```
1
2
3   /* compile with
4   gcc -Wall bow-classification.cpp -I/usr/local/include/ \
5       -L/usr/lib/ -lstdc++ \
6       -L/usr/local/lib -lopencv_highgui -lopencv_core -lopencv_imgcodecs \
7       -lopencv_imgproc -lopencv_videoio -lopencv_video -lopencv_videostab \
8       -lopencv_flann -lopencv_ml -lm -lopencv_xfeatures2d -lopencv_features2d \
9       -o bow-classification -std=c++11
10  */
11
12  #include "opencv2/opencv_modules.hpp"
13  #include "opencv2/imgcodecs.hpp"
14  #include "opencv2/highgui.hpp"
15  #include "opencv2/imgproc.hpp"
16  #include "opencv2/features2d.hpp"
17  #include "opencv2/xfeatures2d.hpp"
18  #include "opencv2/ml.hpp"
19
20  #include <fstream>
21  #include <iostream>
22  #include <memory>
23  #include <functional>
24  #include <string>
25
26  // Includes for directory operations
27  #include <dirent.h>
28  #include <unistd.h>
29  #include <sys/stat.h>
30  #include <sys/types.h>
31
32  using namespace cv;
33  using namespace cv::xfeatures2d;
34  using namespace std;
35  using namespace cv::ml;
36
37  int main (int argc, char * const argv[]) {
38
39          string dir = "/home/lindo/dev/opencv-test/bow/TRAIN", filepath;
40          DIR *dp = NULL;
41          struct dirent *dirp;
42          struct stat filestat;
43
44          // Open directory containing training images
45          dp = opendir( dir.c_str() );
46          if (dp == NULL) {
47            cerr << "could not open directory of training images" << endl;
48            exit(EXIT_FAILURE);
49          }
50
51          // Step 1: Extract features of choice from training set that contains all classes
52          // (features are the descriptors of the training images' keypoints)
53
54          // Create SURF feature detector
55          double hessianThreshold = 1500;
56          Ptr<SURF> detector = SURF::create(hessianThreshold);
57          if( !detector ) {
58            cerr << "feature detector was not created" << endl;
59            exit(EXIT_FAILURE);
60          }
61          //int minHessian = 400;
62          //detector->setHessianThreshold(minHessian);
63
64          cout << "------- extract descriptors -------" << endl;
65          int count = 0;
66          vector<KeyPoint> keypoints;
67          Mat descriptors;
```

```
68          Mat training_descriptors(1, detector->descriptorSize(), detector->descriptorType());
69          Rect clipping_rect = Rect(0,120,640,480-120);
70          Mat bg_ = imread("background.png")(clipping_rect), img_fg, img;
71          while ( (dirp = readdir( dp )) ) {
72            //if (count++ > 15) break;
73
74            filepath = dir + "/" + dirp->d_name;
75
76            // If the file is a directory (or is in some way invalid) we'll skip it
77            if (stat( filepath.c_str(), &filestat )) continue;
78            if (S_ISDIR( filestat.st_mode ))         continue;
79
80            img = imread(filepath);
81            if( img.empty() ) { // Check for invalid input
82              cout << endl << "Could not open or find image, skipping" << filepath << endl;
83              continue;
84            }
85
86            img = img(clipping_rect);
87            img_fg = img - bg_;
88
89            // Compute keypoints amd thier associated descriptors
90            keypoints.clear();
91            detector->detect( img_fg, keypoints, noArray() ); // detect keypoints
92
93            /*{
94              Mat out; //img_fg.copyTo(out);
95              drawKeypoints(img_fg, keypoints, out, Scalar(255));
96              imshow("fg",img_fg);
97              imshow("keypoints", out);
98              waitKey(0);
99            }*/
100
101           detector->compute( img, keypoints, descriptors ); // compute descriptors
102
103           training_descriptors.push_back(descriptors);
104           cout << ".";
105         }
106         cout << endl;
107         closedir( dp );
108
109         cout << "Total training descriptors: " << training_descriptors.rows << endl;
110         //cout << "Desciptors: " << training_descriptors << endl;
111
112         // Step 2: Create a vocabulary of features by clustering the features
113         // (use BOW trainer to figure out which keypoints seem to form meaningful clusters)
114
115         cout << "------- create vocabulary ---------" << endl;
116
117         int clusterCount = 500; //num clusters = size of vocabulary
118         int attempts = 3;
119         int flags = cv::KMEANS_PP_CENTERS;
120         //int flags = cv::KMEANS_RANDOM_CENTERS;
121         TermCriteria terminate_criterion( TermCriteria::EPS | TermCriteria::COUNT, 10, 1.0 );
122         BOWKMeansTrainer bowTrainer( clusterCount, terminate_criterion, attempts, flags );
123         bowTrainer.add(training_descriptors);
124         //cout << "descriptors count: " << bowTrainer.descriptorsCount() << endl;
125         cout << "cluster BOW features" << endl;
126         Mat vocabulary = bowTrainer.cluster();
127         //cout << "vocabulary: " << vocabulary << endl;
128
129         // Step 3: Train classifier
130
131         // Setup extractor to convert images to presence vectors
132         Ptr< cv::DescriptorMatcher > descMatcher;
133         descMatcher = DescriptorMatcher::create( "BruteForce" );
134         Ptr< SURF > descExtractor = SURF::create();
```

```
135         Ptr< cv::BOWImgDescriptorExtractor > bowExtractor;
136         bowExtractor = new BOWImgDescriptorExtractor( descExtractor, descMatcher );
137         bowExtractor->BOWImgDescriptorExtractor::setVocabulary( vocabulary );
138 
139         // Setup training data for classifiers
140         map<string, Mat> classes_training_data;
141         classes_training_data.clear();
142 
143         cout << "------- create presence vectors ---------" << endl;
144 
145         Mat response_hist; // histogram of the presence (or absence) of each word in vocabulary
146         count = 0;
147         char buf[255];
148         ifstream ifs("/home/lindo/dev/opencv-test/bow/training.txt");
149         int total_samples = 0;
150 
151         do {
152           ifs.getline(buf, 255);
153           string line(buf);
154           istringstream iss(line);
155           //cout << line << endl;
156           iss >> filepath; // path to image name
157           Rect r; char delim;
158           iss >> r.x >> delim; // x coord of region of interest
159           iss >> r.y >> delim; // y coord of region of interest
160           iss >> r.width >> delim; // width of region of interest
161           iss >> r.height; // height of region of interest
162           //cout << r.x << "," << r.y << endl;
163           string class_;
164           iss >> class_; // class number
165           //cout << "class_ " << class_ << endl;
166 
167           if (class_ == "") {
168             cout << endl << "Image without class id, skipping" << endl;
169             continue;
170           }
171 
172           img = imread(filepath);
173           if( img.empty() ) { // Check for invalid input
174             cout << endl << "Could not open or find image, skipping" << filepath << endl;
175             continue;
176           }
177 
178           //rectangle(img, r, Scalar(0,255,0), 2, 8, 0); // draw region of interest on image
179           //imshow("before crop", img);
180 
181           r &= Rect(0, 0, img.cols, img.rows); // region of interest in terms of img rows and cols
182           if(r.width != 0) {
183             img = img(r); // crop to interesting region
184           }
185 
186           //char c__[] = { (char)atoi(class_.c_str()), '\0' };
187           //string c_(c__);
188           //cout << "c_ " << c_ << endl;
189           //putText(img, class_, Point(20,20), CV_FONT_HERSHEY_PLAIN, 2.0, Scalar(255), 2); //
    display img w/class #
190           //imshow("after crop", img);
191 
192           keypoints.clear();
193           detector->detectAndCompute( img, noArray(), keypoints, noArray() );
194           //cout << endl << "number of keypoints in training image: " << keypoints.size() << endl;
195           // create presence vector which will be normalized to # of keypoints in image
196           bowExtractor->BOWImgDescriptorExtractor::compute(img, keypoints, response_hist);
197           //cout << endl << "train response histogram: " << response_hist << endl;
198 
199           if(classes_training_data.count(class_) == 0) { // class not yet created...
200             classes_training_data[class_].create( 0, response_hist.cols, response_hist.type() );
```

```
201             }
202             classes_training_data[class_].push_back( response_hist );
203 
204             total_samples++;
205 
206             cout << ".";
207 
208             //waitKey(0);
209 
210           } while (!ifs.eof());
211           cout << endl;
212           cout << "Number of presence vectors: " << total_samples << endl;
213 
214           // Train 1-vs-all SVMs
215           Ptr<SVM> svm = SVM::create();
216           svm->setType(SVM::C_SVC);
217           svm->setKernel(SVM::RBF);
218           svm->setTermCriteria(TermCriteria(TermCriteria::MAX_ITER, 1000, FLT_EPSILON));
219 
220           map<string, Mat>::iterator it;
221           for (it = classes_training_data.begin(); it != classes_training_data.end(); ++it) {
222             string class_ = it->first;
223 
224             Mat samples(0, response_hist.cols, response_hist.type());
225             Mat labels(0, 1, CV_32SC1);
226 
227             // Copy class sample and label
228             cout << "adding " << classes_training_data[class_].rows << " positive" << endl;
229             samples.push_back(classes_training_data[class_]);
230             Mat class_label = Mat::ones(classes_training_data[class_].rows, 1, CV_32SC1);
231             labels.push_back(class_label);
232 
233             // Copy rest of samples and labels
234             map<string, Mat>::iterator it1;
235             for (it1 = classes_training_data.begin(); it1 != classes_training_data.end(); ++it1) {
236               string not_class_ = it1->first;
237               if(not_class_ == class_) continue;
238               samples.push_back(classes_training_data[not_class_]);
239               class_label = Mat::ones(classes_training_data[not_class_].rows, 1, CV_32SC1)*-1;
240               labels.push_back(class_label);
241             }
242 
243             Mat samples_32f;
244             samples.convertTo(samples_32f, CV_32F);
245             if(samples.rows == 0) {
246               cout << "No rows in samples, skipping" << endl;
247               continue;
248             }
249 
250             //cout << "class: " << class_ << endl;
251             //cout << "samples: " << samples_32f << endl;
252             //cout << "labels: " << labels << endl;
253 
254             // Construct training data from samples read from file above
255             Ptr<TrainData> td = TrainData::create(
256                                 samples_32f,                // Array of samples
257                                 ROW_SAMPLE,                 // Data in rows
258                                 labels,                     // Array of responses
259                                 noArray(),                  // Use all features
260                                 noArray(),                  // Use all data points
261                                 noArray(),                  // Do not use samples weights
262                                 noArray()                   // Do not specify inp and out types
263                                 );
264 
265             cout << "svm training for class: " << class_ << " starting" << endl;
266 
267             // Auto train using k-fold cross-validation
```

```
268             svm->trainAuto(
269                            td,
270                            10,
271                            SVM::getDefaultGrid(SVM::C),
272                            SVM::getDefaultGrid(SVM::GAMMA),
273                            SVM::getDefaultGrid(SVM::P),
274                            SVM::getDefaultGrid(SVM::NU),
275                            SVM::getDefaultGrid(SVM::COEF),
276                            SVM::getDefaultGrid(SVM::DEGREE),
277                            false
278                           );
279
280             cout << "svm training for class: " << class_ << " completed" << endl;
281
282             string svmFilename = "/home/lindo/dev/opencv-test/bow/CLASS/class_" + class_ + ".xml";
283             svm->save( svmFilename );
284
285             cout << "saved svm classifier to file" << endl;
286
287             // Prepare for the next run.
288             svm->clear();
289             samples.release();
290             labels.release();
291             class_label.release();
292           }
293
294           cout << "------- test ---------\n";
295
296           // evaluate
297           dir = "/home/lindo/dev/opencv-test/bow/TEST";
298           dp = opendir( dir.c_str() );
299           count = 0;
300
301           while ( (dirp = readdir( dp )) ) {
302             //if (count++ > 15) break;
303
304             filepath = dir + "/" + dirp->d_name;
305
306             // If the file is a directory (or is in some way invalid) we'll skip it
307             if (stat( filepath.c_str(), &filestat )) continue;
308             if (S_ISDIR( filestat.st_mode ))         continue;
309             if (dirp->d_name[0] == '.')              continue; //hidden file!
310
311             cout << "eval file " << filepath << endl;
312
313             img = imread(filepath);
314             keypoints.clear();
315             detector->detectAndCompute( img, noArray(), keypoints, noArray() );
316             bowExtractor->BOWImgDescriptorExtractor::compute(img, keypoints, response_hist); //
    create presence vector
317
318             // test vs. SVMs
319             map<string, Mat>::iterator it;
320             for (it = classes_training_data.begin(); it != classes_training_data.end(); ++it) {
321               string class_ = it->first;
322
323               // read classifier from disk
324               string svmFilename = "/home/lindo/dev/opencv-test/bow/CLASS/class_" + class_ + ".xml";
325               svm = StatModel::load<SVM>( svmFilename );
326
327               float res = svm->predict( response_hist );
328
329               cout << "class: " << class_ << ", response: " << res << endl;
330             }
331           }
332
333           closedir( dp );
```

```
334
335             cout << "done" << endl;
336         return 0;
337     }
```