Introduction to image processing and computer vision

Project II

Surface Crack Detection

Supervisor: Rafal Jozwiak

Author

M Hasibur Rahman

295459

January 29, 2021

Contents

I Introduction

I.I Image classification

Image classification involves assigning a class label to an image. For example in the medical field, an image classification algorithm can be used to produce output classification for identifying whether a disease is present or not. For this report, I am going to try to classify the given data-set of positive and negative samples of surfaces. Each class has 20,000 samples. For classifications I will use K-Nearest Neighbor, Random Forest classifier, Support Vector Machine, Naive Bayes. These algorithms will map the input data set to a specific category.

Note: I will be using HDF5 [1] to save the feature vectors and labels. Moving on, I will also check the accuracy of the classification as well as a confusion matrix and a classification report. The confusion matrix will give us the insights on the errors being made. Each row of the matrix corresponds to a predicted class and each column column corresponds to an actual class. It also provide the type of errors being made. We can summarize the confusion matrix as such:
$$\begin{array}{c c} & \begin{array}{cc} event & no-event \end{array} \\ \begin{array}{c} event \\ no-event \end{array} & \begin{pmatrix} true-positive & false-positive \\ false-negative & true-negative \end{pmatrix} \end{array}$$
where,

- true-positive” for correctly predicted event values.
- “false-positive” for incorrectly predicted event values.
- “true-negative” for correctly predicted no-event values.
- “false-negative” for incorrectly predicted no-event values.

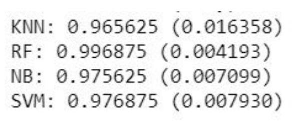
Thus using this matrix we get a classification report giving a breakdown of each class by precision, recall, f1-score and support. This will allow to check accuracy of trained models and will be used to test on our set. This step are repeated for different features to check which is more efficient. The texture feature I will be using are Hu moments, Haralick Texture, Color Histogram, Edge oriented histogram. I will check quality of each features and see which feature is the best and most efficient.

II Algorithm Description

For learning and classification evaluation, training and testing set is divided as such - 80% training / 20% testing. In the upcoming subsections I will be describing the features and how it was implemented with short code snippets. Before we get to that, I will mention how the solution is run and the description of the components of the folders in solution (attached folder) and what they do:

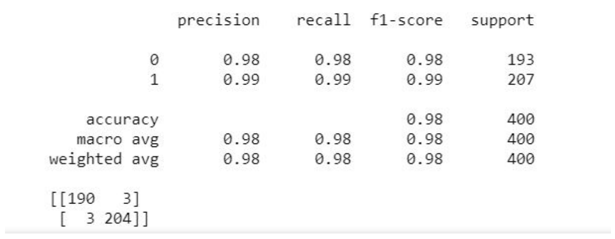
II.I The Solution

The jupyter notebook: **crack_classific.ipynb** is structured as such: The first 2 cells define the code for all the features implemented together. The first cell of the first 2 cell should be run first because its the cell that initiates training. After running the first cell, run the next cell which gives results of testing and should look something like this:



```
KNN: 0.965625 (0.016358)
RF: 0.996875 (0.004193)
NB: 0.975625 (0.007099)
SVM: 0.976875 (0.007930)
```

Figure 1: Example of accuracy output of classifiers



```
              precision    recall  f1-score   support

           0       0.98      0.98      0.98       193
           1       0.99      0.99      0.99       207

    accuracy                           0.98       400
   macro avg       0.98      0.98      0.98       400
weighted avg       0.98      0.98      0.98       400

[[190   3]
 [  3 204]]
```

Figure 2: Example of accuracy output of feature accuracy

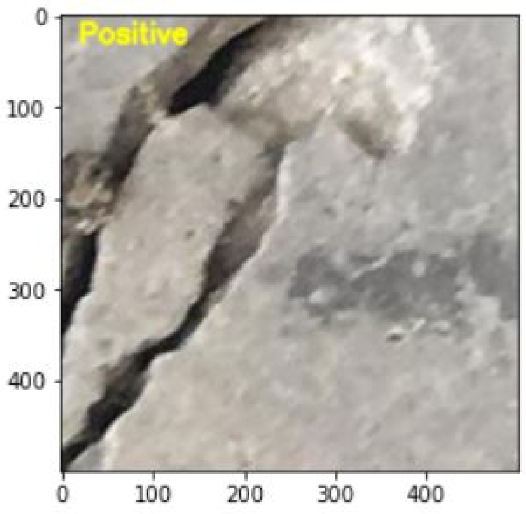


Figure 3: Example of image output after testing for positive

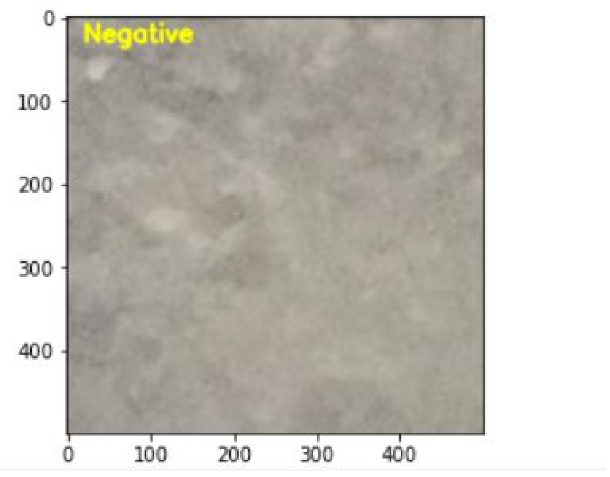


Figure 4: Example of image output after testing for negative

This is how the jupyter notebook is structured where each two cells represent for a specific research scenario and first cell of each scenario should be run first. And the folders **output, output1, output2, output3** are the HDF5 which stores the feature vector and labels. The folder: **check** contains the images we use for the testing. All the library installed was set up in an anaconda environment [2] and I mainly followed the guide - [3].

II.II Global Feature Descriptors

Shape - Hu moments

Hu moments are image moments allow for extracting properties of a image such as area, centroid and information about its rotations. we will use OpenCV which has the built-in function for calculating Hu Moments. Code Snippet 1:

```
# feature-descriptor-1: Hu Moments
def fd_hu_moments(image):
image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
feature = cv2.HuMoments(cv2.moments(image)).flatten()
return feature
```

Texture - Haralick

Haralick texture feature calculated from gray level co-occurence matrix (GLCM) is a common method to represent image texture. It is used to quantify image according to texture. Mahotas global feature - haralick was used as seen below: Code Snippet 2:

```
# feature-descriptor-2: Haralick Texture
def fd_haralick(image):
# convert the image to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
# compute the haralick texture feature vector
haralick = mahotas.features.haralick(gray).mean(axis=0)
# return the result
return haralick
```

Histogram - Color histogram

A color histogram is a representation of the distribution of colors in an image. For digital images, a color histogram represents the number of pixels that have colors in each of a fixed list of color ranges, that span the image's color space, the set of all possible colors. Code Snippet 3:

```
# feature-descriptor-3: Color Histogram
def fd_histogram(image, mask=None):
# convert the image to HSV color-space
image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
# compute the color histogram
hist  = cv2.calcHist([image], [0, 1, 2], None, [bins, bins, bins], [0,
    256, 0, 256, 0, 256])
# normalize the histogram
cv2.normalize(hist, hist)
# return the histogram
return hist.flatten()
```

Histogram - Edge oriented histogram

Edge oriented histogram is a feature extraction scheme based on the information provided by edges of different orientation within an image. This is the basis of many other relaated popular features such as Histogram of Oriented Gradients (HOG). Code Snippet 4:

```
# feature descriptor 4: Edge oriented histogram
def eog_histogram(image):
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(image,100,200)
histo  = cv2.calcHist([image], [0, 1, 2], None, [bins, bins, bins], [0,
    256, 0, 256, 0, 256])
# normalize the histogram
cv2.normalize(histo, histo)
# return the histogram
return histo.flatten()
```

Cross validation

We create a training model and get mean for all classes and compare the algorithms as such:
Code Snippet 5:

```
# 10-fold cross validation
for name, model in models:
kfold = KFold(n_splits=10, random_state=7)
cv_results = cross_val_score(model, trainDataGlobal, trainLabelsGlobal,
    cv=kfold, scoring=scoring)
results.append(cv_results)
names.append(name)
msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
print(msg)

# boxplot algorithm comparison
```

```
fig = pyplot.figure()
fig.suptitle('Machine Learning algorithm comparison')
ax = fig.add_subplot(111)
pyplot.boxplot(results)
ax.set_xticklabels(names)
pyplot.show()
```

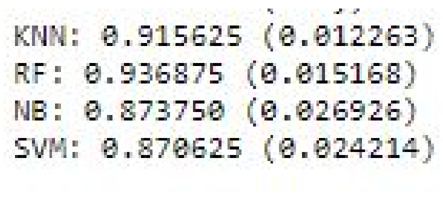
Thereafter we create testing model using RF:
Code Snippet 6:

```
# create the model - Random Forests
#clf  = RandomForestClassifier(n_estimators=100, random_state=9)
#clf =SVC()
# fit the training data to the model
clf=RandomForestClassifier(max_depth=2, random_state=0)
clf.fit(trainDataGlobal, trainLabelsGlobal)

# path to test data - Professor please put the url to where the datasets
    are in your computer when
# running the code
test_path = "C:/Users/Hasib/Desktop/crack_classification/check"

# loop through the test images
for file in glob.glob(test_path + "/*.jpg"):
# read the image
image = cv2.imread(file)

# resize the image
image = cv2.resize(image, fixed_size)

####################################
# Global Feature extraction
####################################
fv_hu_moments = fd_hu_moments(image)
fv_haralick   = fd_haralick(image)
fv_histogram  = fd_histogram(image)
fv_eoghistogram = eog_histogram(image)
###################################
# Concatenate global features
###################################
start_time = time.time()
global_feature = np.hstack([fv_histogram, fv_haralick,
    fv_hu_moments,fv_eoghistogram])

# predict label of test image
prediction = clf.predict(global_feature.reshape(1,-1))[0]
print("--- %s seconds ---" % (time.time() - start_time))
if prediction==0:
    blurred = cv2.GaussianBlur(image, (11, 11), 0)
    #conver from bgr to hsv
    hsv=cv2.cvtColor(blurred,cv2.COLOR_BGR2HSV)
    #creating mask
    lower_green=np.array([36, 10, 50])
    upper_green=np.array([100, 180, 190])

    red=cv2.inRange(hsv,lower_green,upper_green)
    #apply morpological featurs
    kernal = np.ones((5 ,5), "uint8")
    red=cv2.dilate(red,None, iterations=2)
    red = cv2.erode(red, None, iterations=2)
    res=cv2.bitwise_and(image,image,mask=red)
    #draw rectangle
    #(contours,hierarchy)=cv2.findContours(red,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)


if prediction==1:
    blurred = cv2.GaussianBlur(image, (11, 11), 0)
    #conver from bgr to hsv
    hsv=cv2.cvtColor(blurred,cv2.COLOR_BGR2HSV)
    #creating mask
    lower_green=np.array([36, 10, 50])
    upper_green=np.array([100, 180, 190])

    red=cv2.inRange(hsv,lower_green,upper_green)

    #apply morpological featurs
    kernal = np.ones((5 ,5), "uint8")
    red=cv2.dilate(red,None, iterations=2)
    red = cv2.erode(red, None, iterations=2)
    res=cv2.bitwise_and(image,image,mask=red)
    #draw rectangle
    #(contours,hierarchy)=cv2.findContours(red,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)



# show predicted label on image
cv2.putText(image, train_labels[prediction], (20,30),
   cv2.FONT_HERSHEY_SIMPLEX, 1.0, (0,255,255), 3)

# display the output image
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.show()

y_pred= clf.predict(testDataGlobal)
print(classification_report(testLabelsGlobal,y_pred) )
mat=confusion_matrix(testLabelsGlobal, y_pred)
print(mat)
```

III Results

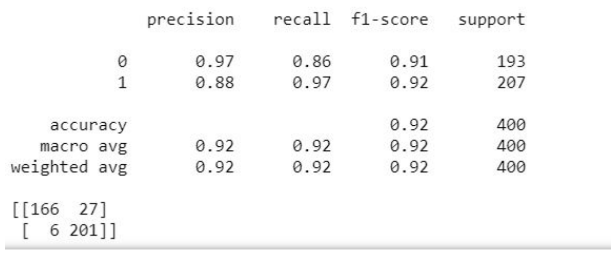
After running the code for each individual feature and combinatiion of different groups of features together, the following results were achieved for **Accuracy**:

Hu Moments (Shape):



```
KNN: 0.915625 (0.012263)
RF: 0.936875 (0.015168)
NB: 0.873750 (0.026926)
SVM: 0.870625 (0.024214)
```

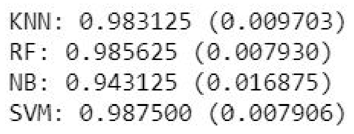
Figure 5: Classifier accuracy



```
              precision    recall  f1-score   support

           0       0.97      0.86      0.91       193
           1       0.88      0.97      0.92       207

    accuracy                           0.92       400
   macro avg       0.92      0.92      0.92       400
weighted avg       0.92      0.92      0.92       400

[[166  27]
 [  6 201]]
```

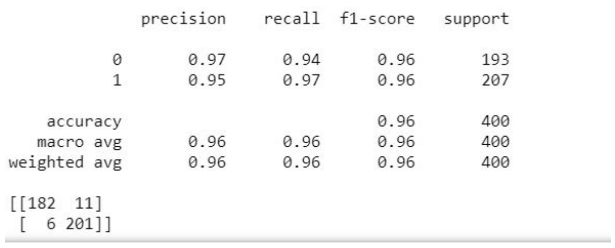
Figure 6: Hu Accuracy and Confusion matrix

Haralick (texture):



```
KNN: 0.983125 (0.009703)
RF: 0.985625 (0.007930)
NB: 0.943125 (0.016875)
SVM: 0.987500 (0.007906)
```

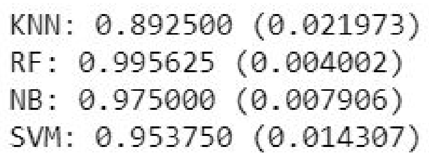
Figure 7: Classifier accuracy



```
              precision    recall  f1-score   support

           0       0.97      0.94      0.96       193
           1       0.95      0.97      0.96       207

    accuracy                           0.96       400
   macro avg       0.96      0.96      0.96       400
weighted avg       0.96      0.96      0.96       400

[[182  11]
 [  6 201]]
```

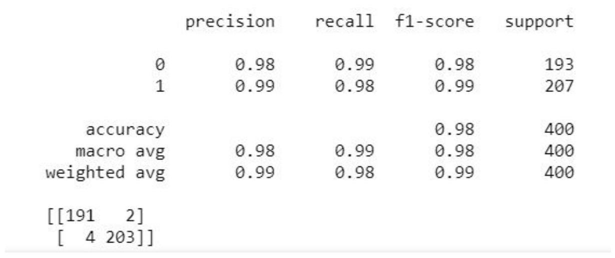
Figure 8: Haralick Accuracy and Confusion matrix

Color and Edge oriented Histogram:



```
KNN: 0.892500 (0.021973)
RF: 0.995625 (0.004002)
NB: 0.975000 (0.007906)
SVM: 0.953750 (0.014307)
```

Figure 9: Classifier accuracy



```
              precision    recall  f1-score   support

           0       0.98      0.99      0.98       193
           1       0.99      0.98      0.99       207

    accuracy                           0.98       400
   macro avg       0.98      0.99      0.98       400
weighted avg       0.99      0.98      0.99       400

[[191   2]
 [  4 203]]
```

Figure 10: Histogram Accuracy and Confusion matrix

Note: The above values are not displayed in percentage. So, for example 0.92 of Hu moment actually if represented in percentage it is 92%.

For combination of different features in groups (Accuracy):
Haralick + Hu moments + color histogram + Edge oriented histogram = 98%
Hu moments + Haralick = 48%
Haralic + color histogram + Edge oriented histogram = 99%
Hu moments + Color histogram = 98%
Edge oriented histogram + Haralick = 99%

IV Conclusion and Analysis

As seen from the output, Random Forest Classifier achieved the highest accuracy amongst the tested classifiers. Amongst features, when Edge oriented histogram and Haralick were grouped together we got the highest accuracy of 99%. This makes sense since we are evaluating cracks so texture and edge would provide more precise analysis and give high accuracy.

When comparing the different features whether individual or as a group, it can be observed that, Histogram had the most impact on the accuracy due to its high individual result of 98%. Consequently, groups with Histogram always provided high accuracy compared to the low 48% accuracy of Hu moments + Haralick. It can safely be said that Histogram should be used for testing such crack involving images i.e. better not to use shape features. Final establishment is Random Forest Classifier should be utilized as classifier due to its high accuracy.

V Source code

```
#The following code was written by M Hasibur Rahman, 295459
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
import numpy as np
import mahotas
import cv2
import os
import h5py
from PIL import Image
import glob

# fixed-sizes for image
fixed_size = tuple((227, 227))

# path to training data - Professor please put the url to where the
    datasets are in your computer when # running the code
train_path = "C:/Users/Hasib/Desktop/crack_classification/dataset_1/train"

# no.of.trees for Random Forests
num_trees = 100

# bins for histogram
bins = 8

# train_test_split size
test_size = 0.20

# seed for reproducing same results
seed = 9

# feature-descriptor-1: Hu Moments
def fd_hu_moments(image):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    feature = cv2.HuMoments(cv2.moments(image)).flatten()
    return feature

# feature-descriptor-2: Haralick Texture
def fd_haralick(image):
    # convert the image to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # compute the haralick texture feature vector
    haralick = mahotas.features.haralick(gray).mean(axis=0)
    # return the result
    return haralick

# feature-descriptor-3: Color Histogram
def fd_histogram(image, mask=None):
    # convert the image to HSV color-space
    image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    # compute the color histogram
    hist  = cv2.calcHist([image], [0, 1, 2], None, [bins, bins, bins], [0,
        256, 0, 256, 0, 256])
    # normalize the histogram
    cv2.normalize(hist, hist)
    # return the histogram
    return hist.flatten()
# feature descriptor 4: Edge oriented histogram
def eog_histogram(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    edges = cv2.Canny(image,100,200)
    histo  = cv2.calcHist([image], [0, 1, 2], None, [bins, bins, bins],
       [0, 256, 0, 256, 0, 256])
    # normalize the histogram
    cv2.normalize(histo, histo)
    # return the histogram
    return histo.flatten()
# get the training labels
train_labels = os.listdir(train_path)

# sort the training labels
train_labels.sort()
print(train_labels)

# empty lists to hold feature vectors and labels
global_features = []
labels = []

i, j = 0, 0
k = 0

# num of images per class change accorrding to your images
images_per_class =1000

# loop over the training data sub-folders
for training_name in train_labels:
# join the training data path and each species training folder
dir = os.path.join(train_path, training_name)

# get the current training label
current_label = training_name

k = 1
# loop over the images in each sub-folder
for x in range(1,images_per_class+1):
    # get the image file name
    file = dir + "/" + str(x) + ".jpg"

    # read the image and resize it to a fixed-size
    image = cv2.imread(file)

    image = cv2.resize(image, fixed_size)

    ####################################
    # Global Feature extraction
    ####################################
    fv_hu_moments = fd_hu_moments(image)
    fv_haralick   = fd_haralick(image)
    fv_histogram  = fd_histogram(image)
    fv_eoghistogram = eog_histogram(image)
    ###################################
    # Concatenate global features
    ###################################
```

```
        global_feature = np.hstack([fv_histogram, fv_haralick, fv_hu_moments,
            fv_eoghistogram])

        # update the list of labels and feature vectors
        labels.append(current_label)
        global_features.append(global_feature)

        i += 1
        k += 1
    print ("[STATUS] processed folder: {}".format(current_label))
    j += 1

print ("[STATUS] completed Global Feature Extraction...")

print ("[STATUS] feature vector size
    {}".format(np.array(global_features).shape))

# get the overall training label size
print ("[STATUS] training Labels {}".format(np.array(labels).shape))

# encode the target labels
targetNames = np.unique(labels)
le = LabelEncoder()
target = le.fit_transform(labels)
print ("[STATUS] training labels encoded...")

# normalize the feature vector in the range (0-1)
scaler = MinMaxScaler(feature_range=(0, 1))
rescaled_features = scaler.fit_transform(global_features)
print ("[STATUS] feature vector normalized...")

print ("[STATUS] target labels: {}".format(target))
print ("[STATUS] target labels shape: {}".format(target.shape))

# save the feature vector using HDF5
h5f_data = h5py.File('output/data.h5', 'w')
h5f_data.create_dataset('dataset_1', data=np.array(rescaled_features))

h5f_label = h5py.File('output/labels.h5', 'w')
h5f_label.create_dataset('dataset_1', data=np.array(target))

h5f_data.close()
h5f_label.close()

print ("[STATUS] end of training..")
```

```
import h5py
import time
import numpy as np
import os
import glob
import cv2
from matplotlib import pyplot
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.model_selection import KFold, StratifiedKFold
from sklearn.metrics import confusion_matrix, accuracy_score,
    classification_report
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix
from sklearn.svm import SVC
from sklearn import svm
import mahotas
import sklearn.externals
import joblib
num_trees = 1000
fixed_size = tuple((500, 500))
# bins for histogram
bins = 8

# train_test_split size - Professor please put the url to where the
    datasets are in your computer when # running the code
test_size = 0.20
train_path = "C:/Users/Hasib/Desktop/crack_classification/dataset_1/train"

# seed for reproducing same results
seed = 9

# feature-descriptor-1: Hu Moments
def fd_hu_moments(image):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    feature = cv2.HuMoments(cv2.moments(image)).flatten()
    return feature

# feature-descriptor-2: Haralick Texture
def fd_haralick(image):
    # convert the image to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # compute the haralick texture feature vector
    haralick = mahotas.features.haralick(gray).mean(axis=0)
    # return the result
    return haralick

# feature-descriptor-3: Color Histogram
def fd_histogram(image, mask=None):
    # convert the image to HSV color-space
    image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    # compute the color histogram
    hist  = cv2.calcHist([image], [0, 1, 2], None, [bins, bins, bins], [0,
        256, 0, 256, 0, 256])
    # normalize the histogram
    cv2.normalize(hist, hist)
    # return the histogram
    return hist.flatten()
def eog_histogram(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    edges = cv2.Canny(image,100,200)
    histo  = cv2.calcHist([image], [0, 1, 2], None, [bins, bins, bins],
        [0, 256, 0, 256, 0, 256])
    # normalize the histogram
    cv2.normalize(histo, histo)
    # return the histogram
    return histo.flatten()
train_labels = os.listdir(train_path)

# sort the training labels
train_labels.sort()
print(train_labels)
# create all the machine learning models
models = []
#models.append(('LR', LogisticRegression(random_state=9)))
#models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
#models.append(('CART', DecisionTreeClassifier(random_state=9)))
models.append(('RF', RandomForestClassifier(n_estimators=num_trees,
   random_state=9)))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC(random_state=0)))

# variables to hold the results and names
results = []
names = []
scoring = "accuracy"

# import the feature vector and trained labels
h5f_data = h5py.File('output/data.h5', 'r')
h5f_label = h5py.File('output/labels.h5', 'r')

global_features_string = h5f_data['dataset_1']
global_labels_string = h5f_label['dataset_1']

global_features = np.array(global_features_string)
global_labels = np.array(global_labels_string)

h5f_data.close()
h5f_label.close()


# verify the shape of the feature vector and labels
print ("[STATUS] features shape: {}".format(global_features.shape))
print ("[STATUS] labels shape: {}".format(global_labels.shape))

print ("[STATUS] training started...")

# split the training and testing data
(trainDataGlobal, testDataGlobal, trainLabelsGlobal, testLabelsGlobal) =
   train_test_split(np.array(global_features),




print ("[STATUS] splitted train and test data...")
print ("Train data  : {}".format(trainDataGlobal.shape))
print ("Test data   : {}".format(testDataGlobal.shape))
print ("Train labels: {}".format(trainLabelsGlobal.shape))
print ("Test labels : {}".format(testLabelsGlobal.shape))

# filter all the warnings
import warnings
```

```
warnings.filterwarnings('ignore')

# 10-fold cross validation
for name, model in models:
    kfold = KFold(n_splits=10, random_state=7)
    cv_results = cross_val_score(model, trainDataGlobal,
        trainLabelsGlobal, cv=kfold, scoring=scoring)
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)

# boxplot algorithm comparison
fig = pyplot.figure()
fig.suptitle('Machine Learning algorithm comparison')
ax = fig.add_subplot(111)
pyplot.boxplot(results)
ax.set_xticklabels(names)
pyplot.show()

#-----------------------------------
# TESTING OUR MODEL
#-----------------------------------

# to visualize results
import matplotlib.pyplot as plt

# create the model - Random Forests
#clf  = RandomForestClassifier(n_estimators=100, random_state=9)
#clf =SVC()
# fit the training data to the model
clf=RandomForestClassifier(max_depth=2, random_state=0)
clf.fit(trainDataGlobal, trainLabelsGlobal)

# path to test data - Professor please put the url to where the test
   datasets are in your computer
# when running the code
test_path = "C:/Users/Hasib/Desktop/crack_classification/check"

# loop through the test images
for file in glob.glob(test_path + "/*.jpg"):
# read the image
image = cv2.imread(file)

# resize the image
image = cv2.resize(image, fixed_size)

####################################
# Global Feature extraction
####################################
fv_hu_moments = fd_hu_moments(image)
fv_haralick   = fd_haralick(image)
fv_histogram  = fd_histogram(image)
fv_eoghistogram = eog_histogram(image)
###################################
# Concatenate global features
###################################
start_time = time.time()
```

```
global_feature = np.hstack([fv_histogram, fv_haralick,
    fv_hu_moments,fv_eoghistogram])

# predict label of test image
prediction = clf.predict(global_feature.reshape(1,-1))[0]
print("--- %s seconds ---" % (time.time() - start_time))
if prediction==0:
    blurred = cv2.GaussianBlur(image, (11, 11), 0)
    #conver from bgr to hsv
    hsv=cv2.cvtColor(blurred,cv2.COLOR_BGR2HSV)
    #creating mask
    lower_green=np.array([36, 10, 50])
    upper_green=np.array([100, 180, 190])

    red=cv2.inRange(hsv,lower_green,upper_green)

    #apply morpological featurs
    kernal = np.ones((5 ,5), "uint8")
    red=cv2.dilate(red,None, iterations=2)
    red = cv2.erode(red, None, iterations=2)
    res=cv2.bitwise_and(image,image,mask=red)
    #draw rectangle
    #(contours,hierarchy)=cv2.findContours(red,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)


if prediction==1:
    blurred = cv2.GaussianBlur(image, (11, 11), 0)
    #conver from bgr to hsv
    hsv=cv2.cvtColor(blurred,cv2.COLOR_BGR2HSV)
    #creating mask
    lower_green=np.array([36, 10, 50])
    upper_green=np.array([100, 180, 190])

    red=cv2.inRange(hsv,lower_green,upper_green)

    #apply morpological featurs
    kernal = np.ones((5 ,5), "uint8")
    red=cv2.dilate(red,None, iterations=2)
    red = cv2.erode(red, None, iterations=2)
    res=cv2.bitwise_and(image,image,mask=red)
    #draw rectangle
    #(contours,hierarchy)=cv2.findContours(red,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)



# show predicted label on image
cv2.putText(image, train_labels[prediction], (20,30),
    cv2.FONT_HERSHEY_SIMPLEX, 1.0, (0,255,255), 3)

# display the output image
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.show()

ed= clf.predict(testDataGlobal)
t(classification_report(testLabelsGlobal,y_pred) )
confusion_matrix(testLabelsGlobal, y_pred)
t(mat)
```

Note: The rest of the code is in the jupyter notebook (crack_classific.ipynb). The above is for the code for all features and classifiers together. Individual and other combinations can be found in the aforementioned jupyeter notebook (crack_classific.ipynb). Mostly the combinations were done by commenting out the features we dont want to use currently e.g. when testing Hu moment and harlaick together all I simply did was comment out the other feature. This is illustrated below:

```
#So below you can see I commented out fv_histogram and fv_eoghistogram as
   I only want to test for Hu
# and Haralick

# Global Feature extraction

fv_hu_moments = fd_hu_moments(image)
fv_haralick   = fd_haralick(image)
#fv_histogram  = fd_histogram(image)
#fv_eoghistogram = eog_histogram(image)

# Concatenate global features
#Also make sure to add only the feature you are testing in concatenation

start_time = time.time()
global_feature = np.hstack([fv_haralick, fv_hu_moments])
```

Bibliography

[1] *h5py*. URL: https://www.h5py.org/ (page 2).

[2] *Setup Anaconda*. 2020. URL: https://machinelearningmastery.com/setup-python-environment-machine-learning-deep-learning-anaconda/ (page 4).

[3] *Machine Learning Project - how to start*. 2020. URL: https://machinelearningmastery.com/machine-learning-in-python-step-by-step/ (page 4).