CS 216: Image Understanding

Homework 4, Spring 2017

Due: a PDF file containing your writeup and result images and a zip file containing your code should be uploaded to the EEE dropbox before 11:59pm on Tuesday 5/30/2017.

Reading: papers on the website and Szeleski Chapter 14

In this assignment, you will develop an object detector based on gradient features and sliding window classification. The class website directory contains the some skeleton code to get you started along with some test images `http://www.ics.uci.edu/ fowlkes/class/cs216/hwk4/downloads/` which should hopefully make the assignment go relatively quickly.

1. **Histograms of Gradient Orientations:** Write a function that computes gradient orientation histograms over disjoint 8x8 block of pixels in an image. Your function should bin the orientation into 9 equal sized bins between -pi/2 and pi/2. The input of your function will be an image of size HxW. The output should be a three-dimensional array `ohist` whose size is (H/8)x(W/8)x9 where `ohist(i,j,k)` contains the count of how many edges of orientation k fell in block (i,j).

   You will want to base this function on your gradient calculation from the previous assignment

   To determine if a pixel is an edge, we need to choose some threshold. I suggest using a threshold that is a tenth the maximum magnitude in the image (i.e. `thresh = 0.1*max(mag(:))`). Since each block will contain a different number of edges, you should normalize the resulting histogram so that `sum(ohist,3)` is 1 everywhere.

   I would suggest your function loops over the orientation bins. For each orientation bin you'll need to identify those pixels in the image whose magnitude is above the threshold and whose orientation falls in the given bin. You can do this easily in MATLAB using logical operations in order to generate an array the same size as the image that contains 1s at the locations of every edge pixel that falls in the given orientation bin and is above threshold. To collect up pixels in each 8x8 spatial block you can use the function `im2col(..,[8 8],'distinct')`. The im2col function will automatically pad out the image to a multiple of 8 which is convenient.

2. **Detection:** Write a function that takes a template and an image and returns the top detections found in the image. Your function should have the prototype:

   ```
   [x,y,score] = detect(I,template,ndet)
   ```

   where `ndet` is the number of detections to return. In your function you should first compute the histogram-of-gradient-orientation feature map for the image, then correlate the template with the feature map. Since the feature map and template are both three dimensional, you will want to filter each orientation separately and then sum up the results to get the final response. This final response map will be of size (H/8)x(W/8).

   When constructing the list of top detections, your code should implement non-maxima suppression. You can do this by sorting the responses in descending order of their score. Every time you add a detection to the list to return, check to make sure that the location of this detection is not too close (e.g., more that 50% overlap) to any of the detections already in the output list.

   Your code should return the locations of the detections in terms of the original image pixel coordinates so if your detector had a high response at block (i,j) then you should return (8*i,8*j) as the pixel coordinates.

   Test your detection code using the provided script, modifying it as necessary. The script loads in a training image and a test image. You can click on a patch of the training image. The script then builds a template using the histogram feature map. Finally the script calls your detect function using this average template to detect objects in the test image.

3. **Multi-scale Detection:** Write a function `multiscale_detect` that extends the detector to handle multiple scales by (1) generating an image pyramid from the original input, (2) repeatedly calling your detect function on each level of the pyramid, (3) combining the results across different scales to yield a final set of detections.

   For the image pyramid, you may find it useful to re-scale the image by less than a factor of 0.5 at each layer. For example, you may want to try scaling by 0.7 in order to search over a finer range of scales. You can use MATLABs function `imresize` which will take care of pre-filtering (anti-aliasing) and resizing the image. Your code should

stop generating smaller versions of the image as soon as the resulting image size gets to be smaller than your template.

Note that when combining detections from different scales, you will have to scale the x,y coordinates returned by detect to be correct for the original un-scaled image. You will also need to do some non-max suppression when combining results in order to remove overlapping detections from different scales.

Demonstrate your multiscale detection code by showing the detector output on an image that contains multiple instances of an object at several different scales where the results should show appropriately scaled bounding boxes around each object detected (i.e. if an object is detected at an image scaled by 0.5 it should have twice the height and width as one detected at the original image scale).

4. **Learning Templates:** Write an improved script `detect_script.m` that allows the user to train the detector. You do not need to build a fancy user interface (e.g. you can leave paths hard-coded, etc.) but there are two key functional improvements you should implement

   (1) Your improved script should let the user mark positive examples from several different images. The script provided for building the template makes a hard-coded assumption that the positive training example is 128x128 pixels. This doesn't work for training with multiple examples since they may be different sizes. Instead, use the `getrect` function in MATLAB to allow the user to draw a box around each positive example in one or more images. You should extract the image patch specified by the user marked rectangles. Once you have extracted the patches for all the positive training examples you will want to resize them to all be the same size.

   To determine the target size we should resize them to we have a few requirements (a) the aspect ratio should remain as close as possible to the original aspect ratio marked by the user (b) the resolution should also be close to the original so that the resized examples have a height and width which is roughly the average of the training examples (c) the resized positive examples should have a height and width which are multiples of 8 so that they align nicely with the histogram bins. I've provided a function `average_boxes` which takes a bunch of bounding boxes and estimates a target size that satisfies these properties. Feel free to use and modify it as you see fit.

(2) Once you have extracted the positive examples, determined the template size and resized them, you can build an initial template by computing the HOG descriptors for each (resized) example and then averaging them together to get an average positive template. One difficulty is that this template may not be particularly discriminative (e.g. it may fire on some background regions in your images). In order to improve it, take some random image patches of the same size from negative training images which you know do not contain your object and compute their average HOG descriptor. Finally, construct a template which is a differenced of these two averages, i.e.

$$T_{final} = \frac{1}{N_{pos}} \sum_{i \in pos} T_i - \frac{1}{N_{neg}} \sum_{i \in neg} T_i$$

Where T_i is the array containing the extracted HOG descriptor for a positive or negative example.

5. **Writeup:** In addition to your code, please include a writeup which includes the following figures (with appropriate captions to identify them).

   (a) Single scale detector output
      i. template extracted from 1 positive example
      ii. template which is average of 5 positive examples
      iii. template which is average of 5 positive examples minus average of 100 negative examples

   (b) Multi-scale detector output using whichever template works the best

   You should show these results on two different test images so you have 4+4=8 figures depicting detection results. In each case show the top 5 detections. Finally, please include a figure which shows the 5 positive training examples used to build your template. You may find the `montage` function useful for this step.

   You are welcome to choose whatever object you would like to build a detector for (faces, stop signs, dilbert, waldo, etc.) but please make sure that the results you demonstrate the multiscale detector well (i.e. they actually show detections at multiple scales).

   Finally, please write a few sentences comparing this detector to the detector we experimented with in the first assignment. In particular, we

previously found that simply cross-correlating image patches did not make for a good detector (we had to use SSD or some other method). Why are we able to use cross-correlation here? What is different?