# “Road Pothole Detection”

## Introduction

Roads are widely used transport media that support a variety of businesses, people and are critical for the further development of the country. Damage to the integrity of road networks will significantly impact individuals, communities, business organizations and other national entities.

Road defects occur in various varieties and along with multiple structural anomalies. Prevalent road defects include Cracking, Rutting, Surface deterioration and Potholes. Cracking occurs in various types such as longitudinal, transversal and alligator which result due to sudden variations in temperature, aging of roads and traffic loads in a given timeframe. Rutting is another defect that occurs due to heavy loads of traffic forming groves into roads that lack in construction design. This compromises smoothness and grip offered to vehicles which further leads to risks associated with road traffic accidents. Surface deterioration occurs overtime gradually decreasing skid resistance. This defect is often correlated with roads built out of poor quality of asphalt and lack of maintenance.

Generally, potholes start to form due to the force of water pouring into other defects such as Cracking on roads which leads to formation of rivets in roads. Further contact of roads with high-speed vehicles, further contact with rain and other debris leads to the expansion of these potholes significantly. Potholes act like cancer where formation of one pothole subsequently leads to the formation of multiple other faults in the same section. Water seeps through formed pores in roads and internally starts eroding the construct which leads to the weakening of the road entirely, from the inside out.

There are serious consequential impacts of the formation of these defects which impact machines that operate over these roads on a daily basis, indirectly impacting personal and organizational expenditure on resources. It is found that decrease in quality of roads impacts vehicle transportation and maintenance to a large extent. A study shows that potholes damage internal vehicle components such as suspensions, inflict damage to tyres and rims and cause misalignment in transportation machines such as two wheelers as well as four wheelers. These faults induced into vehicles can be fatal for the driver as well as for those around them. The Ministry of Road Transport and Highways of India reported over 9300 causalities over a span of 3 years (2015 – 2017) caused directly due to potholes.

As damage to roads progresses in severity overtime, repair costs increase significantly which lead to ridiculously high amounts of projected costs for reforming roads across country. A study revealed that failure to spend 1$ in road repairs typically leads to an increased cost of 7$ to fix the

same piece of road 5 years down the line. Today, an estimated cost of $2.7 Trillion will be required to fix faulty highways and bridge infrastructure in the US.

In the recent years, official methods involve manual reporting of such defects which leads to a long, inconsistent and thus a very time-consuming process of repairing roads. Bringing automation in the reporting process will save a considerable amount of time, leading to faster repairs, and in the long-term efficient expenditure of money on maintenance.

# Table of Content



# Problem Statement

The problem at hand is to identify what features and their corresponding combination of their configurations are suitable for effectively classifying image to determine it contains pothole or not.

# Dataset Description

This system was trained by images of image dataset containing three hundred and sixty seven images of plain roads (roads without potholes) and three hundred and fifty seven images containing potholes. These images were scrapped using google image downloader and are publicly available on Kaggle.

The dataset which we have taken is titled "Pothole and Plain Road Images" on Kaggle. The author of this dataset is Viren. This dataset was last updated in the year 2019. The author of this dataset downloaded these images from google images search results, scrapped them using "google_images_download" library. This dataset has 724 image files which are spread in two folders "train" and "test" both of them are again subdivided into two subfolders "Plain" and

"Pothole" which separates the images of roads with and without potholes. Each image is around 80 to 120 kilobytes of size. It has 367 images of “Plain” class and 357 images of “Pothole” class.

All of the images present in the dataset are of non-uniform sizes, and any kind of pre-processing is not performed on the images of this dataset. And images also contains objects other than potholes such as cars, pedestrians, trash and other types of vehicles such as trucks and bikes.

# Exploratory data analysis (EDA) and Data Pre-Processing.

## importing necessary libraries

```
import os

import cv2
from PIL import Image

import matplotlib.pyplot as plt
import plotly.express as px
import seaborn as sns

import numpy as np
import pandas as pd
```

## for creating a DataFrame of Images with other information as well

```
# paths ------------------------------------------------------------

# train
path_plain_train = "../Dataset/train/Plain/"
path_pothole_train = "../Dataset/train/Pothole/"


# test
path_plain_test = "../Dataset/test/Plain/"
path_pothole_test = "../Dataset/test/Pothole/"

# combined
dataset_paths = [path_plain_train, path_pothole_train, path_plain_test,
path_pothole_test]
```

## function for RGB average intensities extraction

```
def avg_intensities(image_path):
    image = cv2.imread(image_path)
    if image is not None:
        b_array, g_array, r_array = cv2.split(image)
        b = b_array.mean()
        g = g_array.mean()
        r = r_array.mean()
        return b, g, r
    else:
        return np.nan,np.nan,np.nan
```

## creating the dataframe

```
def extractImageData() -> pd.DataFrame:
# [
#     {size: (x * y * channels), resolution: [x,y]},
#     {...}
# ]
    data = []
    for image_path in dataset_paths:
        image_paths = os.listdir(image_path)
        for file_name in image_paths:
            className = ""
            img_data = {}

            # data extraction
==========================================================================================
=========
            # image class extraction
            if("plain" in image_path.lower()):
                className = "Plain"
            else:
                className = "Pothole"

            try:
                img_arr = np.asarray(Image.open(image_path +
file_name))
            except:
                print(image_path + file_name, img_arr.shape)
```

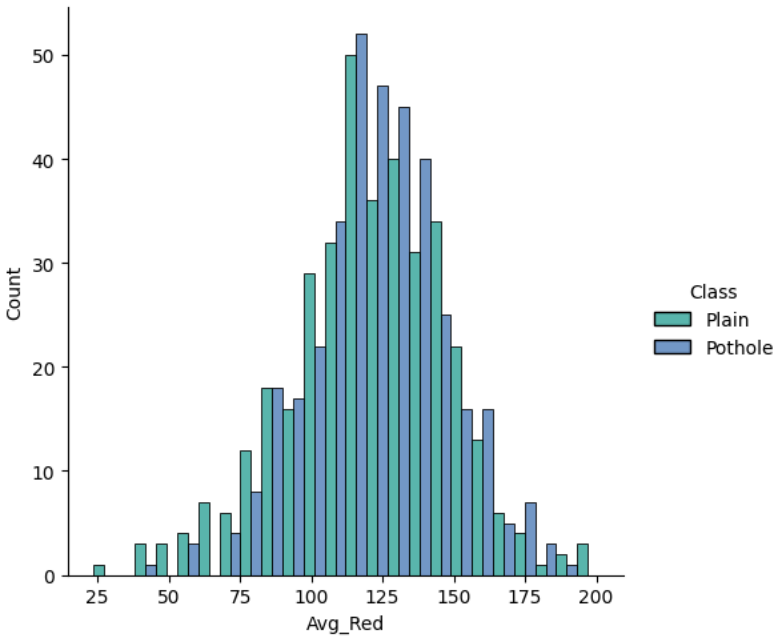
```
                continue

            # Image rgb intensity exrtaction
            b_mean, g_mean, r_mean = avg_intensities(image_path + file_name)

            # image dims extraction
            img_data["Size"], img_data["resolution"] = np.prod(img_arr.shape),
[img_arr.shape[0], img_arr.shape[1]]

            # img data dumping
            record = {
                        "Path": image_path + file_name,
                        "Class": className,
                        "Size": np.prod(img_arr.shape),
                        "Height": img_arr.shape[0],
                        "Width": img_arr.shape[1],
                        "Ratio": img_arr.shape[1] / img_arr.shape[0],
                        "Avg_Red": r_mean,
                        "Avg_Green": g_mean,
                        "Avg_Blue": b_mean,
                    }
            data.append(record)
    image_data = pd.DataFrame(data)
    return image_data
```

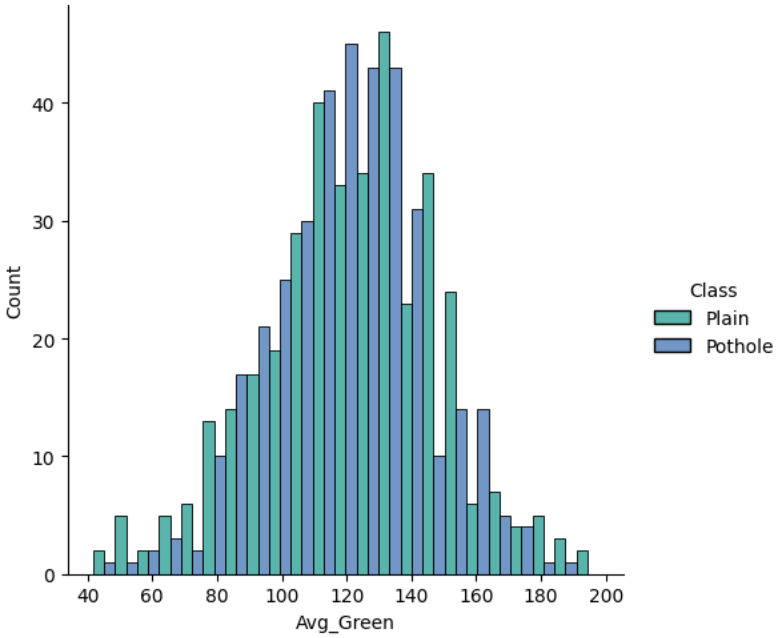
```
image_data = None
if("image_data.csv" in os.listdir(os.getcwd())):
    image_data = pd.read_csv("./image_data.csv")
else:
    image_data = extractImageData()
    image_data.to_csv("image_data.csv")
```

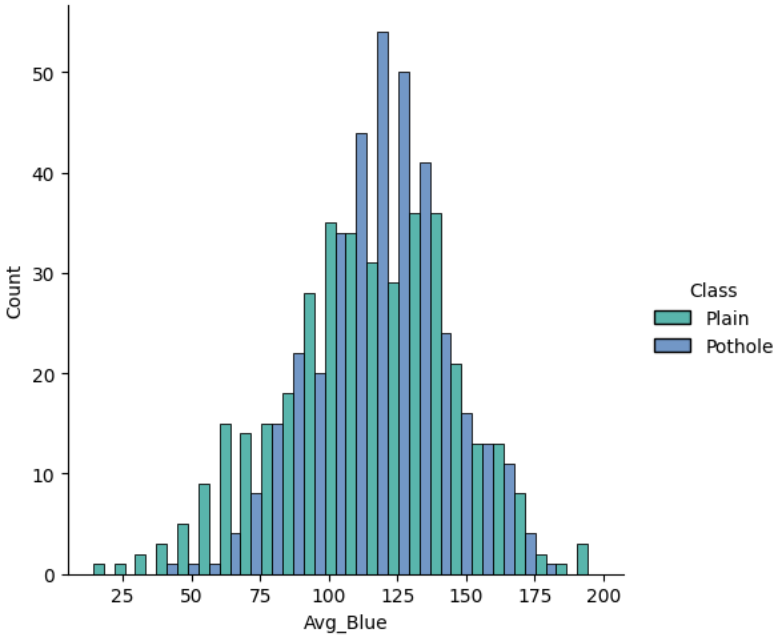
```
image_data.isna().sum()
```

```
Unnamed: 0    0
Path          0
Class         0
Size          0
Height        0
Width         0
Ratio         0
Avg_Red       2
Avg_Green     2
Avg_Blue      2
dtype: int64
```

The question at hand is whether there is a significant difference in the average intensities of the red, green and blue color channels in the images of both the classes or not, can we classify Images on the basis of their average color channels' intensities or not.





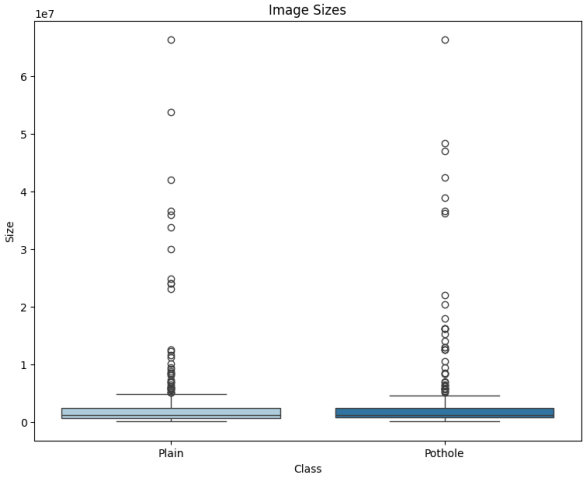

Inference: in the distribution plots of the red and green channels, the difference in their average intensities is not clear, but in the case of blue channel, images in the pothole class have higher average intensity values of blue channel.

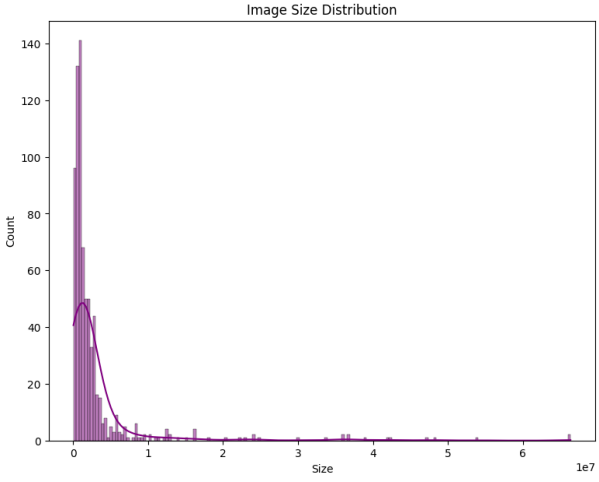
color channel analysis conclusions:

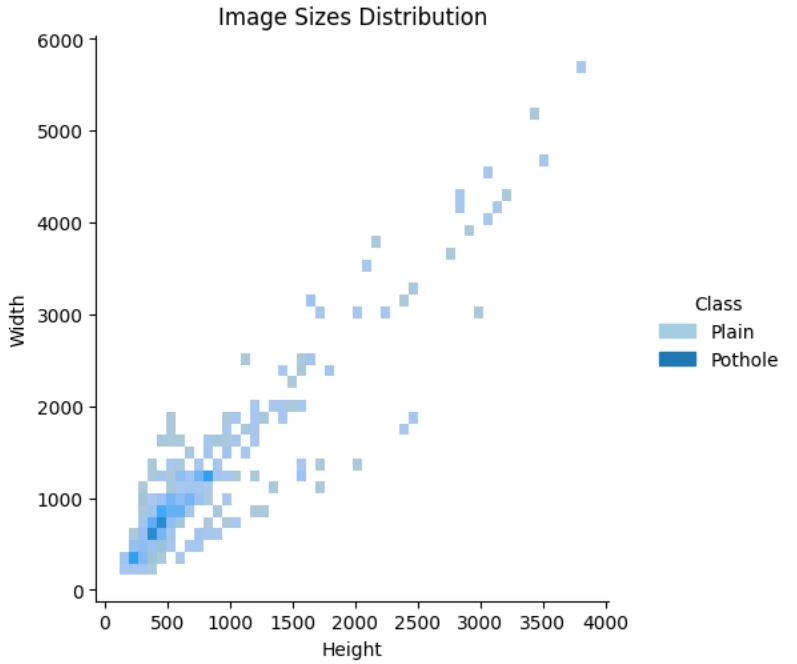
The overall color channel intensities of images in pothole class were higher than those of the plain class, but the difference is not significant enough for it to be use as a feature/parameter for classification task.

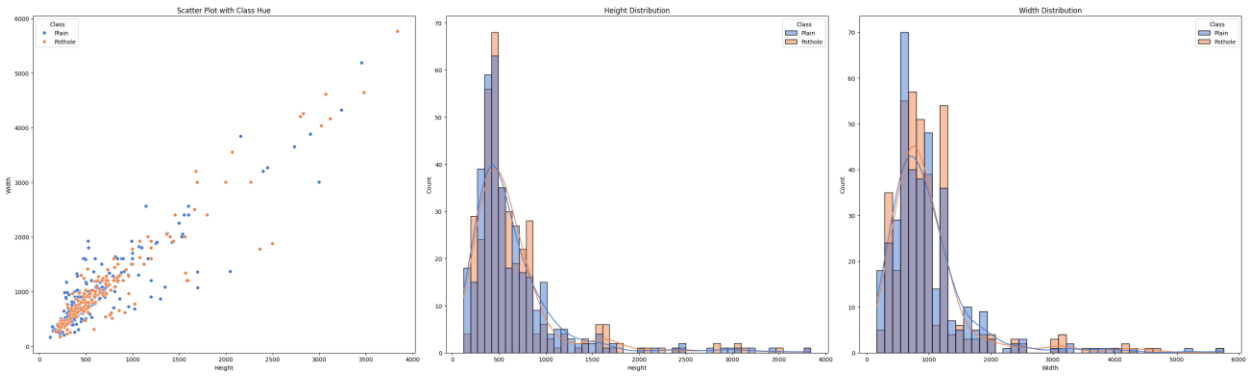
The difference in the blue channel seemed a bit higher by analyzing the distribution plot but from analyzing the violin plot and KDE function of distribution plot, the conclusion made aligned with the inference made from red and green intensities

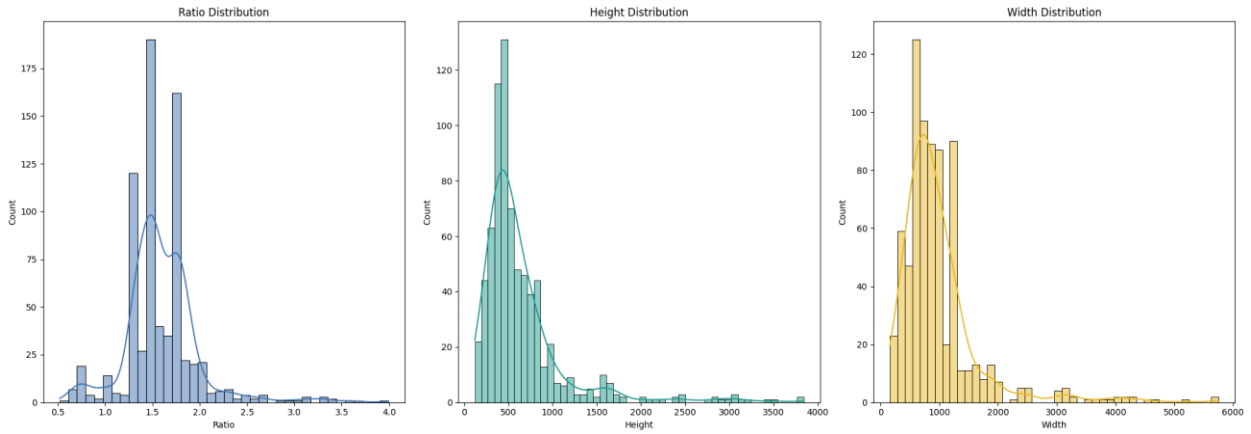
# Image Size Analysis






```
mean_ratio = np.mean(image_data["Ratio"])
scale_height = np.mean(image_data["Height"])
scale_width = scale_height * mean_ratio
print(f"Height: {int(scale_height)}\nWidth: {int(scale_width)}")
```

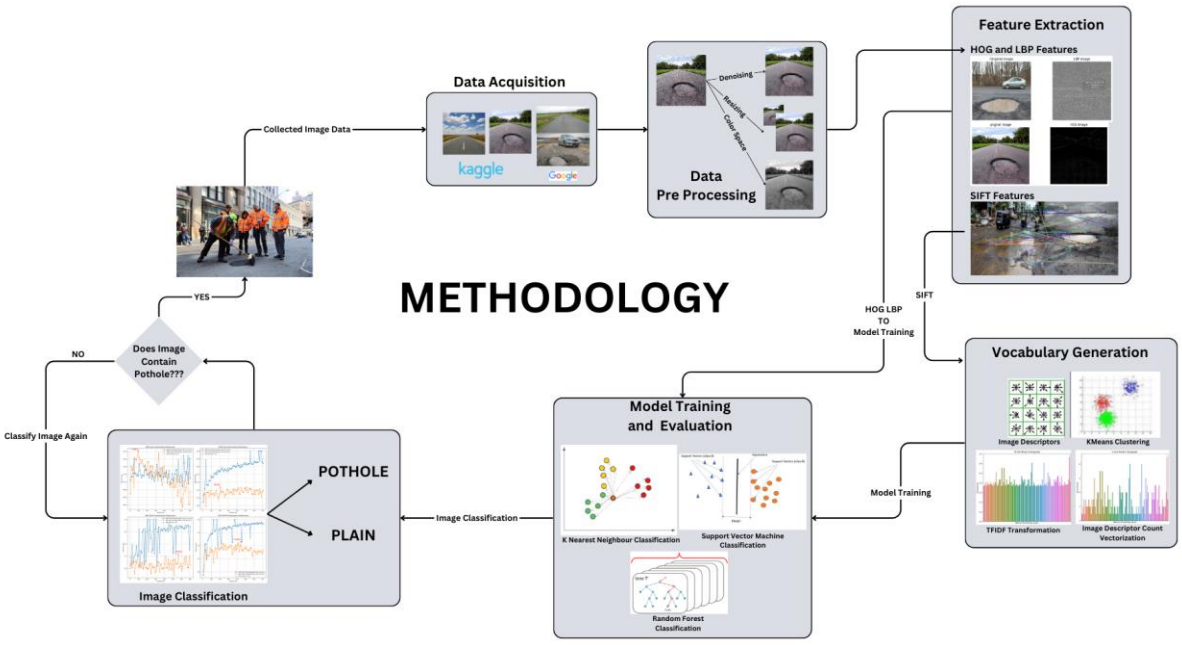
**we found out the best size for image resizing using image size analysis**

```
Height: 646
Width: 1033
```

We decided to proceed with 645*645 as size for every image

For classification task, we have solved this problem using two approaches, Approach 1 using HOG and LBP features, Approach 2 using SIFT features

# Methodology



# APPROACH 1 : HOG and LBP features

## Why HOG and LBP:

**LBPs capture texture information of the image where as capture shape and edge information (gradients) hence they collectively cover two different aspects of the image.**

### # importing necessary libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import plotly.express as px
import seaborn as sns


import os

import cv2
from sklearn.preprocessing import LabelEncoder

from skimage import feature
from skimage.feature import hog
from skimage.feature import local_binary_pattern
from skimage import img_as_ubyte

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy_score, classification_report
from sklearn.preprocessing import LabelEncoder
import pickle

import warnings
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, jaccard_score, roc_auc_score,
confusion_matrix
```

### # creating a data frame

```
traindata_resized_plain_path = "/Users/harshkotadiya/CodeSpace/PROJECTS/EDA_DPL/My Dataset/resized_plain"
```

```
traindata_resized_pothole_path = "/Users/harshkotadiya/CodeSpace/PROJECTS/EDA_DPL/My
Dataset/resized_pothole"
img_path = [os.path.join(traindata_resized_plain_path,filename) for filename in
os.listdir(traindata_resized_plain_path)] + \
   [os.path.join(traindata_resized_pothole_path,filename) for filename in os.listdir(traindata_resized_pothole_path)]
img_class = ["Plain" if "plain" in path.lower() else "Pothole" if "pothole" in path.lower() else "Unknown" for path in
img_path]
data = {"path":img_path,"class":img_class}
dfr=pd.DataFrame(data)
dfr
```

dfr

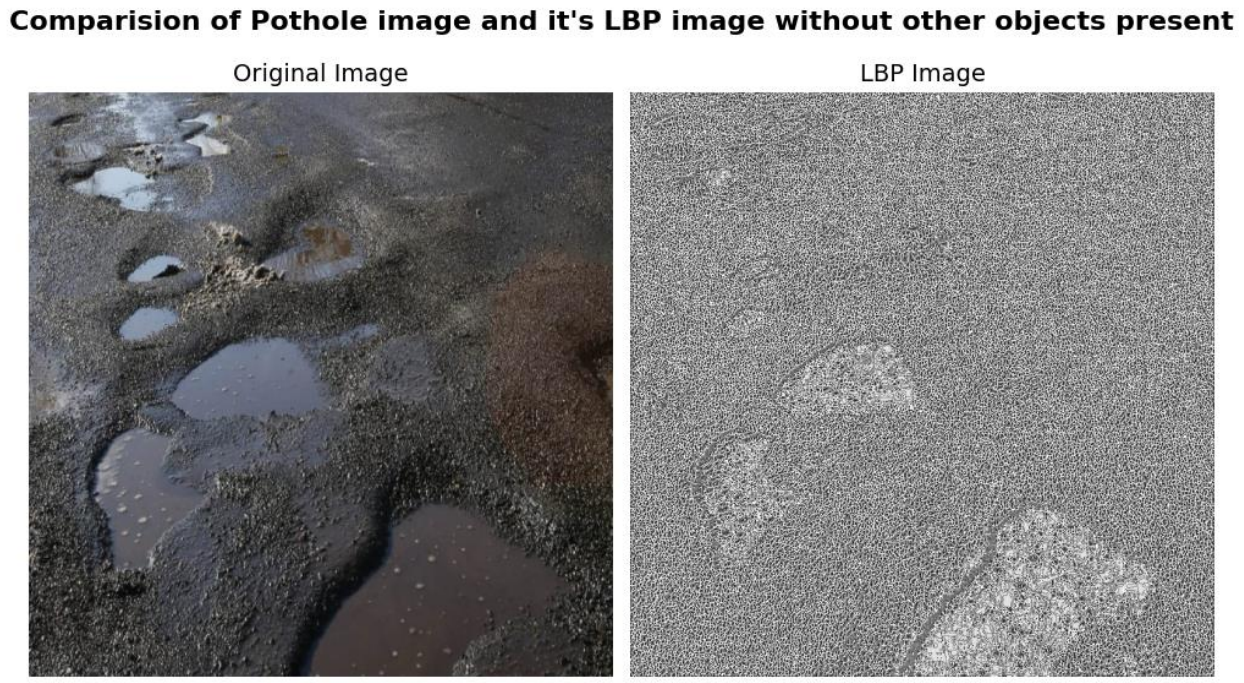
| | path | class |
|---|---|---|
| 0 | /Users/harshkotadiya/CodeSpace/PROJECTS/EDA_DP... | Plain |
| 1 | /Users/harshkotadiya/CodeSpace/PROJECTS/EDA_DP... | Plain |
| 2 | /Users/harshkotadiya/CodeSpace/PROJECTS/EDA_DP... | Plain |
| 3 | /Users/harshkotadiya/CodeSpace/PROJECTS/EDA_DP... | Plain |
| 4 | /Users/harshkotadiya/CodeSpace/PROJECTS/EDA_DP... | Plain |
| ... | ... | ... |
| 695 | /Users/harshkotadiya/CodeSpace/PROJECTS/EDA_DP... | Pothole |
| 696 | /Users/harshkotadiya/CodeSpace/PROJECTS/EDA_DP... | Pothole |
| 697 | /Users/harshkotadiya/CodeSpace/PROJECTS/EDA_DP... | Pothole |
| 698 | /Users/harshkotadiya/CodeSpace/PROJECTS/EDA_DP... | Pothole |
| 699 | /Users/harshkotadiya/CodeSpace/PROJECTS/EDA_DP... | Pothole |

700 rows × 2 columns

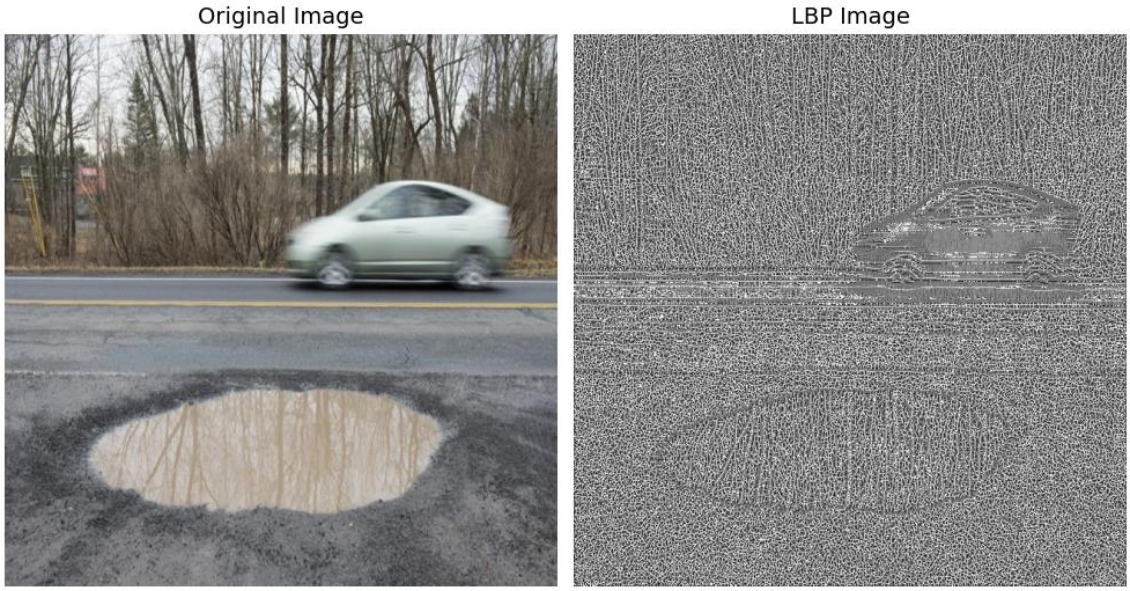
To Check whether lbp features can be used for pothole classification we can visualise lbp image

```
original_image_gray = cv2.imread('/Users/harshkotadiya/CodeSpace/PROJECTS/EDA_DPL/My
Dataset/resized_pothole/339.potholes-700x467.jpg', cv2.IMREAD_GRAYSCALE)
original_image_color = cv2.cvtColor( cv2.imread('/Users/harshkotadiya/CodeSpace/PROJECTS/EDA_DPL/My
Dataset/resized_pothole/339.potholes-700x467.jpg'),cv2.COLOR_BGR2RGB)
radius = 1
n_points = 8 * radius
lbp_image = feature.local_binary_pattern(original_image_gray,n_points,radius,method='uniform')
lbp_image = np.uint8((lbp_image / lbp_image.max()) * 255)
```

```
fig,axes = plt.subplots(1,2,figsize = (10,6))

axes[0].imshow(original_image_color)
axes[0].axis("off")
axes[0].set_title("Original Image", fontsize=14)

axes[1].imshow(lbp_image, cmap='gray')
axes[1].axis('off')
axes[1].set_title("LBP Image", fontsize=14)

figure_title = "Comparision of Pothole image and it's LBP image with the presence of other objects"
fig.suptitle(figure_title,fontweight = "bold",fontsize = 16)

plt.tight_layout()
plt.show()
```



**Comparision of Pothole image and it's LBP image with the presence of other objects**



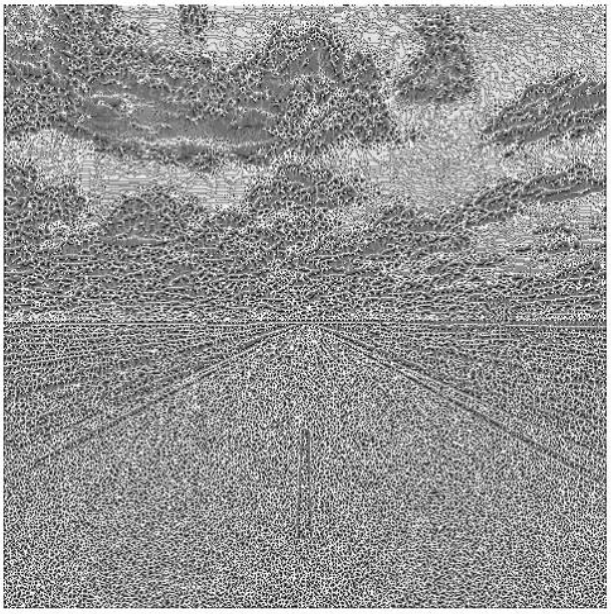

From the plots we can clearly see that Pothole region is highlighted alongside with car. but since it has pothole region highlighted car won't affect the model as the dominant feature/pattern in the Pothole class will still be of pothole

For classification, alongside with identification of pothole features it is also important that plain images have homogenous patterns on the roads since texture of road remains same in the absence of potholes.

**Comparison of Orignal vs LBP images of plain images without any objects**

Original Image

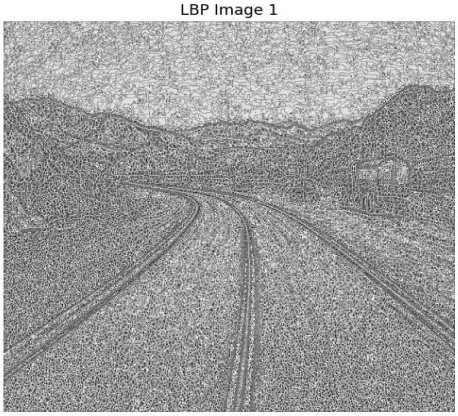
LBP Image



**Comparison of Orignal vs LBP images of plain images with presence of objects**

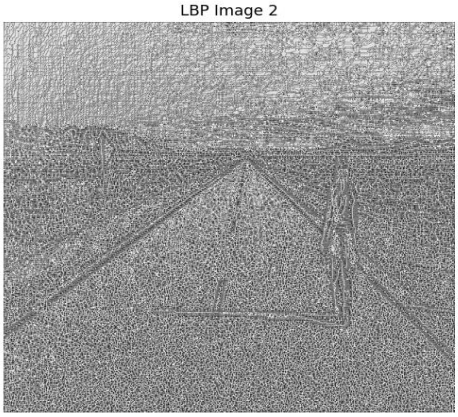
Original Image 1

LBP Image 1



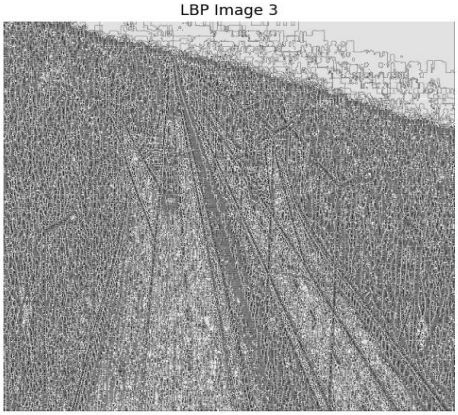
Original Image 2

LBP Image 2



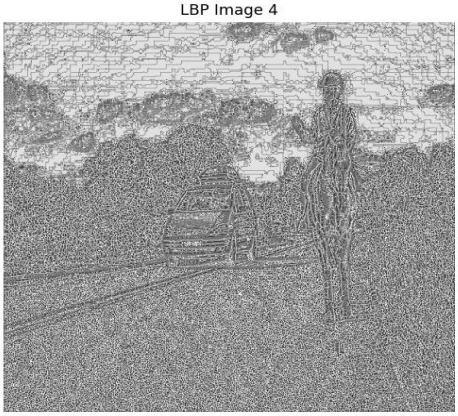
Original Image 3

LBP Image 3



Original Image 4

LBP Image 4



**Conclusion:**

The patterns on the plain images stays homogenous because of the absence of potholes and we can clearly see the difference in the patterns on the roads where potholes can be observed.

**FUNCTION for EXTRACTING LB patterns.**

```
def extract_lbp_features(image_path, radius=1, n_points=8):
    try:

        image = cv2.cvtColor(cv2.imread(image_path), cv2.COLOR_BGR2RGB)
        image = cv2.cvtColor(image,cv2.COLOR_RGB2GRAY)
        image = img_as_ubyte(image)

        lbp_image = local_binary_pattern(image, n_points, radius, method='uniform')

        # Calculating the histogram of the LBP image
        lbp_hist, _ = np.histogram(lbp_image.ravel(), bins=np.arange(0, n_points + 3), range=(0, n_points + 2))

        # Normalizing the histogram
        lbp_hist = lbp_hist.astype("float")
        lbp_hist /= (lbp_hist.sum() + 1e-8)

        return lbp_hist
    except Exception as e:
        print(f"Error processing image '{image_path}': {str(e)}")
        return None
```

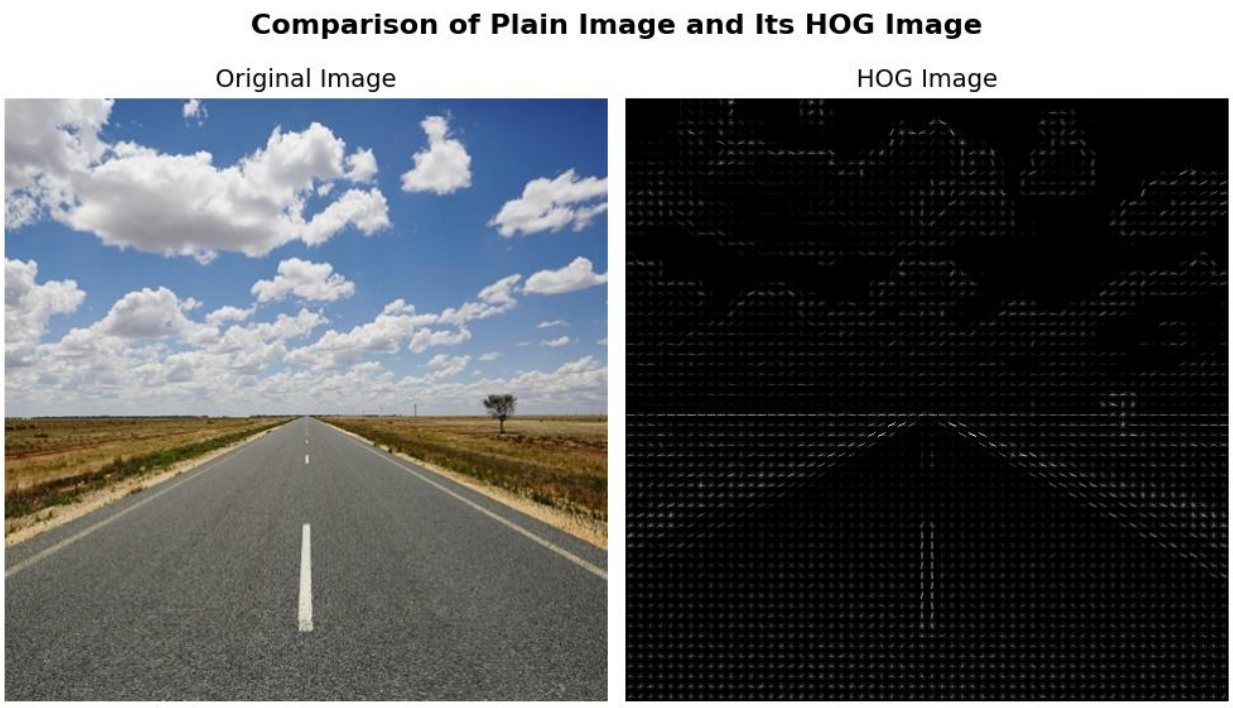
# Histogram of Oriented Gradients

HOG features represent a vector containing shape and edge information of the objects in the image. It analyses the distribution of local gradient orientations in the image.

Same question as above arises where we have to visually check whether the hog features are at least visibly little bit different

```
image_path = "/Users/harshkotadiya/CodeSpace/PROJECTS/EDA_DPL/My Dataset/train/Plain/2.federation-way-1.jpg"
```

```
original_image_gray = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
original_image_color = cv2.cvtColor(cv2.imread(image_path), cv2.COLOR_BGR2RGB)

hog_image = calculate_hog_image(image_path)

fig, axes = plt.subplots(1, 2, figsize=(10, 6))

axes[0].imshow(original_image_color)
axes[0].axis("off")
axes[0].set_title("Original Image", fontsize=14)


axes[1].imshow(hog_image, cmap='gray')
axes[1].axis('off')
axes[1].set_title("HOG Image", fontsize=14)

figure_title = "Comparison of Plain Image and Its HOG Image"
fig.suptitle(figure_title, fontweight="bold", fontsize=16)

fig.tight_layout()
plt.show()
```



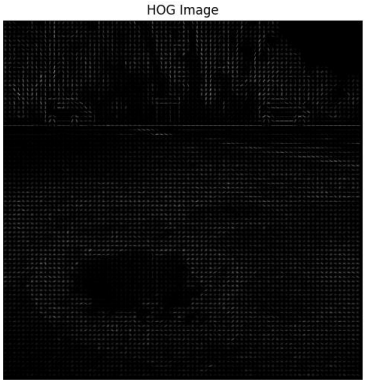
**Comparison of Orignal vs HOG images of plain images without presence of objects**

Original Image

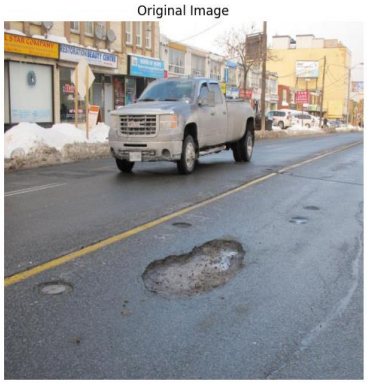
HOG Image

Original Image

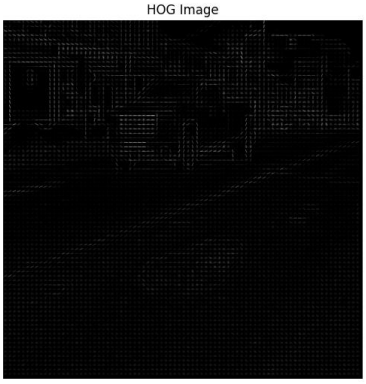
HOG Image



Original Image



HOG Image



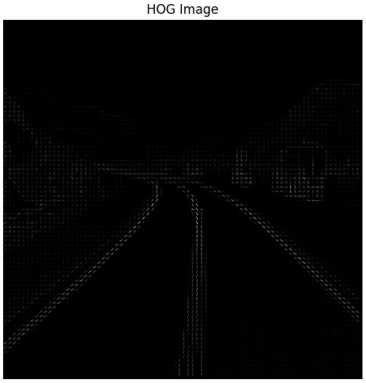
Original Image

HOG Image

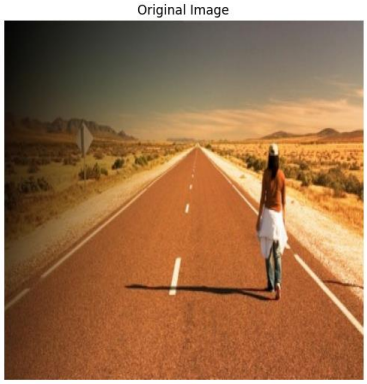
**Comparison of Orignal vs HOG images of plain images with presence of objects**

Original Image

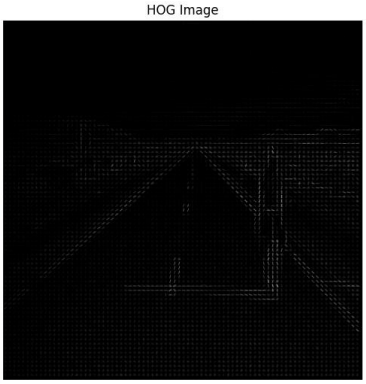
HOG Image



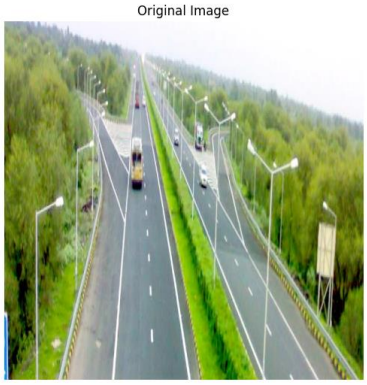
Original Image



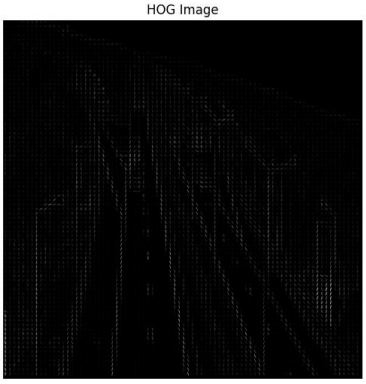
HOG Image



Original Image

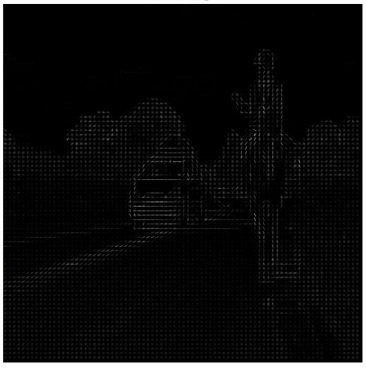


HOG Image



Original Image

HOG Image



**Inference:**

It can be observed that HOG features effectively capture the borders of potholes in form of gradients, whether there is/are any objects' such as cars, are present in the image or not. In case of pothole images and in case of plain images we can see that the gradients captured are different than that or potholes and roads have uniform/no gradient.

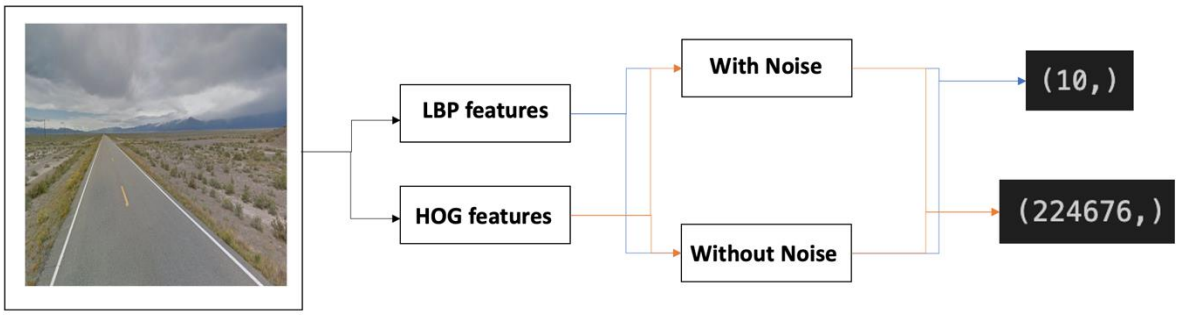
Hence, we can conclude that HOG features will be affective for the task of pothole classification.

**FUNCTION for EXTRACTING HOG with and without NOISE**

```
def calculate_hog_features_noise(image_path):
    image = cv2.imread(image_path)
    image = cv2.fastNlMeansDenoisingColored(image, None, 10, 10, 7, 21)
    image = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
    features, hog_image = hog(
        image,
        pixels_per_cell=(8, 8),  # Cell size in pixels
        cells_per_block=(2, 2),  # Number of cells in each block
        visualize=True,
        block_norm='L2-Hys'
    )
    return features
```

```
def calculate_hog_features(image_path):
    image = cv2.imread(image_path)
    image = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
    features, hog_image = hog(
        image,
        pixels_per_cell=(8, 8),  # Cell size in pixels
        cells_per_block=(2, 2),  # Number of cells in each block
        visualize=True,
        block_norm='L2-Hys'
    )
    return features
```

# Feature Extraction




So we extracted LBP and HOG features of every image with and without noise

And then stored them in the same data frame.

```
dfr["lbp"] = dfr["path"].apply(lambda x: extract_lbp_features(x))

dfr["hog"] = dfr["path"].apply(lambda x:calculate_hog_features(x))

dfr["lbp_noise_removed"] = dfr["path"].apply(lambda x: extract_lbp_features_noise(x))

dfr["hog_noise_removed"] = dfr["path"].apply(lambda x:calculate_hog_features_noise(x))
```

Since hog features are of long length (big dimensions) that's why we had to perform dimensionality reduction on them. Now to make them small in size without losing much variability we used PCA (principal component analysis)

On them and after trail and error method we chose N_components value to be 30 , also before reducing the data we performed standard scaling on them.

```
hog_data = dfr["hog"].to_numpy()

hog_data = np.vstack(hog_data)

scaler_for_hog = StandardScaler()

scaler_for_hog.fit(hog_data)

dfr["hog_scaled"] = list(scaler_for_hog.transform(hog_data))

with open('scaler_for_hog.pkl', 'wb') as file:
  pickle.dump(scaler_for_hog, file)
```

```
lbp_data = dfr["lbp"].to_numpy()

lbp_data = np.vstack(lbp_data)

scaler_for_lbp = StandardScaler()

scaler_for_lbp.fit(lbp_data)

dfr["lbp_scaled"] = list(scaler_for_lbp.transform(lbp_data))

with open('scaler_for_lbp.pkl', 'wb') as file:
  pickle.dump(scaler_for_lbp, file)
```

```
from sklearn.decomposition import PCA
import numpy as np

n_components = 30
pca = PCA(n_components=n_components)

hog_features_array = np.array(dfr["hog_scaled"].tolist())

hog_features_pca = pca.fit_transform(hog_features_array)

with open("pca.pkl","wb") as file:
    pickle.dump(pca,file)

dfr["hog_features_pca"] = list(hog_features_pca)
```

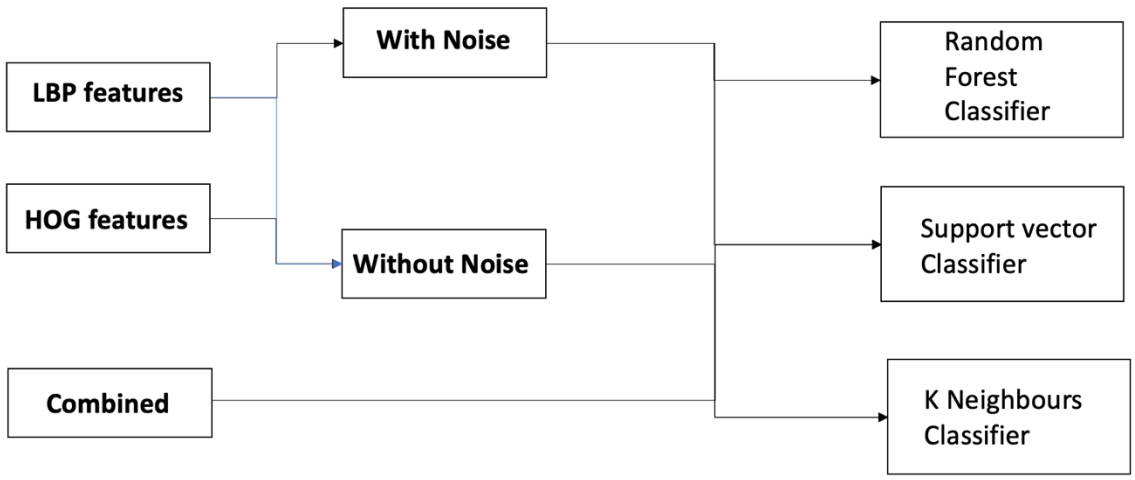
After till this step our data frame looks like this

| | path | class | lbp | hog | lbp_noise_removed | hog_noise_removed | lbp_scaled | hog_scaled | hog_noise_removed_scaled |
|---|---|---|---|---|---|---|---|---|---|
| 0 | /Users/harshkotadiya/CodeSpace/PROJECTS/EDA_DP... | Plain | [0.07231776936482008, 0.10639504837449416, 0.0... | [0.14287851127424933, 0.0992827836832692, 0.15... | [0.03200048074033936, 0.07715642088816592, 0.0... | [0.11048629577442327, 0.05443893213632476, 0.... | [1.1669326007523528, 1.913963638307584, 1.2578... | [-0.6244129628763632, 0.17916328093341138, 0.5... | [-0.3383890736153234, 0.17649271862538857, 0.5... |
| 1 | /Users/harshkotadiya/CodeSpace/PROJECTS/EDA_DP... | Plain | [0.037062676521842745, 0.059931494501530916, 0... | [0.5198449653914823, 0.0, 0.0, 0.0, 0.0, 0.0, ... | [0.013292470404422491, 0.029197764557417772, 0.... | [0.5773502689490632, 0.0, 0.5773502689490632, ... | [-0.40867359577216644, -0.6072478930919404, -0... | [2.8074744767933244, -0.9285432865861625, -1.1... | [2.7992874934160126, -0.5120042544087389, 4.02... |
| 2 | /Users/harshkotadiya/CodeSpace/PROJECTS/EDA_DP... | Plain | [0.034800793221560364, 0.08284357911183023, 0.... | [0.2466929609983344, 0.2466929609983344, 0.194... | [0.003081545580193424, 0.010206117420827783, 0.... | [0.2692838927302386, 0.0, 0.37962807409838656, ... | [-0.509760075953637, 0.6360106851439761, -1.012... | [0.32070961281443905, 1.8238313238041637, 0.99... | [0.7288500527645694, -0.5120042544087389, 2.38... |
| 3 | /Users/harshkotadiya/CodeSpace/PROJECTS/EDA_DP... | Plain | [0.022606814494320712, 0.05850850309476453, 0.... | [0.28753436762091206, 0.28753436762091206, 0.2... | [0.003103178895498994, 0.019318555056784706, 0.... | [0.37088095195308 7, 0.353311221044638, 0.37088... | [-1.0547290717316986, -0.6844624418031932, -0.... | [0.6925281363875665, 2.2795024146439586, 1.828... | [1.4116586015483938, 3.956373369514522 2, 2.310... |
| 4 | /Users/harshkotadiya/CodeSpace/PROJECTS/EDA_DP... | Plain | [0.06847184664382985, 0.08069947719487817, 0.0... | [0.18150618856423603, 0.2688712599780256, 0.27... | [0.006196742984195512, 0.01805660717504914, 0.... | [0.13480663590515257, 0.0, 0.0476613423830505,... | [0.9950521947770867, 0.5196671421533723, -0.55... | [-0.27274814212843346, 2.0712765154558213, 1.8... | [-0.1749381206289038, -0.5120042544087389, -0.... |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 695 | /Users/harshkotadiya/CodeSpace/PROJECTS/EDA_DP... | Pothole | [0.10658253710714244, 0.10079682711375276, 0.0... | [0.2473467669385689, 0.1354722327211811, 0.065... | [0.05389099212787563, 0.06863770206117256, 0.0... | [0.29595878069607634, 0.0, 0.2863755897230311,... | [2.6982795927382415, 1.6101922127188883, 0.973... | [0.32666183580342273, 0.5829320803870605, -0.4... | [0.9081253357328839, -0.5120042544087389, 1.60... |
| 696 | /Users/harshkotadiya/CodeSpace/PROJECTS/EDA_DP... | Pothole | [0.05869358812571219, 0.08480499969953524, 0.0... | [0.3708011263367391, 0.04871847573128 43, 0.018... | [0.014393365783305946, 0.03584640346132 9585, 0.... | [0.46484311585203664, 0.0, 0.0, 0.056148325842... | [0.5580462625849723, 0.7424415495838109, 1.044... | [1.4505853594050397, -0.3849870573485981, -0.9... | [2.043154919091421 6, -0.5120042544087389, -0.7... |
| 697 | /Users/harshkotadiya/CodeSpace/PROJECTS/EDA_DP... | Pothole | [0.0369713358572192 3, 0.06840935039961377, 0.0... | [0.009606554950210115, 0.014320606809096045, 0.... | [0.008138933958295 58, 0.02971936782645202, 0.0... | [0.006815354257358575, 0.0, 0.0, 0.0, 0.436521... | [-0.41275575541407905, -0.14722135034770198, 0.... | [-1.83771549410488, -0.7687670431650748, -0.92... | [-1.0351356691348927, -0.5120042544087389, -0.... |
| 698 | /Users/harshkotadiya/CodeSpace/PROJECTS/EDA_DP... | Pothole | [0.04224505738837708, 0.07088997055465246, 0.0... | [0.10208678381422719, 0.013331284144772516, 0.... | [0.013516014662580048, 0.03603148849227724, 0.... | [0.09288181781367008, 0.0, 0.2107470377363576... | [-0.17706474872049513, -0.012617610026734463, ... | [-0.9957792093512888, -0.7798050013287299, 1.5... | [-0.4567043884524618 6, -0.5120042544087389, 0.... |
| 699 | /Users/harshkotadiya/CodeSpace/PROJECTS/EDA_DP... | Pothole | [0.07265188390120611, 0.09102818340243757, 0.0... | [0.09087878641540674, 0.0, 0.0, 0.0, 0.0151464... | [0.018861847244756473, 0.03801454239528783, 0.... | [0.08057320576890276, 0.0, 0.0, 0.0, 0.0152193... | [1.1818647110214542, 1.0801247702146115, 0.782... | [-1.0978163669730125, -0.9285432865861625, -1.... | [-0.5394275065438622, -0.5120042544087389, -0.... |

700 rows × 9 columns

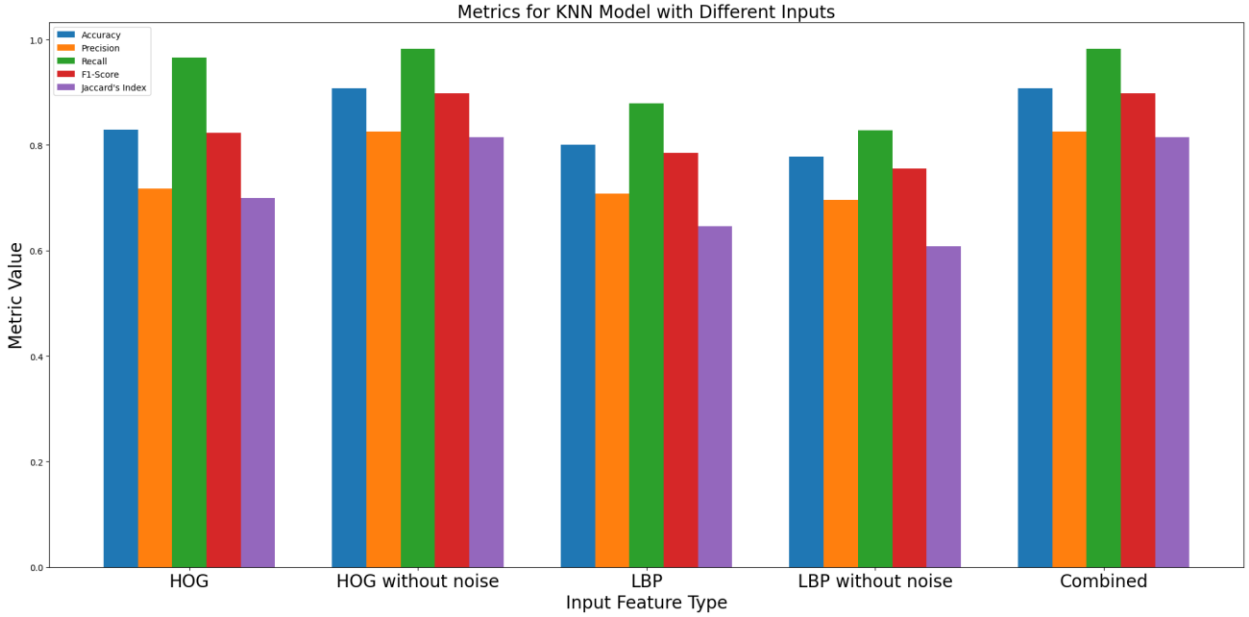
Now that we had the features ready to be plugged into classifiers for classification , We tried and tested different classification models using different combinations of features.

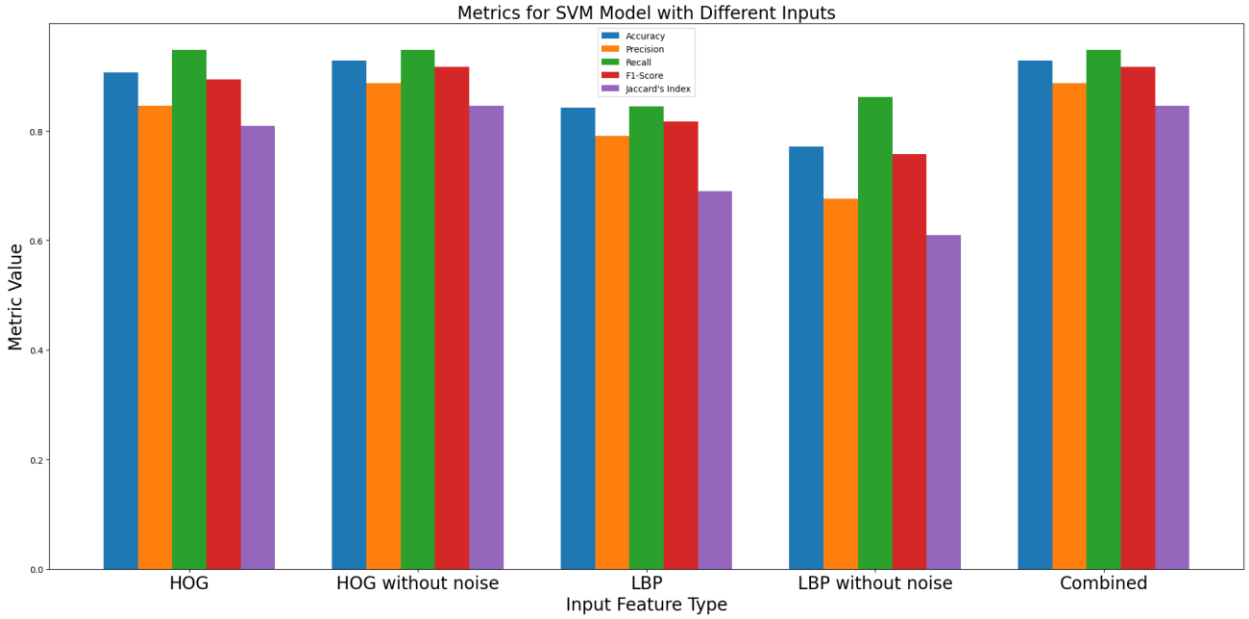
# Model Training and Testing

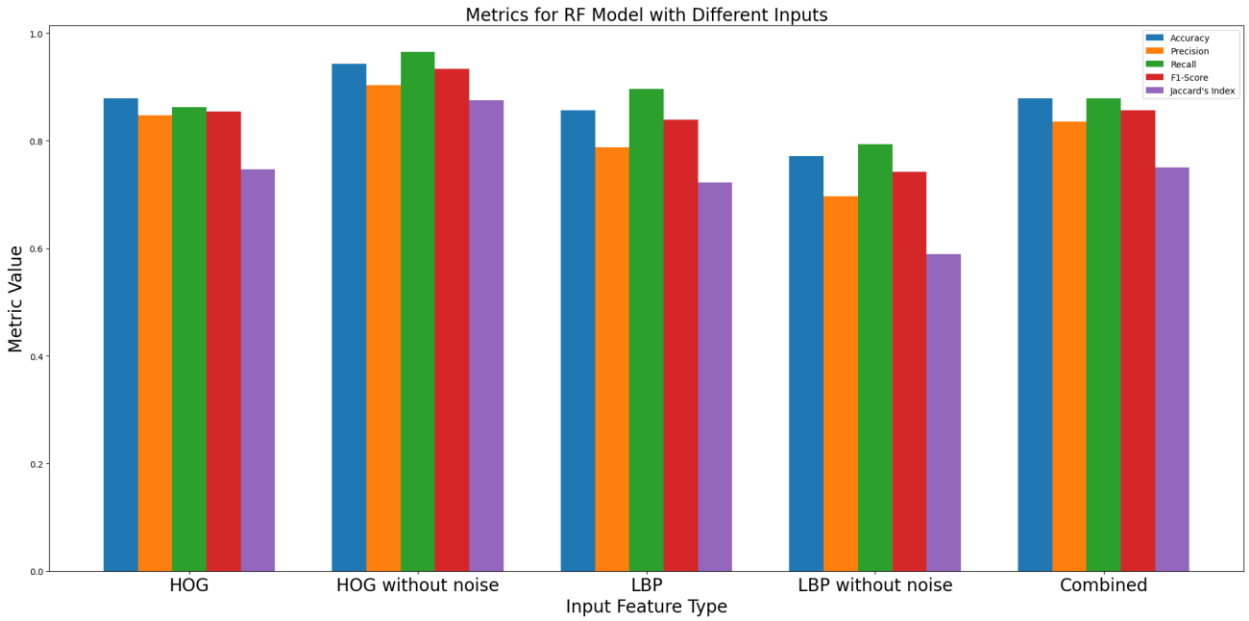



We performed grid search CV for each model and each combination and found out the best parameters for each and then calculated different model performance matrices for comparision analysis of models and feature combinations

**We then visually compared each combination and then selected the most optimal model and feature combination.**






After comparison , we concluded that the best feature out of HOG and LBP features for classification task is HOG features and in HOG features also features after removal of noise lead to a higher accuracy across all three classifiers. And the best classifier is Random forest classifier. Since we have deployed this system so the deployed system will use Random forest classifier with HOG features of image after removal of noise for image classification.

# APPROACH 2: Bag of Visual Words Using SIFT Features

## Why SIFT?

Image content is transformed into features that are invariant to

- image translation
- Rotation
- Scale

**They are partially invariant to**

- illumination changes
- affine transformations and 3D projections

**Suitable for detecting visual landmarks**

- from different angles and distances
- with a different illumination

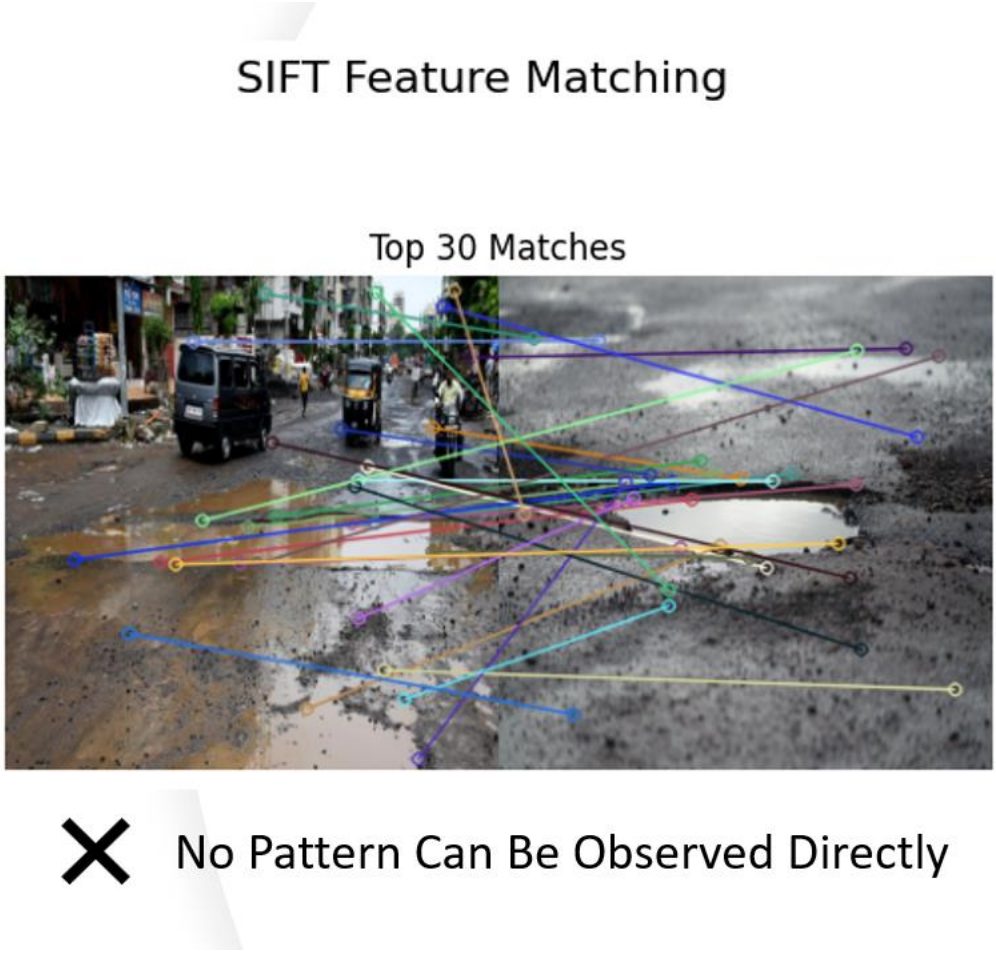
We use 2 main classes of information returned by a SIFT Feature extractor

1) **Key point:** Information about the location of descriptor in the image.
2) **Descriptor:** Information about the region around the key point.

# Image Pre – Processing


We convert all images to a uniform size of a lower resolution and shift it to the grayscale colour space. This means less data to process and uniformity in the dataset.

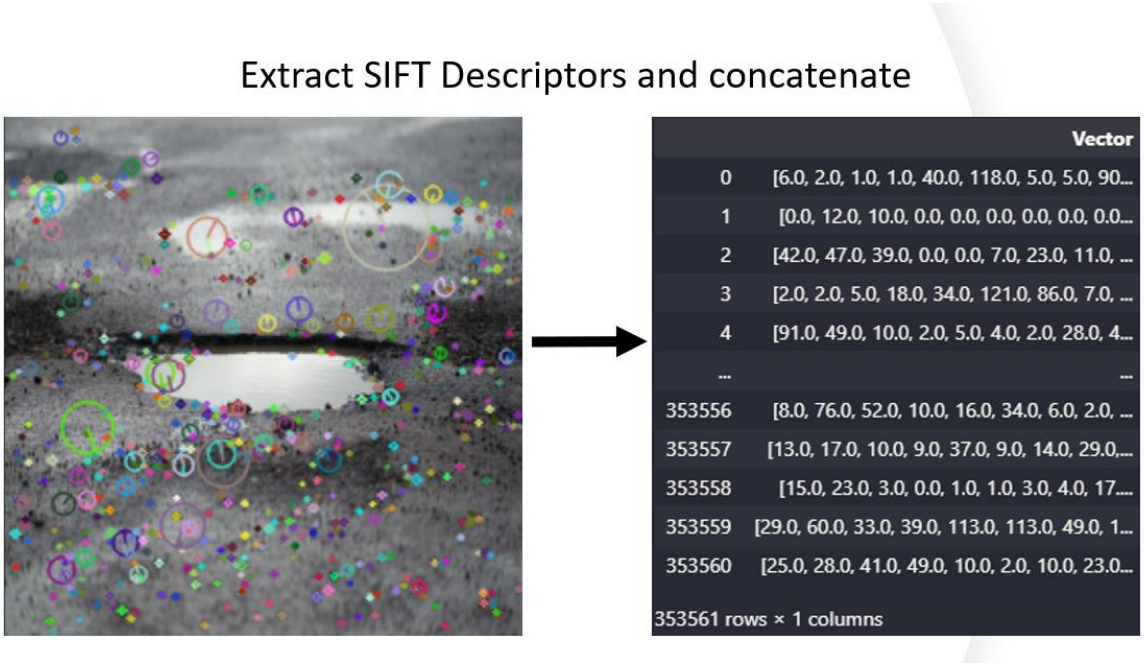
**Why JUST SIFT won't cut it**



As it is obvious, SIFT fails to directly match potholes of varying sizes, shapes and orientations in 3D space. So, we need to find out a different way that utilizes SIFT Features in a more effective way. Not to mention that each image has a different number of descriptors which makes the data even more inconsistent.

The bag of visual words is one way to go about where we group descriptors and classify images based on how often each group occurs. This solves the problem of inconsistent data for each image which we will discuss ahead.

So, the next step is to group these descriptors into clusters using Unsupervised Machine learning techniques. We Concatenate All descriptors together for further processing.



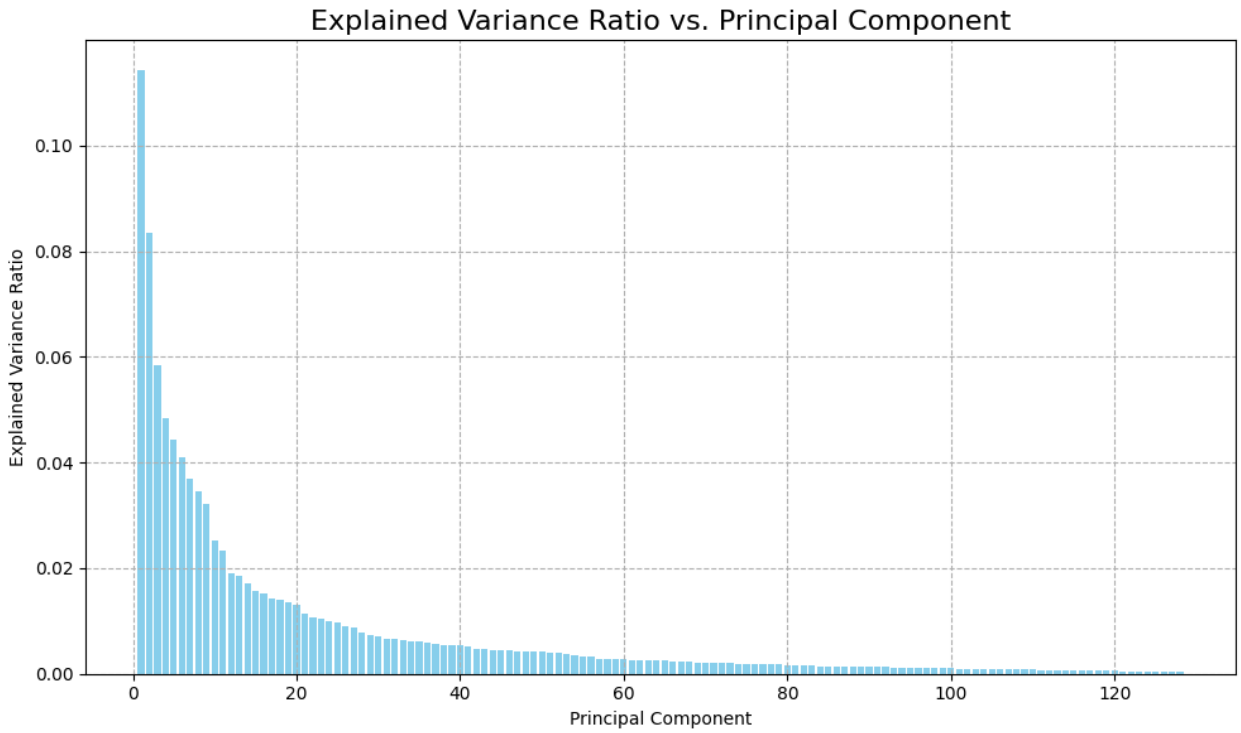

N x 128 descriptor matrix per image combined

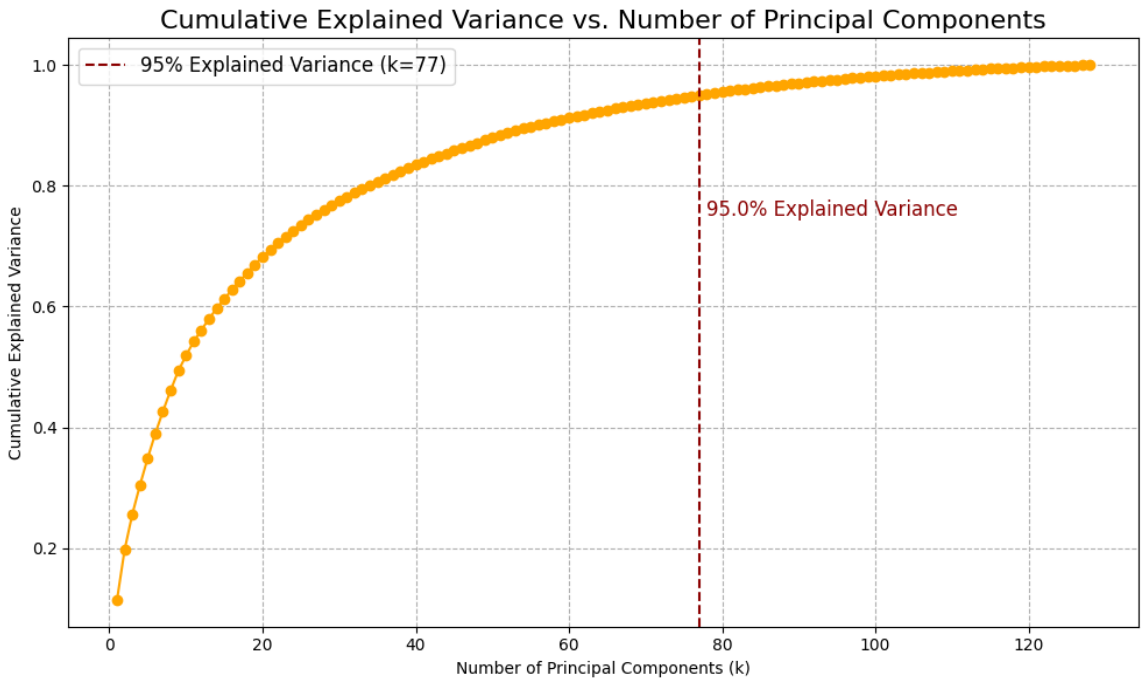
Total 350000 x 128

Given this concatenated data, we realize the size is HUGE. With data having 128 Features, our models are highly prone to overfitting as well as taking up lots of time for computation. So, we decide to reduce this data into a simpler form using principal component analysis.

The idea here is to find and keep only those principal components that contribute to more or less all the variance and leave out the ones that give insignificant amount of variance.

```
pca = PCA()
pca.fit(np.vstack(descriptor_df["Vector"]))

# Desired Explained Variance
desired_explained_variance = 0.95

# Calculate the explained variance ratio
explained_variance_ratio = pca.explained_variance_ratio_

# Calculate the cumulative explained variance
cumulative_explained_variance = np.cumsum(explained_variance_ratio)

# Find the smallest k that achieves the desired explained variance
optimal_k = np.argmax(cumulative_explained_variance >=
desired_explained_variance) + 1
```

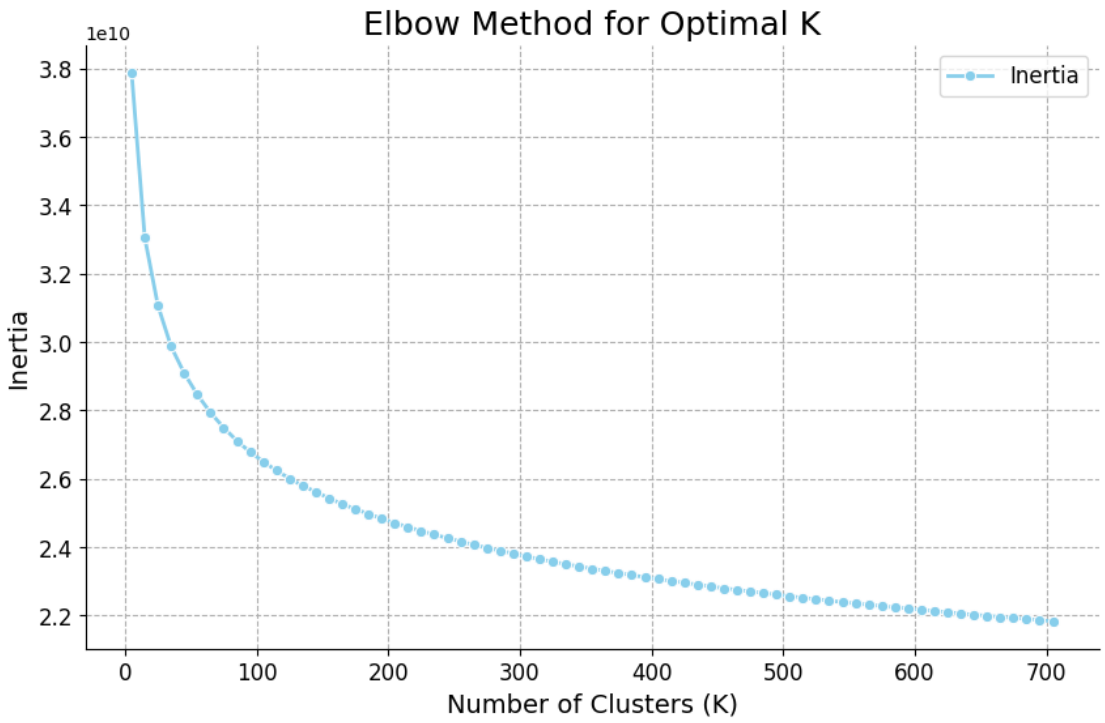





From the given plots, we notice that the first 20 to 25 principle components contribute to a huge amount of variance. Taking the cumulative sum of all explained variance ratios of the principle components, we realize that the first 77 principle components contribute to 95% of variance in the concatinated descriptors. This allows us to drop about 50 FEATURES and still retain 95% of variance, which will reduce computation time significantly.

Once we finally have our reduced descriptors, its time to cluster them using KMeans clustering. To find the optimal number of clusters, we can use the elbow method taking Inertia, or the within cluster sum of squared distances as the evaluation metric.

```
kmeanModel = KMeans(n_clusters=k)
kmeanModel.fit(reduced_descriptors)
inertias[k] = kmeanModel.inertia_
sns.lineplot(x=list(inertias.keys()), y=list(inertias.values()), marker='o',
linewidth=2, color='skyblue', label='Inertia')
```




$$Inertia = \sum_{K=0}^{i} \left( \sum_{x=0}^{j} \left( x^j - c^i \right)^2 \right)$$

*where*;

$i$ = Number of clusters
$j$ = Number of Elements in the $i^{\text{th}}$ Cluster
$x^j$ = Position of the $j^{\text{th}}$ Datapoint of $i^{\text{th}}$ Cluster
$c^i$ = Centroid of the $i^{\text{th}}$ Cluster

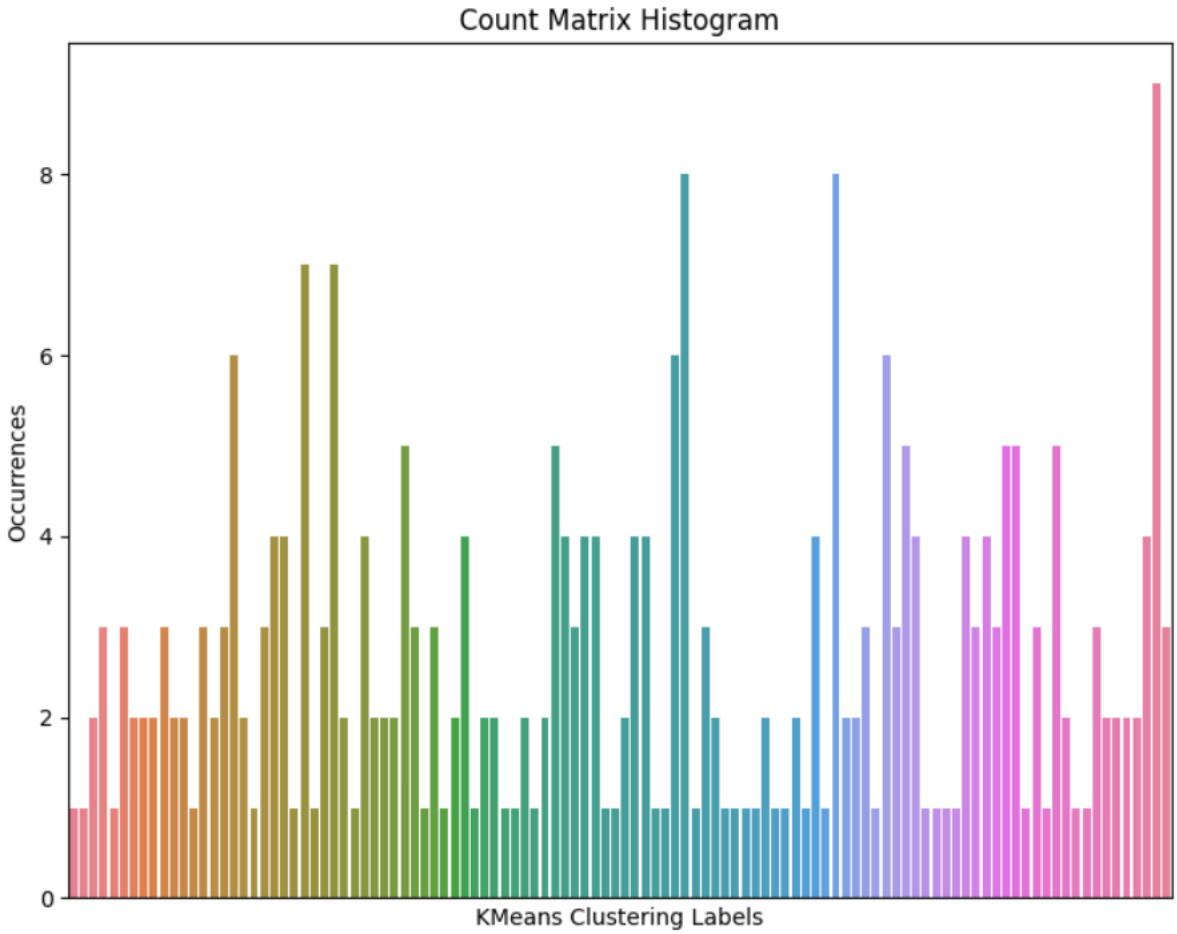
We notice that there is no obvious inflection point in the graph, so we will have to end up trying all number of clusters with the further steps to assess the best models.

Once we have built clustering models to classify individual descriptors into classes, we take each image and build a histogram which shows which descriptor class occurs how many times in that image, for all images in our dataset. This will give as a 1xK dimensional histogram where k is the number of clusters the clustering model that you are using to predict classes has.

```
# create array of image descriptors (not flattened)
    for i in range(len(paths)):
        for image_path in paths[i]:

            # read image array grayscale color space
```

```
            img_arr = cv2.imread(image_path, 0)

            # extract descriptors from image
            keypoints, descriptor = extract_sift_features(img_arr)

            # skip image because pca cannot be done
            if(descriptor.shape[0]) < 77:
                continue

            # add image descriptors
            descriptor_list.append(descriptor)

            # add image class
            if(i == 0):
                image_class.append("Pothole")
            else:
                image_class.append("Plain")

    for i in descriptor_list:
        reduced_descriptors = pca.fit_transform(i)
        class_labels = KMeansModel.predict(reduced_descriptors)
        class_list.append(class_labels)

    count_vectorizer = CountVectorizer(tokenizer=lambda x: x.split())
    tfidf_transformer = TfidfTransformer()
```

The histogram for a kmeans model of 50 clusters for a random image looks like this




Here, the x axis are the different classes of descriptors and Y axis is how many times they occurred for an image.

Here now we have a good, consistent feature for our images that can be further used to classify our images. But we never accounted for the fact that our images contain objects other than potholes, such as trash, people and vehicles!

We notice that lots of images contain these objects and we need to find a way to tell our model to give these objects less importance.

To solve this issue, we can use tfidf transformation to reweight the count vectors for each image.

$$\text{tf-idf}_{t,d} = \text{tf}_{t,d} \times \text{idf}_t = \text{tf}_{t,d} \times \log\left(\frac{N}{df_t}\right)$$

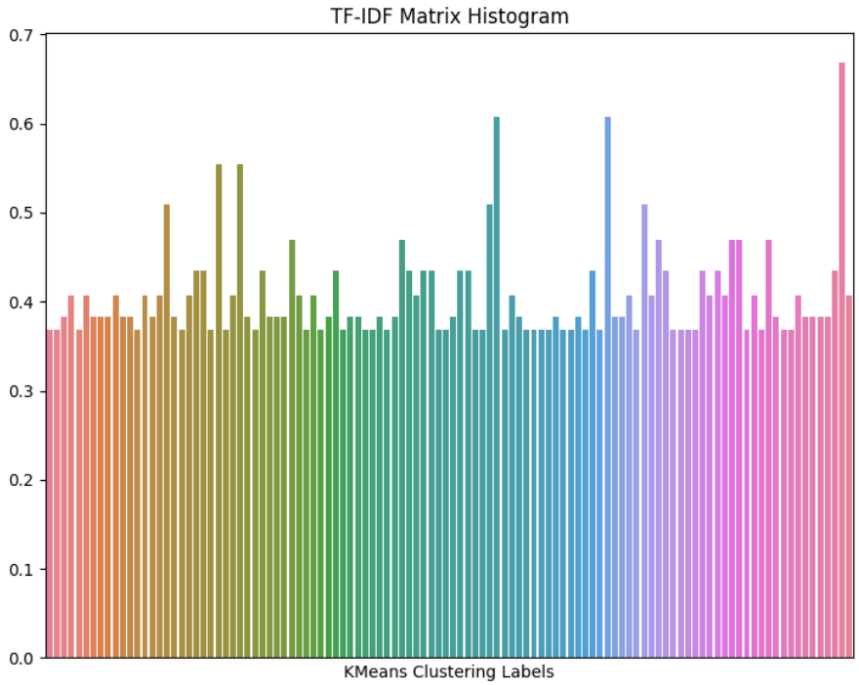
$\text{tf}_{t,d}$ = number of times visual word t occurs in the image d

N = total number of images

$\text{df}_t$ = number of images containing the visual word t

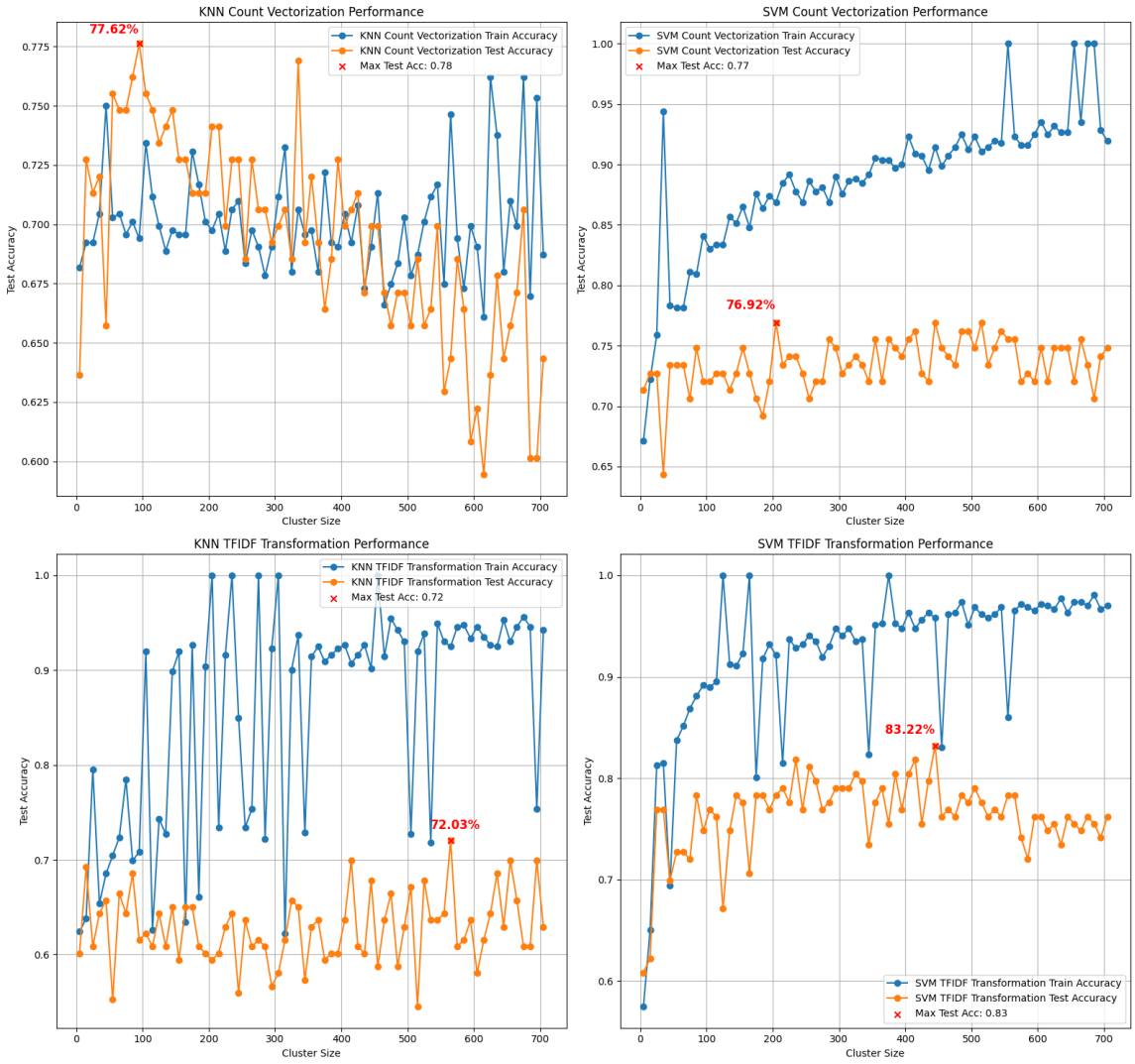
Notice how increase in the document frequency (how many images have that descriptor) will lead to a smaller weight multiplied to the term frequency element (our calculated count vectors / histograms), giving that descriptor lesser importance.

Its graphical representation is as follows




We notice that less occurring classes are given relatively more weight now and have more influence over prediction outcomes.

This data can then be put into KNN and SVM classifiers for training. Remember how we trained multiple kmeans models, we train KNN and SVM models for the count vectors and tfidf vectors we calculate using ALL the KMeans models we trained. This can be very time consuming; the results are as below.




KNN Count Vectorization Accuracy : 0.7762237762237763 at KMeans Cluster Size : 95

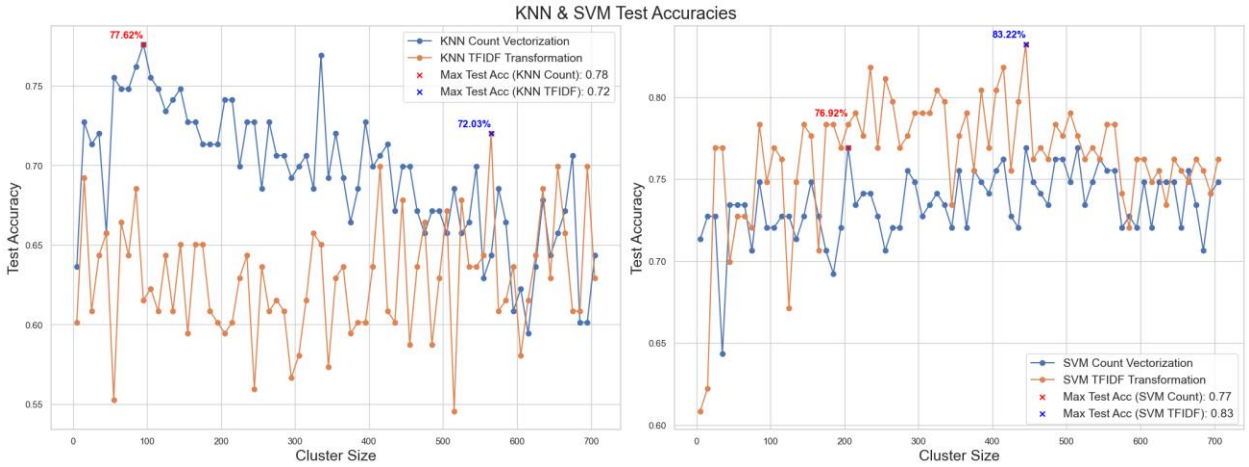
SVM Count Vectorization Accuracy : 0.7692307692307693 at KMeans Cluster Size : 205

KNN TFIDF Transformation Accuracy : 0.7202797202797203 at KMeans Cluster Size : 565

SVM TFIDF Transformation Accuracy : 0.8321678321678322 at KMeans Cluster Size : 445

We notice that the best performing model is SVM with TFIDF Vectors using Kmeans model with cluster size of 445.

Further we make comparisons to see which model worked will with which type of feature.




We notice that KNN works better with Count vectors while SVM works better with tfidf vectors.

Overall, SVM performs much better with TFIDF vectors.

Another observation is that the more SIFT features we extract, the better-quality results we get. This will be observed when we perform the same steps on all images resized to a resolution larger than 256x256 from the current approach.

**LINKS TO THIS PROJECT:**

Project python notebooks and deployment code:

https://github.com/devilb2103/Road-Pothole-Detection

dataset link:

https://www.kaggle.com/datasets/virenbr11/pothole-and-plain-rode-images