Untitled

#include "cv.h"   
#include "highgui.h"   
#include "cxcore.h"  
#include <ml.h>   
#include <iostream>   
#include <fstream>   
#include <string>   
#include <vector>   
using namespace std;  
using namespace cv;  
  
  
  
int main(int argc, char\*\* argv)  
{  
 vector<string> img\_path; //this variable will record the names of image files   
 vector<int> img\_catg;  
 int nLine = 0;  
 string buf;  
 ifstream svm\_data("E://forfun/trainlist.txt"); //In .txt, it's a list of all the file names'  
   
 unsigned long n;  
   
 while (svm\_data) //read the train files one by one   
 {  
 if (getline(svm\_data, buf))  
 {  
 nLine++;  
 if (nLine < 1400) //this number is the number of cats training images  
 {  
 img\_catg.push\_back(1); //give the label 1 denotes cat  
 img\_path.push\_back(buf);   
 }  
 else  
 {  
 img\_catg.push\_back(0); // give the label 0 denotes dog  
 img\_path.push\_back(buf);   
 }  
 }  
 }  
 svm\_data.close(); //close file  
   
 CvMat \*data\_mat, \*res\_mat;  
 int nImgNum = nLine; //the number of the files  
 data\_mat = cvCreateMat(nImgNum, 1764, CV\_32FC1); //1764 can be obtained by descriptors.size()   
   
 res\_mat = cvCreateMat(nImgNum, 1, CV\_32FC1);  
 cvSetZero(res\_mat);  
  
 IplImage\* src;  
 IplImage\* trainImg = cvCreateImage(cvSize(64, 64), 8, 3); //one image, give the size 64\*64  
   
 //start to get HOG descriptor  
 for (string::size\_type i = 0; i != img\_path.size(); i++)  
 {  
 src = cvLoadImage(img\_path[i].c\_str(), 1);  
 if (src == NULL)  
 {  
 cout << " can not load the image: " << img\_path[i].c\_str() << endl;  
 continue;  
 }  
  
 cout << " processing " << img\_path[i].c\_str() << endl;  
  
 cvResize(src, trainImg); //read the image   
 HOGDescriptor \*hog = new HOGDescriptor(cvSize(64, 64), cvSize(16, 16), cvSize(8, 8), cvSize(8, 8), 9);   
 //There are 4 parameters:  
 //winSize(64,128): the size of the window   
 //blockSize(16,16): the size of the block  
 //blockStride(8,8): the move length of the block  
 //cellSize(8,8): the size of the cells  
   
 vector<float>descriptors;   
 hog->compute(trainImg, descriptors, Size(1, 1), Size(0, 0));   
 cout << "HOG dims: " << descriptors.size() << endl;   
 n = 0;  
 for (vector<float>::iterator iter = descriptors.begin(); iter != descriptors.end(); iter++)  
 {  
 cvmSet(data\_mat, i, n, \*iter); //save the HOG descriptor   
 n++;  
 }   
 cvmSet(res\_mat, i, 0, img\_catg[i]);  
 cout << " end processing " << img\_path[i].c\_str() << " " << img\_catg[i] << endl;  
 }  
  
  
 CvSVM svm;   
 CvSVMParams param;   
 CvTermCriteria criteria;  
 criteria = cvTermCriteria(CV\_TERMCRIT\_EPS, 1000, FLT\_EPSILON);  
 param = CvSVMParams(CvSVM::C\_SVC, CvSVM::RBF, 10.0, 0.09, 1.0, 10.0, 0.5, 1.0, NULL, criteria);  
   
 //train SVM  
 svm.train(data\_mat, res\_mat, NULL, NULL, param);   
 svm.save("E://test/SVM\_DATA.xml");  
  
   
 IplImage \*test;  
 vector<string> img\_tst\_path;  
 ifstream img\_tst("E://test/testlist.txt"); //The same as reading training files, but we dont need to add labels.  
 while (img\_tst)  
 {  
 if (getline(img\_tst, buf))  
 {  
 img\_tst\_path.push\_back(buf);  
 }  
 }  
 img\_tst.close();  
  
  
  
 CvMat \*test\_hog = cvCreateMat(1, 1764, CV\_32FC1);   
 char line[512];  
 ofstream predict\_txt("E://test/predict.txt");   
 for (string::size\_type j = 0; j != img\_tst\_path.size(); j++)   
 {  
 test = cvLoadImage(img\_tst\_path[j].c\_str(), 1);  
 if (test == NULL)  
 {  
 cout << " can not load the image: " << img\_tst\_path[j].c\_str() << endl;  
 continue;  
 }  
  
 cvZero(trainImg);  
 cvResize(test, trainImg); //read the image   
 HOGDescriptor \*hog = new HOGDescriptor(cvSize(64, 64), cvSize(16, 16), cvSize(8, 8), cvSize(8, 8), 9);   
 vector<float>descriptors;   
 hog->compute(trainImg, descriptors, Size(1, 1), Size(0, 0));   
 cout << "HOG dims: " << descriptors.size() << endl;  
 CvMat\* SVMtrainMat = cvCreateMat(1, descriptors.size(), CV\_32FC1);  
 n = 0;  
 for (vector<float>::iterator iter = descriptors.begin(); iter != descriptors.end(); iter++)  
 {  
 cvmSet(SVMtrainMat, 0, n, \*iter);  
 n++;  
 }  
  
 int ret = svm.predict(SVMtrainMat);   
 std::sprintf(line, "%s %d\r\n", img\_tst\_path[j].c\_str(), ret);  
 predict\_txt << line;  
 }  
 predict\_txt.close();  
  
   
 cvReleaseMat(&data\_mat);  
 cvReleaseMat(&res\_mat);  
  
 return 0;  
}