**Introduction to Image Processing and Computer**

**Vision**

Filip Zawadka

Project 2:

PLANT SPECIES RECOGNITION

1. Introduction
2. Step by step algorithm description and analyse
3. Code
4. Bibliography

1.Introduction

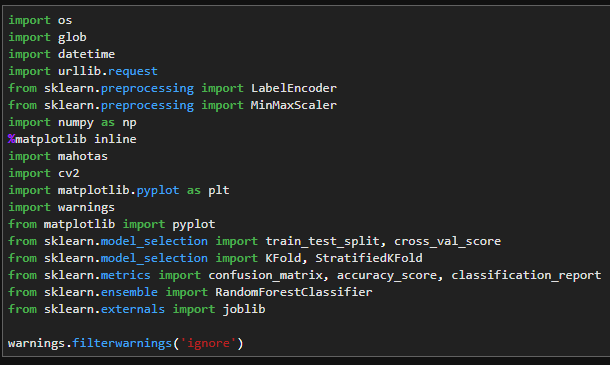
Our goal in in this exercise is to classify 6 plant species using images of their leaves.

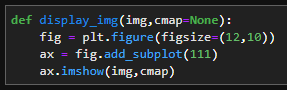
Images are split basing on their classes so we will use name of the folder which there are in as a true label extracting it from the path. Then using our algorithm we will try to predict one and then compare both strings, if there are equal, that means a success. To test the success rate we will need a testing set which will be 20% of the given images.

For classification we will use Random Forrest Classifier from sklearn library, as I found out that it is the best for such data. It works by building multiple decision trees and after analysing, it searches for the most common data, in our case most common features for a given tree. After that we check the accuracy of the trained model and then test it on our testing set. We repeat steps for different features to check which ones work the best.

2 .Step by step algorithm description

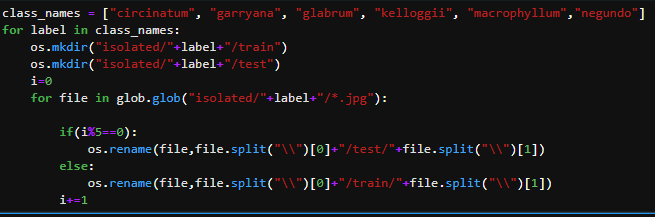
We start with importing all of the necessary libraries and write the function for displaying the images.





Then we need to segregate the images in the “isolated” folder.

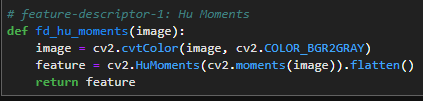
Using the os library we can create a train and test directory in each of the subfolders of “isolated”, which correspond to the name of the given plant.



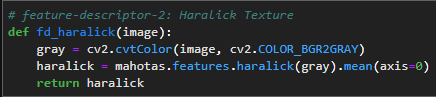
With help of glob library, we can go through every picture and then using os again, we can change the path to add it to corresponding folder. We add every 5-th image to the “test” folder and the rest to the “train” folder to achieve wanted 80%-20% ratio.

Next we define the feature-descriptor functions, which will return features of a given image.

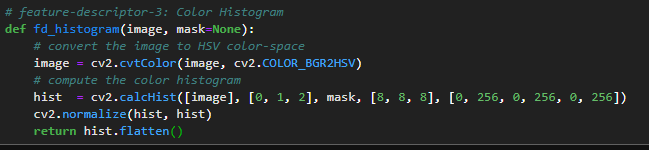
First one is Hu moments, which describes the shape of objects:



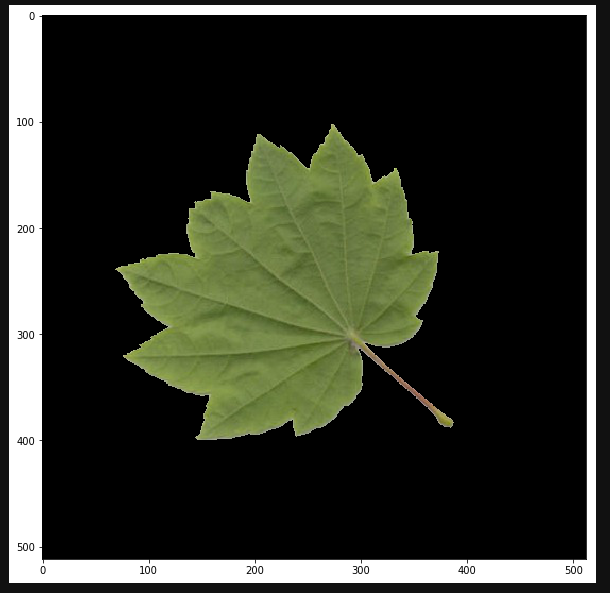
Next one describes the texture:



And the last one colours:

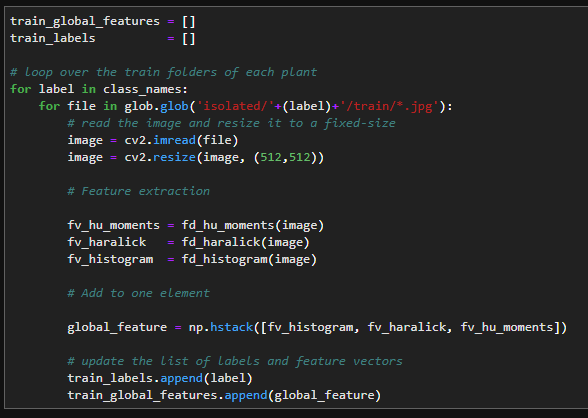


We could apply a simple mask using the fact that the background is blue:

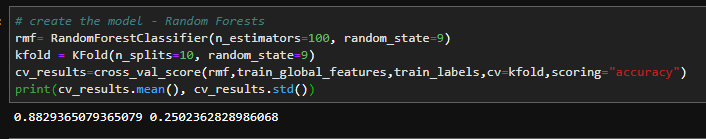


But it turned out that the mask worsens the resultants. It might be because the shade of the leave helps to classify it.

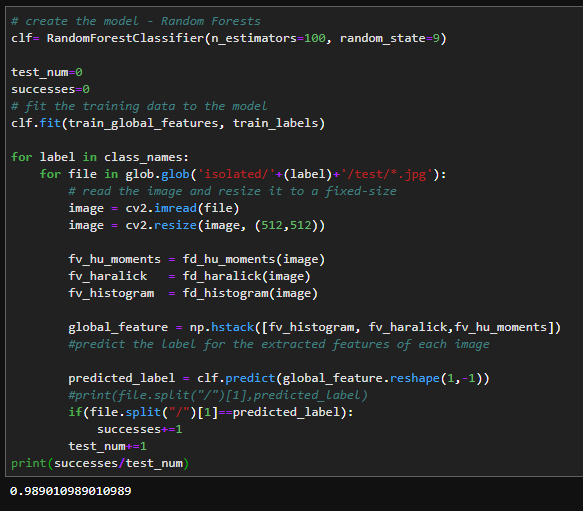
We need to extract features and add them to one vector:



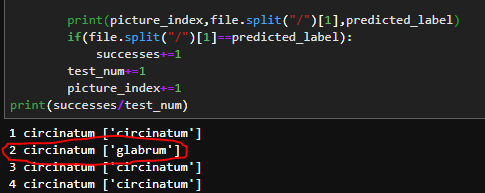
We test the model:



Then we train the model:



We can see that one test image was wrongly classified. After printing labels with predicted ones, we can check which one it was:



This is the one:

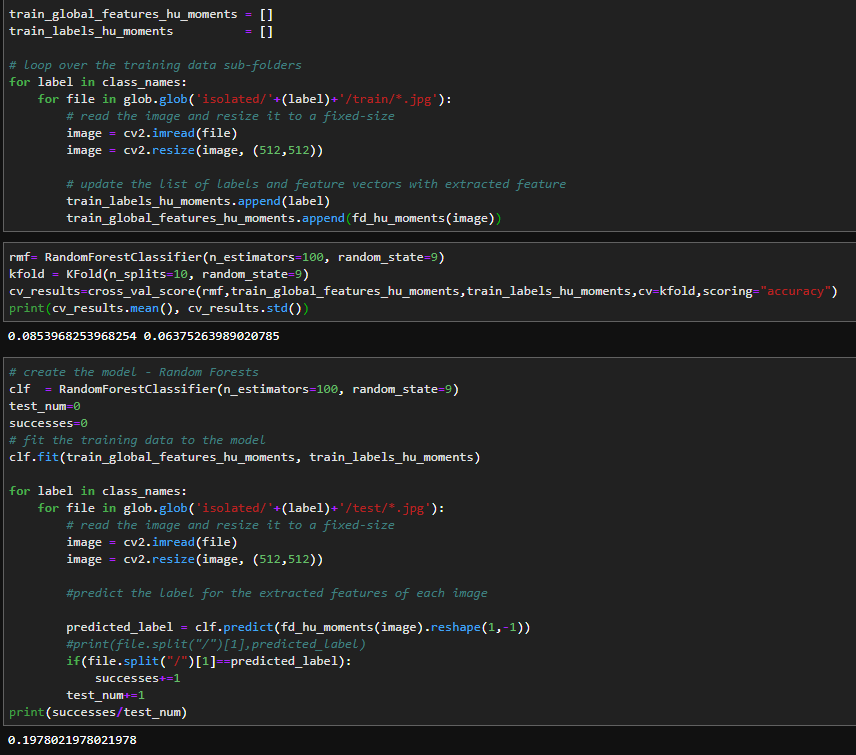


We can see that it’s very similar to glabrum:



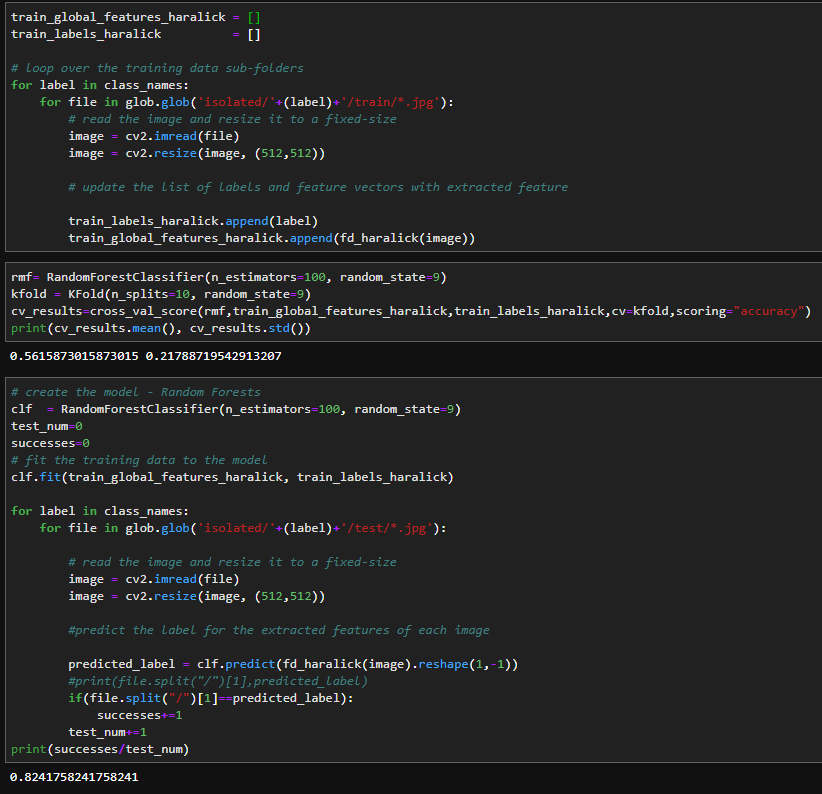
We can check which descriptor failed to be correct by training the model basing on each one of the features:

Shape



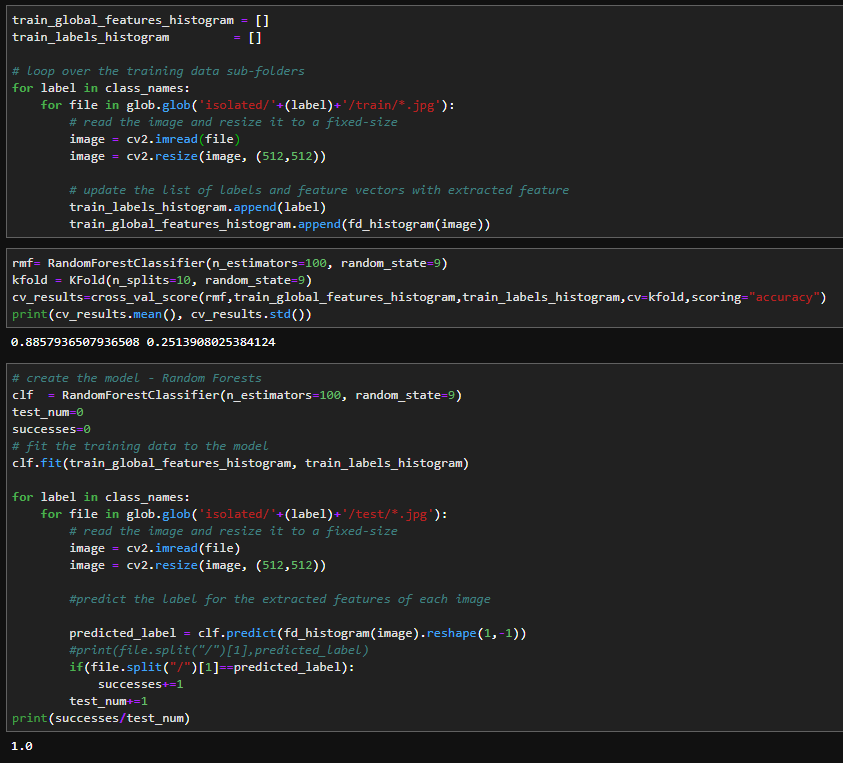
Shape has very low success rate.

Texture



Texture is better.

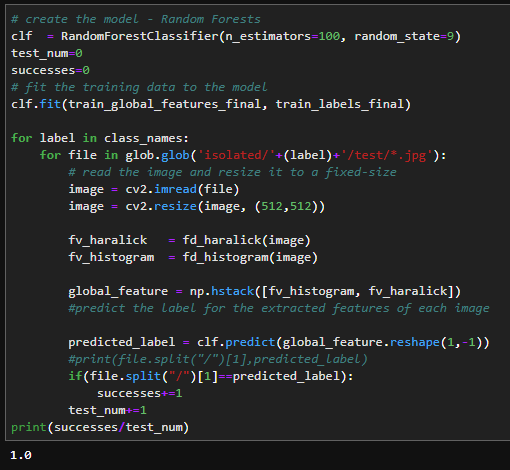
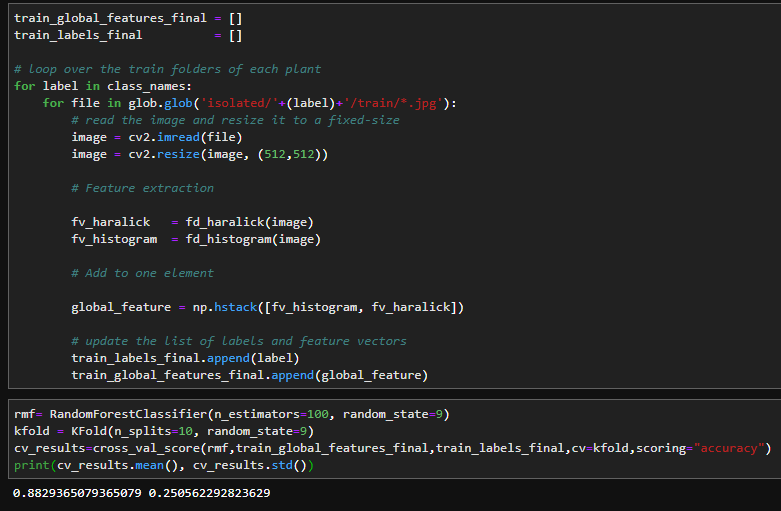
Colour



Colour is the best.

We can see that we could use only this feature to have 100% success rate on the test dataset.

For training my final model the best choice would be using both texture and colour, because it would help in a different, bigger dataset test pictures with different colours.



3.Code

import os

import glob

import datetime

import urllib.request

from sklearn.preprocessing import LabelEncoder

from sklearn.preprocessing import MinMaxScaler

import numpy as np

%matplotlib inline

import mahotas

import cv2

import matplotlib.pyplot as plt

import warnings

from matplotlib import pyplot

from sklearn.model\_selection import train\_test\_split, 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

from sklearn.externals import joblib

import math

warnings.filterwarnings('ignore')

def display\_img(img,cmap=None):

    fig = plt.figure(figsize=(12,10))

    ax = fig.add\_subplot(111)

    ax.imshow(img,cmap)

class\_names = ["circinatum", "garryana", "glabrum", "kelloggii", "macrophyllum","negundo"]

for label in class\_names:

    os.mkdir("isolated/"+label+"/train")

    os.mkdir("isolated/"+label+"/test")

    i=0

    for file in glob.glob("isolated/"+label+"/\*.jpg"):

        if(i%5==0):

            os.rename(file,file.split("\\")[0]+"/test/"+file.split("\\")[1])

        else:

            os.rename(file,file.split("\\")[0]+"/train/"+file.split("\\")[1])

        i+=1

# 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):

    # 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 the result

    return haralick

# feature-descriptor-3: 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], mask, [8, 8, 8], [0, 256, 0, 256, 0, 256])

    # normalize the histogram

    cv2.normalize(hist, hist)

    # return the histogram

    return hist.flatten()

train\_global\_features\_final = []

train\_labels\_final          = []

# loop over the train folders of each plant

for label in class\_names:

    for file in glob.glob('isolated/'+(label)+'/train/\*.jpg'):

        # read the image and resize it to a fixed-size

        image = cv2.imread(file)

        image = cv2.resize(image, (512,512))

        # Feature extraction

        fv\_haralick   = fd\_haralick(image)

        fv\_histogram  = fd\_histogram(image)

        # Add to one element

        global\_feature = np.hstack([fv\_histogram, fv\_haralick])

        # update the list of labels and feature vectors

        train\_labels\_final.append(label)

        train\_global\_features\_final.append(global\_feature)

rmf= RandomForestClassifier(n\_estimators=100, random\_state=9)

kfold = KFold(n\_splits=10, random\_state=9)

cv\_results=cross\_val\_score(rmf,train\_global\_features\_final,train\_labels\_final,cv=kfold,scoring="accuracy")

print(cv\_results.mean(), cv\_results.std())

# create the model - Random Forests

clf  = RandomForestClassifier(n\_estimators=100, random\_state=9)

test\_num=0

successes=0

# fit the training data to the model

clf.fit(train\_global\_features\_final, train\_labels\_final)

for label in class\_names:

    for file in glob.glob('isolated/'+(label)+'/test/\*.jpg'):

        # read the image and resize it to a fixed-size

        image = cv2.imread(file)

        image = cv2.resize(image, (512,512))

        fv\_haralick   = fd\_haralick(image)

        fv\_histogram  = fd\_histogram(image)

        global\_feature = np.hstack([fv\_histogram, fv\_haralick])

        #predict the label for the extracted features of each image

        predicted\_label = clf.predict(global\_feature.reshape(1,-1))

        #print(file.split("/")[1],predicted\_label)

        if(file.split("/")[1]==predicted\_label):

            successes+=1

        test\_num+=1

print(successes/test\_num)

4.Bibliography

I used jupyter lab to compile and this site was the most helpful:

<https://gogul.dev/software/image-classification-python>

But I also used:

<https://en.wikipedia.org/wiki/Random_forest#Algorithm>

<https://en.wikipedia.org/wiki/Overfitting>

<https://en.wikipedia.org/wiki/Mode_(statistics)>

<http://studentnet.cs.manchester.ac.uk/resources/library/thesis_abstracts/MSc14/FullText/Alwanin-Rawabi-fulltext.pdf>

<https://gist.github.com/jstadler/c47861f3d86c40b82d4c>

<https://medium.com/machine-learning-101/chapter-5-random-forest-classifier-56dc7425c3e1>