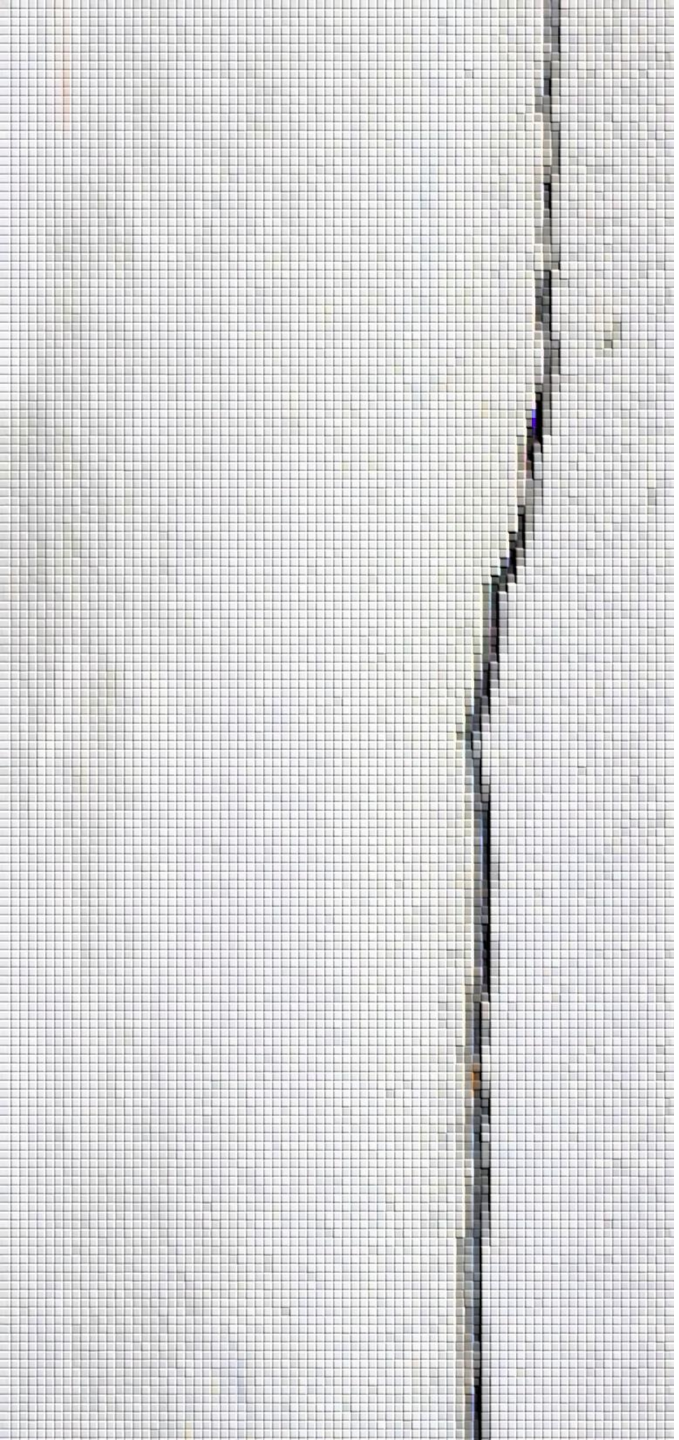
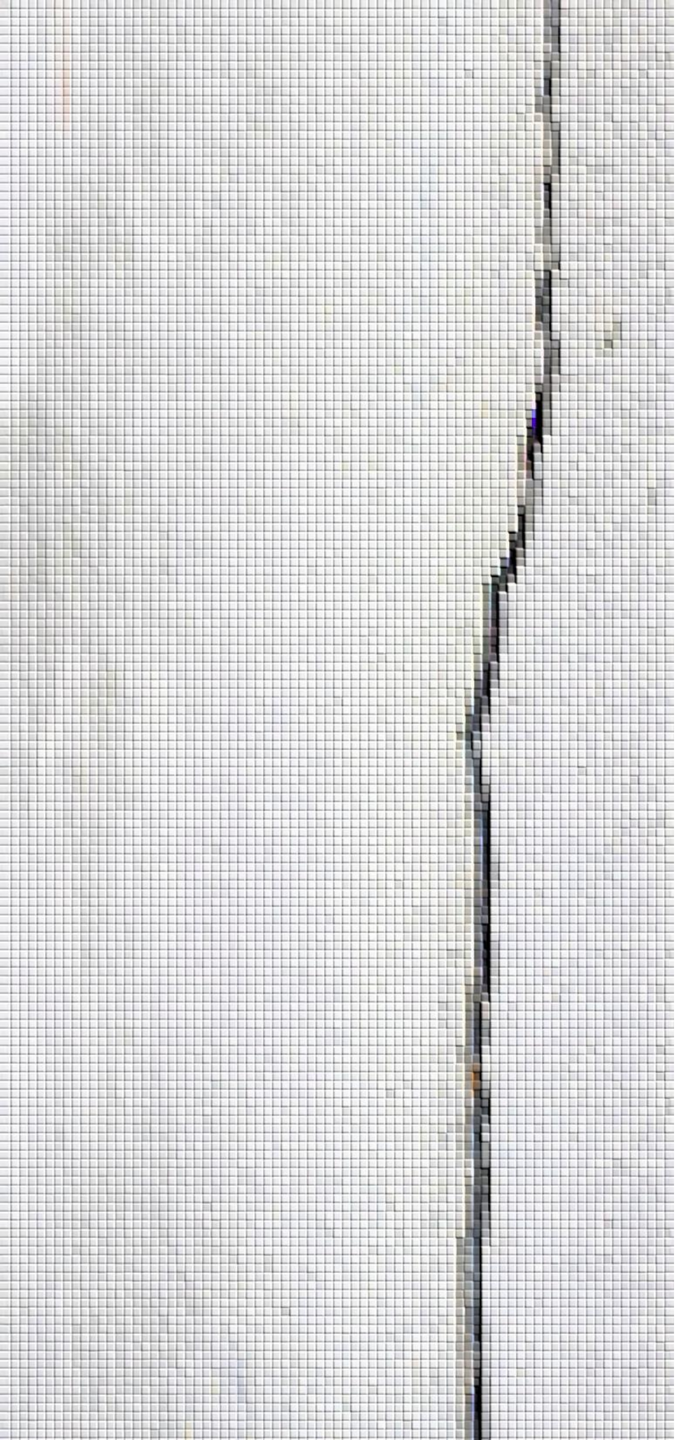
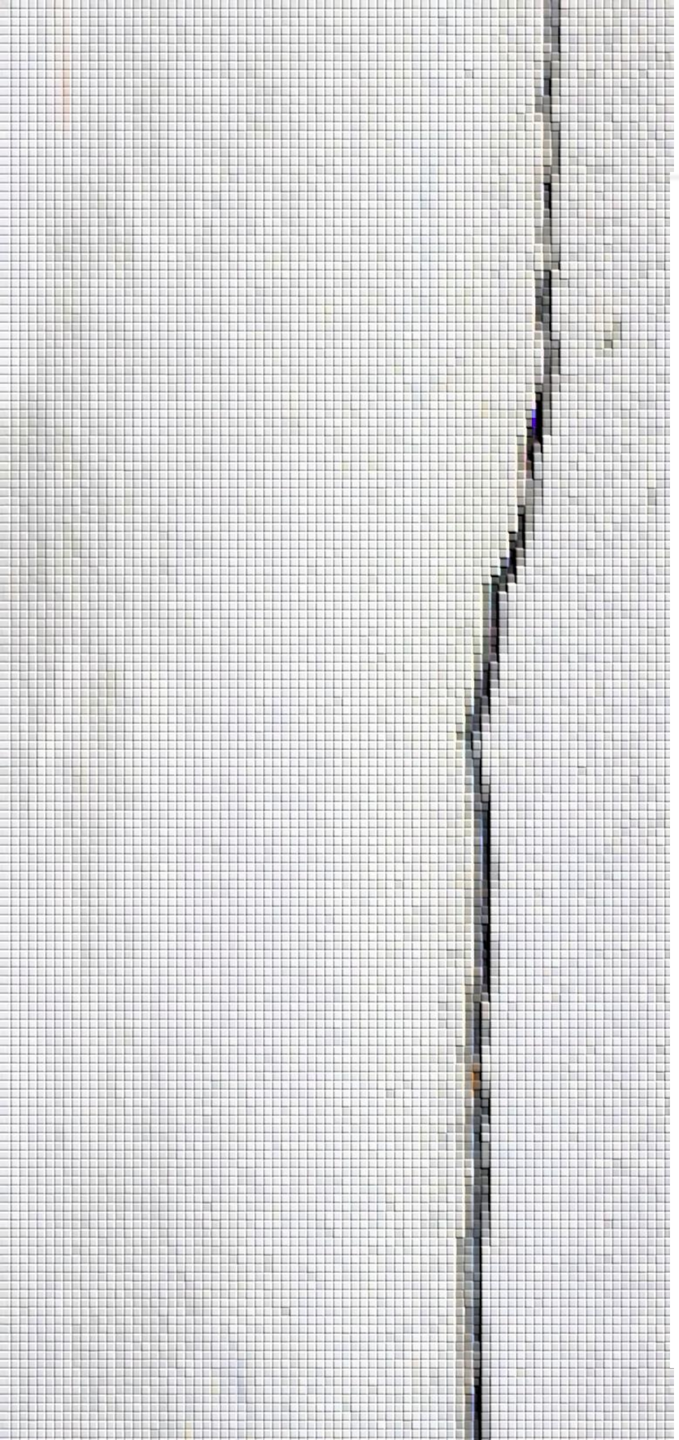


Crack Detection in Concrete surfaces



Problem Statement

Build a **classifier model** that can **reliably and accurately detect cracks** in typical **concrete surfaces**



Why?

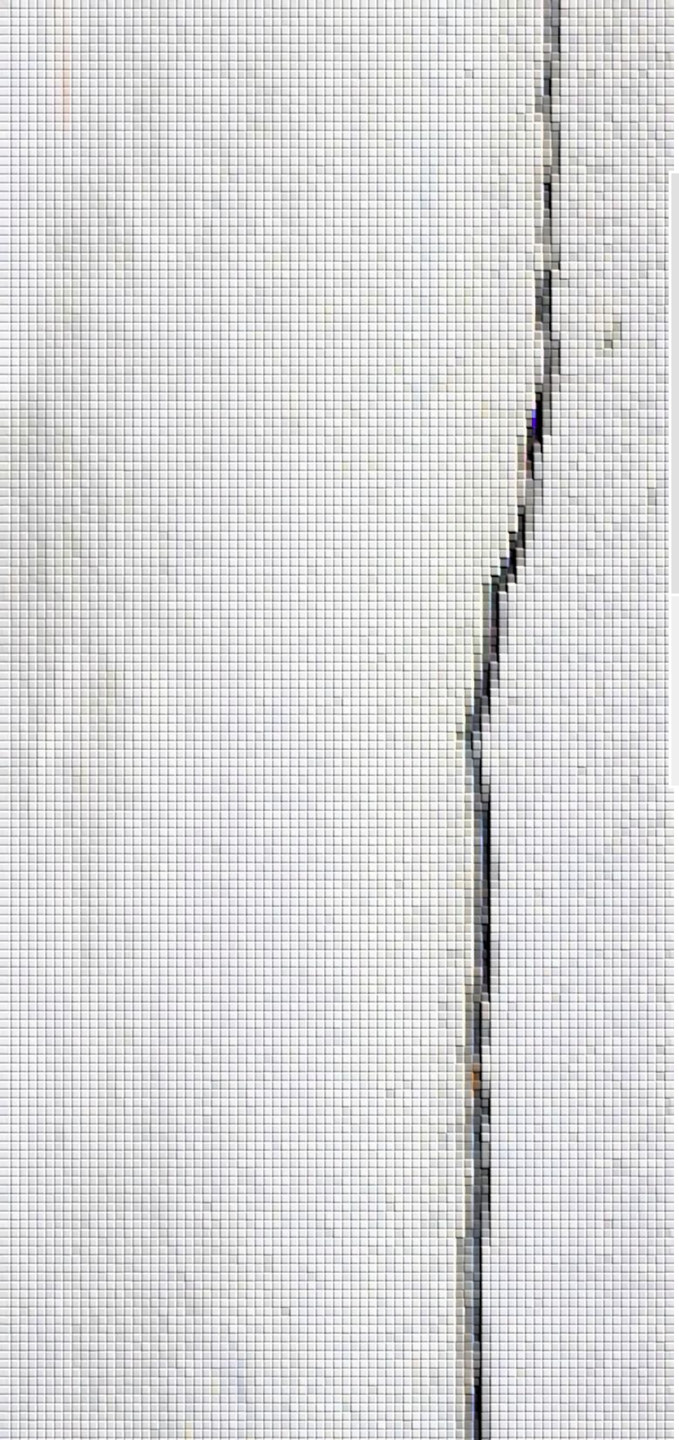
'Tragedy waiting to happen': Italian bridge that collapsed had been riddled with structural problems for decades

Genoa's Morandi motorway bridge had required constant maintenance for cracks and other woes, as a result of 'failed' construction techniques employed in the 1960s

Agence France-Presse
Published: 12:17pm, 15 Aug, 2018

Why?

Downtown line is 44 km long

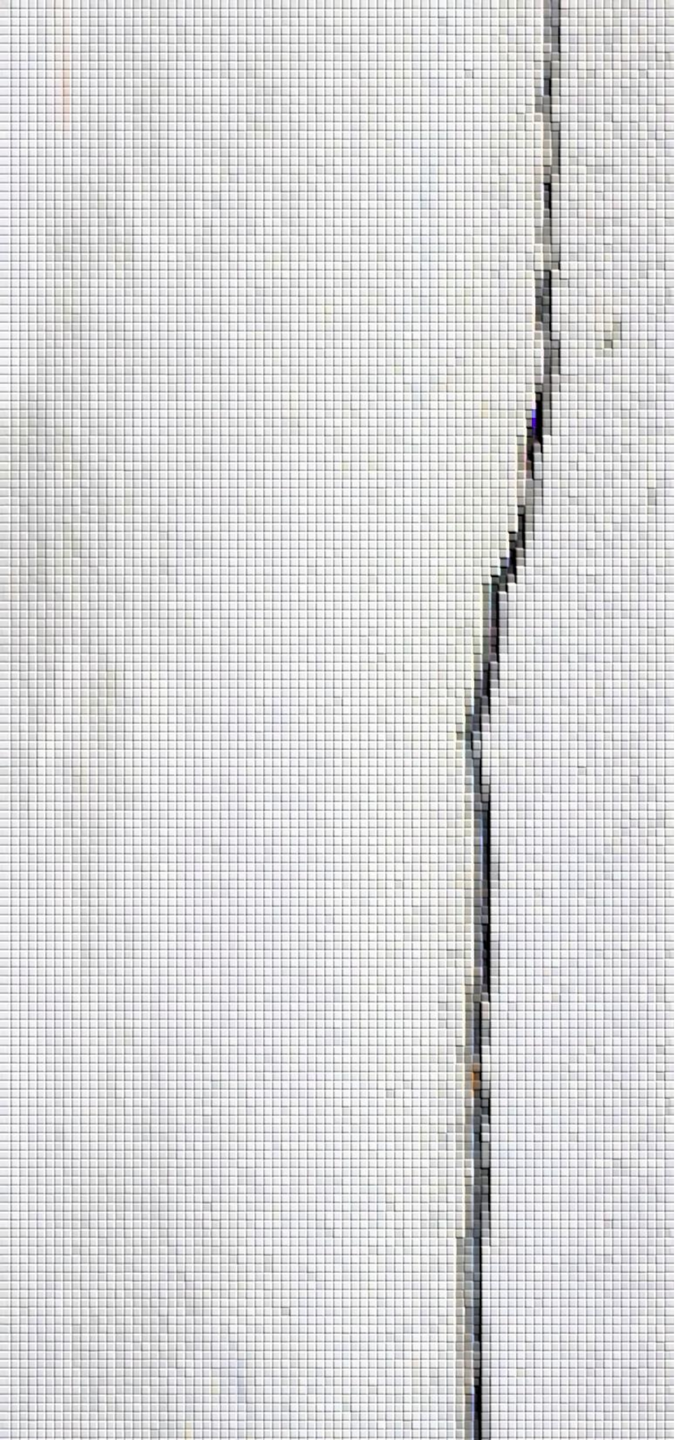


Dataset

Source	https://data.mendeley.com/datasets/5y9wdsg2zt/2 2018 – Özgenel, Ç.F., Gönenç Sorguç, A. “Performance Comparison of Pretrained Convolutional Neural Networks on Crack Detection in Buildings”, ISARC 2018, Berlin. Lei Zhang , Fan Yang , Yimin Daniel Zhang, and Y. J. Z., Zhang, L., Yang, F., Zhang, Y. D., & Zhu, Y. J. (2016). Road Crack Detection Using Deep Convolutional Neural Network. In 2016 IEEE International Conference on Image Processing (ICIP). http://doi.org/10.1109/ICIP.2016.7533052
Format	**RGB** Images of size 227 x 227 concrete surfaces: • 20,000 **positive class** images (surfaces **with cracks**) • 20,000 **negative class** images (surfaces **without cracks**)

Class 0 (Negative class)

Class 1 (Positive class)



Data preprocessing

Import image data as **array**

```
In [10]:  im.shape
Out[10]: (227, 227)
```

Shuffle data and divide to **train & test** sets (0.75 train, 0.25 test)

```
# Shuffle the data
random.shuffle(data)
```

```
# Split data into train and test set
train = data[0:int(len(data)*0.75)]
test = data[int((len(data)*0.75)): ]
```

```
# Check len of train and test
print(len(train))
print(len(test))
```

```
30000
10000
```

Save data in **h5py** format

Model Construction – Deep Learning

- Convolutional Neural Network – Model 1

```
model1 = Sequential()

model1.add(Conv2D(filters=8,
                  kernel_size=3,
                  activation='relu',
                  input_shape=(227, 227, 1)))
model1.add(MaxPooling2D(pool_size = (2,2)))
model1.add(Dropout(0.5))

model1.add(Conv2D(16, 3, activation='relu'))
model1.add(MaxPooling2D(pool_size = (2,2)))
model1.add(Dropout(0.5))

model1.add(Flatten())
model1.add(Dense(230, activation='relu'))

model1.add(Dense(1, activation='sigmoid'))
```

```
model1.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 225, 225, 8)       80
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 112, 112, 8)       0
_________________________________________________________________
dropout_1 (Dropout)          (None, 112, 112, 8)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 110, 110, 16)      1168
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 55, 55, 16)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 55, 55, 16)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 48400)             0
_________________________________________________________________
dense_1 (Dense)              (None, 230)               11132230
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 231
=================================================================
Total params: 11,133,709
Trainable params: 11,133,709
Non-trainable params: 0
```

```
Epoch 5/5
30000/30000 [==============================] - 1119s 37ms/step - loss: 0.6932 - accuracy: 0.4977 - val_loss: 0.6931 - val_ac
curacy: 0.5024
```

Model Construction – Deep Learning

- Convolutional Neural Network – Model 2

```python
# Double no. of filters from model 1
# No dropout
model2 = Sequential()

model2.add(Conv2D(filters=16,
                  kernel_size=3,
                  activation='relu',
                  input_shape=(227, 227, 1)))
model2.add(MaxPooling2D(pool_size = (2,2)))

model2.add(Conv2D(32, 3, activation='relu'))
model2.add(MaxPooling2D(pool_size = (2,2)))

model2.add(Flatten())
model2.add(Dense(230, activation='relu'))

model2.add(Dense(1, activation='sigmoid'))
```

```python
model2.compile(loss='binary_crossentropy',
               optimizer='adam',
               metrics=['accuracy'])
```

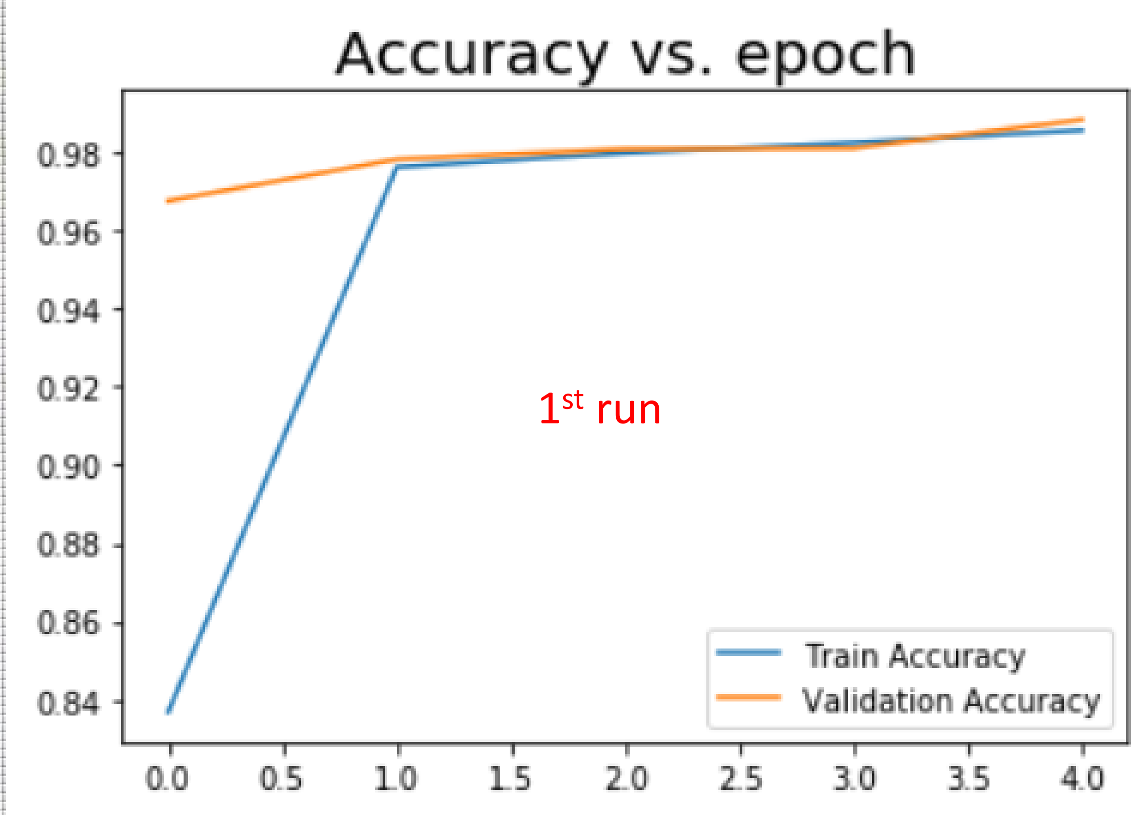
```python
history2 = model2.fit(X_train,
           y_train,
           batch_size=300,
           validation_data=(X_test, y_test),
           epochs=5)
```

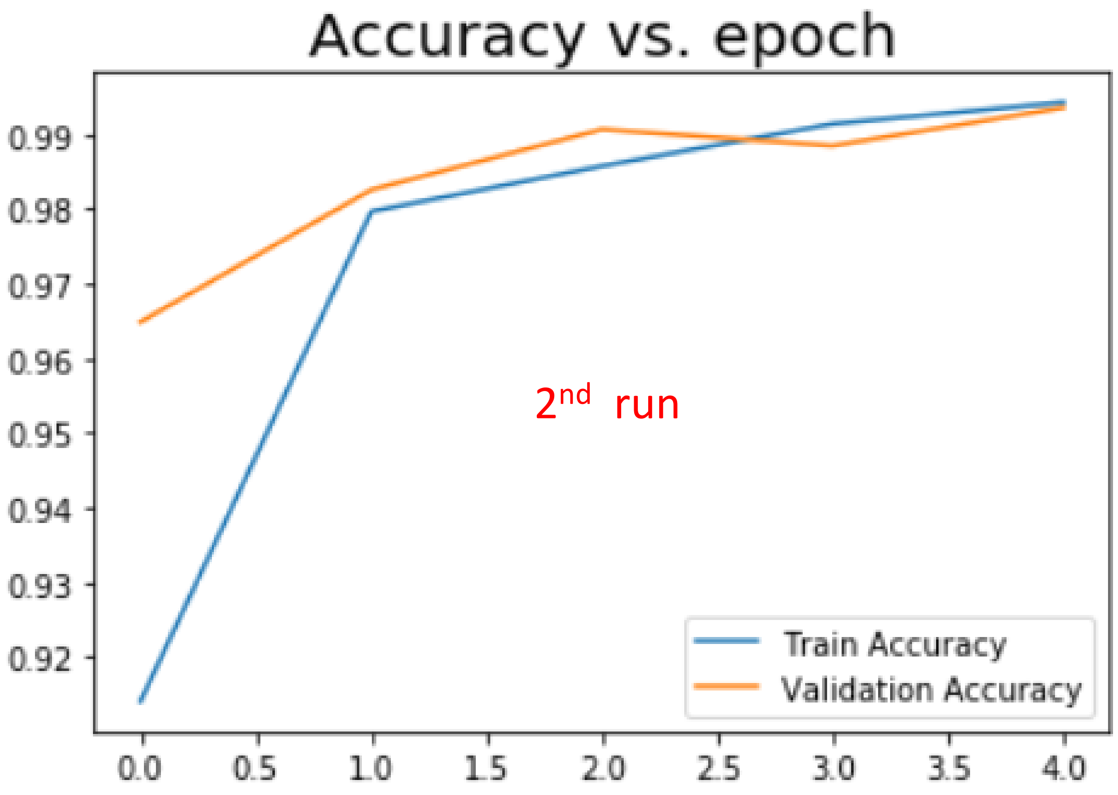
```python
model2.summary()
```

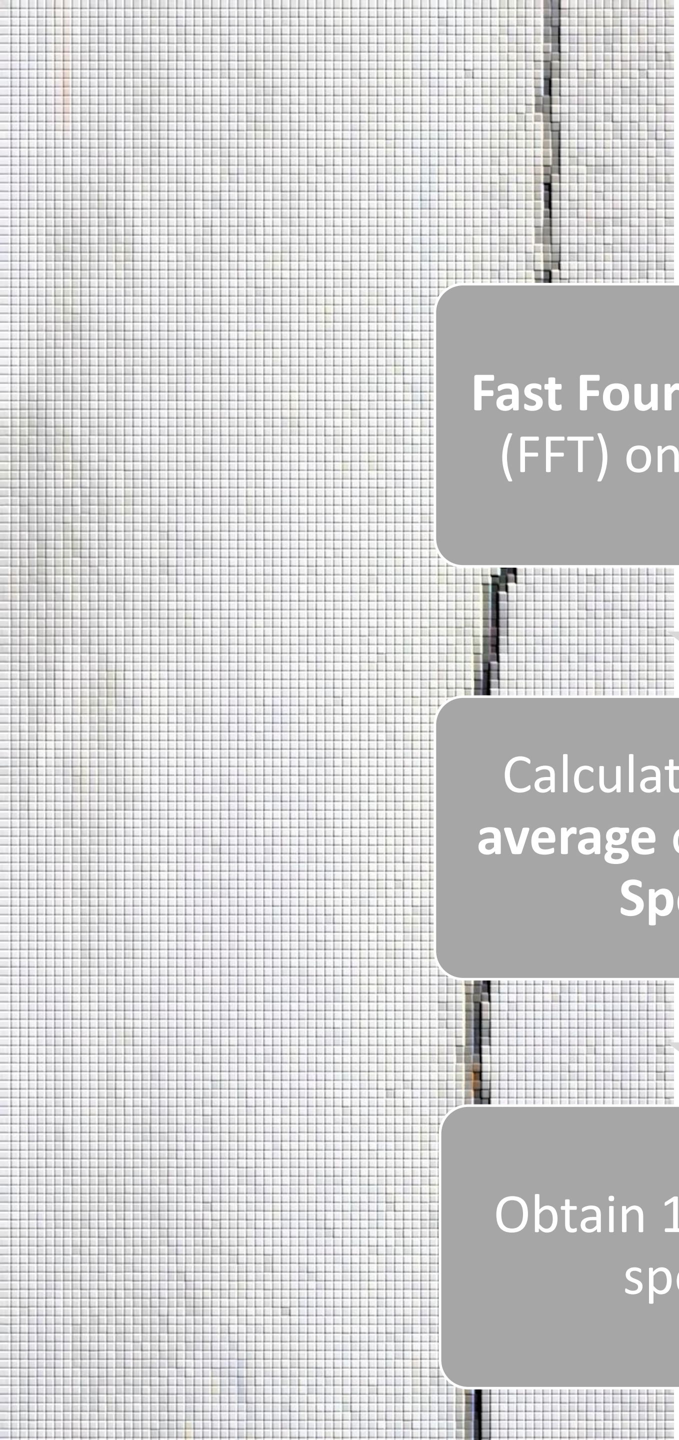
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 225, 225, 16)      160
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 112, 112, 16)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 110, 110, 32)      4640
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 55, 55, 32)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 96800)             0
_________________________________________________________________
dense_1 (Dense)              (None, 230)               22264230
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 231
=================================================================
Total params: 22,269,261
Trainable params: 22,269,261
Non-trainable params: 0
```

Model Construction – Deep Learning

- Convolutional Neural Network – Model 2







Model Construction – Machine Learning

- Pre-processing

Fast Fourier Transform (FFT) on image array

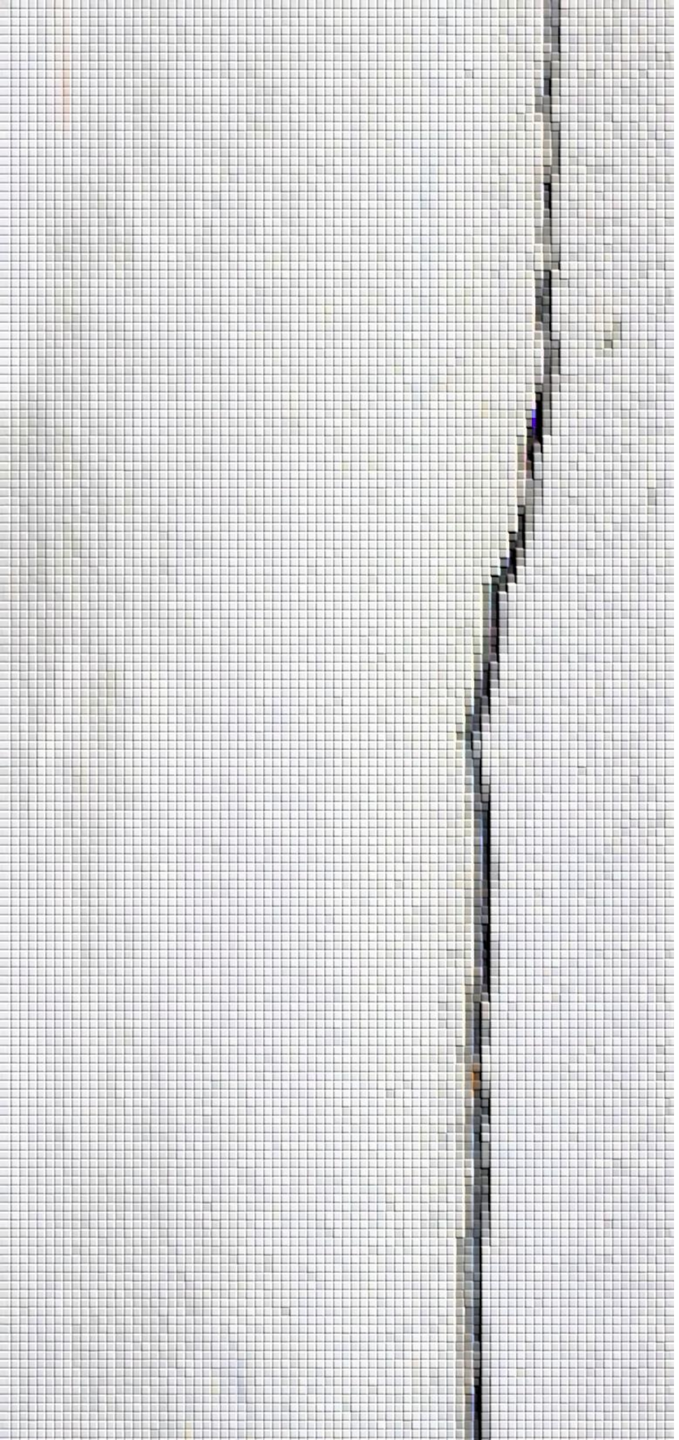
↓

Calculate **Azimuthal average** of **Magnitude Spectrum**

↓

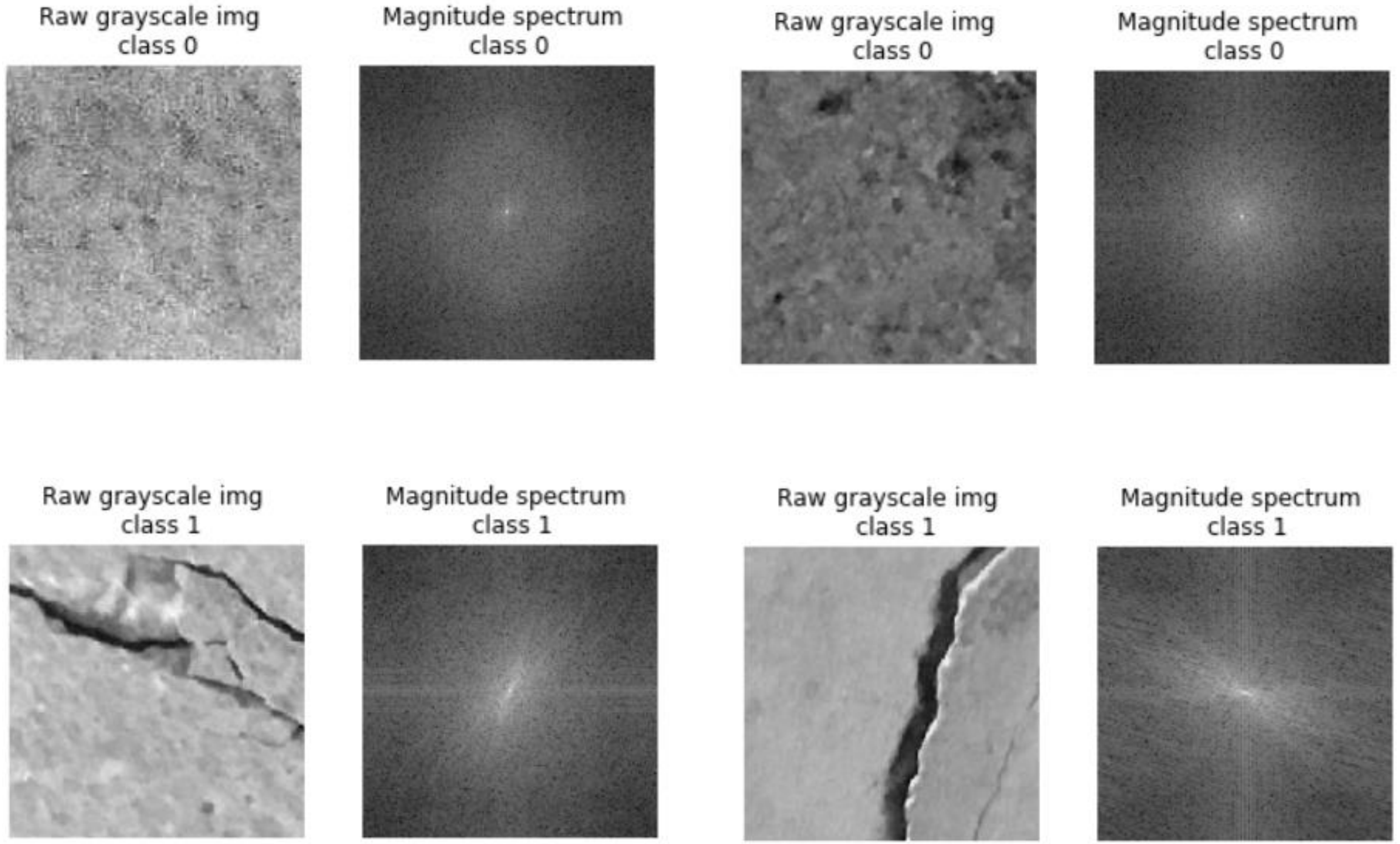
Obtain 1D amplitude spectrum

```python
# Define function to apply Direct Fourier Transfrom on image array
def dff_img(img_array, filename):
    psd1D_array = np.empty([img_array.shape[0], 158])
    for i, img in enumerate(img_array):
        # use numpy library to apply fourier transform
        f = np.fft.fft2(img.reshape(227,227))
        # shift areas of low frequency (by default at the top left corner) to the center
        # areas of low frequency indicate that
        fshift = np.fft.fftshift(f)
        magnitude_spectrum = 20*np.log(np.abs(fshift))
        # Calc the 1D amplitude spectrum from the magnitude spectrum
        # This converts the image data into a single row of numbers, making modelling with ML possible
        psd1D = radialProfile.azimuthalAverage(magnitude_spectrum)
        psd1D_array[i,:] = psd1D

    # Code below was to play around with high pass filtering - not used
    #rows, cols = img.shape
    #crow, ccol = rows/2, cols/2
    #fshift[crow-30:crow+30, ccol-30:ccol+30] = 0
    #f_ishift = np.fft.ifftshift(fshift)
    #img_back = np.fft.ifft2(f_ishift)
    #X_train_dft.append(img_back)

    pd.DataFrame(psd1D_array).to_csv(f'../dataset/{filename}.csv', index=False)
```



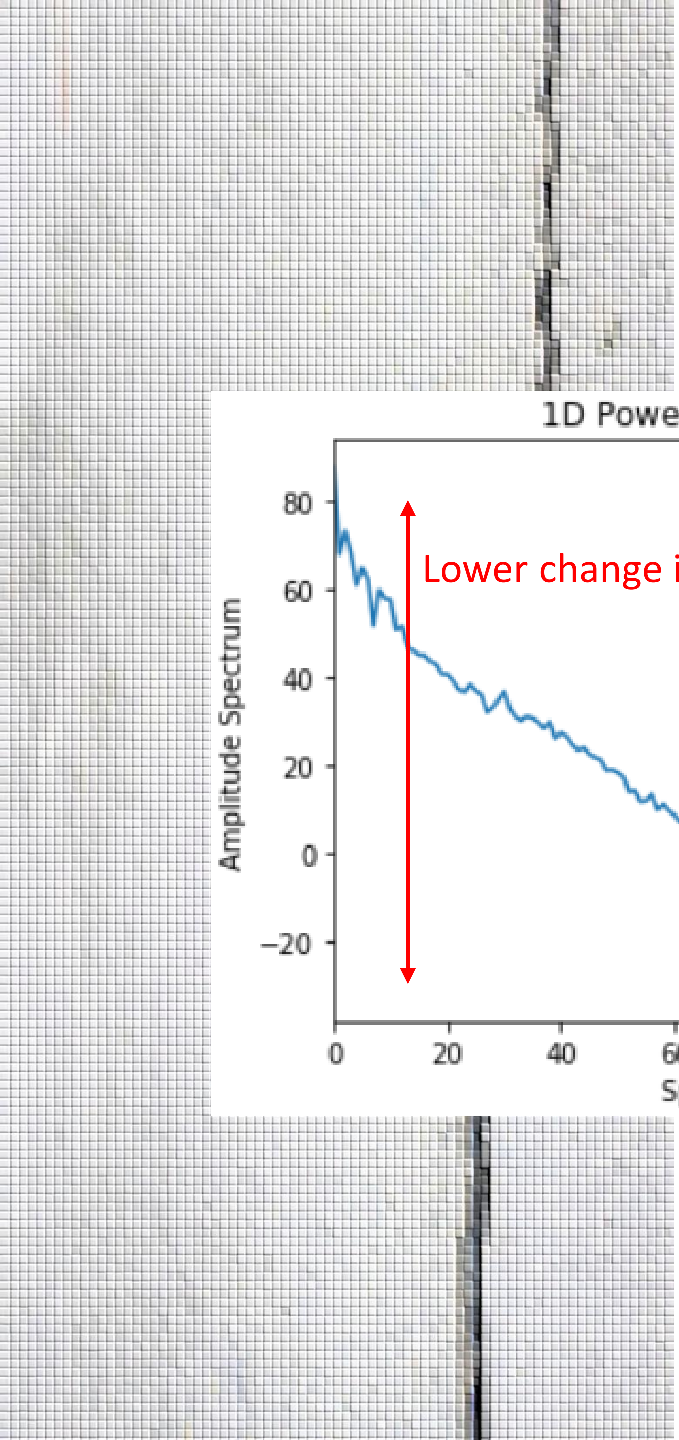
Model Construction – Machine Learning

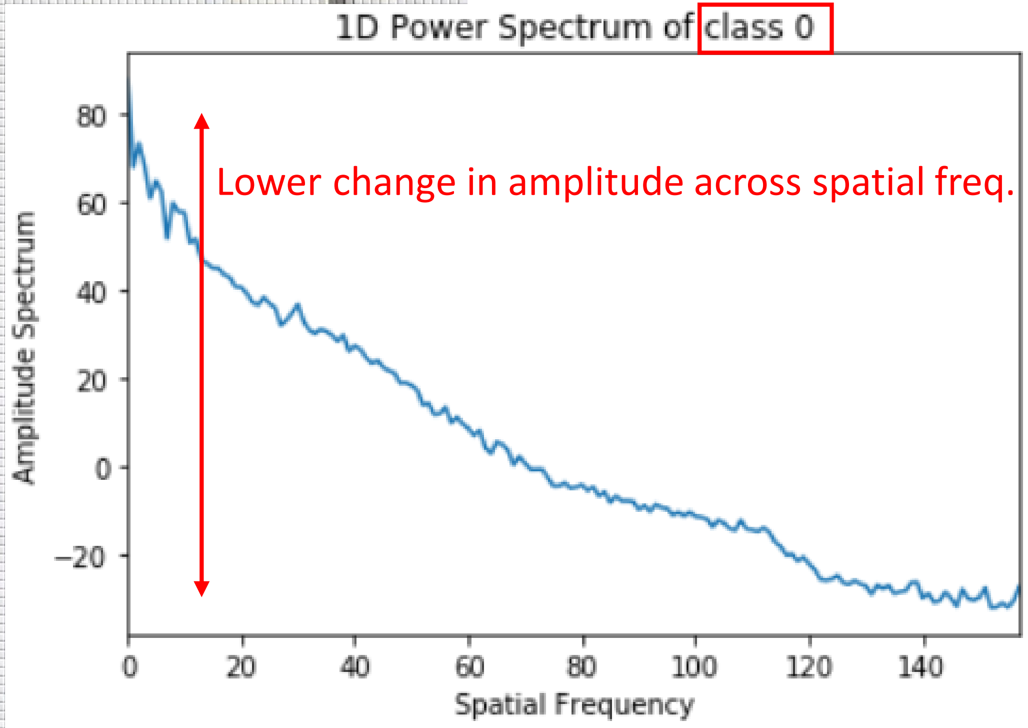
- Visualization of Magnitude Spectrum

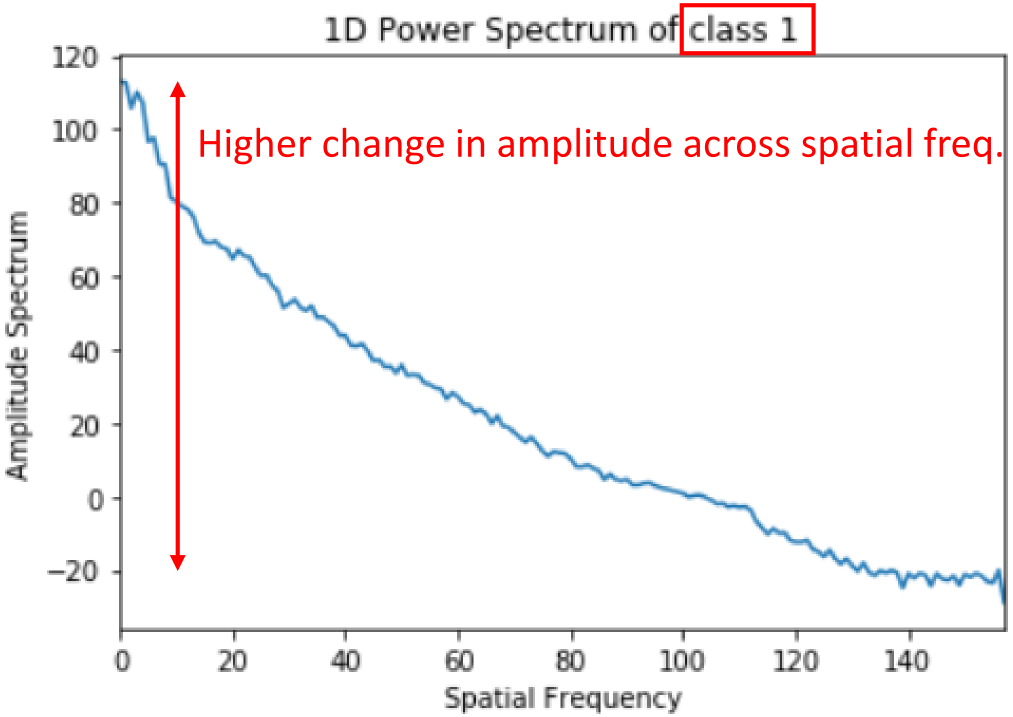


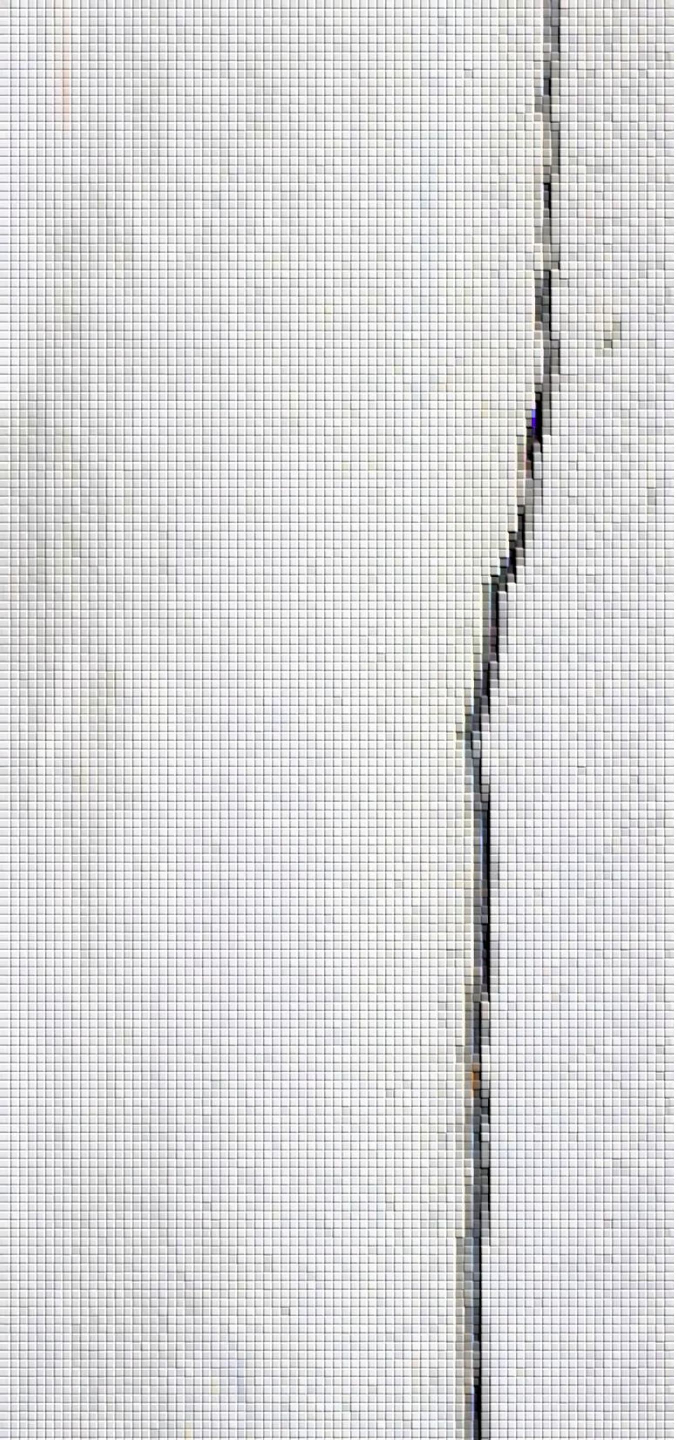
Model Construction – Machine Learning

- Visualization of 1D Amplitude spectrum





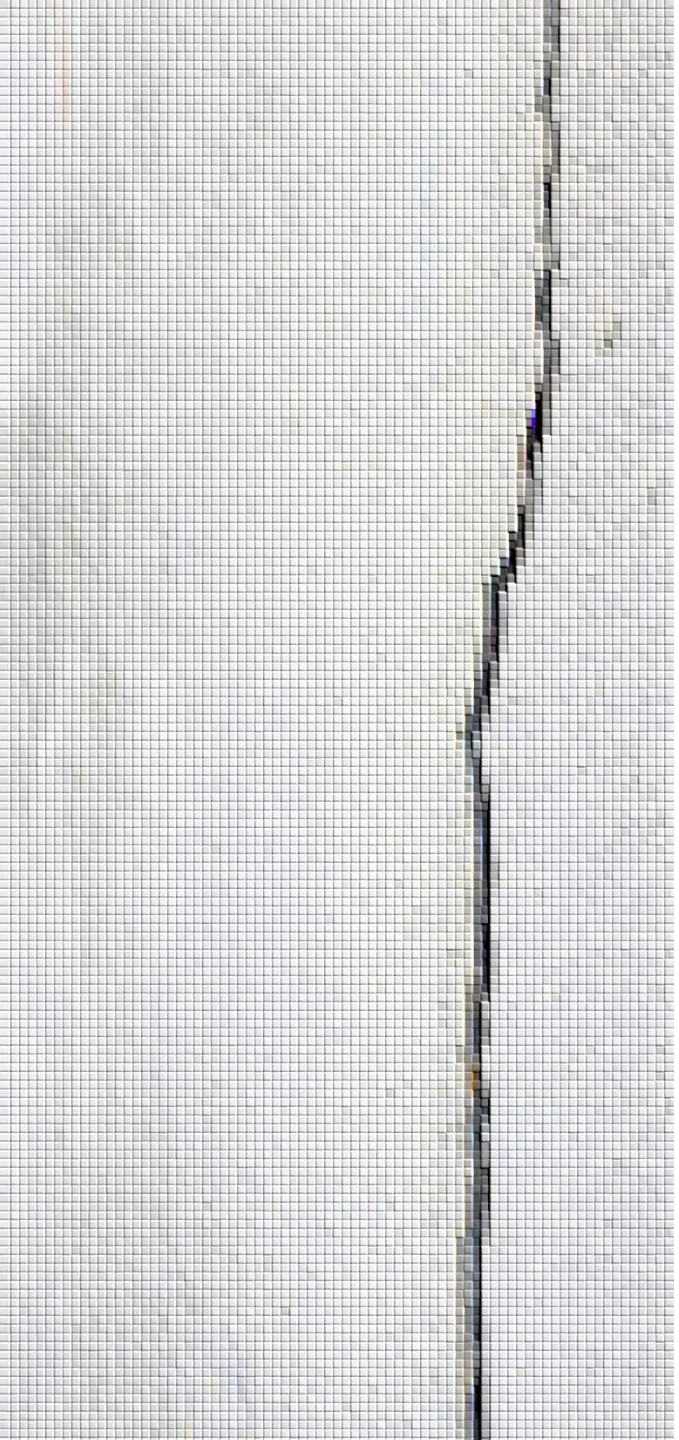




Model Construction – Machine Learning

- Evaluation of metrics

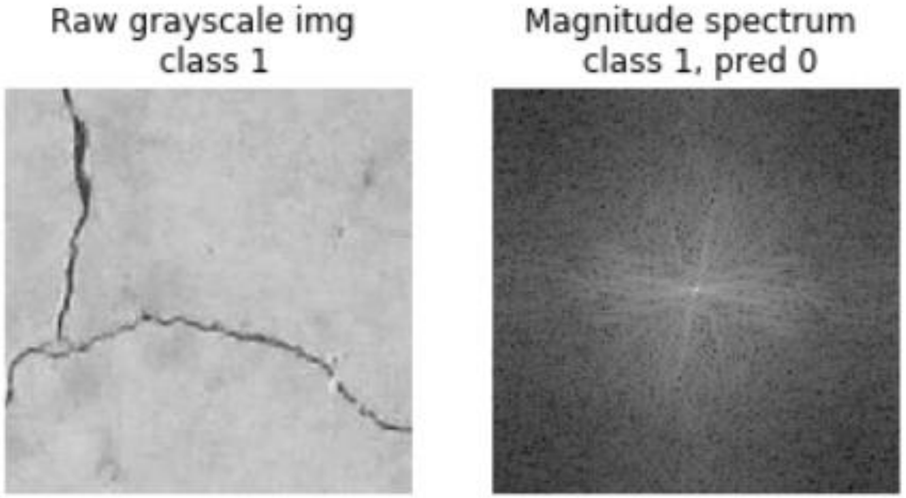
Model	Accuracy	F1-score	Recall	Specificity	Precision
Logistic Regression	0.969	0.969	0.972	0.972	0.967
Random Forest	0.976	0.976	0.976	0.976	0.976
AdaBoost	0.969	0.969	0.967	0.971	0.971
XGBoost	0.971	0.971	0.969	0.974	0.974
KNN	0.974	0.974	0.974	0.975	0.975
SVM	0.952	0.954	0.987	0.917	0.923



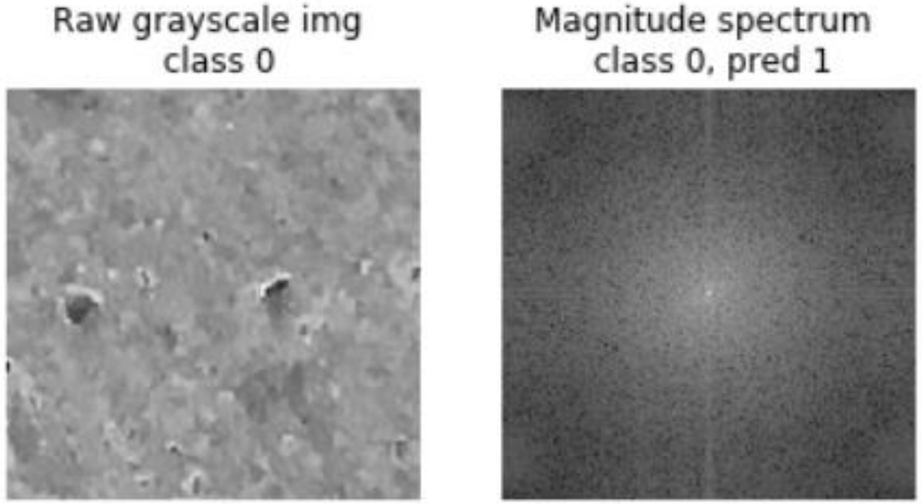
Model Evaluation – Machine Learning

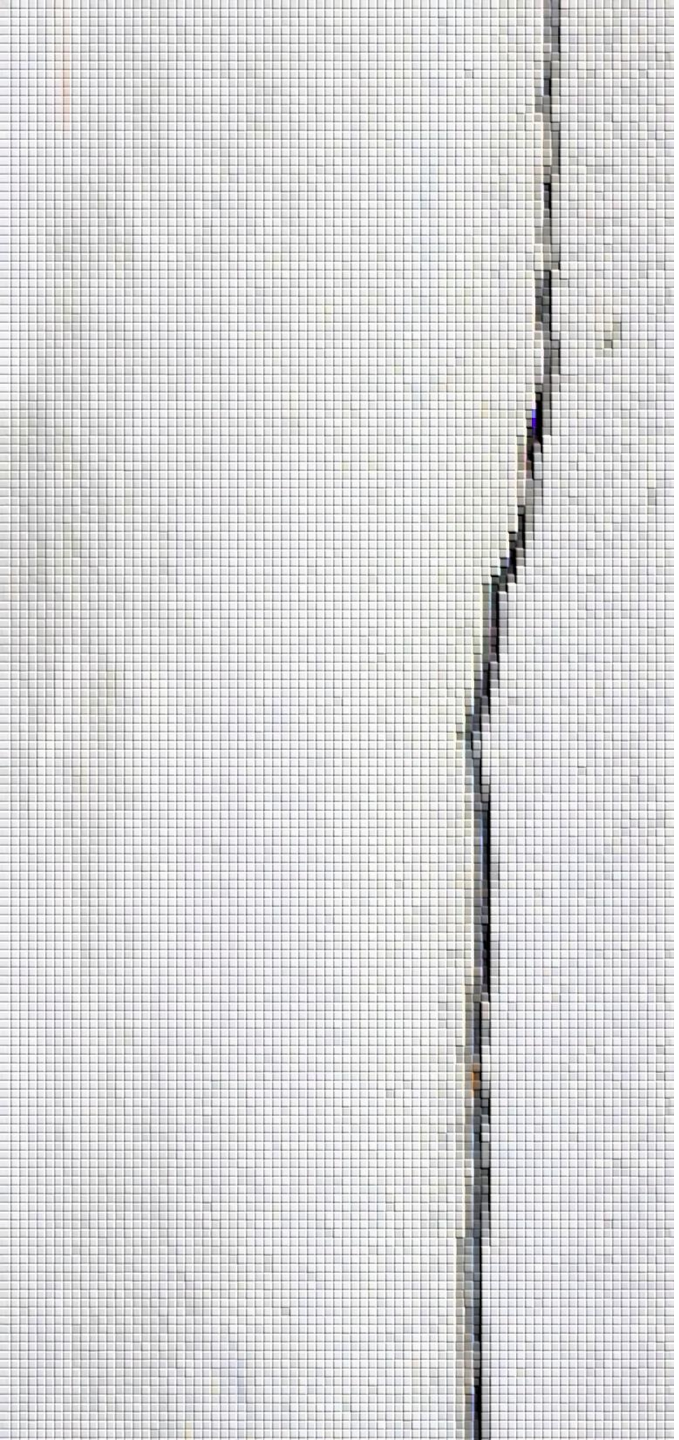
- Misclassified images

Random Forest classifier:



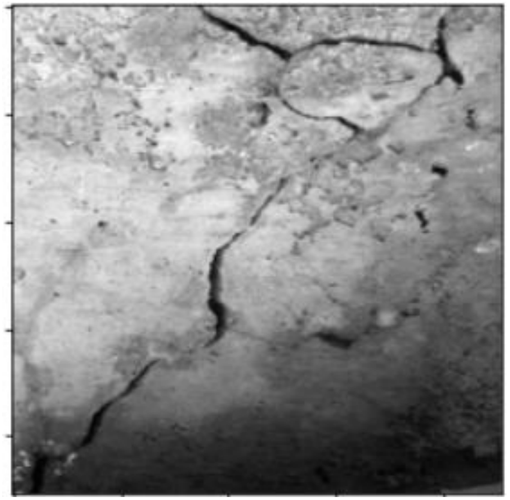
AdaBoost classifier:





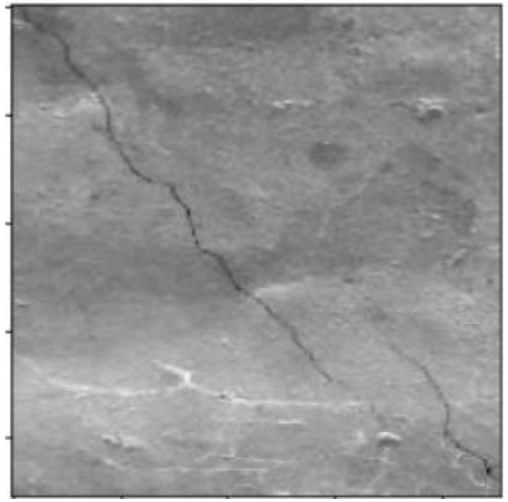
Models Performance

- Testing on unseen images – Positive Class



```
prediction = cnn_model2.predict_proba(img)
prediction[0][0]
```

0.9999999



```
prediction = cnn_model2.predict_proba(img)
prediction[0][0]
```

0.0010422585

```
logreg_pkl = joblib.load('../model/logreg.pkl')
logreg_pkl.predict_proba(img_dff)[0][1]
```

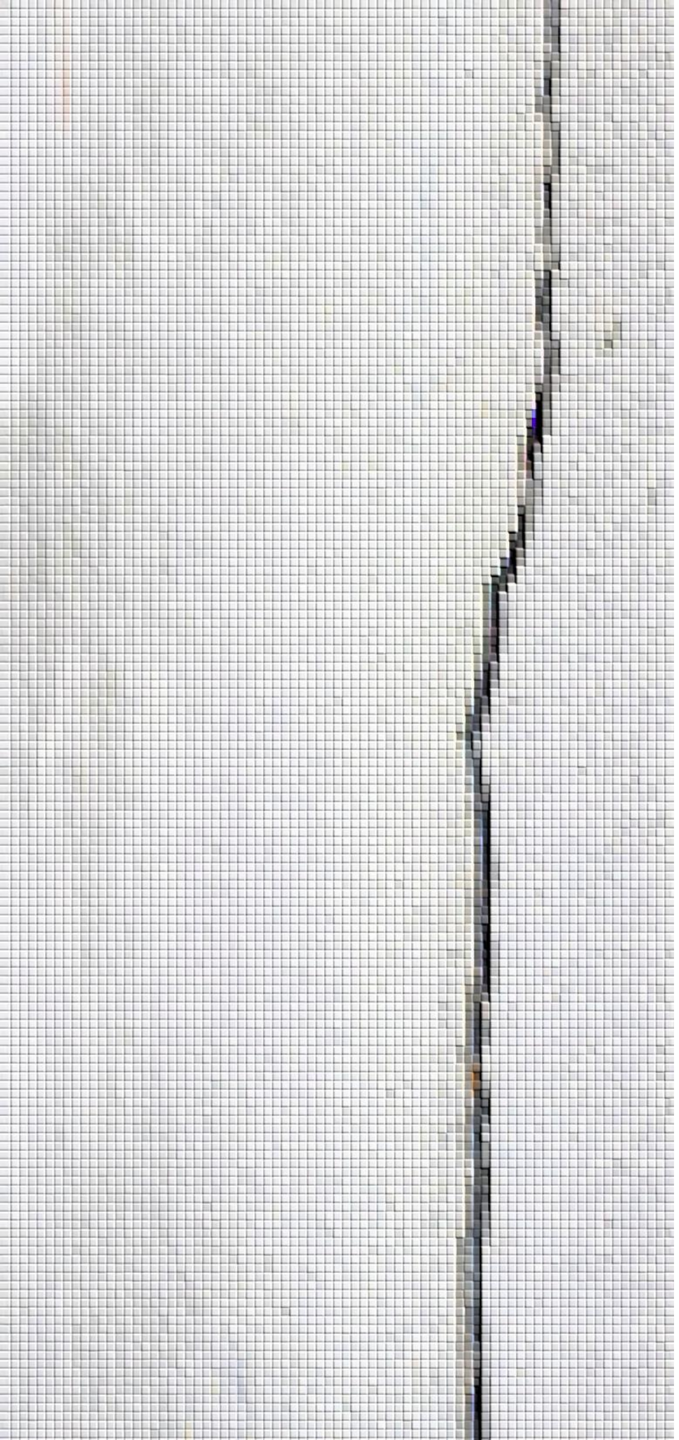
0.9645558888529413

Meta classifier

- Voting Classifier

```python
def voting_classifier(cnn_model = [cnn_model2],
                      ml_models = [rf_model]): #[logreg_model, rf_model, ada_model, xgb_model, knn_model, svm_model]):
    pred_probas = []
    preds = None
    pred = None
    for model in cnn_model:
        pred_probas.append(model.predict_proba(img)[0][0])

    for model in ml_models:
        pred_probas.append(model.predict_proba(img_dff)[0][1])

    print(pred_probas)
    preds = [1 if proba >= 0.5 else 0 for proba in pred_probas]
    #print(preds)
    if (sum(preds) >= 1) & (pred_probas[0] > 0.02):
        pred = 1
        #print(f"{sum(preds)} out of {len(preds)} models: Surface has a crack.")
        print("Conclusion: Cracked Surface (Class 1)")
    else:
        pred = 0
        #print(f"{sum(preds)} out of {len(preds)} models: Surface has a crack.")
        print("Conclusion: Non cracked surface (Class 0)")

    return pred

```

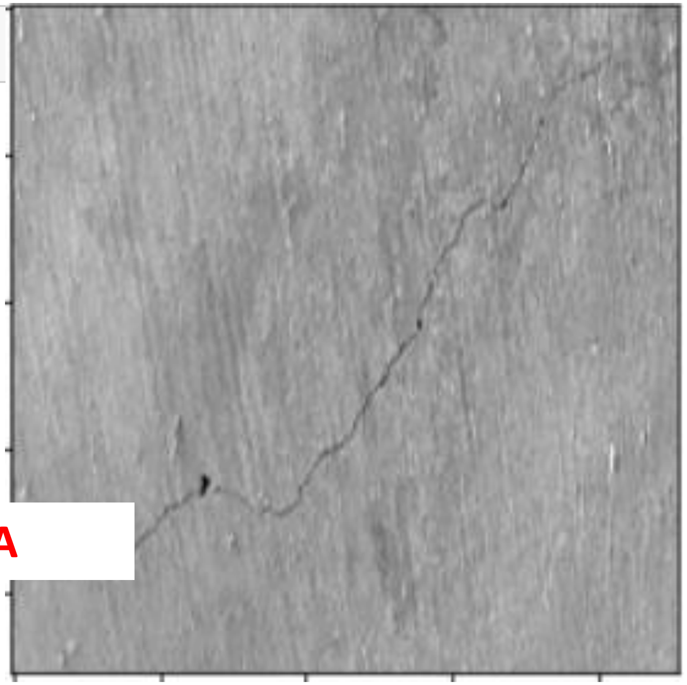
2 models total



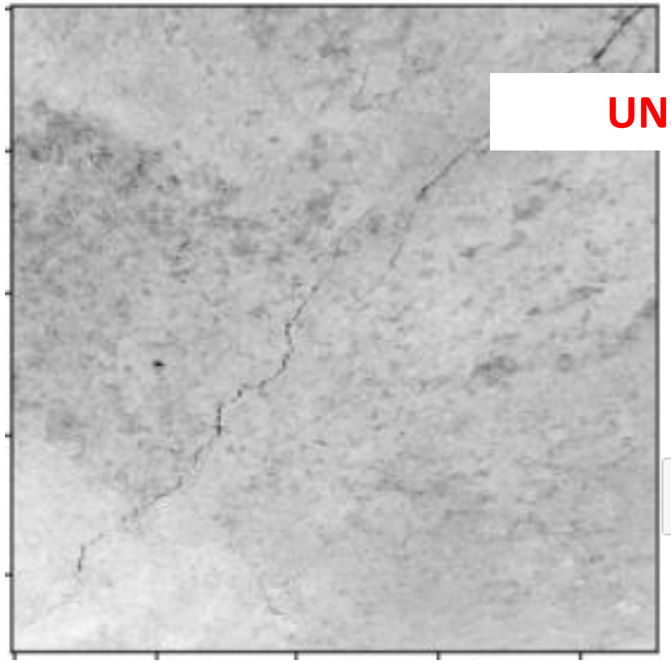
Voting Classifier – Class 1

```
voting_classifier()
```

```
[0.050882954, 0.6754761904761903]
Conclusion: Cracked Surface (Class 1)
```



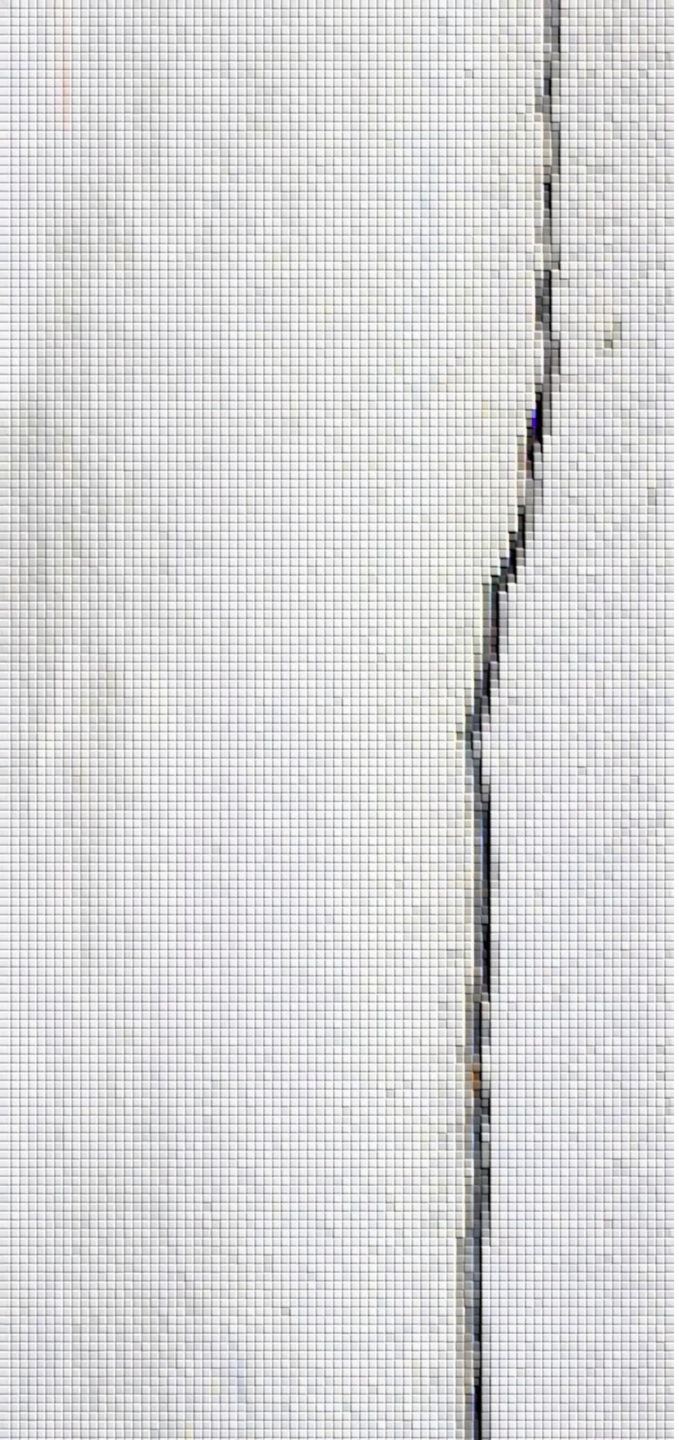
UNSEEN DATA



```
voting_classifier()
```

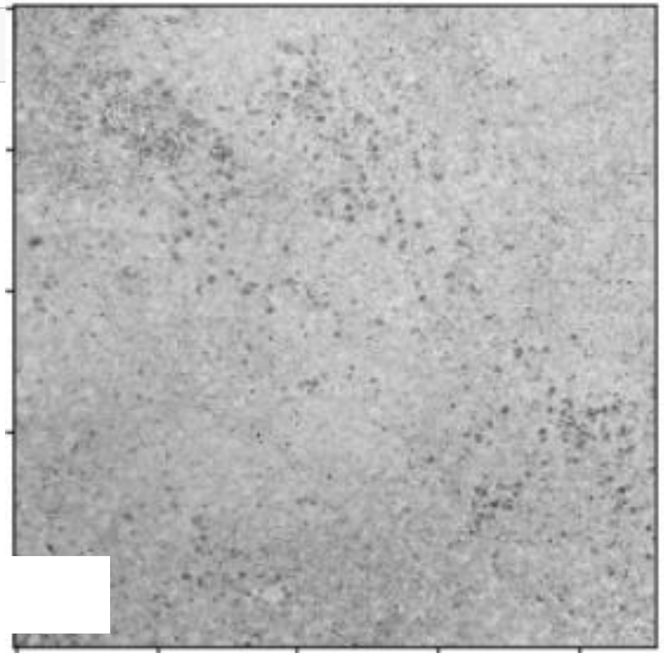
```
[0.02636993, 0.6546428571428572]
Conclusion: Cracked Surface (Class 1)
```

Voting Classifier – Class 0

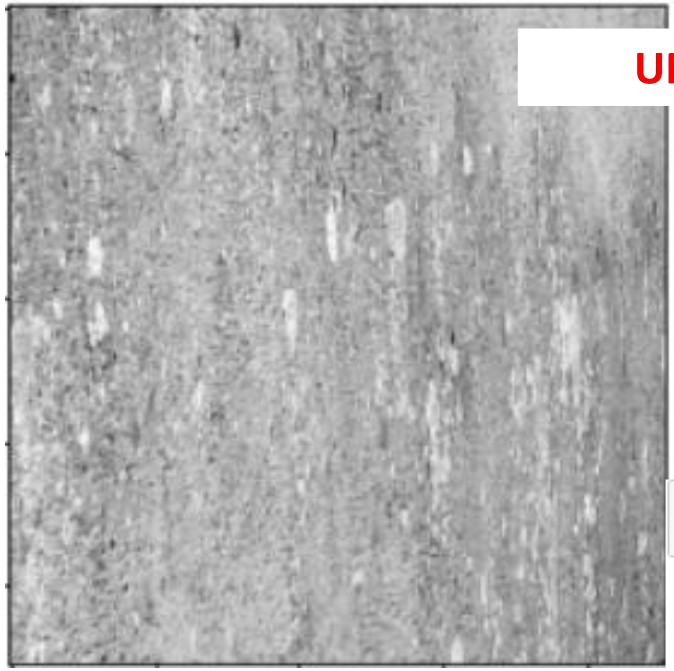


```
voting_classifier()
```

```
[0.010949683, 0.5233214285714285]
Conclusion: Non cracked surface (Class 0)
```

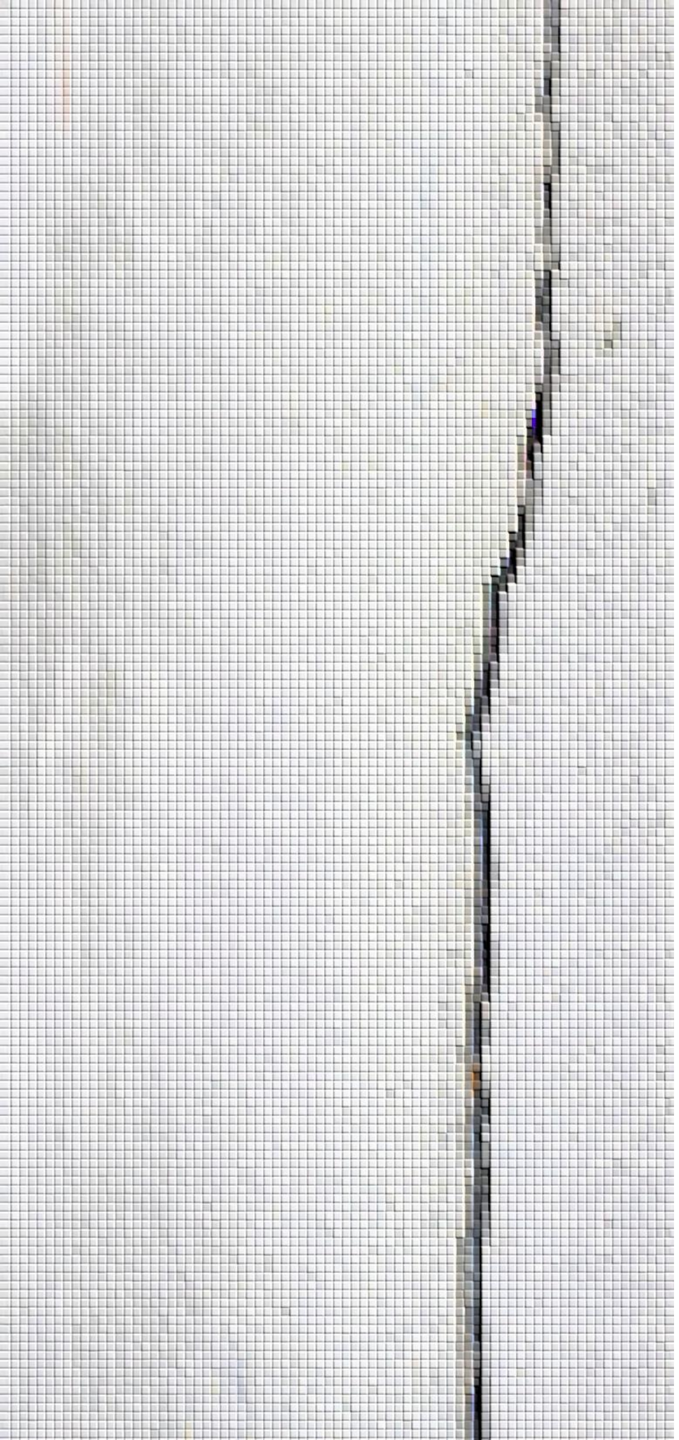


UNSEEN DATA



```
voting_classifier()
```

```
[7.211921e-05, 0.5633214285714286]
Conclusion: Non cracked surface (Class 0)
```



Conclusion & Summary

CNN model	Machine Learning Models
Good at differentiating craters/bumps on surface from actual cracks	Bad at this
Bad at detecting narrow/small cracks	Good at this

- Meta classifier (voting classifier) used to increase reliability

Limitations

- Generally, 2 scenarios that caused misclassification:
  - Thin, narrow cracks
  - Surfaces with a lot of small craters
- Cracks are only 1 type of defect

Future work

- Extend to other types of defect