Introduction to image processing and computer vision

Plant Species Recognition

Laboratory Project 2

Patryk Walczak

January 18, 2020

Contents

1 Introduction **1**
1.1 Project Description 2

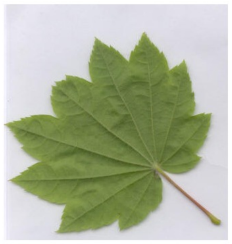
2 Classification **2**
2.1 Feature Descriptors 2
2.1.1 Hu Moments 2
2.1.2 Haralick Texture 3
2.1.3 Color Histogram 3
2.1.4 Histogram of Oriented Gradients 3
2.1.5 Unused 4
2.2 Cross validation 4
2.3 Division of data set 5
2.4 Train and test 6
2.5 Summary 6

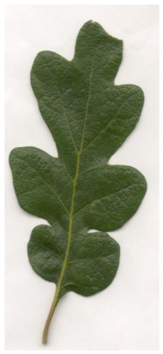
3 Results **7**

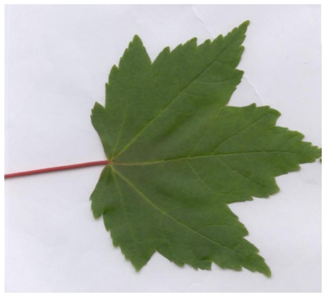
4 References **7**

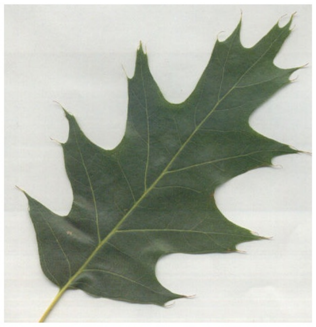
1 Introduction

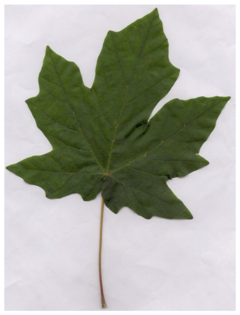
Main task of the project is elaboration of discriminative feature vector for the leaves of trees. Using computer vision processing find the vectors then learn the machine learning model with 80% of the data set and test the results on remaning part. Six tree species each in separate dictionary are stored 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

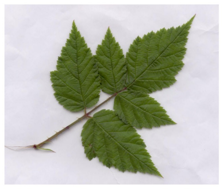

(f) Acer Negundo
Boxelder

Figure 1: Pictures of leaves

1.1 Project Description

For classification I chose random forest method, which is ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees. All images are analized by 3 descriptors, all of them are used to learn the algorithm and after that to recognise images from test set. For all purpose I prepared 4 python files.

List of files :

- descriptors.py - contains all descriptiors funtions
- summary.py - uses to save all results in files
- separateImages.py - copy files to "train&test" directory and separate them in train and test sets
- train&test.py - main file contains all train and test functions

2 Classification

In the section is described process of classifiaction leaves, using the descriptors and training and testing model.

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. The feature descriptor is in first funtion in "descriptors.py". For 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.

```python
def separeteImages():
# 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 a input only image, change the image's colors to black and white and compute Hu Moments. At the end it flattens the results to one array.

2.1.2 Haralick Texture

Texture defines the consistency of patterns and colors in a image. Haralick suggested the use of gray level co-occurrence matrices (GLCM).The basic idea of GLCM - it looks 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 second funtion in "descriptors.py".

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

The funciton is simple, takes only image as input, and return 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 convert to the vector is result of the third featre 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 then the previous two. As the two, the descripor for computation needs only image. Then even no changes of colors are required. Using the OpemCV library and the calcHist method the function compute histogram. After that, the historgram 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 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 descriptor functions, the one takes only image as input also. Then changing of the colors to black and white occures. The feature library helps with computing of the histogram of the oriented gradients. And result of the hog method is returned.

2.1.5 Unused

I prepared Oriented Fast and Rotated BRIEF descripor also but the methon SIFT_create() from cv2.xfeatures2d returned the comunicat "This algorithm is patented and is excluded in this configuration". So that I decided not to use the descriptor in the project.

```
def fd_orb(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)
```

2.2 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))
```

Every image in data set is loaded and resize to the same dimensions.

```
            fv_hu_moments = fd_hu_moments(image)
            fv_haralick   = fd_haralick(image)
            fv_histogram  = fd_histogram(image)
            fd_hog        = fd_histog(image)
            # save them in descriptors variable
            descriptors = np.hstack([fv_histogram, fv_haralick,
  fv_hu_moments, fd_hog])
```

The stack of descriptors is saved in all_descriptors list and spiece is saved in labels list.

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

Then from library "sklearn.model_selection" take KFold function and prepare algorithm for 10-fold cross validation. After that function create ML model - Random Forest Classifier. At the end the function computes the cross validataion and printf mean and standard deviation of the ten results fo 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 splited in two parts. The first with 80% of all images is used to train the method. The second with remaining 20% is used for test of the method.

In ”separete_images.py” I prepared function for split the data set.

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

def separeteImages():
```

Firstly, the program create new empty direcories 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 spiece create test and train direcotries.

```
  # 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, program start spliting 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 modulo to 5 is saved in test directory, and the rest are stored in train directory. For comparision of the descriptors I preformed the code only once at start of the project. The fact provides that for all results algorith always has the same input data.

2.4 Train and test

The section covers the most important part of the project. Here is desciption 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 start program reads all imagines in train directories from all spieces and resize all of them to common dimensions.

```
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 spiece name of the image is stored in labels list.

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

When the traning set of data completed, 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, algorithm start testing section by reads all images and resize 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 the 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)
        # save them in descriptors variable
        descriptors = np.hstack([fv_histogram, fv_haralick,
   fv_hu_moments, fd_hog])
```

Next step is the prediction and checking the results. Name of the spiece is printed out with the predited spiece. User can easly and in real-time check the results.

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

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

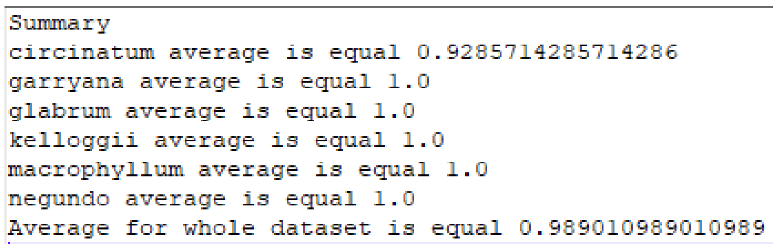
```
save_and_summary(results)
```

2.5 Summary

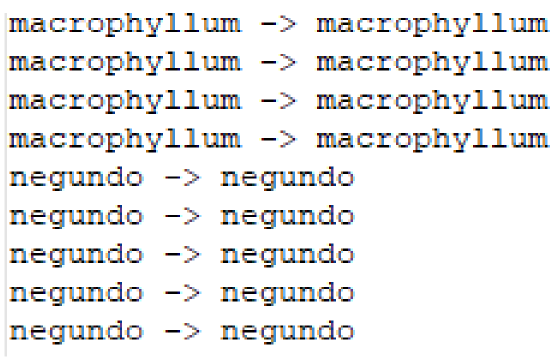
For saving results for each image, each spiece and for all data set I prepared summary.py file.

At first, the program create empty files in directory "test&file". AllResults.txt file includes all results in form "spiece"-¿"predition", and Summary.txt file includes 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 correct answer at the 1st postion and predition of the model at the 2nd position in each tuple.

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

Then create two dictionaries and save the results to AllResults.txt. In the same loop compute 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 next step program starts summary. Firstly, compute result for each spiece.

```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 close the files.

3 Results

4 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
8. https://en.wikipedia.org/wiki/Color_histogram
9. https://software.intel.com/en-us/ipp-dev-reference-histogram-of-oriented-gradients
10. https://software.intel.com/en-us/ipp-dev-reference-histogram-of-oriented-gradients