**Detection of potholes on the roads**

**Group 39**

Enrol. No. (s) - 9919103092**,** 9919103094**,** 9919103106

Name of Student (s) - Aditri Srivastava, Sanyam Jain, Mudit Virmani

Name of supervisor(s) - Dr Raju Pal



**Department of CSE/IT**

**Jaypee Institute of Information Technology University, Noida**

**October 2021**

Submitted in partial fulfillment of the Degree of Bachelor of Technology

in

Computer Science Engineering

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING & INFORMATION TECHNOLOGY

JAYPEE INSTITUTE OF INFORMATION TECHNOLOGY, NOIDA

1. **INTRODUCTION**
   1. **General Introduction**

**Title**: Detection of potholes on the roads

Poorly maintained roads are a fact of life in most developing countries, including India. A well-maintained road network is a must for the well-being and development of any country. So, it is necessary to create an effective road surface monitoring system.

A pothole is a depression in a road surface, usually asphalt pavement, where traffic has removed broken pieces of the pavement. It is usually the result of water in the underlying soil structure and traffic passing over the affected area. Water first weakens the underlying soil; traffic then fatigues and breaks the poorly supported asphalt surface in the affected area. Continued traffic action ejects both asphalt and the underlying soil material to create a hole in the pavement.

Pothole detection is our focus in the system. Our objective isto reduce the number of accidents caused by poor conditions of roads. Also, to provide drivers safety and warn them about upcoming potholes.

Potholes are formed due to wear and tear and weathering of roads. They cause not only discomforts to citizens but also deaths due vehicle accidents. The US records more than 2000 fatal accidents per year due to potholes and bad road conditions.



* 1. **PROBLEM STATEMENT:**

Potholes are a nuisance, especially in the developing world, and can often result in vehicle damage or physical harm to the vehicle occupants. Drivers can be warned to take evasive action if potholes are detected in real-time.

* 1. **Significance/novelty of the problem**

Front cameras in cars are convenient and provide safety but are expensive. Very few cars have this feature built in, and it is out of budget for an average consumer. Our vision is to get a Front camera in every car for a minimal cost to bridge the gap between luxury and necessity. Our solution is to use a mobile phone camera mounted on dashboards. Our app will detect potholes and alert the user with minimal operating cost.

* 1. **Empirical Study**
     1. CNN

Convolution Neural Networks or covnets are neural networks that share their parameters. For example, imagine you have an image. It can be represented as a cuboid having its length, width (dimension of the image), and height (as images generally have red, green, and blue channels).

There are three types of layers in Convolutional Neural Networks:

1) Convolutional Layer: In a typical neural network each input neuron is connected to the next hidden layer. In CNN, only a small region of the input layer neurons connect to the neuron hidden layer.

2) Pooling Layer: The pooling layer is used to reduce the dimensionality of the feature map. There will be multiple activation & pooling layers inside the hidden layer of the CNN.

3) Fully-Connected layer: Fully Connected Layers form the last few layers in the network. The input to the fully connected layer is the output from the final Pooling or Convolutional Layer, which is flattened and fed into the fully connected layer.

After the preprocessing, we implemented CNN because it automatically detects the important features without human supervision. This is why CNN would be an ideal solution to computer vision and image classification problems.

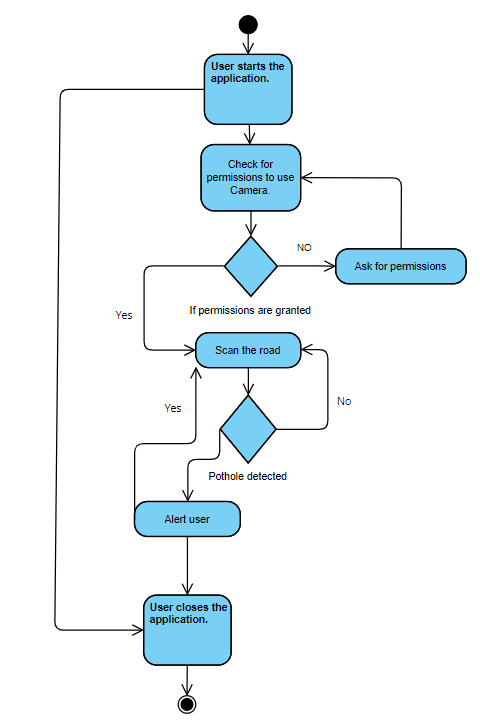
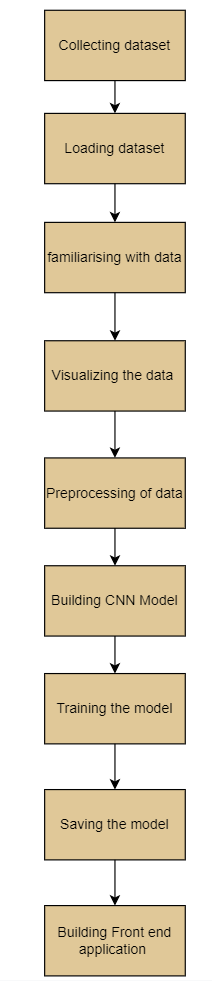
The convolutional neural network is better than a feed-forward network since CNN has feature parameter sharing and dimensionality reduction. Because of parameter sharing in CNN, the number of parameters is reduced. Thus the computations also decreased. The main intuition is that learning from one part of the image is also useful for another part of the image. Because of the dimensionality reduction in CNN, the computational power needed is reduced.

* + 1. Flutter

Flutter is a widget-based technology. This means that you can apply object-oriented programming to any element. One of the benefits of using Flutter is that you can modify or customize widgets with ease. In addition, it provides UI widgets that meet key web application design requirements. Flutter, as an open-source framework, has attracted a broad and active community of developers since its release. This community constantly publishes usable code examples and supports developers in creating new, innovative, beautiful cross-platform apps. Flutter’s code reusability allows you to write just one codebase and use it not only for mobile Android and iOS but even for the web, desktop and more. This cuts development time significantly removes costs and enables you to launch your app much faster.

* 1. **Brief Description of the Solution Approach**
     1. Scanning road: The app asks for required camera permissions. Then, it keeps clicking pictures periodically, and the image is processed to detect potholes.
     2. Processing: The image is preprocessed, resized to 128x128, and normalized. The algorithm used to detect potholes in images is CNN.
     3. Alerting user: If a pothole is detected a prompt is given on the user's screen with a subtle sound/alarm.
  2. **Comparison of existing approaches to the problem framed**
     1. Satellite instead of the camera
        1. There are current solutions to this problem which use satellite images instead using live cameras. This won’t give users real-time updates in case of route changes. Satellite images change after a longer interval and are not that precise. Most importantly, this doesn’t work in areas of low connectivity as they require internet.

1. **Literature Survey**
   1. **Summary of papers studied.**
      1. [Indian pothole detection based on CNN and anchor-based deep learning method](https://www.researchgate.net/publication/358593575_Indian_pothole_detection_based_on_CNN_and_anchor-based_deep_learning_method)
         1. The main contribution of this paper lies in collecting the pothole data in different Indian traffic conditions and detecting of the same using a vision-based method by defining the performance of deep learning methods like sequential convolutional neural network (CNN), and anchor-based You only Look Once3 and analyzing the models in terms of resources and performance of detection. The experiments were conducted on both models, and a conclusion was drawn to bring out the benefits of the model with 98% accuracy using CNN and 83% precision using Yolov3.
      2. [Detecting Potholes Using Simple Image Processing Techniques and Real-World Footage](https://www.researchgate.net/publication/279538022_Detecting_Potholes_Using_Simple_Image_Processing_Techniques_and_Real-World_Footage)
         1. A model of potholes is constructed using the image library, which is used in an algorithmic approach that combines a road colour model with simple image processing techniques such as a Canny filter and contour detection. Using this approach, it was possible to detect potholes with a precision of 81.8% and recall of 74.4.%.
      3. <https://www.researchgate.net/publication/353258429_Real-Time_Pothole_Detection_Using_Deep_Learning/link/6196dafa61f0987720b280ab/download>
      4. [(PDF) Dataset of images used for pothole detection](https://www.researchgate.net/publication/282807920_Dataset_of_images_used_for_pothole_detection)
         1. The entire dataset consists of two different sets, one was considered to be simple and the other more complex. Each dataset contains folders containing the training (positive and negative) images and a collection of positive test images. This dataset overall contains around 12 thousand images. As per our observation, the images seemed to be taken from the car's dashboard.
      5. [(PDF) A Real-Time Pothole Detection Approach for Intelligent Transportation System](https://www.researchgate.net/publication/282832097_A_Real-Time_Pothole_Detection_Approach_for_Intelligent_Transportation_System)

1. **Requirement Analysis and Solution Approach**
   1. **Functional requirements**
      1. The system should be granted the required permissions.
      2. The system should be able to scan the road in intervals of 5 seconds.
      3. The system should be able to process the scanned image for potholes.
      4. The system should be able to give a prompt/alert if a pothole is encountered.
   2. **Non-functional requirements**
      1. The system should be accurate.
      2. Pothole detection should be fast and efficient.
      3. There should be low latency in the prompt if a pothole is detected.
      4. Alert should be synced with car speed and distance between the car and the pothole.
   3. **Activity diagram**
   4. **Workflow diagram  
       **
   5. **Solution Approach**
      1. Collection of the dataset for images of roads, some containing potholes and others are normal roads.
      2. Preprocessing of Data.
      3. Model creation using CNN.
      4. The user interface will be provided to the user using an application
      5. Live images will be captured using the camera.
      6. Feeding these images after preprocessing to the model.
      7. Altering the user about the potholes on the images.
2. **Modelling and Implementation Details**
   1. Implementation details and issues
      1. Code

# -\*- coding: utf-8 -\*-

"""Copy of Major-1.ipynb

Automatically generated by Colaboratory.

Original file is located at

https://colab.research.google.com/drive/1aGRB6RK7Oz\_Wg638aR-2mHBWZC90F8vx

"""

! pip install -q kaggle

from google.colab import files

files.upload()

! cp kaggle.json ~/.kaggle/

! chmod 600 ~/.kaggle/kaggle.json

! kaggle datasets list

! kaggle datasets download -d atulyakumar98/pothole-detection-dataset

! mkdir pothole

! unzip pothole-detection-dataset.zip -d pothole

from google.colab import drive

drive.mount("/content/gdrive")

import numpy as np

import matplotlib.pyplot as plt

import cv2

import pandas as pd

imagepaths = []

import os

for dirname, \_, filenames in os.walk('/content/pothole'):

for filename in filenames:

path = os.path.join(dirname, filename)

imagepaths.append(path)

print(len(imagepaths))

IMG\_SIZE=128

x=[]

y=[]

for image in imagepaths:

try:

img = cv2.imread(image,cv2.IMREAD\_COLOR)

img = cv2.resize(img, (IMG\_SIZE,IMG\_SIZE))

x.append(np.array(img))

if(image.startswith('/content/pothole/normal/')):

y.append('NORMAL')

else:

y.append('POTHOLES')

except:

pass

print(len(x))

print(y)

from sklearn.utils import shuffle

x,y = shuffle(x, y, random\_state=5)

print(y)

import random as rn

fig,ax=plt.subplots(2,5)

plt.subplots\_adjust(bottom=0.3, top=0.7, hspace=0)

fig.set\_size\_inches(15,15)

for i in range(2):

for j in range(5):

l=rn.randint(0,len(y))

print(imagepaths[l])

ax[i,j].imshow(x[l][:,:,::-1])

ax[i,j].set\_title(y[l])

ax[i,j].set\_aspect('equal')

from keras.layers import Input, Lambda, Dense, Flatten

from keras.models import Model

from keras.preprocessing import image

from keras.preprocessing.image import ImageDataGenerator

# from sklearn.metrics import confusion\_matrix

# from glob import glob

from sklearn.preprocessing import LabelEncoder

from keras.utils.np\_utils import to\_categorical

from sklearn.model\_selection import train\_test\_split

le=LabelEncoder()

Y=le.fit\_transform(y)

Y=to\_categorical(Y,2)

print(Y)

x=np.array(x)

x=x/255

x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,Y,test\_size=0.25,random\_state=5)

from keras.models import Sequential

from keras.layers.convolutional import Conv2D, MaxPooling2D

from keras.layers import Dense, Flatten, Dropout

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.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

print(model.summary())

training\_history=model.fit(x\_train, y\_train, epochs= 100, batch\_size=12, verbose=2,

validation\_data=(x\_test, y\_test))

def render\_training\_history(training\_history):

loss = training\_history.history['loss']

val\_loss = training\_history.history['val\_loss']

accuracy = training\_history.history['accuracy']

val\_accuracy = training\_history.history['val\_accuracy']

plt.figure(figsize=(14, 4))

plt.subplot(1, 2, 1)

plt.title('Loss')

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.plot(loss, label='Training set')

plt.plot(val\_loss, label='Test set', linestyle='--')

plt.legend()

plt.grid(linestyle='--', linewidth=1, alpha=0.5)

plt.subplot(1, 2, 2)

plt.title('Accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.plot(accuracy, label='Training set')

plt.plot(val\_accuracy, label='Test set', linestyle='--')

plt.legend()

plt.grid(linestyle='--', linewidth=1, alpha=0.5)

plt.show()

render\_training\_history(training\_history)

loss, accuracy = model.evaluate(x\_test, y\_test)

print('Test accuracy: {:2.2f}%'.format(accuracy\*100))

from keras.models import Sequential

from keras.layers.convolutional import Conv2D, MaxPooling2D

from keras.layers import Dense, Flatten, Dropout

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.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

print(model.summary())

training\_history=model.fit(x\_train, y\_train, epochs= 50, batch\_size=12, verbose=2,

validation\_data=(x\_test, y\_test))

def render\_training\_history(training\_history):

loss = training\_history.history['loss']

val\_loss = training\_history.history['val\_loss']

accuracy = training\_history.history['accuracy']

val\_accuracy = training\_history.history['val\_accuracy']

plt.figure(figsize=(14, 4))

plt.subplot(1, 2, 1)

plt.title('Loss')

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.plot(loss, label='Training set')

plt.plot(val\_loss, label='Test set', linestyle='--')

plt.legend()

plt.grid(linestyle='--', linewidth=1, alpha=0.5)

plt.subplot(1, 2, 2)

plt.title('Accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.plot(accuracy, label='Training set')

plt.plot(val\_accuracy, label='Test set', linestyle='--')

plt.legend()

plt.grid(linestyle='--', linewidth=1, alpha=0.5)

plt.show()

render\_training\_history(training\_history)

loss, accuracy = model.evaluate(x\_test, y\_test)

print('Test accuracy: {:2.2f}%'.format(accuracy\*100))

model\_json = model.to\_json()

with open("model.json", "w") as json\_file:

json\_file.write(model\_json)

# serialize weights to HDF5

model.save\_weights("model.h5")

print("Saved model to disk")

from keras.models import Sequential

from keras.layers.convolutional import Conv2D, MaxPooling2D

from keras.layers import Dense, Flatten, Dropout

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.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

print(model.summary())

training\_history=model.fit(x\_train, y\_train, epochs= 35, batch\_size=12, verbose=2,

validation\_data=(x\_test, y\_test))

# model\_json = model.to\_json()

# with open("model.json", "w") as json\_file:

# json\_file.write(model\_json)

# # serialize weights to HDF5

# model.save\_weights("model.h5")

# print("Saved model to disk")

def render\_training\_history(training\_history):

loss = training\_history.history['loss']

val\_loss = training\_history.history['val\_loss']

accuracy = training\_history.history['accuracy']

val\_accuracy = training\_history.history['val\_accuracy']

plt.figure(figsize=(14, 4))

plt.subplot(1, 2, 1)

plt.title('Loss')

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.plot(loss, label='Training set')

plt.plot(val\_loss, label='Test set', linestyle='--')

plt.legend()

plt.grid(linestyle='--', linewidth=1, alpha=0.5)

plt.subplot(1, 2, 2)

plt.title('Accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.plot(accuracy, label='Training set')

plt.plot(val\_accuracy, label='Test set', linestyle='--')

plt.legend()

plt.grid(linestyle='--', linewidth=1, alpha=0.5)

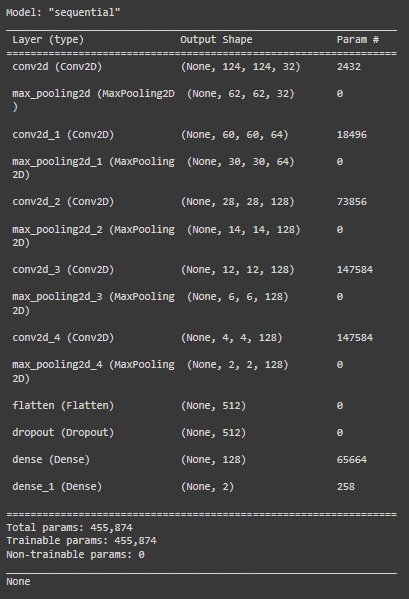
plt.show()

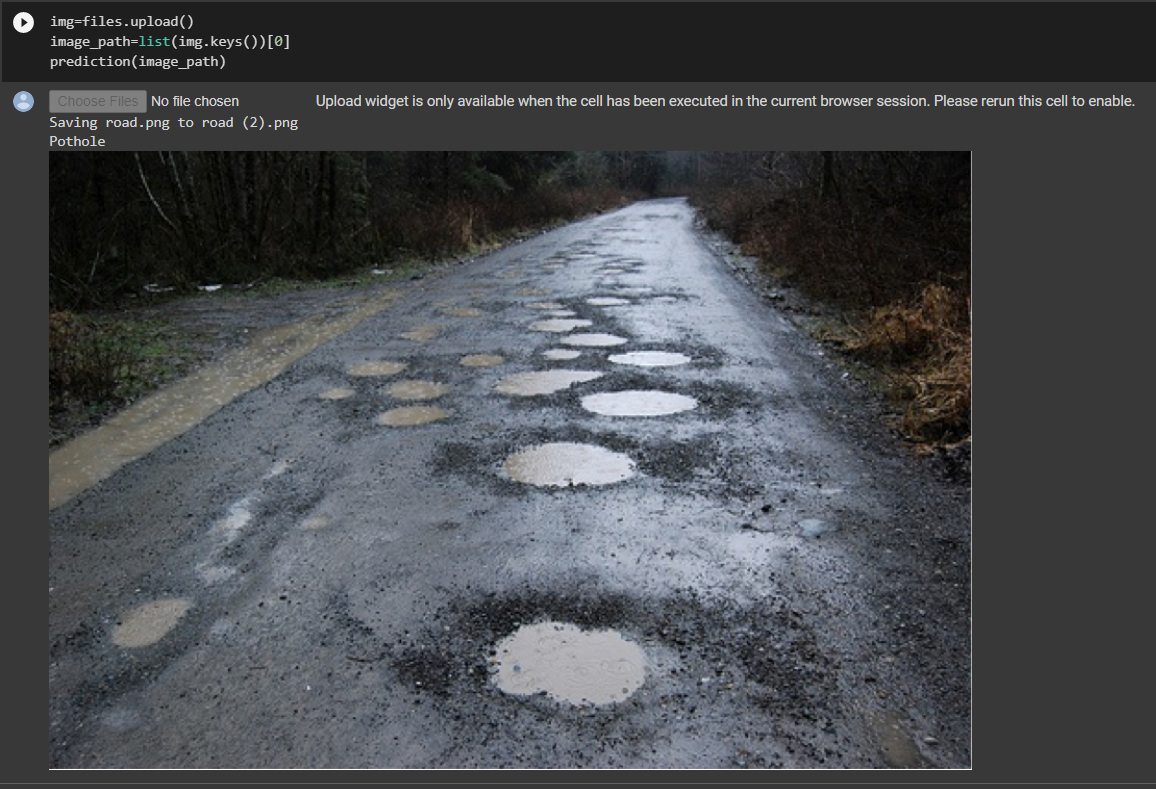
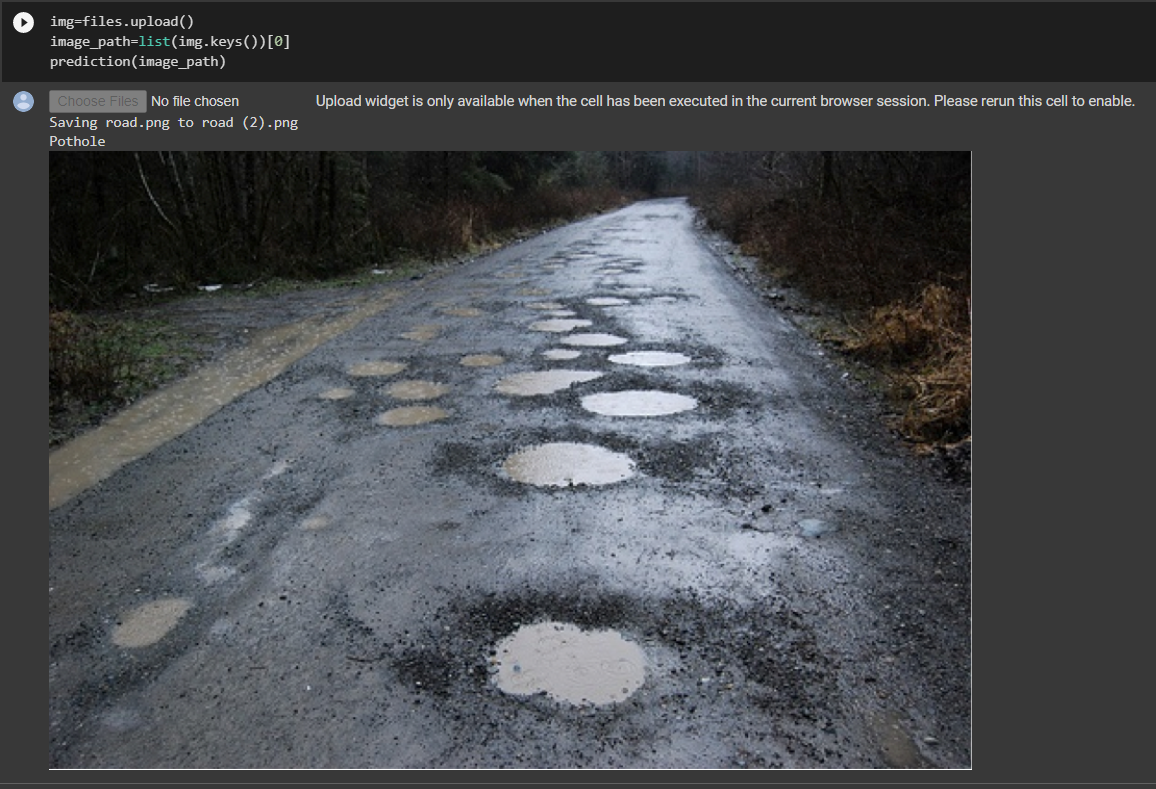
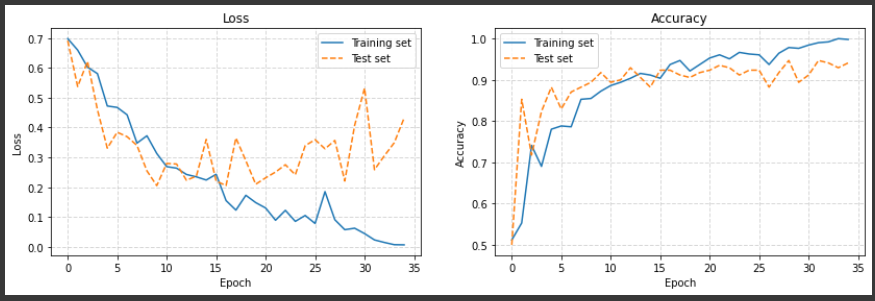
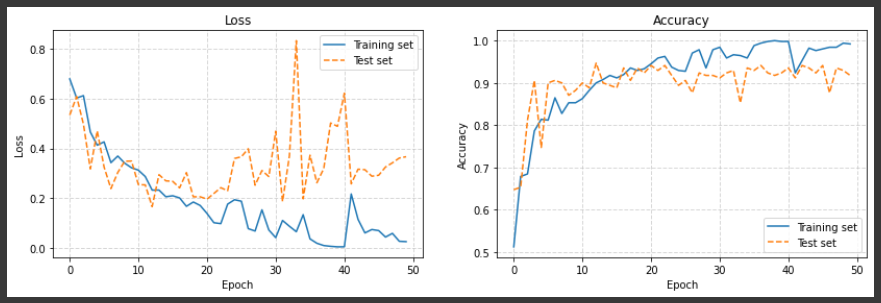
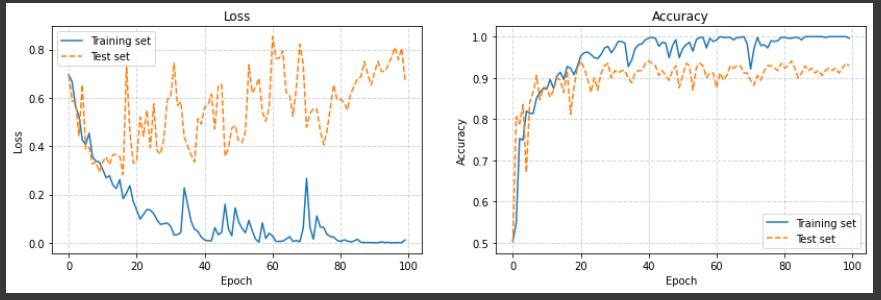
render\_training\_history(training\_history)

loss, accuracy = model.evaluate(x\_test, y\_test)

print('Test accuracy: {:2.2f}%'.format(accuracy\*100))

* + 1. Explanation
       1. Importing dataset from Kaggle.
       2. Unzipping and saving datasets in the drive.
       3. Preprocessing image dataset:
          1. resizing images into 128 X 128.
          2. Labelling images using folder names from the dataset as normal and pothole images.
          3. Shuffling images to create diverse training data.
          4. Normalising images by dividing them by 255 so that all values are between 0 and 1.
          5. Creating training and testing data with 25% of data in testing.
       4. Model is created using CNN adding various conv2D layers, MaxPooling layers using Relu as the activation function and softmax as the activation function in the last layers.
       5. The model is compiled using adam optimizer, categorical cross entropy and accuracy as measures for evaluating the model.



* + - 1. Model is saved and visualised.
    1. Test cases:
       1. For input of a image with potholes, we are getting pothole as result.
       2. For input of image with normal road, we are getting output of normal.
    2. Accuracy for different epochs:  
       1. For 35 epochs test accuracy is 92.94%
       2. For 50 epochs test accuracy is 91.76%
       3. For 100 epochs test accuracy is 94.12%

1. **Conclusion**

We have achieved 90 % accuracy for this dataset using CNN Model. In the future, users will be able to use this on their smartphones, making their travel hassle-free and safer.

1. **References**
   1. Python libraries
      1. <https://www.tensorflow.org/api_docs/python/tf/keras>
      2. <https://numpy.org/doc/>
      3. <https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html>
      4. <https://matplotlib.org/stable/index.html>
      5. <https://pandas.pydata.org/docs/>
      6. <https://scikit-learn.org/0.21/documentation.html>
   2. Colab
      1. <https://colab.research.google.com/drive/1aGRB6RK7Oz_Wg638aR-2mHBWZC90F8vx?usp=sharing>
   3. Research papers
      1. [Indian pothole detection based on CNN and anchor-based deep learning method](https://www.researchgate.net/publication/358593575_Indian_pothole_detection_based_on_CNN_and_anchor-based_deep_learning_method)
      2. [Detecting Potholes Using Simple Image Processing Techniques and Real-World Footage](https://www.researchgate.net/publication/279538022_Detecting_Potholes_Using_Simple_Image_Processing_Techniques_and_Real-World_Footage)