Introduction to Image Processing and Computer Vision

Laboratory Project 2:

Image classification

Contents:

Bartłomiej Żyła   27.01.2020

1. Introduction

Image classification, a topic of pattern recognition in computer vision, is a process of classification based on various information in images. In this example various methods were used, starting from shape and texture recognition, ending with color histogram and local binary moments. The objective of the task was to create a machine learning model which for given descriptors would be able to classify leaves by their species.

circinatum

garryana

1.1 Description of algorythm

Algorythm is divided into 3 main parts – simple segmentation, computing descriptors, and training the model. The main idea was to choose such descriptors and model that would give the highest possible prediction score with the limited amount of data given.

a) Feature descriptors

In total 5 distinct feature descriptors were used. Four of them are global features due to the simplicity of fitting them into the model.

1) Hu Moments
Image moments capture information about the shape of a blob in a binary image, because they contain information about the intensity $I(x,y)$, as well as position x and y of the pixels. Hu Moments (or rather Hu moment invariants) is a set of 7 numbers calculated using central moments which do not depend on image transformations. The first 6 moments have been proved to be invariant to translation, scale, rotation and reflection, while the 7th moment's sign changes according to the image reflection.[1]

id	Image	H[0]	H[1]	H[2]	H[3]	H[4]	H[5]	H[6]
K0	K	2.78871	6.50638	9.44249	9.84018	-19.593	-13.1205	19.6797
S0	S	2.67431	5.77446	9.90311	11.0016	-21.4722	-14.1102	22.0012
S1	S	2.67431	5.77446	9.90311	11.0016	-21.4722	-14.1102	22.0012
S2	S	2.65884	5.7358	9.66822	10.7427	-20.9914	-13.8694	21.3202
S3	S	2.66083	5.745	9.80616	10.8859	-21.2468	-13.9653	21.8214
S4	c	2.66083	5.745	9.80616	10.8859	-21.2468	-13.9653	-21.8214

Example 1: 6 images with their Hu Moments

Due to these features Hu Moments are almost perfect for shape recognition in our leaves. However, to achieve this we should segment the leaves from the images and use their masks for disposing of elements that could lower the accuracy of Hu Moments detection. The segmentation will be discused later, in section **b**. As for Python algorythm, it is very simple to use:

```python
def fd_hu_moments(image):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    image_seg, image_mask = SegmentImage(file, False)
    feature = cv2.HuMoments(cv2.moments(image)).flatten()
    return feature
```

2) Haralick Texture

An image texture is a set of metrics calculated in image processing designed to quantify the perceived texture of an image. Image texture gives us information about the spatial arrangement of color or intensities in an image or selected region of an image.[2] Haralick texture is calculated from a gray level co-occurance matrix (GLCM) and it is a simple and most common technique of obtaining texture descriptors:

```python
def fd_haralick(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    haralick = mahotas.features.haralick(gray, ignore_zeros =
               True).mean(axis=0)
    return haralick
```

3) Color Histogram

In image processing and photography, 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.[3] In this task the histogram were build in three-dimentional space HSV due to it's slightly higher accuracy score than BGR color space:

```
def fd_histogram(image, mask):
    bins = 8
    image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    hist  = cv2.calcHist([image], [0, 1, 2], mask, [bins, bins, bins],
                          [0,256, 0, 256, 0, 256])
    cv2.normalize(hist, hist)
    return hist.flatten()
```
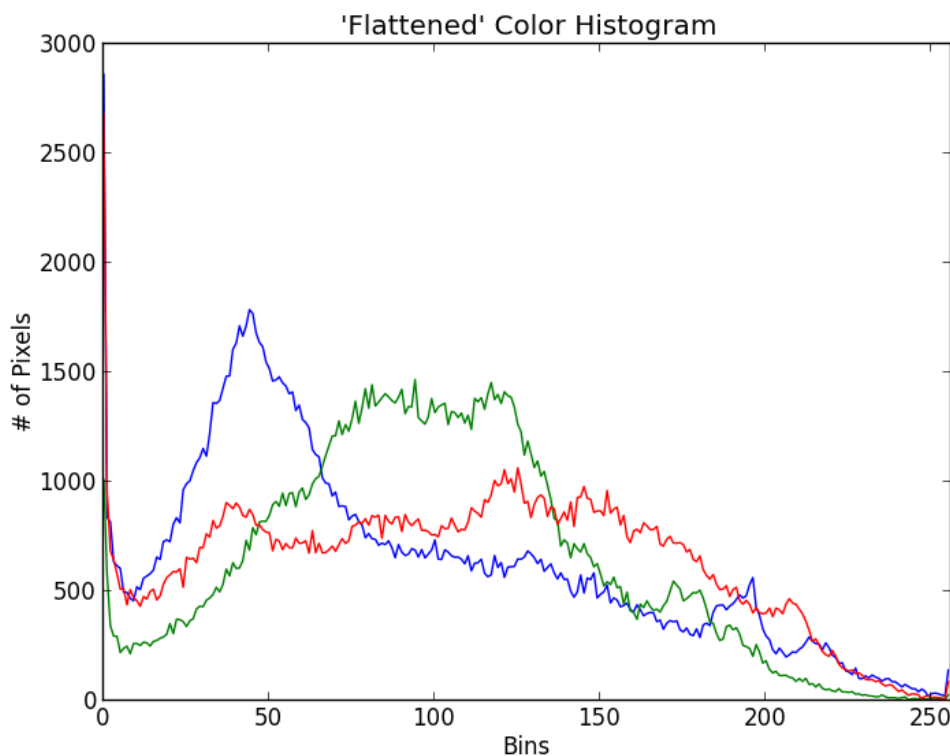


Example 2: Color histogram

4) Local Binary Patterns

It is the only local feature descriptor in this algorythm. Unlike Haralick textures, LBP computes a local representation of the texture by comparing each pixel with its surrounding.[4] It is similar to Haralick though it uses image in grayscale. As for the performance, it is slightly more accurate than Haralick, but it has very long execution time. In implementation it is also more complex, due to number of options and variables like number of circulary symetric neighbour set points, radius of the circle and the method of determining the pattern. In this algorythm values for variables were picked in straight-forward try-and-error way to find the most optimal ones:

```
def fd_loc_bin_patt(image, numPoints=24, radius=8, eps=1e-7):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

3

```
        lbp = feature.local_binary_pattern(image, numPoints, radius,
                                            method="uniform")
        hist, _ = np.histogram(lbp.ravel(), bins=np.arange(0, numPoints + 3),
                    range=(0, numPoints + 2))
        hist = hist.astype("float")
        hist /= (hist.sum() + eps)
        return hist
```
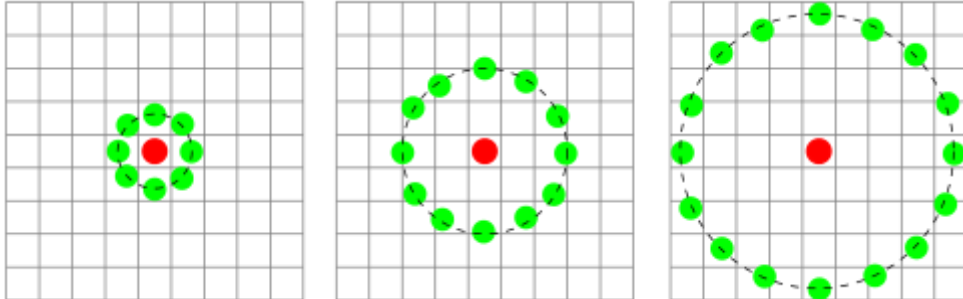

Example 3: Three neighborhood examples used to define a texture and calculate a local binary pattern

5) Zernike Moments

Zernike moments feature descriptors can be treated as an alternative for Hu moments, but they are not as popular as Hu's due to their complexity which greatly impacts the algorythm execution time.[5] Also unlike Hu moments function, they are not present in OpenCV library and have to be imported from the *mahotas* library. Although their usage is the simplest one of all:

```
def fd_zernike(image):
    return mahotas.features.zernike_moments(image, min(image.shape))
```

b) Segmentation

Even though mask computing isn't strictly a part of the task, it is necessary in order to get any positive results when using Hu and Zernike moments. It also may seem that the segmented leaf would be easier to classify using other feature descriptors, but, suprisingly, it was not the case. The results, however will be discussed in section **2**.

Segmentation algorythm is a simpler version of the algorythm used for previous task. We start from changing the color pallete to HSV and getting the mask of colors in specific range. Next, we find the contours of the biggest element and remove everything else in order to get rid of noise. Finally to get the segmented image we apply bitwise AND operation on the original image using calculated mask.

```
def SegmentImage(path, debug):
    image = cv2.imread(path,cv2.IMREAD_COLOR)
    hsv = cv2.cvtColor(image , cv2.COLOR_BGR2HSV)
    low = np.array([0, 0, 0],np.uint8)
    high = np.array([180, 255, 165],np.uint8)
    mask = cv2.inRange(hsv, low , high)

    ret,thresh = cv2.threshold(mask, 127, 255, 0)
```
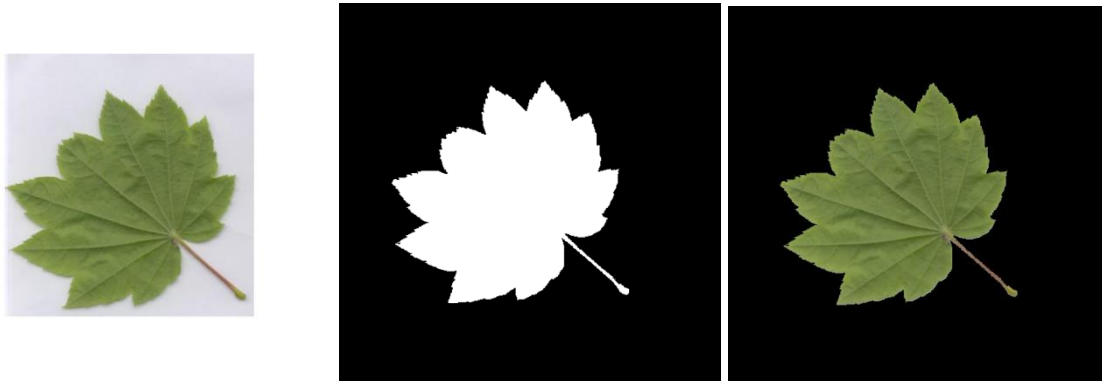
```python
        contours, hierarchy = cv2.findContours(thresh,
            cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    cont = sorted(contours, key = cv2.contourArea, reverse = True)[0]
    mask = cv2.drawContours(np.zeros((image.shape[0], image.shape[1], 1),
        np.uint8), [cont], 0, (255,255,255), cv2.FILLED)

    segmented = cv2.bitwise_and(image , image , mask=mask)
    ret, mask = cv2.threshold(mask, 127, 255, 0)

    return segmented, mask
```


Example 4: Original image, image mask and segmented image

c) Classifier

Since there are many classifiers to choose from, the one selected for this task was the one that should have the highest prediction score for such limited amount of input data. According to this[6] article it's Random Forest classifier from sklearn.ensemble library. The main idea behind this classifier is to construct a multitude of decision trees at training time and output the class that is the mode of the classes or mean prediction of the individual trees. Random decision forests correct the decision trees' habit of overfitting to their training set.[7]

To classify input data, the algorythm iterates over all leaves. First it creates segmented image and image mask, then it extracts all features descriptors from each image, tosses them into a stack and appends this stack into global array of descriptors.

```python
for training_name in train_labels:
    dir = os.path.join(train_path, training_name)
    current_label = training_name

    for x in range(1,len([im for im in os.listdir(dir)])+1):
        # get the image file name
        if x < 10:
            file = dir + "/l0" +  str(x) + ".jpg"
        else:
            file = dir + "/l" +  str(x) + ".jpg"

        image = cv2.imread(file)
        image_seg, image_mask = SegmentImage(file, False)

        fv_hu_moments = fd_hu_moments(image_mask)
        fv_haralick = fd_haralick(image_seg)
```

```python
                fv_histogram = fd_histogram(image, None)
                fv_loc_bin_patt = fd_loc_bin_patt(image, 24, 4)
                fv_zernike = fd_zernike(image_mask)


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

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

        print("[STATUS] processed folder: {}".format(current_label))
    print("[STATUS] completed Global Feature Extraction...")
```

Next step is to encode the target labels and scale global features to fit them into range (0-1)

```python
        # encode the target labels
        le = LabelEncoder()
        target = le.fit_transform(labels)
        # scale features in the range (0-1)
        scaler = MinMaxScaler(feature_range=(0, 1))
        rescaled_features = scaler.fit_transform(global_features)
```

All what's left is to initiate classifier, input the descriptors and the labels and calculate the results of the predictions using 10-fold validation.

```python
        clf  = RandomForestClassifier(n_estimators=num_trees, random_state=seed)
        kfold = KFold(n_splits=10, random_state=seed)
        cv_results = cross_val_score(clf, rescaled_features, target, cv=kfold,
                        scoring=scoring)
        print(cv_results.mean())
```

2. Results evalution and conclusions

From all of the feature descriptors the most useful for our model turned out to be the **Color Histogram**, which even as a single descriptor in a stack resulted in 88.42% of correct guesses. What seems to be strange is the fact that applying color histogram descriptor with the addition of image mask and just the segmented image changes the results to 84.11% and 83.88% respectively. The cause of such outcomes can be that the paper on which the leaves were laying on while taking the pictures. It differs between species, which further differentiates leaves' categories, positively impacting the predictions score.

Results of using shape recognition functions – **Hu and Zernike moments** were lower than the histogram ones – 67.81% and 57.14%. Having to choose between one of them, it would certainly be better to use Hu moments due to it's higher accuracy and way better (about 5x faster) performance. From those results we can conclude that for this sample in which leaves were in various stages of growth and fracture and the species themselves were similar to themselves, shape recognition alone may not be the best way to classify the sample.

As for the **Haralick texture** feature descriptor the results were slightly higher than Hu moments - 69.41%. It is quite good outcome, especially considering that for the human eye the differences between some of those species texture are hard to notice. It is worth to mention that this score was obtained for the segmented image. The score for the original was significantly lower – 61.82%. The conclusion can be that the paper which in case of color histogram improved the accuracy, this time did the opposite, probably due to fact that the texture of the paper was still the same since it was the same material.

The results for the local feature descriptor – **Local Binary Pattern** are also on the satisfactory level of 71.53%. The biggest downfall of this solution though is it's complexity which results in high execution time. Therefore, we can conclude that LBP may not be the best option for big samples of data.

That was the results of using only one feature descriptor. Here are some results of the few selected features combined in groups:

1) Haralick texture + Hu moments – 77.10%

2) Color histogram + Hu moments – 89.33%

3) Color histogram + Haralick textures – 89.55%

4) Color histogram + LBP – 89.33%

5) Color histogram + Haralick texture + Hu moments – 89.55%

6) Color histogram + LBP + Haralick texture + Hu moments – 90.46%

The highest prediction score was obtained after combining 4 features descriptors – which wasn't surprising at all. Furthermore, we can observe that combining features in the groups has noticeable effect on the score only in case of combining descriptors which didn't have very high score to begin with and also the difference between those descriptors were not big. (i.e. combining Haralick texture and Hu moments – from [69.41%, 67.81%] to 77.10%).

3. Source code

```python
import numpy as np
import cv2
import mahotas
import os
import glob
import shutil
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.ensemble import RandomForestClassifier
from skimage import feature

import warnings
warnings.filterwarnings('ignore')

#variables declaration
train_path = '../../isolated'
bins = 8
num_trees = 100
test_size = 0.20
seed = 9
scoring = "accuracy"

# 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):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    haralick = mahotas.features.haralick(gray, ignore_zeros = True).mean(axis=0)
    return haralick

# feature-descriptor-3: Color Histogram
def fd_histogram(image, mask):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    hist  = cv2.calcHist([image], [0, 1, 2], mask, [bins, bins, bins], [0, 256, 0, 256, 0, 256])
    cv2.normalize(hist, hist)
    return hist.flatten()

# feature-descriptor-4: Local Binary Pattern
def fd_loc_bin_patt(image, numPoints=24, radius=8, eps=1e-7):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    lbp = feature.local_binary_pattern(image, numPoints, radius, method="uniform")
    hist, _ = np.histogram(lbp.ravel(), bins=np.arange(0, numPoints + 3), range=(0, numPoints + 2))
    hist = hist.astype("float")
    hist /= (hist.sum() + eps)
    return hist

# feature-descriptor-5: Zernike Moments
def fd_zernike(image):
```

```python
        return mahotas.features.zernike_moments(image, min(image.shape))


def SegmentImage(path, debug):

    #read image
    image = cv2.imread(path,cv2.IMREAD_COLOR)

    #change color palette and create a mask containing specific color range
    hsv = cv2.cvtColor(image , cv2.COLOR_BGR2HSV)
    low = np.array([0, 0, 0],np.uint8)
    high = np.array([180, 255, 165],np.uint8)
    mask = cv2.inRange(hsv, low , high)

    #compute contours and keep only the largest one
    ret,thresh = cv2.threshold(mask, 127, 255, 0)
    contours, hierarchy = cv2.findContours(thresh, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    cont = sorted(contours, key = cv2.contourArea, reverse = True)[0]
    mask = cv2.drawContours(np.zeros((image.shape[0], image.shape[1], 1), np.uint8),
[cont], 0, (255,255,255), cv2.FILLED)

    #create segmented image using bitwise AND operation
    segmented = cv2.bitwise_and(image , image , mask=mask)
    ret, mask = cv2.threshold(mask, 127, 255, 0)

    if debug:
        cv2.imshow("Image", image)
        cv2.imshow("mask", mask)
        cv2.imshow("segmented", segmented)
        cv2.waitKey()

    return segmented, mask



train_labels = os.listdir(train_path)
train_labels.sort()

global_features = []
labels = []


# iterate over species folders
for training_name in train_labels:
    dir = os.path.join(train_path, training_name)
    current_label = training_name

    #iterate over all images in folder
    for x in range(1,len([im for im in os.listdir(dir)])+1):
        # get the image file name
        if x < 10:
            file = dir + "/l0" +  str(x) + ".jpg"
        else:
            file = dir + "/l" +  str(x) + ".jpg"

        #read image and get it's mask and segmented version
        image = cv2.imread(file)
        image_seg, image_mask = SegmentImage(file, False)

        #get all features descriptors
        fv_hu_moments = fd_hu_moments(image_mask)
```

```python
        fv_haralick    = fd_haralick(image_seg)
        fv_histogram  = fd_histogram(image, None)
        fv_loc_bin_patt = fd_loc_bin_patt(image, 24, 4)
        fv_zernike      = fd_zernike(image_mask)

        #put desired descriptors in a stack
        global_feature = np.hstack([fv_histogram, fv_loc_bin_patt])
        #global_feature = np.hstack([fv_hu_moments])
        #global_feature = np.hstack([fv_haralick])
        #global_feature = np.hstack([fv_histogram])
        #global_feature = np.hstack([fv_loc_bin_patt])
        #global_feature = np.hstack([fv_zernike])

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

    print("[STATUS] processed folder: {}".format(current_label))

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


# encode the target labels
le           = LabelEncoder()
target       = le.fit_transform(labels)
# scale features in the range (0-1)
scaler             = MinMaxScaler(feature_range=(0, 1))
rescaled_features = scaler.fit_transform(global_features)

#initiate ML model
clf  = RandomForestClassifier(n_estimators=num_trees, random_state=seed)

#initiate 10-fold validation function
kfold = KFold(n_splits=10, random_state=seed)

#predict the images labels using model with input data and 10-fold validation
cv_results = cross_val_score(clf, rescaled_features, target, cv=kfold,
scoring=scoring)
print(cv_results.mean())
```

4. References

[1] https://www.learnopencv.com/shape-matching-using-hu-moments-c-python/

[2] https://en.wikipedia.org/wiki/Image_texture
https://gogul.dev/software/texture-recognition
https://www.nature.com/articles/s41598-017-04151-4

[3] https://en.wikipedia.org/wiki/Color_histogram
https://www.pyimagesearch.com/2014/01/22/clever-girl-a-guide-to-utilizing-color-histograms-for-computer-vision-and-image-search-engines

[4] https://en.wikipedia.org/wiki/Local_binary_patterns
https://www.pyimagesearch.com/2015/12/07/local-binary-patterns-with-python-opencv

[5] https://pdfs.semanticscholar.org/b4d3/d68e098cde69b8608a8c61d12027ef9caf81.pdf

[6] https://gogul.dev/software/image-classification-python

[7] https://en.wikipedia.org/wiki/Random_forest