16-720 Computer Vision: Homework 5
Image Understanding

Instructor: Deva Ramanan
TAs: Allie Del Giorno, Jai Prakash Esha Uboweja, Guanhang Wu

Due: Refer to course website

Make sure to start early!

1 Image detection using Histogram of Gradients

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://16720.courses.cs.cmu.edu/hw/hw5.zip`

1.1 Image Gradients: [5 Points]

Write a function that takes a grayscale image as input and returns two arrays the same size as the image, the first of which contains the magnitude of the image gradient at each pixel and the second containing the orientation.

```
[mag,ori] = mygradient(I)
```

Your function should filter the image with the simple x- and y-derivative filters described in class. Once you have the derivatives you can compute the orientation and magnitude of the gradient vector at each pixel. You could use imfilter with the 'replicate' option in order to nicely handle the image boundaries.

In your writeup, include `imagesc` visualization of magnitude and orientation of the gradient of any two test images.

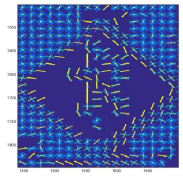
1.2 Histograms of Gradient Orientations: [20 points]

Write a function that computes gradient orientation histograms over each 8x8 block of pixels. 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)`.

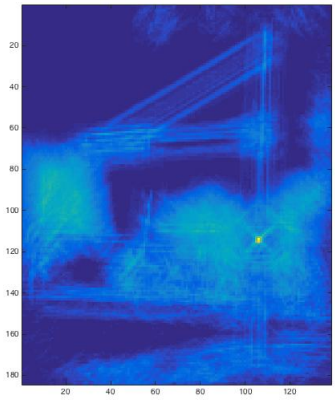
```
ohist = hog(I)
```

To determine if a pixel is an edge, we need to choose some threshold. We suggest using a threshold that is a tenth the maximum magnitude in the image (i.e. `thresh`

(a) The template



(b) visualization of HOG features with the attached script `hogdraw.m`



(c) Heatmap showing the cross-correlation with the template

(d) Detection of the template

Figure 1: Detection using HOG features

`= 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.

We 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.

In your writeup, include HOG feature visualization of the test images used in previous sub-section.

1.3 Detection: [20 points]

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** (NMS). 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 (atleast a bounding-box away) 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.

Putting it all together

Test your detection code using the provided script (`detect_script.m`) and 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.

In your writeup, show the heatmap visualization of correlation and detected window of all the test images.

1.4 Extra Credit - Multiple detections: [5 Points]

In your writeup, show an example of multiple detection (around 2-5 distinct instances) of the template in the same image. Find some image from internet and show the heatmap and the detection.

Also report examples with multiple templates as input and show multiple detections. Play around with `nClick` and `ndet` in `detect_script.m` for this task.

2 Learning Templates

2.1 Select Patches: [5 Points]

Write an script or function called `select_patches.m` that allows the user to select patches. In the end of this script, you should save the patches, which will be used for training purpose. Your function should let the user mark 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. We have a couple requirements on this resizing operation (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. After resizing, you need to save the patches in the signature we give you. After you selected the patches, please follow the pipeline we gave you: `tl_detect_script`. It will guide you to complete all the functions.

2.2 Positive Template Learning: [5 Points]

For positive template learning, you need to complete the function `tl_pos.m`. Basically, it computes the HOG descriptors for each positive patches and then averages them together to get an average positive template.

2.3 Positive Negative Template Learning: [5 Points]

For this section, you need to complete `tl_pos_neg.m` From last example, you may see that the positive 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.

$$T_{final} = \mu_{pos} - \mu_{neg} \quad \text{where} \quad \mu_{set} = \frac{1}{|set|} \sum_{i \in set} T_i$$

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

2.4 LDA template: [20 Points]

You will also explore template learned through a simple variant of linear discriminant analsis (LDA), as described in this paper `http://www.ics.uci.edu/~dramanan/`

`papers/who.pdf`. The final template will be

$$T_{final} = \Sigma_{neg}^{-1}(\mu_{pos} - \mu_{neg}) \quad \text{where} \quad \Sigma_{set} = \frac{1}{|set|} \sum_{i \in set} (T_i - \mu_{set})(T_i - \mu_{set})^T + \lambda I_d$$

where the sample covariance matrix Σ is estimated on a large negative set. Because it will likely be low-rank, you will need to regularize it (by adding λ to the diagonal entries) in order to compute the inverse. Experiment with different λ. Please complete `tl_lda.m` for this section.

2.5 Multi-scale Detection: [20 Points]

Write a function `multiscale_detect.m` which 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).

2.6 Extra Credit Mixture of templates: [10 Points]

Select an object category with appearance variation (like a cat face, hand, cup, or chair). Rather than learning a single average template, learn a collection of templates (tuned for say, different viewpoints). For example, you may learn a frontal face template and a side face template. You must use at least 2 mixtures. Use this mixture of templates as an object detector, using whatever learning strategy worked the best above to learn each template. Make sure your NMS procedure competes templates across different mixtures. To make the scores of different templates comparable, you may have to consider different template scoring functions, including sum-of-squared differences or normalized correlation. Find an image where

the mixture model performs better than the single-template model and explain why.

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

- Single scale detector output
- trained with 1 positive example
- trained with average of 5 positive examples
- trained with average of 5 positive examples + 100 negative examples
- trained with LDA using 5 positive examples + 100 negative examples
- Multi-scale detector output using which ever combination of learning performed the best.

You should show these results on two different test images so you have 5(6 if extra credit)+5=10 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 explaining why you think certain learning strategies multi-scale modeling and/or mixture modeling worked well or didn't work. Also compare your best detector to the one you explored in your earlier assignment, based on pixel values. What makes the current detector in this assignment work better?

Submission guidelines

If your andrew id is bovik, your submission should be a zip file **bovik.zip** containing a folder bovik/. This should contain

- bovik.pdf : A write-up containing the required explanations and results for various tasks.
- code/ : Folder containing Matlab files with your code
- data/ : Folder containing the *.mat files and images used for the assignment