Skoltech

Crack detection

Using deep learning approach

By :Addisu
Amare
Zena

content

- Problem statement
- Introduction
- Methodology
- Result
- conclusion

Problem statement

- The problem is to develop an effective and efficient system for the detection of cracks in structures, such as buildings or bridges.
- This involves creating a model capable of accurately identifying and localizing cracks in images or sensor data.
- The goal is to automate the detection process, providing a reliable tool for structural health monitoring to prevent potential failures and ensure the safety and integrity of infrastructure.
- Manual crack detection is laborious, time-consuming, and prone to subjective judgments from inspectors.
- Particularly challenging for tall buildings and bridges, manual inspection becomes impractical.

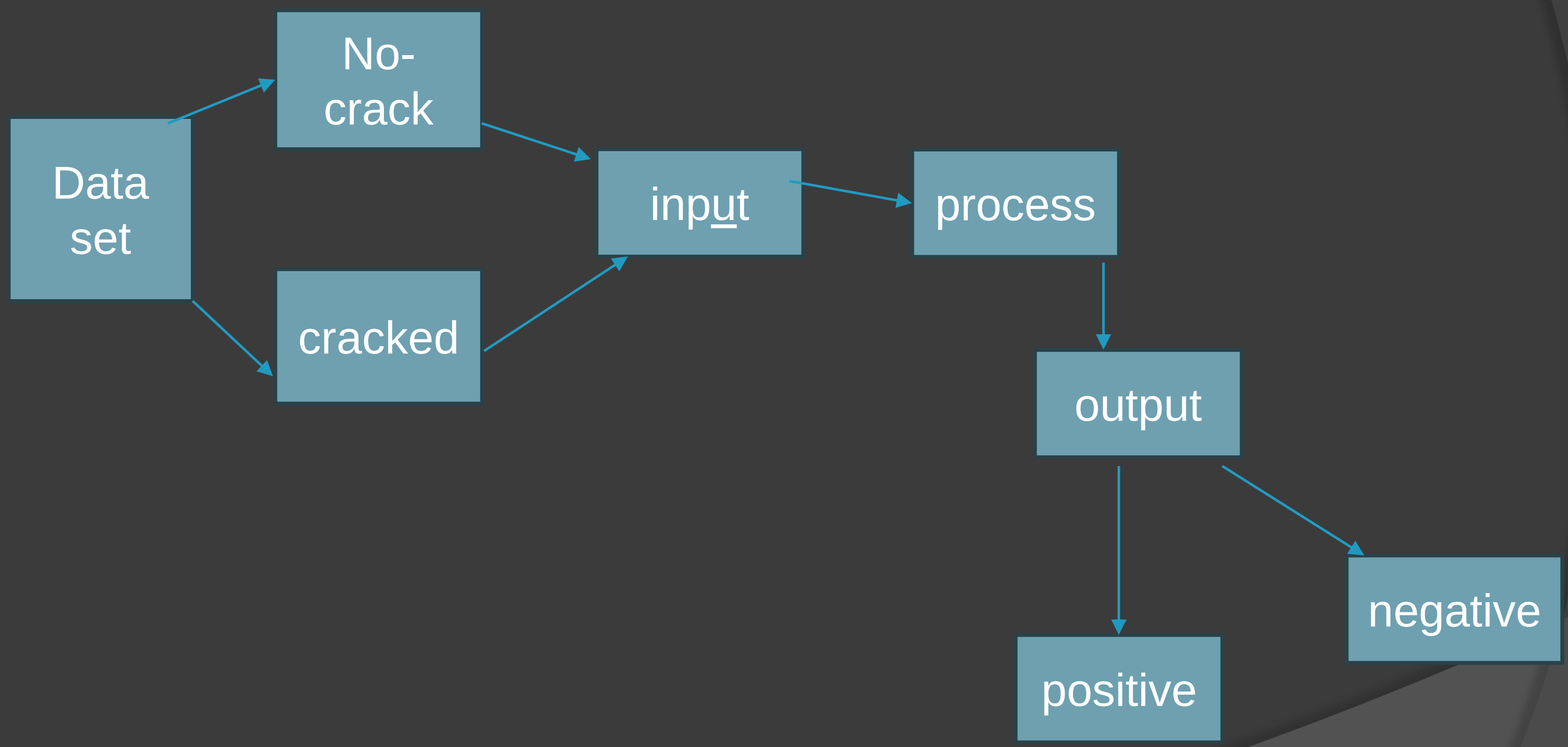
Introduction

- Detecting cracks is a crucial aspect of monitoring the structural integrity of concrete structures and other surface structures
- The goal of this project is to develop a robust crack detection system using deep learning techniques.
- The system will analyse images of surfaces to identify and localize crack
- Automated methods offer a more efficient and objective solution to identify and analyze surface cracks in concrete structures.

Skoltech

Methodology:

1 using deep learning approach



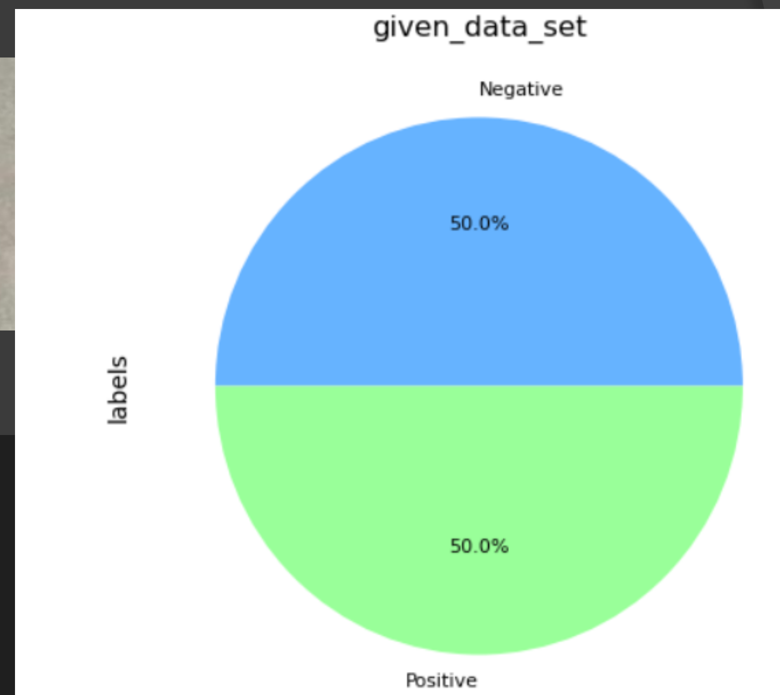
2 using non max suppression approach and edge detector

Dataset

Data set consist of 40000 images

20,000

20,000

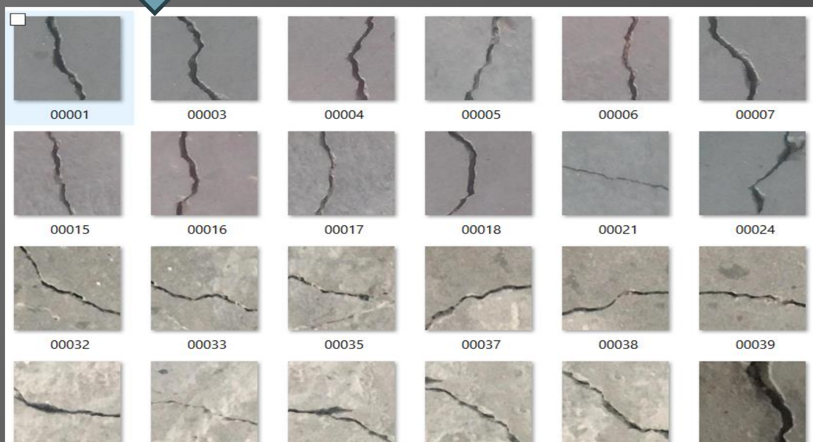


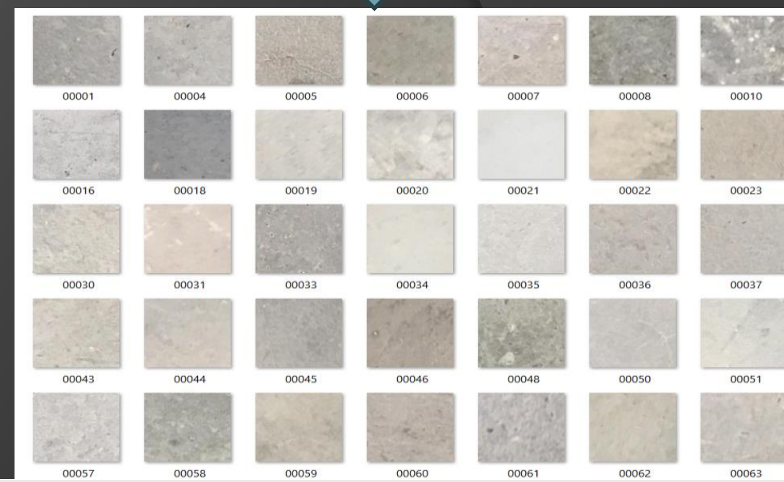
Skoltech

positive

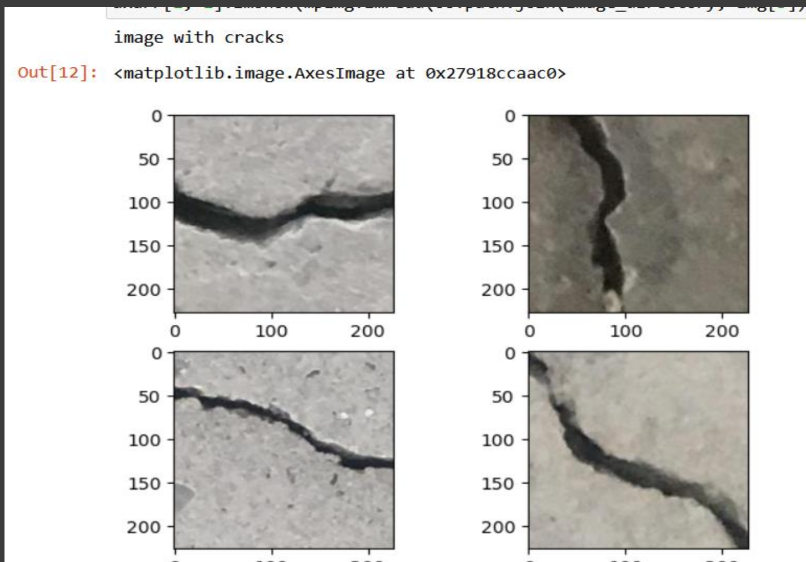
Both NAP

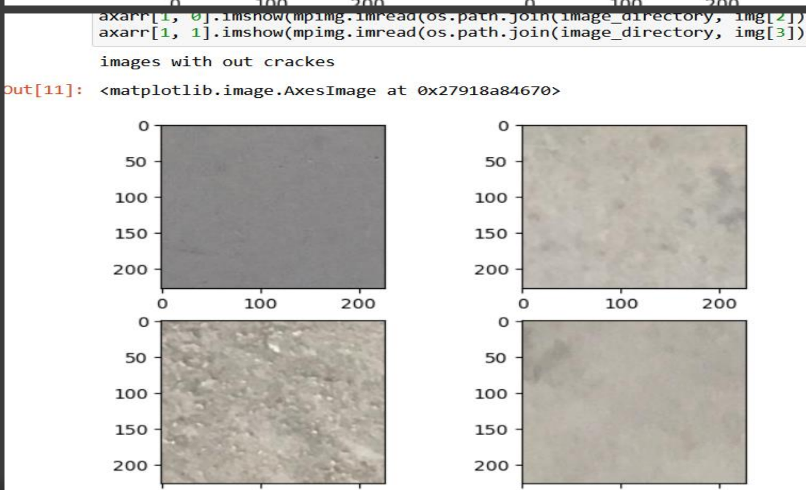
Negative





preprocessing





crack

Non crack

Out[14]:

	Filepath	Label
0	C:\Users\HP\Downloads\archive (4)\Positive\038...	Positive
1	C:\Users\HP\Downloads\archive (4)\Positive\128...	Positive
2	C:\Users\HP\Downloads\archive (4)\Positive\150...	Positive
3	C:\Users\HP\Downloads\archive (4)\Negative\167...	Negative
4	C:\Users\HP\Downloads\archive (4)\Positive\092...	Positive
...	...	...
39995	C:\Users\HP\Downloads\archive (4)\Positive\078...	Positive
39996	C:\Users\HP\Downloads\archive (4)\Negative\125...	Negative
39997	C:\Users\HP\Downloads\archive (4)\Positive\051...	Positive
39998	C:\Users\HP\Downloads\archive (4)\Positive\121...	Positive
39999	C:\Users\HP\Downloads\archive (4)\Negative\130...	Negative

40000 rows × 2 columns

Split the DataSet

In [46]:

```
train_df, test_df = train_test_split(all_df.sample(30000, random_state=1),
                                     train_size=0.7,
                                     shuffle=True,
                                     random_state=42)
```

n [48]:

```
train_df.shape
```

Out[48]: (21000, 2)

n [49]:

```
test_df.shape
```

Out[49]: (9000, 2)

Data augmentation and pre-processing

```
gen = ImageDataGenerator(rescale         = 1./255.,
                         horizontal_flip = True,
                         vertical_flip   = True,
                         zoom_range      = 0.05,
                         rotation_range  = 25)
```

modeling

- 1. Input Layer:
- tf.keras.Input(shape=(120, 120, 3))
- Defines the input layer with a shape of (120, 120, 3), indicating an input image size of 120x120 pixels with 3 color channels (RGB).

- 2. Convolutional Layers:
- 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)
-
- - Applies two convolutional layers with 16 and 32 filters, respectively, each using a 3x3 kernel.
- - Applies max pooling with a 2x2 pool size after each

Conti.....

3. Global Average Pooling Layer:

- x=tf.keras.layers.GlobalAveragePooling2D()(x)

- Performs global average pooling to reduce the spatial dimensions to a single value per channel, creating a flat vector.

4. Output Layer:

- outputs = tf.keras.layers.Dense(1, activation='sigmoid')(x)
- Connects the output of the global average pooling layer to a dense layer with a single neuron (for binary classification) and a sigmoid activation function.

Training from scratch

Training model ¶

In [53]:

```
inputs = tf.keras.Input(shape=(128,128,3))
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.Conv2D(filters=64, kernel_size=(3,3), activation='relu')(x)
x = tf.keras.layers.MaxPool2D(pool_size=(2,2))(x)
x = tf.keras.layers.Conv2D(filters=128, 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 summary of training from scratch

Layer (type)	Output Shape	Param #
input_4 (InputLayer)	[(None, 128, 128, 3)]	0
conv2d_11 (Conv2D)	(None, 126, 126, 16)	448
max_pooling2d_11 (MaxPooling2D)	(None, 63, 63, 16)	0
conv2d_12 (Conv2D)	(None, 61, 61, 32)	4640
max_pooling2d_12 (MaxPooling2D)	(None, 30, 30, 32)	0
conv2d_13 (Conv2D)	(None, 28, 28, 64)	18496
max_pooling2d_13 (MaxPooling2D)	(None, 14, 14, 64)	0
conv2d_14 (Conv2D)	(None, 12, 12, 128)	73856
max_pooling2d_14 (MaxPooling2D)	(None, 6, 6, 128)	0
global_average_pooling2d_3 (GlobalAveragePooling2D)	(None, 128)	0

For 15000 sample

```
])

Epoch 1/10
280/280 [==============================] - 97s 342ms/step - loss: 0.1775 - accuracy: 0.9312 - val_loss: 0.0585 - val_accurac
y: 0.9790
Epoch 2/10
280/280 [==============================] - 35s 124ms/step - loss: 0.0738 - accuracy: 0.9795 - val_loss: 0.0388 - val_accurac
y: 0.9862
Epoch 3/10
280/280 [==============================] - 33s 118ms/step - loss: 0.0626 - accuracy: 0.9824 - val_loss: 0.0461 - val_accurac
y: 0.9853
Epoch 4/10
280/280 [==============================] - 33s 118ms/step - loss: 0.0539 - accuracy: 0.9839 - val_loss: 0.0475 - val_accurac
y: 0.9839
Epoch 5/10
280/280 [==============================] - 33s 118ms/step - loss: 0.0533 - accuracy: 0.9844 - val_loss: 0.0439 - val_accurac
y: 0.9857
```

Cont.........................

For 30,000 sample

```
Epoch 1/10
525/525 [==============================] - 213s 404ms/step - loss: 0.1291 - accuracy: 0.9573 - val_loss: 0.0600 - val_accura
cy: 0.9807
Epoch 2/10
525/525 [==============================] - 151s 287ms/step - loss: 0.0779 - accuracy: 0.9779 - val_loss: 0.0860 - val_accura
cy: 0.9705
Epoch 3/10
525/525 [==============================] - 70s 133ms/step - loss: 0.0659 - accuracy: 0.9826 - val_loss: 0.0460 - val_accurac
y: 0.9886
Epoch 4/10
525/525 [==============================] - 68s 129ms/step - loss: 0.0428 - accuracy: 0.9870 - val_loss: 0.0554 - val_accurac
y: 0.9879
Epoch 5/10
525/525 [==============================] - 68s 130ms/step - loss: 0.0540 - accuracy: 0.9851 - val_loss: 0.0406 - val_accurac
y: 0.9862
Epoch 6/10
525/525 [==============================] - 68s 129ms/step - loss: 0.0359 - accuracy: 0.9889 - val_loss: 0.0258 - val_accurac
y: 0.9924
Epoch 7/10
525/525 [==============================] - 68s 129ms/step - loss: 0.0352 - accuracy: 0.9898 - val_loss: 0.0352 - val_accurac
y: 0.9912
Epoch 8/10
525/525 [==============================] - 69s 131ms/step - loss: 0.0333 - accuracy: 0.9901 - val_loss: 0.0505 - val_accurac
y: 0.9829
Epoch 9/10
525/525 [==============================] - 69s 131ms/step - loss: 0.0303 - accuracy: 0.9908 - val_loss: 0.0417 - val_accurac
y: 0.9900
```

Using Resnet_v2

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception_resnet_v2/inception_resnet_v2_w
eights_tf_dim_ordering_tf_kernels_notop.h5
219055592/219055592 [==============================] - 43s 0us/step
Unfreezing number of layers in base model =  0
Epoch 1/3
WARNING:tensorflow:From C:\Users\HP\AppData\Roaming\Python\Python39\site-packages\keras\src\utils\tf_utils.py:492: The name
tf.ragged.RaggedTensorValue is deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue instead.

WARNING:tensorflow:From C:\Users\HP\AppData\Roaming\Python\Python39\site-packages\keras\src\engine\base_layer_utils.py:384:
The name tf.executing_eagerly_outside_functions is deprecated. Please use tf.compat.v1.executing_eagerly_outside_functions i
nstead.

750/750 [==============================] - 4655s 6s/step - loss: 0.0448 - accuracy: 0.9855 - val_loss: 0.0096 - val_accurac
y: 0.9965
Epoch 2/3
750/750 [==============================] - 5218s 7s/step - loss: 0.0250 - accuracy: 0.9920 - val_loss: 0.0094 - val_accurac
y: 0.9965
Epoch 3/3
750/750 [==============================] - 4653s 6s/step - loss: 0.0211 - accuracy: 0.9932 - val_loss: 0.0103 - val_accurac
y: 0.9965
```

Performance metric

$$\text{True Positive Rate (TPR)} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

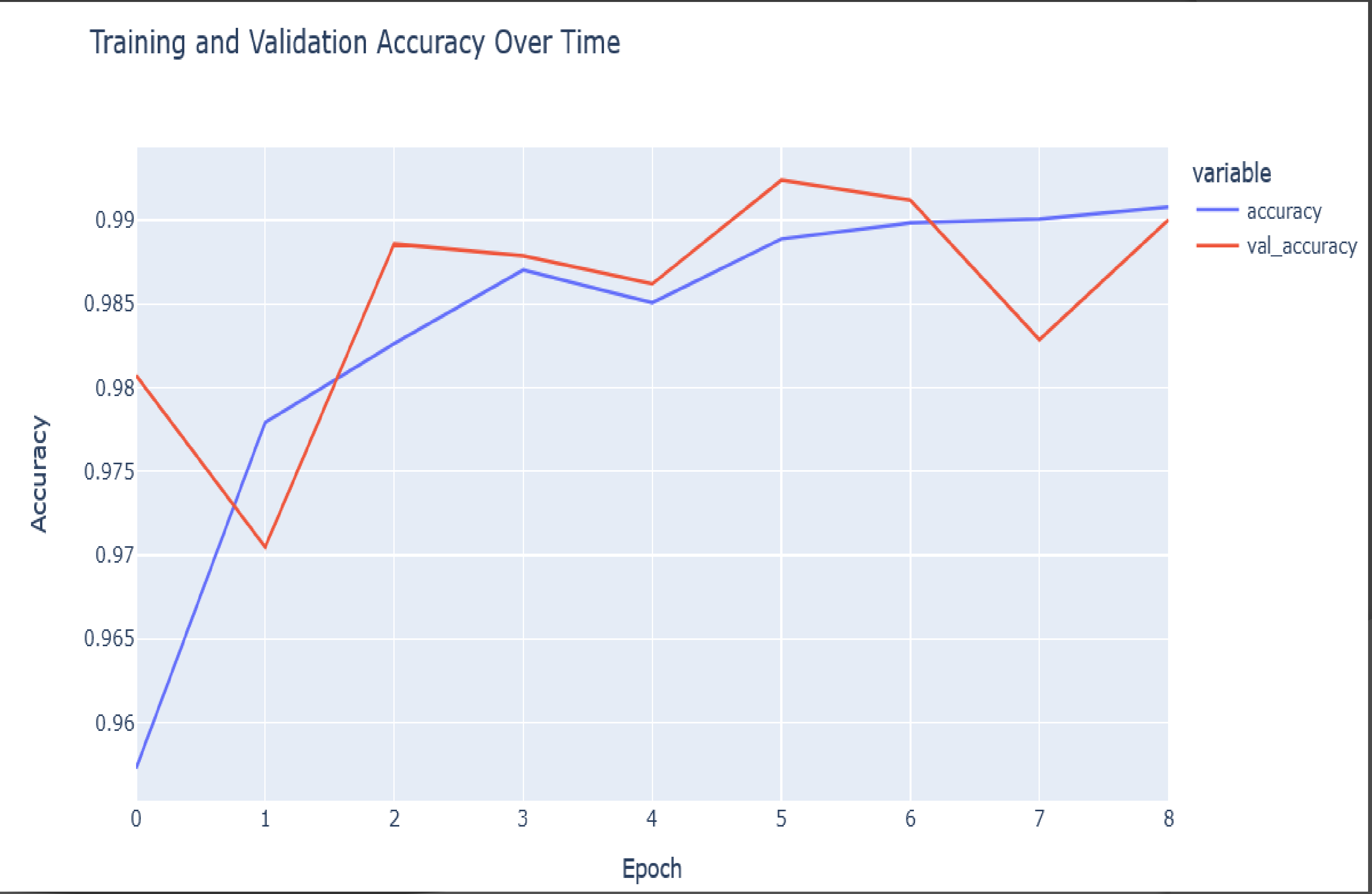
$$\text{False Positive Rate (FPR)} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

$$\text{Specificity} = \frac{\text{TN}}{\text{FP} + \text{TN}}$$

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FN} + \text{TN} + \text{FP}}$$

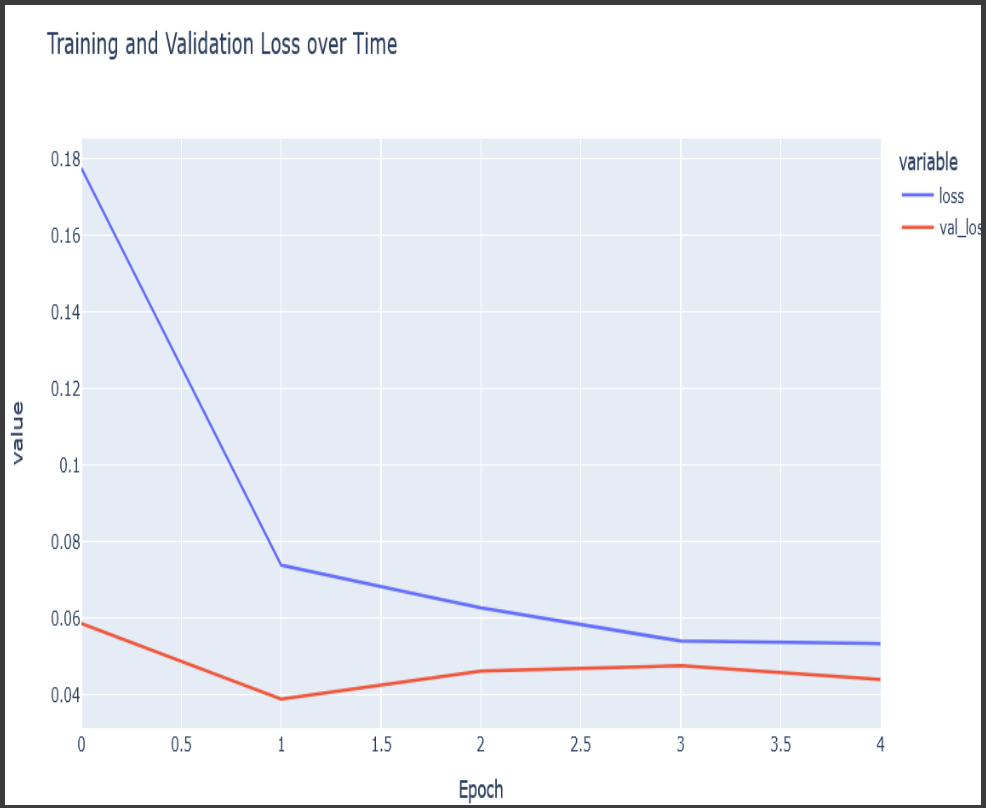
$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

Skoltech

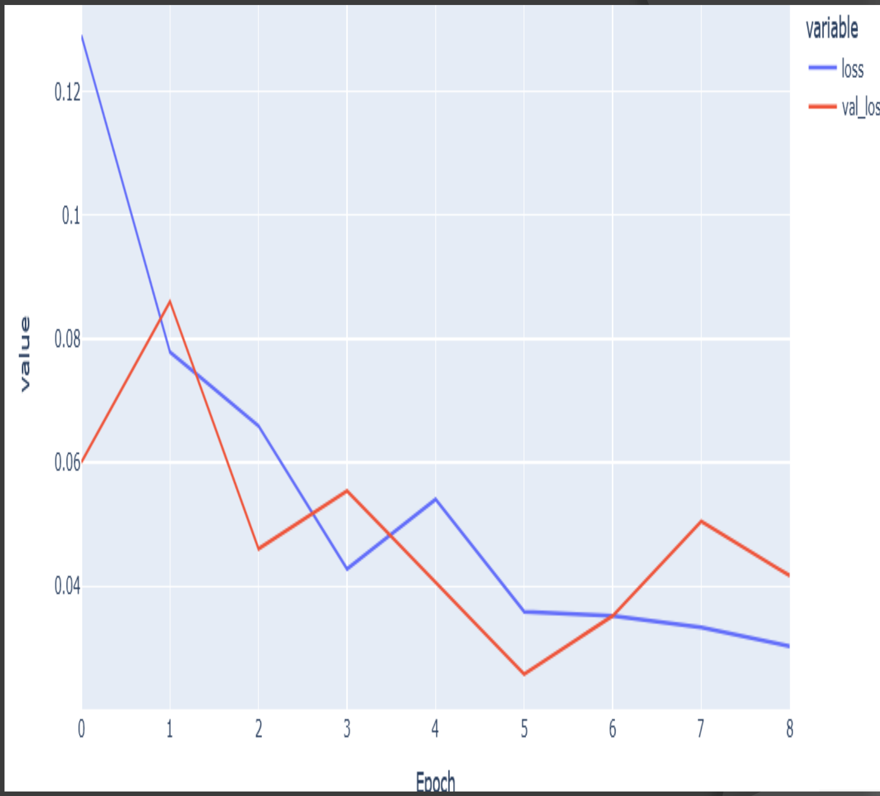


Training and Validation Accuracy Over Time
variable
accuracy
val_accuracy
Accuracy
0.99
0.985
0.98
0.975
0.97
0.965
0.96
0
1
2
3
4
5
6
7
8
Epoch

Result

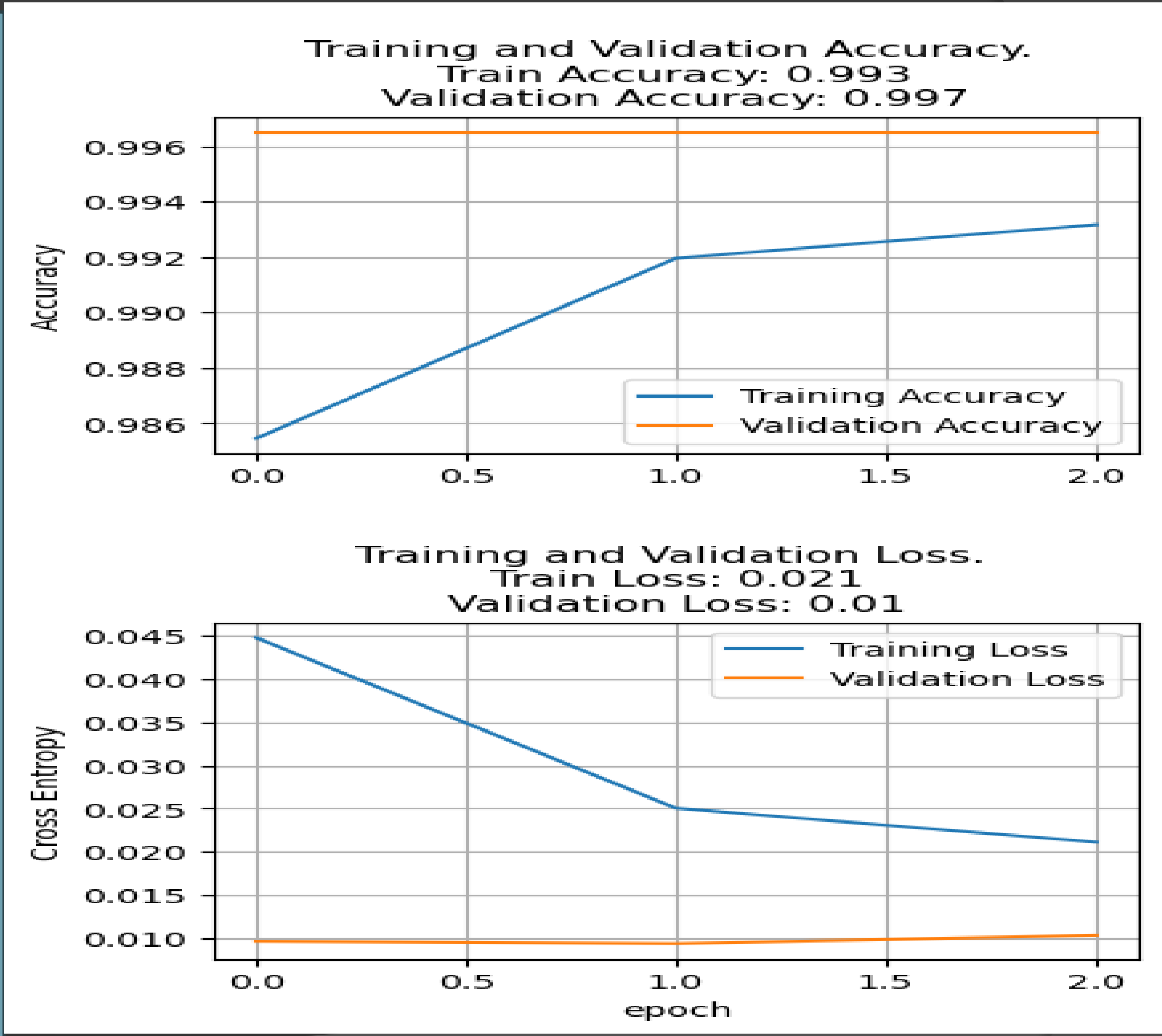


For 15000 sample



For 30000 sample

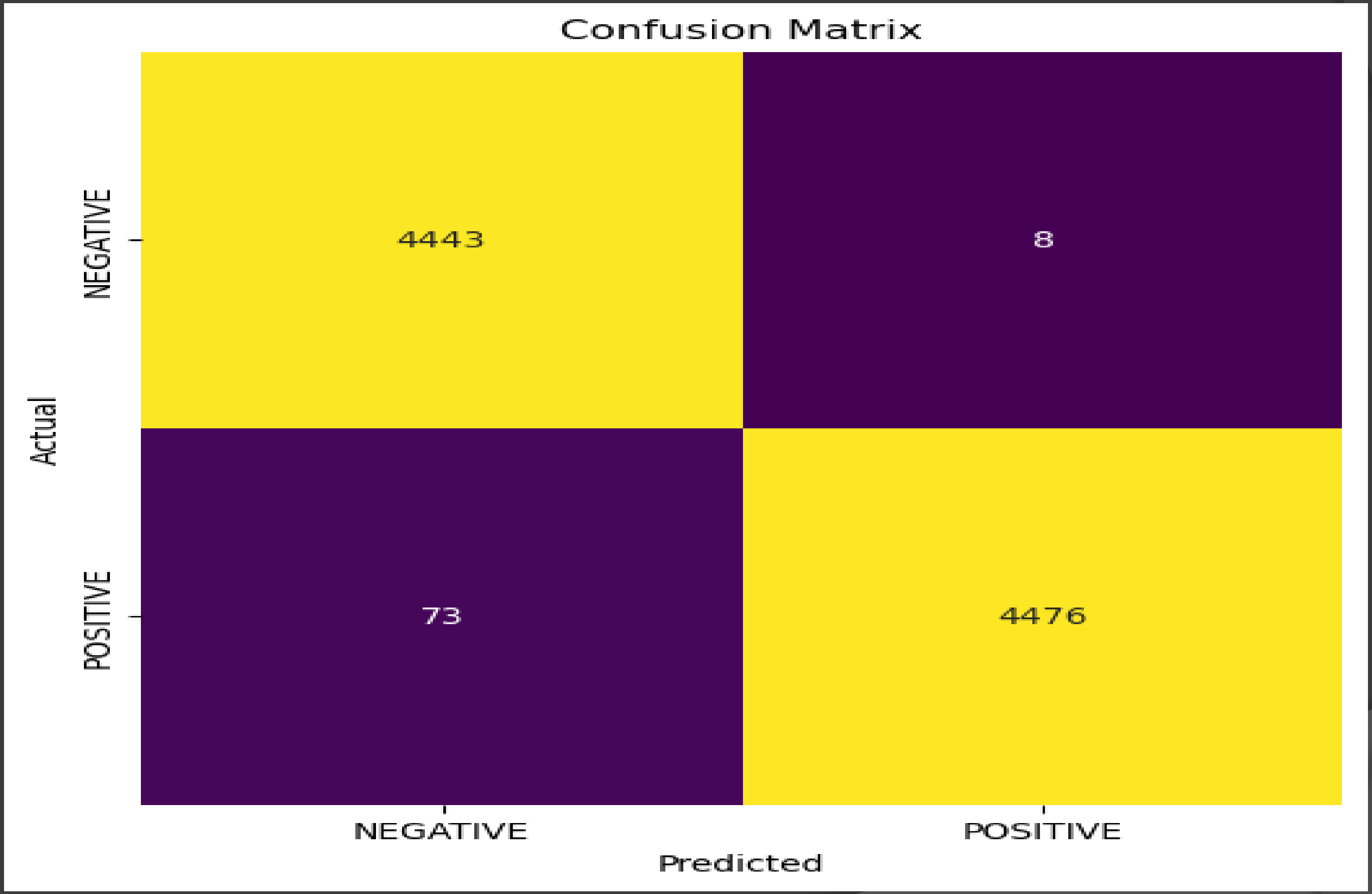
Using pretrained model



Result

```
]:    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)

]:    evaluate_model(model, test_data)

          Test Loss: 0.02858
      Test Accuracy: 99.10%
```

Confusion matrix



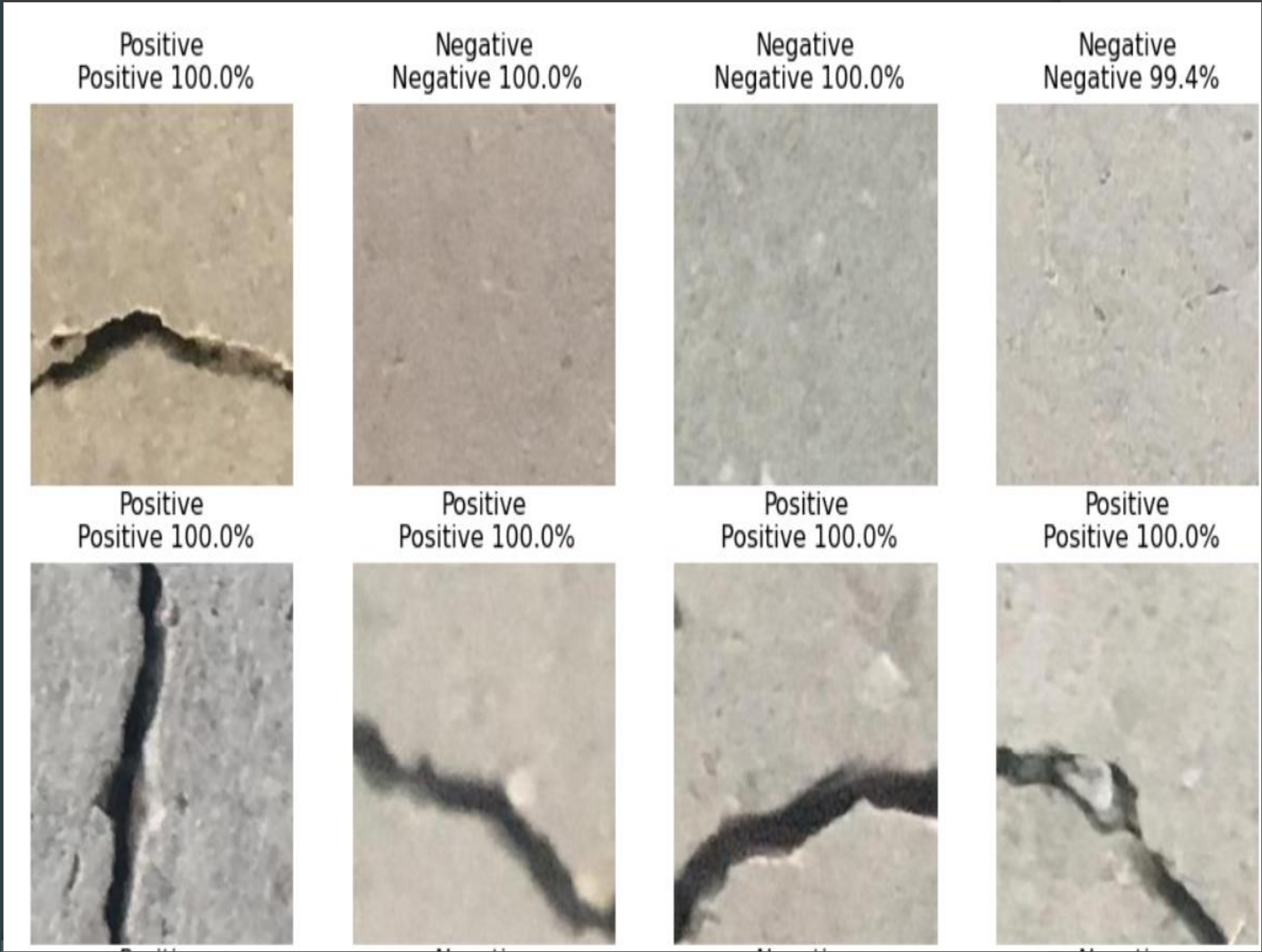
Cont…………..

Classification Report:

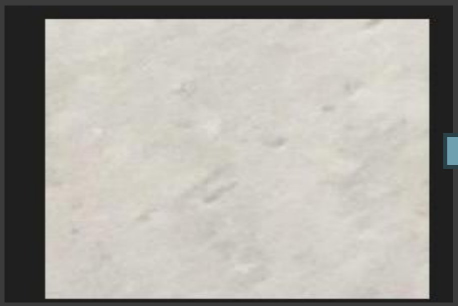
----

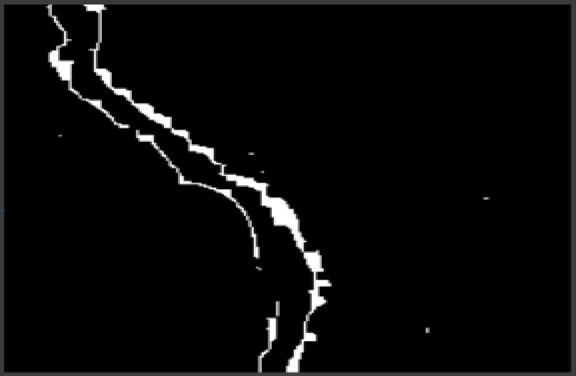
	precision	recall	f1-score	support
NEGATIVE	0.98	1.00	0.99	4451
POSITIVE	1.00	0.98	0.99	4549
accuracy			0.99	9000
macro avg	0.99	0.99	0.99	9000
weighted avg	0.99	0.99	0.99	9000

Predicted output



Using non max suppression and edge detector





input

output

Skoltech

Conclusion

- By comparing result of using pretrained model and training from scratch
- Utilizing pre-trained model is better than training from scratch in term s of accuracy.
-

Challenges and future work

- When I reduce the number of sample, overfitting happened
- Incorporate better accurate model to hard ware .make it more real time.
- Applying non max suppression with deep learning

Thank you