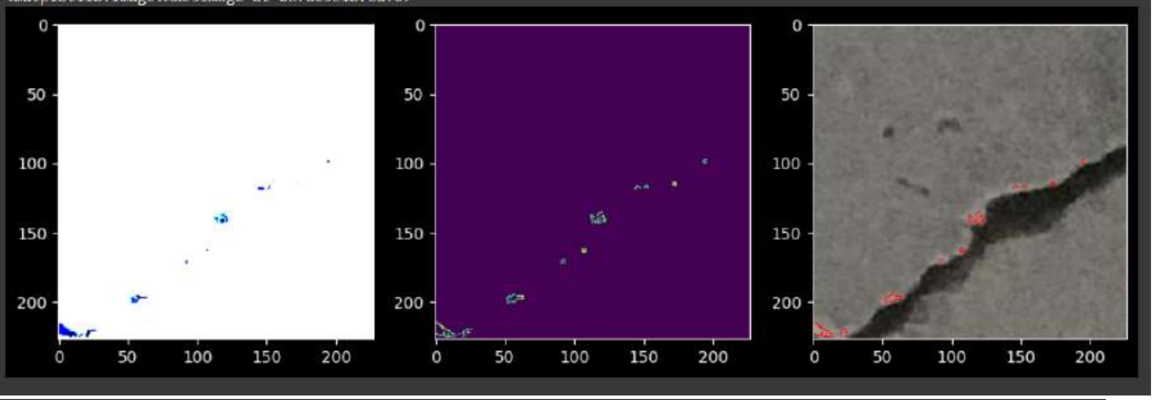
```
[ ] figure,axis = plt.subplots(nrows=1,ncols=3,figsize=(12,12))

    Reading_Img = cv2.imread(Main_Surface_Data["JPG"][4])
    Reading_Img = cv2.cvtColor(Reading_Img,cv2.COLOR_BGR2RGB)

    _,Threshold_Img = cv2.threshold(Reading_Img,150,255,cv2.THRESH_BINARY_INV)
    Canny_Img = cv2.Canny(Threshold_Img,90,100)
    contours,_ = cv2.findContours(Canny_Img,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
    Draw_Contours = cv2.drawContours(Reading_Img,contours,-1,(255,0,0),1)

    axis[0].imshow(Threshold_Img)
    axis[1].imshow(Canny_Img)
    axis[2].imshow(Draw_Contours)
```

<matplotlib.image.AxesImage at 0x78c3e0778520>

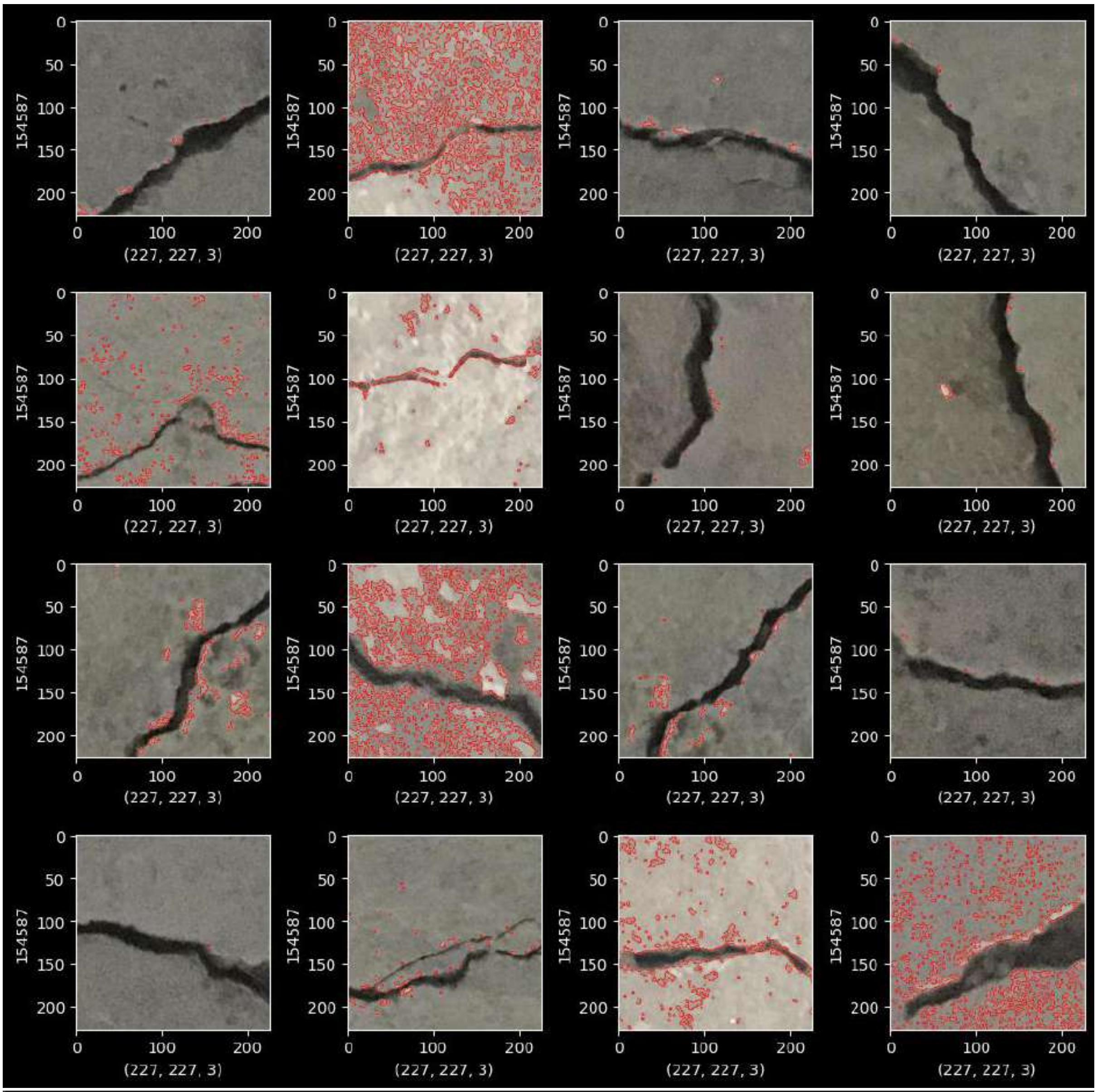


```
figure,axis = plt.subplots(nrows=1,ncols=3,figsize=(12,12))

Reading_Img = cv2.imread(Main_Surface_Data["JPG"][2])
Reading_Img = cv2.cvtColor(Reading_Img,cv2.COLOR_BGR2RGB)

_,Threshold_Img = cv2.threshold(Reading_Img,150,255,cv2.THRESH_BINARY_INV)
Canny_Img = cv2.Canny(Threshold_Img,90,100)
contours,_ = cv2.findContours(Canny_Img,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
Draw_Contours = cv2.drawContours(Reading_Img,contours,-1,(255,0,0),1)

axis[0].imshow(Threshold_Img)
axis[1].imshow(Canny_Img)
axis[2].imshow(Draw_Contours)
```

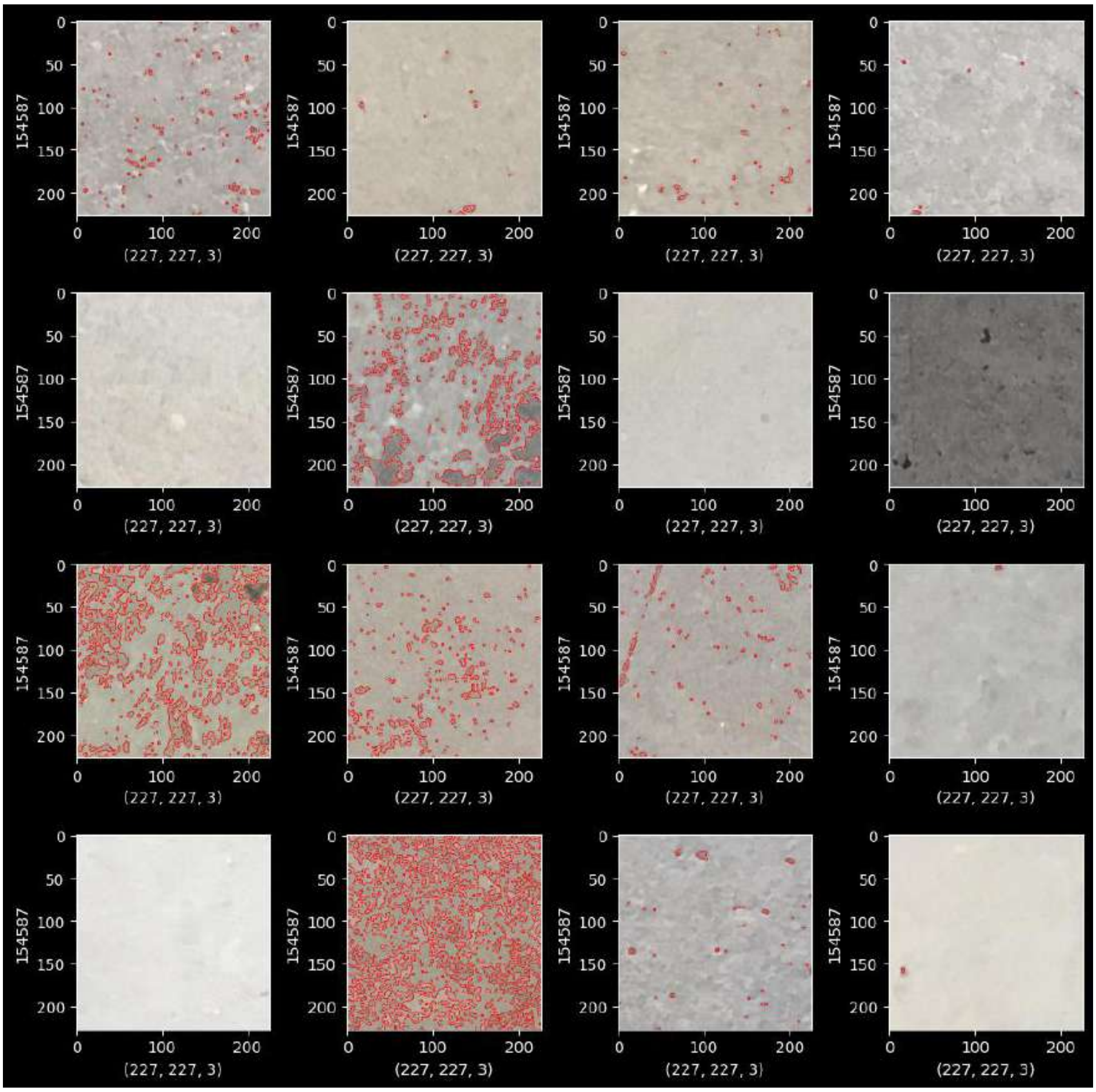
<matplotlib.image.AxesImage at 0x78c3e1b7bdf0>



```
figure,axis = plt.subplots(4,4,figsize=(10,10))

for indexing,operations in enumerate(axis.flat):

    Reading_Img = cv2.imread(Positive_Surface["JPG"][indexing])
    Reading_Img = cv2.cvtColor(Reading_Img,cv2.COLOR_BGR2RGB)

    _,Threshold_Img = cv2.threshold(Reading_Img,150,255,cv2.THRESH_BINARY_INV)
    Canny_Img = cv2.Canny(Threshold_Img,90,100)
    contours,_ = cv2.findContours(Canny_Img,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
    Draw_Contours_Positive = cv2.drawContours(Reading_Img,contours,-1,(255,0,0),1)

    operations.set_xlabel(Draw_Contours_Positive.shape)
    operations.set_ylabel(Draw_Contours_Positive.size)
    operations.imshow(Draw_Contours_Positive)

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



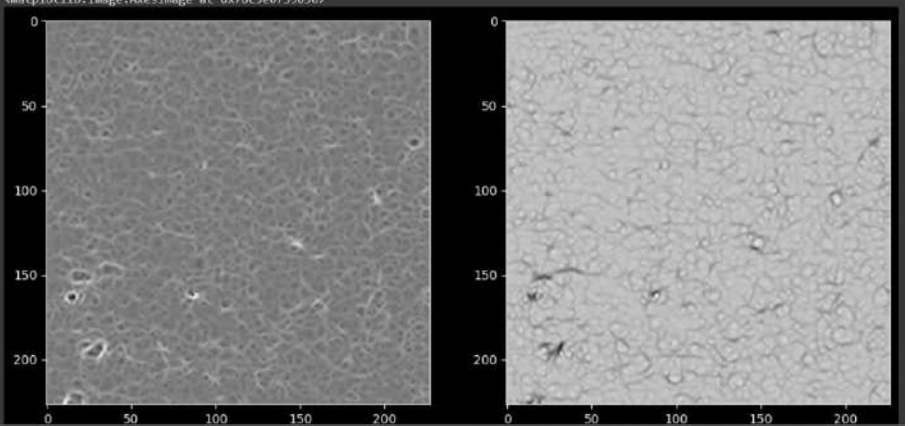

```python
figure,axis = plt.subplots(4,4,figsize=(10,10))

for indexing,operations in enumerate(axis.flat):

    Reading_Img = cv2.imread(Negative_Surface["JPG"][indexing])
    Reading_Img = cv2.cvtColor(Reading_Img,cv2.COLOR_BGR2RGB)

    _,Threshold_Img = cv2.threshold(Reading_Img,150,255,cv2.THRESH_BINARY_INV)
    Canny_Img = cv2.Canny(Threshold_Img,90,100)
    contours,_ = cv2.findContours(Canny_Img,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
    Draw_Contours_Negative = cv2.drawContours(Reading_Img,contours,-1,(255,0,0),1)

    operations.set_xlabel(Draw_Contours_Negative.shape)
    operations.set_ylabel(Draw_Contours_Negative.size)
    operations.imshow(Draw_Contours_Negative)

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



Hessian matrix:

```
figure,axis = plt.subplots(nrows=1,ncols=2,figsize=(12,12))

Reading_Img = cv2.imread(Negative_Surface["JPG"][2])
Reading_Img = cv2.cvtColor(Reading_Img,cv2.COLOR_BGR2GRAY)

Hessian_Matrix_Img = hessian_matrix(Reading_Img,sigma=0.15,order="rc")
maxima_Img,minima_Img = hessian_matrix_eigvals(Hessian_Matrix_Img)
 (variable) axis: Any
axis[0].imshow(maxima_Img,cmap="Greys_r")
axis[1].imshow(minima_Img,cmap="Greys_r")
```

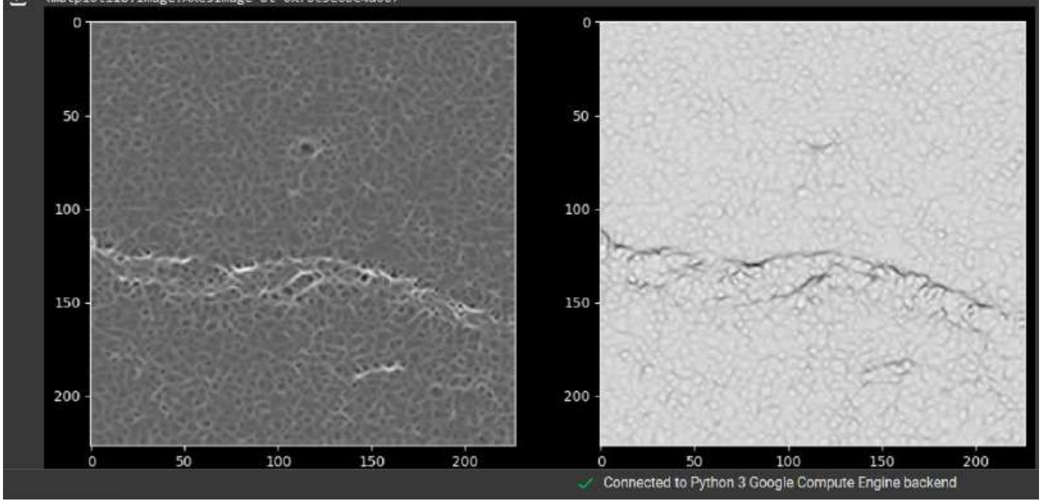
```
<matplotlib.image.AxesImage at 0x78c3e0f59090>
```



Connected to Python 3 Google Compute Engine backend

```
figure,axis = plt.subplots(nrows=1,ncols=2,figsize=(12,12))

Reading_Img = cv2.imread(Positive_Surface["JPG"][2])
Reading_Img = cv2.cvtColor(Reading_Img,cv2.COLOR_BGR2GRAY)

Hessian_Matrix_Img = hessian_matrix(Reading_Img,sigma=0.15,order="rc")
maxima_Img,minima_Img = hessian_matrix_eigvals(Hessian_Matrix_Img)

axis[0].imshow(maxima_Img,cmap="Greys_r")
axis[1].imshow(minima_Img,cmap="Greys_r")
```

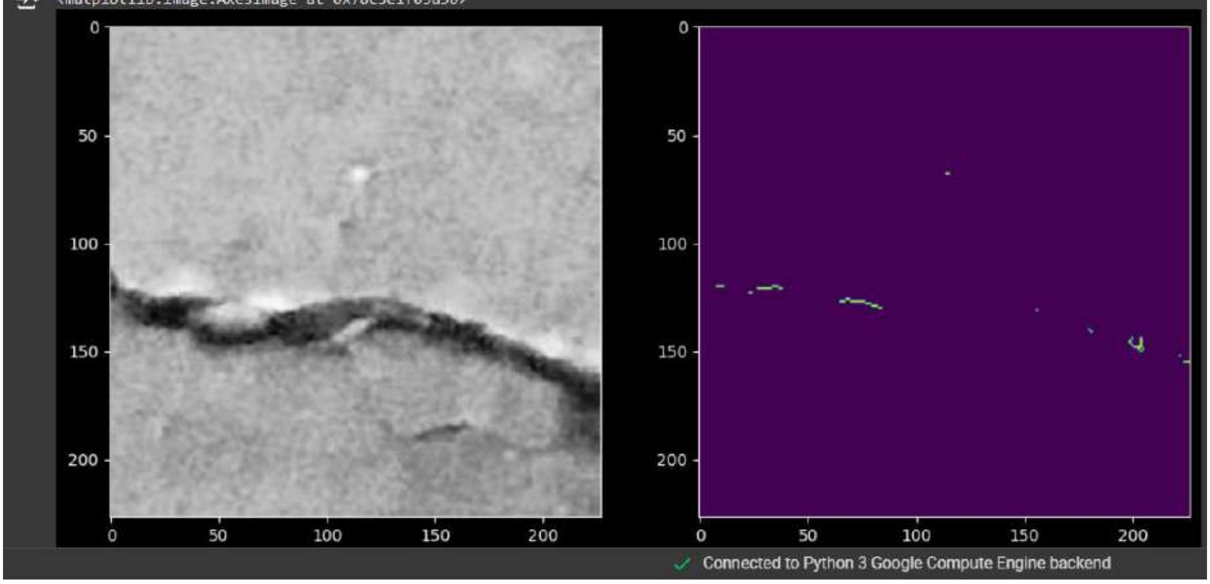
```
<matplotlib.image.AxesImage at 0x78c3e6be4a60>
```



Threshold skeleton morphology:

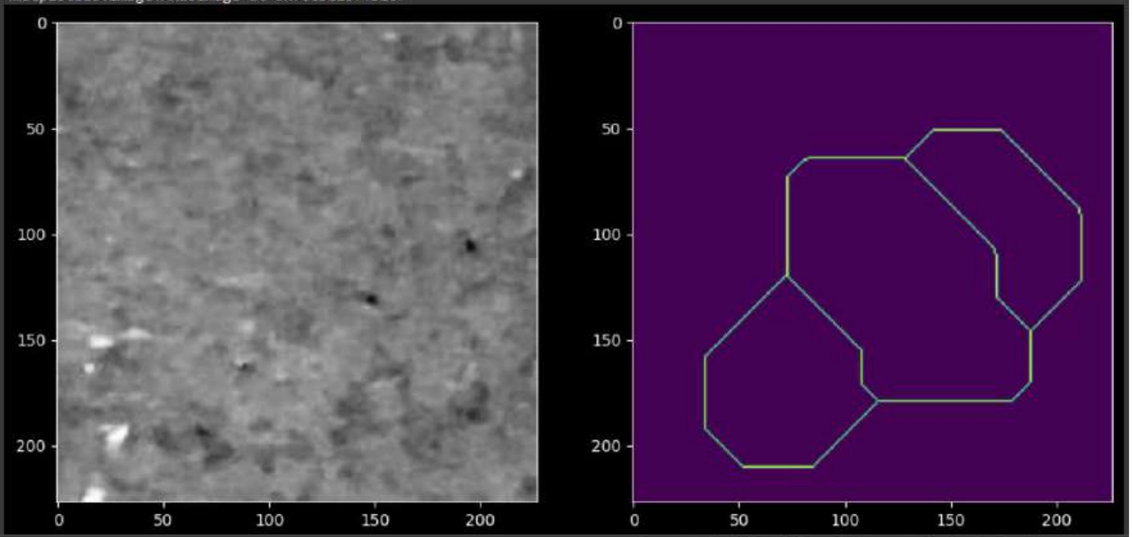
```
figure,axis = plt.subplots(nrows=1,ncols=2,figsize=(12,12))
Reading_Img = cv2.imread(Positive_Surface["JPG"][2])
Reading_Img = cv2.cvtColor(Reading_Img,cv2.COLOR_BGR2GRAY)
_,Threshold_Img = cv2.threshold(Reading_Img,150,255,cv2.THRESH_BINARY_INV)

Array_Img = np.array(Reading_Img > Threshold_Img).astype(int)
Skeleton_Morphology_Img = skimage.morphology.skeletonize(Array_Img)

axis[0].imshow(Reading_Img,cmap="Greys_r")
axis[1].imshow(Skeleton_Morphology_Img)
```

```
<matplotlib.image.AxesImage at 0x78c3e1f03a30>
```



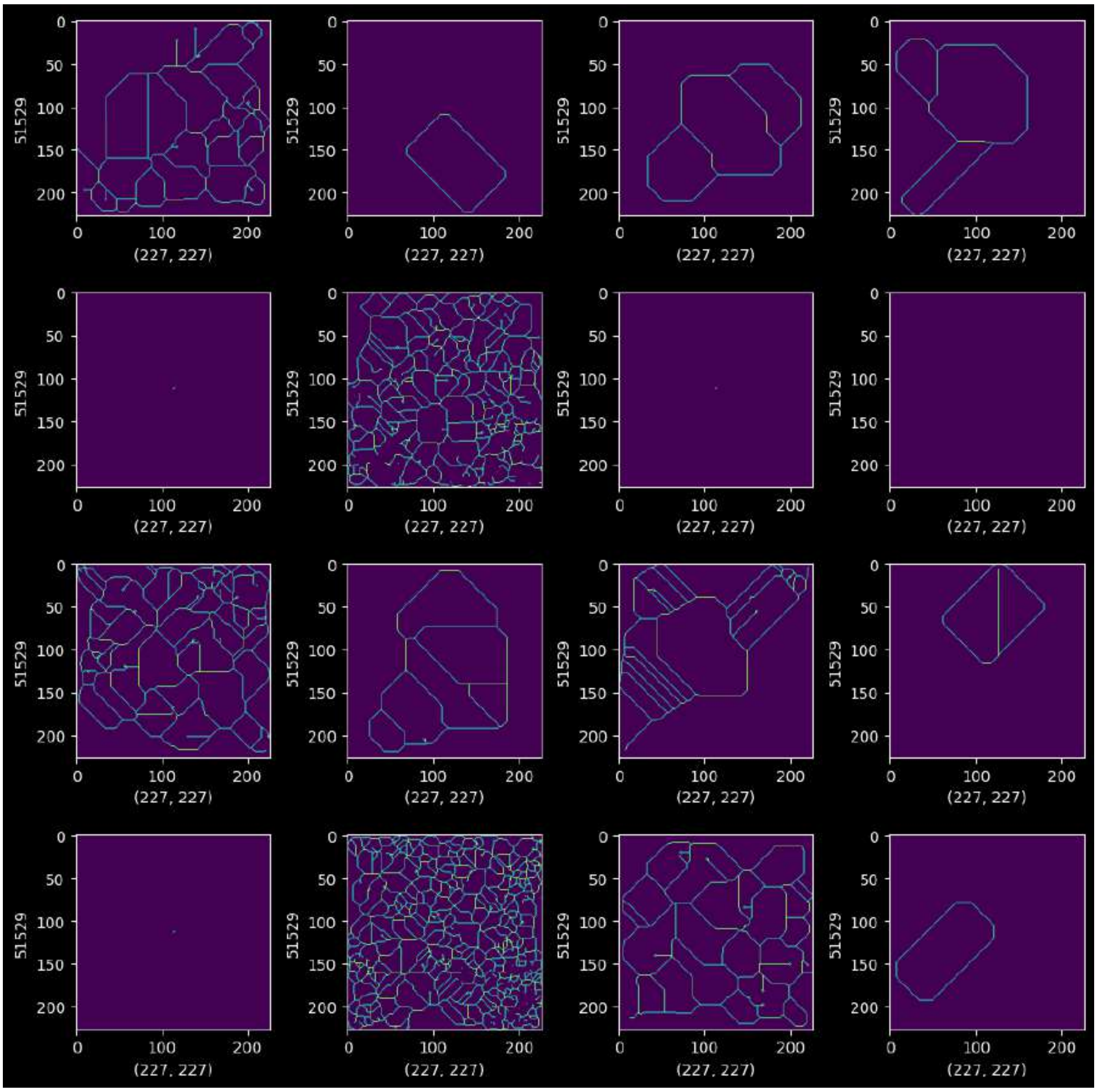
```python
figure,axis = plt.subplots(nrows=1,ncols=2,figsize=(12,12))
Reading_Img = cv2.imread(Negative_Surface["JPG"][2])
Reading_Img = cv2.cvtColor(Reading_Img,cv2.COLOR_BGR2GRAY)
_,Threshold_Img = cv2.threshold(Reading_Img,150,255,cv2.THRESH_BINARY_INV)

Array_Img = np.array(Reading_Img > Threshold_Img).astype(int)
Skeleton_Morphology_Img = skimage.morphology.skeletonize(Array_Img)

axis[0].imshow(Reading_Img,cmap="Greys_r")
axis[1].imshow(Skeleton_Morphology_Img)
```

```
<matplotlib.image.AxesImage at 0x78c3e13ff310>
```

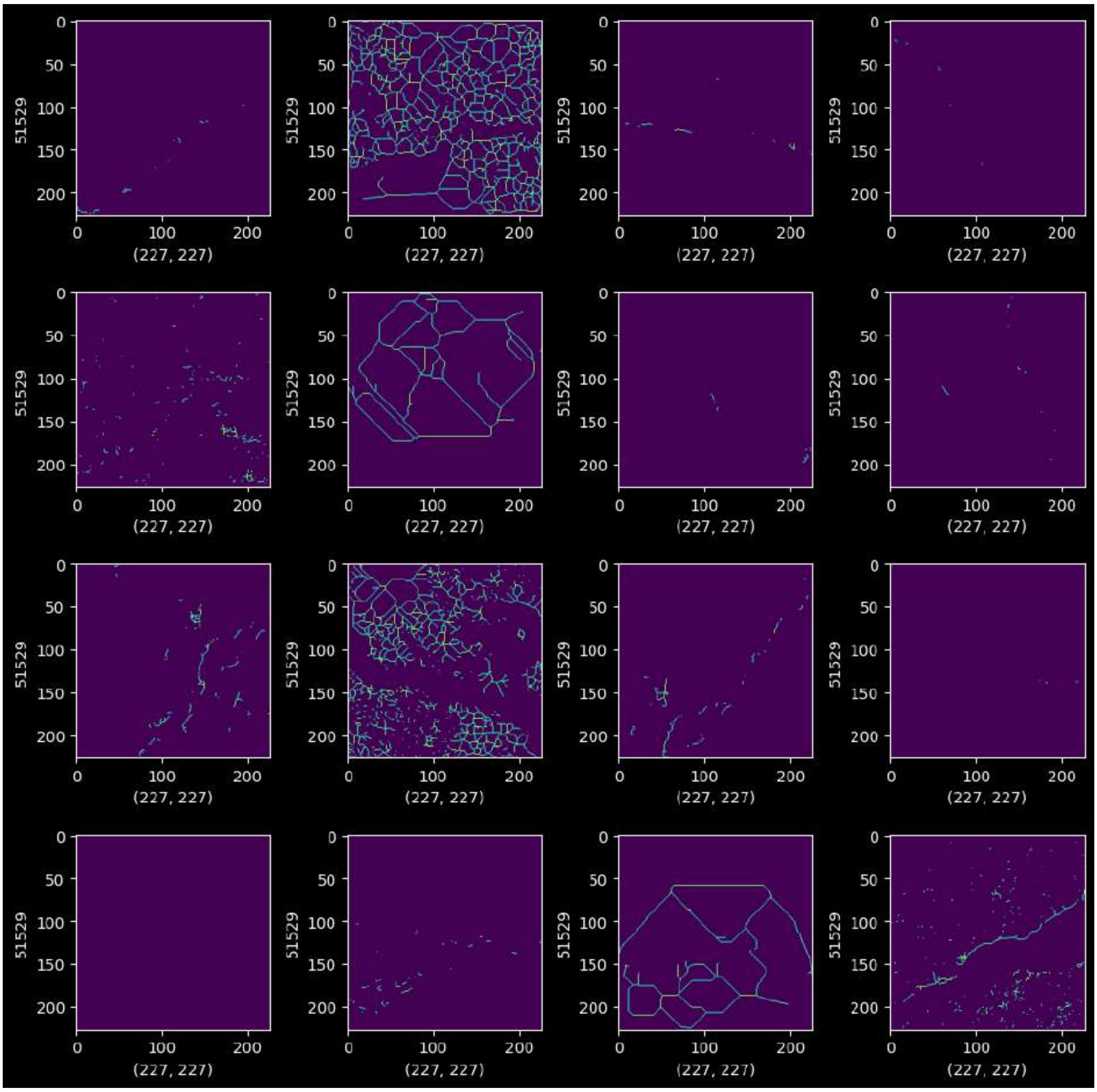


Connected to Python 3 Google Compute Engine backend

```python
figure,axis = plt.subplots(4,4,figsize=(10,10))

for indexing,operations in enumerate(axis.flat):

    Reading_Img = cv2.imread(Negative_Surface["JPG"][indexing])
    Reading_Img = cv2.cvtColor(Reading_Img,cv2.COLOR_BGR2GRAY)

    _,Threshold_Img = cv2.threshold(Reading_Img,150,255,cv2.THRESH_BINARY_INV)
    Array_Img = np.array(Reading_Img > Threshold_Img).astype(int)
    Skeleton_Morphology_Img = skimage.morphology.skeletonize(Array_Img)

    operations.set_xlabel(Skeleton_Morphology_Img.shape)
    operations.set_ylabel(Skeleton_Morphology_Img.size)
    operations.imshow(Skeleton_Morphology_Img)

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

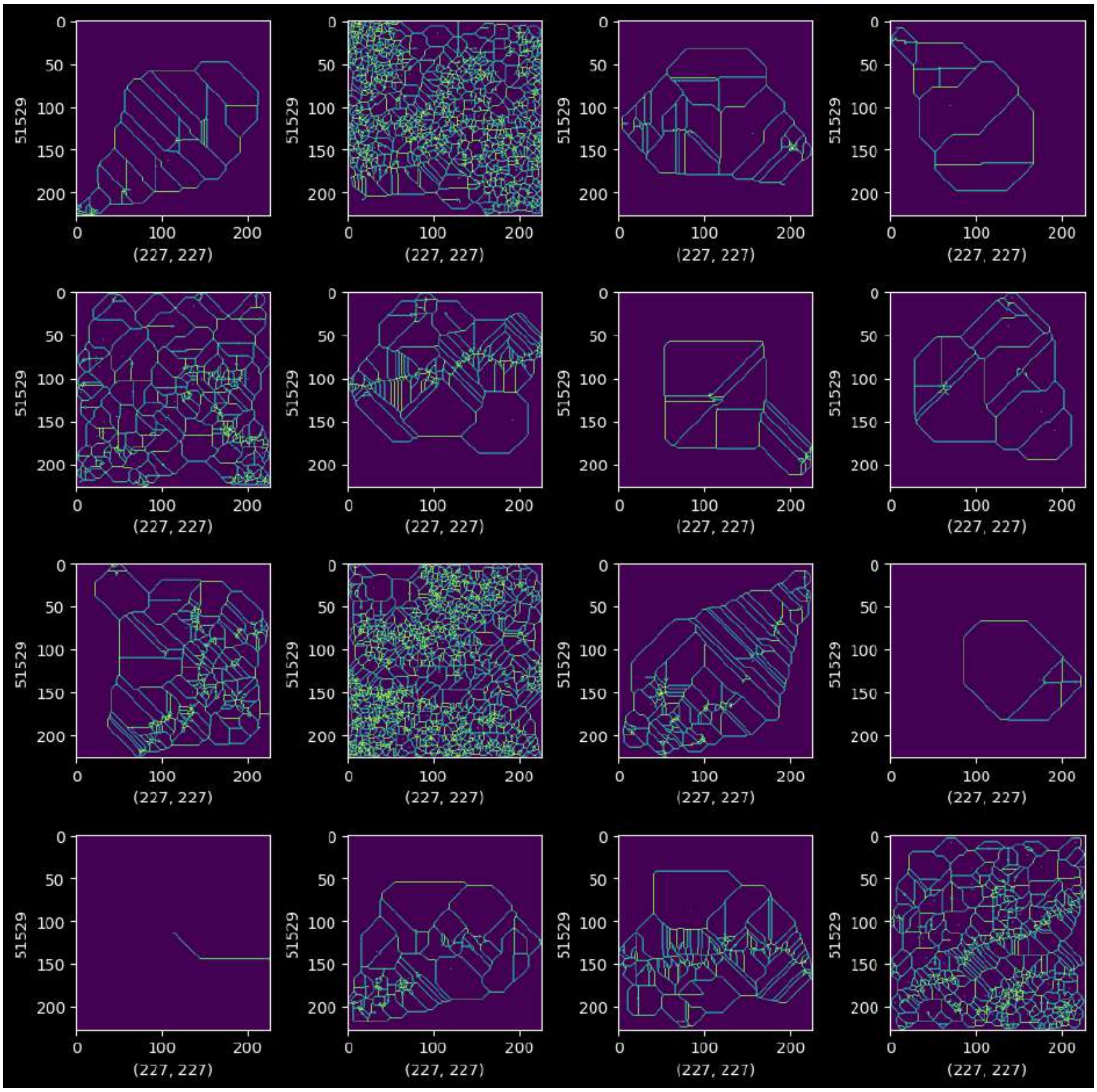


```
figure,axis = plt.subplots(4,4,figsize=(10,10))

for indexing,operations in enumerate(axis.flat):

    Reading_Img = cv2.imread(Positive_Surface["JPG"][indexing])
    Reading_Img = cv2.cvtColor(Reading_Img,cv2.COLOR_BGR2GRAY)

    _,Threshold_Img = cv2.threshold(Reading_Img,150,255,cv2.THRESH_BINARY_INV)
    Array_Img = np.array(Reading_Img > Threshold_Img).astype(int)
    Skeleton_Morphology_Img = skimage.morphology.skeletonize(Array_Img)

    operations.set_xlabel(Skeleton_Morphology_Img.shape)
    operations.set_ylabel(Skeleton_Morphology_Img.size)
    operations.imshow(Skeleton_Morphology_Img)

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

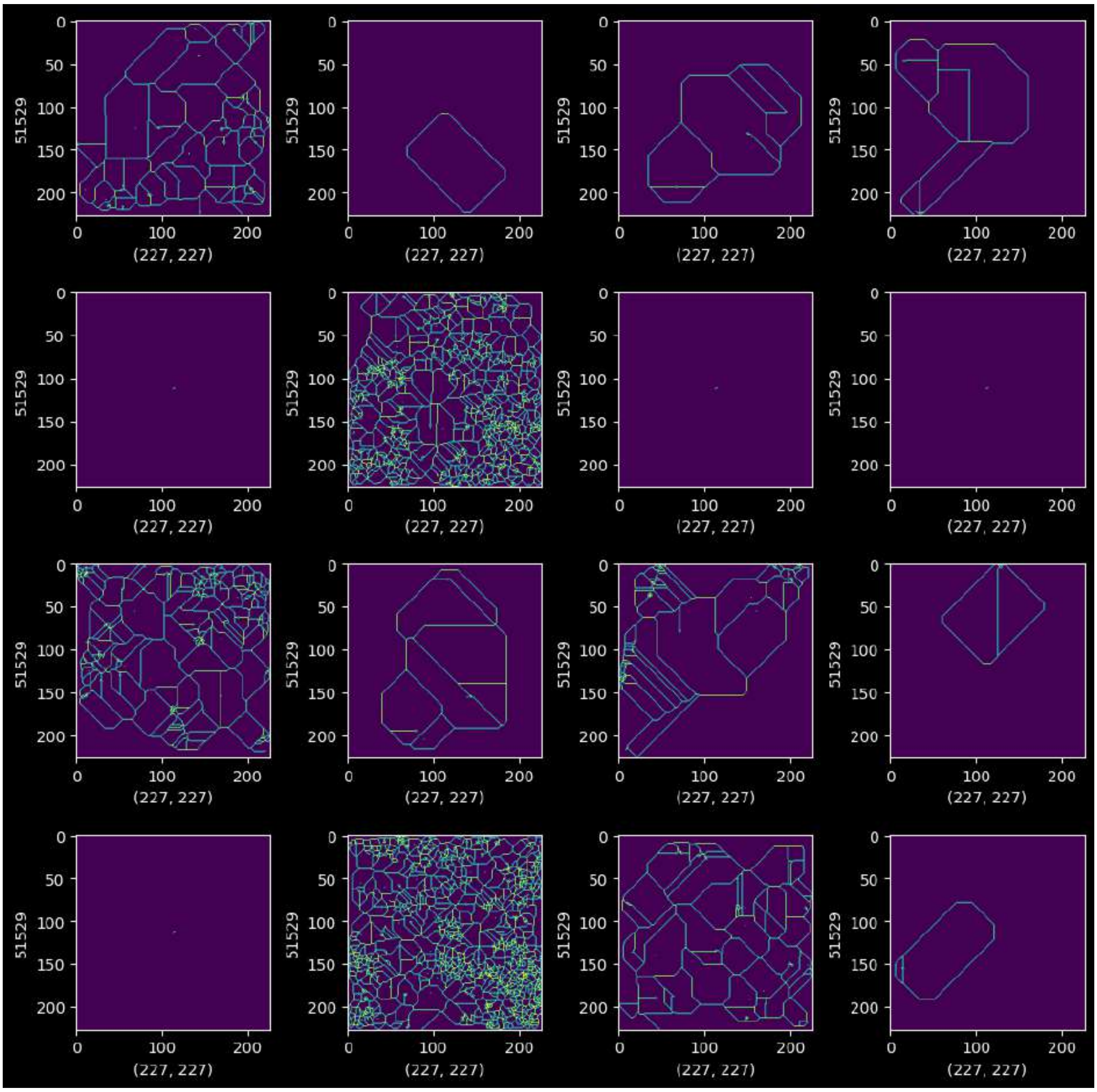


<u>Canny skeleton morphology:</u>

```
[ ]  figure,axis = plt.subplots(4,4,figsize=(10,10))

     for indexing,operations in enumerate(axis.flat):

         Reading_Img = cv2.imread(Positive_Surface["JPG"][indexing])
         Reading_Img = cv2.cvtColor(Reading_Img,cv2.COLOR_BGR2GRAY)

         _,Threshold_Img = cv2.threshold(Reading_Img,150,255,cv2.THRESH_BINARY_INV)
         Canny_Img = cv2.Canny(Threshold_Img,90,100)
         Array_Img = np.array(Reading_Img > Canny_Img).astype(int)
         Skeleton_Morphology_Img = skimage.morphology.skeletonize(Array_Img)

         operations.set_xlabel(Skeleton_Morphology_Img.shape)
         operations.set_ylabel(Skeleton_Morphology_Img.size)
         operations.imshow(Skeleton_Morphology_Img)

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



```python
figure,axis = plt.subplots(4,4,figsize=(10,10))

for indexing,operations in enumerate(axis.flat):

    Reading_Img = cv2.imread(Negative_Surface["JPG"][indexing])
    Reading_Img = cv2.cvtColor(Reading_Img,cv2.COLOR_BGR2GRAY)

    _,Threshold_Img = cv2.threshold(Reading_Img,150,255,cv2.THRESH_BINARY_INV)
    Canny_Img = cv2.Canny(Threshold_Img,90,100)
    Array_Img = np.array(Reading_Img > Canny_Img).astype(int)
    Skeleton_Morphology_Img = skimage.morphology.skeletonize(Array_Img)

    operations.set_xlabel(Skeleton_Morphology_Img.shape)
    operations.set_ylabel(Skeleton_Morphology_Img.size)
    operations.imshow(Skeleton_Morphology_Img)

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



Splitting train and test:

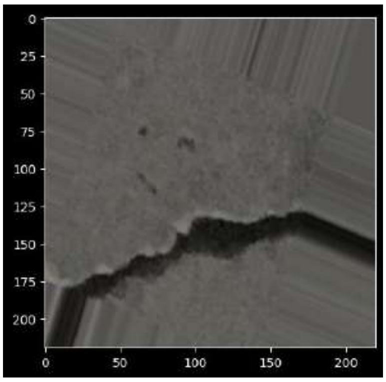
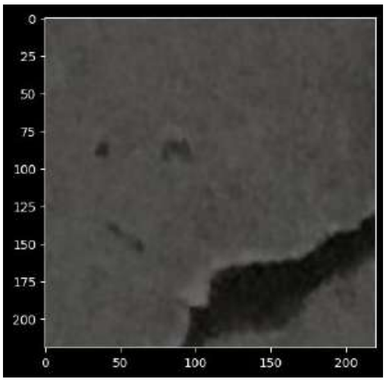
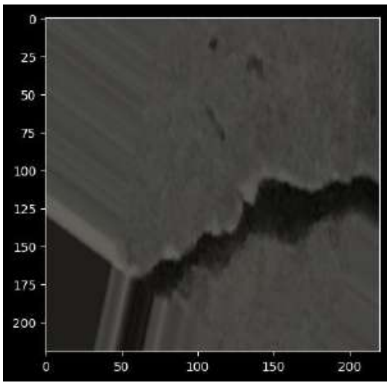
```
[ ] xTrain,xTest = train_test_split(Main_Surface_Data,train_size=0.9,shuffle=True,random_state=42)

[ ] print(xTrain.shape)
    print(xTest.shape)

    (2423, 2)
    (270, 2)
```

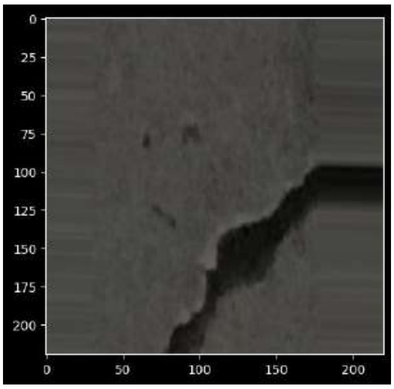
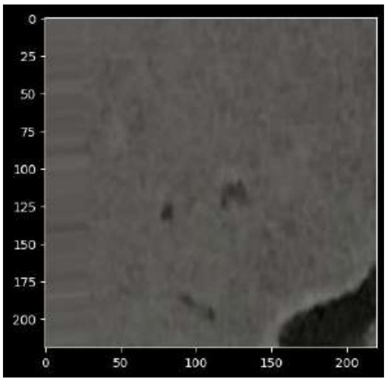
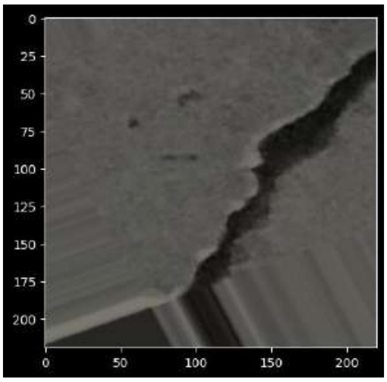
Image generator:

```
Train_IMG_Generator = ImageDataGenerator(rescale=1./255,
                                         rotation_range=25,
                                         shear_range=0.5,
                                         zoom_range=0.5,
                                         width_shift_range=0.2,
                                         height_shift_range=0.2,
                                         brightness_range=[0.6,0.9],
                                         vertical_flip=True,
                                         validation_split=0.1)
```

```
Test_IMG_Generator = ImageDataGenerator(rescale=1./255)
```

```
Example_Surface_Img = Main_Surface_Data["JPG"][2]
Loading_Img = image.load_img(Example_Surface_Img,target_size=(220,220))
Array_Img = image.img_to_array(Loading_Img)
Array_Img = Array_Img.reshape((1,) + Array_Img.shape)

i = 0

for batch in Train_IMG_Generator.flow(Array_Img,batch_size=32):
    plt.figure(i)
    Image_Out = plt.imshow(image.img_to_array(batch[0]))
    i += 1

    if i % 6 == 0:
        break
```

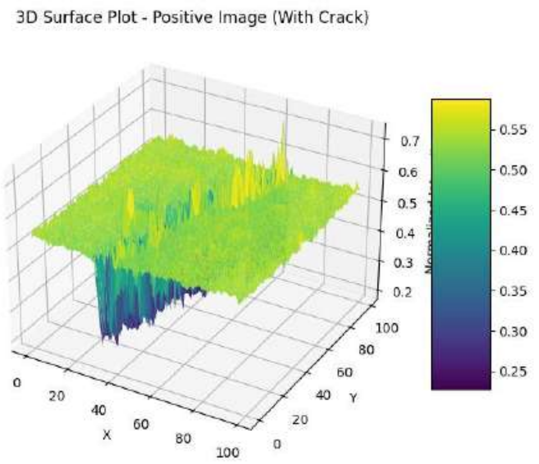




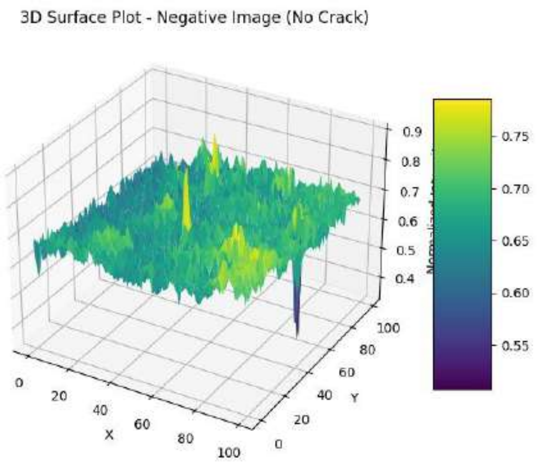




3. **3D plot:**

The 3D surface plot employs a height-mapped visualization where the Z-axis represents the crack intensity, while the X and Y axes correspond to the spatial coordinates on the surface. The intensity is calculated using a sliding window approach, where for each local region, the density of crack pixels is computed relative to the total area of the window. This creates a continuous surface where peaks indicate areas of high crack concentration, and valleys represent regions with minimal or no cracking.

Color mapping further enhances the visualization, typically utilizing a carefully chosen colormap (such as 'viridis' or 'jet') where darker colors might represent lower intensities and brighter or warmer colors indicate higher crack concentrations. This dual representation - through both height and color - provides redundant visual cues that make it easier to identify critical areas requiring attention.





Applying:

```
[ ] Train_Set = Train_IMG_Generator.flow_from_dataframe(dataframe=xTrain,
                                                          x_col="JPG",
                                                          y_col="CATEGORY",
                                                          color_mode="rgb",
                                                          class_mode="binary",
                                                          target_size=(200,200),
                                                          subset="training",
                                                           batch_size=32,
                                                           seed=32)
```

```
Found 2181 validated image filenames belonging to 2 classes.
```

```
Validation_Set = Train_IMG_Generator.flow_from_dataframe(dataframe=xTrain,
                                                          x_col="JPG",
                                                          y_col="CATEGORY",
                                                          color_mode="rgb",
                                                          class_mode="binary",
                                                          target_size=(200,200),
                                                          subset="validation",
                                                           batch_size=32,
                                                           seed=32)
```

```
Found 242 validated image filenames belonging to 2 classes.
```

```
[ ] Test_Set = Test_IMG_Generator.flow_from_dataframe(dataframe=xTest,
                                                          x_col="JPG",
                                                          y_col="CATEGORY",
                                                          color_mode="rgb",
                                                          class_mode="binary",
                                                          target_size=(200,200),
                                                           batch_size=32,
                                                           seed=32)
```

```
Found 270 validated image filenames belonging to 2 classes.
```

Testing :

```
print("TRAIN: ")
print(Train_Set.class_indices)
print(Train_Set.classes[0:5])
print(Train_Set.image_shape)
print("---"*20)
print("VALIDATION: ")
print(Validation_Set.class_indices)
print(Validation_Set.classes[0:5])
print(Validation_Set.image_shape)
print("---"*20)
print("TEST: ")
print(Test_Set.class_indices)
print(Test_Set.classes[0:5])
print(Test_Set.image_shape)
```

```
TRAIN:
{'Negative': 0, 'Positive': 1}
[0, 1, 0, 0, 1]
(200, 200, 3)
------------------------------------------------------------
VALIDATION:
{'Negative': 0, 'Positive': 1}
[0, 0, 0, 0, 1]
(200, 200, 3)
------------------------------------------------------------
TEST:
{'Negative': 0, 'Positive': 1}
[1, 0, 0, 0, 0]
(200, 200, 3)
```

```
[ ] print(Train_Set.image_shape[0],Train_Set.image_shape[1],Train_Set.image_shape[2])
```

```
200 200 3
```

```
[ ] compile_optimizer = "adam"
    compile_loss = "binary_crossentropy"
    input_dim = (Train_Set.image_shape[0],Train_Set.image_shape[1],Train_Set.image_shape[2])
    class_dim = 1
```

```
Early_Stopper = tf.keras.callbacks.EarlyStopping(monitor="loss", patience=3, mode="min")

Checkpoint_Model = tf.keras.callbacks.ModelCheckpoint(
    monitor="val_accuracy",
    save_best_only=True,
    save_weights_only=True,
    filepath="./modelcheck.weights.h5"  # Modified to end with `.weights.h5`
)
```

```
Model = Sequential()

Model.add(Conv2D(32,(3,3),activation="relu",input_shape=input_dim))
Model.add(BatchNormalization())
Model.add(MaxPooling2D((2,2),strides=2))

Model.add(Conv2D(64,(3,3),activation="relu",padding="same"))
Model.add(Dropout(0.3))
Model.add(MaxPooling2D((2,2),strides=2))

Model.add(Conv2D(128,(3,3),activation="relu",padding="same"))
Model.add(Dropout(0.3))
Model.add(MaxPooling2D((2,2),strides=2))

Model.add(Conv2D(256,(3,3),activation="relu",padding="same"))
Model.add(Dropout(0.3))
Model.add(MaxPooling2D((2,2),strides=2))

Model.add(Flatten())
Model.add(Dense(1024,activation="relu"))
Model.add(Dropout(0.5))
Model.add(Dense(class_dim,activation="sigmoid"))
```

```
[ ] Model.compile(optimizer=compile_optimizer,loss=compile_loss,metrics=["accuracy"])
```

```
CNN_Model = Model.fit(Train_Set,
                      validation_data=Validation_Set,
                      callbacks=[Early_Stopper,Checkpoint_Model],
                      epochs=10)
```

```
Epoch 1/10
69/69 ━━━━━━━━━━━━━━━━━━━━ 2179s 28s/step - accuracy: 0.7824 - loss: 1.5678 - val_accuracy: 0.8306 - val_loss: 0.5290
Epoch 2/10
69/69 ━━━━━━━━━━━━━━━━━━━━ 41s 485ms/step - accuracy: 0.9508 - loss: 0.1429 - val_accuracy: 0.8306 - val_loss: 0.5854
Epoch 3/10
69/69 ━━━━━━━━━━━━━━━━━━━━ 37s 489ms/step - accuracy: 0.9796 - loss: 0.0620 - val_accuracy: 0.8306 - val_loss: 0.4939
Epoch 4/10
69/69 ━━━━━━━━━━━━━━━━━━━━ 36s 483ms/step - accuracy: 0.9884 - loss: 0.0321 - val_accuracy: 0.8306 - val_loss: 0.5499
Epoch 5/10
69/69 ━━━━━━━━━━━━━━━━━━━━ 37s 475ms/step - accuracy: 0.9807 - loss: 0.0559 - val_accuracy: 0.8306 - val_loss: 0.7117
Epoch 6/10
69/69 ━━━━━━━━━━━━━━━━━━━━ 39s 531ms/step - accuracy: 0.9863 - loss: 0.0550 - val_accuracy: 0.8430 - val_loss: 0.3594
Epoch 7/10
69/69 ━━━━━━━━━━━━━━━━━━━━ 41s 548ms/step - accuracy: 0.9850 - loss: 0.0607 - val_accuracy: 0.9091 - val_loss: 0.2093
```

```python
print(Model.summary())
```

Model: "sequential"

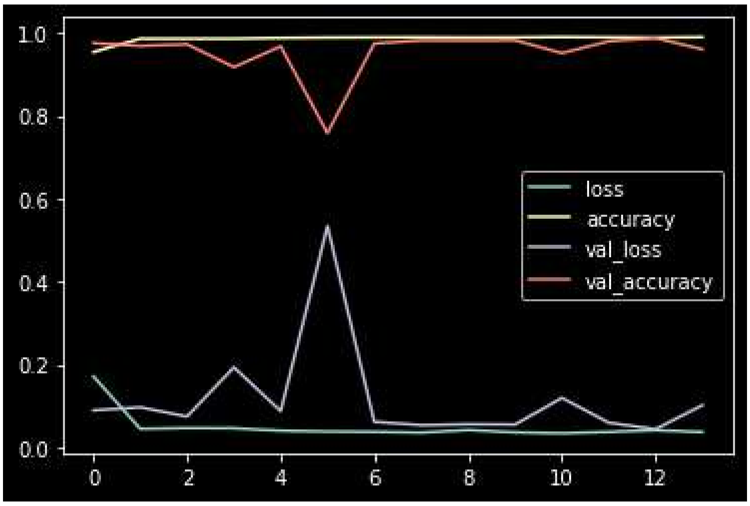
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 198, 198, 32)	896
batch_normalization (BatchNormalization)	(None, 198, 198, 32)	128
max_pooling2d (MaxPooling2D)	(None, 99, 99, 32)	0
conv2d_1 (Conv2D)	(None, 99, 99, 64)	18,496
dropout (Dropout)	(None, 99, 99, 64)	0
max_pooling2d_1 (MaxPooling2D)	(None, 49, 49, 64)	0
conv2d_2 (Conv2D)	(None, 49, 49, 128)	73,856
dropout_1 (Dropout)	(None, 49, 49, 128)	0
max_pooling2d_2 (MaxPooling2D)	(None, 24, 24, 128)	0
conv2d_3 (Conv2D)	(None, 24, 24, 256)	295,168
dropout_2 (Dropout)	(None, 24, 24, 256)	0
max_pooling2d_3 (MaxPooling2D)	(None, 12, 12, 256)	0
flatten (Flatten)	(None, 36864)	0
dense (Dense)	(None, 1024)	37,749,760
dropout_3 (Dropout)	(None, 1024)	0
dense_1 (Dense)	(None, 1)	1,025

Total params: 114,417,861 (436.47 MB)
Trainable params: 38,139,265 (145.49 MB)
Non-trainable params: 64 (256.00 B)

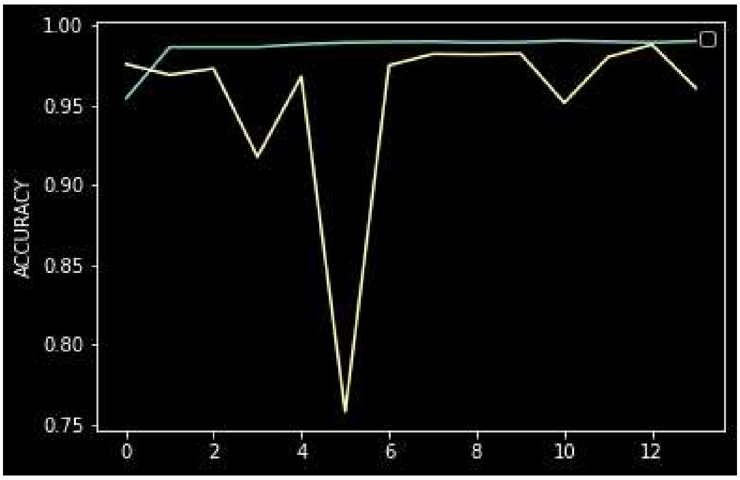
```python
Model.save("Model_Last_Prediction.h5")
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`.
```

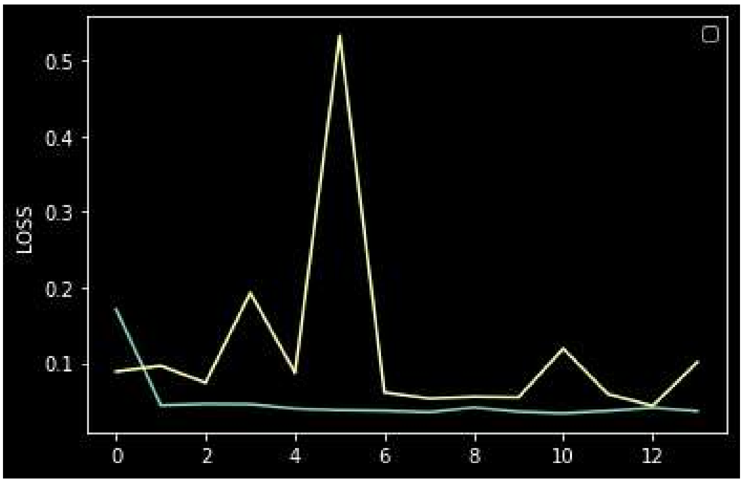
```python
Grap_Data = pd.DataFrame(CNN_Model.history)
Grap_Data.plot()
```



```
[ ] plt.plot(CNN_Model.history["accuracy"])
    plt.plot(CNN_Model.history["val_accuracy"])
    plt.ylabel("ACCURACY")
    plt.legend()
    plt.show()
```




```
[ ] plt.plot(CNN_Model.history["loss"])
    plt.plot(CNN_Model.history["val_loss"])
    plt.ylabel("LOSS")
    plt.legend()
    plt.show()
```



Prediction process:

```
[ ] Model_Results = Model.evaluate(Test_Set)
    print("LOSS:  " + "%.4f" % Model_Results[0])
    print("ACCURACY:  " + "%.2f" % Model_Results[1])
```

```
9/9 ━━━━━━━━━━━━━━━━━━━━ 190s 24s/step - accuracy: 0.9200 - loss: 0.2000
LOSS:  0.2104
ACCURACY:  0.92
```

```
Model_Test_Prediction = Model.predict(Test_Set)
Model_Test_Prediction = Model_Test_Prediction.argmax(axis=-1)
print(Model_Test_Prediction)
```

```
9/9 ━━━━━━━━━━━━━━━━━━━━ 2s 140ms/step
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0]
```

```
[ ] Model_Test_Prediction_Probabilities = Model.predict(Test_Set)
    Model_Test_Prediction_Classes = np.argmax(Model_Test_Prediction_Probabilities, axis=1)
    print(Model_Test_Prediction_Classes)
```

```
9/9 ━━━━━━━━━━━━━━━━━━━━ 1s 106ms/step
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0]
```

✓ Connected to Pyth

```
fig, axes = plt.subplots(nrows=5,
                         ncols=5,
                         figsize=(20, 20),
                        subplot_kw={'xticks': [], 'yticks': []})

for i, ax in enumerate(axes.flat):
    ax.imshow(plt.imread(xTest["JPG"].iloc[i]))
    ax.set_title(f"PREDICTION:{Model_Test_Prediction_Classes[i]}")
    ax.set_xlabel(xTest["CATEGORY"].iloc[i])
plt.tight_layout()
plt.show()
```

Implementation using fuzzy logic:

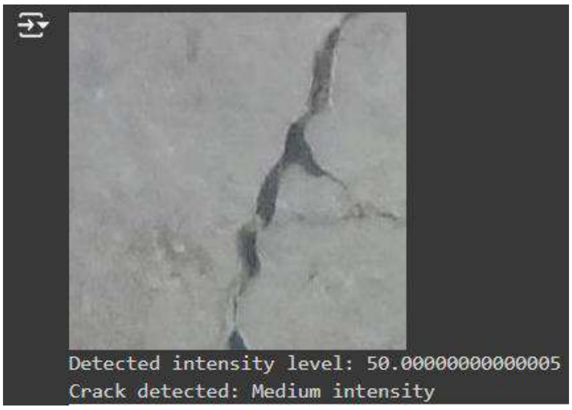
```
import numpy as np
import pandas as pd
!pip install scikit-fuzzy
import skfuzzy as fuzz
from skfuzzy import control as ctrl
from google.colab import drive
import zipfile
import os
import cv2
```

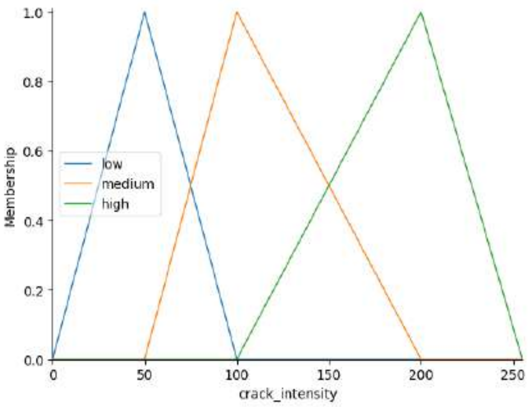
```
Collecting scikit-fuzzy
  Downloading scikit_fuzzy-0.5.0-py2.py3-none-any.whl.metadata (2.6 kB)
Downloading scikit_fuzzy-0.5.0-py2.py3-none-any.whl (920 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 920.8/920.8 kB 8.6 MB/s eta 0:00:00
Installing collected packages: scikit-fuzzy
Successfully installed scikit-fuzzy-0.5.0
```

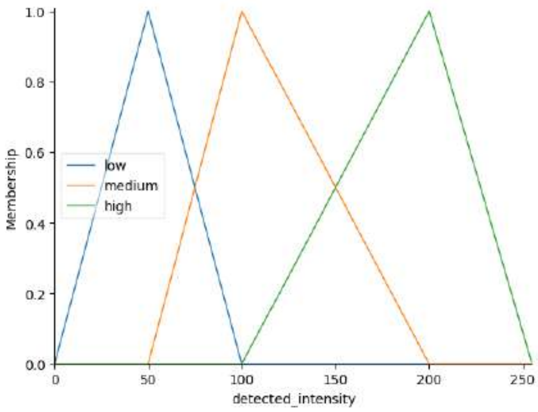
Asigning membership function to the surface :

```
[ ] from google.colab.patches import cv2_imshow
    import cv2
    import numpy as np
    import skfuzzy as fuzz
    from skfuzzy import control as ctrl
    import matplotlib.pyplot as plt

    # Load the image
    image_path = '/content/drive/MyDrive/Surface crack/Positive/00002.jpg'  # Update with
    image = cv2.imread(image_path)

    # Display the image using cv2_imshow
    cv2_imshow(image)

    # Preprocess the image (if needed)
    # Replace this with your actual preprocessing steps
    processed_image = cv2.resize(image, (224, 224))
    processed_image = processed_image.astype(np.float32) / 255.0  # Normalize to [0, 1]

    # Example fuzzy logic for crack detection
    # Define input variable (crack intensity)
    crack_intensity = ctrl.Antecedent(np.arange(0, 256, 1), 'crack_intensity')

    # Define membership functions for crack intensity
    crack_intensity['low'] = fuzz.trimf(crack_intensity.universe, [0, 50, 100])
    crack_intensity['medium'] = fuzz.trimf(crack_intensity.universe, [50, 100, 200])
    crack_intensity['high'] = fuzz.trimf(crack_intensity.universe, [100, 200, 255])

    # Define output variable (detected intensity)
    detected_intensity = ctrl.Consequent(np.arange(0, 256, 1), 'detected_intensity')

    # Define membership functions for detected intensity (output)
    # You need to define appropriate membership functions for the output
    detected_intensity['low'] = fuzz.trimf(detected_intensity.universe, [0, 50, 100])
    detected_intensity['medium'] = fuzz.trimf(detected_intensity.universe, [50, 100, 200])
    detected_intensity['high'] = fuzz.trimf(detected_intensity.universe, [100, 200, 255])

    # Define rules
    rule1 = ctrl.Rule(crack_intensity['low'], detected_intensity['low'])
    rule2 = ctrl.Rule(crack_intensity['medium'], detected_intensity['medium'])
    rule3 = ctrl.Rule(crack_intensity['high'], detected_intensity['high'])

    # Create a fuzzy control system
    crack_detection_ctrl = ctrl.ControlSystem([rule1, rule2, rule3])

    # Create a control system simulation
    crack_detection_sim = ctrl.ControlSystemSimulation(crack_detection_ctrl)
```

```
# Pass the input (mean intensity) to the control system simulation
mean_intensity = np.mean(processed_image)
crack_detection_sim.input['crack_intensity'] = mean_intensity

# Compute the output
crack_detection_sim.compute()

# Determine the detected intensity level
detected_intensity_level = crack_detection_sim.output['detected_intensity']
print("Detected intensity level:", detected_intensity_level)

# Determine the detected intensity level label
if detected_intensity_level < 50:
    print("Crack detected: Low intensity")
elif detected_intensity_level < 150:
    print("Crack detected: Medium intensity")
else:
    print("Crack detected: High intensity")

# Visualize membership functions
crack_intensity.view()
detected_intensity.view()

# Show plots
plt.show()
```


Detected intensity level: 50.00000000000005
Crack detected: Medium intensity


1.0
0.8
0.6
0.4
0.2
0.0
Membership
low
medium
high
0
50
100
150
200
250
crack_intensity


1.0
0.8
0.6
0.4
0.2
0.0
Membership
low
medium
high
0
50
100
150
200
250
detected_intensity

```python
import importlib.util
import sys
import os

# Check if skfuzzy is installed
if importlib.util.find_spec("skfuzzy") is None:
    print("Installing skfuzzy...")
    !pip install scikit-fuzzy

# Now import the necessary modules
import cv2
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl
import matplotlib.pyplot as plt

# Path to the directory containing images
images_dir = '/content/drive/MyDrive/Surface crack/Positive'  # Update with the actual

# Load images
images = []
for filename in os.listdir(images_dir):
    if filename.endswith('.jpg') or filename.endswith('.png'):
        image_path = os.path.join(images_dir, filename)
        image = cv2.imread(image_path)
        if image is not None:
            images.append(image)
        else:
            print(f"Failed to load image: {image_path}")

# Fuzzy logic for crack detection
# Define input variable (crack intensity)
crack_intensity = ctrl.Antecedent(np.arange(0, 256, 1), 'crack_intensity')
crack_intensity['low'] = fuzz.trimf(crack_intensity.universe, [0, 50, 100])
crack_intensity['medium'] = fuzz.trimf(crack_intensity.universe, [50, 100, 200])
crack_intensity['high'] = fuzz.trimf(crack_intensity.universe, [100, 200, 255])

# Define output variable (detected intensity)
detected_intensity = ctrl.Consequent(np.arange(0, 256, 1), 'detected_intensity')
detected_intensity['low'] = fuzz.trimf(detected_intensity.universe, [0, 50, 100])
detected_intensity['medium'] = fuzz.trimf(detected_intensity.universe, [50, 100, 200])
detected_intensity['high'] = fuzz.trimf(detected_intensity.universe, [100, 200, 255])

# Define rules
rule1 = ctrl.Rule(crack_intensity['low'], detected_intensity['low'])
rule2 = ctrl.Rule(crack_intensity['medium'], detected_intensity['medium'])
rule3 = ctrl.Rule(crack_intensity['high'], detected_intensity['high'])

# Create fuzzy control system
crack_detection_ctrl = ctrl.ControlSystem([rule1, rule2, rule3])
```

```python
# Create control system simulation
crack_detection_sim = ctrl.ControlSystemSimulation(crack_detection_ctrl)

# Process only the first 10 images and apply fuzzy logic
for idx, image in enumerate(images[:10]):  # Limiting to the first 10 images
    # Preprocess the image (if needed)
    processed_image = cv2.resize(image, (224, 224))
    processed_image = processed_image.astype(np.float32) / 255.0  # Normalize to [0, 1]

    # Compute mean intensity
    mean_intensity = np.mean(processed_image)

    # Pass the input (mean intensity) to the control system simulation
    crack_detection_sim.input['crack_intensity'] = mean_intensity

    # Compute the output
    crack_detection_sim.compute()

    # Determine the detected intensity level
    detected_intensity_level = crack_detection_sim.output['detected_intensity']

    # Determine the detected intensity level label
    if detected_intensity_level < 50:
        intensity_label = "Low"
    elif detected_intensity_level < 150:
        intensity_label = "Medium"
    else:
        intensity_label = "High"

    print(f"Image {idx+1}: Detected intensity level: {intensity_label}")

    # Visualize membership functions
    crack_intensity.view()
    detected_intensity.view()

    # Show plots
    plt.show()
```

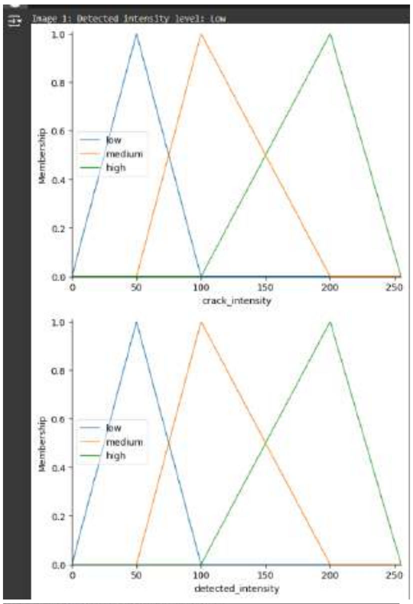


Image 1: Detected intensity level: Low
Membership
low
medium
high
crack_intensity
detected_intensity

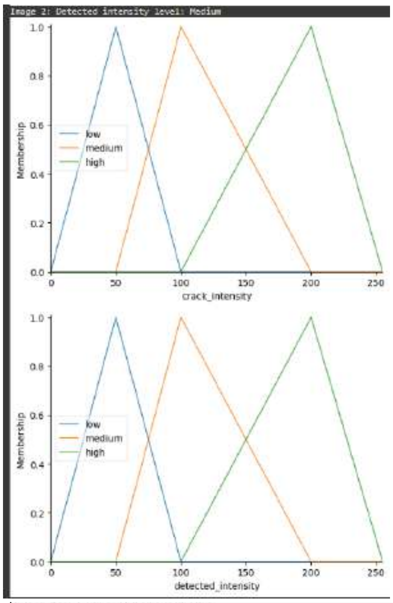


Image 2: Detected intensity level: Medium
Membership
low
medium
high
crack_intensity
detected_intensity

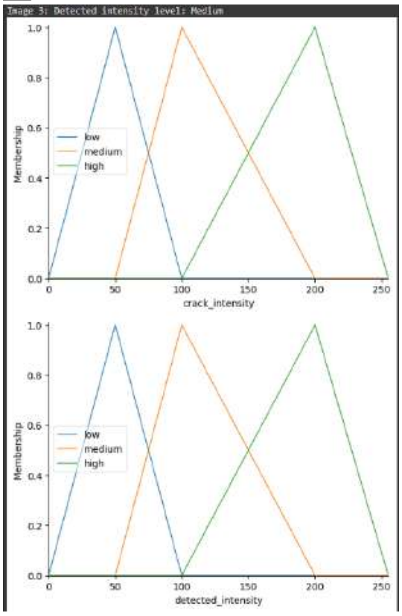


Image 3: Detected intensity level: Medium
Membership
low
medium
high
crack_intensity
detected_intensity

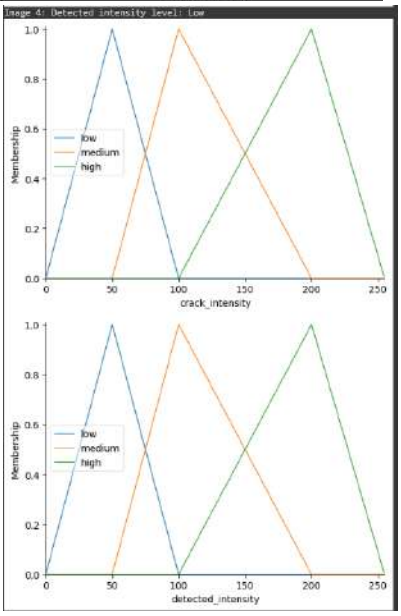


Image 4: Detected intensity level: Low
Membership
low
medium
high
crack_intensity
detected_intensity

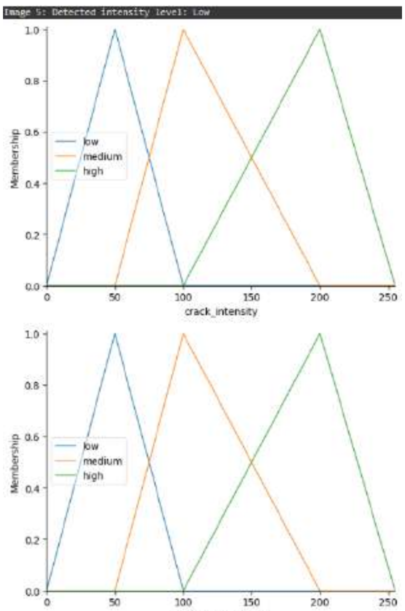


Image 5: Detected intensity level: Low
Membership
low
medium
high
crack_intensity
detected_intensity

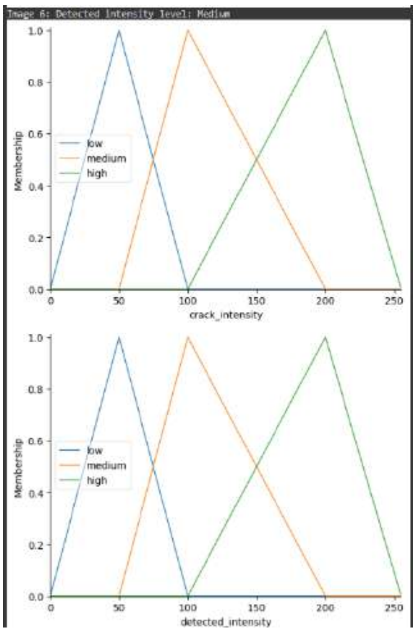


Image 6: Detected intensity level: Medium
Membership
low
medium
high
crack_intensity
detected_intensity

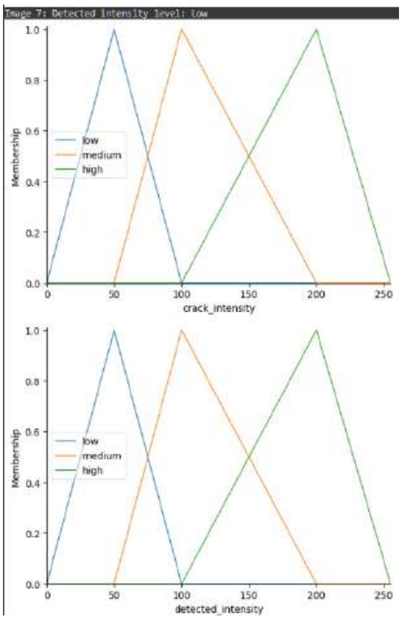


Image 7: Detected intensity level: Low
Membership
low
medium
high
crack_intensity
detected_intensity

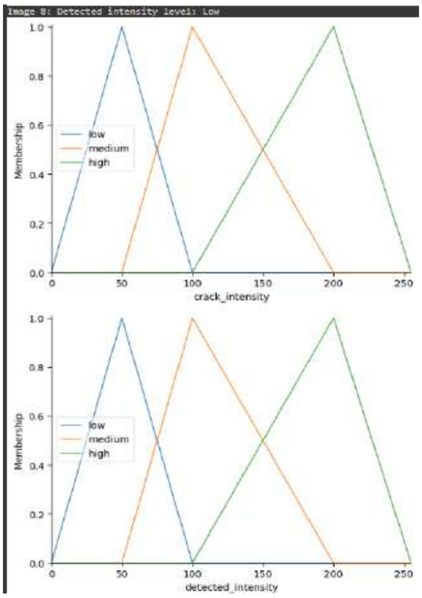


Image 8: Detected intensity level: Low
Membership
low
medium
high
crack_intensity
detected_intensity

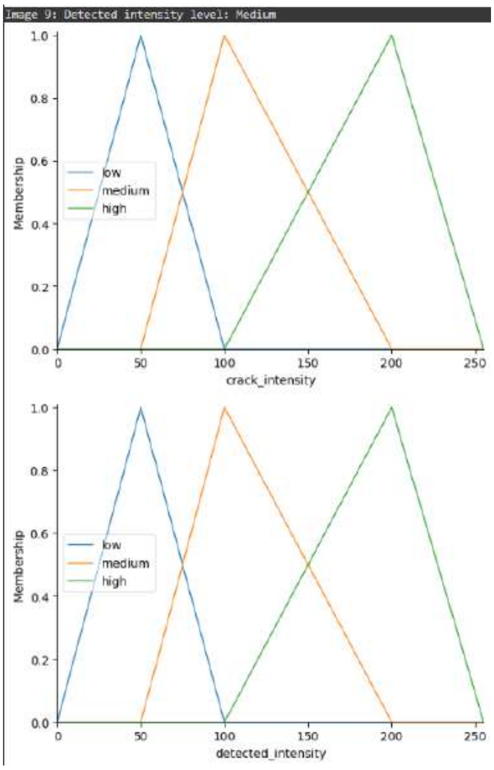


Image 9: Detected intensity level: Medium
Membership
low
medium
high
crack_intensity
detected_intensity

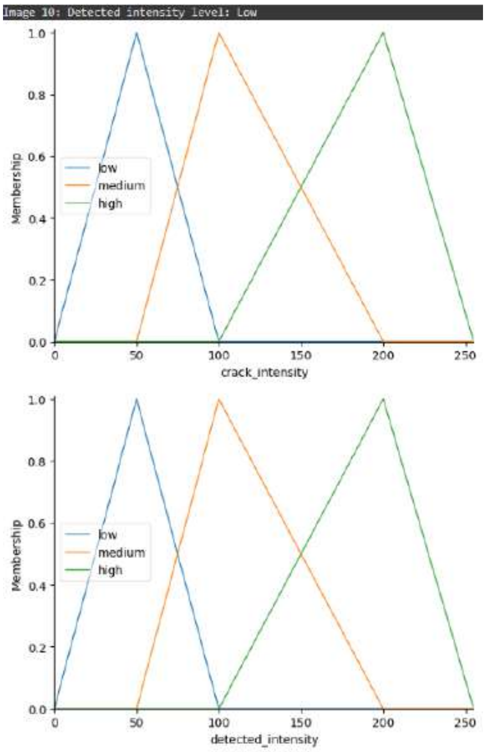


Image 10: Detected intensity level: Low
Membership
low
medium
high
crack_intensity
detected_intensity

```python
# Load and preprocess images
images_dir = '/content/drive/MyDrive/Surface crack/Positive'  # Update with the actual path to your images directory
images = []
for filename in os.listdir(images_dir):
    if filename.endswith('.jpg') or filename.endswith('.png'):
        image_path = os.path.join(images_dir, filename)
        image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)  # Convert to grayscale
        image = cv2.resize(image, (224, 224))  # Resize to 224x224
        image = image.astype(np.float32) / 255.0  # Normalize to [0, 1]
        images.append(image)

# Convert images to numpy array
images = np.array(images)

# Define CNN model
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 1)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])

# Compile the model
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

# Define labels (assuming images with cracks are labeled as 1 and without cracks are labeled as 0)
labels = np.zeros(len(images))  # Initialize labels array with zeros
# You need to manually label your images based on whether they contain cracks or not.
# Then update the labels array accordingly.

# Train the model
history = model.fit(images, labels, epochs=10, batch_size=32, validation_split=0.2)

# Convert training history to dataframe
history_df = pd.DataFrame(history.history)

# Display epochs and corresponding accuracies
print(history_df[['accuracy', 'val_accuracy']])
```

```
Epoch 1/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 32s 3s/step - accuracy: 1.0000 - loss: 0.1804 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 2/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 30s 3s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 3/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 31s 3s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 4/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 31s 3s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 5/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 41s 3s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 6/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 41s 3s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 7/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 31s 3s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 8/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 41s 3s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 9/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 41s 3s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 10/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 41s 3s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
   accuracy  val_accuracy
0       1.0           1.0
1       1.0           1.0
2       1.0           1.0
3       1.0           1.0
4       1.0           1.0
5       1.0           1.0
6       1.0           1.0
7       1.0           1.0
8       1.0           1.0
9       1.0           1.0
```

Checking parameters to change accuracy:

```python
import os
import cv2
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, models
import pandas as pd

# Load and preprocess images
images_dir = '/content/drive/MyDrive/Surface crack/Positive'  # Update with the actual path to your
images = []
for filename in os.listdir(images_dir):
    if filename.endswith('.jpg') or filename.endswith('.png'):
        image_path = os.path.join(images_dir, filename)
        image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)  # Convert to grayscale
        image = cv2.resize(image, (224, 224))  # Resize to 224x224
        image = image.astype(np.float32) / 255.0  # Normalize to [0, 1]
        images.append(image)

# Convert images to numpy array
images = np.array(images)

# Define CNN model
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 1)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])

# Compile the model
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

# Define labels (assuming images with cracks are labeled as 1 and without cracks are labeled as 0)
labels = np.zeros(len(images))  # Initialize labels array with zeros
# You need to manually label your images based on whether they contain cracks or not.
# Then update the labels array accordingly.

# Calculate percentage of training samples
total_samples = len(images)
validation_split = 0.2
percentage_training_samples = (1 - validation_split) * 100
```

```python
print(f"Percentage of training samples: {percentage_training_samples}%")

# Train the model
history = model.fit(images, labels, epochs=10, batch_size=32, validation_split=validation_split)

# Convert training history to dataframe
history_df = pd.DataFrame(history.history)

# Display epochs and corresponding accuracies
print(history_df[['accuracy', 'val_accuracy']])
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Percentage of training samples: 80.0%
Epoch 1/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 32s 3s/step - accuracy: 1.0000 - loss: 0.1865 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 2/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 41s 3s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 3/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 41s 3s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 4/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 30s 3s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 5/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 31s 3s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 6/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 41s 3s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 7/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 41s 3s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 8/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 41s 3s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 9/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 41s 3s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 10/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 41s 3s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
   accuracy  val_accuracy
0       1.0           1.0
1       1.0           1.0
2       1.0           1.0
3       1.0           1.0
4       1.0           1.0
5       1.0           1.0
6       1.0           1.0
7       1.0           1.0
8       1.0           1.0
9       1.0           1.0
```

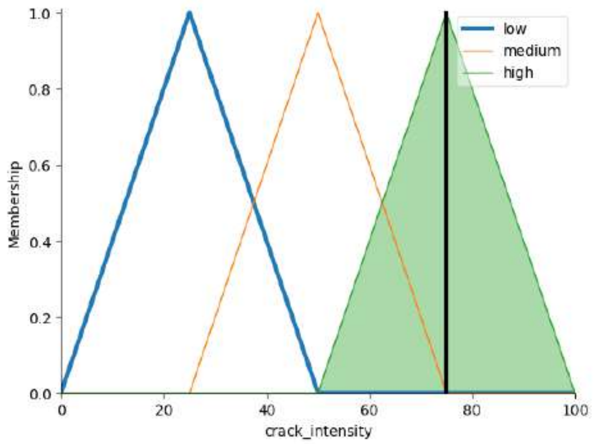
Using Membership Function and Fuzzy Logic Detecting the Crack:

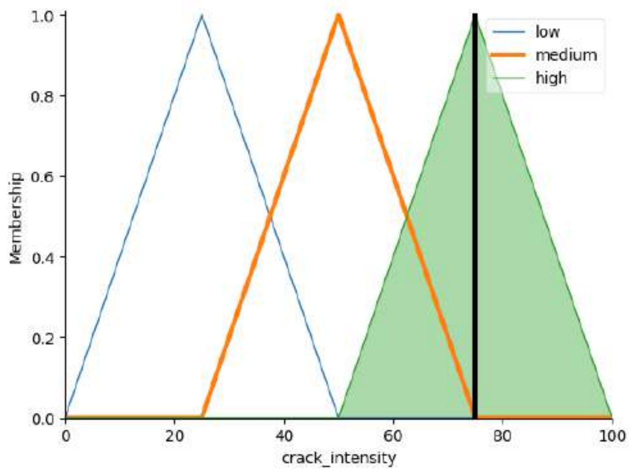
```
[ ] import cv2
    import numpy as np
    import skfuzzy as fuzz
    from skfuzzy import control as ctrl
    import os

    # Load and preprocess images
    images_dir = '/content/drive/MyDrive/Surface crack/Positive'
    images = []

    for filename in os.listdir(images_dir):
        if filename.endswith(".jpg") or filename.endswith(".png"):
            image_path = os.path.join(images_dir, filename)
            image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
            images.append(image)

    # Convert images to numpy array
    images = np.array(images)

    # Compute intensity feature (mean intensity)
    intensities = [np.mean(image) for image in images]

    # Define fuzzy variables and membership functions
    intensity = ctrl.Antecedent(np.arange(0, 256, 1), 'intensity')
    crack_intensity = ctrl.Consequent(np.arange(0, 101, 1), 'crack_intensity')

    # Define membership functions for intensity
    intensity['low'] = fuzz.trimf(intensity.universe, [0, 50, 100])
    intensity['medium'] = fuzz.trimf(intensity.universe, [50, 100, 150])
    intensity['high'] = fuzz.trimf(intensity.universe, [100, 150, 255])

    # Define membership functions for crack intensity
    crack_intensity['low'] = fuzz.trimf(crack_intensity.universe, [0, 25, 50])
    crack_intensity['medium'] = fuzz.trimf(crack_intensity.universe, [25, 50, 75])
    crack_intensity['high'] = fuzz.trimf(crack_intensity.universe, [50, 75, 100])

    # Define fuzzy rules
    rule1 = ctrl.Rule(intensity['low'], crack_intensity['low'])
    rule2 = ctrl.Rule(intensity['medium'], crack_intensity['medium'])
    rule3 = ctrl.Rule(intensity['high'], crack_intensity['high'])

    # Create a fuzzy control system
    crack_detection_ctrl = ctrl.ControlSystem([rule1, rule2, rule3])
    crack_intensity_ctrl = ctrl.ControlSystemSimulation(crack_detection_ctrl)

    # Compute the membership grades for each intensity level
    for intensity_value in intensities:
        crack_intensity_ctrl.input['intensity'] = intensity_value
        crack_intensity_ctrl.compute()
        print("Crack Intensity:", crack_intensity_ctrl.output['crack_intensity'])

    # View the membership graph (optional)
    crack_intensity['low'].view(sim=crack_intensity_ctrl)
    crack_intensity['medium'].view(sim=crack_intensity_ctrl)
    crack_intensity['high'].view(sim=crack_intensity_ctrl)
```

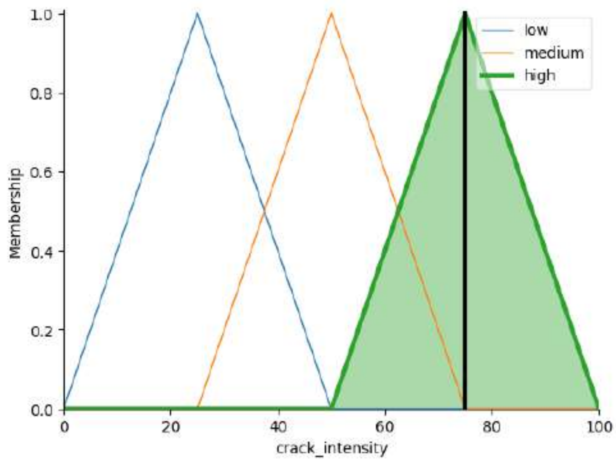
```
Crack Intensity: 74.99999999999999
Crack Intensity: 75.0
Crack Intensity: 61.13206214108184
Crack Intensity: 66.86923330622075
Crack Intensity: 74.2871971797587
Crack Intensity: 74.99999999999999
Crack Intensity: 55.95407417011593
Crack Intensity: 55.72958989440247
Crack Intensity: 75.0
Crack Intensity: 50.421193677277266
Crack Intensity: 75.0
Crack Intensity: 56.586332128274854
Crack Intensity: 74.99999999999999
Crack Intensity: 60.53949530841085
Crack Intensity: 67.18172386977098
Crack Intensity: 75.0
Crack Intensity: 75.0
Crack Intensity: 59.23214260040896
Crack Intensity: 63.571951419199365
Crack Intensity: 52.94327758329112
Crack Intensity: 61.68561260378529
Crack Intensity: 53.222564117927654
Crack Intensity: 67.836956010319
Crack Intensity: 70.37561759199986
Crack Intensity: 75.0
Crack Intensity: 74.99999999999997
Crack Intensity: 64.81756150055158
Crack Intensity: 74.99999999999999
Crack Intensity: 56.38703777666577
Crack Intensity: 75.00000000000001
Crack Intensity: 62.02065467175664
Crack Intensity: 59.12827876041739
Crack Intensity: 71.61784913312384
Crack Intensity: 63.341376498585404
Crack Intensity: 58.78724423289606
Crack Intensity: 74.99999999999997
Crack Intensity: 74.99999999999999
Crack Intensity: 74.32007540953498
Crack Intensity: 52.007600601532616
Crack Intensity: 52.302733290218335
Crack Intensity: 74.99999999999997
Crack Intensity: 65.60908984259748
Crack Intensity: 59.073431907108215
Crack Intensity: 53.461686579614366
Crack Intensity: 54.63862964660345
```

```
Crack Intensity: 75.00000000000001
Crack Intensity: 64.13817373590528
Crack Intensity: 64.80706545555786
Crack Intensity: 55.31980501979002
Crack Intensity: 61.4005312123574
Crack Intensity: 64.63095431001972
Crack Intensity: 74.99999999999999
Crack Intensity: 55.740211266608206
Crack Intensity: 56.489553614294366
Crack Intensity: 67.19958616196384
Crack Intensity: 58.89001404090769
Crack Intensity: 74.55583090356528
Crack Intensity: 64.29079124151694
Crack Intensity: 74.99999999999997
Crack Intensity: 70.69227819565758
Crack Intensity: 75.0
Crack Intensity: 59.105203781274184
Crack Intensity: 60.75612538109869
Crack Intensity: 51.45965980597515
Crack Intensity: 55.081003575847575
Crack Intensity: 63.782470828583136
Crack Intensity: 63.165819559029146
Crack Intensity: 59.59012884115429
Crack Intensity: 75.00000000000001
Crack Intensity: 58.244808698378954
Crack Intensity: 74.99999999999999
Crack Intensity: 59.05141618625751
Crack Intensity: 71.12896409296697
Crack Intensity: 74.99999999999999
Crack Intensity: 58.70329541819477
Crack Intensity: 75.0
Crack Intensity: 75.00000000000001
Crack Intensity: 60.38658648586082
Crack Intensity: 64.51589768220131
Crack Intensity: 62.627290669499466
Crack Intensity: 66.88081578289531
Crack Intensity: 58.430033318451926
Crack Intensity: 58.00420246014321
Crack Intensity: 56.23603195792204
Crack Intensity: 61.63347232613435
Crack Intensity: 61.1620898195872
Crack Intensity: 60.54222648050034
Crack Intensity: 58.815861574274535
Crack Intensity: 74.99999999999994
Crack Intensity: 51.741660242599245
```

```
Crack Intensity: 75.00000000000001
Crack Intensity: 61.33270826599607
Crack Intensity: 67.57532061846679
Crack Intensity: 65.74842576060958
Crack Intensity: 75.00000000000003
Crack Intensity: 57.482207757942334
Crack Intensity: 65.91223477312604
Crack Intensity: 65.19607192580803
Crack Intensity: 60.445803579532786
Crack Intensity: 53.64058930106718
Crack Intensity: 56.83947647497458
Crack Intensity: 51.98180593513456
Crack Intensity: 53.14983699785767
Crack Intensity: 56.51025977103094
Crack Intensity: 74.31373341899392
Crack Intensity: 58.6525292178201
Crack Intensity: 62.67504640333874
Crack Intensity: 50.77353609281963
Crack Intensity: 59.862756907630036
Crack Intensity: 61.67791357932274
Crack Intensity: 59.67687131535544
Crack Intensity: 57.950525956937284
Crack Intensity: 57.49786128316666
Crack Intensity: 60.819960207557244
Crack Intensity: 62.544776002159985
Crack Intensity: 56.44499197141418
Crack Intensity: 59.55087541219593
Crack Intensity: 65.35206043200904
Crack Intensity: 56.52936593053577
Crack Intensity: 61.07146981116533
Crack Intensity: 57.75880890965594
Crack Intensity: 70.22363177306796
Crack Intensity: 52.582936459579464
Crack Intensity: 74.99999999999999
Crack Intensity: 75.00000000000003
Crack Intensity: 75.0
Crack Intensity: 58.517470461927765
Crack Intensity: 64.99568236232442
Crack Intensity: 62.64643260108119
Crack Intensity: 58.867596421944185
Crack Intensity: 56.94130835793179
Crack Intensity: 62.73921679015679
Crack Intensity: 74.99999999999997
Crack Intensity: 75.0
Crack Intensity: 57.5396261943952
```








```
[ ] import cv2
    import numpy as np
    import skfuzzy as fuzz
    from skfuzzy import control as ctrl
    import os

    # Load and preprocess images
    images_dir = '/content/drive/MyDrive/Surface crack/Negative'
    images = []

    for filename in os.listdir(images_dir):
        if filename.endswith(".jpg") or filename.endswith(".png"):
            image_path = os.path.join(images_dir, filename)
            image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
            images.append(image)

    # Convert images to numpy array
    images = np.array(images)

    # Compute intensity feature (mean intensity)
    intensities = [np.mean(image) for image in images]

    # Define fuzzy variables and membership functions
    intensity = ctrl.Antecedent(np.arange(0, 256, 1), 'intensity')
    crack_intensity = ctrl.Consequent(np.arange(0, 101, 1), 'crack_intensity')

    # Define membership functions for intensity
    intensity['low'] = fuzz.trimf(intensity.universe, [0, 50, 100])
    intensity['medium'] = fuzz.trimf(intensity.universe, [50, 100, 150])
    intensity['high'] = fuzz.trimf(intensity.universe, [100, 150, 255])

    # Define membership functions for crack intensity
    crack_intensity['low'] = fuzz.trimf(crack_intensity.universe, [0, 25, 50])
    crack_intensity['medium'] = fuzz.trimf(crack_intensity.universe, [25, 50, 75])
    crack_intensity['high'] = fuzz.trimf(crack_intensity.universe, [50, 75, 100])

    # Define fuzzy rules
    rule1 = ctrl.Rule(intensity['low'], crack_intensity['low'])
    rule2 = ctrl.Rule(intensity['medium'], crack_intensity['medium'])
    rule3 = ctrl.Rule(intensity['high'], crack_intensity['high'])

    # Create a fuzzy control system
    crack_detection_ctrl = ctrl.ControlSystem([rule1, rule2, rule3])
    crack_intensity_ctrl = ctrl.ControlSystemSimulation(crack_detection_ctrl)

    # Compute the membership grades for each intensity level
    for intensity_value in intensities:
        crack_intensity_ctrl.input['intensity'] = intensity_value
        crack_intensity_ctrl.compute()
        print("Crack Intensity:", crack_intensity_ctrl.output['crack_intensity'])

    # View the membership graph (optional)
    crack_intensity['low'].view(sim=crack_intensity_ctrl)
    crack_intensity['medium'].view(sim=crack_intensity_ctrl)
    crack_intensity['high'].view(sim=crack_intensity_ctrl)
```

```
Crack Intensity: 74.99999999999996
Crack Intensity: 75.00000000000004
Crack Intensity: 75.00000000000004
Crack Intensity: 75.00000000000001
Crack Intensity: 74.99999999999999
Crack Intensity: 55.855928071111094
Crack Intensity: 75.00000000000001
Crack Intensity: 74.99999999999999
Crack Intensity: 75.0
Crack Intensity: 75.00000000000001
Crack Intensity: 75.00000000000001
Crack Intensity: 75.00000000000003
Crack Intensity: 74.99999999999999
Crack Intensity: 75.0
Crack Intensity: 75.0
Crack Intensity: 74.99999999999999
Crack Intensity: 75.0
Crack Intensity: 75.00000000000003
Crack Intensity: 74.99999999999996
Crack Intensity: 75.00000000000003
Crack Intensity: 75.00000000000003
Crack Intensity: 75.00000000000003
Crack Intensity: 75.00000000000006
Crack Intensity: 75.0
Crack Intensity: 75.00000000000001
Crack Intensity: 75.00000000000003
Crack Intensity: 74.99999999999999
Crack Intensity: 74.99999999999999
Crack Intensity: 74.99999999999996
Crack Intensity: 75.0
Crack Intensity: 74.99999999999997
Crack Intensity: 74.99999999999999
Crack Intensity: 74.99999999999999
Crack Intensity: 75.00000000000001
Crack Intensity: 75.0
Crack Intensity: 74.99999999999994
Crack Intensity: 75.0
Crack Intensity: 74.99999999999996
Crack Intensity: 75.00000000000001
Crack Intensity: 75.0
Crack Intensity: 75.00000000000003
Crack Intensity: 75.00000000000001
Crack Intensity: 74.99999999999997
Crack Intensity: 75.00000000000001
```

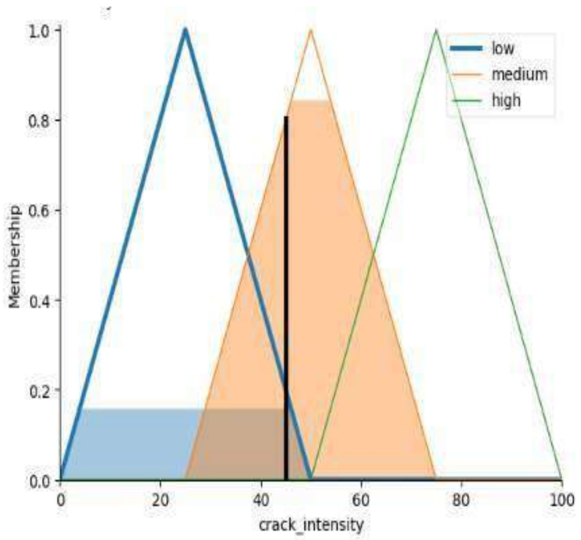
```
Crack Intensity: 75.00000000000003
Crack Intensity: 74.99999999999997
Crack Intensity: 74.99999999999996
Crack Intensity: 74.99999999999994
Crack Intensity: 75.0
Crack Intensity: 67.80569830300075
Crack Intensity: 74.99999999999997
Crack Intensity: 70.45657197863476
Crack Intensity: 75.0
Crack Intensity: 74.99999999999999
Crack Intensity: 75.00000000000004
Crack Intensity: 75.00000000000001
Crack Intensity: 75.0
Crack Intensity: 75.0
Crack Intensity: 75.0
Crack Intensity: 75.0
Crack Intensity: 75.00000000000004
Crack Intensity: 75.0
Crack Intensity: 75.0
Crack Intensity: 75.00000000000001
Crack Intensity: 74.99999999999994
Crack Intensity: 75.00000000000003
Crack Intensity: 75.0
Crack Intensity: 75.00000000000003
Crack Intensity: 75.00000000000001
Crack Intensity: 74.99999999999999
Crack Intensity: 75.00000000000001
Crack Intensity: 75.00000000000003
Crack Intensity: 75.00000000000001
Crack Intensity: 74.99999999999997
Crack Intensity: 75.0
Crack Intensity: 74.99999999999996
Crack Intensity: 74.99999999999997
Crack Intensity: 64.50970774043148
Crack Intensity: 74.99999999999997
Crack Intensity: 74.99999999999997
Crack Intensity: 75.0
Crack Intensity: 67.3459454144083
Crack Intensity: 75.00000000000001
Crack Intensity: 75.0
Crack Intensity: 70.0768026287707
Crack Intensity: 74.99999999999999
Crack Intensity: 75.00000000000001
Crack Intensity: 75.0
Crack Intensity: 61.97297914270365
```

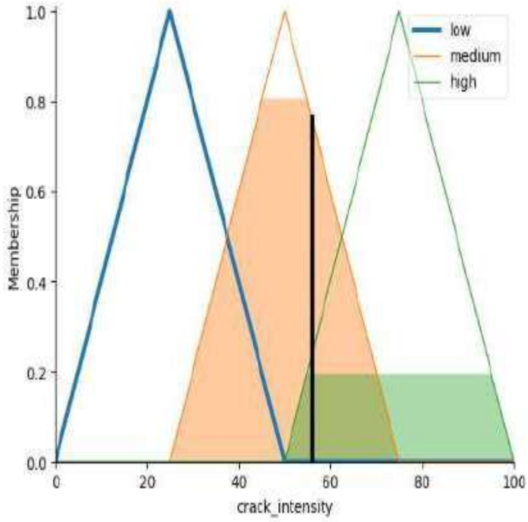
```
Crack Intensity: 74.99999999999997
Crack Intensity: 75.0
Crack Intensity: 74.99999999999997
Crack Intensity: 75.00000000000001
Crack Intensity: 66.83097101801431
Crack Intensity: 74.99999999999999
Crack Intensity: 75.00000000000001
Crack Intensity: 75.00000000000004
Crack Intensity: 74.99999999999999
Crack Intensity: 58.30550442808280
Crack Intensity: 75.00000000000001
Crack Intensity: 74.99999999999997
Crack Intensity: 74.99999999999996
Crack Intensity: 75.0
Crack Intensity: 64.14831038867108
Crack Intensity: 75.0
Crack Intensity: 75.0
Crack Intensity: 74.99999999999999
Crack Intensity: 75.00000000000003
Crack Intensity: 75.00000000000001
Crack Intensity: 75.0
Crack Intensity: 75.0
Crack Intensity: 74.99999999999999
Crack Intensity: 75.00000000000001
Crack Intensity: 74.99999999999999
Crack Intensity: 74.99999999999996
Crack Intensity: 75.0
Crack Intensity: 75.00000000000001
Crack Intensity: 74.99999999999996
Crack Intensity: 75.0
Crack Intensity: 74.99999999999997
Crack Intensity: 74.99999999999996
Crack Intensity: 73.10493773187075
Crack Intensity: 74.99999999999996
Crack Intensity: 75.00000000000001
Crack Intensity: 74.99999999999999
Crack Intensity: 75.00000000000001
Crack Intensity: 74.99999999999996
Crack Intensity: 74.99999999999997
Crack Intensity: 74.99999999999999
Crack Intensity: 74.99999999999999
Crack Intensity: 67.60821074483904
Crack Intensity: 75.0
Crack Intensity: 74.99999999999997
Crack Intensity: 75.00000000000001
```

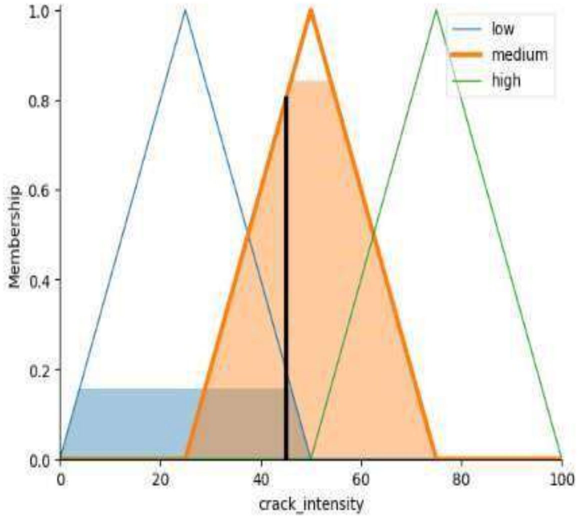
```
Crack Intensity: 75.00000000000001
Crack Intensity: 74.99999999999996
Crack Intensity: 74.99999999999997
Crack Intensity: 65.07580253505526
Crack Intensity: 75.00000000000003
Crack Intensity: 74.99999999999997
Crack Intensity: 75.0
Crack Intensity: 75.0
Crack Intensity: 74.99999999999996
Crack Intensity: 75.00000000000001
Crack Intensity: 75.00000000000003
Crack Intensity: 75.00000000000003
Crack Intensity: 74.99999999999996
Crack Intensity: 74.99999999999996
Crack Intensity: 75.00000000000001
Crack Intensity: 75.0
Crack Intensity: 74.99999999999997
Crack Intensity: 74.99999999999997
Crack Intensity: 74.99999999999997
Crack Intensity: 75.00000000000001
Crack Intensity: 74.99999999999997
Crack Intensity: 75.00000000000001
Crack Intensity: 75.0
Crack Intensity: 74.99999999999996
Crack Intensity: 74.99999999999996
Crack Intensity: 75.00000000000003
Crack Intensity: 75.0
Crack Intensity: 74.99999999999999
Crack Intensity: 74.99999999999994
Crack Intensity: 75.0
Crack Intensity: 75.0
Crack Intensity: 74.99999999999997
Crack Intensity: 74.99999999999999
Crack Intensity: 75.0
Crack Intensity: 75.0
Crack Intensity: 74.99999999999996
Crack Intensity: 75.0
Crack Intensity: 75.0
Crack Intensity: 75.00000000000001
Crack Intensity: 74.99999999999999
Crack Intensity: 74.99999999999996
Crack Intensity: 75.0
Crack Intensity: 74.99999999999997
Crack Intensity: 75.0
Crack Intensity: 75.00000000000001
```

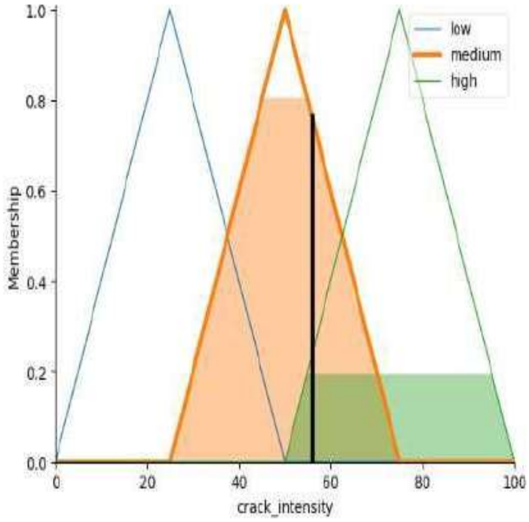
```
Crack Intensity: 75.0
Crack Intensity: 74.99999999999997
Crack Intensity: 74.99999999999997
Crack Intensity: 74.99999999999996
Crack Intensity: 75.0
Crack Intensity: 75.00000000000001
Crack Intensity: 74.99999999999999
Crack Intensity: 74.99999999999996
Crack Intensity: 75.00000000000006
Crack Intensity: 75.0
Crack Intensity: 75.0
Crack Intensity: 74.99999999999997
Crack Intensity: 75.00000000000001
Crack Intensity: 75.0
Crack Intensity: 75.00000000000001
Crack Intensity: 62.932600968092615
Crack Intensity: 73.9851277122425
Crack Intensity: 75.0
Crack Intensity: 75.0
Crack Intensity: 75.00000000000004
Crack Intensity: 75.0
Crack Intensity: 75.00000000000001
Crack Intensity: 75.0
Crack Intensity: 74.99999999999996
Crack Intensity: 75.0
Crack Intensity: 74.99999999999997
Crack Intensity: 75.00000000000001
Crack Intensity: 74.99999999999999
Crack Intensity: 75.0
Crack Intensity: 74.99999999999997
Crack Intensity: 75.00000000000001
Crack Intensity: 75.00000000000003
Crack Intensity: 74.99999999999996
Crack Intensity: 74.99999999999993
Crack Intensity: 74.99999999999996
Crack Intensity: 74.99999999999997
Crack Intensity: 74.99999999999999
Crack Intensity: 74.99999999999999
Crack Intensity: 75.0
Crack Intensity: 75.00000000000004
Crack Intensity: 75.0
Crack Intensity: 75.00000000000003
Crack Intensity: 75.0
Crack Intensity: 74.99999999999996
```

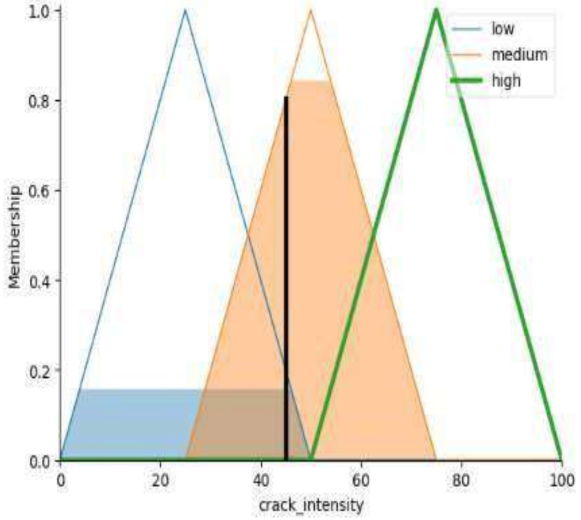
```
Crack Intensity: 74.99999999999997
Crack Intensity: 74.99999999999996
Crack Intensity: 74.99999999999999
Crack Intensity: 75.00000000000001
Crack Intensity: 75.0
Crack Intensity: 74.99999999999996
Crack Intensity: 74.99999999999999
Crack Intensity: 75.0
Crack Intensity: 74.99999999999999
Crack Intensity: 74.99999999999999
Crack Intensity: 74.99999999999996
Crack Intensity: 75.00000000000001
Crack Intensity: 62.09389481105525
Crack Intensity: 75.00000000000003
Crack Intensity: 75.00000000000001
Crack Intensity: 75.00000000000003
Crack Intensity: 75.00000000000006
Crack Intensity: 75.00000000000001
Crack Intensity: 74.99999999999994
Crack Intensity: 74.99999999999997
Crack Intensity: 75.0
Crack Intensity: 75.00000000000003
Crack Intensity: 74.99999999999999
Crack Intensity: 74.99999999999999
Crack Intensity: 74.99999999999994
Crack Intensity: 75.00000000000003
Crack Intensity: 75.00000000000001
Crack Intensity: 74.99999999999997
Crack Intensity: 74.99999999999999
Crack Intensity: 74.99999999999999
Crack Intensity: 74.99999999999997
Crack Intensity: 74.99999999999999
Crack Intensity: 74.99999999999999
Crack Intensity: 74.99999999999997
Crack Intensity: 75.00000000000001
Crack Intensity: 75.00000000000003
Crack Intensity: 75.00000000000003
Crack Intensity: 74.99999999999993
Crack Intensity: 74.99999999999999
Crack Intensity: 63.2778121572539
Crack Intensity: 75.00000000000004
Crack Intensity: 74.99999999999996
Crack Intensity: 75.00000000000003
Crack Intensity: 50.36611141382277
```

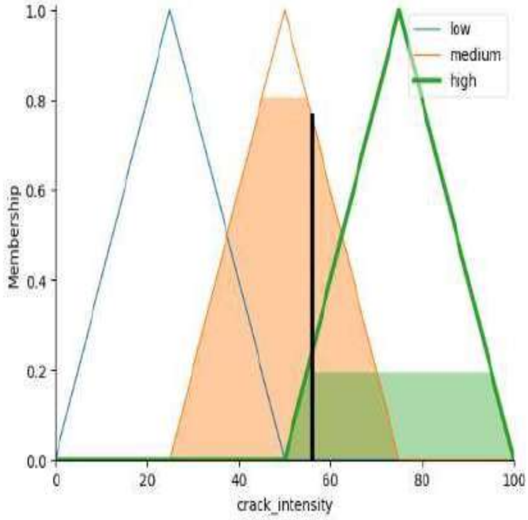

low
medium
high
Membership
crack_intensity
0.0 0.2 0.4 0.6 0.8 1.0
0 20 40 60 80 100

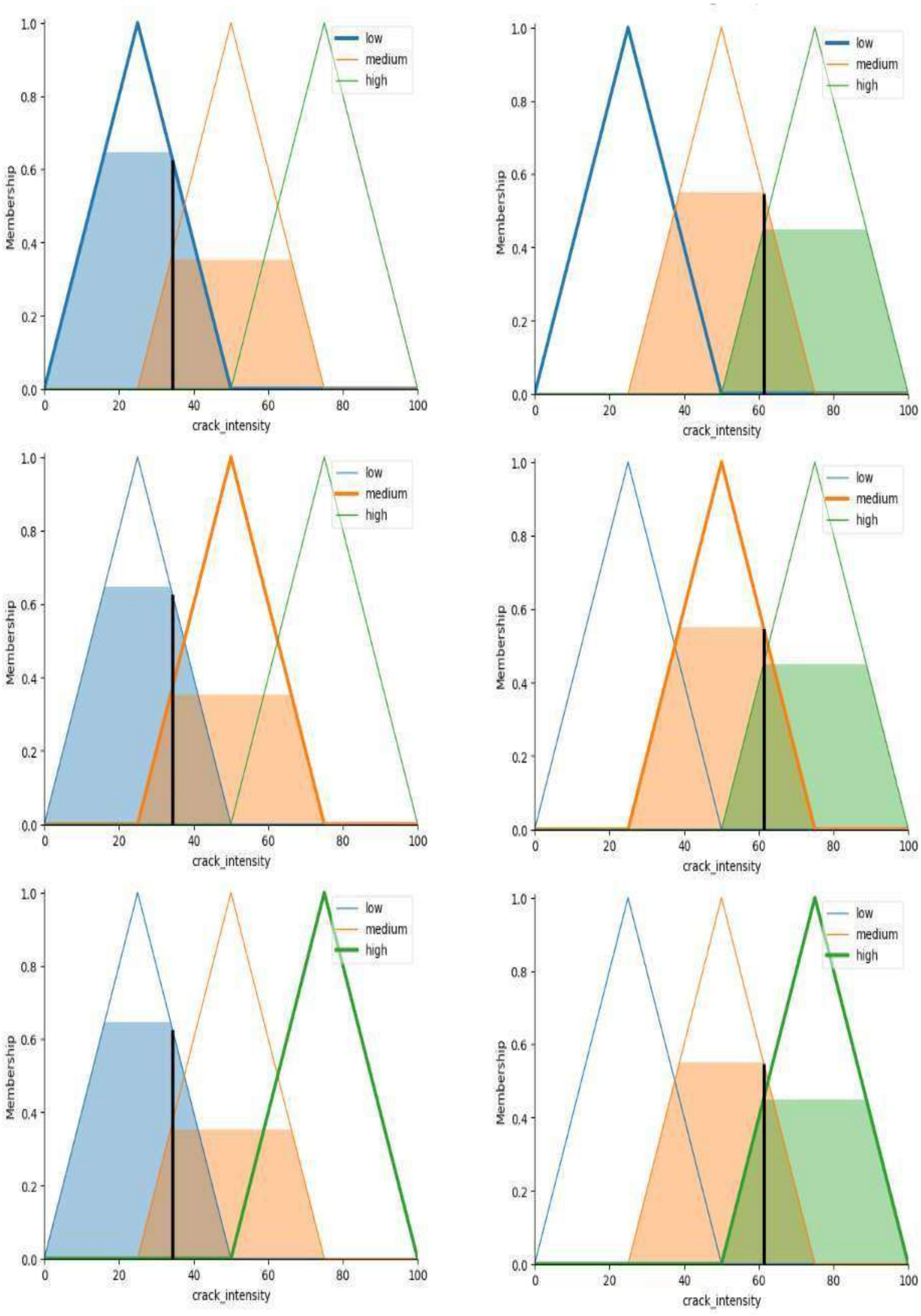

low
medium
high
Membership
crack_intensity
0.0 0.2 0.4 0.6 0.8 1.0
0 20 40 60 80 100


low
medium
high
Membership
crack_intensity
0.0 0.2 0.4 0.6 0.8 1.0
0 20 40 60 80 100


low
medium
high
Membership
crack_intensity
0.0 0.2 0.4 0.6 0.8 1.0
0 20 40 60 80 100


low
medium
high
Membership
crack_intensity
0.0 0.2 0.4 0.6 0.8 1.0
0 20 40 60 80 100


low
medium
high
Membership
crack_intensity
0.0 0.2 0.4 0.6 0.8 1.0
0 20 40 60 80 100



low
medium
high
Membership
crack_intensity

CONFUSION MATRIX:

Getting started:

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px

from pathlib import Path
from sklearn.model_selection import train_test_split

import tensorflow as tf

from sklearn.metrics import confusion_matrix, classification_report
```

```
positive_dir = Path('/content/drive/MyDrive/Surface crack/Negative')
negative_dir = Path('/content/drive/MyDrive/Surface crack/Negative')
```

```
def generate_df(image_dir, label):
    filepaths = pd.Series(list(image_dir.glob(r'*.jpg')), name='Filepath').astype(str)
    labels = pd.Series(label, name='Label', index=filepaths.index)
    df = pd.concat([filepaths, labels], axis=1)
    return df
```

Creating dataframes:

```
positive_df = generate_df(positive_dir, label="POSITIVE")
negative_df = generate_df(negative_dir, label="NEGATIVE")

all_df = pd.concat([positive_df, negative_df], axis=0).sample(frac=1.0, random_state=1).reset_index(drop=True)
all_df
```

	Filepath	Label
0	/content/drive/MyDrive/Surface crack/Negative/...	NEGATIVE
1	/content/drive/MyDrive/Surface crack/Negative/...	POSITIVE
2	/content/drive/MyDrive/Surface crack/Negative/...	NEGATIVE
3	/content/drive/MyDrive/Surface crack/Negative/...	POSITIVE
4	/content/drive/MyDrive/Surface crack/Negative/...	POSITIVE
...	...	...
4583	/content/drive/MyDrive/Surface crack/Negative/...	NEGATIVE
4584	/content/drive/MyDrive/Surface crack/Negative/...	NEGATIVE
4585	/content/drive/MyDrive/Surface crack/Negative/...	POSITIVE
4586	/content/drive/MyDrive/Surface crack/Negative/...	NEGATIVE
4587	/content/drive/MyDrive/Surface crack/Negative/...	POSITIVE

4588 rows × 2 columns

Loading image data:

```python
train_data = train_gen.flow_from_dataframe(
    train_df,
    x_col='Filepath',
    y_col='Label',
    target_size=(120, 120),
    color_mode='rgb',
    class_mode='binary',
    batch_size=32,
    shuffle=True,
    seed=42,
    subset='training'
)

val_data = train_gen.flow_from_dataframe(
    train_df,
    x_col='Filepath',
    y_col='Label',
    target_size=(120, 120),
    color_mode='rgb',
    class_mode='binary',
    batch_size=32,
    shuffle=True,
    seed=42,
    subset='validation'
)

test_data = train_gen.flow_from_dataframe(
    test_df,
    x_col='Filepath',
    y_col='Label',
    target_size=(120, 120),
    color_mode='rgb',
    class_mode='binary',
    batch_size=32,
    shuffle=False,
    seed=42
)
```

```
Found 2569 validated image filenames belonging to 2 classes.
Found 642 validated image filenames belonging to 2 classes.
Found 1377 validated image filenames belonging to 2 classes.
```

Training :

```python
inputs = tf.keras.Input(shape=(120, 120, 3))
x = tf.keras.layers.Conv2D(filters=16, kernel_size=(3, 3), activation='relu')(inputs)
x = tf.keras.layers.MaxPool2D(pool_size=(2, 2))(x)
x = tf.keras.layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu')(x)
x = tf.keras.layers.MaxPool2D(pool_size=(2, 2))(x)
x = tf.keras.layers.GlobalAveragePooling2D()(x)
outputs = tf.keras.layers.Dense(1, activation='sigmoid')(x)

model = tf.keras.Model(inputs=inputs, outputs=outputs)

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

print(model.summary())
```

Model: "functional_4"

Layer (type)	Output Shape	Param #
input_layer_4 (InputLayer)	(None, 120, 120, 3)	0
conv2d_12 (Conv2D)	(None, 118, 118, 16)	448
max_pooling2d_8 (MaxPooling2D)	(None, 59, 59, 16)	0
conv2d_13 (Conv2D)	(None, 57, 57, 32)	4,640
max_pooling2d_9 (MaxPooling2D)	(None, 28, 28, 32)	0
global_average_pooling2d (GlobalAveragePooling2D)	(None, 32)	0
dense_8 (Dense)	(None, 1)	33

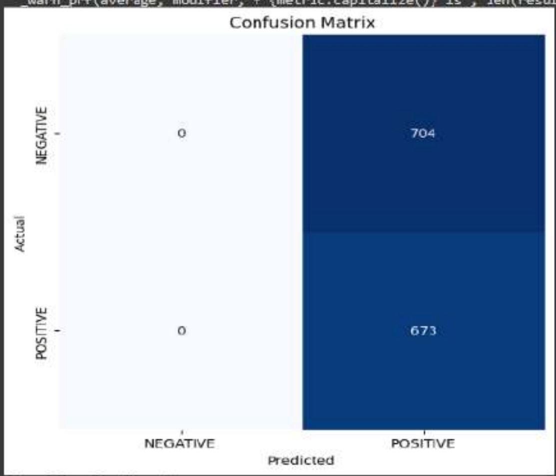
```
Total params: 5,121 (20.00 KB)
Trainable params: 5,121 (20.00 KB)
Non-trainable params: 0 (0.00 B)
None
```

Result of confusion matrix:

```python
def evaluate_model(model, test_data):

    results = model.evaluate(test_data, verbose=0)
    loss = results[0]
    acc = results[1]

    print("    Test Loss: {:.5f}".format(loss))
    print("Test Accuracy: {:.2f}%".format(acc * 100))

    y_pred = np.squeeze((model.predict(test_data) >= 0.5).astype(np.int))
    cm = confusion_matrix(test_data.labels, y_pred)
    clr = classification_report(test_data.labels, y_pred, target_names=["NEGATIVE", "POSITIVE"])

    plt.figure(figsize=(6, 6))
    sns.heatmap(cm, annot=True, fmt='g', vmin=0, cmap='Blues', cbar=False)
    plt.xticks(ticks=np.arange(2) + 0.5, labels=["NEGATIVE", "POSITIVE"])
    plt.yticks(ticks=np.arange(2) + 0.5, labels=["NEGATIVE", "POSITIVE"])
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.title("Confusion Matrix")
    plt.show()

    print("Classification Report:\n---------------------\n", clr)
```

```python
[ ] def evaluate_model(model, test_data):

    results = model.evaluate(test_data, verbose=0)
    loss = results[0]
    acc = results[1]

    print("    Test Loss: {:.5f}".format(loss))
    print("Test Accuracy: {:.2f}%".format(acc * 100))

    y_pred = np.squeeze((model.predict(test_data) >= 0.5).astype(int)) # Changed np.int to int
    cm = confusion_matrix(test_data.labels, y_pred)
    clr = classification_report(test_data.labels, y_pred, target_names=["NEGATIVE", "POSITIVE"])

    plt.figure(figsize=(6, 6))
    sns.heatmap(cm, annot=True, fmt='g', vmin=0, cmap='Blues', cbar=False)
    plt.xticks(ticks=np.arange(2) + 0.5, labels=["NEGATIVE", "POSITIVE"])
    plt.yticks(ticks=np.arange(2) + 0.5, labels=["NEGATIVE", "POSITIVE"])
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.title("Confusion Matrix")
    plt.show()

    print("Classification Report:\n---------------------\n", clr)
```

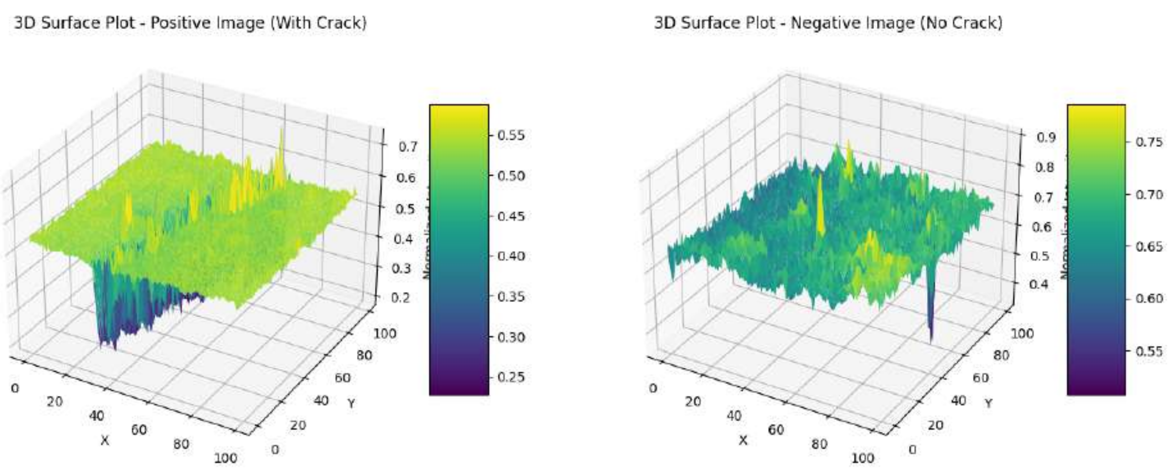
```
evaluate_model(model, test_data)

/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_
  self._warn_if_super_not_called()
    Test Loss: 0.69696
Test Accuracy: 48.87%
44/44 ━━━━━━━━━━━━━━━━━━━━ 8s 168ms/step
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```




```
Classification Report:
----------------------
              precision    recall  f1-score   support

    NEGATIVE       0.00      0.00      0.00       704
    POSITIVE       0.49      1.00      0.66       673

    accuracy                           0.49      1377
   macro avg       0.24      0.50      0.33      1377
weighted avg       0.24      0.49      0.32      1377
```

3D PLOT:

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

def plot_3d_surface(image_path, title, ax):
    # Load the image in grayscale
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

    # Resize the image for consistent plotting size (optional)
    img = cv2.resize(img, (100, 100))  # Resize to 100x100 or desired size

    # Generate X, Y coordinates
    X, Y = np.meshgrid(range(img.shape[1]), range(img.shape[0]))
    Z = img.astype(np.float32) / 255.0  # Normalize pixel values to range [0, 1]

    # Plot the surface with color mapping
    surf = ax.plot_surface(X, Y, Z, cmap='viridis', edgecolor='none')

    # Set labels and title for each subplot
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Normalized Intensity')
    ax.set_title(title)

    return surf

# Paths to the positive and negative images
positive_image_path = '/content/drive/MyDrive/Surface crack/Positive/00001.jpg'
negative_image_path = '/content/drive/MyDrive/Surface crack/Negative/00001.jpg'

# Create a figure with two subplots for side-by-side comparison
fig = plt.figure(figsize=(16, 8))

# Plot positive image
ax1 = fig.add_subplot(121, projection='3d')
surf1 = plot_3d_surface(positive_image_path, "3D Surface Plot - Positive Image (With Crack)", ax1)

# Plot negative image
ax2 = fig.add_subplot(122, projection='3d')
```



Google drive link – video presentation

https://drive.google.com/file/d/18uGg8M75gT1aCuE4edUChV6pF2000UiN/view?usp=drivesdk

Google colab link:

https://colab.research.google.com/drive/1UVaiqvoV1AN9_brjOzfbmVKL2VaoScb9?usp=sharing

REFERENCES :

[1] Wu, Lijun, Xu Lin, Zhicong Chen, Peijie Lin, and Shuying Cheng. "Surface crack detection based on image stitching and transfer learning with pretrained convolutional neural network." Structural Control and Health Monitoring 28, no. 8 (2021): e2766.

[2] Yu, Dawen, Shunping Ji, Xue Li, Zhaode Yuan, and Chaoyong Shen. "Earthquake crack detection from aerial images using a deformable convolutional neural network." IEEE Transactions on Geoscience and Remote Sensing 60 (2022): 1-12.

[3] Ali, Sayyed Bashar, Reshul Wate, Sameer Kujur, Anurag Singh, and Santosh Kumar. "Wall crack detection using transfer learning-based CNN models." In 2020 IEEE 17th India Council International Conference (INDICON), pp. 1-7. IEEE, 2020.

[4] Li, Peigen, Bin Zhou, Chuan Wang, Guizhang Hu, Yong Yan, Rongxin Guo, and Haiting Xia. "CNN-based pavement defects detection using grey and depth images." Automation in Construction 158 (2024): 105192.

[5] Elghaish, Faris, Saeed Talebi, Essam Abdellatef, Sandra T. Matarneh, M. Reza Hosseini, Song Wu, Mohammad Mayouf, Aso Hajirasouli, and The-Quan Nguyen. "Developing a new deep learning CNN model to detect and classify highway cracks." Journal of Engineering, Design and Technology 20, no. 4 (2022): 993-1014.