Elaboration of discriminative feature vector (for classification purpose) Lab 2

Maciej Zalewski
m.zalewski2@student.mini.pw.edu.pl

February 3, 2020

Contents

1 Introduction

Features are pieces of information characteristic for given image which are understandable to computer. They range from color distribution to localization of edges. By combining such selected data together one can create a discriminative feature vector. Data prepared like this are very handy in classification problem where the program has to automatically select a group label to an analyzed image. In this solution machine learning random forest method will be used to classify images of leaves. Dataset will be divided into two groups where the first will be used for training purposes and the second for tests. Main focus is put on different methods of features extraction.
Dataset consists of 6 kinds of leaves(442): circinatum(66), garryana(84), glabrum(75), kelloggii(97), macrophyllum(82), negundo(38).

2 Methods description

2.1 Random Forest Classifier

For classification purposes we will use random forest method (Machine Learning) from scikit-learn Python library. We won't go into details because it's outside the scope of this laboratory project. Parameters will be the same for every method.
Each of methods below produces features which are combined together at the end into one vector.

```
global_feature = np.hstack([fv_orb, fv_zernike,
fv_lbp, fv_histogram, fv_haralick, fv_hu_moments])
```

2.2 Hu Moments

Hu Moments is from the shape group of methods. It computes a set of seven numbers calculated using central moments that are invariant to image transformations. The first six moments are proved to be invariant to translation, scale, rotation and reflection. Function used here comes from OpenCV library.

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

2.3 Haralick

Haralick texture uses gray level co-occurrence matrices. Method is based on the joint porbability distributions of pairs of pixels. Function used here comes from mahotas library.

```
# Haralick Texture (texture)
def fd_haralick(image):
image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
haralick = mahotas.features.haralick(image).mean(axis=0)
return haralick
```

2.4 Color Histogram

Color Histogram method is simply based on creating a color histogram of a given image and then normalizing and flattening it to the single vector. Extracted features tell about color.

```
# Color Histogram (color)
def fd_histogram(image, mask=None):
image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
hist  = cv2.calcHist([image], [0, 1, 2], None, [8, 8, 8], [0, 256, 0, 256, 0, 256])
cv2.normalize(hist, hist)
return hist.flatten()
```

2.5 Local Binary Patterns

Local Binary Patterns is another texure method. Each pixel is compared with the chosen number of pixels on the chosen radius distance. Based on this comparisions histograms are created which are also our features. Function used here comes from scikit-learn library.

```
# Local Binary Patterns (texture)
def fd_localBinaryPatterns(image):
eps = 1e-7
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
lbp = feature.local_binary_pattern(gray, LBPpoints, LBPradius, method="uniform")
(hist, _) = np.histogram(lbp.ravel(), bins=np.arange(0,LBPpoints + 3), range=(0, LBPpoints +
hist = hist.astype("float")
eps=1e-7
hist /= (hist.sum() + eps)
return hist.flatten()
```

2.6 Zernike Moments

Zernike Moments shape descriptor is based on Zernike polynomials to calculate image moments. Function comes from mahotas library. As default degree of polynomials is set to 8 and the radius in pixels from the center of mass is to be chosen. Pixels outside of the circle are ignored.

```
# Zernike Moments (shape)
def fd_zernikeMoments(image):
image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
return mahotas.features.zernike_moments(image, zernikeRadius)
```

2.7 Histogram of Oriented Gradients

Histogram of Oriented Gradients descriptor counts occurences of gradient orientation in localized portions of an image. Image is divided into small cells and histograms of gradient directions are computed for each one. Groups of adjacent cells are called blocks. After some transformations on the cells and grouping the set of block histograms is received which is our feature.

```
# Histogram of Oriented Gradients
def fd_histogramOrientedGradients(image):
image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
H = feature.hog(image, orientations=9, pixels_per_cell=(8, 8),
cells_per_block=(2, 2), transform_sqrt=True, block_norm="L1")
return H.flatten()
```

2.8 Oriented FAST and Rotated BRIEF

Oriented FAST and Rotated BRIEF is a local feature detector. It's based on the FAST keypoint detector and visual descriptor BRIEF. Good and efficient alternative to SIFT which is patented and one must pay to use it while ORB is free. Maximal number of retained can be set, by trial and error the best results were for nfeatures = 100. They can be bounded from the top, but we have to be sure that 100 is always received at the end. If the method produes less then 100 features, the rest of the vector is filled with 0.

```
# Oriented FAST and Rotated BRIEF (local feature)
def fd_orb(image):
orb = cv2.ORB_create(nfeatures = 100)
keypoints = orb.detect(image,None)
(_, descriptors) = orb.compute(image, keypoints)
```

```
ft=descriptors.flatten()
if len(ft)<100:
ft = np.zeros(100-len(ft),dtype=int)
else:
ft = np.array(ft[0:100])
return ft
```

3 Analysis of results

Dataset was divided into two parts, 80% was used as training data and the rest was a testing set. Two ways of testing the results have been chosen. The first one is k-fold Cross-Validation which is a statistical method used to estimate machine learining models skills. Computes accuracy for our model with data used to train it, here number of folds has been set to 10.

```
# KFOLD test
kfold = KFold(n_splits=10, random_state=seed)
cv_results = cross_val_score(model, glob_features, global_labels, cv=kfold, scoring=scoring)
msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
print(msg)
```

The second way of testing was classification for the test data and comparision with the right groups. Built in function for Random Forest Classifier, predict, has been used. As the output program counted number of correct predictions for given groups and saved labeled image in the output folder. Based on collected informations, average accuracy for every group and overall accuracy were calculated.

```
# 80/20 test
model.fit(glob_features, global_labels)

count = 1
results = [0,0,0,0,0,0]

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

image = cv2.imread(file)
image = cv2.resize(image, fixed_size)

fv_hu_moments = fd_hu_moments(image)
fv_haralick   = fd_haralick(image)
fv_histogram  = fd_histogram(image)
```

```
fv_lbp = fd_localBinaryPatterns(image)
fv_zernike = fd_zernikeMoments(image)
# fv_hog = fd_histogramOrientedGradients(image)
fv_orb = fd_orb(image)

test_global_feature = np.hstack([fv_orb, fv_zernike, fv_lbp, fv_histogram, fv_haralick, fv_h

# predict label of test image
prediction = model.predict(test_global_feature.reshape(1,-1))[0]

# Hardcoded numbers of given leaves to test
if count < 14:
if prediction == 0:
results[0] += 1
if 13 < count < 28:
if prediction == 1:
results[1] += 1
if 27 < count < 41:
if prediction == 2:
results[2] += 1
if 40 < count < 58:
if prediction == 3:
results[3] += 1
if 57 < count < 73:
if prediction == 4:
results[4] += 1
if count > 72:
if prediction == 5:
results[5] += 1

# show predicted label on image
cv2.putText(image, train_labels[prediction], (20,30), cv2.FONT_HERSHEY_SIMPLEX, 1.0, (0,255,
cv2.imwrite(output_path + '/' + str(count) + '.jpg', image)
count = count + 1

average = []
for i in range(6):
average.append(results[i]/images_for_test[i])

print(*results, sep = ", ")
print(*average, sep = ", ")
print(sum(results)/79)
```

3.1 Analysis of results

Results for single methods and groups of them are presented in the tabel below.

Results								
Methods	k-fold	prediction	cir.	gar.	gla.	kel.	mac.	neg.
Hu Moments	0.12	0.15	0.08	0.07	0.08	0.29	0.20	0.14
Haralick	0.59	0.78	0.69	0.79	0.77	0.82	0.80	0.86
Color Histogram	0.87	0.99	1.0	1.0	1.0	0.94	1.0	1.0
LBP	0.68	0.86	0.77	0.93	0.69	1.0	0.93	0.71
Zernike Moments	0.31	0.56	0.46	0.43	0.62	0.82	0.67	0.0
HOG	0.21	0.52	0.3	0.64	0.46	0.76	0.40	0.43
ORB	0.23	0.47	0.23	0.36	0.54	0.53	0.80	0.14
CH + LBP	0.89	0.99	1.0	1.0	1.0	0.94	1.0	1.0
Har. + CH + LBP	0.88	1.0	1.0	1.0	1.0	1.0	1.0	1.0
All - Hu Mom. - ORB - HOG	0.89	0.99	1.0	1.0	1.0	0.94	1.0	1.0
All - Hu Mom.	0.41	0.82	0.69	0.86	0.92	0.88	0.86	0.57
All	0.43	0.84	0.69	0.86	1.0	0.88	0.87	0.57

It is easy to see that the best single methods for this model are Haralick, Color Histogram and Local Binary Patterns, so as their combinations. Surprisingly using together all the methods does not give the best result. It is because of the methods which are bad or not very good in this case, such as Hu Moments with terrible prediction accuracy at 0.15 or ORB being in the lower half at 0.47 which drasticaly lowers overall results.
Perfect result have been received for Haralick + Color Histogram + LBP with prediction at 1.0, which means that all images from test set has been labeled correctly. Color Histogram + LBP and Haralick + Color Histogram + LBP + Zernike Moments were also very close to this with small error for kelloggii leaves so that average prediction were 0.99, but their k-fold score were 0.89 compared to 0.88 for our perfect solution. They were very close and possibly for different dataset they would perform the same as Haralick + Color Histogram + LBP or even better, but k-fold uses statistics, so we cannot be quite sure.
In conclusion, more features descriptors don't mean better result. We were able to find the combination of methods which in predicted every label without any mistake and two similar ones with nearly the same accuracy. Choosing proper methods depends on the dataset and Machine Learning model.

3.2 Possible changes

Fortunately quality of images in the given dataset was very good, so there was no need for preprocessing them and performing segmentation. For different one such steps could be necessary, for example when big images are given and only part of them is interesting for us in terms of classification (ex. car logos).

4 Source code

```
from sklearn.preprocessing import LabelEncoder
import copy
from sklearn.preprocessing import MinMaxScaler
from skimage import feature
from matplotlib import pyplot as plt
import numpy as np
import mahotas
import cv2
import os
import h5py

import glob
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold, StratifiedKFold
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
from sklearn.ensemble import RandomForestClassifier
import imutils
import warnings


warnings.filterwarnings('ignore')

# parameters
images_per_class = {
"circinatum": 53,
"garryana": 70,
"glabrum": 62,
"kelloggii": 80,
"macrophyllum": 67,
"negundo": 31
}

images_for_test = [13,14,13,17,15,7]

fixed_size       = tuple((512, 512))
train_path       = "dataset/train"
test_path        = "dataset/test"
output_path      = "output"

LBPradius        = 3
LBPpoints        = 8 * LBPradius

zernikeRadius    = 150
```

```
num_trees = 100
test_size = 0.10
seed      = 9
scoring    = "accuracy"

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

# Haralick Texture (texture)
def fd_haralick(image):
image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
haralick = mahotas.features.haralick(image).mean(axis=0)
return haralick

# Color Histogram (color)
def fd_histogram(image, mask=None):
image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
hist  = cv2.calcHist([image], [0, 1, 2], None, [8, 8, 8], [0, 256, 0, 256, 0, 256])
cv2.normalize(hist, hist)
return hist.flatten()

# Local Binary Patterns (texture)
def fd_localBinaryPatterns(image):
eps = 1e-7
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
lbp = feature.local_binary_pattern(gray, LBPpoints, LBPradius, method="uniform")
(hist, _) = np.histogram(lbp.ravel(), bins=np.arange(0,LBPpoints + 3), range=(0, LBPpoints +
hist = hist.astype("float")
eps=1e-7
hist /= (hist.sum() + eps)
return hist.flatten()

# Zernike Moments (shape)
def fd_zernikeMoments(image):
image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
return mahotas.features.zernike_moments(image, zernikeRadius)

# Histogram of Oriented Gradients
def fd_histogramOrientedGradients(image):
image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
H = feature.hog(image, orientations=9, pixels_per_cell=(8, 8),
cells_per_block=(2, 2), transform_sqrt=True, block_norm="L1")
```

```
return H.flatten()

# Oriented FAST and Rotated BRIEF (local feature)
def fd_orb(image):
orb = cv2.ORB_create(nfeatures = 100)
keypoints = orb.detect(image,None)
(_, descriptors) = orb.compute(image, keypoints)
ft=descriptors.flatten()
if len(ft)<100:
ft = np.zeros(100-len(ft),dtype=int)
else:
ft = np.array(ft[0:100])
return ft

# 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          = []

# 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

# loop over the images in each sub-folder
for x in range(1,images_per_class[current_label]+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 = cv2.resize(image, fixed_size)

fv_hu_moments = fd_hu_moments(image)
fv_haralick   = fd_haralick(image)
```

```
fv_histogram  = fd_histogram(image)
fv_lbp = fd_localBinaryPatterns(image)
fv_zernike = fd_zernikeMoments(image)
fv_hog = fd_histogramOrientedGradients(image)
fv_orb = fd_orb(image)

global_feature = np.hstack([fv_orb, fv_hog, fv_zernike, fv_lbp, fv_histogram, fv_haralick, f

# 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...")

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

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

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

glob_features = np.array(global_features)
global_labels   = np.array(target)

print("[STATUS] features shape: {}".format(glob_features.shape))
print("[STATUS] labels shape: {}".format(global_labels.shape))

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

model = RandomForestClassifier(n_estimators=num_trees, random_state=seed)
name = "RandomForestClassifier"

# KFOLD test
kfold = KFold(n_splits=10, random_state=seed)
cv_results = cross_val_score(model, glob_features, global_labels, cv=kfold, scoring=scoring)
msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
print(msg)
```

```
# 80/20 test
model.fit(glob_features, global_labels)

count = 1
results = [0,0,0,0,0,0]

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

image = cv2.imread(file)
image = cv2.resize(image, fixed_size)

fv_hu_moments = fd_hu_moments(image)
fv_haralick   = fd_haralick(image)
fv_histogram  = fd_histogram(image)
fv_lbp = fd_localBinaryPatterns(image)
fv_zernike = fd_zernikeMoments(image)
fv_hog = fd_histogramOrientedGradients(image)
fv_orb = fd_orb(image)

test_global_feature = np.hstack([fv_orb, fv_hog, fv_zernike, fv_lbp, fv_histogram, fv_harali

# predict label of test image
prediction = model.predict(test_global_feature.reshape(1,-1))[0]

# Hardcoded numbers of given leaves to test
if count < 14:
if prediction == 0:
results[0] += 1
if 13 < count < 28:
if prediction == 1:
results[1] += 1
if 27 < count < 41:
if prediction == 2:
results[2] += 1
if 40 < count < 58:
if prediction == 3:
results[3] += 1
if 57 < count < 73:
if prediction == 4:
results[4] += 1
if count > 72:
if prediction == 5:
results[5] += 1

# show predicted label on image
```

```
cv2.putText(image, train_labels[prediction], (20,30), cv2.FONT_HERSHEY_SIMPLEX, 1.0, (0,255,
cv2.imwrite(output_path + '/' + str(count) + '.jpg', image)
count = count + 1

average = []
for i in range(6):
average.append(results[i]/images_for_test[i])

print(*results, sep = ", ")
print(*average, sep = ", ")
print(sum(results)/79)
```

5 Sources

https://gogul.dev/software/image-classification-python?fbclid=IwAR1gbnXYX5ZqSjzq7etUDlsJ-csS

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

http://wiki.awf.forst.uni-goettingen.de/wiki/index.php/Haralick_Texture

https://en.wikipedia.org/wiki/Local_binary_patterns

https://software.intel.com/en-us/ipp-dev-reference-histogram-of-oriented-gradients-hog-descr

https://www.pyimagesearch.com/2014/04/07/building-pokedex-python-indexing-sprites-using-shap

https://www.pyimagesearch.com/2015/12/07/local-binary-patterns-with-python-opencv/

https://gurus.pyimagesearch.com/lesson-sample-histogram-of-oriented-gradients-and-car-logo-r

And lecture slides

circinatum

Figure 1: Labeling image