INTRODUCTION TO IMAGE PROCESSING AND COMPUTER VISION

**Project 2: Plant species recognition**

Szymon Majorek

[s.majorek@student.mini.pw.edu.pl](mailto:s.majorek@student.mini.pw.edu.pl)

Table of contents

1. Introduction
   1. General introduction
   2. Data set presentation
2. Part 1 – Feature extraction
   1. Short Introduction
   2. Features and potential drawbacks
   3. Algorithm
3. Part 2 – testing obtained data
   1. Introduction
   2. Cross validation
   3. 80-20 split
4. Part 3 – results
   1. Cross validation
   2. 80-20 split
5. Conclusions

**1. Introduction**

**1) General Introduction**

Machine learning is a study of algorithms and statistical models that computer systems use to perform a specific task without using explicit instruction, relying on patterns and inference instead. In other words, we can teach program to do something for us, without us giving explicit input for our decision. In our case we will be using Random forests model.

Image recognition on the other hand is the ability to identify and detect objects or features of digital images or videos. It is a method for capturing, processing, examining and sympathizing images. We can use machine learning for image recognition, as we will in this project.

The goal of this project is to use machine learning in order to recognize different species of plants based on images of their leaves. We aim to teach our program to recognize different plant species by teaching it how do their leaves look like.

**2) Data set presentation**

**Our data set is quite simple actually. We have 6 types of leaves with irregular number of images of them. The types are as follows*: Acer Circinatum, Acer Glabrum, Acer Macrophyllum, Acer Negundo, Quercus Garryana* and *Quercus Kelloggii*:

Figure 5 - Acer Negundo

Figure 6 - Acer Macrophyllum

Figure 4 - Quercus Kelloggii

Figure 1 - Acer Circinatum

Figure 3 - Quercus Garryana

Figure 2 - Acer Glabrum

When looking at those images it’s quite easy to say that they are all different species. For example the one on fig 3 is sort of compact but with rounded tips, which is different from fig. 4 which has pointy tips instead. In fact all of them have pointy tips apart from the 3rd one however they have some slight differences which are easy to notice a first glance. They even have different colours but that might just be a coincidence that I randomly chose those examples which actually have visibly different colours. Now that is all obvious by looking at the pictures, however in terms of our project, we need to think a little differently. We know what are leaves, we know how they look like and where to look for differences between leaves of different plants (shape, number of pointy ends, whether the ends are actually pointy or more rounded etc.). One noticeable thing that us, humans don’t usually look for when trying to distinguish different leaves is their colour because we tend to think of leaves as generally green (well with small variations depending on seasons).

**2. Feature extraction**

**1) Short introduction**

Features are the information or list of numbers that are extracted from an image. These are real-valued numbers. As mentioned in the introduction, we need to think about what we want to actually find, as we probably won’t be able to categorize a leaf by it’s number of veins or width of it’s stem. We could settle on just one feature however as we can see above some of the leaves are rather similar so it’s better to use more than one feature to try to find their classes.

**2) Features and potential drawbacks**

We choose to look at texture (haralick, local binary pattern), color (histogram), shape (moments). Even though we tend to think of leaves as just green, it might be worth it to use color histogram in order to categorize those leaves, after all those 6 picked example do actually vary in color slightly, so why not try to use it. Example:

Figure 8 - Quercus Garryana

Figure 7 - Acer Glabrum example

Here we can actually see that the difference in color is quite visible and obvious. Well maybe not the color itself but rather the shade, and the fact that Garryana leaf is much darker in tone. They are additionally quite different in texture, as the one on left appear to be much smoother judging by this image.

However, as usually, it might not be the best choice afterall. We can take a look at few examples that would steer us away from choosing this feature, as some classes, for example:

For example if we add this leaf to our consideration we see that it’s much more similar to Acer Glabrum than the Quercus Garryana leaf. Actually it’s very similar in both categories, colour, as well as shape. The texture also doesn’t appear to be vastly different, with the slight difference being the veins, in Circinatum leaf they appear to a little convex, whereas in Glabrum they’re more flat.

Figure 9 - Acer Circinatum example

But the problems may arise not only between different classes of leaves. They can just as well arise between leaves of theoretically the same class.



Figure 11 - Macrophyllum ex 2

Figure 10 - Macrophyllum ex 1

Those 2 leaves we see above are both from the same category – Acer Macrophyllum. The difference between them is extremely vivid and we can immediately pin point it to the fact that the leaf from figure 11 was picked up during fall, and not summer or spring. For us, humans, it’s fairly obvious. Thankfully for I’ve picked rather extreme case for this example as this takes place only once in this class so we can still expect fairly good results.

 These 3 images are all of the same class of leaf – Quercus Kelloggii. As we can clearly see all of 3 those example have different colour. It’s not even fair to say that it’s a slight difference. The colours are nothing like each other, one is blue, one is green and the last one is very dark red. The situation would be similar as before, however in this case the situation keeps repeating, so inside this class of leaves we have few classes of colours, hence we should expect our results when using colour histogram to be slightly worse compared to other classes of leaves.

Figure 13 - Kelloggii ex 2

Figure 14 - Kelloggii ex 3

Figure 12 - Kelloggii ex 1

Having considered colour, we can move onto the next feature we will be considering, namely texture. We’ve already briefly mentioned in when discussing colour, but I’d like to expand a little on that.

Image texture in image processing is a set of metrics calculated in image processing designed to quantify the perceived texture of an image. Image texture gives us information about the spatial arrangement of colour or intensities in an image or selected region of an image.

Figure 15 - Circinatum ex

Figure 16 - Macrophyllum

Going from figure 15 to 16 we can notice gradual change in the texture. Image presented on figure 15 Is clearly the one with the smoothest surface whereas figure 17 presents heavily wrinkled leaf. With each vein leaving visible mark on the leaf. Of course those example are rather cherry picked just to show that there are actual differences in texture and using this feature actually makes some sense. There’s one more feature that will be considered here – shape. We will be considering shape, however I’m not actually

Figure 17 - Negundo ex

that optimistic about it, I think that looking at shape may turn out to be not as good as using other features even though there’s a clear distinction between some of the classes of leaves. We can take a look at the following comparisons:

Here we have images of 2 different classes which look fairly similar. They both have big, solid body in sort of rectangular shape, with few parts sticking out. The difference being that in fig. 18 example the parts that are sticking out are much more round compared to figure 19 where they’re spiky. It could lead to some false positives during training and then later on testing the method.

Figure 19 - Kelloggii ex

Figure 18 - Garryana ex

If we were to look at just the shape feature then definitely these 2 examples – namely Circinatum from figure 20 as well as globrum (figure 21). I think that those two are much more similar than they are different, which in turn means that comparing them by their shape might not be the greatest idea. Having discussed features, we can actually proceed to the way we get them.

Figure 21 - Glabrum

Figure 20 - Circinatum

**3) Algoritgm**

Even though we have two .py scripts for extracting features they are basically the same and they only differ in the sense that one of them is meant for testing the whole data set using cross validation, whereas the other is meant for 80-20 split of our dataset (80-20 meaning that we take 80% of our data and provide it as training part, and then we test our results using the remaining 20%).

Apart from opening, closing, writing to and from files, the most important thing is obviously the extraction of features from given directory or file.

Figure 22 - training data set

for i, trainName in enumerate(trainLabels):

    # path

    dir = os.path.join(trainPath, trainName)

    # loop over the images in each sub-folder

    for file in os.listdir(dir):

        globalFeature = getGlobalFeatures(dir+'/'+file)

        # add results

        labels.append(trainName)

        globalFeatures.append(globalFeature)

    print("{} class processed".format(trainName))

This is the core behind features extraction in our global\_features\*.py files. We iterate over all our train labels in some directory and then over all the files in that directory. In this part of the code there’s one function which is probably the most important one in the whole code. The getGlobalFeatures. This is a function defined in helper.py file. Along with that there are few more important ones, namely:

* *fu\_hu\_moments(image)* -> responsible for shape
* *fd\_haralick(image)* -> responsible for texture
* *fd\_histogram(image)* -> responsible for colour
* *fd\_localBinaryPatterns(image, numPoints = 24, radius = 8)* -> responsible for texture

When acquiring results, we can simply comment out whatever’s not the point of their interest as well as remove it from np.hstack method right before returning final result.

After our feature vectors are calculated, we can save them to some file. Since we’re dealing with datasets I used .h5 file which seem to be handling datasets very well, and additionally they’re pretty easy to deal with.

In fact the algorithms for 80-20 split and whole dataset are identical, they only differ by paths that we use in order to get to the images, and eventually output, depending on what we want to actually do.

**3. Testing obtained data**

**1) Introduction**

Now that we have our features extracted and saved to files we can proceed to training a model and then validating said model, or just generally checking it’s success rate for some test sample.

The whole testing process occurs in train\_test.py. First we declare all the paths, arrays, etc. so as to make our lives easier, and then we proceed:

**2) Cross validation**

Funnily the whole train and then test only takes up very few lines of code.

Figure 23 - KFold testing

model = RandomForestClassifier(n\_estimators=num\_trees, random\_state=seed)

(trainDataGlobal, testDataGlobal, trainLabelsGlobal, testLabelsGlobal) = train\_test\_split(np.array(global\_features),

               np.array(

               global\_labels),

test\_size=test\_size,

random\_state=seed)

# 10-fold cross validation

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

cvScore = cross\_val\_score(

    model, trainDataGlobal, trainLabelsGlobal, cv=kfold, scoring=scoring)

In this case, we again want to not only print the results to the terminal, but to also store them somewhere on the actual disk. Again I chose .h5 file for this purpose.

About the code itself, we split our total data (trainLabelsGlobal & trytestLablesGlobals) into trainData and testData based on test\_size and random\_state which were initialized earlier. Then we use KFold() and cross\_ val\_score() functions to obtained our score for given method. After that we yet again use .h5 file to save our results.

**3) 80-20 split**

Situation in this case is slightly different due to the fact that we can’t pick images at random to be the ones that are being tested against learned data.

What’s very important for this part of the train\_test is that

clf = RandomForestClassifier(n\_estimators=num\_trees, random\_state=seed)

global\_labels\_split.sort()

# fitting data

clf.fit(global\_features\_split, global\_labels\_split)

When we fit our data we use whole sets of data instead of just some part that we calculated.

Here we also iterate over all the images and we try to predict the resulting image in every single input. Then we take that result, binarize it and once again save it to some .h5 file.

4.