7 PLANT CLASSIFICATION

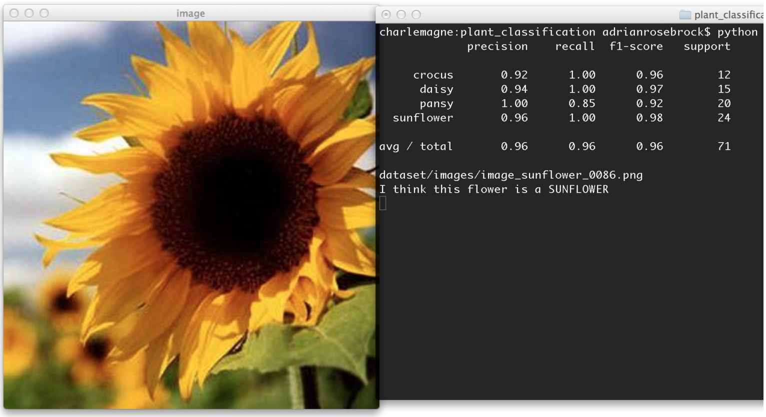
Stale coffee. Again.

Just think, thought Charles, *The New York Museum of Natural History*, one of the most prestigious museums in the world, and all they had to offer for their curators is stale, old coffee.

With a flick of his wrist and a sigh of dissatisfaction, Charles shoved the coffee pot back into its holder and continued down the hall to his office.

Charles had been with the museum for ten years now and quickly ascended within the ranks. And he had no doubt that his college minor in computer science was certainly a huge aid in his success.

While his more senior colleagues were struggling with Excel spreadsheets and Word documents, Charles was busy writing Python scripts and maintaining IPython Notebooks to manage and catalog his research.



```
charlemagne:plant_classification adrianrosebrock$ python
             precision    recall  f1-score   support

     crocus       0.92      1.00      0.96        12
      daisy       0.94      1.00      0.97        15
      pansy       1.00      0.85      0.92        20
  sunflower       0.96      1.00      0.98        24

avg / total       0.96      0.96      0.96        71

dataset/images/image_sunflower_0086.png
I think this flower is a SUNFLOWER
```

Figure 7.1: Charles' goal is to classify plant species using color histograms and machine learning. This model is able to correctly classify the image on the left as a sunflower.

His latest line of work involves the automatic classification of flower species.

Normally, identifying the species of a flower requires the eye of a trained botanist, where subtle details in the flower petals or the stem can indicate a drastically different species of flower. This type of flower classification is very time consuming and tedious.

And from the museum's point of view, it's also very expensive. Charles' time isn't cheap – and the museum was paying him dearly for it.

But truth be told, Charles admitted to himself, he would rather be performing research, working on his manuscripts and publishing his results in the latest issue of *Museum*, although he heard Margo, the new chief editor, was a real stickler.

His heart was in his love of academia and research – not the dull task of classifying flower species day in and day out.

That's why he decided to build a classifier to *automatically* classify the species of flowers in images. A classifier like this one would save him a bunch of time and free him up to get back to his precious research.

Charles has decided to quantify the flower images using a 3D RGB histogram. This histogram will be used to characterize the color of the flower petals, which is a good starting point for classifying the species of a flower.

Let's investigate his image descriptor:

Listing 7.1: rgbhistogram.py

```python
import cv2

class RGBHistogram:
    def __init__(self, bins):
        self.bins = bins

    def describe(self, image, mask = None):
        hist = cv2.calcHist([image], [0, 1, 2],
            mask, self.bins, [0, 256, 0, 256, 0, 256])
        cv2.normalize(hist, hist)
```

```
11
12          return hist.flatten()
```

Charles starts off by importing `cv2`, the only package that he'll be needing to create his image descriptor.

He then defines the `RGBHistogram` class on **Line 3** used to encapsulate how the flower images are quantified. The `__init__` method on **Line 4** takes only a single argument – a list containing the number of bins for the 3D histogram.

Describing the image will be handled by the `describe` method on **Line 7**, which takes two parameters, an `image` that the color histogram will be built from, and an optional `mask`. If Charles supplies a `mask`, then only pixels associated with the masked region will be used in constructing the histogram. This allows him to describe *only* the petals of the image, ignoring the rest of the image (i.e., the background, which is irrelevant to the flower itself).

Note: More information on masking can be found in Chapter 6 of Practical Python and OpenCV.

Constructing the histogram is accomplished on **Line 8**. The first argument is the image that Charles wants to describe. The resulting image is normalized on **Line 10** and returned as a feature vector on **Line 12**.

Note: The `cv2.normalize` function is slightly different between OpenCV 2.4.X and OpenCV 3.0. In OpenCV 2.4.X, the `cv2.normalize` function would actually return *the normalized histogram. However, in OpenCV 3.0+, `cv2.normalize` actually normalizes the histogram* within the function *and updates the second parameter (i.e., the "output") passed in. This is a sub-*

tle, but important difference to keep in mind when working with the two OpenCV versions.

Clearly, Charles took the time to read through Chapter 7 of *Practical Python and OpenCV* to understand how histograms can be used to characterize the color of an image.

Now that the image descriptor has been defined, Charles can create the code to classify what species a given flower is:

Listing 7.2: classify.py

```python
1  from __future__ import print_function
2  from pyimagesearch.rgbhistogram import RGBHistogram
3  from sklearn.preprocessing import LabelEncoder
4  from sklearn.ensemble import RandomForestClassifier
5  from sklearn.cross_validation import train_test_split
6  from sklearn.metrics import classification_report
7  import numpy as np
8  import argparse
9  import glob
10 import cv2
11
12 ap = argparse.ArgumentParser()
13 ap.add_argument("-i", "--images", required = True,
14     help = "path to the image dataset")
15 ap.add_argument("-m", "--masks", required = True,
16     help = "path to the image masks")
17 args = vars(ap.parse_args())
```

Lines 1-10 handle importing the packages to build Charles' flower classifier. First, Charles imports his `RGBHistogram` used to describe each of his images, stored in the `pyimagesearch` package for organization purposes.

The `LabelEncoder` class from the scikit-learn library is imported on **Line 3**. In order to build a machine learning classifier to distinguish between flower species, Charles first needs a way to encode "class labels" associated with each species.

Charles wants to distinguish between sunflowers, crocus, daisies, and pansies, but in order to construct a machine learning model, these species (represented as strings) need to be converted to integers. The `LabelEncoder` class handles this process.

The actual classification model Charles uses is a `RandomForestClassifier`, imported on **Line 4**. A random forest is an ensemble learning method used for classification, consisting of multiple decision trees.

For each tree in the random forest, a bootstrapped (sampling with replacement) sample is constructed, normally consisting of 66% of the dataset. Then, a decision tree is built based on the bootstrapped sample. At each node in the tree, only a sample of predictors are taken to calculate the node split criterion. It is common to use $\sqrt{n}$ predictors, where n is the number of predictors in the feature space. This process is then repeated to train multiple trees in the forest.

Note: A detailed review of random forest classifiers is outside the scope of this case study, as machine learning methods can be heavily involved.

However, if you are new at using machine learning, random forests are a good place to start, especially in the computer vision domain where they can obtain higher accuracy with very little effort.

Again, while this doesn't apply to all *computer vision classification problems, random forests are a good starting point to obtain a baseline accuracy.*

Charles then imports the `train_test_split` function from scikit-learn. When building a machine learning model, Charles needs two sets of data: a *training* set and a *testing* (or validation) set.

The machine learning model (in this case, a random forest) is trained using the *training* data and then evaluated against the *testing* data.

It is *extremely important* to keep these two sets exclusive as it allows the model to be evaluated on data points that it has not already seen. If the model has already seen the data points, then the results are biased since it has an unfair advantage!

Finally, Charles uses NumPy for numerical processing, `argparse` to parse command line arguments, `glob` to grab the paths of images off disk, and `cv2` for his OpenCV bindings.

Wow, that certainly was a lot of packages to import!

But it looks like Charles only needs two command line arguments: `--images`, which points to the directory that contains his flower images, and `--masks`, which points to the directory that contains the masks for his flowers. These masks allow him to focus only on the parts of the flower (i.e the petals) that he wants to describe, ignoring the background and other clutter that would otherwise distort the feature vector and insert unwanted noise.

Note: The images used in this example are a sample of the Flowers 17 dataset by Nilsback and Zisserman. I have also updated their tri-maps as binary masks to make describing the images easier. More about this dataset can be found at: http://www.robots.ox.ac.uk/ vgg/data/flowers/17/index.html.

Now that all the necessary packages have been imported and the command line arguments are parsed, let's see what Charles is up to now:

Listing 7.3: classify.py

```
19 imagePaths = sorted(glob.glob(args["images"] + "/*.png"))
20 maskPaths = sorted(glob.glob(args["masks"] + "/*.png"))
21
22 data = []
23 target = []
24
25 desc = RGBHistogram([8, 8, 8])
26
27 for (imagePath, maskPath) in zip(imagePaths, maskPaths):
28     image = cv2.imread(imagePath)
29     mask = cv2.imread(maskPath)
30     mask = cv2.cvtColor(mask, cv2.COLOR_BGR2GRAY)
31
32     features = desc.describe(image, mask)
33
34     data.append(features)
35     target.append(imagePath.split("_")[-2])
```

On **Lines 19 and 20**, Charles uses `glob` to grab the paths of his images and masks, respectively. By passing in the directory containing his images, followed by the wild card `*.png`, Charles is able to quickly construct his list of image paths.

Lines 22 and 23 simply initialize Charles' data matrix and list of class labels (i.e., the species of flowers).

Line 25 then instantiates his image descriptor – a 3D RGB color histogram with 8 bins per channel. This image descriptor will yield an $8 \times 8 \times 8 = 512$-dimensional feature vector used to characterize the color of the flower.

On **Line 27** Charles starts to loop over his images and masks. He loads the image and mask off disk on **Line 28 and 29**, and then converts the mask to grayscale on **Line 30**.

Applying his 3D RGB color histogram on **Line 32** yields his feature vector, which he then stores in his `data` matrix on **Line 34**.

The species of the flower is then parsed out and the list of targets updated on **Line 35**.

Now Charles can apply his machine learning method:

Listing 7.4: classify.py

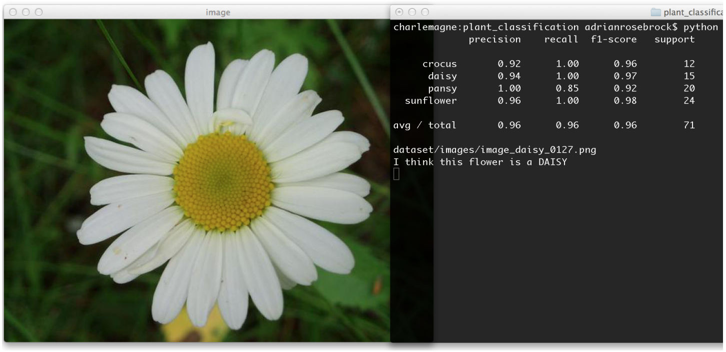
```python
targetNames = np.unique(target)
le = LabelEncoder()
target = le.fit_transform(target)

(trainData, testData, trainTarget, testTarget) = train_test_split(data, target,
    test_size = 0.3, random_state = 42)

model = RandomForestClassifier(n_estimators = 25, random_state = 84)
model.fit(trainData, trainTarget)

print(classification_report(testTarget, model.predict(testData),
    target_names = targetNames))
```

First, Charles encodes his class labels on **Lines 37-39**. The unique method of NumPy is used to find the unique species names, which are then fed into the LabelEncoder. A call to fit_transform "fits" the unique species names into integers, a category for each species, and then "transforms" the strings into their corresponding integer classes. The target variable now contains a list of integers, one for each data point, where each integer maps to a flower species name.

From there, Charles must construct his training and testing split on **Line 41**. The train_test_split function takes care of the heavy lifting for him. He passes in his data matrix and list of targets, specifying that the test dataset should be 30% of the size of the entire dataset. A pseudo random state of 42 is used so that Charles can reproduce his results in later runs.

The RandomForestClassifier is trained on **Line 44 and 45** using 25 decision trees in the forest. Again, a pseudo random state is explicitly used so that Charles' results are reproducible.

Line 47 then prints out the accuracy of his model using the classification_report function. Charles passes in the actual testing targets as the first parameter and then lets the model predict what it thinks the flower species are for the testing data. The classification_report function then compares the *predictions* to the *true targets* and prints an accuracy report for both the overall system and each individ-



```
charlemagne:plant_classification adrianrosebrock$ python
             precision    recall  f1-score   support

     crocus       0.92      1.00      0.96        12
      daisy       0.94      1.00      0.97        15
      pansy       1.00      0.85      0.92        20
  sunflower       0.96      1.00      0.98        24

avg / total       0.96      0.96      0.96        71

dataset/images/image_daisy_0127.png
I think this flower is a DAISY
```

Figure 7.2: Charles has used color histograms a random forest classifier to obtain a high accuracy plant classifier.

ual class label.

The results of Charles' experiment can be seen in Figure 7.2.

His random forest classifier is able to classify a crocus correctly 100% of the time, a daisy 100% of the time, a pansy 90% of the time, and a sunflower 96% of the time.

Not bad for using just a color histogram for his features!

To investigate the classification further, Charles defines the following code:

Listing 7.5: classify.py

```python
for i in np.random.choice(np.arange(0, len(imagePaths)), 10):
```

```
51      imagePath = imagePaths[i]
52      maskPath = maskPaths[i]
53
54      image = cv2.imread(imagePath)
55      mask = cv2.imread(maskPath)
56      mask = cv2.cvtColor(mask, cv2.COLOR_BGR2GRAY)
57
58      features = desc.describe(image, mask)
59
60      flower = le.inverse_transform(model.predict([features]))[0]
61      print(imagePath)
62      print("I think this flower is a {}".format(flower.upper()))
63      cv2.imshow("image", image)
64      cv2.waitKey(0)
```

On **Line 50**, he randomly picks 10 different images to investigate, then he grabs the corresponding image and mask paths on **Lines 51 and 52**.

The `image` and `mask` are then loaded on **Lines 54 and 55** and the `mask` converted to grayscale on **Line 56**, just as in the data preparation phase.

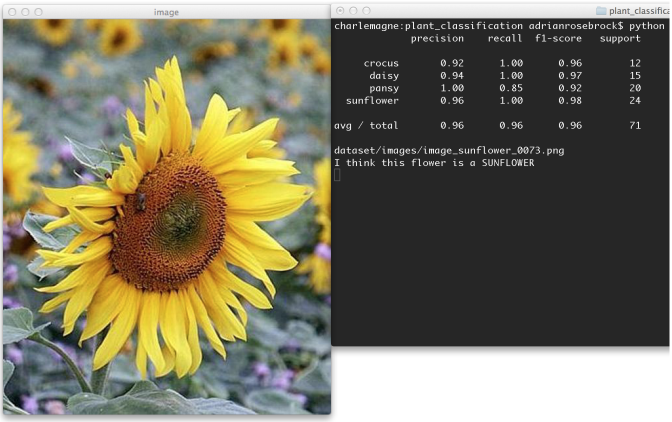
He then extracts his feature vector on **Line 58** to characterize the color of the flower.

His random forest classifier is queried on **Line 60** to determine the species of the flower, which is then printed to console and displayed on screen on **Lines 61-64**.

Charles executes his flower classifier by issuing the following command:

Listing 7.6: classify.py

```
$ python classify.py --images dataset/images --masks dataset/
    masks
```



```
charlemagne:plant_classification adrianrosebrock$ python
             precision    recall  f1-score   support

     crocus       0.92      1.00      0.96        12
      daisy       0.94      1.00      0.97        15
      pansy       1.00      0.85      0.92        20
  sunflower       0.96      1.00      0.98        24

avg / total       0.96      0.96      0.96        71

dataset/images/image_sunflower_0073.png
I think this flower is a SUNFLOWER
```

Figure 7.3: Charles' classifier correctly labels the flower as a sunflower.

First, his script trains a random forest classifier on the color histograms of the flowers and then is evaluated on a set of testing data.

Then, Figure 7.3 and Figure 7.4 display a sampling of his results. In each case, his classifier is able to correctly classify the species of the flower.

Satisfied with his work, Charles decides it's time for that cup of coffee.

And not the stale junk in the museum cafeteria either.

No, a triumphant day like this one deserves a splurge.