INTRODUCTION TO IMAGE PROCESSING AND COMPUTER VISION

LABORATORY PROJECT 2

Author Elif Ozdemir

Instructor Rafal Jozwiak

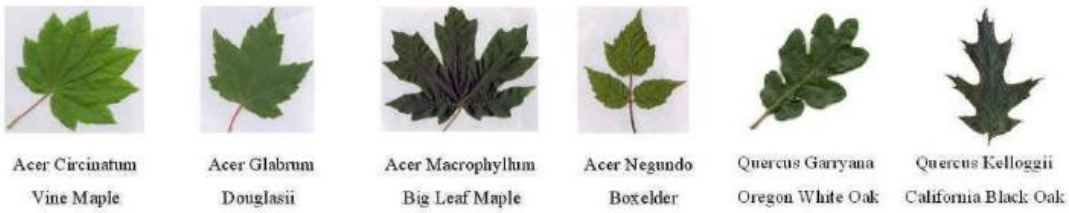
Contents

31.01.2020

Introduction

In this project the goal is getting familiar with classification and machine learning. The following samples of leaves are given. The task is to train a classifier and test it accordingly. The most important part is deciding on features that determine the success of a classification.

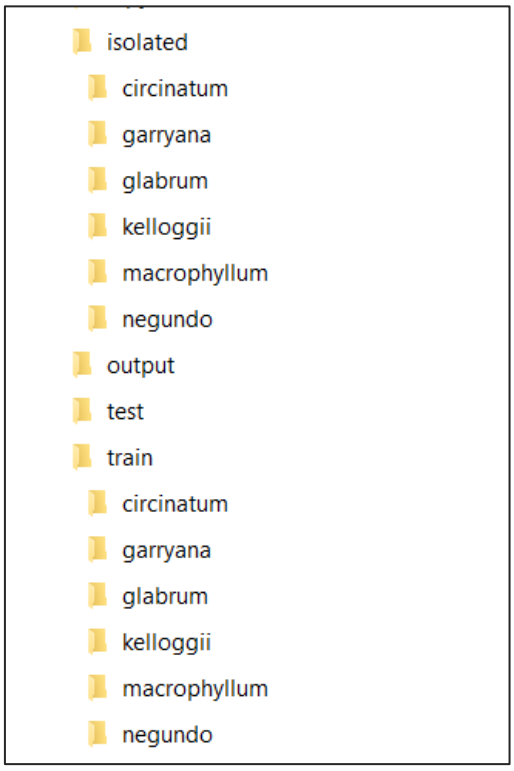
- input: samples of different plant species (6 classes, from 38 up to 97 samples per class)



Description of the Approach

Arrangement of folders

First of all, I organize the folders. My aim is to have two directories named “train” and “test” that have directories of each class with the images of the leaves.



So, in train folder I have separate folders for each class of the leaves. I place the 80% of the images in specific folder for that class of the leaf. And the rest which is 20% I put into test folder.

```
# creating data sets for training and testing
def create_data_sets():
    directory = "isolated"

    pic_count = []  # count of pictures in each folder
    for subdir, dirs, files in os.walk(directory):
        pic_count.append(int(len(files)))

    folder = 0
    testi = 0
    for subdir, dirs, files in os.walk(directory):
        # variables for iteration and naming
        pici = 0
        traini = 0


        for file in os.listdir(subdir):
            if folder == 0:
                pass
            # 80% of the data (images) goes to training
            elif pici % 5 != 0:
                shutil.copyfile(subdir + "\\" + file.title(),
                                'train\\' + class_names[folder - 1] + "\\" +
str(traini) + ".Jpg")
                traini = traini + 1
            # 20% of the data goes to test directory
            else:
                shutil.copyfile(subdir + "\\" + file.title(),
                                'test\\' + str(testi) + ".Jpg")
                testi = testi + 1

            pici = pici + 1
        folder = folder + 1
```

The rest of the code for organization is in env.py

Selection of features

Global features

The next step is choosing the features that will give the best results.

I start with the count of corners. I don't expect good results for this feature since some of the leaves have very similar count of corners despite the fact that they are from different species. However, I test this feature to have an idea how corners are defined for each leaf.

K-fold Cross Validation (more words about it in the following pages) value is below

RF: 0.321069

Test success level: 0.24175824175824176

```
# global feature corner count
def get_corners(image):
    # convert image to gray scale image
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # detect corners with the goodFeaturesToTrack function
    feature_params = dict(maxCorners=100, qualityLevel=0.6, minDistance=7,
blockSize=7)
    corners = cv2.goodFeaturesToTrack(gray, mask=None, **feature_params)
    corners = np.int0(corners)
    return len(corners)
```

The Hu Moments is 2nd of the global features that we analyze. It gives the data about the shape of the leaves. Since leaf classes have very specific shapes, this feature is helpful. However, when we train just by `cv2.moments()` the testing results are a bit low. The K-Fold Cross Validation is 0.392333

Before computing moments we convert an image to grayscale. After I have performed thresholding I have the silhouette of the object in the image.

```
# global feature Hu Moments
def fd_hu_moments(image):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    feature = cv2.HuMoments(cv2.moments(image)).flatten()
    cv2.waitKey(1000)
    cv2.destroyAllWindows()
    return feature
```

The third feature is color histogram. I use the following function for this purpose.

`cv2.calcHist(images, channels, mask, histSize, ranges[, hist[, accumulate]])`

The arguments I give are the image, channels, mask, histSize (bins) and ranges for each channel [typically 0-256).

Just Color Histogram 10-fold Cross validation results:

RF: 0.971333

```
# global feature Color Histogram
def fd_histogram(image, mask=None):
    # convert the image to HSV color-space
    image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    # compute the color histogram
    hist = cv2.calcHist([image], [0, 1, 2], None, [bins, bins, bins], [0, 256, 0,
256, 0, 256])
    # normalize the histogram
    cv2.normalize(hist, hist)
    return hist.flatten()
```

The next global feature is the texture. I use `mahotas.features.haralick()` function from mahotas library. As previously before using it I convert the images into grayscale.

The result for training with texture feature only:

RF: 0.762833

```
# global feature Haralick Texture
def fd_haralick(image):
    # convert the image to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # compute the haralick texture feature vector
    haralick = mahotas.features.haralick(gray).mean(axis=0)
    return haralick
```

In the next part you can find the results for different combinations of features.

Hu moments and haralick texture: 0.701667

Hu_moments and color histogram (the best in double combinations): 0.979667

Haralick texture and histogram: 0.975500

My preference would be going with all of them (hu_moments, haralick, histogram, corners)

The 10-folds Cross validation: 0.984274

Result on the unseen data: 0.68

Local Features

In this part of my analysis I decided to have a look at SIFT (Scale-invariant feature transform). It is the widest local feature descriptor algorithm.

What is SIFT all about shortly:

SIFT keypoints of objects are first extracted from a set of reference images and stored in a database. An object is recognized in a new image by individually comparing each feature from the new image to this database and finding candidate matching features based on Euclidean distance of their feature vectors. ~Wikipedia

The main steps of SIFT are

1. Scale-space extrema detection
2. Keypoint localization
    - Interpolation of nearby data for accurate position
    - Discarding low-contrast keypoints
    - Eliminating edge responses
3. Orientation assignment
4. Keypoint descriptor

To be able to use `cv2.xfeatures2d.SIFT_create()` method of cv2 library I needed the following versions of opencv-python and opencv-contrib-python. It is connected to the fact that the desired feature is nonfree in the following versions. And it is not so in other cases.

```
pip install opencv-python==3.4.2.16
pip install opencv-contrib-python==3.4.2.16
```

```
def gen_sift_features(img):
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    sift = cv2.xfeatures2d.SIFT_create()
    # kp is the keypoint
    # desc is the SIFT descriptors, they're 128-dimensional vectors
    # that we can use for our final features
    kp, desc = sift.detectAndCompute(gray, None)
    return kp, desc
```

To be able to use this feature I needed to resize the arrays since they were diverse.

```
kp, desc=gen_sift_features(image)
resized_desc=np.resize(desc, (1,200))[0]
```

The computed 10-fold value is 0.403629.

The success rate in the test dataset is 0.38

Machine Learning

In training.py I basically prepare data to train my machine learning model which is Random Forest Classifier.

In testing.py I split the data properly with train_test_split() method. Which does the following

Used to split our data into random train and test subsets of it. The first two parameters are simply data we obtained and then

• test_size – a float between 0 and 1 representing the proportion of the dataset to include in the test split.

• random_state – seed used by the random number generator

```
# split the training and testing data
(trainDataGlobal, testDataGlobal, trainLabelsGlobal, testLabelsGlobal) = \
    train_test_split(np.array(global_features),
                     np.array(global_labels),
                     test_size=test_size,
                     random_state=seed)
```

Next, I create my model and train it.

```
# create the model - Random Forests
clf = RandomForestClassifier(n_estimators=num_trees, random_state=seed)

# fit the training data to the model
clf.fit(trainDataGlobal, trainLabelsGlobal)
```

The rest of the code is in testing.py.

Why did I use K-fold Cross Validation?

Other than it being in the task description it's a commonly used tool to evaluate the success of a machine learning model. The main reason of its popularity is that it's less biased or less optimistic.

The general procedure is as follows:

1. Shuffle the dataset randomly.
2. Split the dataset into k groups
3. For each unique group:

1. Take the group as a hold out or test data set
2. Take the remaining groups as a training data set
3. Fit a model on the training set and evaluate it on the test set
4. Retain the evaluation score and discard the model

4. Summarize the skill of the model using the sample of model evaluation scores

K is 10 in my case. It is one of the common choices. I split the entire data into 9 parts for training and 1 part for testing uniquely over each round upto 10 times.

The following snippet of the code is representation of the implementation

```
# 10-fold cross validation
kfold = KFold(n_splits=10, random_state=seed)
cv_results = cross_val_score(model, trainDataGlobal, trainLabelsGlobal, cv=kfold,
scoring=scoring)

total = "%s: %f " % ("RF", cv_results.mean())
print(total)
```

Data save format

I use h5py to save my features and labels locally in .h5 file format.

Conclusion

Classification task in image processing is one of the most vital ones. It gives an opportunity of solving problems that are nearly impossible to solve by human beings in short amount of time.

In my project I used different types of features. Local and global ones. I tested them by different techniques. The results are rather low. However, a few methods of approach were implemented and researched. There is a lot to improve definitely.

References and used materials

1. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
2. https://gogul.dev/software/image-classification-python?fbclid=IwAR1_PHT3XavjmtLZk4Rvo_5UnsC8XeEPCU16EmpLVTtMXZnQCxnCmUf1H1c
3. https://docs.opencv.org/2.4/modules/imgproc/doc/histograms.html
4. https://www.pyimagesearch.com/2014/10/27/opencv-shape-descriptor-hu-moments-example/
5. https://machinelearningmastery.com/k-fold-cross-validation/
6. https://ianlondon.github.io/blog/how-to-sift-opencv/
7. https://en.wikipedia.org/wiki/Scale-invariant_feature_transform