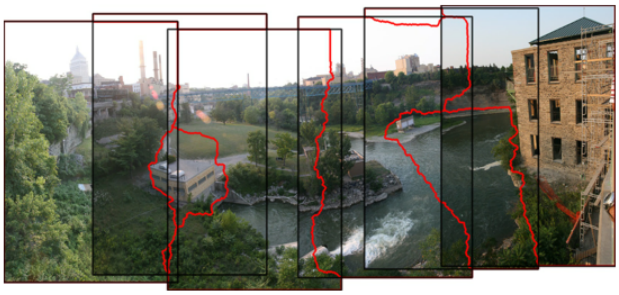
Computer Vision by Learning Assignments Day 2

March 26^{th} 2014

Laurens van der Maaten

Computer Vision Laboratory, Delft University of Technology



TU Delft

1 Pedestrian Detection

Exercise 1.1. Training and using a Dalal-Triggs detector

In this exercise, you will train and evaluate a Dalal-Triggs detector for pedestrian detection. The folder `images` contains three folders: a folder with `positive` training examples, a folder with `negative` training examples, and a folder with `test` images.

Inspect some of the training images. Why do the positive images contain two versions of the same image? What type of invariance does this introduce in the detector? An why do you think the collection of negative images is so much larger than that of positive images?

Load one of the training images and extracts histogram-of-oriented gradient (HOG) features using a block size of 8 pixels:

```
>> block_size = 8;
>> im = im2double(imread('images/positive/crop_000010a.png'));
>> features = hog(im, block_size);
```

(You may need to do `mex -O hog.cc` first for this to work.) You can visualize the resulting HOG features using the following command:

```
>> imshow(visualizeHOG(features));
```

Run the code snippet above on a number of positive images. Do you recognize the pedestrians in the HOG features?

The function `load_hog_images` loads in all the training images, extracts HOG features from these images, and stores the resulting feature representations in a PRTools data set. It also returns the original size of the HOG feature matrices (before they were concatenated in a single feature vector). Build a HOG pedestrian-detection training set:

```
>> pos_folder = 'images/positive/';
>> neg_folder = 'images/negative/';
>> pos_files = dir([pos_folder '*.png']);
>> neg_files = dir([neg_folder '*.png']);
>> [A, hog_size] = load_hog_images(pos_folder, pos_files, ...
                                   neg_folder, neg_files, block_size);
```

This may take a few minutes to complete depending on the speed of your processor and network connection. If you encounter memory problems during this step, use a smaller, random subset of the negative examples in `neg_files` as input.

Next, train an L2-regularized linear logistic regressor on the pedestrian data set. Use a value of 0.1 for the L2-regularization parameter λ:

```
>> lambda = 0.1;
>> W = loglc2(A, lambda);
```

Why is it important to use L2-regularization when training the pedestrian detector? What is the size of weight matrix learned by the logistic regressor? Why this size?

Evaluate the classification error of your logistic regressor via 5-fold cross-validation:

```
>> err = crossval(A, loglc2([], lambda), 5)
```

Do you think this error is sufficiently low to get a good pedestrian detector?

The weights learned by the logistic regressor can be visualized as a HOG image as follows:

```
>> imshow(visualizeHOG(reshape(W.data.E(:,2), hog_size)));
```

What has the logistic regressor learned?

The function `sliding_window_detector` applies the trained pedestrian classifier to a window that is slid over the image at multiple scales, and performs non-maxima suppression to make the final pedestrian detections. Why is it necessary to run the detector at multiple scales? Why do we need to perform non-maxima suppression?

Apply the pedestrian detector to the test images. The sliding window implementation is rather naive: it may take about a minute to perform the detection on a single image.

```
>> % Loop over test images
>> test_files = dir('images/test/*.png');
>> for j=1:length(test_files)
>>     % Load image
>>     im = im2double(imread(['images/test/' test_files(j).name]));
>>     [boxes, response] = sliding_window_detector(im, W, ...
                                          block_size, hog_size);
>>     % Plot results
>>     subplot(1, 2, 1); imagesc(response, [0 1]);
>>     colormap jet; axis equal tight off;
>>     subplot(1, 2, 2); imshow(im); hold on
>>     for i=1:size(boxes, 1)
>>         rectangle('Position', [boxes(i, 2)   boxes(i, 1) ...
>>                                boxes(i, 4) - boxes(i, 2) ...
>>                                boxes(i, 3) - boxes(i, 1)], ...
>>                           'LineWidth', 2, 'EdgeColor', [1 0 0]);
>>     end
>>     hold off; pause
>> end
```

The above code snippet shows the response surface in the left sub-image, and the detections in the right sub-image. What are false positives in the detections? And false negatives?

2 Pedestrian Detection using Pictorial Structures

Exercise 2.1. Using a Felzenszwalb detector

In this exercise, you will evaluate a Felzenszwalb detector for pedestrian detection. Because training a latent SVM is computationally very expensive, we will focus on the evaluation of the detector in this exercise.

Initialize the pictorial-structures detection software:

```
>> startup;
```

Load a pretrained pedestrian classifier:

```
>> load('VOC2007/person_grammar_final');
>> model.class = 'person grammar';
>> model.vis = @() visualize_person_grammar_model(model, 6);
```

You can visualize the pedestrian detector as follows:

```
>> model.vis();
```

What information about pedestrians does the root filter capture? And the part filters? Why are the HOG cells of the part filters smaller in the visualization?

The folder `test_images` contains the same test images as were used in the previous exercise. Load an image and apply the pedestrian detector to the image:

```
>> % Loop over test images
>> test_files = dir('test_images/*.png');
>> for j=1:length(test_files)
>>    % Load image
>>    im = imread(['test_images/' test_files(j).name]);
>>    % Apply pedestrian detector
>>    thresh = -0.6;
>>    [ds, bs] = imgdetect(im, model, thresh);
>>    if ~isempty(ds)
>>        top = nms(ds, 0.5);
>>        bs = [ds(:,1:4) bs];
>>        showboxes(im, reduceboxes(model, bs(top,:)));
>>    else
>>        showboxes(im, []);
>>    end
>>    pause
>> end
```

What do the blue boxes inside the red detection boxes mean? How are the locations of these blue boxes relative to the red boxes determined? And the sizes / aspect ratios?

Perceptually, does the performance of the pictorial-structures detector seem better or worse than that of the Dalal-Triggs detector? How could you measure which detector works better?