ECSE-415 Introduction to Computer Vision

Assignment #2: Bag of words for object classification

Due: Friday, Feb. 26, 2016, 11:59pm

Please submit your assignment solutions electronically via the myCourses assignment dropbox. The solutions should be in PDF format. Attempt all parts of this assignment. Directives about the report is given in Section 4.

This assignment is out of a total of **100 points**. It is designed to familiarize the students with writing image processing algorithms in using OpenCV library. In this assignment, the students obtain deeper understanding of using bag of words for object classification. The assignment requires the students to create an **OpenCV 2.4.11** application in an IDE of your choice (MSVC, Qt, Eclipse, XCode etc.). If you are not using MS Visual Studio on Windows, you would need to build the OpenCV 2.4.11 library using CMake and a platform-compatible C/C++ compiler.

The assignment has two parts. In the first part, the student is required to create a codebook of SIFT features to describe a set of training images from different object categories (see Section 2). A set of training images is provided for this part. The outlines of each object has been manually annotated in these pictures. In the second part, the student is asked to evaluate the performance of the bag of words method in classifying different objects categories (see Section 11)). A set of separate testing images is provided for this part. Both the training and the testing images and the annotations are from the Caltech 101 database. A helper class has been provided to assist in loading the training and testing images and the annotations.

The student is expected to write his/her own code. To submit your code, just submit your commented .cpp file (including the definition for the `main`, `Train` and `Test` functions, and the included header files), along with the report. You do not need to submit the project files nor the built binaries of your code nor the OpenCV library. You are not allowed to use any other library besides the C++ STD and OpenCV 2.4.11.

Assignments received up to 24 hours late will be penalized by 30%. Assignments received more than 24 hours late will not be marked.

1 Overview of the Code Template

Please use the code template in Table 1 for writing you code. If you need help with the syntax, argument list and parameters of any of the OpenCV related class/methods, please refer to the "OpenCV Reference Manual Release 2.4.11.0" (e.g. if you need to find out how to use the imshow method, search imshow() and check the syntax and the argument list for C++).

Table 1: Code template for Assignment 2

```
#include <opencv2/opencv.hpp>
#include <opencv2/nonfree/nonfree.hpp>
#include <string>
#include <fstream>
#include <sstream>
#include <iomanip>
#include <algorithm>

using namespace cv;
using namespace std;

/* Helper class declaration and definition */
class Caltech101 { … };

/* Function prototypes */
void Train(const Caltech101 &Dataset, Mat &codeBook, vector<vector<Mat>> imageDescriptors, const int numCodewords);
void Test(const Caltech101 &Dataset, const Mat codeBook, const vector<vector<Mat>> imageDescriptors);


void main(void)
{
	/* Initialize OpenCV nonfree module */
	initModule_nonfree();

	/* Put the full path of the Caltech 101 folder here */
	const string datasetPath = "C:/Caltech 101";

	/* Set the number of training and testing images per category */
	const int numTrainingData = 40;
	const int numTestingData = 2;

	/* Set the number of codewords*/
	const int numCodewords = 100;

	/* Load the dataset by instantiating the helper class */
	Caltech101 Dataset(datasetPath, numTrainingData, numTestingData);

	/* Terminate if dataset is not successfull loaded */
	if (!Dataset.isSuccessfullyLoaded())
	{
		cout << "An error occurred, press Enter to exit" << endl;
		getchar();
		return;
	}

	/* Variable definition */
	Mat codeBook;
	vector<vector<Mat>> imageDescriptors;

	/* Training */
	Train(Dataset, codeBook, categoryDescriptor, numCodewords);

	/* Testing */
	Test(Dataset, codeBook, categoryDescriptor);
}
```

```
void Train(const Caltech101 &Dataset, Mat &codeBook, vector<vector<Mat>> imageDescriptors, const int numCodewords)
{

}

void Test(const Caltech101 &Dataset, const Mat codeBook, const vector<vector<Mat>> imageDescriptors)
{

}
```

1.1 The helper class

A helper class (`Caltech101`) is provided to assist you in loading the training and testing images and annotations and displaying them. The `Caltech101` class contains the following public data and function members:

Table 2: Public data members and functions of the `Caltech101` class

Class member	Description
Functions	
`Caltech101::Caltech101 (…)`	The class constructor. By instantiating an object of the class, this function is called to load the training and testing images and annotations.
`bool isSuccessfullyLoaded ()`	Call this function to check if the data has been successfully loaded.
`void dispTrainingImage (int categoryIdx, int imageIdx)`	Displays an annotated training image, specified by the input indices.
`void dispTestImage (int categoryIdx, int imageIdx)`	Displays an annotated test image, specified by the input indices.
Data members	
`vector<string> categoryNames`	1D array of strings, containing the name of each object category.
`vector<vector<Mat>> trainingImages`	2D array of Mat objects, containing the training images.
`vector<vector<Rect>> trainingAnnotations`	2D array of Rect objects, containing the training annotations.
`vector<vector<Mat>> testImages`	2D array of Mat objects, containing the test images.
`vector<vector<Rect>> testAnnotations`	2D array of Rect objects, containing the test annotations.

As illustrated in the Table 1, all you need to do is to instantiate an object from this class (the `Dataset` object) and pass the required arguments to its constructor `Caltech101::Caltech101(string datasetPath, int numTrainingImages, int numTestImages)`. Keep in mind that a class constructor is automatically called whenever an object is instantiated from the class. The `main` routine passes the dataset directory in `string datasetPath`, and the number of training and testing images in `int numTrainingImages` and `int numTestImages`. Note that the `Caltech101` class assumes that the number of training and testing images are similar for all object categories. After loading the dataset, the code queries the `Dataset.isSuccessfullyLoaded()` function (which returns `true` if loading is successful) to check whether the data is successfully loaded. Note that you might need to set the values for the dataset directory, the number of training and testing images and the number of codewords using the local variables at the beginning of the `main` routine. As shown in Table 1, these variables are defined as `const`, so their values should be only set during their initialization (any attempt to change their values gives an error from the compiler).

You can now easily access all the training and testing images and their annotation using the `Dataset` object. The helper class creates a vector of vector of `Mat` objects (`vector<vector<Mat>>`) to store the training and the testing images. Note that in C++, an `std::vector<T>` vector object is a dynamic array of `T` objects with many useful built-in routines. The most important differences between a C++ vector and a C++ array is the ability of the vector to change its size dynamically during runtime, erasing an element, adding element and runtime queries about the size of the vector (see http://www.cplusplus.com/reference/vector/vector/). If we have a `vector<Mat> T` object, we can easily access each element by simply indexing the veector, e.g. `T[0]` and `T[1]`. We can also determine the number of elements in `T` using the `T.size()` function.

Similarly in our case, we are dealing with a two dimensional vector which requires two indices to access each element; one to specify a category and the other to specify an image in the category. For example, you can access the `Mat` object of the second training image of the third category and the `Mat` object of the first test image of the second category by `Dataset.trainingImages[2][1]` and `Dataset.testImages[1][0]`, respectively (keep in mind that the indices start at zero). We can also determine the number of object categories and the number of images in each category (let's say the second category) using `Dataset.trainingImages.size()` and `Dataset.trainingImages[1].size()`, respectively.

Similarly, you can access their corresponding annotation `Rect` objects using `Dataset.trainingAnnotations[2][1]` and `Dataset.testAnnotations[1][0]`. Using the annotation rectangles, you can easily extract the objects bounding boxes. For example, the top-left corner row, column and the height and the width of the rectangle could be accessed by `Dataset.trainingAnnotations[2][1].y`, `Dataset.trainingAnnotations[2][1].x`, `Dataset.trainingAnnotations[2][1].height`, and `Dataset.trainingAnnotations[2][1].width` respectively.

You are required to write the description for `Train` and `Test` functions, which are described in Sections 2 and 11), respectively. The `Train` function first creates a codebook from the training images and then computes the bag of words representation (i.e. the histogram of codewords) for each training image of each category. The `Test` function evaluates the performance of the codebook in object classifying using the test images.

2 Train function (40 pts)

To do object classification using bag of words, we use training images from each object category to create a codebook of visual features. In this assignment, we are going to extract SIFT features from the training images and create a codebook by clustering the feature descriptors into a given number of codewords. After obtaining the codebook, each training image is converted to its bag of words representation, i.e. a histogram representing the number of image features quantized into each of the codewords (i.e. cluster centers). Then, to classify a new test image, the same procedure is used to convert the image into its bag of words histogram representation and by comparing it with the bag of words histogram of all of the training images, the closest match is used to assign the category label of the test image (this is called nearest neighbor classifier). This is shown in Figure 1.

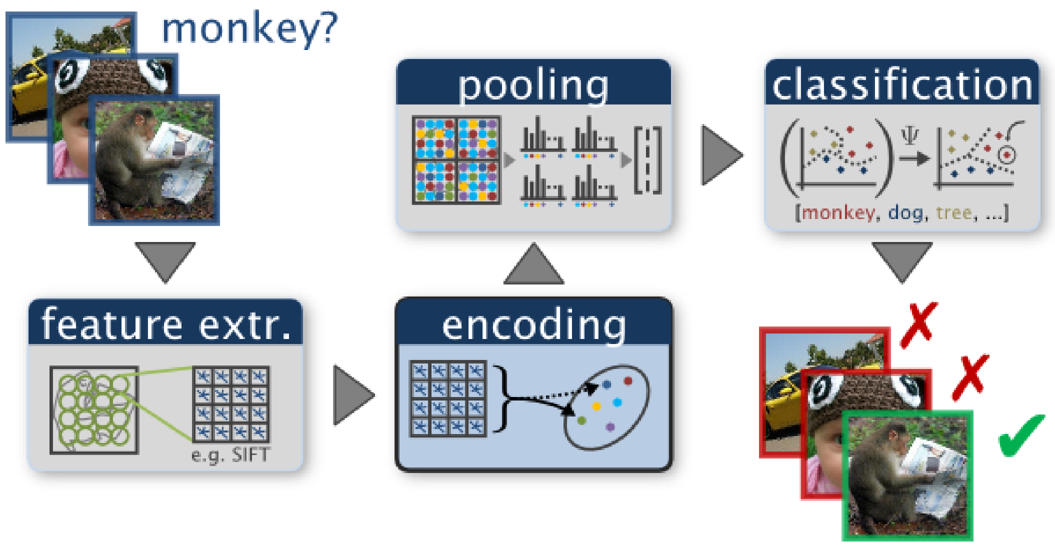

Figure 1: Typical steps of bag of visual words image classification pipeline (image from http://www.robots.ox.ac.uk/~vgg/research/encoding_eval/).

Extracting features, clustering to build a codebook, and building histograms can be slow. A good implementation can run the entire pipeline in less than 10 minutes (this depends on the number of codewords). Follow the following steps while implementing `Train` function:

1) Create a SIFT feature detector object
    - Use `Ptr<FeatureDetector>` class with a `"SIFT"` argument in the create method
2) Create a SIFT descriptor extractor object
    - Use `Ptr<DescriptorExtractor>` class with a `"SIFT"` argument for in the create method
3) Create a `Mat` object to store all the SIFT descriptors of all training images of all categories (let's call it D)
4) For each training image of each object category (let's call it I), do
    - Detect SIFT key points in I
        - Use the `detect` method of the feature detector object
        - You need a `vector<KeyPoint>` object for storing the detected features
        - You can use the OpenCV `drawKeypoints(.)` function to draw the sift keypoints and visually inspect them (see Figure 2)
    - Discard the keypoints outside of the annotation rectangle
        - Go through each keypoint and keep it iff its location (`KeyPoint::pt.x` and `KeyPoint::pt.y`) is inside the annotation rectangle of I
        - You can use the OpenCV `rectangle(.)` function to draw the annotation rectangle (see Figure 2)
    - Compute the SIFT descriptor for the remaining keypoints

    - Use the `compute` method of the feature detector object
    - You need a `Mat` object for storing the descriptors (each descriptor will be stored as a row of the `Mat` object)
- Add the descriptors of the current image to D
    - Use the `push_back` method of `Mat` objects to easily append a `Mat` object at the end of another `Mat` object (note that the type and the number of columns of the `Mat` objects should be the same)

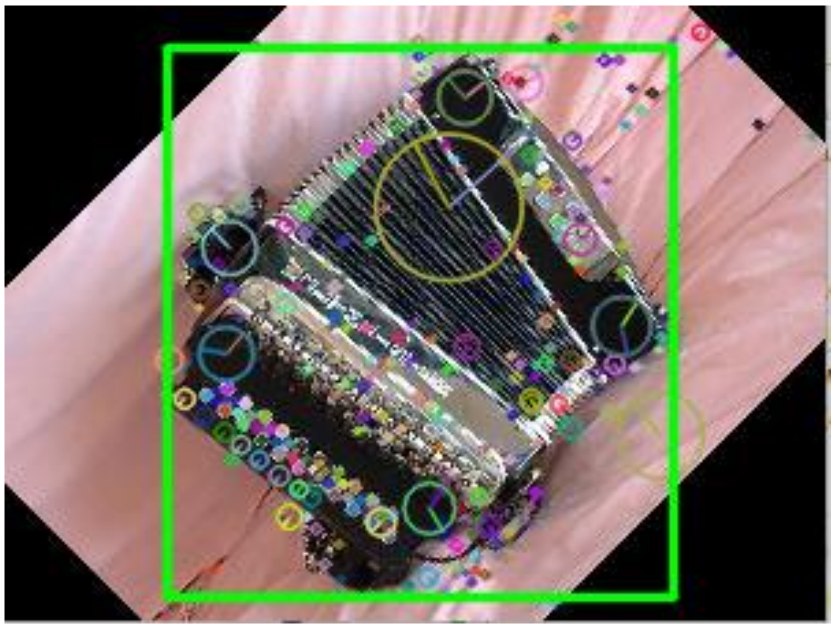
Figure 2: An annotated training image with SIFT keypoints from the accordion category.

At this point, we have all the descriptors of all the training images of all categories stored in the `Mat` object D. Next, we need to create the codebook by clustering all the descriptors and store it in the input variable `Mat &codeBook`. Note that this variable is created inside the `main` routine and is the passed by reference to the `Train` function so that its value would remain valid when the function returns). To do so, follow these steps:

5) Create a bag of words trainer object
    - Use the `BOWKMeansTrainer` class
6) Add the descriptors to the bag of words trainer object
    - Use the `add` method of the bag of words trainer object
7) Compute the codebook
    - Use the `cluster` method of the bag of words trainer object
    - Store the codebook to the input variable `codeBook`

Finally, we need to represent the object categories using the codebook. This enables us to later compare a new, unseen test image and assign it to a category by comparing it to the training images. To do this, we create a bag of words histogram representation for each training image and store it in the input variable `vector<vector<Mat>> imageDescriptors` (use the same ordering as

`Dataset.trainingImages` to store the bag of words histograms of training images). To do so, follow these steps:

8) Create a Brute Force descriptor matcher object
    - Use the `Ptr<DescriptorMatcher>` class with a `"BruteForce"` argument in the create method
9) Create a bag of words descriptor extractor object
    - Use the `Ptr<BOWImgDescriptorExtractor>` class
    - You need to pass the `Ptr<DescriptorExtractor>` and the `Ptr<DescriptorMatcher>` objects to the constructor of the bag of words descriptor extractor object
10) Set the codebook of the bag of words descriptor extractor object
    - Use the `setVocabulary` method of the bag of words descriptor extractor object
11) For each training image of each object category (let's call it I), do
    - Detect SIFT key points in I
    - Discard the keypoints outside of the annotation rectangle
    - Compute the bag of words histogram representation
        - Use the `compute2` method of the bag of words descriptor extractor object
        - You need a `Mat` object for storing the bag of words histogram (the `Mat` object would be one by the number of codewords in size)
    - Store the bag of words histogram of I in the corresponding indices of the input variable `imageDescriptors`

3 Test function (20 pts)

To classify a new test image in the bag of words methodology, we use the learned codebook to compute a bag of words histogram representation for the test image and then compare it to all of the bag of words histogram representation of training images. We classify the test image to the same category as the closest (based on Euclidean distance) training image. Follow the following steps while implementing `Test` function:

1) Create the following object
    - SIFT feature detector object, SIFT descriptor extractor object, Brute Force descriptor matcher object, and bag of words descriptor extractor object
2) Set the codebook of the bag of words descriptor extractor object
3) For each test image of each object category (let's call it I), do
    - Detect SIFT key points in I
    - Discard the keypoints outside of the annotation rectangle
    - Compute the bag of words histogram representation
    - Find the best matching histogram amongst the bag of words histogram representation of the training images
        - You can use the OpenCV `norm` function to calculate the Euclidean distance between two `Mat` objects
    - Assign the category label of the test image to the category of the closest match

4) Compare the estimated and the true category of the test images to obtain the recognition ratio computed as follows: $\frac{number\ of\ correctly\ assigned\ test\ images}{total\ number\ of\ test\ images}$.
5) Print the recognition rate of the method on the console.

4 Report (40 pts)

For the report, answer the following questions. In all cases, set the number of training and test images to 40 and 2, respectively.

1. Display two annotated randomly-chosen images with their SIFT features (as shown in Figure 2) from the same category. Comment on the distinctive features appearing on both images that could help identify the object category. (5 pts)
2. Compute the recognition rate with the following number of codewords: $\{10, 20,50,100,200,300,400,500,600,700,800,900,1000\}$. Plot the recognition rate against the number codewords. Based on the results, determine the best choice for the number of codewords. (10 pts)
3. Repeat the above question without discarding the SIFT features outside of the object bounding boxes (do not discard features while creating the codebook, and while computing the bag of words histograms for both training and testing images). Based on the results, does discarding features improve the recognition rate? You do not need to submit the code for this. (10 pts)
4. Compute and plot the recognition rate for the training images (by using the training images as testing images) for the following number of codewords: $\{10, 20,50,100,200,300,400,500,600,700,800,900,1000\}$. You do not need to submit the code for this. (10 pts)

To accelerate in producing the following figure, you should save the corresponding variables into text files and load them in Matlab.

5. Compute the average of the bag of words histogram representation of all training images for five categories (set the number of codewords to 100). Use Matlab to plot and compare the category histograms. Are they distinctive enough? (5 pts)

Bonus question (15 pts): Create another codebook using Harris feature detector with a SIFT descriptor. Compute the recognition rate with the following number of codewords: $\{10, 20,50,100,200,300,400,500,600,700,800,900,1000\}$. Compare the results with the SIFT codebook. Based on the comparison, which feature detector performs best in detecting distinctive features? (if you are reporting this, create another copy of your code and change the required part to replace SIFT with Harris. Send this file along with the original file and the report).