CS 423/523 Computer Vision
Fall 2014

Project 2 and 3

Due: November 14th, Friday 2014, 23:55 (sharp)

Basic Object Recognition

Description: In this project you will develop an OpenCV program that will run on Caltech 101 dataset. Caltech 101 dataset is a very popular and easy object collection from the year 2004. It includes 101 different object categories, each with different numbers of samples. For instance, wheelchair object has 59 different samples whereas umbrella category has 75 different samples. Each object category has its own folder. Under the category folder, you have *.jpg and *.mat files. *.mat files are annotation files that we are not going to use in this project.

Details:

1) Download from http://www.vision.caltech.edu/Image_Datasets/Caltech101/. Ignore the annotations. You will only need 101_ObjectCategories.tar.gz. Extract the archive.

2) Create a new cpp (C++ source code) file whose name uses the template PRJ2-3-**STUDENTID**.cpp (insert your own student id in the bold part)
   a. You may use any operating system you want (Windows, Linux, MacOS).
   b. The output executable file must be in the format PRJ2-3-**STUDENTID**.exe. *.exe suffix is only required for Windows compiles.
   c. You **must** use CMake for producing the makefile/solutionfile required to compile your project. Hence, you can use any compiler you want (MSVC, GCC, Clang, ...)
   d. You **must** use OpenCV (2.4.*) as the assistant library.
   e. No MATLAB allowed in the project.

3) Put the extracted object categories to the same folder with you programs executable into a folder named DATASET. In other words, the DATASET folder which resides in the same location of your application exe file must contain all the object category folders (i.e. accordion, airplanes, ...).

4) Write the code for traversing all the *.jpg files in all the object category folders under the DATASET folder.

5) Read each *.jpg file and store its cv::Mat in the memory. Remember to assign the folder name with the read cv::Mat file. In other words, given a cv::Mat you should be able to tell the label of it (i.e. umbrella, airplanes, ...).

6) Assume that the first 50% of the images belonging to the same object category to be your training set. The other 50% will therefore be your test (validation) set.

For instance, we have 800 airplane pictures. The first 400 airplanes will be in your training set whereas the last 400 will be considered as your test set.

7) Write "cv::Mat *getHistogram*(cv::Mat *image*, int quantum)" function. This function will extract a color histogram from the supplied input *image* by assuming a color quantization value "*quantum*". For instance, if quantum is given as 1, each histogram bin will represent a different illumination level. If quantum is 4, illuminations of 0-3 is the first bin, 4-8 is 2nd bin, etc. With a quantum of 4, you will have 64 bins per color channel. Also remember that the images are RGB images. So, you need to extract histogram information for each different channel and concatenate them. If you call *getHistogram* function with a color image and quantum = 4, then you will have (256/4)*3 different histogram bins. Put all of them in a cv::Mat of 1 rows and (256/4)*3 columns. This will be the returned cv::Mat from your *getHistogram* function. Also don't forget to normalize your histogram bins. The total of all the bins in your histogram must be 1.0. You must use a float cv::Mat type (CV_32FC1).

8) Extract all the normalized color histograms for all images (both training and test images) using your function and keep them associated with the image it belongs.

9) Now, for all the test images, do the following:
   a. Compare its histogram array gathered using your function to each and every training image in the memory. Remember that you don't actually know the category of the test image in real life. So, when comparing a test image to the training images, you must compare it to all the categories.
   b. For comparing the two histograms use the following formula:

$$\text{histint}(h_i, h_j) = 1 - \sum_{m=1}^{K} \min\big(h_i(m), h_j(m)\big)$$

Histogram intersection (assuming normalized histograms)

   c. After comparing your single test image with all the possible training images from all the categories, store the category, which gives the closest histogram distance.
   d. If the closest category found is matching the original category of the image (remember that you actually know the category of the image, but didn't use it for the test images) then count this as a successful classification.

10) Report your success rate per each object category. For instance, in the "barrel" category we have 47 images. First 23 of them are in the training set. The other 24 of them are in the test set. For all the 24 images in the test set, you checked for the closest histograms in the training images. If the closest histogram's label also belongs to the "barrel" category, then this is a successful classification. Let's assume that 10 of these 24 test images are correctly classified as "barrel". Then you will report a performance of 10/24 = 41.67% for "barrel" category.

11) Now weight your category performances by the number of test images in them and report an overall weighted performance.

12) Change the quantum and do the above again. Use values as 1, 4, 8, 16, 32, and report.

13) **(10% bonus)** Divide each image to 4 equal sized cells in 2x2 grid. Extract color histograms for each cell and create your histograms by concatenating them. In this scenario, your histogram arrays will be 4 times longer. Do the same things as illustrated above, and report your results again. Discuss the performance increase/decrease clearly.

14) **(10% bonus)** Use the Chi-squared Histogram matching distance for comparing histograms. Do the same things again. Report your results and discuss the performance increase/decrease clearly. For the ones who do the 2x2 grid bonus combined with this bonus, you should report the effect of this improvement both on single cell histograms and 2x2 cell histograms.

$$\chi^2(h_i, h_j) = \frac{1}{2}\sum_{m=1}^{K} \frac{[h_i(m) - h_j(m)]^2}{h_i(m) + h_j(m)}$$

Chi-squared Histogram matching distance

15) Your executable is going to be used as below:
PRJ2-3-**STUDENTID**.exe
It will automatically do the above steps and report the performances to the console.

16) Write a short Word document. Paste your results into this document and add your discussions for the required questions above. Also submit this report file along with your Project.

Your code should be clean and easy to read by possessing the following properties;

- *Clean structure:* The overall code should be neatly organized, where the related statements are grouped together with enough spacing among them.
- *Consistent indentation:* Statements should be consistently indented according to the nesting.
- *Appropriate use of comments:* There should be comments explaining what the program, and different groups of statements are supposed to do. Don't overdo it.
- *Meaningful and consistent variable naming:* The names of variables should be meaningful with respect to the purpose and usage of these variables.

IMPORTANT:

Submission: By uploading your whole project file to the LMS as a single ZIP archive. No other methods (e.g., by e-mail) accepted. (You may resubmit as many times as you want until the deadline).

Warning: This homework is an individual assignment. DO NOT SHARE YOUR CODE WITH OTHERS. Your programs are checked and compared against each other using automated tools. Any act of cheating will be punished severely. The code that does not compile will receive 0 points.

Also:

- Name your archive file uploaded exactly as requested. Your archive file must be named as **PRJ2-3-STUDENTID.zip**
- Make sure that your program runs and gives the **expected output**
- The first lines of your code must include your name, surname, student number, and department as a **comment**. An example comment is as follows:
  / John Smith S0001 Department of Computer Science */*
- **Don't include** debug files such as *.obj, *.pch, *.ilk, **or your image, dataset files** in your archive. The single file that can be greater than 100KB in your archive must be the executable file.

Good luck ☺