Introduction to image processing and computer vision

Plant Species Recognition

Laboratory Project 2

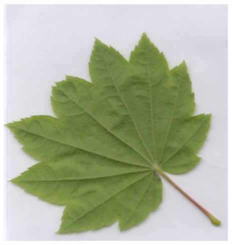
Patryk Walczak

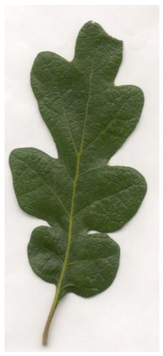
January 27, 2020

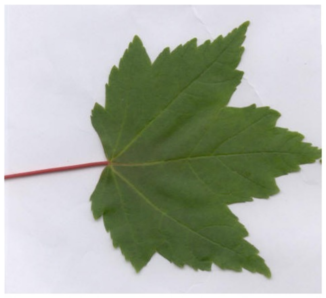
Contents

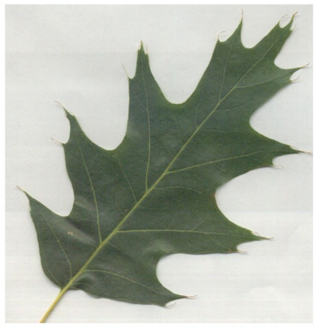
1 Introduction

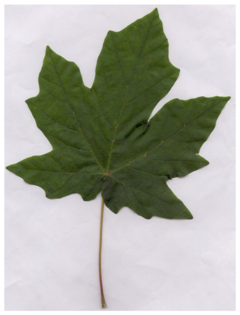
The main task of the project is an elaboration of a discriminative feature vector for the leaves of trees. Using computer vision processing find the vectors then learn the machine learning model with 80% of the data set and test the results on the remaining part. Six tree species each in the separated dictionary are stored in "isolated" dictionary. In each class is from 38 up to 97 images of leaves.


(a) Acer Circinatum
Vine Maplel


(b) Quercus Garryana
Oregon White Oak


(c) Acer Glabrum
Douglasii


(d) Quercus Kelloggii
California Black Oak


(e) Acer Macrophyllum
Big Leaf Maple

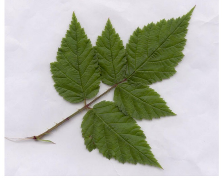

(f) Acer Negundo
Boxelder

Figure 1: Pictures of leaves

1.1 Project Description

For classification, I chose a random forest method, which is the ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees. All images are analyzed by 6 descriptors, all of them are used to learn the algorithm and after that to recognise images from the test set. For all-purpose, I prepared 5 python files.

List of files :

- descriptors.py - contains all descriptors functions
- summary.py - uses to save all results in files
- separate_images.py - copies files to ”train&test” directory and separates them to train and test sets
- train_test.py - the main file contains all train and test functions
- cross_validation.py - file contains function with cross-validation

2 Classification

In the section the processes of classification leaves, using the descriptors, training and testing model are described.

2.1 Feature Descriptors

The subsection includes descriptions of all descriptors.

2.1.1 Hu Moments

Hu moments are used to calculate moments that are invariant to translation, scale, and rotation. To clarify, an image moment is a certain particular weighted average (moment) of the image pixels' intensities or a function of such moments, usually chosen to have some attractive property or interpretation. The feature descriptor is in the first function in "descriptors.py".

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

This is a simple function. It takes as an input only one image, changes the image's colors to black and white and computes Hu Moments. In the end, it flattens the results to one array.

2.1.2 Haralick Texture

Texture defines the consistency of patterns and colors in an image. Haralick suggested the use of grey level co-occurrence matrices (GLCM). The basic idea of GLCM - it searches for pairs of adjacent pixel values that occur in an image and keeps recording it over the entire image. The Haralick Texture is computed in the second function in "descriptors.py".

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

The function takes only one image as input, and returns haralick descriptor. Firstly it changes the colors of the image to black and white. After that using "mahotas" library function computes the feature descriptor.

2.1.3 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. The ranges converted to the vector is the result of the third feature descriptor in ”descriptors.py”.

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

The function is even easier than the previous two. The descriptor for computation needs only one image. Then even no changes of colours are required. Using the OpenCV library and the calcHist method the function computes the histogram. After that, the histogram is normalized and returned flattened.

2.1.4 Histogram of Oriented Gradients

The Histogram of Oriented Gradients descriptor technique counts occurrences of gradient orientation in localized portions of an image. The feature descriptor is computed as the fourth one in descriptors.py”.

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

As the rest of descriptor functions, the one takes only one image as input also. Then changing the colors to black and white occurs. The ”feature” library helps with the computing of the histogram of the oriented gradients. And the result of the hog method is returned.

2.1.5 Local Binary Pattern

In the whole project, there is only two local descriptors - the first one is Local Binary Pattern, which computes a local representation of texture. The input image is changed to the black-and-white photo, then using ”feature” library and its method local_binary_pattern computes the descriptor.

```python
def fd_binary_pattern(image):
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
lbp = feature.local_binary_pattern(gray, 24, 8, method="uniform")
(hist, _) = np.histogram(lbp.ravel(),
bins=np.arange(0, 27),
range=(0, 26))
hist = hist.astype("float")
hist /= (hist.sum() + 1e-7)
return hist
```

2.1.6 Oriented FAST and rotated BRIEF

The second local descriptor is Oriented FAST and rotated BRIEF. For that one, the results were different for each kind of tree. To prevent the different lengths of the descriptor vectors, after computing the descriptor the lengths are changed to 100. If the vector has less then 100 values the remaining is a sequence of zeros, in another case when vector has more than 100 values only the first hundred is taken as the result.

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

2.1.7 Unused

I prepared Scale Invariant Feature Transform descriptor also but the method SIFT_create() from cv2.xfeatures2d returned the communication "This algorithm is patented and is excluded in this configuration". So that I decided not to use the descriptor in the project.

```
# SIFT
def fd_SIFT(image):
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
freakExtractor = cv2.xfeatures2d.FREAK_create()
keypoints,descriptors= freakExtractor.compute(gray,None)
return keypoints
```

After searching for information, I found the free version of SIFT algorithm but it does not return any features. The algorithm throws an error because the machine learning model cannot learn from an empty feature vector. So as in the previous case, I decided to not use the descriptor in the project.

```
# SIFT v2
def fd_SIFT_v2(image):
gray= cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
sift = cv2.xfeatures2d.SIFT_create()
kp = sift.detect(gray,None)
image=cv2.drawKeypoints(gray,kp,image)
cv2.show('sift_keypoints.jpg',image)
return kp
```

2.2 Cross validation

Cross-validation is a process which predicts the ML model's accuracy. In the project is used 10-fold cross-validation which means that the data set is divided into 10 subsets. The algorithm predicts the result 10 times, every time 9 sets are used to train and one for tests. Each iteration other set is the test one, so each set is once used for testing and nine times for training. The average of the ten results is returned.

```
print("10-fold cross validation")
class_names = ["circinatum", "garryana", "glabrum", "kelloggii", "
    macrophyllum","negundo"]
all_descriptors = []
labels = []
for label in class_names:
for jpgfile in glob.glob("isolated/"+label+"/*.jpg"):
image = cv2.imread(jpgfile)
image = cv2.resize(image, (512,512))
```

Every image in the data set is loaded and resized to the same measurements.

```
fv_hu_moments  = fd_hu_moments(image)
fv_haralick    = fd_haralick(image)
fv_histogram   = fd_histogram(image)
fv_hog         = fd_histog(image)
fv_binpat      = fd_binary_pattern(image)
fv_orb         = fd_orb(image)
fv_SIFT        = fd_SIFT(image)
# save them in descriptors variable
 descriptors = np.hstack([fv_histogram , fv_haralick, fv_hu_moments
    , fv_hog , fv_binpat, fv_orb])
```

The stack of descriptors are saved in all_descriptors list and spieces are saved in labels list.

```
# save labels and feature as the vectors
labels.append(label)
all_descriptors.append(descriptors)
```

Then from the library "sklearn.model_selection" KFold function is taken and the algorithms prepares the 10-fold cross-validation. After that function creates the ML model - Random Forest Classifier. In the end, the program computes the cross-validation and prints mean and standard deviation of the ten results of the 10-fold algorithm.

```
# # 10-fold cross validation
kfold = KFold(n_splits=10)
rf = RandomForestClassifier(n_estimators=100, random_state=9)
cv_results = cross_val_score(rf, train_data, train_labels, cv=kfold,
     scoring="accuracy")
msg = "%s: %f (%f)" % ('RF', cv_results.mean(), cv_results.std())
print(msg)
```

2.3 Division of data set

For classification purposes, the data set has to be split into two parts. The first with 80% of all images are used to train the method. The second with remaining 20% is used for test of the method.

In ”separate_images.py” I prepared the function for spliting the data set.

```
class_names = ["circinatum", "garryana", "glabrum", "kelloggii", "
    macrophyllum","negundo"]
directory = "train&test"

def separeteImages():
```

Firstly, the program creates new empty directories to store the images.

```
# create new empty directories
if os.path.exists(directory):
shutil.rmtree(directory, ignore_errors=True)
os.makedirs(directory)
```

Next, for every kind of tree creates test and train directories.

```
# for every spiece
for image in class_names:
# prepare two directories
# one for train images
train_dir = directory+"/"+image
os.mkdir(train_dir)
train_dir += "/train"
os.mkdir(train_dir)
# one for test images
test_dir = directory+"/"+image+"/test"
os.mkdir(test_dir)
```

After that, the program starts splitting all photos in the set

```
# for every image in data set
i=0
for jpgfile in glob.glob("isolated/"+image+"/*.jpg"):
if(i%5==0):
shutil.copy(jpgfile, test_dir)
else:
shutil.copy(jpgfile, train_dir)
i+=1
```

Every image with index number equal to 0 modulo 5 is saved in the test directory, and the rest are stored in train directory. For comparison of the descriptors, I preformed the code only once at the start of the project. This provides that for all results algorithm always has the same input data.

2.4 Train and test

The section covers the most important part of the project. Here is the description of the main algorithm which learn the ML model and after that tests the results. The algorithm is included in "train_test.py"

At the beginning, the program reads all images in train directories from all species and resizes all of them to common measurements.

```
all_descriptors = []
labels = []
# learning from the train directories
for label in class_names:
for file in glob.glob('train&test/'+(label)+'/train/*.jpg'):
# read the image and resize it to the common size
image = cv2.imread(file)
image = cv2.resize(image, (512,512))
```

For each image, all descriptors are computed and saved in all_descriptors list, while the species names are stored in labels list.

```
# compute descriptors
fv_hu_moments = fd_hu_moments(image)
fv_haralick   = fd_haralick(image)
fv_histogram  = fd_histogram(image)
fv_hog        = fd_histog(image)
fv_binpat     = fd_binary_pattern(image)
fv_orb        = fd_orb(image)
# fv_SIFT        = fd_SIFT(image)
# save them in descriptors variable
descriptors = np.hstack([fv_histogram, fv_haralick, fv_hu_moments,
    fv_hog, fv_binpat, fv_orb])
# save labels and feature as the vectors
labels.append(label)
all_descriptors.append(descriptors)
```

When the training set of data is completed, the program creates the Random Forest Classifier and learns the ML model.

```
# Random Forests Classifier
clf = RandomForestClassifier(n_estimators=100, random_state=9)
# learning the model
clf.fit(all_descriptors, labels)
```

When learning is done, the algorithm starts testing section by reading all images and resizing them.

```
for label in class_names:
for file in glob.glob('train&test/'+(label)+'/test/*.jpg'):
# read the image and resize it to the common size
image = cv2.imread(file)
image = cv2.resize(image, (512,512))
```

Once again program computes all descriptors but in this case for the testing images.

```
        # compute descriptors
        fv_hu_moments = fd_hu_moments(image)
        fv_haralick   = fd_haralick(image)
        fv_histogram  = fd_histogram(image)
        fd_hog        = fd_histog(image)
        fd_binpat     = fd_binary_pattern(image)
        fv_orb        = fd_orb(image)
        # fv_SIFT        = fd_SIFT(image)
        # save them in descriptors variable
        descriptors = np.hstack([fv_histogram, fv_haralick,
    fv_hu_moments, fv_hog, fv_binpat, fv_orb])
```

Next step is the prediction and checking the results. Name of the kind of tree is printed out with the predicted name. User can easily and in real-time check the results.

```
prediction = clf.predict(descriptors.reshape(1,-1))[0]
print(label,prediction)
results.append((label,prediction))
```

All of the results are stored in the results list. The reason for that is an optional function which can save the data to files and compute the mean. The function "save_and_summary" is described in the next subsection.

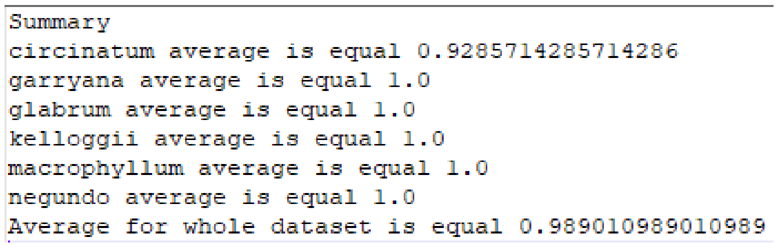
```
save_and_summary(results)
```

2.5 Summary

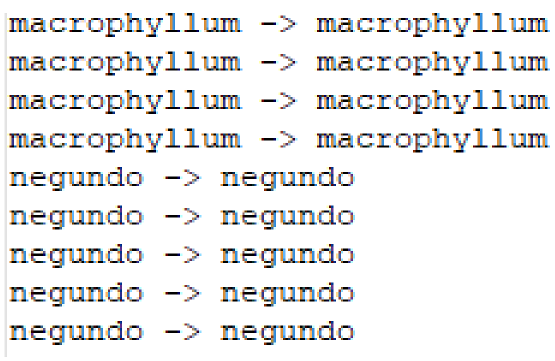
For saving results for each image, each kind of tree and for all data set I prepared summary.py file.

At first, the program creates two empty files in directory "test&file". AllResults.txt file includes all results in form "kind of tree" → "prediction", and Summary.txt file includes a summary for each class and whole data set.

```
# function for save results and summary
def save_and_summary( results ):
# create files to save results
if not os.path.exists("./train&test"):
os.makedirs("./train&test")
AllResults = open("./train&test/AllResults.txt" , 'w')
Summary = open("./train&test/Summary.txt" , 'w')
```



```
Summary
circinatum average is equal 0.9285714285714286
garryana average is equal 1.0
glabrum average is equal 1.0
kelloggii average is equal 1.0
macrophyllum average is equal 1.0
negundo average is equal 1.0
Average for whole dataset is equal 0.989010989010989
```



```
macrophyllum -> macrophyllum
macrophyllum -> macrophyllum
macrophyllum -> macrophyllum
macrophyllum -> macrophyllum
negundo -> negundo
negundo -> negundo
negundo -> negundo
negundo -> negundo
negundo -> negundo
```

Variable "results" is a list of tuples with the correct answer at the 1st position and prediction of the model at the 2nd position in each tuple.

```
# from train&test.py
results.append((label,prediction))
```

Then the algorithm creates two dictionaries and saves the results to AllResults.txt. In the same loop it computes the number of correct predictions.

```python
# create 2 dictionaries for computing correct preditions
num = dict()
den = dict()
for name in class_names:
num[name] = 0
den[name] = 0
for result in results:
# write all preditions to AllResults file
AllResults.write(' -> '.join(result)+'\n')
# compute correctness for each spiecie
den[result[0]] += 1
if result[0] == result[1]:
num[result[0]] += 1
```

In the next step, the program starts summarising. Firstly, it computes the result for each kind of tree.

```python
n = 0
d = 0
# Summary
Summary.write("Summary\n")
for name in class_names:
# write results for each class
if den[name] != 0:
line = name+" average is equal "+ str(num[name]/den[name])
Summary.write(line+"\n")
print(line)
n += num[name]
d += den[name]
```

Then it computes mean for all data set.

```python
if d != 0:
# write result for whole data set
line = "Average for whole dataset is equal "+ str(n/d)
Summary.write(line+"\n")
print(line)
```

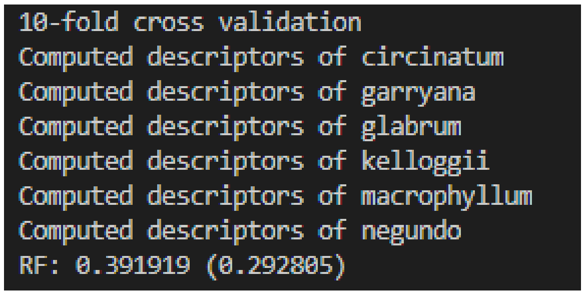
At the end program closes the files.

3 Results

3.1 10-fold cross validataion

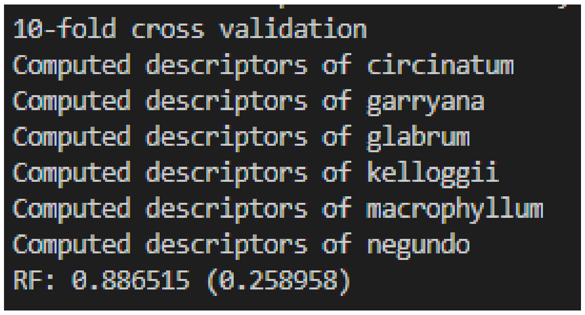
For better results I prepared 10-fold cross validataion for whole set of data.

The 10-fold algorithm divides the set into 10 parts and checks the results of machine learning for each set as the tested one and rest as the training group. Then mean of all 10 outputs it the final result. The image below shows the results for all descriptors.



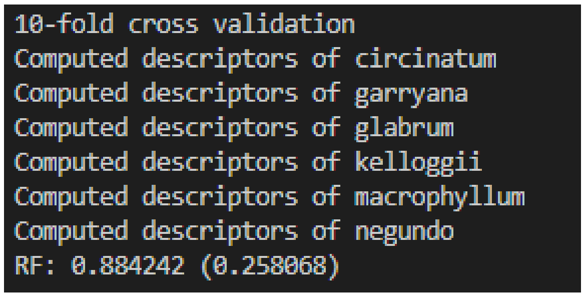
```
10-fold cross validation
Computed descriptors of circinatum
Computed descriptors of garryana
Computed descriptors of glabrum
Computed descriptors of kelloggii
Computed descriptors of macrophyllum
Computed descriptors of negundo
RF: 0.391919 (0.292805)
```

The mean is only 40%. However, for the 3 descriptors (Hu moments, Haralick Texture, Color Histogram) the cross-validation result has the highest mean which is equal to 89%.



```
10-fold cross validation
Computed descriptors of circinatum
Computed descriptors of garryana
Computed descriptors of glabrum
Computed descriptors of kelloggii
Computed descriptors of macrophyllum
Computed descriptors of negundo
RF: 0.886515 (0.258958)
```

The most interesting fact is that if I had used only Color Histogram descriptor the results were close to the highest result.



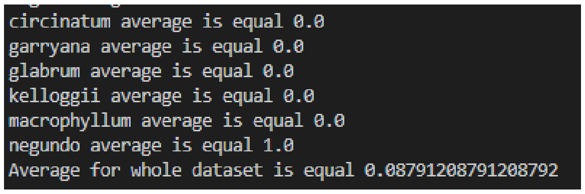
```
10-fold cross validation
Computed descriptors of circinatum
Computed descriptors of garryana
Computed descriptors of glabrum
Computed descriptors of kelloggii
Computed descriptors of macrophyllum
Computed descriptors of negundo
RF: 0.884242 (0.258068)
```

So after adding 3 global and 2 local descriptors the result increases by only 0,2%. The 5 remaining descriptors have result between 40% and 70%. The results are around 90% of accuracy as the mean of 10-fold cross-validation.

3.2 80% training / 20% testing

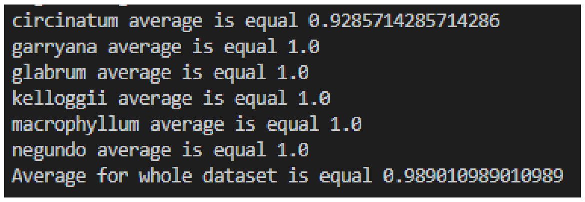
In the project description is the statement that the whole data set should be divided into two parts. First with 80% of the images as the training and rest 20% as the test set. Using ”separeteImages()” function which is described in the 2.3 section I divide the data and perform the test.

For the results, I prepared ”save_and_summary(results)” function which is described in the 2.5 section. All the results are saved as text files. Moreover, each species has the mean results and each prediction is saved also. In the report, I show only the summary of the few sets of descriptors. Firstly the result for all descriptors.



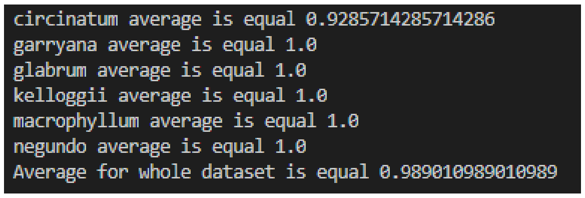
```
circinatum average is equal 0.0
garryana average is equal 0.0
glabrum average is equal 0.0
kelloggii average is equal 0.0
macrophyllum average is equal 0.0
negundo average is equal 1.0
Average for whole dataset is equal 0.08791208791208792
```

The result is low - only 9%. For the negundo kind, the result is equal to 100% but this is the only class without 0% as a result. However below is shown the result of using only 3 descriptors (Hu moments, Haralick Texture, Color Histogram).



```
circinatum average is equal 0.9285714285714286
garryana average is equal 1.0
glabrum average is equal 1.0
kelloggii average is equal 1.0
macrophyllum average is equal 1.0
negundo average is equal 1.0
Average for whole dataset is equal 0.989010989010989
```

For the 3 descriptors, the results are much better - around 99%. And only for the circinatum kind of tree the results are not 100% correct. The result belew looks like 2nd time the same image but this one is different - only Color Histogram was used to obtain the result.



```
circinatum average is equal 0.9285714285714286
garryana average is equal 1.0
glabrum average is equal 1.0
kelloggii average is equal 1.0
macrophyllum average is equal 1.0
negundo average is equal 1.0
Average for whole dataset is equal 0.989010989010989
```

So in the case, the 5 descriptors (3 globals and 2 locals) do not affect the best result. The machine learning model correctly interprets the same number of images using only Color Histogram, the same number using the 3 descriptors and obtains better result then all set of descriptors. The 10-fold cross-validation shows the small difference but for the 80% training / 20% testing the results are the same for comparision of 1 and 3 descriports.

4 Source Code

4.1 descriptors.py

```python
import cv2
import mahotas
import numpy as np
from skimage import feature

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

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

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

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

# Local Binary Pattern
def fd_binary_pattern(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    lbp = feature.local_binary_pattern(gray, 24, 8, method="uniform"
   )
    (hist, _) = np.histogram(lbp.ravel(),
        bins=np.arange(0, 27),
        range=(0, 26))
    hist = hist.astype("float")
    hist /= (hist.sum() + 1e-7)
    return hist

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

# SIFT
def fd_SIFT(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    freakExtractor = cv2.xfeatures2d.FREAK_create()
    keypoints,descriptors= freakExtractor.compute(gray,None)
    return keypoints

# SIFT v2
def fd_SIFT_v2(image):
    gray= cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
    sift = cv2.xfeatures2d.SIFT_create()
    kp = sift.detect(gray,None)
    image=cv2.drawKeypoints(gray,kp,image)
    cv2.show('sift_keypoints.jpg',image)
    return kp

# for tests
if __name__ == "__main__":
    image = cv2.imread('.\isolated/negundo\l20.jpg')
    print(fd_histog(image))
```

4.2 summary.py

```
import os

# list of all spiecies
class_names = ["circinatum", "garryana", "glabrum", "kelloggii", "
    macrophyllum","negundo"]

# function for save results and summary
def save_and_summary( results ):
    # create files to save results
    if not os.path.exists("./train&test"):
        os.makedirs("./train&test")
    AllResults = open("./train&test/AllResults.txt" , 'w')
    Summary = open("./train&test/Summary.txt" , 'w')
    # create 2 dictionaries for computing correct preditions
    num = dict()
    den = dict()
    for name in class_names:
        num[name] = 0
        den[name] = 0
    for result in results:
        # write all preditions to AllResults file
        AllResults.write(' -> '.join(result)+'\n')
        # compute correctness for each spiecie
        den[result[0]] += 1
        if result[0] == result[1]:
            num[result[0]] += 1
```

```python
    n = 0
    d = 0
    # Summary
    Summary.write("Summary\n")
    for name in class_names:
        # write results for each class
        if den[name] != 0:
            line = name+" average is equal "+ str(num[name]/den[name
    ])
            Summary.write(line+"\n")
            print(line)
        n += num[name]
        d += den[name]
    if d != 0:
        # write result for whole data set
        line = "Average for whole dataset is equal "+ str(n/d)
        Summary.write(line+"\n")
        print(line)
    # close files
    AllResults.close()
    Summary.close()

if __name__ == "__main__":
    results = list()
    save_and_summary(results)
```

4.3 separate_images.py

```python
import os
import glob
import shutil

class_names = ["circinatum", "garryana", "glabrum", "kelloggii", "
    macrophyllum","negundo"]
directory = "train&test"

def separeteImages():
    # create new empty directories
    if os.path.exists(directory):
        shutil.rmtree(directory, ignore_errors=True)
    os.makedirs(directory)
    # for every spiece
    for image in class_names:
        # prepare two directories
        # one for train images
        train_dir = directory+"/"+image
        os.mkdir(train_dir)
        train_dir += "/train"
        os.mkdir(train_dir)
        # one for test images
        test_dir = directory+"/"+image+"/test"
        os.mkdir(test_dir)
        # for every image in data set
        i=0
        for jpgfile in glob.glob("isolated/"+image+"/*.jpg"):
            if(i%5==0):
                shutil.copy(jpgfile, test_dir)
            else:
                shutil.copy(jpgfile, train_dir)
            i+=1

if __name__ == "__main__":
    separeteImages()
```

4.4 train_test.py

```
import glob
import cv2
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import KFold, cross_val_score
from descriptors import fd_hu_moments
from descriptors import fd_haralick
from descriptors import fd_histogram
from descriptors import fd_histog
from descriptors import fd_binary_pattern
from descriptors import fd_SIFT
from descriptors import fd_orb
from summary import save_and_summary

class_names = ["circinatum", "garryana", "glabrum", "kelloggii", "
    macrophyllum","negundo"]

all_descriptors = []
labels          = []
# learning from the train directories
for label in class_names:
    for file in glob.glob('train&test/'+(label)+'/train/*.jpg'):
        # read the image and resize it to the common size
        image = cv2.imread(file)
        image = cv2.resize(image, (512,512))
        # compute descriptors
        fv_hu_moments = fd_hu_moments(image)
        fv_haralick   = fd_haralick(image)
        fv_histogram  = fd_histogram(image)
        fv_hog        = fd_histog(image)
        fv_binpat     = fd_binary_pattern(image)
        fv_orb        = fd_orb(image)
        # fv_SIFT        = fd_SIFT(image)
        # save them in descriptors variable
        descriptors = np.hstack([fv_histogram, fv_haralick,
    fv_hu_moments, fv_hog, fv_binpat, fv_orb])
        # save labels and feature as the vectors
        labels.append(label)
        all_descriptors.append(descriptors)
    print("Computed descriptors of "+label)

# Random Forests Classifier
clf  = RandomForestClassifier(n_estimators=100, random_state=9)
# learning the model
```

```
clf.fit(all_descriptors, labels)

results = list()
print("\nPredictions\n")
for label in class_names:
    for file in glob.glob('train&test/'+(label)+'/test/*.jpg'):
        # read the image and resize it to the common size
        image = cv2.imread(file)
        image = cv2.resize(image, (512,512))
        # compute descriptors
        fv_hu_moments = fd_hu_moments(image)
        fv_haralick   = fd_haralick(image)
        fv_histogram  = fd_histogram(image)
        fd_hog        = fd_histog(image)
        fd_binpat     = fd_binary_pattern(image)
        fv_orb        = fd_orb(image)
        # fv_SIFT       = fd_SIFT(image)
        # save them in descriptors variable
        descriptors = np.hstack([fv_histogram, fv_haralick,
    fv_hu_moments, fv_hog, fv_binpat, fv_orb])
        prediction = clf.predict(descriptors.reshape(1,-1))[0]
        print(label,prediction)
        results.append((label,prediction))

save_and_summary(results)
```

4.5 cross_validation.py

```
from descriptors import fd_hu_moments
from descriptors import fd_haralick
from descriptors import fd_histogram
from descriptors import fd_histog
from descriptors import fd_binary_pattern
from descriptors import fd_SIFT
from descriptors import fd_orb
import glob
import cv2
import numpy as np
from sklearn.model_selection import train_test_split,
    cross_val_score
from sklearn.model_selection import KFold
from sklearn.ensemble import RandomForestClassifier

# 10-fold cross validation
def cross_validation():
    print("10-fold cross validation")
    class_names = ["circinatum", "garryana", "glabrum", "kelloggii",
    "macrophyllum","negundo"]
    all_descriptors = []
    labels          = []
    for label in class_names:
        for jpgfile in glob.glob("isolated/"+label+"/*.jpg"):
            image = cv2.imread(jpgfile)
            image = cv2.resize(image, (512,512))
            # compute descriptors
```

```python
            fv_hu_moments = fd_hu_moments(image)
            fv_haralick   = fd_haralick(image)
            fv_histogram  = fd_histogram(image)
            fv_hog        = fd_histog(image)
            fv_binpat     = fd_binary_pattern(image)
            fv_orb        = fd_orb(image)
            fv_SIFT       = fd_SIFT(image)
            # save them in descriptors variable
            descriptors = np.hstack([fv_histogram , fv_haralick, fv_hu_moments , fv_hog , fv_binpat, fv_orb])
            # save labels and feature as the vectors
            labels.append(label)
            all_descriptors.append(descriptors)
        print("Computed descriptors of "+ label)

    # # 10-fold cross validation
    kfold = KFold(n_splits=10)
    rf = RandomForestClassifier(n_estimators=100, random_state=9)
    cv_results = cross_val_score(rf, all_descriptors, labels, cv=kfold, scoring="accuracy")
    msg = "%s: %f (%f)" % ('RF', cv_results.mean(), cv_results.std())
    print(msg)

if __name__ == "__main__":
    cross_validation()
```

5 References

1. https://en.wikipedia.org/wiki/Random_forest
2. https://gogul.dev/software/image-classification-python
3. https://www.learnopencv.com/histogram-of-oriented-gradients/
4. https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_orb/py_orb.html
5. https://en.wikipedia.org/wiki/Image_moment
6. http://wiki.awf.forst.uni-goettingen.de/wiki/index.php/Haralick_Texture
7. https://software.intel.com/en-us/ipp-dev-reference-histogram-of-oriented-\gradients-hog-descriptor
8. https://en.wikipedia.org/wiki/Color_histogram
9. https://software.intel.com/en-us/ipp-dev-reference-histogram-of-oriented-\gradients-hog-descriptor
10. https://www.pyimagesearch.com/2015/12/07/local-binary-patterns-with-python-\opencv/