VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

School of Computer Science and Engineering

J Component report

Programme : Integrated M.tech Specialisation in Business Analytics

Course Title : DEEP LEARNING TECHNIQUES

Course Code : CSE4037

Slot **:** G2

Title

CONCRETE CRACK DETECTION

Team Members:

P B S S Jaswanth | 19MIA1039

J Mouli Prasad | 19MIA1067

R Sree Karthik | 19MIA1088

Faculty: Dr. K Kadar Nawas **Sign:**

Date:

INDEX

ACKNOWLEDGMENT

Primarily, we would like to thank the almighty for all the blessings he showered over us to complete this project without any flaws.

The success and final outcome of this assignment required a lot of guidance and assistance from many people and we are extremely fortunate to have got this all along with the completion of our project. Whatever we have done is only due to such guidance and assistance by our faculty, Dr. K Khadar Nawas, to whom we are really thankful for giving us an opportunity to do this project.

Last but not the least, we are grateful to all our fellow classmates and our friends for the suggestions and support given to us throughout the completion of our project.

ABSTRACT

Cracks on the concrete surface are one of the earliest indications of degradation of the structure which is critical for the maintenance, quality as well the continuous exposure will lead to the severe damage to the environment. Manual inspection is the acclaimed method for the crack inspection. In the manual inspection, the sketch of the crack is prepared manually, and the conditions of the irregularities are noted. Since the manual approach completely depends on the specialist's knowledge and experience, it lacks objectivity in the quantitative analysis. So, automatic image-based crack detection is proposed as a replacement.

Inspired by recent success on applying deep learning to computer vision and medical problems, a deep-learning based method for crack detection is proposed in this project. Firstly, a supervised deep convolutional neural network (CNN) is trained to classify each image patch in the collected images and then using transfer learning we will use pretrained models like VGG16, ResNet50, Xception and MobileNet and apply to this dataset and with the resulted models we will find the test accuracies for all the models and with that we will find the best model.

INTRODUCTION

Infrastructures, such as bridges, roads, and dams becoming worse due to environmental and loading effects. For instance, a large portion of bridges in Japan and the USA have been in service for more than 50 years. Statistical analysis of bridge inspection data shows 46% of collapsed bridges were structurally not healthy prior to the collapse. And also, age and structural deficiency are shown to be related.

It is an important task to monitor the structural health of concrete structures. If cracks develop and continue to propagate, they reduce the effective load bearing surface area and can over time cause failure of the structure. The manual process of crack detection is time-consuming and also need specialized skilled people. Manual inspection can be difficult to perform in case of high-rise buildings and bridges.

So, we need an efficient maintenance strategy for detecting early signs of structural weaknesses. Therefore, we use deep learning to build a very accurate model for crack detection. Furthermore, we test the model on real world data and see that the model is accurate in detecting surface cracks in concrete.

1. DATASET:

We have taken the dataset from Mendeley Data

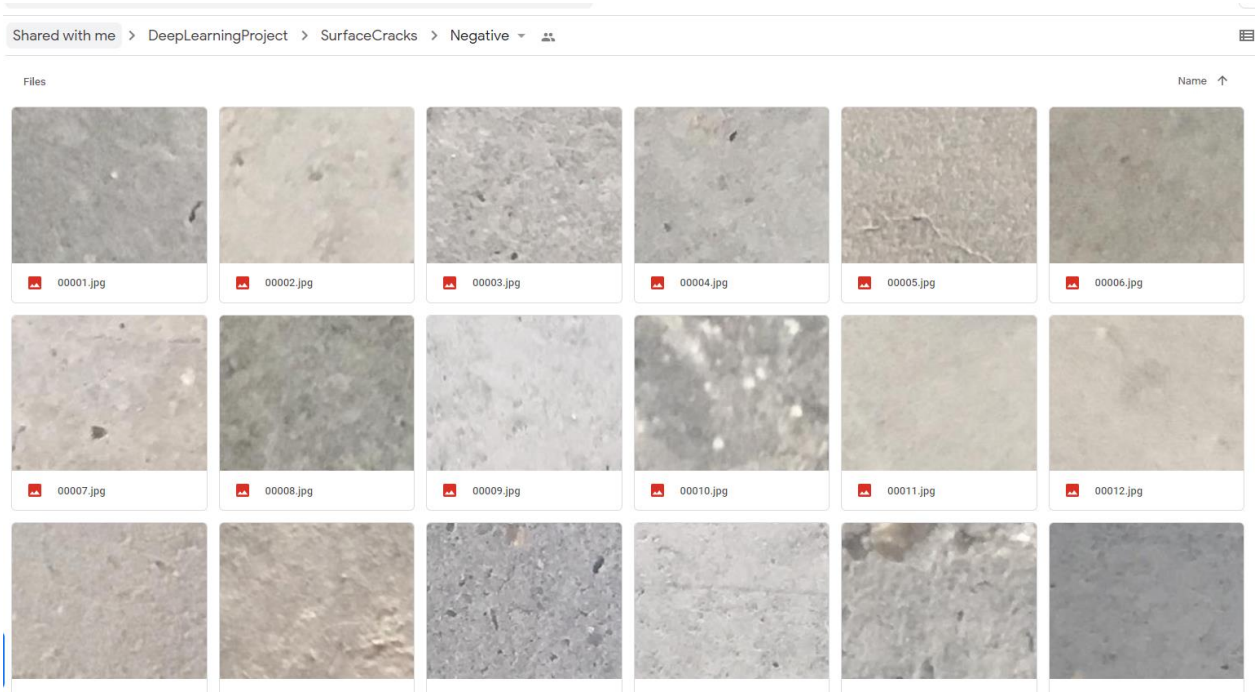
Link: https://data.mendeley.com/datasets/5y9wdsg2zt/2

Description

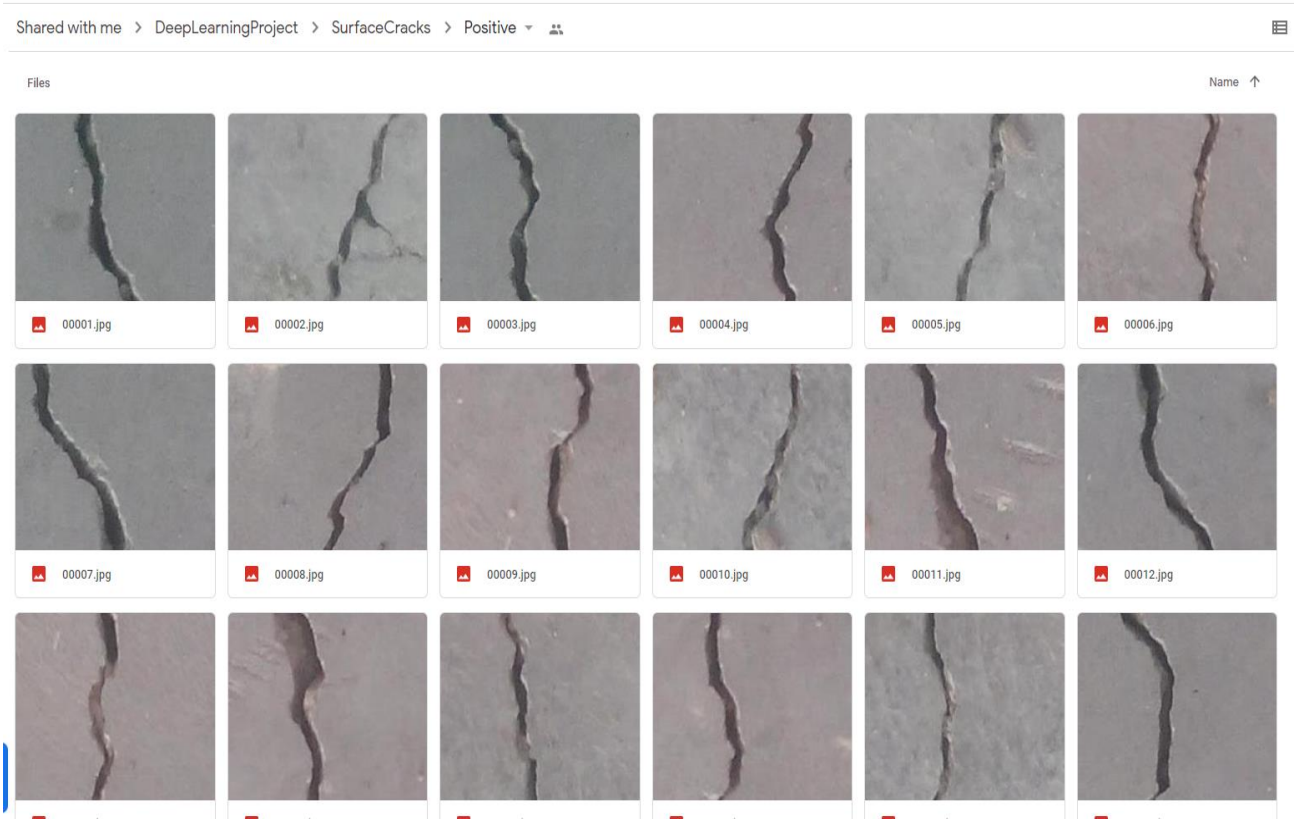
- The dataset contains concrete images having cracks. The dataset is divided into two i.e., negative and positive crack images for image classification.
- Each class has 20000images with a total of 40000 images with 227 x 227 pixels with RGB channels.

Dataset Screenshots:

1.1. Negative Class Images:



1.2. Positive Class Images:



2. PROBLEM STATEMENT:

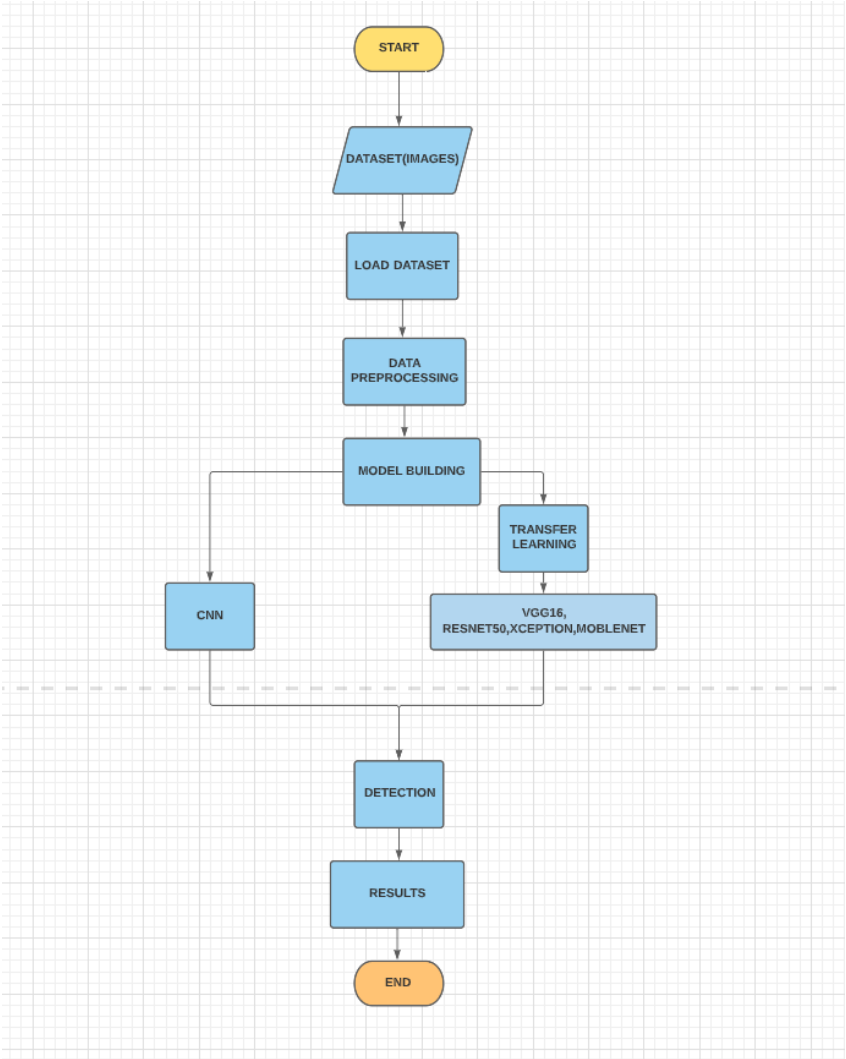
Annually, so much money has been spending to get various tools to detect defects from major infrastructure like roads, bridges, buildings, and water bodies. Civil structures such as roads, bridges, buildings are often exposed to extreme physical stress which may be caused by natural disasters like earthquakes etc.

So, in order to mitigate the problem of getting affected. Detecting the concrete cracks at early stage will be a good way to get progress for this particular type of problem. Here in this project we will be using CNN and Transfer Learning techniques to create prediction models and using that we will be predicting the crack detection.

3. EXISTING SOLUTIONS:

ANN (Artificial Neural Network) is used as crack detection technique. As one of the machine learning techniques, ANN learns given data inspired by biological neural networks. Particularly, deep learning. ANN-based method can contribute to overcoming the difficult or infeasible problems of the IP-based methods. For example, ANN using a restricted Boltzmann machine can successfully detect a crack in spite of the limited training samples. Compared to the ANNs, the CNNs learn image features using fewer parameters' computations due to the partial connections, sharing weights, and pooling process between neurons.

4. PROJECT WORK FLOW:



5. PROPOSED METHODOLOGY:

5.1. Loading Dataset:

Firstly, we have imported all necessary packages and libraries then, we have Loaded the dataset containing Images of 2 types

1. Positive (images with cracks)
2. Negative (images without cracks)

Each class has 20,000 images with 227 x 227 pixels with RGB channels.

```
import matplotlib.pyplot as plt
import seaborn as sns
import keras
from keras.models import Sequential
from keras.layers import Dense, Conv2D , MaxPool2D , Flatten , Dropout
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam, RMSprop, Adagrad
from keras.layers import BatchNormalization
from sklearn.metrics import classification_report,confusion_matrix
import tensorflow as tf
import cv2
import os
import numpy as np
import warnings
import pandas as pd
warnings.filterwarnings('ignore')
```

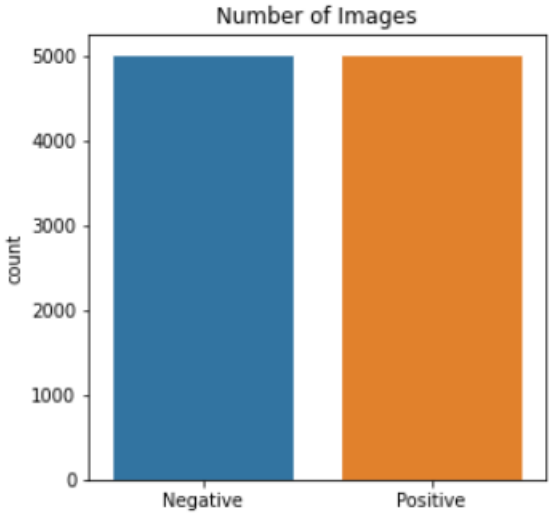
```
[ ] from google.colab import drive
    drive.mount('/content/drive')

    Mounted at /content/drive
```

```
[ ]  labels = ['Negative', 'Positive']
     img_size = 120
     def read_images(data_dir):
         data = []
         for label in labels:
             path = os.path.join(data_dir, label)
             class_num = labels.index(label)
             for img in os.listdir(path):
                 try:
                     img_arr = cv2.imread(os.path.join(path, img), cv2.IMREAD_GRAYSCALE)
                     resized_arr = cv2.resize(img_arr, (img_size, img_size))
                     data.append([resized_arr, class_num])
                 except Exception as e:
                     print(e)
         return np.array(data)

     Dataset = read_images('/content/drive/MyDrive/DeepLearningProject/SurfaceCracks')
```

- **Distribution of Negative and Positive images with their respective count**

```
Im = []
for i in Dataset:
    if(i[1] == 0):
        Im.append("Negative")
    elif(i[1] == 1):
        Im.append("Positive")

plt.figure(figsize=(10, 10))
plt.subplot(2, 2, 1)
sns.set_style('darkgrid')
axl = sns.countplot(Im)
axl.set_title("Number of Images")
```

```
Text(0.5, 1.0, 'Number of Images')
```



5.2. Train Test split

Splitting train and test datasets in 80:20 ratio:

```
[ ] from sklearn.model_selection import train_test_split
    X_train,X_test,y_train,y_test=train_test_split(x,y, train_size=0.8, random_state=42)
```

5.3. MODEL BUILDING:

This section will discuss in detail on the proposed methodology. The method consists of 2 different approaches which will be CNN and Transfer Learning for crack detection and classification. First, we have implemented Convolution Neural Network (CNN) model.

5.3.1. CNN Model:

Convolutional neural networks (CNNs) remain a special type of multilayer neural networks. CNNs differ from neural networks in the shape and function of the layers.

The layers of known neural networks are one-dimensional, and the neurons in this layer are completely interconnected. On the other hand, the layers in CNNs are three-dimensional in terms of width, height and depth parameters.

A CNN is a deep learning algorithm that can take an input image, attach importance to various directions/objects in the image and distinguish one from the others. The required pre-processing on CNN is much lower compared to other classification algorithms. CNNs are structures designed to take images as inputs and are used effectively in computer vision. CNN consists of one or more convoluted layer and one or more fully bonded layers, such as a standard multilayer neural network which will get differ from problem to problem.

Architecture:

We have implemented our own CNN model by configuring the layers of it.

It consists the following layers

- ✓ 2D Convolution layer with **"relu"** activation function
- ✓ 2D Max Pool layer
- ✓ 2D Convolution layer with **"relu"** activation function

- ✓ 2D Max Pool layer
- ✓ 2D Convolution layer with **“relu”** activation function
- ✓ 2D Max Pool layer
- ✓ Then we added 2 dense layers, one with **“relu”** activation function and the other with **“softmax”** along with a dropout layer

CNN Model

```
[ ]  model = Sequential()
     model.add(Conv2D(64,3,padding="same", activation="relu", input_shape = x.shape[1:]))
     model.add(MaxPool2D())

     model.add(Conv2D(64, 3, padding="same", activation="relu"))
     model.add(MaxPool2D())

     model.add(Conv2D(128, 3, padding="same", activation="relu"))
     model.add(MaxPool2D())

     model.add(Flatten())
     model.add(Dense(256,activation="relu"))
     model.add(Dropout(0.5))
     model.add(BatchNormalization())
     model.add(Dense(2, activation="softmax"))

     model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 120, 120, 64)      640

 max_pooling2d (MaxPooling2D  (None, 60, 60, 64)       0
 )

 conv2d_1 (Conv2D)           (None, 60, 60, 64)        36928

 max_pooling2d_1 (MaxPooling  (None, 30, 30, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 30, 30, 128)       73856

 max_pooling2d_2 (MaxPooling  (None, 15, 15, 128)      0
 2D)

 flatten (Flatten)           (None, 28800)             0

 dense (Dense)               (None, 256)               7373056

 dropout (Dropout)           (None, 256)               0

 batch_normalization (BatchN  (None, 256)              1024
 ormalization)

 dense_1 (Dense)             (None, 2)                 514

=================================================================
Total params: 7,486,018
Trainable params: 7,485,506
Non-trainable params: 512
_________________________________________________________________
```

Model Training:

- We have trained our implemented model with 15 epochs

Model Training

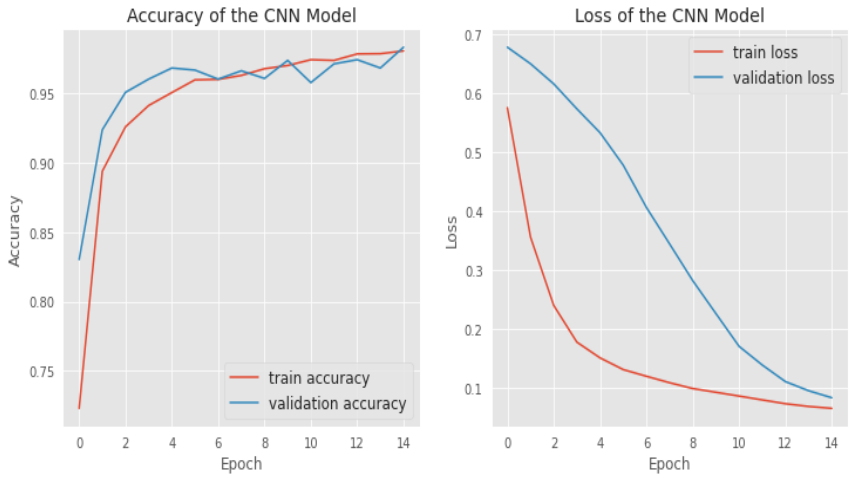
```
opt = Adam(lr=1e-5)
model.compile(loss="sparse_categorical_crossentropy", optimizer=opt, metrics=["accuracy"])
history = model.fit(X_train, y_train, epochs = 15, batch_size = 128, validation_split = 0.25, verbose=1)
```

```
Epoch 1/15
47/47 [==============================] - 24s 253ms/step - loss: 0.5743 - accuracy: 0.7230 - val_loss: 0.6770 - val_accuracy: 0.8302
Epoch 2/15
47/47 [==============================] - 9s 200ms/step - loss: 0.3547 - accuracy: 0.8937 - val_loss: 0.6489 - val_accuracy: 0.9236
Epoch 3/15
47/47 [==============================] - 9s 201ms/step - loss: 0.2390 - accuracy: 0.9257 - val_loss: 0.6148 - val_accuracy: 0.9505
Epoch 4/15
47/47 [==============================] - 9s 201ms/step - loss: 0.1766 - accuracy: 0.9410 - val_loss: 0.5724 - val_accuracy: 0.9600
Epoch 5/15
47/47 [==============================] - 9s 201ms/step - loss: 0.1499 - accuracy: 0.9504 - val_loss: 0.5320 - val_accuracy: 0.9680
Epoch 6/15
47/47 [==============================] - 9s 201ms/step - loss: 0.1301 - accuracy: 0.9595 - val_loss: 0.4769 - val_accuracy: 0.9665
Epoch 7/15
47/47 [==============================] - 9s 201ms/step - loss: 0.1187 - accuracy: 0.9599 - val_loss: 0.4052 - val_accuracy: 0.9600
Epoch 8/15
47/47 [==============================] - 9s 201ms/step - loss: 0.1077 - accuracy: 0.9627 - val_loss: 0.3433 - val_accuracy: 0.9660
Epoch 9/15
47/47 [==============================] - 9s 201ms/step - loss: 0.0979 - accuracy: 0.9675 - val_loss: 0.2810 - val_accuracy: 0.9605
Epoch 10/15
47/47 [==============================] - 9s 200ms/step - loss: 0.0915 - accuracy: 0.9699 - val_loss: 0.2251 - val_accuracy: 0.9735
Epoch 11/15
47/47 [==============================] - 9s 201ms/step - loss: 0.0851 - accuracy: 0.9740 - val_loss: 0.1695 - val_accuracy: 0.9575
Epoch 12/15
47/47 [==============================] - 9s 201ms/step - loss: 0.0786 - accuracy: 0.9735 - val_loss: 0.1378 - val_accuracy: 0.9710
Epoch 13/15
47/47 [==============================] - 9s 201ms/step - loss: 0.0721 - accuracy: 0.9782 - val_loss: 0.1096 - val_accuracy: 0.9740
Epoch 14/15
47/47 [==============================] - 9s 200ms/step - loss: 0.0675 - accuracy: 0.9783 - val_loss: 0.0942 - val_accuracy: 0.9680
Epoch 15/15
47/47 [==============================] - 9s 201ms/step - loss: 0.0643 - accuracy: 0.9803 - val_loss: 0.0823 - val_accuracy: 0.9830
```

- Plotting the Accuracy and Loss of the CNN model

```
plt.figure(figsize=(12, 12))
plt.style.use('ggplot')
plt.subplot(2,2,1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Accuracy of the CNN Model')
plt.ylabel('Accuracy', fontsize=12)
plt.xlabel('Epoch', fontsize=12)
plt.legend(['train accuracy', 'validation accuracy'], loc='lower right', prop={'size': 12})

plt.subplot(2,2,2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Loss of the CNN Model')
plt.ylabel('Loss', fontsize=12)
plt.xlabel('Epoch', fontsize=12)
plt.legend(['train loss', 'validation loss'], loc='best', prop={'size': 12})
```

```
<matplotlib.legend.Legend at 0x7f620665a6d0>
```



Model Testing

```
[ ]  scores = model.evaluate(X_test, y_test, verbose=0)
     print("CNN score: %.2f%%" % (scores[1]*100))
     print("CNN Error: %.2f%%" % (100-scores[1]*100))
```

```
CNN score: 97.95%
CNN Error: 2.05%
```

5.3.2.Transfer Learning:

In This we have implemented 4 pre-trained models:

- VGG16
- ResNet50
- Xception
- MobileNet

```
[ ]  import os
     import glob
     import numpy as np
     import pandas as pd
     import seaborn as sns
     import matplotlib.pyplot as plt
     from sklearn.model_selection import train_test_split
     from tensorflow.keras.preprocessing.image import ImageDataGenerator
     from sklearn.metrics import confusion_matrix, classification_report
     from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,GlobalAveragePooling2D, Dropout
     from tensorflow.keras.models import Sequential,Model
     from tensorflow.keras.callbacks import Callback,EarlyStopping
```

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

```
Mounted at /content/drive
```

```
[ ]  path = '/content/drive/MyDrive/DeepLearningProject/SurfaceCracks/'
     path_all = list(glob.glob(path+'**/*jpg'))
     path_Negative = path + 'Negative/*.jpg'
     path_Positive = path + 'Positive/*.jpg'
```

```
[ ] labels = list(map(lambda x:os.path.split(os.path.split(x)[0])[1], path_all))
    file_Path = pd.Series(path_all, name='File_Path').astype(str)
    labels = pd.Series(labels, name='Label')
    data = pd.concat([file_Path, labels], axis=1)
    data = data.sample(frac=1).reset_index(drop=True)
    data.head()
```

	File_Path	Label
0	/content/drive/MyDrive/DeepLearningProject/Sur...	Positive
1	/content/drive/MyDrive/DeepLearningProject/Sur...	Positive
2	/content/drive/MyDrive/DeepLearningProject/Sur...	Positive
3	/content/drive/MyDrive/DeepLearningProject/Sur...	Positive
4	/content/drive/MyDrive/DeepLearningProject/Sur...	Negative

```
fig, axes = plt.subplots(nrows=2, ncols=3,figsize=(10,5), subplot_kw={'xticks':[], 'yticks':[]})
for i, ax in enumerate(axes.flat):
    ax.imshow(plt.imread(data.File_Path[i]))
    ax.set_title(data.Label[i])
plt.tight_layout()
plt.show()
```


```
[ ] train_df, test_df = train_test_split(data, test_size=0.2, random_state=42)
```

```
[ ]  def gen(pre,train,test):
         train_datagen = ImageDataGenerator(
                             preprocessing_function=pre,
                             validation_split=0.2)
         test_datagen = ImageDataGenerator(
                             preprocessing_function=pre)

         train_gen = train_datagen.flow_from_dataframe(
             dataframe=train,
             x_col='File_Path',
             y_col='Label',
             target_size=(100,100),
             class_mode='categorical',
             batch_size=64,
             shuffle=True,
             seed=42
             )
         valid_gen = train_datagen.flow_from_dataframe(
             dataframe=train,
             x_col='File_Path',
             y_col='Label',
             target_size=(100,100),
             class_mode='categorical',
             batch_size=64,
             shuffle=False,
             seed=42
         )
         test_gen = test_datagen.flow_from_dataframe(
             dataframe=test,
             x_col='File_Path',
             y_col='Label',
             target_size=(100,100),
             color_mode='rgb',
             class_mode='categorical',
             batch_size=64,
             shuffle=False
         )
         return train_gen, valid_gen, test_gen
```

```
[ ]  def func(name_model):
         pre_model = name_model(input_shape=(100,100, 3),
                                include_top=False,
                                weights='imagenet',
                                pooling='avg')
         pre_model.trainable = False
         inputs = pre_model.input

         x = Dense(64, activation='relu')(pre_model.output)
         x = Dense(64, activation='relu')(x)
         outputs = Dense(2, activation='softmax')(x)

         model = Model(inputs=inputs, outputs=outputs)
         model.compile(loss = 'categorical_crossentropy',optimizer='Adam',metrics=['accuracy'])
         my_callbacks  = [EarlyStopping(monitor='val_loss',
                                        min_delta=0,
                                        patience=1,
                                        mode='auto')]
         return model, my_callbacks
```

```
[ ]  def plot(history,df_test,test_gen,train_gen):
         # Plotting Accuracy, val_accuracy, loss, val_loss
         fig, ax = plt.subplots(1, 2, figsize=(10, 3))
         ax = ax.ravel()

         for i, met in enumerate(['accuracy', 'loss']):
             ax[i].plot(history.history[met])
             ax[i].plot(history.history['val_' + met])
             ax[i].set_title('Model {}'.format(met))
             ax[i].set_xlabel('epochs')
             ax[i].set_ylabel(met)
             ax[i].legend(['train', 'val'])

         # Predict Data Test
         pred = model.predict(test_gen )
         pred = np.argmax(pred,axis=1)
         labels = (train_gen.class_indices)
         labels = dict((v,k) for k,v in labels.items())
         pred = [labels[k] for k in pred]

         # Classification report
         cm=confusion_matrix(df_test.Label,pred)
         clr = classification_report(df_test.Label, pred, target_names=["NEGATIVE", "POSITIVE"])
         print(clr)
         # Display 6 picture of the dataset with their labels
         fig, axes = plt.subplots(nrows=2, ncols=3, figsize=(12, 8),
                                  subplot_kw={'xticks': [], 'yticks': []})

         for i, ax in enumerate(axes.flat):
             ax.imshow(plt.imread(df_test.File_Path.iloc[i+1]))
             ax.set_title(f"True: {df_test.Label.iloc[i+1]}\nPredicted: {pred[i+1]}")
         plt.tight_layout()
         plt.show()

         return history
```

```
[ ]  def result_test(test,model_use):
         results = model_use.evaluate(test, verbose=0)

         print("Test Loss: {:.5f}".format(results[0]))
         print("Test Accuracy: {:.2f}%".format(results[1] * 100))

         return results
```

1. VGG16

```
[ ]  from tensorflow.keras.applications import VGG16
     from tensorflow.keras.applications.vgg16 import preprocess_input
     model, callback=func(VGG16)

     vgg_pre=preprocess_input
     train_gen_VGG, valid_gen_VGG, test_gen_VGG = gen(vgg_pre,train_df,test_df)
```

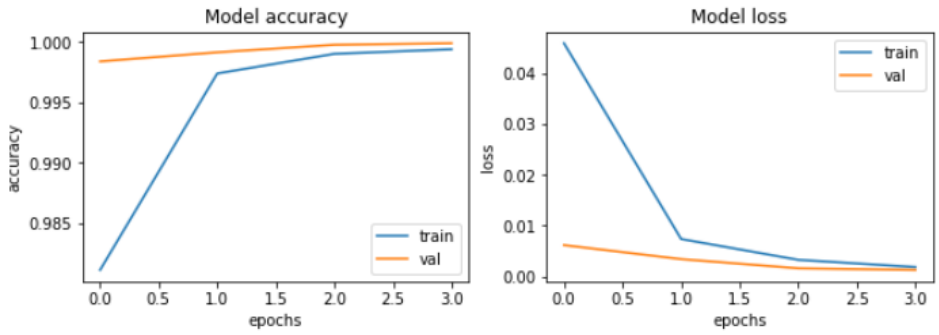
```
Found 8006 validated image filenames belonging to 2 classes.
Found 8006 validated image filenames belonging to 2 classes.
Found 2002 validated image filenames belonging to 2 classes.
```

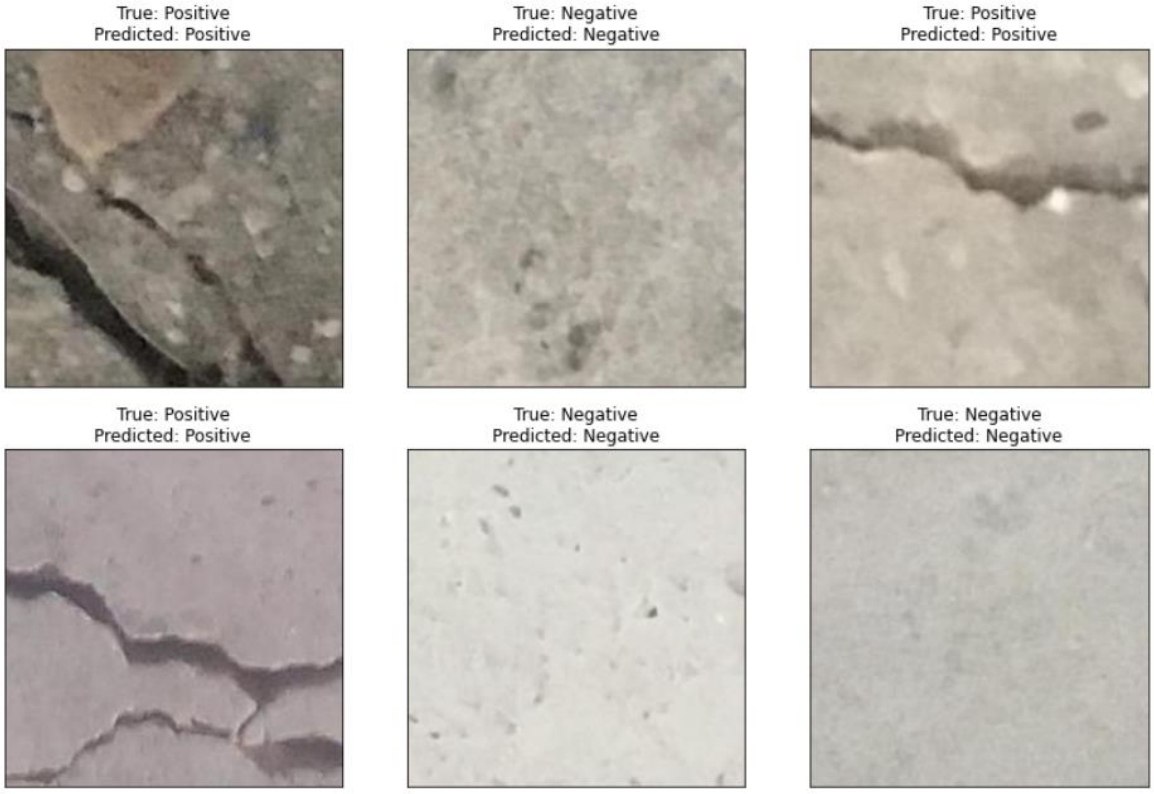
```
[ ]  history = model.fit(
         train_gen_VGG,
         validation_data=valid_gen_VGG,
         epochs=4,
         callbacks=callback
     )
```

```
Epoch 1/4
126/126 [==============================] - 39s 291ms/step - loss: 0.0459 - accuracy: 0.9811 - val_loss: 0.0061 - val_accuracy: 0.9984
Epoch 2/4
126/126 [==============================] - 37s 293ms/step - loss: 0.0073 - accuracy: 0.9974 - val_loss: 0.0034 - val_accuracy: 0.9991
Epoch 3/4
126/126 [==============================] - 37s 291ms/step - loss: 0.0032 - accuracy: 0.9990 - val_loss: 0.0016 - val_accuracy: 0.9998
Epoch 4/4
126/126 [==============================] - 36s 290ms/step - loss: 0.0018 - accuracy: 0.9994 - val_loss: 0.0013 - val_accuracy: 0.9999
```

```
[ ]  history=plot(history,test_df,test_gen_VGG,train_gen_VGG)
```

```
                precision    recall  f1-score   support

      NEGATIVE       0.99      1.00      1.00       986
      POSITIVE       1.00      1.00      1.00      1016

      accuracy                           1.00      2002
     macro avg       1.00      1.00      1.00      2002
  weighted avg       1.00      1.00      1.00      2002
```






```
[ ]  result = result_test(test_gen_VGG,model)
```

```
Test Loss: 0.00680
Test Accuracy: 99.75%
```

2. ResNet50

```
[ ]  from tensorflow.keras.applications import ResNet50
     from tensorflow.keras.applications.resnet50 import preprocess_input
     ResNet50_model, callback=func(ResNet50)

     RestNet_pre=preprocess_input
     train_gen_RestNet, valid_gen_RestNet, test_gen_RestNet = gen(RestNet_pre,train_df,test_df)
```

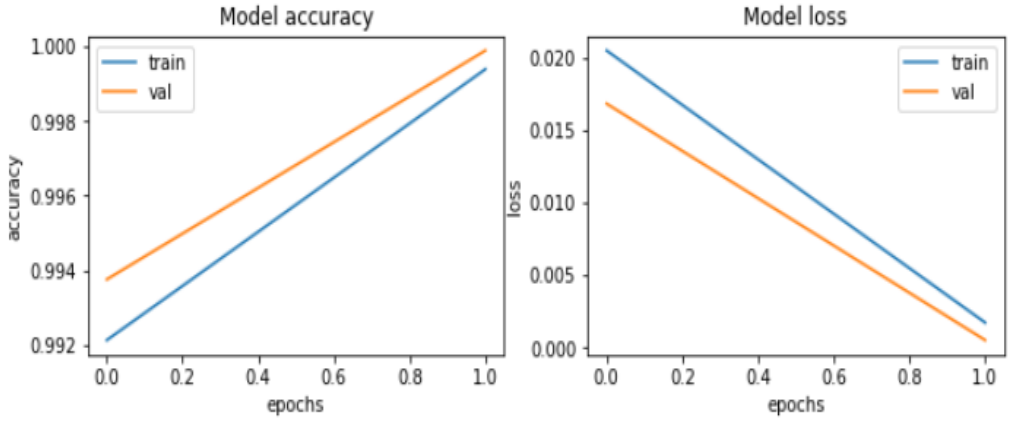
```
Found 8006 validated image filenames belonging to 2 classes.
Found 8006 validated image filenames belonging to 2 classes.
Found 2002 validated image filenames belonging to 2 classes.
```

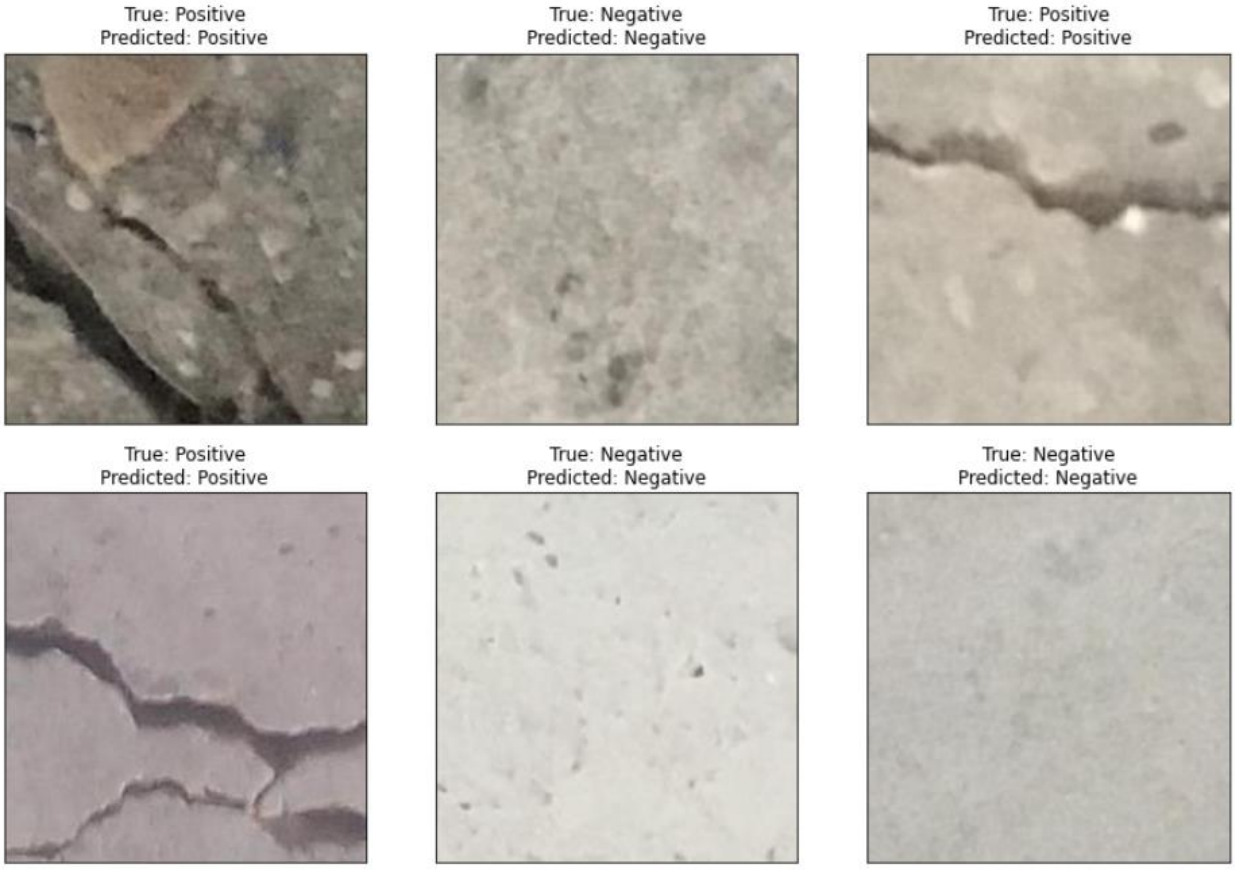
```
[ ]  history = ResNet50_model.fit(
         train_gen_RestNet,
         validation_data=valid_gen_RestNet,
         epochs=4,
         callbacks=callback
     )
```

```
Epoch 1/2
126/126 [==============================] - 41s 304ms/step - loss: 0.0205 - accuracy: 0.9921 - val_loss: 0.0168 - val_accuracy: 0.9938
Epoch 2/2
126/126 [==============================] - 37s 293ms/step - loss: 0.0017 - accuracy: 0.9994 - val_loss: 4.9395e-04 - val_accuracy: 0.9999
```

```
[ ]  history_ResNet=plot(history,test_df,test_gen_RestNet,train_gen_RestNet)
```

	precision	recall	f1-score	support
NEGATIVE	0.99	1.00	1.00	986
POSITIVE	1.00	1.00	1.00	1016
accuracy			1.00	2002
macro avg	1.00	1.00	1.00	2002
weighted avg	1.00	1.00	1.00	2002





```
[ ]  result_ResNet = result_test(test_gen_RestNet,ResNet50_model)
```

```
Test Loss: 0.00565
Test Accuracy: 99.75%
```

3. Xception

```
[ ] from tensorflow.keras.applications import Xception
    from tensorflow.keras.applications.xception import preprocess_input
    Xception_model, callback=func(Xception)

    Xception_pre=preprocess_input
    train_gen_Xception, valid_gen_Xception, test_gen_Xception = gen(Xception_pre,train_df,test_df)
```

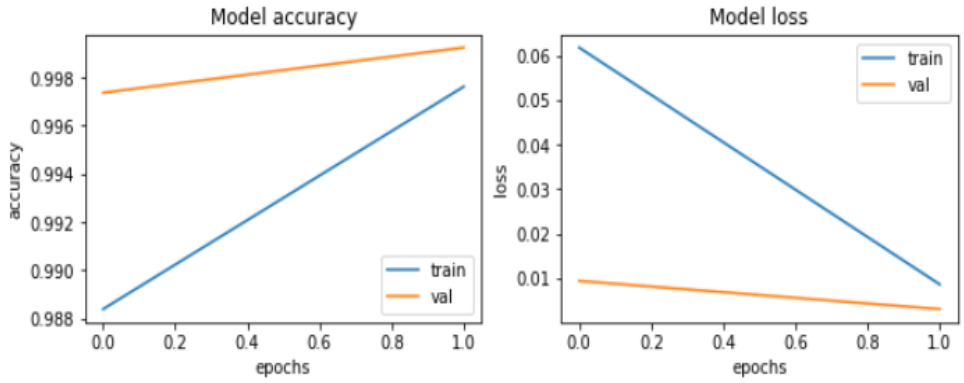
```
Found 8006 validated image filenames belonging to 2 classes.
Found 8006 validated image filenames belonging to 2 classes.
Found 2002 validated image filenames belonging to 2 classes.
```

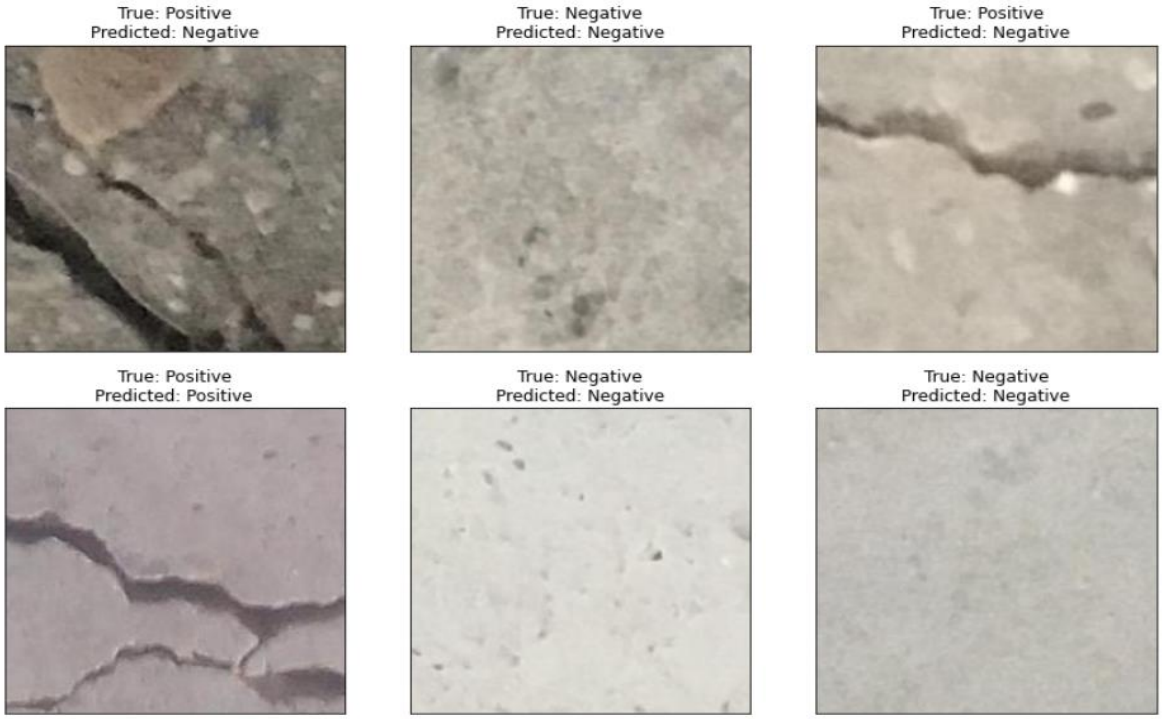
```
[ ] history = Xception_model.fit(
        train_gen_Xception,
        validation_data=valid_gen_Xception,
        epochs=4,
        callbacks=callback
    )
```

```
Epoch 1/2
126/126 [==============================] - 39s 291ms/step - loss: 0.0617 - accuracy: 0.9884 - val_loss: 0.0094 - val_accuracy: 0.9974
Epoch 2/2
126/126 [==============================] - 36s 285ms/step - loss: 0.0086 - accuracy: 0.9976 - val_loss: 0.0031 - val_accuracy: 0.9993
```

```
[ ] history_Xception=plot(history,test_df,test_gen_Xception,train_gen_Xception)
```

```
              precision    recall  f1-score   support

    NEGATIVE       0.56      1.00      0.71       986
    POSITIVE       1.00      0.22      0.36      1016

    accuracy                           0.61      2002
   macro avg       0.78      0.61      0.54      2002
weighted avg       0.78      0.61      0.54      2002
```






```
[ ]  result_Xception = result_test(test_gen_Xception,Xception_model)
```

```
Test Loss: 0.01162
Test Accuracy: 99.65%
```

5. MobileNet

```
[ ] from tensorflow.keras.applications import MobileNet
    from tensorflow.keras.applications.mobilenet import preprocess_input
    MobileNet_model, callback=func(MobileNet)

    MobileNet_pre=preprocess_input
    train_gen_MobileNet, valid_gen_MobileNet, test_gen_MobileNet = gen(MobileNet_pre,train_df,test_df)
```

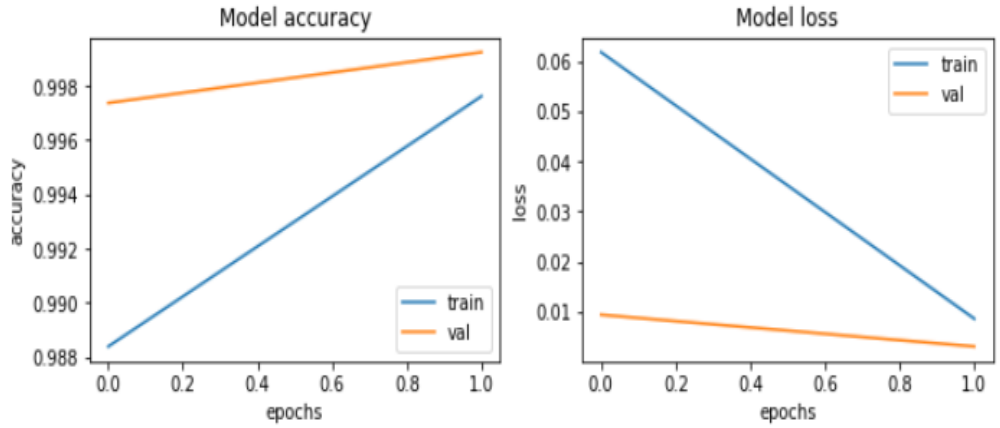
```
WARNING:tensorflow:`input_shape` is undefined or non-square, or `rows` is not in [128, 160, 192, 224]. Weights for input shape (224, 224) will be loaded as the default.
Found 8006 validated image filenames belonging to 2 classes.
Found 8006 validated image filenames belonging to 2 classes.
Found 2002 validated image filenames belonging to 2 classes.
```

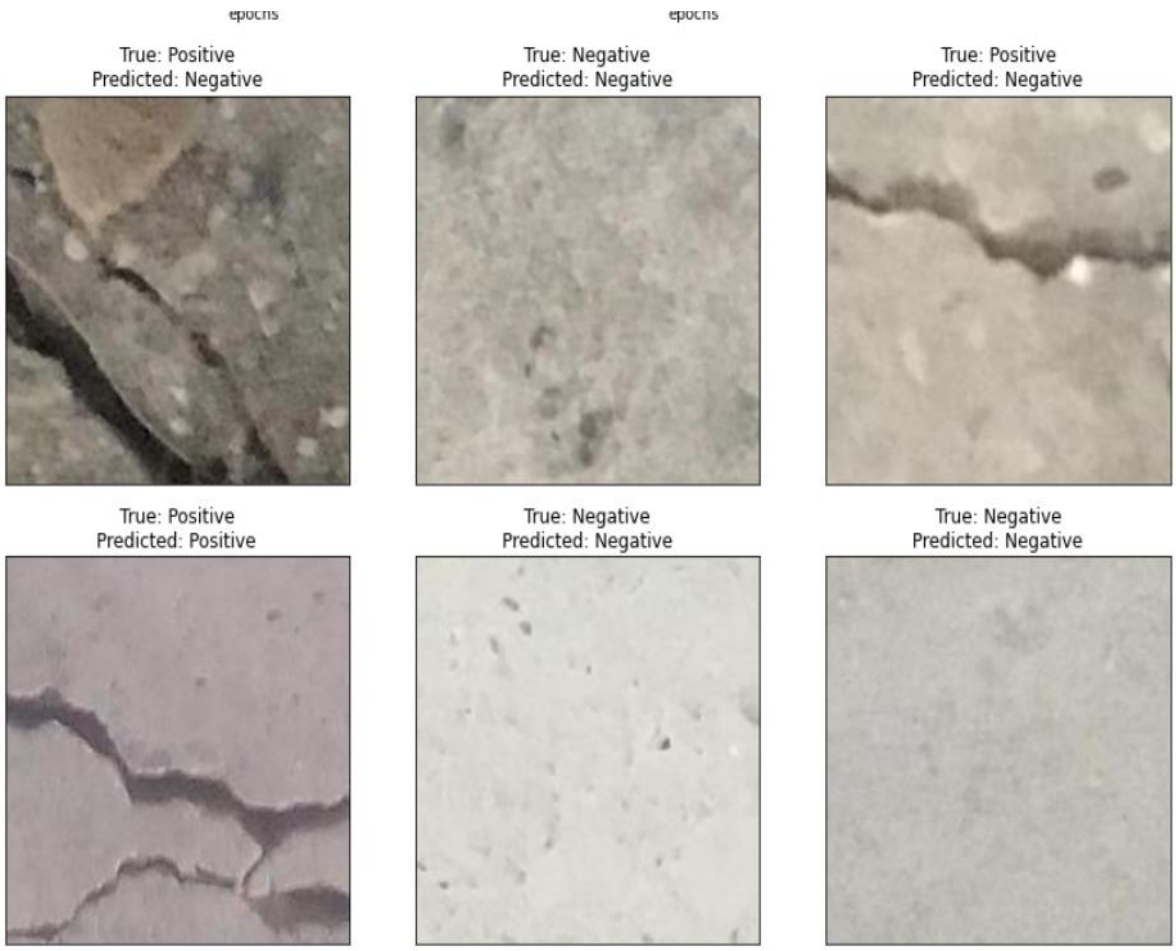
```
[ ] history_MobileNet = MobileNet_model.fit(
        train_gen_MobileNet,
        validation_data=valid_gen_MobileNet,
        epochs=4,
        callbacks=callback
    )
```

```
Epoch 1/4
126/126 [==============================] - 37s 281ms/step - loss: 0.0562 - accuracy: 0.9773 - val_loss: 0.0088 - val_accuracy: 0.9969
Epoch 2/4
126/126 [==============================] - 34s 271ms/step - loss: 0.0045 - accuracy: 0.9988 - val_loss: 0.0011 - val_accuracy: 0.9998
Epoch 3/4
126/126 [==============================] - 34s 269ms/step - loss: 0.0016 - accuracy: 0.9994 - val_loss: 4.8138e-04 - val_accuracy: 1.0000
Epoch 4/4
126/126 [==============================] - 34s 271ms/step - loss: 5.3635e-04 - accuracy: 1.0000 - val_loss: 2.2739e-04 - val_accuracy: 1.0000
```

```
[ ]  history=plot(history,test_df,test_gen_MobileNet,train_gen_MobileNet)
```

	precision	recall	f1-score	support
NEGATIVE	0.56	1.00	0.71	986
POSITIVE	1.00	0.22	0.36	1016
accuracy			0.61	2002
macro avg	0.78	0.61	0.54	2002
weighted avg	0.78	0.61	0.54	2002






```
[ ]  result_MobileNet = result_test(test_gen_MobileNet,MobileNet_model)
```

```
Test Loss: 0.00195
Test Accuracy: 99.95%
```

Final Results

```
[ ]  output = pd.DataFrame({'Model':['VGG16','ResNet50','Xception','MobileNet'],
                            'Accuracy':[result[1], result_ResNet[1], result_Xception[1], result_MobileNet[1]]})
```

```
[ ]  output
```

	Model	Accuracy
0	VGG16	0.997503
1	ResNet50	0.997503
2	Xception	0.996503
3	MobileNet	0.999501

6. RESULT AND CONCLUSION:

After running all the models these are the accuracy results obtained for models:

CNN model: 0.9795 (97.95%)

VGG16: 0.997503 (99.75%)

ResNet50: 0.997503 (99.75%)

Xception: 0.996503 (99.65%)

MobileNet: 0.99501 (99.95%)

So by observing above accuracies we can conclude that MobileNet model is the best model for predicting the concrete cracks followed by VGG16, ResNet50, Xception. CNN model is the least with 97.95% accuracy but this is also very high score. So overall we can use any of these models as the accuracy is very high.

7. REFERENCES:

We have referred the research paper named **"Road Crack Detection Using Deep Convolutional Neural Network"** for our analysis:

Link for the research paper:

DOI: 10.17632/5y9wdsg2zt.2

https://doi.org/10.1016/j.autcon.2018.11.028

https://sci-hub.hkvisa.net/https://data.mendeley.com/datasets/5y9wdsg2zt/2

8. FUTURE ENHANCHEMENT:

In future we will explore and apply this model to various other surfaces like metal, rocks etc. so that it will be easy to find structural defects for the structures built with them and also, we will apply this model for bones as well so that it helps for many species to see if there are any cracks or not and also, we will try to improve and explore other efficient methods to find the detection that are faster and also accurate too.