CS 512 Computer Vision Project Report

Student 2: Vismaya Veeramanju Kalyan ID: A20423189
Student 1: Chethan Kothwal ID: A20423243

Name of the paper: Computer Vision Based Detection and Localization of Potholes in Asphalt Pavement Images Authors: Kanza Azhar, Fiza Murtaza, Muhammad Haroon Yousaf, Hafiz Adnan Habib 2016 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)

ABSTRACT

Asphalt pavement distresses have significant importance in roads and highways. This project addresses the detection and localization of one of the key pavement distresses, the potholes using computer vision. Considering the appearance-shape based nature of the potholes, Histograms of oriented gradients (HOG) features are computed for the input images. Features are trained and classified using Naïve Bayes classifier resulting in labeling of the input as pothole or non-pothole image. To locate the pothole in the detected pothole images, canny edge detection and active contours are used.

PROBLEM STATEMENT

Evaluation of damages in pavements is mandatory for the maintenance of road networks. Real time maintenance requires efficient supervision of pavement surfaces. This process may also be subjective from one person to other. To ensure road surface quality it should be monitored continuously and repaired as necessary. The main asphalt pavement distress that we address is pothole. Potholes are one of the significant problems on roads and highways. Small potholes are a nuisance; big ones can damage cars, and even cause fatal accidents. Currently pothole detection is mostly done either through a manual approach by experts or by high-end equipment using 3D reconstruction, stereo-vision, laser and vibration based approaches. The usual method of pavement condition assessment includes three steps. The first step is to collect the data of the pavements. Second step is identification of asphalt pavement distress and the last step is assessment of the distress. Second and third steps are time consuming and cost ineffective as they are performed manually. This project aims to propose a technique for detection of potholes and subsequently to localize them in the image. Visual features of the road are analyzed and the pothole is localized using computer vision that requires much less manual effort and capital investment.

PROPOSED SOLUTION

This project we have chosen implements a way to detect potholes using histogram of oriented gradients and predicting it by training the naive bayes classifier. If a pothole is

detected, localization of the pothole is done using canny edge detector and active contours. The characteristics of a pothole that can be used to differentiate it from the rest of the image. A pothole may contain dry of smooth texture compared to the neighborhood. Overall surface of the pothole can not be the same due to the varying intensities of light reflected by it. Many arbitrary shapes of potholes can be found due to varying amount of wear and tear in the pavement.

Operations to be performed on the image before HOG feature extraction include:

- Convert the image from RGB color space to grayscale.
- Resize the image to 200x200.

STEP 1: HOG feature extraction

HOG feature descriptors are used for object detection and recognition. HOG is mainly based on the distribution of the edge directions and are extensively focused on the shape of the project. Therefore, a local pothole shape is represented using a HOG feature descriptor.

- 1) The horizontal gradient Fx and the vertical gradient Fy are computed using the masks
  Mx = [-1 0 1] My = [[1], [0],[-1]]

- Magnitude of the gradient vector is calculated using

$$|G|_F = \sqrt{F_x{}^2 + F_y{}^2} \ .$$

- Orientation of the gradient vector is calculated using:

$$\theta = \tan^{-1}\left(F_y / F_x\right) \ .$$

- Image size of 200x200 is divided into 625 (25x25) non-overlapping cells of 8 x 8 pixels. Each 8x8 cell is further divided into 4 blocks having 4x4 pixels in each block.
- Orientation calculated are normalized in the range of [0,180] to focus only on unsigned orientations.
- Then gradient orientations histogram is calculated for each block by quantizing gradient orientations into 8 bins. These 8 bins are evenly spaced over the interval [0°, 180°]
- Each pixel has a weighted vote for its corresponding bin based upon the

gradient magnitude and orientation. HOG feature extraction results in a feature vector of size 1x 20000 (625x4x8).

STEP 2: Training the classifier

A dataset of 120 images consisting of pothole and non-pothole images were collected from google image search and captured by roaming around.

- HOG feature vectors were obtained for all the images in the dataset and was then manually labeled as pothole and non-pothole images. The data was then split into training and test data.
- Naive Bayes classifier (Gaussian Naive Bayes) is trained using the training data. The scikit-learn library implementation of the Naive Bayes classifier is used and tested for different parameters that gives the best performance.
- Performance measures like accuracy, precision and recall of the model is checked with the test data.
- Performance measures were determined for few other models in scikit-learn library such KNN, SVM, Bernoulli Naive Bayes.
- A new image is read by the application using cv2.imread() function from the OpenCV library. The trained model is used to predict if the input image contains a pothole or not.
- If a pothole is detected it is then localized using canny edge detection and active contours.

STEP 3: Pothole Localization

A pothole in the asphalt pavement may contain the coarser (dry) or smooth (with water) texture as compared to local neighborhood. Edge detectors like Canny would be a good choice to detect the region on the image where the pothole is located. When the program detects if the image contains a pothole we then pass it through the canny edge detection.

1. Canny edge detector is used for Pothole localization.
    - The image is converted into gray scale and perform gaussian blur to remove some noise before further processing of the image.
    - Determine the intensity gradient.
      This is done using sobel filter being convolved with the image . The intensity of the image changes where an edge occurs.

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * I \qquad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * I$$

- Magnitude and angle of the directional gradients

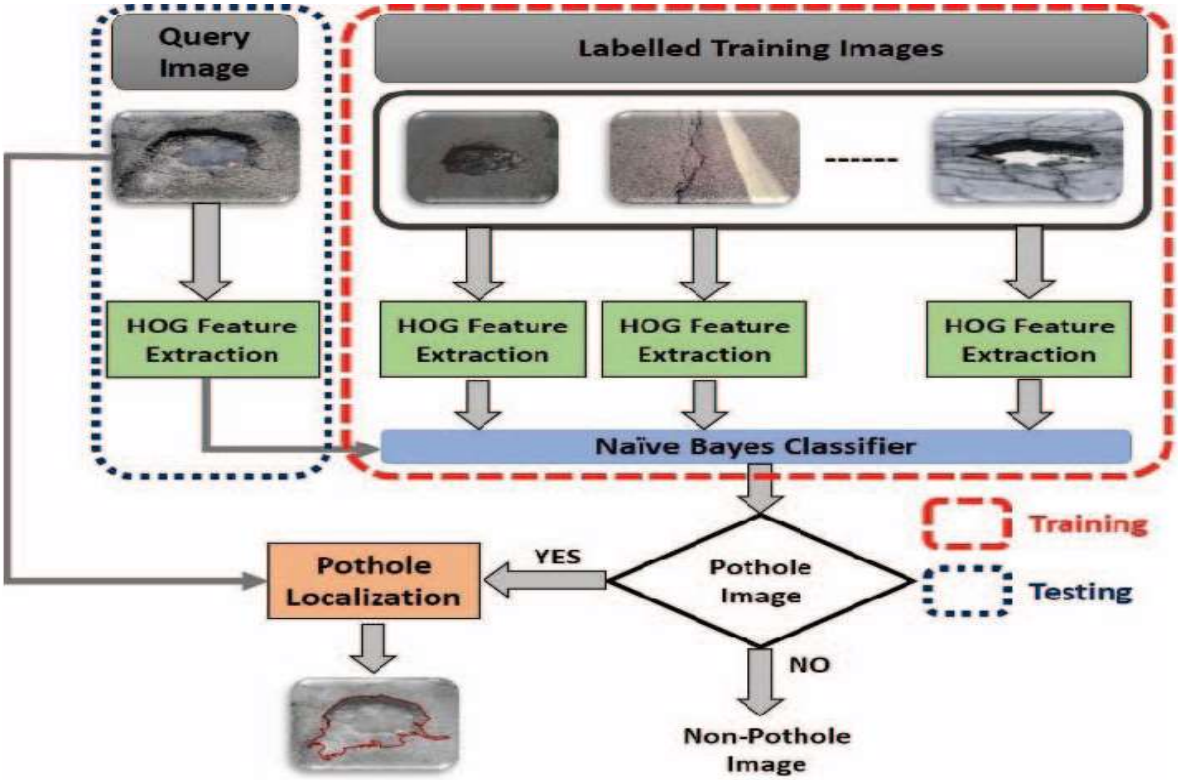
$$|G| = \sqrt{G_x^2 + G_y^2}$$

$$\angle G = arctan(G_y / G_x)$$

- Non-maximum suppression
  The image magnitude results in thick edges. The final image should have thin edges. Thus, we must perform non maximum suppression to thin out the edges. This is done pixel by pixel wise based on the gradient orientations we calculate if the gradient magnitude must be made retained or made 0.
  Alpha = αb + (1−α)a => αb + a −αa => (b − a)α + a

- Double Thresholding
  Once the Non-maximum suppression is carried out we have thin edges but its still contains some noise. Here in this step we choose 2 thresholds (low and high). If you choose the high threshold to be 0.7, this means that all pixels with a value larger than 0.7 will be a strong edge. You might also choose a low threshold of 0.3, this means that all pixels less than it is not an edge and you would set it to 0. The values in between 0.3 and 0.7 would be weak edges, in other words, we do not know if these are actual edges or not edges at all.
- Edge tracking by hysteresis
  From the previous step we will have a clue about which is a weak edge and which is a strong edge. We now have to do a edge tracking algorithm. Weak edges that are connected to strong edge are also strong edges and other weak edges are removed.

2. Edges with high intensity from canny edge detection are marked with contours using cv2.findcontours from opencv library .
3. Pothole is localized on the image by drawing a rectangle around the contours(Pothole).



Fig(a): Process of pothole detection

IMPLEMENTATION DETAILS

Program Design Issues:

1. As dataset that consists of pothole and non-pothole images was not available online and authors of the paper were unable to send us the dataset. The dataset was obtained by browsing through the internet, downloading around 70 images from google image search in which the pothole was notable. For the Non-pothole image we captured around 40 images of the undamaged pavements around the city.

   The HOG features of each image in the dataset was converted into a single array of size 1x20000 using np.ravel() function. The features were then dumped into a pickle (.pkl) file using pickle module from python library. The “Pothole.pkl” file contains the HOG features in which each row represents a image and the columns represent the HOG feature descriptor value of each pixel. The “Labels.pkl” file contains the labels for the images in the dataset. It is a binary classification problem in which label “1” is for pothole present in the image and “0” for non-pothole images.

2. The threshold used in Canny Edge detection sometimes detects the cracks around the pothole making the program detect a bigger region as a pothole. The threshold is chosen based on majority of the images but might fail sometimes.

3. Non-max suppression is a way to eliminate points that do not lie in important edges. Non maximum suppression by interpolating the pixels was done for greater accuracy and gave better localization of the edge.

Instructions to execute the program:
There are six programs present in ./src folder:

Histogram.py : Code for calculating the histogram of oriented gradients

Canny.py: Code for canny edge detection

Create_pickle.py: Code to generate HOG feature vector dataset of the images

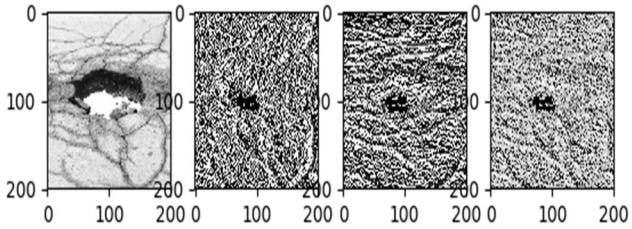
Magnitude_orientation.py: Code to visualize the gradient and magnitude of the input image

Different_classifiers,py: Code to generate performance measures by different classifiers

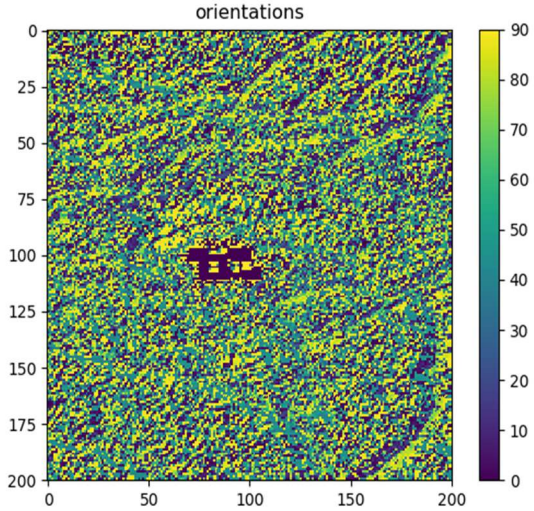
Pothole_predict.py: Code to detect a pothole in a given image

- Execute create_pickle.py to generate the labelled dataset in ./data folder. X.pkl represents the HOG feature vectors of images. Y.pkl is the label for X.pkl. More images can be added into ./data/clean and ./data/pothole folders for testing purposes. Execute create_pickle.py each time after adding new images.
  Python create_pickle.py
- Execute magnitude_orientation.py along with image as a argument to visualize the magnitude and orientation of the image
  Python magnitude_orientation.py “test.jpg”
- Execute pothole_predict.py along with image as a command line argument to predict if the given image contains pothole or not.
  Python pothole_predict.py “test.jpg”
- Execute different_classifiers.py to view the performance metrics for training different classifiers.
  Python different_classifiers.py

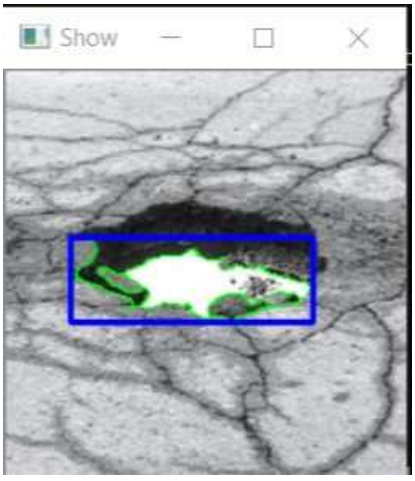
Results:



Fig(b): Gradients of the image along x-axis and y-axis



Fig(c): Orientation of the gradient vector



Fig(d) Localization of the pothole Image.

Fig(e): No pothole detected

Classifier	**Accuracy, Precision and Recall**
Gaussian Naive Bayes	Accuracy = 0.975609756097561 Precision = 0.9655172413793104 Recall = 1.0
Bernoulli Naive Bayes	Accuracy = 0.7692307692307693 Precision = 1.0 Recall = 0.5714285714285714
KNN	Accuracy = 0.8292682926829268 Precision = 1.0 Recall = 0.75
SVM	Accuracy = 0.6829268292682927 Precision = 0.6829268292682927 Recall = 1.0

Table 1: Performance metrics for different classifiers.

The maximum accuracy, precision and recall was obtained for Gaussian Naive Bayes classifier. The results were not comparable to the results obtained in the paper as we were not able to receive the dataset from the publishers. The application developed, predicted all the pothole and non-pothole images correctly without any error. It localizes the pothole on the images on which it predicts the pothole to be detected.

REFERENCES

Paper:
Original paper: https://ieeexplore.ieee.org/abstract/document/7726722

Canny edge detection:
https://www.researchgate.net/publication/279538022_Detecting_Potholes_Using_Simple_Image_Processing_Techniques_and_Real-World_Footage

Histogram of oriented gradients: https://www.learnopencv.com/tag/hog/
https://lilianweng.github.io/lil-log/2017/10/29/object-recognition-for-dummies-part-1.html
https://github.com/JeanKossaifi/python-hog/blob/master/hog/histogram.py

Classifers: https://scikit-learn.org/stable/modules/naive_bayes.html

Accuracy: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html#sklearn.metrics.accuracy_score

Precision: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html

Recall: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.recall_score.html

Canny:
https://www.youtube.com/watch?v=sRFM5IEqR2w
https://stackoverflow.com/questions/13659517/non-max-suppression
https://docs.opencv.org/3.4/da/d22/tutorial_py_canny.html
https://github.com/MadhavEsDios/Canny-Edge-Detector
http://justin-liang.com/tutorials/canny/#grayscale
https://github.com/JustinLiang/ComputerVisionProjects/tree/master/CannyEdgeDetector