Breaking is Bad

Implementation of a Basic Crack Detector

The project aims to employ conventional techniques for the creation of a fundamental crack detection system, contributing to the automation of structural inspection processes. This endeavor is driven by the need to ensure the safety and reliability of various structures, in light of past incidents like the 2018 Ponte Morandi collapse. While structural health monitoring has been effective, the project seeks to enhance inspection capabilities through traditional methods.

Section 1 : Importing Libraries

In [423…
```
import os
import random
import cv2
import xml.etree.ElementTree as ET
import numpy as np
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report
from skimage.morphology import skeletonize
from sklearn.ensemble import RandomForestClassifier
```

Section 2 : Importing the annotations from CVAT tool

In [424…
```
# After performing the annotation task from CVAT, we are using the exported file
# We have used polyline and ellipse while annotating
xml_file_path = "annotations.xml"
tree = ET.parse(xml_file_path)
root = tree.getroot()
```

In [425…
```
# Setting a function for our module to parse the points out of annotations
def parse_points(points_str):
    points = points_str.strip().split(";")
    points = [tuple(map(float, point.split(","))) for point in points]
    return points
```

In [426…
```
# Setting a function for our module to parse the ellipse out of annotations
def parse_ellipse(ellipse_str):
    params = ellipse_str.strip().split(",")
    if len(params) == 5:
        cx, cy, major_axis, minor_axis, angle = map(float, params)
        return (cx, cy, major_axis, minor_axis, angle)
    else:
        return None
```

In [427…
```
# Here we would save the annotations in a dictionary
image_annotations = {}
```

In [428…
```
# Extracting the annotation information from xml

for image_elem in root.findall(".//image"):
    image_id = int(image_elem.get("id"))
```

```
    image_name = image_elem.get("name")

    # Initializing a list to store annotations for this image
    annotations = []

    # Iterating through polyline and ellipse elements
    for annotation_elem in image_elem.findall(".//polyline") + image_elem.findall(".//el
        label = annotation_elem.get("label")
        points = annotation_elem.get("points")

        # Appending the annotation as a dictionary to the list
        annotations.append({
            "label": label,
            "points": points,
        })

    # Storing the annotations for this image in the dictionary
    image_annotations[image_id] = {
        "image_name": image_name,
        "annotations": annotations,
    }
```

In [429…

```
print(image_annotations)
```


```
{0: {'image_name': '0c377323-c884-44f4-b9e1-5698521ad20a.JPG', 'annotations': [{'label':
'Crack', 'points': '672.92,1411.69;653.17,1377.12;638.35,1354.07;628.48,1322.80;648.23,1
288.23;661.40,1260.25;648.23,1225.68;672.92,1192.76;674.57,1171.36;669.63,1136.79'}, {'l
abel': 'Crack', 'points': '672.48,242.95;677.11,219.14;671.16,209.87;675.79,195.32;677.7
7,176.14;683.73,162.24'}, {'label': 'Crack', 'points': '673.16,1136.52;682.68,1101.59;68
1.09,1077.77;667.60,1042.84;673.16,1000.77;656.48,988.07;638.23,969.02;642.20,940.44;63
3.46,928.53;623.94,903.92;623.14,872.17;619.17,847.56;612.82,826.13;621.56,809.46;631.0
8,800.72'}, {'label': 'Crack', 'points': '619.17,791.99;612.82,765.00;601.71,748.33;589.
01,730.87;594.57,711.02;598.53,690.38;600.12,677.68;581.86,655.45;578.69,637.19;573.93,6
11.00;589.01,567.34;598.53,538.76;600.92,518.91;596.15,505.42;600.92,487.16;608.85,463.3
5;616.79,437.15;626.32,410.95;635.05,395.08'}, {'label': 'Crack', 'points': '641.39,389.
15;651.97,377.24;647.34,356.07;649.33,338.87;655.28,328.95;663.88,317.70;667.19,293.23;6
69.83,278.01;675.79,255.52'}]}, 1: {'image_name': '0d1415c4-099c-477f-a290-fea11c9f873f.
JPG', 'annotations': [{'label': 'Crack', 'points': '467.95,373.51;463.98,354.29'}, {'lab
el': 'Crack', 'points': '460.00,53.45;458.45,44.82;458.23,22.01;462.65,15.15;464.43,10.0
6;461.61,0.00'}, {'label': 'Crack', 'points': '460.88,115.22;460.44,107.92;462.21,102.3
8;459.11,91.98;460.00,84.45;461.77,76.04;465.98,65.85'}, {'label': 'Crack', 'points': '4
63.04,152.09;463.36,140.93;466.23,136.79;464.64,120.21'}, {'label': 'Crack', 'points':
'465.99,263.30;461.79,250.29;466.38,235.37;462.17,218.54;470.58,192.14;462.55,167.65;46
4.46,158.85'}, {'label': 'Crack', 'points': '467.06,445.26;463.85,438.60;465.78,395.93;4
70.77,385.17;470.13,381.58'}, {'label': 'Crack', 'points': '459.63,343.53;461.16,336.35;
461.16,325.85;463.85,318.80;462.06,304.07;461.80,298.56;462.96,291.64;465.52,285.49;466.
55,280.36;464.50,271.91;467.57,265.76'}, {'label': 'Crack', 'points': '453.09,1596.92;45
4.68,1575.50;448.33,1563.60;455.47,1556.46;456.27,1539.80;449.13,1523.94;455.47,1511.24;
455.47,1496.96;453.89,1491.41'}, {'label': 'Crack', 'points': '454.68,1461.26;459.10,144
5.80;460.90,1436.10;464.99,1422.39;461.82,1411.28;460.23,1398.59;461.82,1383.52;463.41,1
377.17;458.90,1366.00;464.70,1359.10;465.20,1349.20;468.10,1341.50;462.00,1324.10;464.9
9,1316.88;464.20,1305.77;464.99,1275.62;461.70,1261.60;466.40,1254.10;469.75,1227.23;46
7.37,1219.30;464.99,1207.40;464.99,1201.05'}, {'label': 'Crack', 'points': '467.00,1160.
15;473.05,1149.31;469.86,1131.77;471.78,1110.41;474.30,1105.34;473.85,1098.90;469.83,108
4.75;468.30,1063.39;463.23,1052.45'}, {'label': 'Crack', 'points': '463.23,1050.08;467.8
4,1043.54;467.20,1028.55;467.84,1023.17;465.79,1015.23;468.35,1012.41;467.58,1004.59;46
9.63,994.34;471.30,991.78;470.27,982.04'}, {'label': 'Crack', 'points': '469.63,979.80;4
69.20,971.26;467.28,967.74;469.84,965.28;467.71,956.31;464.82,952.04;465.47,947.13'},
{'label': 'Crack', 'points': '464.61,943.39;466.11,940.08;465.57,933.46;467.71,930.26;46
6.96,927.70'}, {'label': 'Crack', 'points': '467.07,925.46;469.74,922.47;467.28,910.40;4
64.18,905.06;464.18,885.52;470.59,875.81'}, {'label': 'Crack', 'points': '470.06,873.14;
471.12,866.62;471.87,863.31;475.18,857.34;475.29,842.71;485.33,821.03;491.52,816.55'},
{'label': 'Crack', 'points': '473.15,782.78;473.53,769.00;478.12,759.06;475.06,751.40;47
1.24,725.77;473.15,708.94;475.83,702.05;469.70,685.98;471.24,667.24;474.30,660.35;473.1
5,640.84;473.15,630.70;470.58,626.07;471.12,610.57;474.26,605.41'}, {'label': 'Crack',
```

'points': '473.58,598.50;473.96,589.15;471.91,583.64;472.43,573.52;472.17,568.78;469.74,
566.34;469.99,549.82;471.66,547.64'}, {'label': 'Crack', 'points': '469.75,538.03;469.6
2,527.65;468.21,522.01;472.44,514.58;471.67,510.61'}, {'label': 'Crack', 'points': '466.
93,501.64;467.19,494.08;466.93,467.81;468.98,462.43;470.13,452.69'}]}, 2: {'image_name':
'7aaef046-b790-4f97-a77c-19864009ff4d.JPG', 'annotations': [{'label': 'Crack', 'points':
'438.39,934.23;437.84,926.52;436.74,913.29;431.78,899.52;392.67,821.29;397.07,794.29;40
2.03,761.79;395.97,742.51;400.93,717.16;403.69,698.98;411.95,682.45;408.09,664.27;404.2
4,644.99;408.09,632.32;409.75,614.14'}, {'label': 'Crack', 'points': '410.54,614.49;411.
20,604.09;418.06,593.02;420.28,586.82;425.81,571.76'}]}, 3: {'image_name': 'IMG_9001.jp
g', 'annotations': [{'label': 'no-crack', 'points': '1391.09,1785.33;1394.86,1797.72;139
7.55,1802.29;1403.74,1813.06;1411.28,1814.14;1413.16,1817.37;1415.05,1825.72;1417.47,183
1.64;1426.89,1840.79;1429.86,1848.06;1439.82,1857.76;1448.70,1865.29'}, {'label': 'Crac
k', 'points': '1464.81,2258.97;1461.46,2236.19;1445.38,2224.13;1454.09,2207.38;1429.31,2
190.64;1424.62,2179.92;1421.27,2165.18;1428.64,2155.13;1434.67,2147.76'}, {'label': 'Cra
ck', 'points': '1441.37,2120.96;1455.00,2106.30;1459.20,2091.50;1461.60,2075.90;1462.60,
2062.00;1461.46,2051.29;1459.50,2043.30;1454.80,2029.90;1459.90,2017.40'}, {'label': 'Cr
ack', 'points': '1465.13,1957.91;1464.32,1949.30;1462.70,1938.26;1464.05,1926.41;1464.8
6,1917.26;1467.82,1909.99'}, {'label': 'Crack', 'points': '1414.00,1665.79;1420.14,1653.
50;1437.45,1637.31;1448.05,1628.38;1454.19,1614.42;1457.54,1596.00;1454.19,1573.11;1465.
36,1561.95'}, {'label': 'no-crack', 'points': '1457.54,1554.13;1448.05,1542.96;1444.70,1
536.82;1436.89,1527.89;1436.33,1520.63'}, {'label': 'Crack', 'points': '1430.65,1507.42;
1431.82,1501.60;1435.31,1493.85;1434.92,1487.65;1436.86,1481.83;1437.63,1473.69;1440.35,
1467.49;1440.35,1451.59;1441.90,1441.12;1441.90,1432.20;1434.14,1422.12'}, {'label': 'Cr
ack', 'points': '1434.18,1417.24;1437.28,1409.49;1435.34,1396.30'}, {'label': 'Crack',
'points': '1437.28,1381.96;1445.04,1363.74;1450.85,1352.88;1447.36,1336.98'}, {'label':
'Crack', 'points': '1455.51,1326.90;1459.77,1315.27;1460.16,1306.36;1467.14,1288.13;147
1.01,1279.61;1474.12,1274.18;1474.50,1266.42;1479.93,1254.79'}, {'label': 'no-crack', 'p
oints': '1490.01,1224.94;1490.79,1212.53;1490.79,1198.96;1500.09,1189.27;1502.81,1179.9
7'}, {'label': 'Crack', 'points': '1498.15,1052.02;1498.93,1043.11;1502.81,1034.58;1506.
68,1024.88;1511.33,1020.23;1512.11,1010.15'}, {'label': 'Crack', 'points': '1512.09,657.
90;1500.92,664.05;1485.29,688.05;1478.03,693.64;1479.15,704.24;1478.03,715.97'}, {'labe
l': 'Crack', 'points': '1488.99,737.86;1488.53,747.63;1485.27,754.14;1487.13,758.80'},
{'label': 'no-crack', 'points': '1518.73,835.72;1517.62,840.74;1517.62,861.40'}, {'labe
l': 'no-crack', 'points': '1516.98,604.59;1519.89,591.34;1524.74,578.74;1529.58,560.3
3'}, {'label': 'Crack', 'points': '1562.78,461.94;1562.22,450.78;1562.22,432.91;1559.99,
425.66;1556.08,413.37;1559.43,408.91'}, {'label': 'Crack', 'points': '1550.14,355.52;155
0.53,345.82;1551.69,333.03;1548.59,321.79;1549.37,317.52'}, {'label': 'no-crack', 'point
s': '1554.41,250.45;1555.18,241.53;1536.96,203.93'}, {'label': 'no-crack', 'points': '15
66.91,85.77;1574.73,71.25;1573.05,55.06;1580.31,43.90;1584.22,29.38;1576.40,18.77;1581.9
9,7.61'}, {'label': 'Crack', 'points': '967.94,4026.64;977.95,3993.30;966.28,3966.63;96
7.94,3933.29;979.61,3914.95'}, {'label': 'Crack', 'points': '1090.22,3584.04;1100.83,355
0.27;1114.34,3531.94;1124.95,3503.00;1131.70,3491.42;1146.17,3472.13;1150.03,3461.52;116
7.40,3449.94;1179.94,3438.36;1183.80,3429.68'}, {'label': 'Crack', 'points': '1231.84,34
00.83;1246.31,3385.40;1244.38,3359.35;1242.45,3333.30;1252.10,3317.87;1257.89,3303.40;12
63.68,3284.10'}, {'label': 'no-crack', 'points': '1233.56,3196.78;1235.17,3175.88;1224.7
2,3150.15;1215.87,3131.66'}, {'label': 'no-crack', 'points': '1212.85,3079.99;1222.89,30
68.60;1227.58,3033.77;1253.71,3000.27'}, {'label': 'Crack', 'points': '1264.50,2910.83;1
258.47,2884.70;1265.17,2871.30;1266.51,2846.52;1288.62,2828.43;1292.64,2811.68;1304.03,2
802.97;1304.03,2779.52;1317.42,2765.45'}, {'label': 'Crack', 'points': '1358.96,2620.07;
1362.98,2597.96;1348.91,2573.85;1347.57,2563.80;1358.96,2552.41;1375.71,2528.96;1375.71,
2513.55;1373.03,2500.15;1379.73,2492.78;1380.40,2477.37;1413.90,2460.63'}, {'label': 'no
-crack', 'points': '1439.36,2429.14;1449.40,2407.70;1448.06,2390.28;1434.67,2367.50;144
8.73,2350.08'}, {'label': 'no-crack', 'points': '1474.86,2323.29;1474.19,2307.88;1468.1
6,2301.18;1469.50,2283.76'}]}, 4: {'image_name': 'IMG_9003.jpg', 'annotations': [{'labe
l': 'Crack', 'points': None}, {'label': 'Crack', 'points': None}]}, 5: {'image_name': 'I
MG_9004.jpg', 'annotations': [{'label': 'no-crack', 'points': '795.01,3566.43;767.68,356
7.23;742.76,3552.76;697.73,3530.25'}, {'label': 'Crack', 'points': None}]}, 6: {'image_n
ame': 'IMG_9005.jpg', 'annotations': [{'label': 'no-crack', 'points': '6.75,2042.40;220.
92,2199.85'}, {'label': 'no-crack', 'points': '2635.31,836.89;2655.41,849.17;2667.70,85
9.22;2692.82,868.16;2689.47,896.63'}]}, 8: {'image_name': 'IMG_9007.jpg', 'annotations':
[{'label': 'Crack', 'points': '1698.61,2611.65;1716.95,2634.99;1691.94,2655.00;1683.61,2
685.00;1660.27,2720.01;1630.26,2745.02'}, {'label': 'Crack', 'points': '1911.33,2832.04;
1877.99,2852.04;1909.66,2898.72;1907.99,2930.39;1882.99,2963.73;1886.32,2990.41;1902.99,
3032.08'}, {'label': 'no-crack', 'points': '637.08,3359.83;650.59,3380.09;667.96,3383.9
5'}]}, 9: {'image_name': 'IMG_9008.jpg', 'annotations': [{'label': 'no-crack', 'points':
'826.38,2739.62;821.58,2638.80;821.58,2554.78;811.98,2451.56;807.18,2365.14;797.58,2293.

12;795.18,2204.30'}]}, 10: {'image_name': 'IMG_9009.jpg', 'annotations': [{'label': 'Crack', 'points': '379.64,1889.99;393.65,1970.01;431.66,1952.00;459.66,1982.01;473.67,2028.02;493.67,2032.02;539.68,2060.03'}, {'label': 'no-crack', 'points': '1485.98,2814.22;1515.02,3125.33;1502.58,3411.56;1515.02,3710.22;1556.50,3996.44'}]}, 11: {'image_name': 'c04464a0-0f86-40e2-87d0-6e8d3b9d4066.JPG', 'annotations': [{'label': 'Crack', 'points': '1.95,888.50;11.20,890.48;23.10,886.52;35.66,887.18;49.55,888.50;62.77,887.18;70.04,894.45;81.28,901.06;86.57,899.08;99.79,903.05;120.29,909.00;134.83,909.66;155.33,914.95;168.55,911.64;181.11,910.32;193.67,899.74;203.59,893.79;208.88,890.48;230.69,890.48;237.97,887.84;249.87,885.86;258.46,885.86;263.09,889.16;272.34,884.53;278.95,886.52;292.84,883.87'}, {'label': 'Crack', 'points': '628.61,880.48;623.28,880.48;619.54,881.20;615.65,883.79;611.19,886.10;604.71,889.98;601.25,891.71;595.06,892.14;586.99,889.55;583.97,885.95'}, {'label': 'Crack', 'points': '723.46,862.43;717.59,863.98;707.39,864.33;703.07,865.19;696.68,869.69;688.56,873.49;684.06,877.98;680.09,878.84;671.80,875.22;666.78,875.22;663.67,877.12;657.97,877.29;652.96,876.43;649.16,874.35;643.11,874.70;639.65,876.25;635.68,878.15'}, {'label': 'Crack', 'points': '852.97,842.91;845.80,842.91;841.32,841.41;836.90,843.59;821.08,842.58;815.76,848.34;808.84,847.18;805.82,849.92;793.43,849.62;789.96,850.79;784.08,855.39;776.74,855.95;769.09,852.07;766.64,853.51;752.40,853.24;743.17,850.49;738.70,852.94;736.28,853.95'}, {'label': 'Crack', 'points': '898.65,850.67;888.50,849.18;879.54,843.80;867.60,843.50'}, {'label': 'Crack', 'points': '297.61,884.22;309.04,886.61;321.26,886.88;333.75,888.47;347.56,887.15;356.06,883.96;369.08,882.10;376.79,882.90;382.90,886.08;392.20,888.74;400.43,886.61;406.55,884.49;409.10,886.80;416.80,885.90;422.19,882.10;428.92,884.20;437.79,884.78'}, {'label': 'Crack', 'points': '577.62,877.94;567.92,871.47;560.70,870.72;555.23,870.23;546.27,865.00;539.55,862.26;528.85,861.52;527.36,863.51;517.66,866.99;511.19,864.50;505.46,866.00;497.25,864.75;493.77,865.00;485.81,865.25;479.83,867.49;473.36,871.72;470.63,876.70;462.66,879.43;457.69,878.94;450.72,878.19;446.24,878.94;438.28,884.91'}]}}

In [430…
```
print(image_annotations[0]['annotations'])
```

[{'label': 'Crack', 'points': '672.92,1411.69;653.17,1377.12;638.35,1354.07;628.48,1322.80;648.23,1288.23;661.40,1260.25;648.23,1225.68;672.92,1192.76;674.57,1171.36;669.63,1136.79'}, {'label': 'Crack', 'points': '672.48,242.95;677.11,219.14;671.16,209.87;675.79,195.32;677.77,176.14;683.73,162.24'}, {'label': 'Crack', 'points': '673.16,1136.52;682.68,1101.59;681.09,1077.77;667.60,1042.84;673.16,1000.77;656.48,988.07;638.23,969.02;642.20,940.44;633.46,928.53;623.94,903.92;623.14,872.17;619.17,847.56;612.82,826.13;621.56,809.46;631.08,800.72'}, {'label': 'Crack', 'points': '619.17,791.99;612.82,765.00;601.71,748.33;589.01,730.87;594.57,711.02;598.53,690.38;600.12,677.68;581.86,655.45;578.69,637.19;573.93,611.00;589.01,567.34;598.53,538.76;600.92,518.91;596.15,505.42;600.92,487.16;608.85,463.35;616.79,437.15;626.32,410.95;635.05,395.08'}, {'label': 'Crack', 'points': '641.39,389.15;651.97,377.24;647.34,356.07;649.33,338.87;655.28,328.95;663.88,317.70;667.19,293.23;669.83,278.01;675.79,255.52'}]

Section 3 : Saving the annotations from CVAT tool for view

In [431…
```
image_paths = []
annotations_list = []
```

In [432…
```
for image_id, image_info in image_annotations.items():
    # Constructing the image path
    image_name = image_info['image_name']
    image_path = os.path.join("image/", image_name)

    # Appending the image path and annotations to the respective lists
    image_paths.append(image_path)
    annotations_list.append(image_info['annotations'])
print(image_paths)
```

['image/0c377323-c884-44f4-b9e1-5698521ad20a.JPG', 'image/0d1415c4-099c-477f-a290-fea11c9f873f.JPG', 'image/7aaef046-b790-4f97-a77c-19864009ff4d.JPG', 'image/IMG_9001.jpg', 'image/IMG_9003.jpg', 'image/IMG_9004.jpg', 'image/IMG_9005.jpg', 'image/IMG_9007.jpg', 'image/IMG_9008.jpg', 'image/IMG_9009.jpg', 'image/c04464a0-0f86-40e2-87d0-6e8d3b9d4066.JPG']

In [433…
```
# Defining the folder to save the annotated images
```

```
output_folder = "reflecting_annotations"

# Creating the folder if it doesn't exist
if not os.path.exists(output_folder):
    os.makedirs(output_folder)

# Visualizing images with annotations
for i in range(len(image_paths)):
    image_path = image_paths[i]
    annotations = annotations_list[i]

    # Loading the image
    original_image = cv2.imread(image_path)

    # Horizontal flip
    should_flip = False
    if should_flip:
        image = cv2.flip(original_image, 1)  # 1 indicates horizontal flip
    else:
        image = original_image

    # Drawing annotations on the image
    for annotation in annotations:
        points_str = annotation['points']
        ellipse_str = annotation.get('ellipse', '')

        if points_str:
            points = parse_points(points_str)

            # Checking if the polygon is closed
            is_closed = points[0] == points[-1]

            # Drawing the polygon without connecting first and last points if it's not c
            if not is_closed:
                points = points[:-1]

            points = np.array(points, np.int32)
            cv2.polylines(image, [points], isClosed=is_closed, color=(0, 0, 255), thickn

        if ellipse_str:
            ellipse = parse_ellipse(ellipse_str)
            if ellipse is not None:
                cx, cy, major_axis, minor_axis, angle = ellipse
                cv2.ellipse(image, (int(cx), int(cy)), (int(major_axis / 2), int(minor_a

    # Displaying the image with annotations
    #cv2.imshow("Image with Annotations", image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

    # Saving the displayed image in the "reflecting_annotations" folder
    image_name = os.path.splitext(os.path.basename(image_path))[0]
    output_path = os.path.join(output_folder, f"{image_name}_annotated.png")
    cv2.imwrite(output_path, image)
```

Section 4 : Splitting up the dataset

In [434…
```
# Setting up our training and testing data in the ratio 80:20
train_image_paths, test_image_paths, train_annotations, test_annotations = train_test_sp
    image_paths, annotations_list, test_size=0.2, random_state=42
)
```

In [435…
```
print(train_annotations)
```

[[{'label': 'Crack', 'points': '1.95,888.50;11.20,890.48;23.10,886.52;35.66,887.18;49.55,888.50;62.77,887.18;70.04,894.45;81.28,901.06;86.57,899.08;99.79,903.05;120.29,909.00;134.83,909.66;155.33,914.95;168.55,911.64;181.11,910.32;193.67,899.74;203.59,893.79;208.88,890.48;230.69,890.48;237.97,887.84;249.87,885.86;258.46,885.86;263.09,889.16;272.34,884.53;278.95,886.52;292.84,883.87'}, {'label': 'Crack', 'points': '628.61,880.48;623.28,880.48;619.54,881.20;615.65,883.79;611.19,886.10;604.71,889.98;601.25,891.71;595.06,892.14;586.99,889.55;583.97,885.95'}, {'label': 'Crack', 'points': '723.46,862.43;717.59,863.98;707.39,864.33;703.07,865.19;696.68,869.69;688.56,873.49;684.06,877.98;680.09,878.84;671.80,875.22;666.78,875.22;663.67,877.12;657.97,877.29;652.96,876.43;649.16,874.35;643.11,874.70;639.65,876.25;635.68,878.15'}, {'label': 'Crack', 'points': '852.97,842.91;845.80,842.91;841.32,841.41;836.90,843.59;821.08,842.58;815.76,848.34;808.84,847.18;805.82,849.92;793.43,849.62;789.96,850.79;784.08,855.39;776.74,855.95;769.09,852.07;766.64,853.51;752.40,853.24;743.17,850.49;738.70,852.94;736.28,853.95'}, {'label': 'Crack', 'points': '898.65,850.67;888.50,849.18;879.54,843.80;867.60,843.50'}, {'label': 'Crack', 'points': '297.61,884.22;309.04,886.61;321.26,886.88;333.75,888.47;347.56,887.15;356.06,883.96;369.08,882.10;376.79,882.90;382.90,886.08;392.20,888.74;400.43,886.61;406.55,884.49;409.10,886.80;416.80,885.90;422.19,882.10;428.92,884.20;437.79,884.78'}, {'label': 'Crack', 'points': '577.62,877.94;567.92,871.47;560.70,870.72;555.23,870.23;546.27,865.00;539.55,862.26;528.85,861.52;527.36,863.51;517.66,866.99;511.19,864.50;505.46,866.00;497.25,864.75;493.77,865.00;485.81,865.25;479.83,867.49;473.36,871.72;470.63,876.70;462.66,879.43;457.69,878.94;450.72,878.19;446.24,878.94;438.28,884.91'}], [{'label': 'Crack', 'points': '438.39,934.23;437.84,926.52;436.74,913.29;431.78,899.52;392.67,821.29;397.07,794.29;402.03,761.79;395.97,742.51;400.93,717.16;403.69,698.98;411.95,682.45;408.09,664.27;404.24,644.99;408.09,632.32;409.75,614.14'}, {'label': 'Crack', 'points': '410.54,614.49;411.20,604.09;418.06,593.02;420.28,586.82;425.81,571.76'}], [{'label': 'Crack', 'points': '467.95,373.51;463.98,354.29'}, {'label': 'Crack', 'points': '460.00,53.45;458.45,44.82;458.23,22.01;462.65,15.15;464.43,10.06;461.61,0.00'}, {'label': 'Crack', 'points': '460.88,115.22;460.44,107.92;462.21,102.38;459.11,91.98;460.00,84.45;461.77,76.04;465.98,65.85'}, {'label': 'Crack', 'points': '463.04,152.09;463.36,140.93;466.23,136.79;464.64,120.21'}, {'label': 'Crack', 'points': '465.99,263.30;461.79,250.29;466.38,235.37;462.17,218.54;470.58,192.14;462.55,167.65;464.46,158.85'}, {'label': 'Crack', 'points': '467.06,445.26;463.85,438.60;465.78,395.93;470.77,385.17;470.13,381.58'}, {'label': 'Crack', 'points': '459.63,343.53;461.16,336.35;461.16,325.85;463.85,318.80;462.06,304.07;461.80,298.56;462.96,291.64;465.52,285.49;466.55,280.36;464.50,271.91;467.57,265.76'}, {'label': 'Crack', 'points': '453.09,1596.92;454.68,1575.50;448.33,1563.60;455.47,1556.46;456.27,1539.80;449.13,1523.94;455.47,1511.24;455.47,1496.96;453.89,1491.41'}, {'label': 'Crack', 'points': '454.68,1461.26;459.10,1445.80;460.90,1436.10;464.99,1422.39;461.82,1411.28;460.23,1398.59;461.82,1383.52;463.41,1377.17;458.90,1366.00;464.70,1359.10;465.20,1349.20;468.10,1341.50;462.00,1324.10;464.99,1316.88;464.20,1305.77;464.99,1275.62;461.70,1261.60;466.40,1254.10;469.75,1227.23;467.37,1219.30;464.99,1207.40;464.99,1201.05'}, {'label': 'Crack', 'points': '467.00,1160.15;473.05,1149.31;469.86,1131.77;471.78,1110.41;474.30,1105.34;473.85,1098.90;469.83,1084.75;468.30,1063.39;463.23,1052.45'}, {'label': 'Crack', 'points': '463.23,1050.08;467.84,1043.54;467.20,1028.55;467.84,1023.17;465.79,1015.23;468.35,1012.41;467.58,1004.59;469.63,994.34;471.30,991.78;470.27,982.04'}, {'label': 'Crack', 'points': '469.63,979.80;469.20,971.26;467.28,967.74;469.84,965.28;467.71,956.31;464.82,952.04;465.47,947.13'}, {'label': 'Crack', 'points': '464.61,943.39;466.11,940.08;465.57,933.46;467.71,930.26;466.96,927.70'}, {'label': 'Crack', 'points': '467.07,925.46;469.74,922.47;467.28,910.40;464.18,905.06;464.18,885.52;470.59,875.81'}, {'label': 'Crack', 'points': '470.06,873.14;471.12,866.62;471.87,863.31;475.18,857.34;475.29,842.71;485.33,821.03;491.52,816.55'}, {'label': 'Crack', 'points': '473.15,782.78;473.53,769.00;478.12,759.06;475.06,751.40;471.24,725.77;473.15,708.94;475.83,702.05;469.70,685.98;471.24,667.24;474.30,660.35;473.15,640.84;473.15,630.70;470.58,626.07;471.12,610.57;474.26,605.41'}, {'label': 'Crack', 'points': '473.58,598.50;473.96,589.15;471.91,583.64;472.43,573.52;472.17,568.78;469.74,566.34;469.99,549.82;471.66,547.64'}, {'label': 'Crack', 'points': '469.75,538.03;469.62,527.65;468.21,522.01;472.44,514.58;471.67,510.61'}, {'label': 'Crack', 'points': '466.93,501.64;467.19,494.08;466.93,467.81;468.98,462.43;470.13,452.69'}], [{'label': 'no-crack', 'points': '826.38,2739.62;821.58,2638.80;821.58,2554.78;811.98,2451.56;807.18,2365.14;797.58,2293.12;795.18,2204.30'}], [{'label': 'Crack', 'points': None}, {'label': 'Crack', 'points': None}], [{'label': 'Crack', 'points': '1698.61,2611.65;1716.95,2634.99;1691.94,2655.00;1683.61,2685.00;1660.27,2720.01;1630.26,2745.02'}, {'label': 'Crack', 'points': '1911.33,2832.04;1877.99,2852.04;1909.66,2898.72;1907.99,2930.39;1882.99,2963.73;1886.32,2990.41;1902.99,3032.08'}, {'label': 'no-crack', 'points': '637.08,3359.83;650.59,3380.09;667.96,3383.95'}], [{'label': 'no-crack', 'points': '1391.09,1785.33;1394.86,1797.72;1397.55,1802.29;1403.74,1813.06;1411.28,1814.14;1413.16,1817.37;1415.05,1825.72;1417.47,1831.64;1426.89,1840.79;1429.86,1848.06;1439.82,1857.76;1448.70,1865.29'}, {'label': 'Crack', 'points': '1464.81,2258.97;1461.46,

2236.19;1445.38,2224.13;1454.09,2207.38;1429.31,2190.64;1424.62,2179.92;1421.27,2165.18;
1428.64,2155.13;1434.67,2147.76'}, {'label': 'Crack', 'points': '1441.37,2120.96;1455.0
0,2106.30;1459.20,2091.50;1461.60,2075.90;1462.60,2062.00;1461.46,2051.29;1459.50,2043.3
0;1454.80,2029.90;1459.90,2017.40'}, {'label': 'Crack', 'points': '1465.13,1957.91;1464.
32,1949.30;1462.70,1938.26;1464.05,1926.41;1464.86,1917.26;1467.82,1909.99'}, {'label':
'Crack', 'points': '1414.00,1665.79;1420.14,1653.50;1437.45,1637.31;1448.05,1628.38;145
4.19,1614.42;1457.54,1596.00;1454.19,1573.11;1465.36,1561.95'}, {'label': 'no-crack', 'p
oints': '1457.54,1554.13;1448.05,1542.96;1444.70,1536.82;1436.89,1527.89;1436.33,1520.6
3'}, {'label': 'Crack', 'points': '1430.65,1507.42;1431.82,1501.60;1435.31,1493.85;1434.
92,1487.65;1436.86,1481.83;1437.63,1473.69;1440.35,1467.49;1440.35,1451.59;1441.90,1441.
12;1441.90,1432.20;1434.14,1422.12'}, {'label': 'Crack', 'points': '1434.18,1417.24;143
7.28,1409.49;1435.34,1396.30'}, {'label': 'Crack', 'points': '1437.28,1381.96;1445.04,13
63.74;1450.85,1352.88;1447.36,1336.98'}, {'label': 'Crack', 'points': '1455.51,1326.90;1
459.77,1315.27;1460.16,1306.36;1467.14,1288.13;1471.01,1279.61;1474.12,1274.18;1474.50,1
266.42;1479.93,1254.79'}, {'label': 'no-crack', 'points': '1490.01,1224.94;1490.79,1212.
53;1490.79,1198.96;1500.09,1189.27;1502.81,1179.97'}, {'label': 'Crack', 'points': '149
8.15,1052.02;1498.93,1043.11;1502.81,1034.58;1506.68,1024.88;1511.33,1020.23;1512.11,101
0.15'}, {'label': 'Crack', 'points': '1512.09,657.90;1500.92,664.05;1485.29,688.05;1478.
03,693.64;1479.15,704.24;1478.03,715.97'}, {'label': 'Crack', 'points': '1488.99,737.86;
1488.53,747.63;1485.27,754.14;1487.13,758.80'}, {'label': 'no-crack', 'points': '1518.7
3,835.72;1517.62,840.74;1517.62,861.40'}, {'label': 'no-crack', 'points': '1516.98,604.5
9;1519.89,591.34;1524.74,578.74;1529.58,560.33'}, {'label': 'Crack', 'points': '1562.78,
461.94;1562.22,450.78;1562.22,432.91;1559.99,425.66;1556.08,413.37;1559.43,408.91'}, {'l
abel': 'Crack', 'points': '1550.14,355.52;1550.53,345.82;1551.69,333.03;1548.59,321.79;1
549.37,317.52'}, {'label': 'no-crack', 'points': '1554.41,250.45;1555.18,241.53;1536.96,
203.93'}, {'label': 'no-crack', 'points': '1566.91,85.77;1574.73,71.25;1573.05,55.06;158
0.31,43.90;1584.22,29.38;1576.40,18.77;1581.99,7.61'}, {'label': 'Crack', 'points': '96
7.94,4026.64;977.95,3993.30;966.28,3966.63;967.94,3933.29;979.61,3914.95'}, {'label': 'C
rack', 'points': '1090.22,3584.04;1100.83,3550.27;1114.34,3531.94;1124.95,3503.00;1131.7
0,3491.42;1146.17,3472.13;1150.03,3461.52;1167.40,3449.94;1179.94,3438.36;1183.80,3429.6
8'}, {'label': 'Crack', 'points': '1231.84,3400.83;1246.31,3385.40;1244.38,3359.35;1242.
45,3333.30;1252.10,3317.87;1257.89,3303.40;1263.68,3284.10'}, {'label': 'no-crack', 'poi
nts': '1233.56,3196.78;1235.17,3175.88;1224.72,3150.15;1215.87,3131.66'}, {'label': 'no-
crack', 'points': '1212.85,3079.99;1222.89,3068.60;1227.58,3033.77;1253.71,3000.27'},
{'label': 'Crack', 'points': '1264.50,2910.83;1258.47,2884.70;1265.17,2871.30;1266.51,28
46.52;1288.62,2828.43;1292.64,2811.68;1304.03,2802.97;1304.03,2779.52;1317.42,2765.45'},
{'label': 'Crack', 'points': '1358.96,2620.07;1362.98,2597.96;1348.91,2573.85;1347.57,25
63.80;1358.96,2552.41;1375.71,2528.96;1375.71,2513.55;1373.03,2500.15;1379.73,2492.78;13
80.40,2477.37;1413.90,2460.63'}, {'label': 'no-crack', 'points': '1439.36,2429.14;1449.4
0,2407.70;1448.06,2390.28;1434.67,2367.50;1448.73,2350.08'}, {'label': 'no-crack', 'poin
ts': '1474.86,2323.29;1474.19,2307.88;1468.16,2301.18;1469.50,2283.76'}], [{'label': 'no
-crack', 'points': '6.75,2042.40;220.92,2199.85'}, {'label': 'no-crack', 'points': '263
5.31,836.89;2655.41,849.17;2667.70,859.22;2692.82,868.16;2689.47,896.63'}]]

Section 5 : Data Augmentation (Horizontal flipping, Rotation & Contrast Adjustment)

In [436…

```
# Defining the path to the augmented images folder
augmented_images_folder = 'augmented_images'

# Creating the folder if it doesn't exist
os.makedirs(augmented_images_folder, exist_ok=True)

def augment_image(image, annotations):

    # Applying horizontal flipping
    if random.random() > 0.5:
        image = cv2.flip(image, 1)  # Flip horizontally
        # Updating annotation points
        for annotation in annotations:
            points_str = annotation['points']
            if points_str:
                points = parse_points(points_str)
                for i in range(len(points)):
```

```
                    points[i] = (image.shape[1] - points[i][0], points[i][1])
                annotation['points'] = ';'.join([f"{x},{y}" for x, y in points])

    # Randomly applying rotation (between -10 and 10 degrees)
    angle = random.uniform(-10, 10)
    rows, cols, _ = image.shape
    rotation_matrix = cv2.getRotationMatrix2D((cols / 2, rows / 2), angle, 1)
    image = cv2.warpAffine(image, rotation_matrix, (cols, rows))

    # Randomly adjusting contrast and brightness
    alpha = 1.0 + random.uniform(-0.2, 0.2)  # Contrast adjustment
    beta = random.uniform(-30, 30)  # Brightness adjustment
    image = cv2.convertScaleAbs(image, alpha=alpha, beta=beta)

    return image, annotations

# Augmenting the training dataset
augmented_train_images = []
augmented_train_annotations = []
for idx, (image_path, annotations) in enumerate(zip(train_image_paths, train_annotations
    image = cv2.imread(image_path)
    augmented_image, augmented_annotations = augment_image(image, annotations)

    # Defining the path to save the augmented image
    image_filename = os.path.basename(image_path)
    augmented_image_path = os.path.join(augmented_images_folder, f'augmented_{idx}_{imag

    # Saving the augmented image
    cv2.imwrite(augmented_image_path, augmented_image)

    augmented_train_images.append(augmented_image)
    augmented_train_annotations.append(augmented_annotations)
```

In [437…

```
# Displaying augmented images
for i, augmented_image in enumerate(augmented_train_images):
    #cv2.imshow(f"Augmented Image {i + 1}", augmented_image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

In [438…

```
print(train_annotations)
```

```
[[{'label': 'Crack', 'points': '897.05,888.5;887.8,890.48;875.9,886.52;863.34,887.18;84
9.45,888.5;836.23,887.18;828.96,894.45;817.72,901.06;812.4300000000001,899.08;799.21,90
3.05;778.71,909.0;764.17,909.66;743.67,914.95;730.45,911.64;717.89,910.32;705.33,899.74;
695.41,893.79;690.12,890.48;668.31,890.48;661.03,887.84;649.13,885.86;640.54,885.86;635.
9100000000001,889.16;626.6600000000001,884.53;620.05,886.52;606.1600000000001,883.87'},
{'label': 'Crack', 'points': '270.39,880.48;275.72,880.48;279.46000000000004,881.2;283.3
5,883.79;287.80999999999995,886.1;294.28999999999996,889.98;297.75,891.71;303.9400000000
0005,892.14;312.01,889.55;315.03,885.95'}, {'label': 'Crack', 'points': '175.53999999999
996,862.43;181.40999999999997,863.98;191.61,864.33;195.92999999999995,865.19;202.3200000
0000005,869.69;210.44000000000005,873.49;214.94000000000005,877.98;218.90999999999997,87
8.84;227.20000000000005,875.22;232.22000000000003,875.22;235.33000000000004,877.12;241.0
2999999999997,877.29;246.03999999999996,876.43;249.84000000000003,874.35;255.89,874.7;25
9.35,876.25;263.32000000000005,878.15'}, {'label': 'Crack', 'points': '46.0299999999999
7,842.91;53.200000000000045,842.91;57.67999999999995,841.41;62.10000000000002,843.59;77.
91999999999996,842.58;83.24000000000001,848.34;90.15999999999997,847.18;93.1799999999999
5,849.92;105.57000000000005,849.62;109.03999999999996,850.79;114.91999999999996,855.39;1
22.25999999999999,855.95;129.90999999999997,852.07;132.36,853.51;146.60000000000002,853.
24;155.83000000000004,850.49;160.29999999999995,852.94;162.72000000000003,853.95'}, {'la
bel': 'Crack', 'points': '0.35000000000002274,850.67;10.5,849.18;19.460000000000036,843.
8;31.399999999999977,843.5'}, {'label': 'Crack', 'points': '601.39,884.22;589.96,886.61;
577.74,886.88;565.25,888.47;551.44,887.15;542.94,883.96;529.9200000000001,882.1;522.21,8
82.9;516.1,886.08;506.8,888.74;498.57,886.61;492.45,884.49;489.9,886.8;482.2,885.9;476.8
1,882.1;470.08,884.2;461.21,884.78'}, {'label': 'Crack', 'points': '321.38,877.94;331.08
000000000004,871.47;338.29999999999995,870.72;343.77,870.23;352.73,865.0;359.45000000000
```

005,862.26;370.15,861.52;371.64,863.51;381.34000000000003,866.99;387.81,864.5;393.54,86
6.0;401.75,864.75;405.23,865.0;413.19,865.25;419.17,867.49;425.64,871.72;428.37,876.7;43
6.34,879.43;441.31,878.94;448.28,878.19;452.76,878.94;460.72,884.91'}], [{'label': 'Crac
k', 'points': '438.39,934.23;437.84,926.52;436.74,913.29;431.78,899.52;392.67,821.29;39
7.07,794.29;402.03,761.79;395.97,742.51;400.93,717.16;403.69,698.98;411.95,682.45;408.0
9,664.27;404.24,644.99;408.09,632.32;409.75,614.14'}, {'label': 'Crack', 'points': '410.
54,614.49;411.20,604.09;418.06,593.02;420.28,586.82;425.81,571.76'}], [{'label': 'Crac
k', 'points': '467.95,373.51;463.98,354.29'}, {'label': 'Crack', 'points': '460.00,53.4
5;458.45,44.82;458.23,22.01;462.65,15.15;464.43,10.06;461.61,0.00'}, {'label': 'Crack',
'points': '460.88,115.22;460.44,107.92;462.21,102.38;459.11,91.98;460.00,84.45;461.77,7
6.04;465.98,65.85'}, {'label': 'Crack', 'points': '463.04,152.09;463.36,140.93;466.23,13
6.79;464.64,120.21'}, {'label': 'Crack', 'points': '465.99,263.30;461.79,250.29;466.38,2
35.37;462.17,218.54;470.58,192.14;462.55,167.65;464.46,158.85'}, {'label': 'Crack', 'poi
nts': '467.06,445.26;463.85,438.60;465.78,395.93;470.77,385.17;470.13,381.58'}, {'labe
l': 'Crack', 'points': '459.63,343.53;461.16,336.35;461.16,325.85;463.85,318.80;462.06,3
04.07;461.80,298.56;462.96,291.64;465.52,285.49;466.55,280.36;464.50,271.91;467.57,265.7
6'}, {'label': 'Crack', 'points': '453.09,1596.92;454.68,1575.50;448.33,1563.60;455.47,1
556.46;456.27,1539.80;449.13,1523.94;455.47,1511.24;455.47,1496.96;453.89,1491.41'}, {'l
abel': 'Crack', 'points': '454.68,1461.26;459.10,1445.80;460.90,1436.10;464.99,1422.39;4
61.82,1411.28;460.23,1398.59;461.82,1383.52;463.41,1377.17;458.90,1366.00;464.70,1359.1
0;465.20,1349.20;468.10,1341.50;462.00,1324.10;464.99,1316.88;464.20,1305.77;464.99,127
5.62;461.70,1261.60;466.40,1254.10;469.75,1227.23;467.37,1219.30;464.99,1207.40;464.99,1
201.05'}, {'label': 'Crack', 'points': '467.00,1160.15;473.05,1149.31;469.86,1131.77;47
1.78,1110.41;474.30,1105.34;473.85,1098.90;469.83,1084.75;468.30,1063.39;463.23,1052.4
5'}, {'label': 'Crack', 'points': '463.23,1050.08;467.84,1043.54;467.20,1028.55;467.84,1
023.17;465.79,1015.23;468.35,1012.41;467.58,1004.59;469.63,994.34;471.30,991.78;470.27,9
82.04'}, {'label': 'Crack', 'points': '469.63,979.80;469.20,971.26;467.28,967.74;469.84,
965.28;467.71,956.31;464.82,952.04;465.47,947.13'}, {'label': 'Crack', 'points': '464.6
1,943.39;466.11,940.08;465.57,933.46;467.71,930.26;466.96,927.70'}, {'label': 'Crack',
'points': '467.07,925.46;469.74,922.47;467.28,910.40;464.18,905.06;464.18,885.52;470.59,
875.81'}, {'label': 'Crack', 'points': '470.06,873.14;471.12,866.62;471.87,863.31;475.1
8,857.34;475.29,842.71;485.33,821.03;491.52,816.55'}, {'label': 'Crack', 'points': '473.
15,782.78;473.53,769.00;478.12,759.06;475.06,751.40;471.24,725.77;473.15,708.94;475.83,7
02.05;469.70,685.98;471.24,667.24;474.30,660.35;473.15,640.84;473.15,630.70;470.58,626.0
7;471.12,610.57;474.26,605.41'}, {'label': 'Crack', 'points': '473.58,598.50;473.96,589.
15;471.91,583.64;472.43,573.52;472.17,568.78;469.74,566.34;469.99,549.82;471.66,547.6
4'}, {'label': 'Crack', 'points': '469.75,538.03;469.62,527.65;468.21,522.01;472.44,514.
58;471.67,510.61'}, {'label': 'Crack', 'points': '466.93,501.64;467.19,494.08;466.93,46
7.81;468.98,462.43;470.13,452.69'}], [{'label': 'no-crack', 'points': '2197.62,2739.62;2
202.42,2638.8;2202.42,2554.78;2212.02,2451.56;2216.82,2365.14;2226.42,2293.12;2228.82,22
04.3'}], [{'label': 'Crack', 'points': None}, {'label': 'Crack', 'points': None}], [{'la
bel': 'Crack', 'points': '1325.39,2611.65;1307.05,2634.99;1332.06,2655.0;1340.39,2685.0;
1363.73,2720.01;1393.74,2745.02'}, {'label': 'Crack', 'points': '1112.67,2832.04;1146.0
1,2852.04;1114.34,2898.72;1116.01,2930.39;1141.01,2963.73;1137.68,2990.41;1121.01,3032.0
8'}, {'label': 'no-crack', 'points': '2386.92,3359.83;2373.41,3380.09;2356.04,3383.9
5'}], [{'label': 'no-crack', 'points': '1391.09,1785.33;1394.86,1797.72;1397.55,1802.29;
1403.74,1813.06;1411.28,1814.14;1413.16,1817.37;1415.05,1825.72;1417.47,1831.64;1426.89,
1840.79;1429.86,1848.06;1439.82,1857.76;1448.70,1865.29'}, {'label': 'Crack', 'points':
'1464.81,2258.97;1461.46,2236.19;1445.38,2224.13;1454.09,2207.38;1429.31,2190.64;1424.6
2,2179.92;1421.27,2165.18;1428.64,2155.13;1434.67,2147.76'}, {'label': 'Crack', 'point
s': '1441.37,2120.96;1455.00,2106.30;1459.20,2091.50;1461.60,2075.90;1462.60,2062.00;146
1.46,2051.29;1459.50,2043.30;1454.80,2029.90;1459.90,2017.40'}, {'label': 'Crack', 'poin
ts': '1465.13,1957.91;1464.32,1949.30;1462.70,1938.26;1464.05,1926.41;1464.86,1917.26;14
67.82,1909.99'}, {'label': 'Crack', 'points': '1414.00,1665.79;1420.14,1653.50;1437.45,1
637.31;1448.05,1628.38;1454.19,1614.42;1457.54,1596.00;1454.19,1573.11;1465.36,1561.9
5'}, {'label': 'no-crack', 'points': '1457.54,1554.13;1448.05,1542.96;1444.70,1536.82;14
36.89,1527.89;1436.33,1520.63'}, {'label': 'Crack', 'points': '1430.65,1507.42;1431.82,1
501.60;1435.31,1493.85;1434.92,1487.65;1436.86,1481.83;1437.63,1473.69;1440.35,1467.49;1
440.35,1451.59;1441.90,1441.12;1441.90,1432.20;1434.14,1422.12'}, {'label': 'Crack', 'po
ints': '1434.18,1417.24;1437.28,1409.49;1435.34,1396.30'}, {'label': 'Crack', 'points':
'1437.28,1381.96;1445.04,1363.74;1450.85,1352.88;1447.36,1336.98'}, {'label': 'Crack',
'points': '1455.51,1326.90;1459.77,1315.27;1460.16,1306.36;1467.14,1288.13;1471.01,1279.
61;1474.12,1274.18;1474.50,1266.42;1479.93,1254.79'}, {'label': 'no-crack', 'points': '1
490.01,1224.94;1490.79,1212.53;1490.79,1198.96;1500.09,1189.27;1502.81,1179.97'}, {'labe
l': 'Crack', 'points': '1498.15,1052.02;1498.93,1043.11;1502.81,1034.58;1506.68,1024.88;
1511.33,1020.23;1512.11,1010.15'}, {'label': 'Crack', 'points': '1512.09,657.90;1500.92,

664.05;1485.29,688.05;1478.03,693.64;1479.15,704.24;1478.03,715.97'}, {'label': 'Crack', 'points': '1488.99,737.86;1488.53,747.63;1485.27,754.14;1487.13,758.80'}, {'label': 'no-crack', 'points': '1518.73,835.72;1517.62,840.74;1517.62,861.40'}, {'label': 'no-crack', 'points': '1516.98,604.59;1519.89,591.34;1524.74,578.74;1529.58,560.33'}, {'label': 'Crack', 'points': '1562.78,461.94;1562.22,450.78;1562.22,432.91;1559.99,425.66;1556.08,413.37;1559.43,408.91'}, {'label': 'Crack', 'points': '1550.14,355.52;1550.53,345.82;1551.69,333.03;1548.59,321.79;1549.37,317.52'}, {'label': 'no-crack', 'points': '1554.41,250.45;1555.18,241.53;1536.96,203.93'}, {'label': 'no-crack', 'points': '1566.91,85.77;1574.73,71.25;1573.05,55.06;1580.31,43.90;1584.22,29.38;1576.40,18.77;1581.99,7.61'}, {'label': 'Crack', 'points': '967.94,4026.64;977.95,3993.30;966.28,3966.63;967.94,3933.29;979.61,3914.95'}, {'label': 'Crack', 'points': '1090.22,3584.04;1100.83,3550.27;1114.34,3531.94;1124.95,3503.00;1131.70,3491.42;1146.17,3472.13;1150.03,3461.52;1167.40,3449.94;1179.94,3438.36;1183.80,3429.68'}, {'label': 'Crack', 'points': '1231.84,3400.83;1246.31,3385.40;1244.38,3359.35;1242.45,3333.30;1252.10,3317.87;1257.89,3303.40;1263.68,3284.10'}, {'label': 'no-crack', 'points': '1233.56,3196.78;1235.17,3175.88;1224.72,3150.15;1215.87,3131.66'}, {'label': 'no-crack', 'points': '1212.85,3079.99;1222.89,3068.60;1227.58,3033.77;1253.71,3000.27'}, {'label': 'Crack', 'points': '1264.50,2910.83;1258.47,2884.70;1265.17,2871.30;1266.51,2846.52;1288.62,2828.43;1292.64,2811.68;1304.03,2802.97;1304.03,2779.52;1317.42,2765.45'}, {'label': 'Crack', 'points': '1358.96,2620.07;1362.98,2597.96;1348.91,2573.85;1347.57,2563.80;1358.96,2552.41;1375.71,2528.96;1375.71,2513.55;1373.03,2500.15;1379.73,2492.78;1380.40,2477.37;1413.90,2460.63'}, {'label': 'no-crack', 'points': '1439.36,2429.14;1449.40,2407.70;1448.06,2390.28;1434.67,2367.50;1448.73,2350.08'}, {'label': 'no-crack', 'points': '1474.86,2323.29;1474.19,2307.88;1468.16,2301.18;1469.50,2283.76'}], [{'label': 'no-crack', 'points': '3017.25,2042.4;2803.08,2199.85'}, {'label': 'no-crack', 'points': '388.69000000000005,836.89;368.59000000000015,849.17;356.3000000000002,859.22;331.17999999999984,868.16;334.5300000000002,896.63'}]]

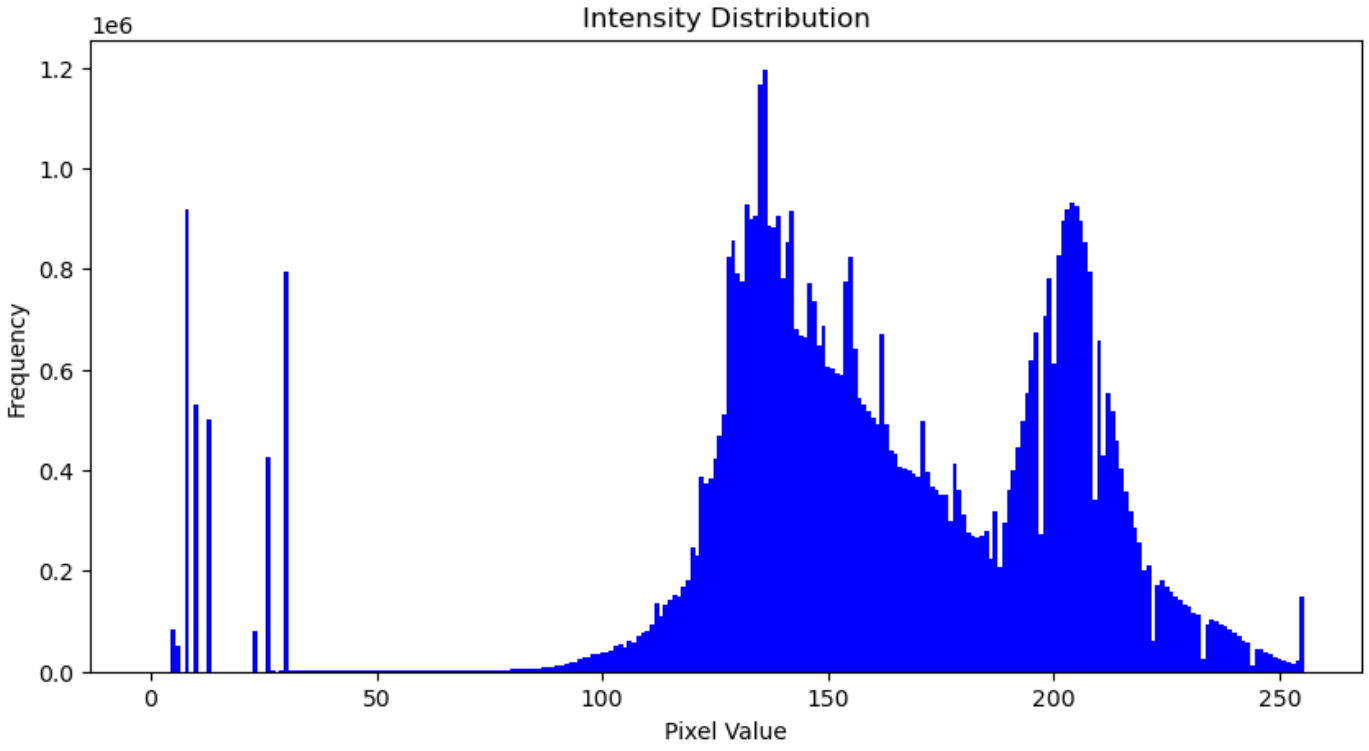
Section 6 : Dataset Statistics

In [439…

```python
# Calculating dataset statistics
num_images = len(image_paths)
num_pixels = sum(image.shape[0] * image.shape[1] for image in augmented_train_images)

# Here we have considered histogram, number of pixels as statistics
# We will consider white pixels and annotated labels
# Calculating intensity distribution
intensity_distribution = np.zeros(256, dtype=int)
for image in augmented_train_images:
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    hist = cv2.calcHist([gray_image], [0], None, [256], [0, 256])
    intensity_distribution += hist.flatten().astype(int)

# Calculating the number of labeled pixels
num_labeled_pixels = 0
for annotations in augmented_train_annotations:
    for annotation in annotations:
        points_str = annotation['points']
        if points_str:
            points = parse_points(points_str)
            num_labeled_pixels += len(points)

# Displaying basic statistics
print(f"Number of images: {num_images}")
print(f"Number of pixels: {num_pixels}")
print(f"Number of labeled pixels: {num_labeled_pixels}")

# Plotting intensity distribution histogram
plt.figure(figsize=(10, 5))
plt.bar(range(256), intensity_distribution, width=1.0, color='b')
plt.title('Intensity Distribution')
plt.xlabel('Pixel Value')
plt.ylabel('Frequency')
plt.show()
```

```
Number of images: 11
```

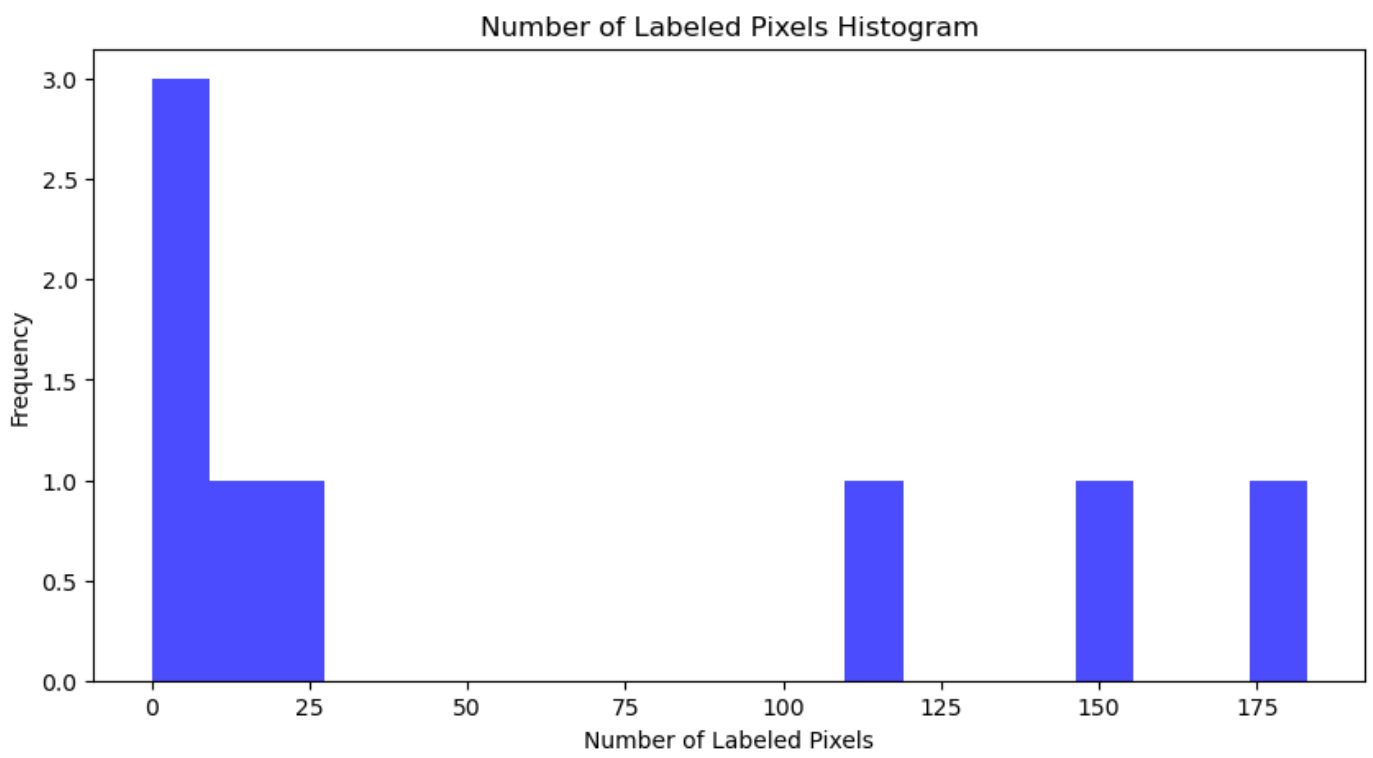
```
Number of pixels: 65276343
Number of labeled pixels: 497
```



In [440…

```python
num_images = len(train_annotations)
num_labeled_pixels = 0

# Lists to store the number of labeled pixels for each image
num_labeled_pixels_per_image = []

# Iterate through each image's annotations
for annotations in train_annotations:
    image_labeled_pixels = 0
    for annotation in annotations:
        points_str = annotation.get('points', '')
        if points_str:
            points = parse_points(points_str)
            image_labeled_pixels += len(points)
    num_labeled_pixels += image_labeled_pixels
    num_labeled_pixels_per_image.append(image_labeled_pixels)

# Display basic statistics
print(f"Number of images: {num_images}")
print(f"Number of labeled pixels: {num_labeled_pixels}")

# Plotting additional statistics
plt.figure(figsize=(10, 5))
plt.hist(num_labeled_pixels_per_image, bins=20, color='b', alpha=0.7)
plt.title('Number of Labeled Pixels Histogram')
plt.xlabel('Number of Labeled Pixels')
plt.ylabel('Frequency')
plt.show()
```

```
Number of images: 8
Number of labeled pixels: 497
```



In [441…
```python
# Calculating the number of "crack" labels in each image
crack_counts = {}

for image_path, annotations in zip(train_image_paths, train_annotations):
    image_name = os.path.basename(image_path)
    crack_count = sum(1 for annotation in annotations if annotation['label'] == 'Crack')
    crack_counts[image_name] = crack_count

# Displaying the number of "crack" labels in each image
for image_name, count in crack_counts.items():
    print(f"Image: {image_name}, Number of 'Crack' labels: {count}")
```

```
Image: c04464a0-0f86-40e2-87d0-6e8d3b9d4066.JPG, Number of 'Crack' labels: 7
Image: 7aaef046-b790-4f97-a77c-19864009ff4d.JPG, Number of 'Crack' labels: 2
Image: 0d1415c4-099c-477f-a290-fea11c9f873f.JPG, Number of 'Crack' labels: 19
Image: IMG_9008.jpg, Number of 'Crack' labels: 0
Image: IMG_9003.jpg, Number of 'Crack' labels: 2
Image: IMG_9007.jpg, Number of 'Crack' labels: 2
Image: IMG_9001.jpg, Number of 'Crack' labels: 18
Image: IMG_9005.jpg, Number of 'Crack' labels: 0
```

Section 7 : Binary Segmentation

In [442…
```python
# Setting masking for the images
mask_dir = 'masks/'
os.makedirs(mask_dir, exist_ok=True)
```

In [443…
```python
# Function to filter annotations and create a binary mask with Crack areas outlined in w
def segment_cracks(image, annotations):
    # Creating an empty binary mask with a black background
    binary_mask = np.zeros_like(image, dtype=np.uint8)

    # Filtering annotations to keep only "Crack" labels
    for annotation in annotations:
        label = annotation['label']
        if label == 'Crack':
            points_str = annotation['points']
            if points_str:
                points = parse_points(points_str)
```

```
                if len(points) >= 2:
                    for i in range(len(points) - 1):
                        cv2.line(binary_mask, points[i], points[i+1], (255, 255, 255), 6

    # Converting the binary mask to grayscale for considring brightness levels
    binary_mask_gray = cv2.cvtColor(binary_mask, cv2.COLOR_BGR2GRAY)

    # Morphological Operations to filter false positives
    kernel = np.ones((5, 5), np.uint8)
    cleaned_mask = cv2.morphologyEx(binary_mask_gray, cv2.MORPH_OPEN, kernel)

    return cleaned_mask

# Function to parse points string into a list of (x, y) coordinates
# To extract the annotated points
def parse_points(points_str):
    points_list = []
    points_str = points_str.split(';')
    for point in points_str:
        x, y = map(float, point.split(','))
        points_list.append((int(x), int(y)))
    return points_list

output_folder = "segmented"
if not os.path.exists(output_folder):
    os.makedirs(output_folder)

# Iterating through the dictionary and process each image
for image_id, image_data in image_annotations.items():
    image_name = image_data['image_name']
    annotations = image_data['annotations']

    image_path = "image/" + image_name
    image = cv2.imread(image_path)

    segmented_mask = segment_cracks(image, annotations)

    output_path = os.path.join(output_folder, f"segmented_mask_{image_id}.png")
    cv2.imwrite(output_path, segmented_mask)
```

Section 7 : Connected Component Analysis

In [444…

```
# Reference: https://pyimagesearch.com/2021/02/22/opencv-connected-component-labeling-an
# Since while annotating we make the big crack as sub-divided into small different crack
# the segmented and the connected results come as the same
def extract_regions(segmented_mask):

    # Applying connected component analysis to the binary mask
    num_labels, labels, stats, centroids = cv2.connectedComponentsWithStats(segmented_ma

    # Creating an empty mask to store individual regions
    region_mask = np.zeros_like(segmented_mask)

    # Iterating through each connected component (excluding background)
    for label in range(1, num_labels):
        # Creating a mask for the current component
        component_mask = (labels == label).astype(np.uint8) * 255

        # Adding the component_mask to the region_mask
        region_mask = cv2.add(region_mask, component_mask)

    return region_mask

# Creating a folder to save the region masks
```

```
if not os.path.exists("region_masks"):
    os.makedirs("region_masks")

# Iterating through the dictionary and process each image
for image_id, image_data in image_annotations.items():
    image_name = image_data['image_name']
    annotations = image_data['annotations']

    image_path = "image/" + image_name
    image = cv2.imread(image_path)

    # Calling the segment_cracks function to process the image
    segmented_mask = segment_cracks(image, annotations)

    # Performing connected component analysis and extract regions
    region_mask = extract_regions(segmented_mask)

    # Saving the region mask in the "region_masks" folder
    cv2.imwrite(f"region_masks/region_mask_{image_id}.png", region_mask)
```

In [445…

```
# Checking values
num_region_pixels = 0
for row in range(region_mask.shape[0]):
    for col in range(region_mask.shape[1]):
        if region_mask[row, col] != 0:
            num_region_pixels += 1
```

In [446…

```
print("Keys in image_annotations:", list(image_annotations.keys()))
#As during annotation we deleted one image which was blur
```

```
Keys in image_annotations: [0, 1, 2, 3, 4, 5, 6, 8, 9, 10, 11]
```

In [447…

```
def get_annotation_for_region(image_id, region_label):
    if image_id in image_annotations:
        annotations = image_annotations[image_id]['annotations']
        for annotation in annotations:
            if annotation['label'] == 'Crack':
                return "Crack"
            elif annotation['label'] == 'no-crack':
                return "no-crack"
    else:
        return "No-Image"


#image_id = 0
#region_label = 0
#annotation_label = get_annotation_for_region(image_id, region_label)
#print(f"Annotation for region {region_label} in image {image_id}: {annotation_label}")
```

In [448…

```
# Reference : https://scikit-image.org/docs/stable/auto_examples/segmentation/plot_regio

def calculate_region_properties(region_mask, image_id):

    # Calculating region properties using OpenCV
    num_labels, labels, stats, centroids = cv2.connectedComponentsWithStats(region_mask,

    # Initializing lists to store feature vectors and labels
    feature_vectors = []
    assigned_labels = []

    # Iterating through each connected component (excluding background)
    for label in range(1, num_labels):
        # Extracting region properties from the stats array
        area = stats[label, cv2.CC_STAT_AREA]
        centroid = centroids[label]
```

```
        left = stats[label, cv2.CC_STAT_LEFT]
        top = stats[label, cv2.CC_STAT_TOP]
        width = stats[label, cv2.CC_STAT_WIDTH]
        height = stats[label, cv2.CC_STAT_HEIGHT]

        # Calculating circularity (example feature)
        circularity = (4.0 * np.pi * area) / (width * height * width * height)

        # Calculating color features (example: mean color)
        region_mean_color = np.mean(region_mask[top:top+height, left:left+width])

        # Creating a feature vector with selected properties
        feature_vector = [area, circularity, region_mean_color]

        # Check if the region has a "Crack" annotation in the image
        region_annotation = get_annotation_for_region(image_id, label)
        print('Image ID ' , image_id)
        print('Region_annotation ' , region_annotation)
        if region_annotation == "Crack":
            print('This image contains crack')
            assigned_label = "Crack"
        else:
            print('This image contains no-crack')
            assigned_label = "no-crack"

        # Appending the feature vector and assigned label to their respective lists
        feature_vectors.append(feature_vector)
        assigned_labels.append(assigned_label)

    return feature_vectors, assigned_labels

# Pathing to the folder containing region masks
region_masks_folder = "region_masks"
```

Section 8 : Feature Engineering

In [449…

```
# Initializing lists to store all feature vectors and labels from all images
all_feature_vectors = []
all_labels = []

# Iterating through region masks in the folder
for filename in os.listdir(region_masks_folder):
    if filename.endswith(".png"):
        # Loading the region mask
        region_mask = cv2.imread(os.path.join(region_masks_folder, filename), cv2.IMREAD
        parts = filename.split('_')
      #  print("filename ",filename)
      #  print("parts " ,parts)
      #  print("length ", len(parts))
        id = parts[2].split(".png")
        image_id = int(id[0])
     #   print("image_id ", id)

        # Calculating region properties and assign labels based on annotations
        if image_id != 7:
            feature_vectors, assigned_labels = calculate_region_properties(region_mask,

        # Extending the lists with feature vectors and labels from this image
        all_feature_vectors.extend(feature_vectors)
        all_labels.extend(assigned_labels)

# Now, all_feature_vectors contains feature vectors for all regions,
# and all_labels contains corresponding labels ("Crack" or "No-Crack").
```

```
Image ID  0
Region_annotation  Crack
This image contains crack
Image ID  0
Region_annotation  Crack
This image contains crack
Image ID  0
Region_annotation  Crack
This image contains crack
Image ID  1
Region_annotation  Crack
This image contains crack
Image ID  1
Region_annotation  Crack
This image contains crack
Image ID  1
Region_annotation  Crack
This image contains crack
Image ID  1
Region_annotation  Crack
This image contains crack
Image ID  1
Region_annotation  Crack
This image contains crack
Image ID  1
Region_annotation  Crack
This image contains crack
Image ID  1
Region_annotation  Crack
This image contains crack
Image ID  10
Region_annotation  Crack
This image contains crack
Image ID  11
Region_annotation  Crack
This image contains crack
Image ID  11
Region_annotation  Crack
This image contains crack
Image ID  11
Region_annotation  Crack
This image contains crack
Image ID  2
Region_annotation  Crack
This image contains crack
Image ID  3
Region_annotation  no-crack
This image contains no-crack
Image ID  3
Region_annotation  no-crack
This image contains no-crack
Image ID  3
Region_annotation  no-crack
This image contains no-crack
Image ID  3
Region_annotation  no-crack
This image contains no-crack
Image ID  3
Region_annotation  no-crack
This image contains no-crack
Image ID  3
Region_annotation  no-crack
This image contains no-crack
Image ID  3
Region_annotation  no-crack
This image contains no-crack
Image ID  3
Region_annotation  no-crack
This image contains no-crack
```

```
Image ID  3
Region_annotation  no-crack
This image contains no-crack
Image ID  3
Region_annotation  no-crack
This image contains no-crack
Image ID  3
Region_annotation  no-crack
This image contains no-crack
Image ID  3
Region_annotation  no-crack
This image contains no-crack
C:\Users\GOTTG\AppData\Local\Temp\ipykernel_15932\1237978453.py:23: RuntimeWarning: over
flow encountered in int_scalars
  circularity = (4.0 * np.pi * area) / (width * height * width * height)
Image ID  3
Region_annotation  no-crack
This image contains no-crack
Image ID  3
Region_annotation  no-crack
This image contains no-crack
Image ID  3
Region_annotation  no-crack
This image contains no-crack
Image ID  3
Region_annotation  no-crack
This image contains no-crack
Image ID  3
Region_annotation  no-crack
This image contains no-crack
Image ID  8
Region_annotation  Crack
This image contains crack
Image ID  8
Region_annotation  Crack
This image contains crack
Image ID  0
Region_annotation  Crack
This image contains crack
Image ID  0
Region_annotation  Crack
This image contains crack
Image ID  0
Region_annotation  Crack
This image contains crack
Image ID  0
Region_annotation  Crack
This image contains crack
Image ID  1
Region_annotation  Crack
This image contains crack
Image ID  1
Region_annotation  Crack
This image contains crack
Image ID  1
Region_annotation  Crack
This image contains crack
Image ID  1
Region_annotation  Crack
This image contains crack
Image ID  1
Region_annotation  Crack
This image contains crack
Image ID  1
Region_annotation  Crack
```

This image contains crack
Image ID 1
Region_annotation Crack
This image contains crack
Image ID 1
Region_annotation Crack
This image contains crack
Image ID 1
Region_annotation Crack
This image contains crack
Image ID 1
Region_annotation Crack
This image contains crack
Image ID 1
Region_annotation Crack
This image contains crack
Image ID 1
Region_annotation Crack
This image contains crack
Image ID 10
Region_annotation Crack
This image contains crack
Image ID 11
Region_annotation Crack
This image contains crack
Image ID 11
Region_annotation Crack
This image contains crack
Image ID 11
Region_annotation Crack
This image contains crack
Image ID 11
Region_annotation Crack
This image contains crack
Image ID 11
Region_annotation Crack
This image contains crack
Image ID 2
Region_annotation Crack
This image contains crack
Image ID 3
Region_annotation no-crack
This image contains no-crack
Image ID 3
Region_annotation no-crack
This image contains no-crack
Image ID 3
Region_annotation no-crack
This image contains no-crack
Image ID 3
Region_annotation no-crack
This image contains no-crack
Image ID 3
Region_annotation no-crack
This image contains no-crack
Image ID 3
Region_annotation no-crack
This image contains no-crack
Image ID 3
Region_annotation no-crack
This image contains no-crack
Image ID 3
Region_annotation no-crack
This image contains no-crack
Image ID 3
Region_annotation no-crack

```
This image contains no-crack
Image ID  3
Region_annotation  no-crack
This image contains no-crack
Image ID  3
Region_annotation  no-crack
This image contains no-crack
Image ID  3
Region_annotation  no-crack
This image contains no-crack
Image ID  3
Region_annotation  no-crack
This image contains no-crack
Image ID  3
Region_annotation  no-crack
This image contains no-crack
Image ID  3
Region_annotation  no-crack
This image contains no-crack
Image ID  3
Region_annotation  no-crack
This image contains no-crack
Image ID  3
Region_annotation  no-crack
This image contains no-crack
Image ID  8
Region_annotation  Crack
This image contains crack
Image ID  8
Region_annotation  Crack
This image contains crack
```

In [450…

```
print(all_feature_vectors)
```

```
[[1058, 0.0038060713806183006, 144.34991974317816], [6942, -0.0001532910516255207, 29.00
0819134993446], [8106, -5.551700193369814e-05, 41.67399193548387], [671, 0.0097912114974
67445, 184.38038793103448], [3417, 0.001077074360431106, 138.00047513462147], [5044, 0.0
0053105811627083 07, 117.73180778032037], [4229, 0.00031152628081339637, 82.5660362912487
5], [3187, 0.0008855383015262406, 120.8453531598513], [1353, 0.0041833731539617485, 171.
13839285714286], [3546, 4.8153756891268924e-05, 29.724852071005916], [1470, 0.0020525072
003453313, 124.95], [430, 0.012868511288709906, 169.21296296296296], [9054, -5.356296299
795059e-05, 49.55292754120879], [4595, 0.00013305568831285477, 56.24639976958525], [526,
0.015266652523426717, 203.8449848024316], [706, 0.009015529125887722, 181.4818548387096
7], [985, 0.001424266095720644, 85.20183175033921], [331, 0.027346934078585707, 216.4230
7692307693], [631, 0.004877742319206851, 126.2], [996, 0.0017517466068454612, 95.0168350
1683502], [653, 0.005319610733283378, 134.07004830917873], [1403, 0.0027763003861088938,
141.9702380952381], [1553, 0.0004245432419683912, 58.40929203539823], [640, 0.0110016445
30884539, 190.87719298245614], [1382, 0.0014406499684311536, 101.50057603686636], [1750,
0.0005647792512925437, 71.51442307692308], [2428, 0.00018495154034689463, 48.20460915602
616], [2079, 0.00023823473702120056, 50.625], [1581, 0.0007564043094844251, 78.664390243
90245], [2375, 0.0001066560359494815, 36.204268292682926], [1488, 0.002414273028259126,
136.34207689543658], [2228, 0.0001517070505586819, 41.82112624217887], [3001, 0.00046692
419215391877, 85.15132969845332], [604, 0.003635056022352673, 106.58823529411765], [102
3, 0.0004374482496839167, 48.12119535141118], [2998, 5.0668135334883494e-05, 28.03615960
0997507], [4687, -2.800387725150017e-05, 25.529413021189338], [391, 0.01088048050908454
8, 148.370535714285 72], [627, 0.005508267787275311, 133.68311036789297], [1365, 0.001644
134985344465, 107.76315789473684], [160, 0.043094549431958756, 188.88888888888889], [48
1, 0.006477399561816847, 126.972049689441], [369, 0.01590188874039278, 174.25], [224, 0.
025848181979949078, 173.0909090909091], [396, 0.013114255047452755, 163.92857142857142],
[1287, 0.002170016903578507, 120.21428571428571], [2480, 0.00023495398271959457, 54.9101
32847095596], [1891, 0.0007673103486365318, 86.64959568733154], [787, 0.0041700681706445
73, 130.31493506493507], [2045, 3.0472630970633868e-05, 17.957128099173552], [858, 0.001
841053546055626 5, 90.4090909090909], [244, 0.013532741463807455, 130.71428571428572], [4
276, 4.436996362382731e-05, 31.332758620689656], [676, 0.002029295541572857, 84.25219941
348973], [376, 0.00523540759113468, 100.92631578947369], [2669, 9.15916985909951e-05, 3
5.566210284280935], [294, 0.019981140944411016, 174.34883720930233], [405, 0.01050621800
```

```
026313, 148.38362068965517], [549, 0.001086378411955654, 55.55357142857143], [174, 0.039
93258249138899, 189.6153846153846], [343, 0.0044245558221676085, 88.61702127659575], [55
6, 0.001401222845603955, 63.49305866547246], [355, 0.004943004507587265, 95.289473684210
52], [809, 0.0026142346365898876, 104.61206896551724], [857, 0.00028904245418066453, 35.
801933158584 53], [360, 0.013309914181304196, 157.46140651801028], [785, 0.00116012418738
29774, 68.64711934156378], [981, 0.00039763116737106987, 44.927262931034484], [1387, 0.0
0012349615715315451, 29.771464646464647], [1189, 0.00016212472504853576, 31.5828125], [9
01, 0.0005648855510193646, 51.31896359169087], [1327, 6.782482317448549e-05, 21.58067602
0408163], [854, 0.0021716448847483812, 97.96221322537112], [1233, 9.684128766418145e-05,
24.856905684243813], [1713, 0.00033676759141438483, 54.636022514071 29]]
```

In [451…
```
binary_labels = [1 if label in [1, 2, 3, 4, 5, 6] else 0 for label in all_labels]
```

In [452…
```
print(all_labels)
```

```
['Crack', 'Crack', 'Crack', 'Crack', 'Crack', 'Crack', 'Crack', 'Crack', 'Crack', 'Crac
k', 'Crack', 'Crack', 'Crack', 'Crack', 'no-crack', 'no-crack', 'no-crack', 'no-crack',
'no-crack', 'no-crack', 'no-crack', 'no-crack', 'no-crack', 'no-crack', 'no-crack', 'no-
crack', 'no-crack', 'no-crack', 'no-crack', 'no-crack', 'no-crack', 'Crack', 'Crack', 'C
rack', 'Crack', 'Crack', 'Crack', 'Crack', 'Crack', 'Crack', 'Crack', 'Crack', 'Crack',
'Crack', 'Crack', 'Crack', 'Crack', 'Crack', 'Crack', 'Crack', 'Crack', 'Crack', 'Crac
k', 'Crack', 'Crack', 'Crack', 'no-crack', 'no-crack', 'no-crack', 'no-crack', 'no-crac
k', 'no-crack', 'no-crack', 'no-crack', 'no-crack', 'no-crack', 'no-crack', 'no-crack',
'no-crack', 'no-crack', 'no-crack', 'no-crack', 'no-crack', 'Crack', 'Crack']
```

Section 9 : Classifier - Decision Tree

In [453…
```
X_train, X_test, y_train, y_test = train_test_split(
    all_feature_vectors, all_labels, test_size=0.2, random_state=42
)
```

In [454…
```
clf = DecisionTreeClassifier(random_state=42)
```

In [455…
```
clfRF = RandomForestClassifier(random_state=42)
```

In [456…
```
clf.fit(X_train, y_train)
```

Out[456]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier(random_state=42)
```

In [457…
```
clfRF.fit(X_train, y_train)
```

Out[457]:
```
▾ RandomForestClassifier
RandomForestClassifier(random_state=42)
```

In [458…
```
y_pred = clf.predict(X_test)
```

In [459…
```
y_predRF = clfRF.predict(X_test)
```

In [460…
```
accuracy = accuracy_score(y_test, y_predRF)
report = classification_report(y_test, y_predRF, zero_division=1)
```

In [472…
```
print(f"Accuracy: {accuracy:.2f}")
print("Classification Report:")
print(report)
```

```
Accuracy: 0.60
Classification Report:
```

```
              precision    recall  f1-score   support

       Crack       0.71      0.56      0.63         9
    no-crack       0.50      0.67      0.57         6

    accuracy                           0.60        15
   macro avg       0.61      0.61      0.60        15
weighted avg       0.63      0.60      0.60        15
```

In [462…
```
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred, zero_division=1)
```

In [471…
```
print(f"Accuracy: {accuracy:.2f}")
print("Classification Report:")
print(report)
```

```
Accuracy: 0.60
Classification Report:
              precision    recall  f1-score   support

       Crack       0.71      0.56      0.63         9
    no-crack       0.50      0.67      0.57         6

    accuracy                           0.60        15
   macro avg       0.61      0.61      0.60        15
weighted avg       0.63      0.60      0.60        15
```

Section 10 : Crack Analytics

In [464…
```
def calculate_iou(gt_mask, predicted_mask):

    # Computing the intersection (logical AND) between the ground truth and predicted ma
    intersection = np.logical_and(gt_mask, predicted_mask)

    # Computing the union (logical OR) between the ground truth and predicted masks
    union = np.logical_or(gt_mask, predicted_mask)

    # Calculating IoU
    iou = np.sum(intersection) / np.sum(union)

    return iou
```

In [465…
```
# Reference : https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-objec
# Reference : https://towardsdatascience.com/intersection-over-union-iou-calculation-for

iou_scores = []

# Iterating through the images and calculating the IoU scores
for image_id in range(11):  #Keeping 11 here as the test scenario
    ground_truth_mask = cv2.imread(f"segmented/segmented_mask_{image_id}.png", cv2.IMREA
    predicted_mask = cv2.imread(f"region_masks/region_mask_{image_id}.png", cv2.IMREAD_G

    # Keeping the masks as binary (0 and 255)
    _, ground_truth_mask = cv2.threshold(ground_truth_mask, 128, 255, cv2.THRESH_BINARY)
    _, predicted_mask = cv2.threshold(predicted_mask, 128, 255, cv2.THRESH_BINARY)

    # Calculating IoU score for the current image
    iou_score = calculate_iou(ground_truth_mask, predicted_mask)

    iou_scores.append(iou_score)

# Looping IoU scores for all images
```

```
for image_id, iou_score in enumerate(iou_scores):
    print(f"Image {image_id}: IoU Score = {iou_score}")
print('Images without crack and only no-crack labels show nan as count')
```

```
C:\Users\GOTTG\AppData\Local\Temp\ipykernel_15932\1155394533.py:10: RuntimeWarning: inva
lid value encountered in long_scalars
  iou = np.sum(intersection) / np.sum(union)
Image 0: IoU Score = 1.0
Image 1: IoU Score = 1.0
Image 2: IoU Score = 1.0
Image 3: IoU Score = 1.0
Image 4: IoU Score = nan
Image 5: IoU Score = nan
Image 6: IoU Score = nan
Image 7: IoU Score = 1.0
Image 8: IoU Score = 1.0
Image 9: IoU Score = nan
Image 10: IoU Score = 1.0
Images without crack and only no-crack labels show nan as count
```

Section 11 : Thinning

In [466…

```
# Reference : https://scikit-image.org/docs/stable/auto_examples/edges/plot_skeleton.htm

# We are saving the images in this folder for preview
thinned_folder = "thinned_images"
if not os.path.exists(thinned_folder):
    os.makedirs(thinned_folder)

# Iterating through the images and thinning each segmented mask
for image_id in range(11):
    segmented_mask = cv2.imread(f"segmented/segmented_mask_{image_id}.png", cv2.IMREAD_G


    _, segmented_mask = cv2.threshold(segmented_mask, 128, 255, cv2.THRESH_BINARY)

    # Performing thinning using scikit-image
    thinned_mask = skeletonize(segmented_mask / 255).astype(np.uint8) * 255

    # Saving the thinned mask to the "thinned_images" folder
    output_path = os.path.join(thinned_folder, f"thinned_mask_{image_id}.png")
    cv2.imwrite(output_path, thinned_mask)

print("Thinning completed and thinned masks saved to the 'thinned_images' folder.")
```

```
Thinning completed and thinned masks saved to the 'thinned_images' folder.
```

Section 12 : Computing Crack Lengths

In [467…

```
# Reference : https://www.researchgate.net/publication/354336108_Crack_Length_Measuremen

def compute_crack_length(thinned_mask):

    # Finding contours in the thinned mask
    contours, _ = cv2.findContours(thinned_mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIM

    # Initializing a variable to accumulate the total length of cracks
    total_length = 0.0

    # Iterating through each contour (crack) and computing its length
    for contour in contours:
        arc_length = cv2.arcLength(contour, closed=True)
        total_length += arc_length
```

```
    return total_length
```

In [468…
```
crack_lengths = []
```

In [470…
```
# Reference : https://www.researchgate.net/publication/354336108_Crack_Length_Measuremen

for image_id in range(0, 8):

    # Loading the thinned crack mask for the current image
    thinned_mask = cv2.imread(f"thinned_images/thinned_mask_{image_id}.png", cv2.IMREAD_


    _, thinned_mask = cv2.threshold(thinned_mask, 128, 255, cv2.THRESH_BINARY)

    # Computing the length of cracks in the current image
    length = compute_crack_length(thinned_mask)


    crack_lengths.append(length)

    print(f"Crack length for image {image_id}: {length} pixels")
print('Images without crack and only no-crack labels show 0 as count')
```

```
Crack length for image 0: 2803.1109426021576 pixels
Crack length for image 1: 3105.7585709095 pixels
Crack length for image 2: 816.9848430156708 pixels
Crack length for image 3: 3595.8014137744904 pixels
Crack length for image 4: 0.0 pixels
Crack length for image 5: 0.0 pixels
Crack length for image 6: 0.0 pixels
Crack length for image 7: 3480.905764102936 pixels
Images without crack and only no-crack labels show 0 as count
```

References

https://www.mdpi.com/2076-3417/13/15/8950