19Z610 - MACHINE LEARNING LABORATORY

POTHOLE DETECTION USING MACHINE LEARNING

PSG COLLEGE OF TECHNOLOGY

TEAM - 8

Team Members

20Z216 - Ganapathi K
20Z224 - Karan H
20Z246 - Sarath Ra
21Z433 - Glory C J
21Z435 - Sundarsree

PROBLEM STATEMENT:

Over the past few years, India has seen exponential growth in the number of vehicles on the road. As a result, the increase in the number of vehicles on the road gave rise to the number of road accidents. According to a study, one fatal road accident occurs every 5 minutes in the country, and 8 die on roads every hour. This has become a major concern in the country. One of the primary causes of these road accidents is the management and maintenance of the roads.

Potholes on roads can cause serious accidents, and any vehicle traveling at some decent speed can lose its track due to them. In the case of four-wheeler vehicles, potholes can cause severe damage to wheels and tires. More specifically, when it comes to two-wheelers like motorbikes, these vehicles are more prone to accidents due to potholes as the tendency to cause imbalance is very high and can lead to fatalities.

Concerned by the increasing number of accidents caused by potholes, the government of India has started initiatives and campaigns to install cameras in the most accident-prone cities and capture the feed to detect the potholes. Apply image processing techniques to process the images and detect the potholes. This will assist the government to identify the extreme potholes depending on their size and handle them as the earliest.

DATASET DESCRIPTION:

The main objective of this project is to detect if the image contains pothole(s) or not and identify them based on their size variations.
Since we are using images as the dataset there are no attributes. However, we can classify the images based on the number of potholes as images with

- Single Pothole
- Multiple Potholes
- No Pothole

- ❖ The dataset includes images with different scales, backgrounds, and under different weather conditions.
- ❖ The images should be labeled to segregate them into positive and negative cases.
- ❖ It also covers different types of roads, different types of potholes, lighting conditions, and angles so that the model can be trained to work in real life conditions.
- ❖ Moreover, the dataset should be balanced with an equal number of images for positive and negative examples.

This dataset was exported via kaggle.com
Kaggle allows users to find datasets they want to use in building AI models. It provides features like

- collect & organize images
- understand unstructured image data
- annotate, and create datasets

The dataset includes 680 images stored in 2 separate folders Normal and Potholes which contain images with and without pothole respectively. The entire dataset is stored in Google Drive.
Images

The following preprocessing was applied to each image:

- Auto-orientation of pixel data (with EXIF-orientation stripping)
- Resize to 640 x 640 (Stretch)

The following augmentation was applied to create 3 versions of each source image:

- Horizontal Flipping -
- Clockwise Rotation by 30 degree
- Anti-clockwise Rotation by 30 degree
- Reshaping.

We have also used an ImageDataGenerator object to preprocess and augment images for training a deep learning model. The code sets the rescale parameter to 1/255, which normalizes the pixel values to a range between 0 and 1. It also sets up the

- Shear range
- Zoom range
- Horizontal flip parameters

which enable random transformation of the images to add variability to the training dataset.

Fig.1:potholes at different conditions of roads with different angles and lightings.

METHODOLOGY USED:

A classification problem is a type of machine learning problem where the goal is to assign a label or class to each input data point. In the case of pothole detection, the goal is to classify each input image as either containing a pothole or not. This is a classification problem because the model must assign a binary label based on the input image.

0 - indicates Image does not contain any pothole.

1 - indicates Image contains one or more potholes.

The methodology used to detect potholes using machine learning typically involves the following steps:

1. Data collection: The project collects images with potholes and without potholes.
2. Data preprocessing: The collected data is preprocessed to remove noise and irrelevant information from the image. This step involves Labeling data, filtering, reshaping and splitting the data.
3. Image Augmentation: As one of the challenges that we faced is we don't have adequate images with and without potholes hence this technique is used to artificially expand the data set. Horizontal Flipping and Clockwise and Anticlockwise rotation by 30 degrees has been implemented.

   We have also used an ImageDataGenerator object to preprocess and augment images for training a deep learning model. The code sets the rescale parameter to 1/255, which normalizes the pixel values to a range between 0 and 1. It also sets up the Shear range,Zoom range and Horizontal flip parameters
4. Dimensionality Reduction: Since images are of high resolution, the input matrix has a high column dimension. So, dimensionality reduction may be useful in this situation. Principal component analysis (PCA) will be employed.

   Principal Component Analysis (PCA) is a statistical method used to reduce the number of variables in a dataset while still preserving the important information.

   PCA works by transforming the input data into a new coordinate system in which the variables are uncorrelated. This is done by finding the eigenvectors of the covariance matrix of the original data, which are then used to transform the data into the new coordinate system. The eigenvectors are sorted in order of decreasing importance, and the most important ones are used to transform the data. The result is a much lower-dimensional representation of the data which still contains the important information. In the context of pothole detection, PCA can be used to reduce the number of input features while still preserving the most important information about the image.

This can be especially useful for large images, since the number of input features can be very large. By reducing the number of input features, the model can detect potholes more efficiently by focusing on the most important information.

5. Feature extraction: The preprocessed image data is transformed into a format that can be fed into the following machine learning models. The models implemented are :
    - Support vector machine (SVM) - PCA-SVM
    - Logistic regression - Baseline Model
    - Convolutional neural network (CNN).
    - Random Forest
    - Convolutional neural network (CNN) with VGG transfer learning.
6. Model training: The above-mentioned models are designed,trained on the extracted features and saved as a '.h5' file to accurately predict the presence of potholes in the image.
7. Model evaluation: The trained models are evaluated on a set of test data to measure its performance in accurately predicting the presence of potholes. Metrics like accuracy, precision and recall are used to analyze the model's performance.

   For a given test image, the model predicts if it contains any potholes or Not. If Yes it prints the pothole and then proceeds to mark the area in the image(with green Rectangle) that contains one or more potholes and then displays the marked image.

   Plots such as Accuracy vs Loss, Testing accuracy vs loss and Train Loss vs Value Loss are drawn to get a better understanding of the model’s performance.
8. Model optimization: The machine learning models are optimized to improve their performance by adjusting various parameters such as learning rate, activation functions, and dropout rate.
9. Contour Mapping: This is done to mark the area in the images containing potholes. Feature detection techniques such as Canny Edge Detection and Hough Line Transformation is used to detect the region in the image containing potholes and mark them.

MODELS USED:

CNN with VGG:

VGG stands for Visual Geometry Group; it is a standard deep Convolutional Neural Network (CNN) architecture with multiple layers. The “deep” refers to the number of layers with VGG-16 or VGG-19 consisting of 16 and 19 convolutional layers.

The VGG architecture is the basis of ground-breaking object recognition models. Developed as a deep neural network, the VGGNet also surpasses baselines on many tasks and datasets beyond ImageNet. Moreover, it is now still one of the most popular image recognition architectures.

The number of filters that we can use doubles on every step or through every stack of the convolution layer. This is a major principle used to design the architecture of the VGG16 network. One of the crucial downsides of the VGG16 network is that it is a huge network, which means that it takes more time to train its parameters. Because of its depth and number of fully connected layers, the VGG16 model is more than 533MB. This makes implementing a VGG network a time-consuming task.

In the context of pothole detection, VGG can be used as a feature extractor to extract useful features from images that can help identify potholes. This is typically done by passing the image through the VGG network, which consists of multiple layers of convolutional and pooling operations.

For initializing the CNN:

- Convolution

  Convolutional layers are used to extract features from the input image. These layers apply a set of filters (also called kernels or weights) to the image, which essentially scan the image and highlight patterns and features that are important for classification. The output of each convolutional layer is a set of feature maps that encode different aspects of the input image.

  The VGG architecture typically consists of a series of convolutional layers followed by a few fully connected layers. The convolutional layers are usually stacked on top of each other, with each layer having a larger number of filters than the previous layer. This allows the network to capture increasingly complex and abstract features as it processes the input image.

- Pooling

  Pooling is a downsampling operation that is often used in convolutional neural networks, including VGG, to reduce the spatial size of feature maps while preserving the important features. In VGG, max pooling is used after each set of convolutional layers to downsample the feature maps.

The pooling operations in VGG can help to reduce the spatial size of the feature maps and make the network more efficient at processing the input images. However, it is also important to consider the trade-off between spatial resolution and the ability to accurately detect small or subtle potholes. Depending on the specifics of the task and the available data, it may be necessary to adjust the pooling operations or use additional techniques, such as upsampling or skip connections, to maintain the necessary level of detail in the feature maps

- Flattening

  Flattening is a process in VGG and other convolutional neural networks where the output feature maps from the last convolutional layer are reshaped into a one-dimensional vector that can be fed into the fully connected layers. In VGG, this is typically done before the fully connected layers, which perform the final classification or regression task.

  In the context of pothole detection, flattening allows the VGG network to efficiently process input images of different sizes and extract important features that are relevant to the task. The flattened representations are then used by the fully connected layers to make a prediction about whether the input image contains a pothole or not, based on the learned patterns and features in the flattened feature vector.

- Full Connection

  The purpose of the fully connected layers is to take the high-dimensional feature representation learned by the convolutional layers and use it to make a prediction about the input image. In the case of pothole detection, this prediction is whether the input image contains a pothole or not.

  In the context of pothole detection, the fully connected layers in VGG are an essential component of the network, as they allow the learned features to be used to make a prediction about the presence and severity of potholes in the input image.

- Output Layer

  The output layer in VGG is responsible for producing the final prediction based on the learned features from the convolutional layers and the fully connected layers. The weights of the output layer are also learned during training using backpropagation and gradient descent optimization, just like the weights in the other layers of the network.

  In the context of pothole detection, the output layer typically consists of a single neuron with a sigmoid activation function. The output of this neuron represents the probability that the input image contains a pothole. A threshold value of 0.5 is chosen to determine the final classification decision, i.e if the probability is greater than 0.5, the input image is classified as having a pothole otherwise it is classified as not having a pothole.

SVM:

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called support vectors, and hence the algorithm is termed as Support Vector Machine.

Support Vector Machines (SVMs) are a type of supervised machine learning algorithm that can be used for various applications, including image recognition and classification. In this context, they can be used to detect potholes in images.

The working of an SVM is based on training a model with labeled examples of potholes and other features in the image. This model is then used to make predictions on new images. The algorithm looks for patterns in the image data and uses that information to classify each image as containing a pothole or not.

The model is trained by feeding it a set of labeled images. Each image is labeled with either “pothole” or “not pothole.” The model then looks for patterns that distinguish between the two classes. For example, it might look for shapes and textures that are associated with potholes. After it has identified these features, it can then classify a new image as containing a pothole or not.

In addition, the SVM model can also be used to detect the size and shape of a pothole. This is done by training the model to identify the shape and size of potholes in the training dataset. The model can then be used to detect the size and shape of a pothole in an image. This can be an important factor in determining the severity of the pothole.

- SVM with PCA performance:

The model was trained with a regularization parameter of 2.9 and a Radial Basis Function kernel, achieving an accuracy of 88.2% on the test set.

- Decision Tree:

Decision tree type of algorithm that uses a hierarchical structure of nodes to classify whether a given input corresponds to a pothole or not. The tree starts with a root node that considers the most significant feature of the input and splits the data into two or more branches.

Each branch leads to a child node that evaluates the next most important feature, creating further splits until a leaf node is reached, which provides the final classification.

The decision tree is trained on a dataset of labeled examples of potholes and non-potholes to learn the most informative features and their thresholds for accurate classification.

- Regularization:

  Regularization is a technique used in machine learning to prevent overfitting and improve the generalization performance of a model. In the context of pothole detection, regularization can be applied to a logistic regression or other classifier model to improve its performance on new, unseen data.

Random forest:

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model.

As the name suggests, "Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset." Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.

It is based on the idea of combining multiple weak decision trees, which in turn produces a strong decision tree. This strong decision tree is then used to make a prediction.

The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.

In the context of detecting potholes in an image, each decision tree in the random forest would be trained on a different set of features extracted from the image. These features can include edges, shapes, color, texture, and other characteristics of the image. Each decision tree would then be used to make a prediction based on the features it was trained on. The final prediction would be based on the votes of all the decision trees in the forest. This technique can be used to detect potholes in an image with high accuracy by taking into account the various characteristics of the image.

Logistic Regression - Baseline model:

Logistic regression is a type of statistical model used for classification problems, where the output is a binary outcome (e.g. yes/no, 0/1). Logistic regression can be used as a baseline model

to detect potholes in an image by training the model on a dataset of labeled images (images with potholes and images without potholes).

The model starts by learning the features of the images (e.g. color, shape, size, etc.) and then applies a logistic regression algorithm to classify the images into two categories: those with potholes and those without. The model will then use the trained weights to assign a probability to each image that indicates the likelihood of it containing a pothole. A threshold can then be set to determine whether an image is classified as containing a pothole or not.

The advantage of using logistic regression as a baseline model for pothole detection is that it is relatively simple and fast. Furthermore, it can be used as a foundation for more complex models, such as convolutional neural networks.

- Logistic Classifier Performance:

  Logistic regression is a type of classifier that uses a logistic function to model the relationship between input features and a binary output, such as pothole or non-pothole. The performance of a logistic classifier for pothole detection is evaluated using various metrics, such as accuracy, precision, recall, F1 score, and ROC curve.

  The performance of a logistic classifier for pothole detection is summarized using Accuracy - The logistic regression achieved an accuracy of 85.3%

Test set accuracy: PCA + SVM > CNN > Logistic classifier

To improve performance, we can use a pretrained network / uncomment the remaining image augmentation (rotation) codes / scrape more data on the website.

TOOLS USED:

Language: Python

Coding Platform: Google colab

Python libraries

- **NumPy** - Provides support for large arrays and matrices, high-level mathematical functions, and efficient computation. It is an essential tool for scientific computing, data analysis, and machine learning applications.
- **SciPy** - SciPy is a Python library that builds on NumPy to provide additional functionality for scientific computing, including optimization, signal processing, linear algebra, and statistical functions. It is widely used in various fields such as engineering, physics, and finance.
- **Pandas** - Pandas is a Python library that provides fast, flexible, and expressive data structures designed to make working with "relational" or "labeled" data both easy and intuitive. It allows for reading and writing of data from different sources, data cleaning

and preparation, and data analysis and modeling. It is widely used for data analysis and manipulation tasks.

- **Matplotlib** - Matplotlib is a Python library that provides functions for creating static, animated, and interactive visualizations in Python. Matplotlib makes it easy to produce visualizations for a wide range of use cases, from exploratory data analysis to publication-quality figures.

Machine Learning algorithms.

Supervised learning algorithms such as

- Support Vector Machine
- Random Forest
- Logistic regression - Baseline Model

Deep learning algorithms such as

- Convolution Neural Network
- Convolution Neural Network with VGG Transfer Learning

In order to find potholes in the photos, Convolutional Neural Networks (CNNs) can also be used.

APIs or frameworks

- **TensorFlow** - Can adopt best practices for data automation, model tracking, performance monitoring, and model retraining. Success depends on the use of production-level technologies to automate and monitor model training over the course of a good, service, or business process.
- **Keras** - Simple to develop, test, and improve deep learning models. for creating and training deep learning models in a range of fields, including reinforcement learning, CV, and NLP.

CHALLENGES :

- Limited and Unstructured Data:
  One of the biggest challenges in pothole detection is the lack of labeled data for training the machine learning algorithms. It was challenging to collect and label data from various sources, and the data may be unstructured and inconsistent, making it difficult to train the models.
- Collection of large and balanced dataset:
  The dataset must contain adequate images with and without potholes . It should represent different types of roads, potholes, lighting conditions, and at different angles.
- Balancing data: The dataset can be unbalanced, meaning that one class (images with potholes - positive, for example) may have a disproportionately high number of examples compared to its counterpart (negative examples). As a result, models may become skewed and accuracy of prediction may decline.
- Pothole Detection:
  - Handling noise(reflections,shadows) in the images

    - Handling different lighting conditions(sunny, cloudy, rainy)
    - Handling different angles
    - Handling different sizes of potholes.
- Model Selection: Selecting an appropriate Convolutional Neural Network (CNN) architecture is also crucial to achieve high accuracy. With several CNN architectures to choose from, selecting the most suitable one could pose a challenge.
- Furthermore, road potholes have irregular shapes, making the geometric assumptions made in such approaches occasionally infeasible.

CONTRIBUTION OF TEAM MEMBERS:

Roll No	Name	Contributions
20Z216	Ganapathi S	Worked in training the CNN Model with VGG Transfer Learning,Random Forest Model,Image Augmentation for the CNN Model which involved Shearing, Zooming and Rescaling the Images.
20Z224	Karan H	Worked in training the CNN Model with VGG Transfer Learning,Random Forest Model,Logistic Regression - Baseline Model,Canny edge Detection to detect and mark contours on images containing potholes, Developing Streamlit Application to simulate the Project.
20Z246	Sarath Ra	Worked in training the SVM(finding regularization parameter and Kernel) with Principal Component Analysis(for dimensionality reduction),CNN Model model with ‘relu’ and ‘sigmoid’ activation.
21Z433	Glory C J	Worked in training the SVM(finding regularization parameter and Kernel) and Logistic Regression - Baseline Model.
21Z435	Sundarsree	Worked in Data preprocessing and Image augmentation which involved Labeling, Flipping , Horizontal and Vertical rotation of images ,Hough Line transformation to mark the areas in the image containing potholes.

ANNEXURE I:

DATA PREPOSSESSING:

```
# Load Libraries
import os
```

```
import numpy as np
import pandas as pd
import cv2
import matplotlib.pyplot as plt
from sklearn import svm
from sklearn.decomposition import PCA
from sklearn.metrics import accuracy_score
from keras import layers
from keras.layers import Input, Dense, Activation, ZeroPadding2D, BatchNormalization, Flatten, Conv2D
from keras.layers import AveragePooling2D, MaxPooling2D, Dropout, GlobalMaxPooling2D, GlobalAveragePooling2D
from keras.applications import vgg16
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Model
from keras.models import Sequential
%matplotlib inline
```

//IMPORTING IMAGE DATASET FROM GOOGLE DRIVE

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

//CREATING LABEL FOR IMAGES

```
# Load images
def load_im():
    input_im, input_label = [], []
    resize = (224, 224)
    # Loop in folders
    for dirname, _, filenames in os.walk('drive/MyDrive/proj dataset/archive'):
        for filename in filenames:
            photo_path = os.path.join(dirname, filename)
            photo_class = dirname.split('/')[-1]
            try:
                read_im = cv2.imread(photo_path)
                input_im.append(cv2.resize(read_im, resize))
                # potholes == 1
```

```
            if photo_class == 'potholes':
                input_label.append(1)
            # normal == 0
            elif photo_class == 'normal':
                input_label.append(0)
        except:
            print(photo_path)
    # return list of images and another list of corresponding labels
    return input_im, input_label

input_im, input_label = load_im()
copyinput_im,copyinput_label=input_im,input_label

# Train/Test split
def train_test_split(test_prop, input_im, input_label):
    # Random sampling of index
    test_size = int(np.floor(test_prop * len(input_label)))
    test_index = np.random.choice(len(input_label), size = test_size, replace = False)
    # Split
     train_x, test_x, train_y, test_y = np.delete(input_im, test_index, axis = 0), np.take(input_im,
test_index, axis = 0), np.delete(input_label, test_index, axis = 0), np.take(input_label, test_index,
axis = 0)
    # Return train and test sets for both images and labels
    return train_x, test_x, train_y, test_y, test_index

//Train Test Split


# 80/20 split for small data set
test_prop = 0.2
train_x, test_x, train_y, test_y, test_index = train_test_split(test_prop, input_im, input_label)
def append_im(input_im, input_label, im_iterator):
    input_label_n = input_label.copy()
    input_im_n = input_im.copy()
    for i in range(len(im_iterator)):
        im = im_iterator[i]
```

```
        im = im.astype('uint8')
        im_lbl = [input_label[i]]
        input_im_n = np.append(input_im_n, im, axis = 0)
        input_label_n = np.append(input_label_n, im_lbl, axis = 0)
    return input_im_n, input_label_n
```

//Image augmentation

//Applied to training set only

//1. Horizontal Flipping

```
# Flipping
flip_data_generator = ImageDataGenerator(horizontal_flip = True)
im_iterator = flip_data_generator.flow(train_x, batch_size = 1, shuffle = False)
input_im_n, input_label_n = append_im(train_x, train_y, im_iterator)
```

//2. Clockwise Rotation by 30 degree

```
#Rotation - 30 deg
rotate_data_generartor = ImageDataGenerator(rotation_range = 30)
im_iterator = rotate_data_generartor.flow(train_x, batch_size = 1, shuffle = False)
input_im_n, input_label_n = append_im(input_im_n, input_label_n, im_iterator)
```

//3. Anti-clockwise Rotation by 30 degree

```
#Rotation - -30 deg
rotate_data_generartor = ImageDataGenerator(rotation_range = 330)
im_iterator = rotate_data_generartor.flow(train_x, batch_size = 1, shuffle = False)
input_im_n, input_label_n = append_im(input_im_n, input_label_n, im_iterator)
# Reshape
nx, ny, nz = train_x.shape[1], train_x.shape[2], train_x.shape[3]
train_x_nn, test_x_nn = input_im_n, test_x
train_x = input_im_n.reshape((input_im_n.shape[0], nx * ny * nz)) / 255
```

```
test_x = test_x.reshape((test_x.shape[0], nx * ny * nz)) / 255
train_y = input_label_n.reshape((input_label_n.shape[0], 1))
test_y = test_y.reshape((test_y.shape[0], 1))
# Dimensionality reduction - Full PCA- reduces the number of features in a #dataset by finding the linear combinations of features that explain the #most variance.
im_pca = PCA()
im_pca.fit(train_x)
variance_explained_list = im_pca.explained_variance_ratio_.cumsum()
print(variance_explained_list)
test_x_pca = im_pca.transform(test_x)
train_x_pca = im_pca.transform(train_x)
```

//RANDOM FOREST MODEL

```
# Implementing Random Forest Model
from sklearn.ensemble import RandomForestClassifier
# Creating the random forest model
rf = RandomForestClassifier(max_depth=8, random_state=0)
# Training the model
rf.fit(train_x_pca, train_y)
# Predicting on the test data
predictions = rf.predict(test_x_pca)
# Calculating the accuracy of the model
accuracy = accuracy_score(test_y, predictions)
# Printing the accuracy
print(accuracy)
```

//Testing if the image contains pothole or not - RANDOM FOREST

```
if prediction[0]==0:
```

```
  print("Normal")
Else:
//Using GrayScale to find contours and Mask the Potholes.
  gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
  # Apply thresholding to detect the boundaries of the pothole
  thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)[1]
  # Find contours of the pothole
                contours = cv2.findContours(thresh, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
  contours = contours[0] if len(contours) == 2 else contours[1]
  # Create a mask of the pothole
  mask = np.zeros(image.shape, np.uint8)
  cv2.drawContours(mask, contours, -1, (255,0,0), 3)
  # Use the mask to mark the area of the pothole
  output = cv2.bitwise_and(image, mask)
  # Show the output
  plt.imshow(output)
  plt.show()
  print("Potholes")
plt.imshow(image)
```

//SVM

```
# Support vector machine with PCA
def svm_grid_search(C, kernel, train_x, train_y):
    accuracy_score_list = []
    for c in C:
        # Model training
```

```
        svmClassifier = svm.SVC(C = c, kernel = kernel)
        svmClassifier.fit(train_x, train_y.ravel())
        # Prediction on test set
        pred_y = svmClassifier.predict(train_x)
        # Accuracy
        accuracy = accuracy_score(train_y, pred_y)
        accuracy_score_list.append(accuracy)
        print('Regularization parameters: ', c, 'Accuracy', accuracy)
    max_accurarcy_id = accuracy_score_list.index(max(accuracy_score_list))
    return C[max_accurarcy_id]
topt_C = svm_grid_search(C, kernel, train_x_pca, train_y)
//
# Test set
svmClassifier = svm.SVC(C = opt_C, kernel = kernel)
svmClassifier.fit(train_x_pca, train_y.ravel())
pred_y = svmClassifier.predict(test_x_pca)
accuracy = accuracy_score(test_y, pred_y)
print(accuracy)
//Prediction is done for entire test test
def predict_img(svmClassifier,test_x,test_y):
  pred_y = svmClassifier.predict(test_x)
  accuracy = accuracy_score(test_y, pred_y)
  print(accuracy)
  for index in range(len(test_y)):
    print("Image :" , index+1)
    plt.imshow(test_x_nn[index])
```

```
    plt.show()
    if test_y[index]==1:
      print("Actual: Image contains potholes")
    else:
      print("Actual: Image does not contain potholes")
    if pred_y[index]==1:
      print("Predicted: Image contains potholes")
    else:
      print("Predicted: Image does not contain potholes")
predict_img(svmClassifier,test_x_pca,test_y)
```

//LOGISTIC REGRESSION

```
# Logistic Regression
def Logistic():
    logistic_model = Sequential()
    logistic_model.add(Dense(1, activation = 'sigmoid'))
    return logistic_model
# Compile Model
logistic_model = Logistic()
logistic_model.compile(loss = 'binary_crossentropy', optimizer = 'adam', metrics = ['accuracy'])
# Training Model
logistic_model.fit(train_x, train_y, batch_size = 32, epochs = 50, verbose = 1)
# Test set
print(logistic_model.metrics_names)
print(logistic_model.evaluate(test_x, test_y, verbose = 0))
# Predicting on the test set
predictions = logistic_model.predict(test_x)
```

```
print(len(predictions),len(test_y))
for i in range(len(predictions)):
  img = input_im[test_index[i]]
  img=cv2.resize(img, (500, 500))
  if predictions[i] > 0.5:
     print("Image contains pothole")
     gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
     #Finding the edges in the image using Canny edge detection
     edges = cv2.Canny(gray,50,150,apertureSize = 3)#Finding the contours in the image
     contours,hierarchy = cv2.findContours(edges, 1, 2)
     #Looping throught the contours
     for cnt in contours:
       #Creating a rectangle around the contours
       x,y,w,h = cv2.boundingRect(cnt)
       width_relative = img.shape[0]/w
       height_relative = img.shape[1]/h
       #Checking if the size of the hole is greater than the specified size
       if w>100 and h>100:
         #if width_relative > 1 and height_relative > 1:
         #Drawing the rectangle
         cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,0),2)
  else:
     print("Image does not contain pothole")
  cv2_imshow(img)
  cv2.waitKey(0)
  cv2.destroyAllWindows()
```

//CNN

```
#Splitting dataset into training and validation and preprocessing the training set

train_datagen = ImageDataGenerator(rescale = 1./255,shear_range = 0.2, zoom_range = 0.2, horizontal_flip = True, validation_split=0.2)

training_set = train_datagen.flow_from_directory('drive/MyDrive/proj dataset/archive', target_size = (64, 64),batch_size = 32, class_mode = 'binary', subset="training")

#Preprocessing the validation set

validation_generator = train_datagen.flow_from_directory(

   "drive/MyDrive/proj dataset/archive", # same directory as training data

  target_size=(64, 64),batch_size=32,class_mode='binary',subset='validation')

#Initialising the CNN

cnn = tf.keras.models.Sequential()

#Step 1 - Convolution

cnn.add(tf.keras.layers.Conv2D(filters=32,kernel_size=3,activation='relu', input_shape=[64, 64, 3]))

#Step 2 - Pooling

cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))

#Adding a second convolutional layer

cnn.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu'))

cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))

#Step 3 - Flattening

cnn.add(tf.keras.layers.Flatten())

#Step 4 - Full Connection

cnn.add(tf.keras.layers.Dense(units=128, activation='relu'))

#Step 5 - Output Layer

cnn.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))

#Training the CNN on the Training set and evaluating it on the Validation set
```

```
cnn.fit(x = training_set, validation_data = validation_generator, epochs = 25)
#Part 4 - Making a single prediction
import keras.utils as image
import numpy as np
#from keras.preprocessing import image
def checker(str):
  test_image = image.load_img('drive/MyDrive/proj dataset/archive/'+str, target_size = (64, 64))
  plt.imshow(cv2.imread("drive/MyDrive/proj dataset/archive/"+str))
  test_image = image.img_to_array(test_image)
  test_image = np.expand_dims(test_image, axis = 0)
  result = cnn.predict(test_image)
  training_set.class_indices
  if result[0][0] == 1:
    prediction = 'pothole'
  else:
    prediction = 'normal'
  return prediction
print(checker('normal/4.jpg'))
```

//CNN WITH VGG TRANSFER LEARNING

```
# Create a CNN Sequential Model
model = Sequential()
model.add(Conv2D(32, (5,5), activation = 'relu', input_shape=(128,128,3)))
model.add(MaxPooling2D((2,2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
```

```
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dropout(0.4))
model.add(Dense(128, activation='relu'))
model.add(Dense(2, activation='softmax'))
#Model configuration for training purpose
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
print(model.summary())
history=model.fit(x_train, y_train, epochs=30, batch_size=12, verbose=2, validation_data=(x_test, y_test))
loss, accuracy = model.evaluate(x_test, y_test)
print('Test accuracy: {:2.2f}%'.format(accuracy*100))
prediction = model.predict(x_test)
y_pred = np.argmax(prediction, axis=1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['loss'])
plt.title('Accuracy and Loss Plot')
plt.xlabel('epochs')
plt.ylabel('accuracy')
plt.legend(['accuracy','loss'], loc='upper right')
plt.show()
plt.plot(history.history['val_accuracy'])
plt.plot(history.history['val_loss'])
plt.title('Testing Accuracy and Loss Plot')
plt.xlabel('epochs')
```

```
plt.ylabel('Validation accuracy and loss')
plt.legend(['val_accuracy','val_loss'], loc='upper left')
plt.show()
#Create a Confusion Matrix for Evaluation
# H = Horizontal
# V = Vertical
pd.DataFrame(confusion_matrix(y_test2, y_pred),
             columns=["Predicted NORMAL", "Predicted POTHOLES"],
             index=["Actual NORMAL", "Actual POTHOLES"])
IMAGE_SIZE = [128, 128]
```

//VGG TRANSFER LEARNING

```
vgg = VGG16(input_shape=IMAGE_SIZE + [3], weights='imagenet', include_top=False)
# don't train existing weights
for layer in vgg.layers:
  layer.trainable = False
#layers
x = Flatten()(vgg.output)
# x = Dense(1000, activation='relu')(x)
prediction = Dense(len(folders), activation='softmax')(x)
# create a model object
model = Model(inputs=vgg.input, outputs=prediction)
# view the structure of the model
model.summary()
# tell the model what cost and optimization method to use
model.compile(loss='categorical_crossentropy',optimizer='rmsprop',metrics=['accuracy'])
# create an instance of ImageDataGenerator
```

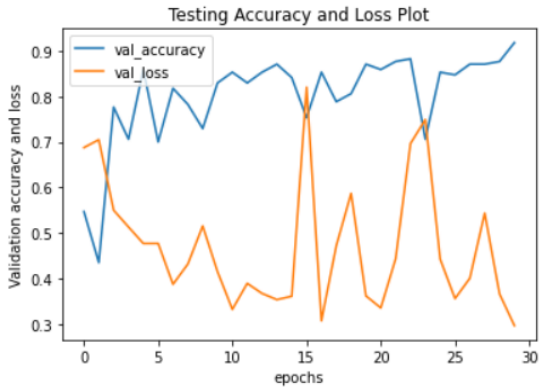
```
gen = ImageDataGenerator(rotation_range=20,width_shift_range=0.1,height_shift_range=0.1,
  shear_range=0.1,zoom_range=0.2,horizontal_flip=True,vertical_flip=True,
preprocessing_function=preprocess_input)
test_gen = gen.flow_from_dataframe(
        dataframe=img_df,
        #directory='drive/MyDrive/proj dataset/archive',
        x_col="filename",
        y_col="y",
        target_size=IMAGE_SIZE)
print(test_gen.class_indices)
import keras.utils as image
def checker(str):
   test_image = image.load_img('drive/MyDrive/proj  dataset/archive/'+str, target_size = (128,
128))
  plt.imshow(cv2.imread("drive/MyDrive/proj dataset/archive/"+str))
  test_image = image.img_to_array(test_image)
  test_image = np.expand_dims(test_image, axis = 0)
  result = model.predict(test_image)
  if result[0][0] > result[0][1]:
    prediction = 'Normal'
  else:
    prediction = 'Potholes'
  if np.argmax(result[0]) == 0:
    prediction = 'Normal'
  else:
    prediction = 'Potholes'
```

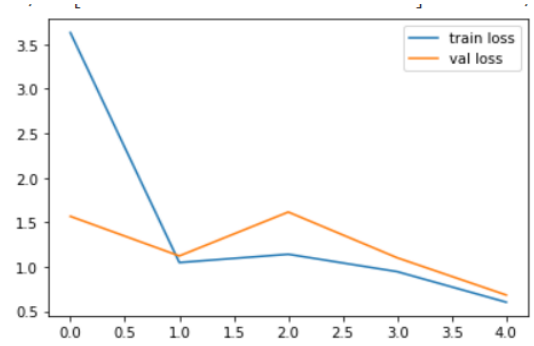
```
  return prediction

print(checker('normal/4.jpg'))
```

ANNEXURE II.

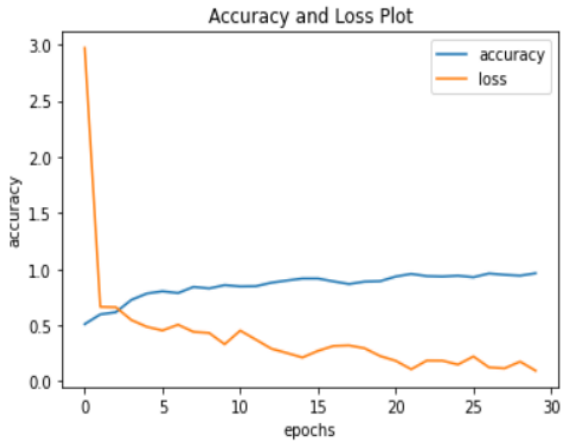
OUTPUT:

CNN With VGG Transfer Learning - Accuracy : 91.76%

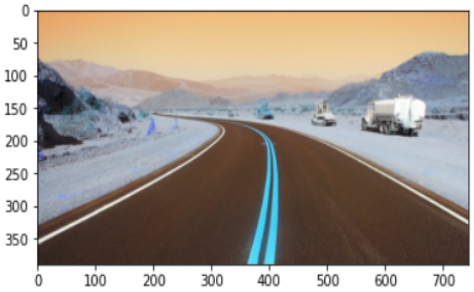





```
<Figure size 432x288 with 0 Axes>
```



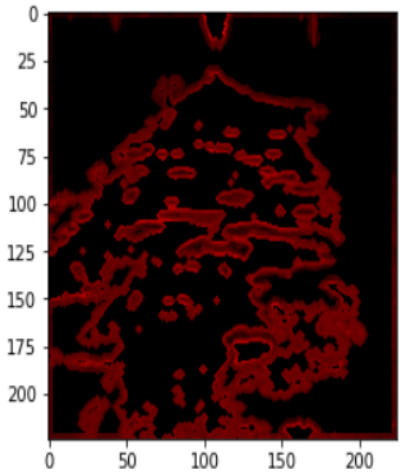
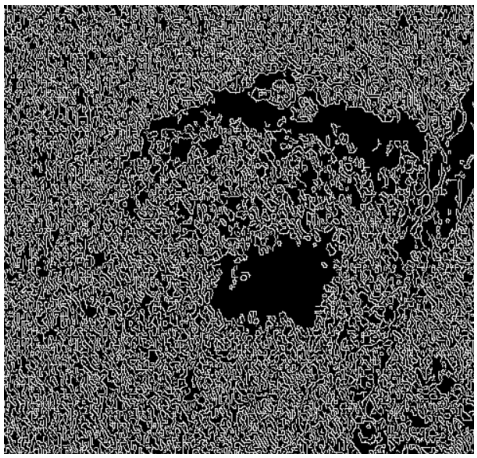
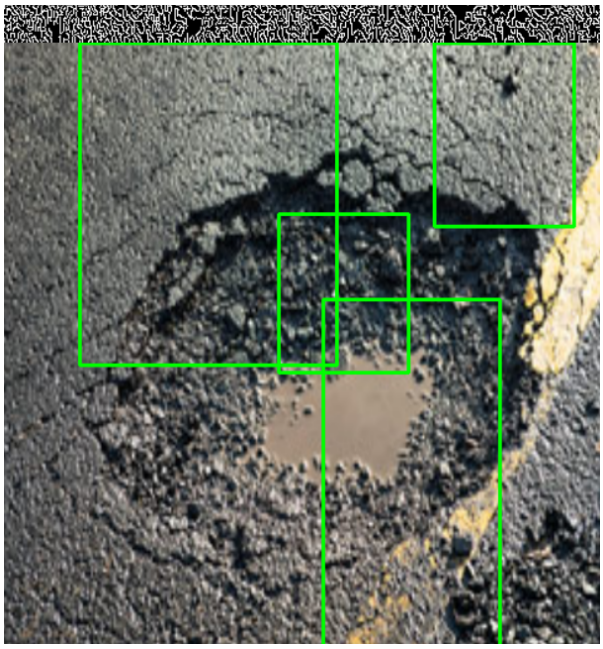

```
1/1 [==============================] - 1s 704ms/step
Normal
```



```
]
print(checker('potholes/103.jpg'))

1/1 [==============================] - 0s 117ms/step
Potholes
```

	Predicted NORMAL	Predicted POTHOLES
Actual NORMAL	85	11
Actual POTHOLES	3	71

RANDOM FOREST

 Potholes <matplotlib.image.AxesImage at 0x7f2e5529dfd0> 	 

LOGISTIC REGRESSION - BASELINE MODEL - Accuracy: 77.9%

Actual: Image contains potholes
Predicted: Image contains potholes

Support vector machine - Accuracy: 86.02%

SVM with PCA performance

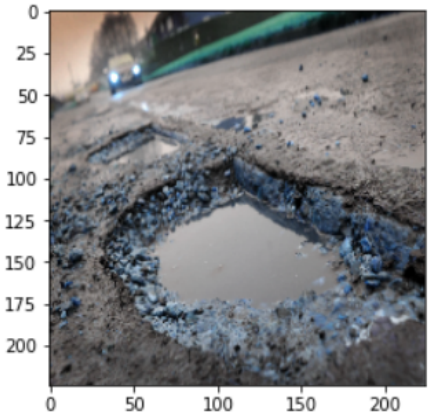
Model parameters:

Regularization parameters = 2.9

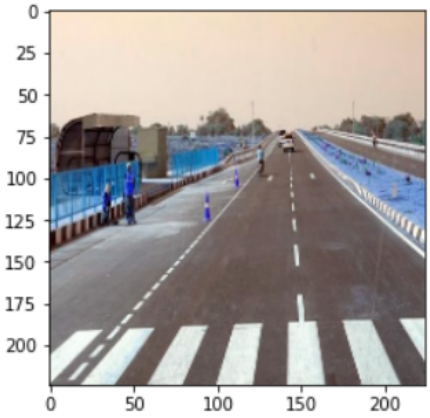
Kernel = Radial Basis Function

Test set accuracy: 86.02%

Predicted: Image contains potholes
Image : 13

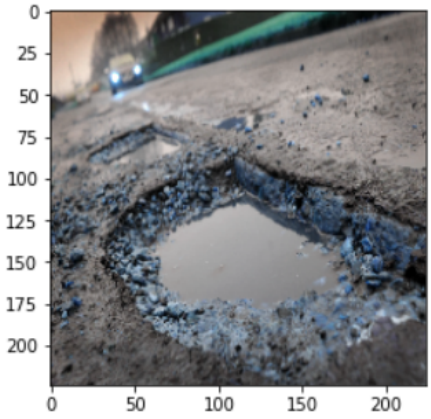


Actual: Image contains potholes
Predicted: Image contains potholes
Image : 14

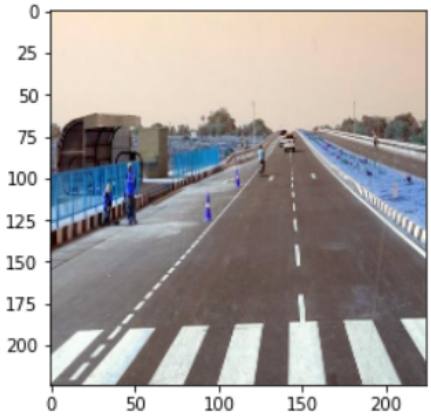


Actual: Image does not contain potholes
Predicted: Image does not contain potholes

Predicted: Image contains potholes
Image : 13



Actual: Image contains potholes
Predicted: Image contains potholes
Image : 14



Actual: Image does not contain potholes
Predicted: Image does not contain potholes

REFERENCES:

https://www.kaggle.com/datasets/rajdalsaniya/pothole-detection-dataset

https://www.researchgate.net/publication/323314224_Pothole_Detection_using_Machine_Learning

https://www.researchgate.net/publication/358977900_Pothole_Detection_Using_Deep_Learning_A_Real-Time_and_AI-on-the-Edge_Perspective/link/627e2afb37329433d9adcbe2/download

https://www.mdpi.com/1950332

https://scholar.google.com/scholar_lookup?title=Pothole%20detection%20based%20on%20disparity%20transformation%20and%20road%20surface%20modeling&author=R%20Fan&author=U%20Ozgunalp&author=B%20Hosking&publication_year=2019&journal=IEEE%20Transactions%20on%20Image%20Processing&volume=29&pages=897-908

https://scholar.google.com/scholar_lookup?title=Pothole%20detection%20in%20asphalt%20pavement%20images&author=C%20Koch&author=I%20Brilakis&publication_year=2011&journal=Advanced%20Engineering%20Informatics&volume=25&pages=507-515

https://www.sciencedirect.com/science/article/pii/S2666165922000436

https://link.springer.com/chapter/10.1007/978-3-030-82199-9_24

https://link.springer.com/chapter/10.1007/978-3-319-52015-5_33

https://iopscience.iop.org/article/10.1088/1742-6596/2162/1/012019/meta