Texture Recognition using Haralick Texture and Python

Computer Vision | 15 December 2016

12 Comments

Contents

When it comes to Global Feature Descriptors (i.e feature vectors that quan
image), there are three major attributes to be considered – Color, Shape a
these three could be used separately or combined to quantify images. In th
learn how to recognize texture in images. We will study a new type of globa
descriptor called Haralick Texture. Let's jump right into it!


```
Objectives


1  How to label and organize our own dataset?
2  What is Haralick Texture and how it is computed?
3  How to use Haralick Texture module in mahotas library?
4  How to recognize textures in images?
5  How to use Linear SVM to train and classify images?
6  How to make predictions using the created model on an unseen data?
```

What is a Texture?

Texture defines the consistency of patterns and colors in an object/image such as bricks, school uniforms, sand, rocks, grass etc. To classify objects in an image based on texture, we

have to look for the consistent spread of patterns and colors in the object's surface. Rough-Smooth, Hard-Soft, Fine-Coarse are some of the texture pairs one could think of, although there are many such pairs.

What is Haralick Texture?

Haralick Texture is used to quantify an image based on texture. It was inve
1973 and you can read about it in detail here. The fundamental concept inv
computing Haralick Texture features is the Gray Level Co-occurrence Matr

What is GLCM?

Gray Level Co-occurrence matrix (GLCM) uses adjacency concept in image
is that it looks for pairs of adjacent pixel values that occur in an image and
over the entire image. Below figure explains how a GLCM is constructed.

Contents

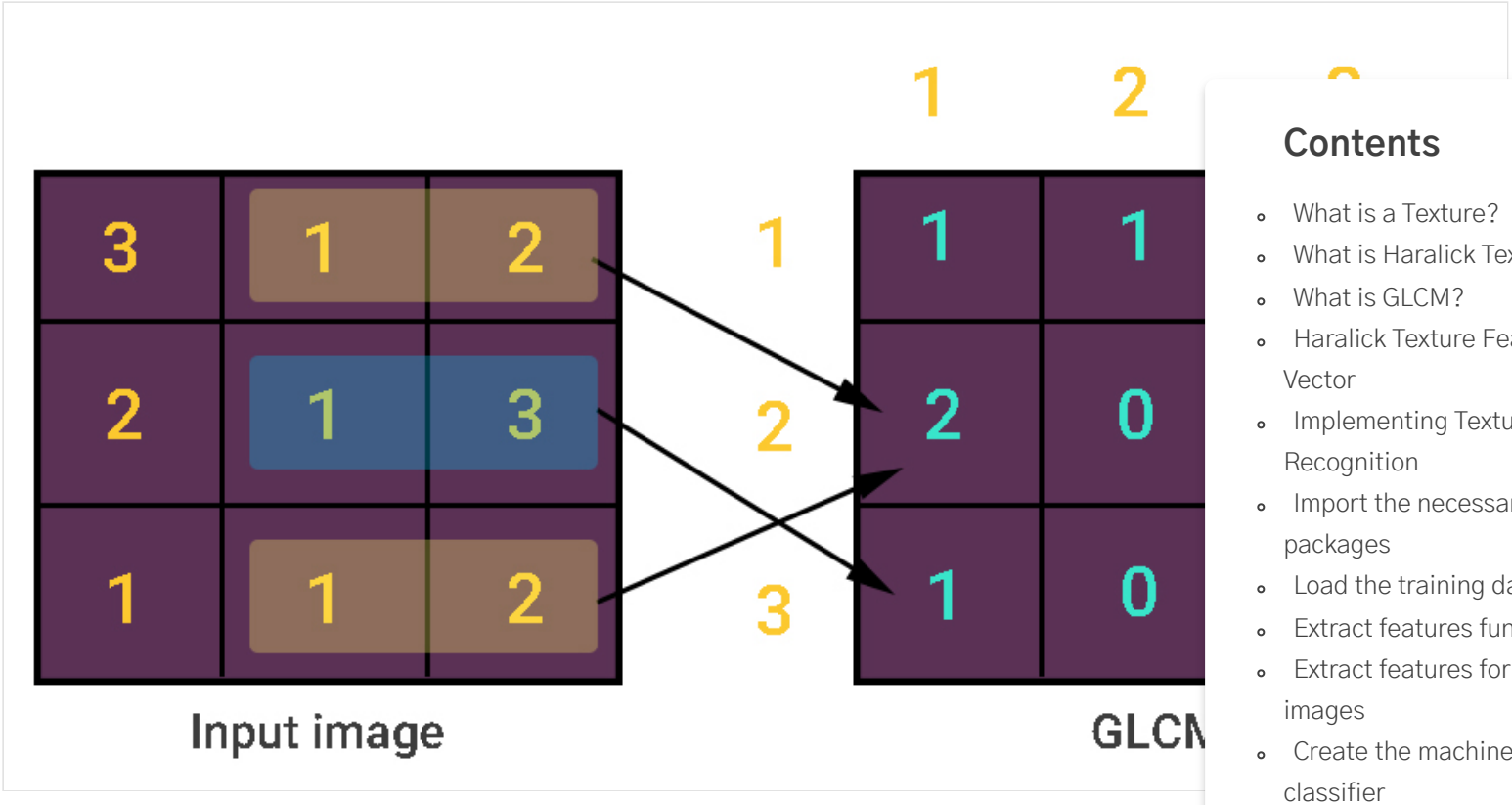


Figure 1. Gray Level Co-occurance Matrix (GLCM)

As you can see from the above image, gray-level pixel value 1 and 2 occurs twice in the image and hence GLCM records it as two. But pixel value 1 and 3 occurs only once in the image and thus GLCM records it as one. Of course, I have assumed the adjacency calculation only from left-to-right. Actually, there are four types of adjacency and hence four GLCM matrices are constructed for a single image. Four types of adjacency are as follows.

- Left-to-Right

- Top-to-Bottom
- Top Left-to-Bottom Right
- Top Right-to-Bottom Left

Haralick Texture Feature Vector

From the four GLCM matrices, 14 textural features are computed that are b
statistical theory. All these 14 statistical features needs a separate blog po
read in detail about those here. Normally, the feature vector is taken to be
computing 14th dim might increase the computational time.

Implementing Texture Recognition

Ok, lets start with the code!

Actually, it will take just 10-15 minutes to complete our texture recognition system using OpenCV, Python, sklearn and mahotas provided we have the training dataset.

Note: In case if you don't have these packages installed, feel free to install these using my environment setup posts given below.

- Deep Learning Environment Setup (Windows)
- Deep Learning Environment Setup (Linux)

Contents x

Import the necessary packages

train_test.py

```
1 import cv2
2 import numpy as np
3 import os
4 import glob
5 import mahotas as mt
6 from sklearn.svm import LinearSVC
```

Load the training dataset

train_test.py

```
1 # load the training dataset
2 train_path = "dataset/train"
3 train_names = os.listdir(train_path)
4
5 # empty list to hold feature vectors and train labels
6 train_features = []
7 train_labels = []
```

- Line 2 is the path to training dataset.
- Line 3 gets the class names of the training data.
- Line 6–7 are empty lists to hold feature vectors and labels.

Contents

Extract features function

train_test.py

```
1  def extract_features(image):
2          # calculate haralick texture features for 4 types of adjacency
3          textures = mt.features.haralick(image)
4
5          # take the mean of it and return it
6          ht_mean = textures.mean(axis=0)
7          return ht_mean
```

- Line 1 is a function that takes an input image to compute haralick texture.
- Line 3 extracts the haralick features for all 4 types of adjacency.
- Line 6 finds the mean of all 4 types of GLCM.
- Line 7 returns the resulting feature vector for that image which describes th

Contents

Extract features for all images

train_test.py

code

```
1
2  # loop over the training dataset
3  print "[STATUS] Started extracting haralick textures.."
4  for train_name in train_names:
5          cur_path = train_path + "/" + train_name
6          cur_label = train_name
7          i = 1
```

```
8           for file in glob.glob(cur_path + "/*.jpg"):
9                   print "Processing Image - {} in {}".format(i, cur_label)
10                  # read the training image
11                  image = cv2.imread(file)
12
13                  # convert the image to grayscale
14                  gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
15
16                  # extract haralick texture from the image
17                  features = extract_features(gray)
18
19                  # append the feature vector and label
20                  train_features.append(features)
21                  train_labels.append(cur_label)
22
23                  # show loop update
                    i += 1
```


- Line 4 loops over the training labels we have just included from training dire
- Line 5 is the path to current image class directory.
- Line 6 holds the current image class label.
- Line 8 takes all the files with .jpg as the extension and loops through each file one by one.
- Line 11 reads the input image that corresponds to a file.
- Line 14 converts the image to grayscale.
- Line 17 extracts haralick features for the grayscale image.
- Line 20 appends the 13-dim feature vector to the training features list.
- Line 21 appends the class label to training classes list.

train_test.py code

```
# have a look at the size of our feature vector and labels
print "Training features: {}".format(np.array(train_features).shape)
print "Training labels: {}".format(np.array(train_labels).shape)
```

Create the machine learning classifier

train_test.py

```
# create the classifier
print "[STATUS] Creating the classifier.."
clf_svm = LinearSVC(random_state=9)

# fit the training data and labels
print "[STATUS] Fitting data/label to model.."
clf_svm.fit(train_features, train_labels)
```

- Line 3 creates the Linear Support Vector Machine classifier.
- Line 7 fits the training features and labels to the classifier.

Test the classifier on testing data

train_test.py code

```
# loop over the test images
test_path = "dataset/test"

for file in glob.glob(test_path + "/*.jpg"):
```

Contents

```
for file in glob.glob(test_path + "/*.jpg"):
        # read the input image
        image = cv2.imread(file)

        # convert to grayscale
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

        # extract haralick texture from the image
        features = extract_features(gray)

        # evaluate the model and predict label
        prediction = clf_svm.predict(features.reshape(1, -1))[0]

        # show the label
        cv2.putText(image, prediction, (20,30), cv2.FONT_HERSHEY_SIMPLEX

      # display the output image
      cv2.imshow("Test_Image", image)
      cv2.waitKey(0)
```


- Line 2 gets the testing data path.
- Line 3 takes all the files with the .jpg extension and loops through each file one by one.
- Line 5 reads the input image.
- Line 8 converts the input image into grayscale image.
- Line 11 extract haralick features from grayscale image.
- Line 14 predicts the output label for the test image.
- Line 17 displays the output class label for the test image.
- Finally, Line 20 displays the test image with predicted label.

Training images

These are the images from which we train our machine learning classifier to learn texture features. You can collect the images of your choice and include it under a label. For example, "Grass" images are collected and stored inside a folder named "grass". Th
either be taken from a simple google search (easy to do; but our model wo
well) or from your own camera/smart-phone (which is indeed time-consum
model could generalize well due to real-world images).

As a demonstration, I have included my own training and testing images. I
training images which holds 3 images per class. Training images with their
class/label are shown below.

Contents

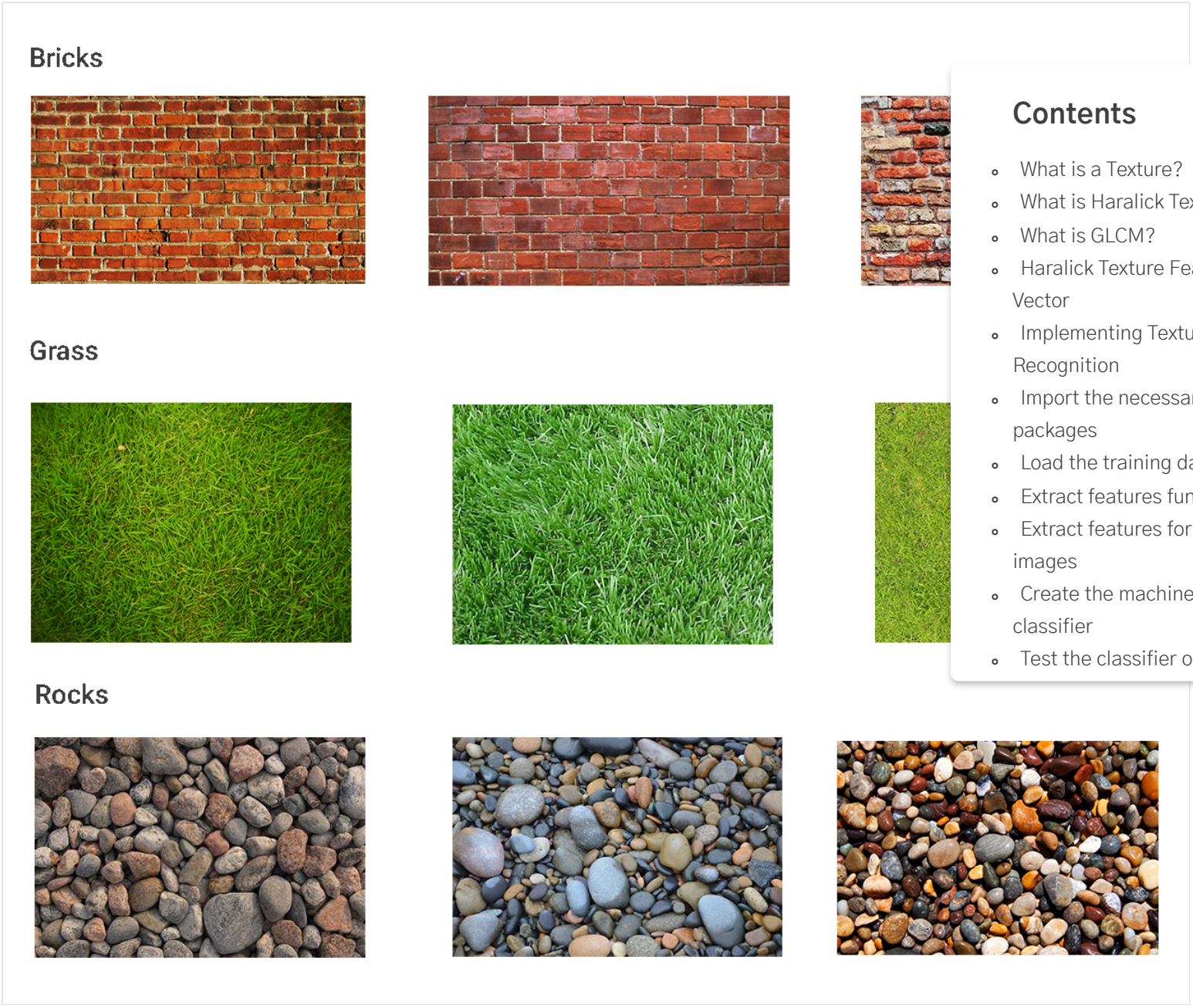


Figure 2. Training Images

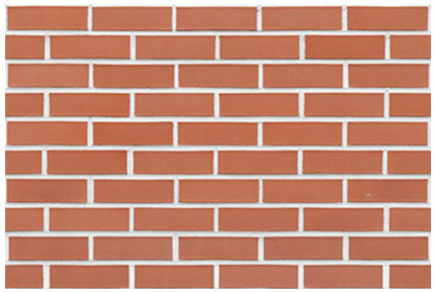
Testing images

These could be images or a video sequence from a smartphone/camera. W
model with this test data so that our model performs feature extraction on
tries to come up with the best possible label/class.

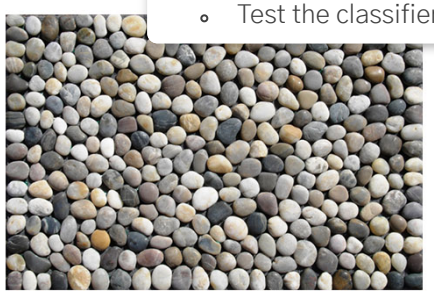
Some of the test images for which we need to predict the class/label are sh

Note: These test images won't have any label associated with them. Ou
purpose is to predict the best possible label/class for the image it sees

Test image - 1



Test image - 2



Test image - 3

Figure 3. Testing Images

After running the code, our model was able to correctly predict the labels for the testing data as shown below.

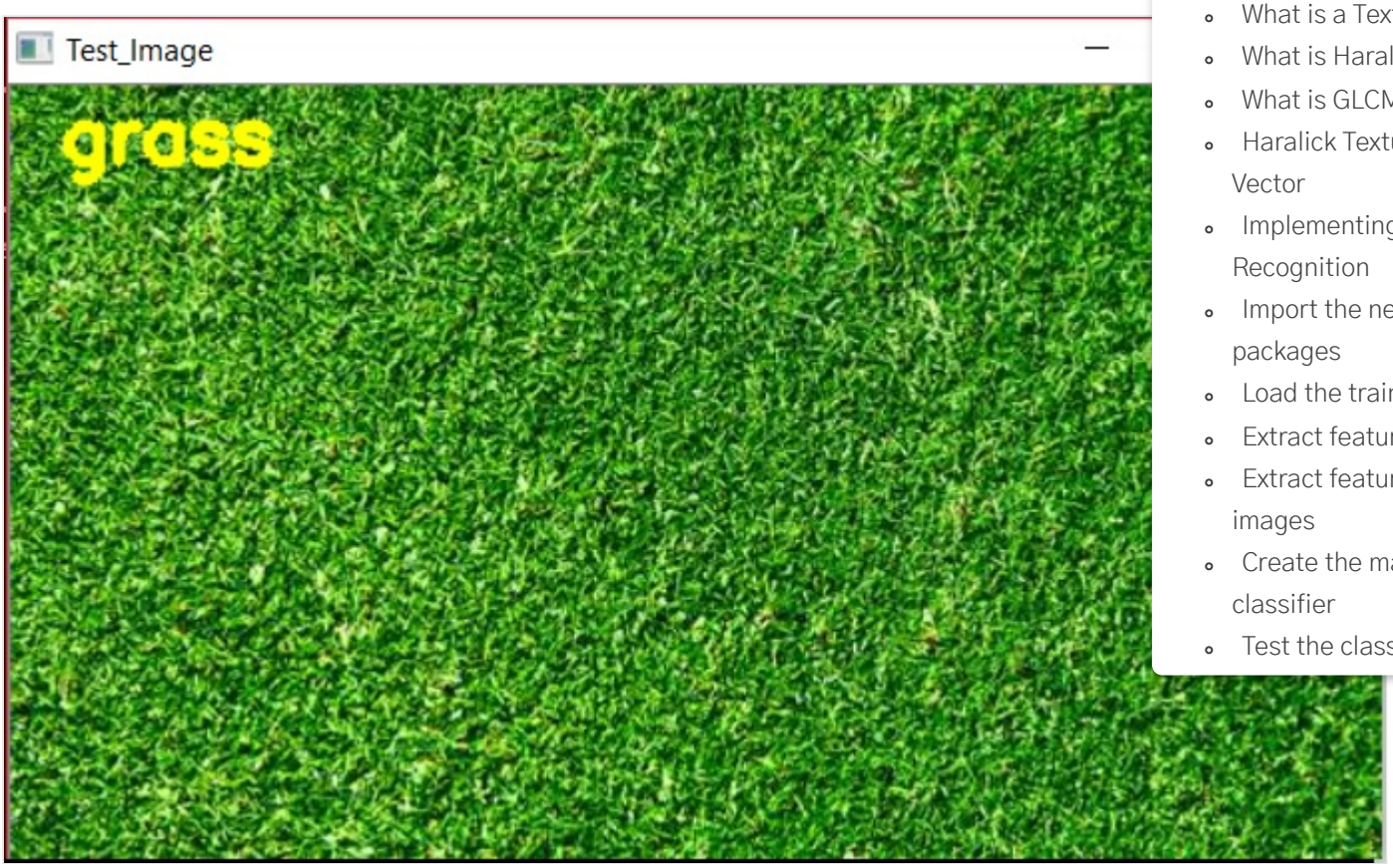


Figure 4. Test Image Prediction – 1

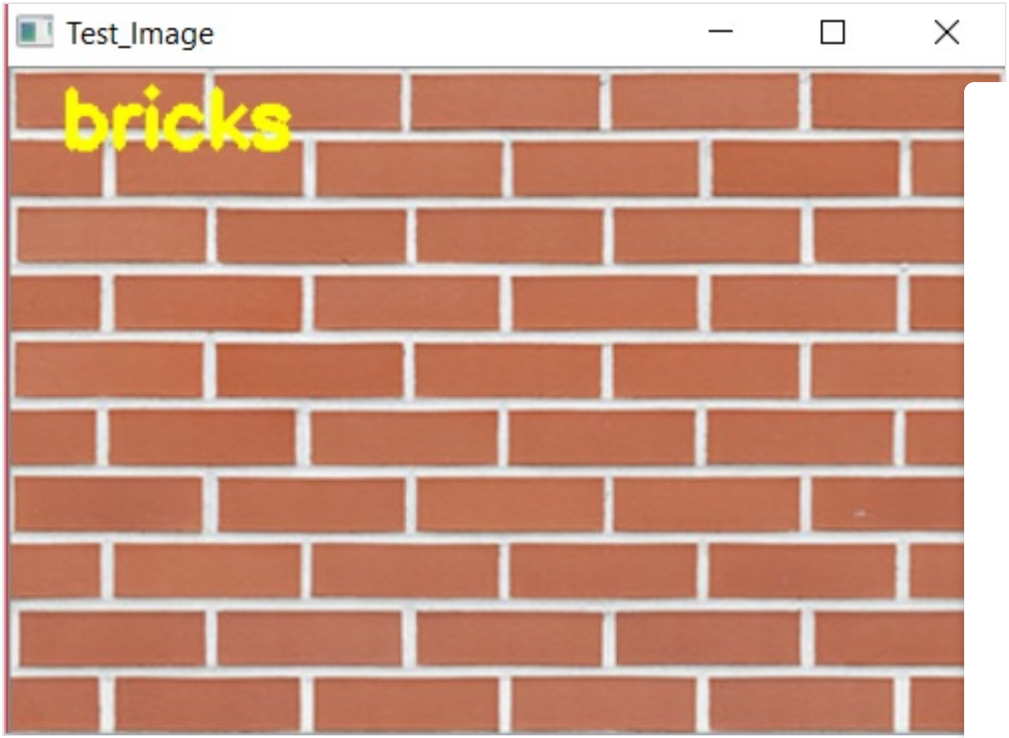


Figure 5. Test Image Prediction – 2

Contents

Figure 6. Test Image Prediction – 3

Here is the entire code to build our texture recognition system.

train_test.py code

```
import cv2
import numpy as np
import os
import glob
import mahotas as mt
from sklearn.svm import LinearSVC

# function to extract haralick textures from an image
def extract_features(image):
  # calculate haralick texture features for 4 types of adjacency
  textures = mt.features.haralick(image)

  # take the mean of it and return it
  ht_mean  = textures.mean(axis=0)
  return ht_mean

# load the training dataset
train_path  = "dataset/train"
train_names = os.listdir(train_path)

# empty list to hold feature vectors and train labels
train_features = []
train_labels   = []

# loop over the training dataset
print "[STATUS] Started extracting haralick textures.."
for train_name in train_names:
  cur_path = train_path + "/" + train_name
  cur_label = train_name
  i = 1

  for file in glob.glob(cur_path + "/*.jpg"):
    print "Processing Image - {} in {}".format(i, cur_label)
    # read the training image
    image = cv2.imread(file)

    # convert the image to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

Contents

```
40
41      # extract haralick texture from the image
42      features = extract_features(gray)
43
44      # append the feature vector and label
45      train_features.append(features)
46      train_labels.append(cur_label)
47
48      # show loop update
49      i += 1
50
51  # have a look at the size of our feature vector and labels
52  print "Training features: {}".format(np.array(train_features).shape)
53  print "Training labels: {}".format(np.array(train_labels).shape)
54
55  # create the classifier
56  print "[STATUS] Creating the classifier.."
57  clf_svm = LinearSVC(random_state=9)
58
59  # fit the training data and labels
60  print "[STATUS] Fitting data/label to model.."
61  clf_svm.fit(train_features, train_labels)
62
63  # loop over the test images
64  test_path = "dataset/test"
65  for file in glob.glob(test_path + "/*.jpg"):
66    # read the input image
67    image = cv2.imread(file)
68
69    # convert to grayscale
70    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
71
72    # extract haralick texture from the image
73    features = extract_features(gray)
74
75    # evaluate the model and predict label
76    prediction = clf_svm.predict(features.reshape(1, -1))[0]
77
78    # show the label
79    cv2.putText(image, prediction, (20,30), cv2.FONT_HERSHEY_SIMPLEX, 1.0, (0,255,255), 3)
```

Contents

```
80    print "Prediction - {}".format(prediction)
81
82    # display the output image
83    cv2.imshow("Test_Image", image)
      cv2.waitKey(0)
```

```
[STATUS] Started extracting haralick textures..
Processing Image - 1 in bricks
Processing Image - 2 in bricks
Processing Image - 3 in bricks
Processing Image - 1 in grass
Processing Image - 2 in grass
Processing Image - 3 in grass
Processing Image - 1 in rocks
Processing Image - 2 in rocks
Processing Image - 3 in rocks
Training features: (9, 13)
Training labels: (9,)
[STATUS] Creating the classifier..
[STATUS] Fitting data/label to model..
Prediction - grass
Prediction - bricks
Prediction - rocks
```


If you *copy-paste* the above code in any of your directory and run `python train_test.py`, you will get the following results.

Thus, we have implemented our very own Texture Recognition system using Haralick Textures, Python and OpenCV.

In case if you found something useful to add to this article or you found
code or would like to improve some points mentioned, feel free to write
comments. Hope you found something useful here.

Contents

12 Comments

ALSO ON **GOGUL'S PERSONAL WEBSITE**

How to install SASS in Linux?

3 years ago • 2 comments

Learn how to install SASS in linux so that you could use it to write neat and clean CSS.

Python for Hardware Design

3 years ago • 1 comment

How python programming language could be used in hardware design such as …

Image Classification using Python and …

4 years ago • 90 comments

Learn how to use Global Feature Descriptors such as RGB Color Histograms, …

3 years ago • 7 comments

Understand how to approach a machine learning regression …