IN4393: Computer Vision Project
Flower species recognition

Feng Lu 4619161
Xiangwei Shi 4614909

July 1, 2017

1 Problem Description

This project is completed to recognize and detect the flower species. Through extracting different features from the images of different flower species, the final recognizing system is able to detect 102 flower species. The main steps for detecting flower species are extracting features from training images and training different classifiers with extracted features. The dataset for this project is '102 category flower data' from University of Oxford[1]. In this image dataset, there are 102 different flower species with over 8000 images in total. Each class in the dataset consists of at least 40 images. The basic algorithm applied for this project is 'bag of visual word'. We firstly extract the features(SIFT descriptors) of RGB images. In order to get a codebook of 'bag of visual word', k-means clustering method is applied to classify SIFT descriptors. Lastly, we choose the best three classifiers out of seven different ones as an ensemble to perform the final classification, which we believe can improve the performance of one single classifier.

2 Approaches and Algorithms

For the implementation of the project, we mainly used Python 2, OpenCV 3 library and library scikit-learn. The main Computer Vision algorithm we constructed is the bag of visual words using SIFT descriptors.

[1] `http://www.robots.ox.ac.uk/~vgg/data/flowers/102/index.html`

The whole dataset contains 8190 images, from which we used 2040 images covering all 102 flower species during the training process. While during the testing, for each time, we tested 100 images randomly selected from the whole database.

2.1 Feature extraction

Instead of using randomly selected image patches like the lab assignment, the main features we used in this project are the SIFT descriptors of each RGB image[1]. By performing bag of visual words algorithm later, we can extract quite representative features(histogram of the codebook) for each image. Due to relative poor performance of RGB images, we also tried computing corresponding HSV images and used them as input of the system.

In this process, we first loaded all the training data into the system and then performed the SIFT detection for each image. After that, we can obtain a list of keypoint descriptors for each image, which can later be used as the features for k-means clustering.

2.2 Descriptors clustering and codebook generation

Once we obtained the descriptors for each image, we fed all of them into the KMeans(from sklearn library) function to perform the k-means clustering. The return of the clustering algorithm is exactly the codebook we need to construct the bag-of-visual-word feature vector for each image. Next, we assigned each descriptor of the image to its nearest neighbor in the codebook (based on the squared Euclidean distance).

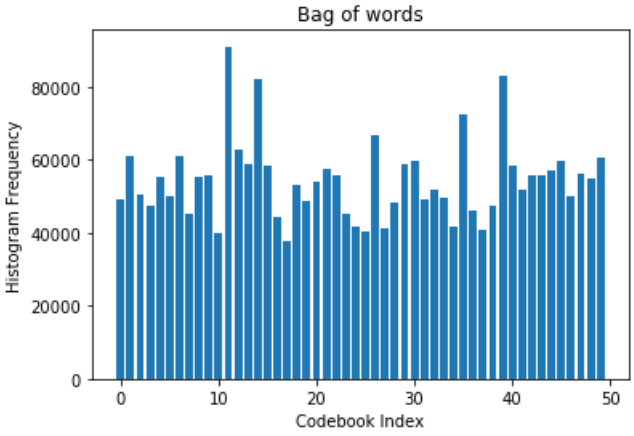

Figure 1: Histogram of bag-of-visual-word

2.3 Obtain the histograms of each image

After we got the nearest neighbor for each descriptor in the codebook, we could finally construct the bag-of-visual-word feature vector for each image by constructing a (normalized)

prototype assignment histogram. These histograms for each image shown in Fig 1 are exactly the feature vectors we used for image classification later on.

2.4 Classification

In this project, generally, we used 7 classifiers:

- AdaBoost classifier
- Multilayer perceptron(nodes for hidden layers: 256,1024,512,128)
- Bagging Classifier
- Logistic Classifier
- SVM one vs all
- Random Forest Classifier without boosting
- Random Forest Classifier with boosting

We preformed cross-validation for each classifier on the whole training set at the beginning. After we compared all the cross-validation scores, we applied ensemble method to the top 3 ranked classifiers to obtain the final classification result.

For convenience, we also implemented a separated function for recognizing one single testing image using the final combined classifier. The final returned results are two parts: the predicted label and the corresponding confidence.

2.5 Interface

The user interface was simply designed by Django package in Python. There are three different kinds of interfaces, which supports several functions. The first interface is the homepage with a background image and two buttons for uploading test image and submitting it. The second interface shows a returned image with a button for detection. The returned image is the uploaded image with a bounding box which can select the flower section in the image. The last interface will present not only the predicted label and the confidence and also a reference image from the dataset with the same label with the predicted one. In this interface, there are two more buttons for re-uploading new test images.

For the bounding-box function, we use the *findContours* function in OpenCV3 package to detect the contours appearing in the test image. Then select one contour with the biggest area and draw a rectangle to represent the contour.

For the detection function, we use the previously saved 'bag of visual word' codebook and classifier model, which have the best performance in testing process, to complete the recognizing flower species function.

3 Experiments and Results

The whole dataset contains 8190 images as mentioned above, which could be quite computationally expensive even if we just use half of them as training. We used 2040 images forming our training dataset which averagely covers all 102 flower species at the same time. The testing dataset is constructed by randomly selecting 100 images from the whole dataset. We tested different size of k(for k-means clustering) to obtain the final ensemble training and testing accuracy. Generally, larger k gives higher classification accuracy. We kept adding different classifiers to the system step by step during the constructing process. Finally, we used 7 classifiers as mentioned in last section. Moreover, to enhance the classification accuracy, we also tried computing the corresponding HSV images with color quantization processing, which were expected to give better performance. While the results showed that it actually helped little.

Table 3.1: Classification results for RGB images

k size	training accuracy	testing accuracy
k = 10	0.99	0.23
k = 20	1.0	0.32
k = 50	1.0	0.45
k = 200	0.99	0.54

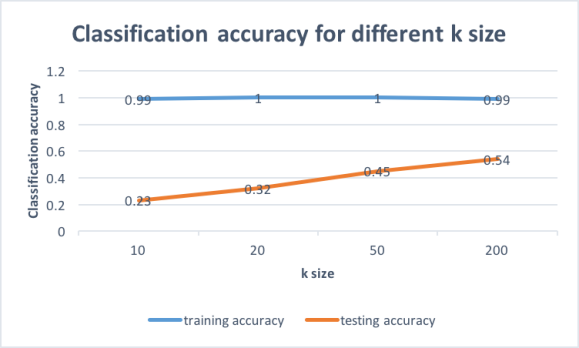

Figure 2: classification accuracy for different k size

3.1 RGB images as input

The execution time of the k-menas clustering depends on the size k. For k = 10, it takes about 15 minutes to finish clustering. While for k = 200, it takes over 5 hours. Due to the high time cost for clustering, we have not tried higher k size in this project. While according to our findings, we believe that higher k can still obtain higher classification accuracy. The results of the experiment can be found in table 3.1 and Fig 2.

The result shows there is a huge overfitting problem in our system. One way to avoid such problem is to increase the size of training dataset. Another way is to keep increasing k size. Since we do not have so much time for training our system in this project, we left it for future work.

3.2 HSV Images with Color Quantization processing as input

Since we are not so satisfied with the performance of our system. We tried to use HSV images with color quantization processing as the input of our system. While the testing result did not show much enhancement, with a testing accuracy of 0.29 for k = 20. Thus, we decided to abandon this approach.

3.3 Predict the label of one single image via our interface

We also tested our interface to predict the label for one single image. Since our project contains over 100 flower species and most of them are quite similar to each other, we do not expect high prediction confidence. Generally, the prediction confidence is around 0.4-0.6. For some species like marigolds, the confidence can be lower since many other species look quite similar to them. For better comparison, we provide reference image from our database for each predicted label by our system. The interface example is shown in Fig 3.

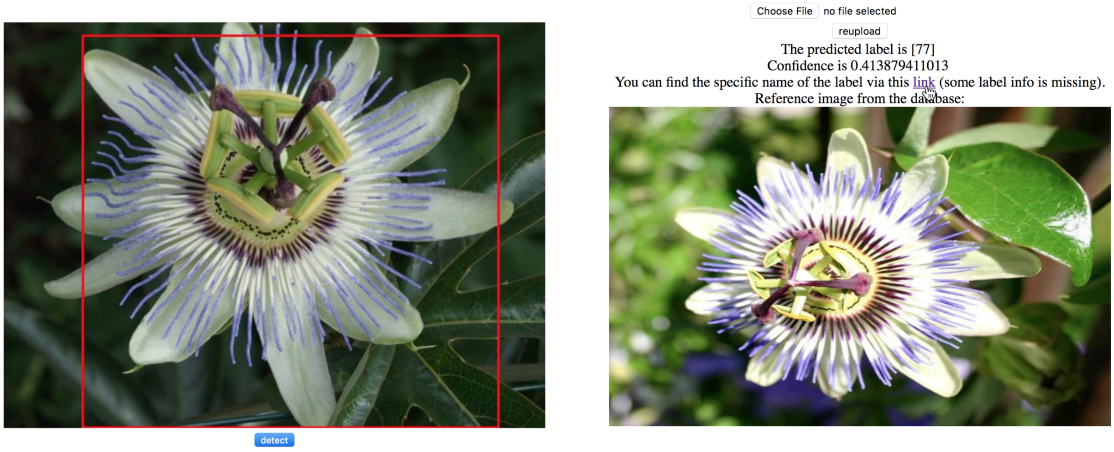

Figure 3: Interface for predicting one single image.

4 Overview of Codes and System Instruction

4.1 Overview of Codes

The main entrance of the system is via maindetector.py. In this file, we implemented three functions: train_Model, test_Model and recognizeImage. These three functions are all writ-

ten by ourselves.

The bow.py file mainly contains the functions to implement the bag of visual words algorithm. In these functions, we referred to OpenCV 3 library, utilizing functions to compute the SIFT descriptors. We also utilized sklearn library here to implement the k-means clustering.

The trainModels.py file contains the models for classification. All of the models we used here are referred to sklearn tutorial[2].

The convert_hsv.py file contains the function for computing the hsv images for our dataset, which is referred from [3].

Files located at the FlowerDetector folder are all related to the interface of our system, which are all written by ourselves.

4.2 System Instruction

The System Instruction can be accessed via our github repository[4]. We also include a demo video at youtube[5].

5 Conclusion

Our system consists of several parts including 'bag of visual word' algorithm, bounding-box function and an user interface. Because there are 102 species of flowers, the accuracy of our system is good, mostly around 40% - 50% and some reaching 60% or so. For the 'bag of visual word' algorithm, we extracted the different features from the images, created the SIFT descriptors, used k-means method to create a codebook and trained multiple classifiers to get a model with good performance. For the bounding-box function, we implemented it in the user interface. It can draw one contour covering the biggest area. As for the user interface, it contains three different kinds of interfaces and is the basic framework to demonstrate the application of our final system. Since the number of training images is far smaller compared to the number of the features extracted from the images, there is a obvious over-fitting problem in our system with very high training accuracy and low testing accuracy. To solve this problem, it needs to enlarge the size of training data, which is time-consuming and needs powerful computational devices.

[2] http://scikit-learn.org/stable/modules/classes.html#module-sklearn.ensemble
[3] http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_ml/py_kmeans/py_kmeans_opencv/py_kmeans_opencv.html
[4] https://github.com/fredericklu/ComputerVision_FinalProject.git
[5] https://www.youtube.com/watch?v=t9dBP9JV2Jk&feature=youtu.be

There is also much further work that can be done on this project. Increasing the size of training dataset and *k* in K-means method for creating 'bag of visual word' codebook can boost the performance of recognition, which should be very time-consuming and needs powerful computational devices. This the major factor that we do not use the whole image dataset for training and testing. Moreover, the bounding-box function can also be improved. Instead of using *findContours* function to find a contour covering the biggest area, drawing bounding-boxes after detecting the locations of flowers (single and plural) can also be an improvement. Since our system can only recognize single flower species, building a system to recognize different flowers in one image can also be a direction for the further work.

References

[1] L. Fan J. Willamowski G. Csurka, C. Dance and C. Bray. Visual categorization with bags of keypoints. *In Workshop on Statistical Learning in Computer Vision vol 1*, page 22, 2004.