Object Detection using Sliding-Window

Jonathan Dyssel Stets Henrik Aanæs

May 6, 2016

Figure 1: Parking Signs on DTU Campus

This exercise will teach you how to do object detection in images using a sliding-window approach.
Object detection is the task of finding one or more specific objects in an image or video. This can be real-world things sunch as cars, bikes, signs, buildings, consumer products etc. Detection of faces and people is also known as object detection.
The sliding-window detector is one way to identify and localize specific objects in images. As the name suggests, a sliding-window is a window or kernell of a given size that moves across an image. For every position the window makes a stop to performs local statistics to determine if a given object is present. You will implement a simple sliding-window detector that should be able to detect round road signs in traffic images, as seen in fig. 1.
The purpose of this exercise is to teach you how a sliding-window detector works and how it could be implemented. Therefore, performance and robustness of the detector in this exercise will not be nearly as good as todays standards.

1 Simple Sliding Window

Begin by implementing a loop that slides a window over an image. The window should begin in the top left corner of the image and slide towards right. Once it has reached the far right side of the image, it should step down and start from left to right again. This should be repeated untill the window has reached the bottom right corner of the image, just like an old fashioned typewriter.
You should be able to control the step size of the sliding-window in the x- and y-direction, and be able to control the size of the window. The window coordinate should be defined as its top left corner, see fig. 2.
Note; depending on the window size, image size and step size, your sliding-window may not reach the pixels on the right and bottom edges of the image. You can ignore this for now.

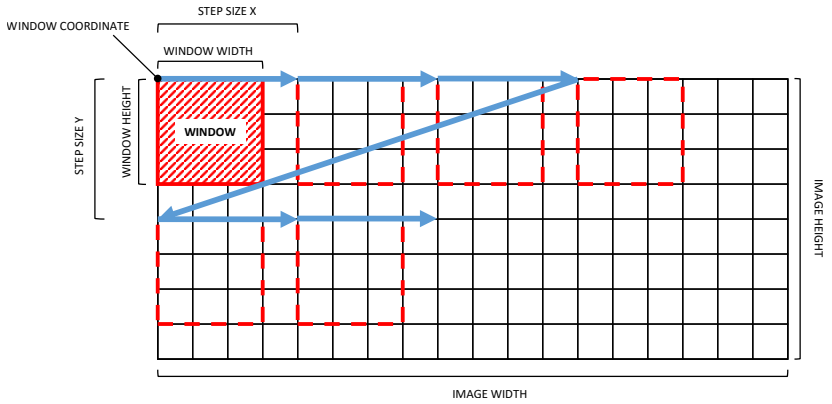

Figure 2: Sliding-window illustration.

EXERCISE 1 Finish the for-loops in the codesample below to realise a sliding-window as described above. You should visualize the window by drawing a rectangle on the image and pause between each step, so that the window slide can be activated with the [ENTER] key on the keyboard. In the code below, the loaded image is downscaled. Use this scaling for now.

```
im = imread('data/dturoad1.jpg');
im = imresize(im,[240 320]); % set size of image

figure; imagesc(im); hold on;
wsize = [30 30]; %windowsize
stepx = 20; %stepsize x-direction
stepy = 20; %stepsize y-direction

for row = %?
```

```
    for col = %?

      % draw a rectangle on the image:
    rectangle('Position',[col row wsize(1) wsize(2)]);


        % wait for keypress:
        pause
    end
end
```

EXERCISE 2 Run the code, and try to change the window size and stepsizes. Explain what is happening when you change these parameters.

EXERCISE 3 Expand the code, so that the window is extracted from the full image (`im`) and stored in a variable; `window = im(?:?,?:?)`. Verify that you are extracting the correct image patch by displaying it in another figure: `imagesc(window)`.

2 HSV Histogram Matching

You have now realised a simple sliding-window. Next, we wish to find a given object in the image using local statistics. We will compare each window with an object-image using their histograms in the HSV-space (Hue Saturation Value).

EXERCISE 4 Load the image of the object and convert it to HSV. HSV is an alternative colorspace to RGB. After converting the image, it still consists of three channels, now these are; Hue, Saturation and Value. If you are not familiar with HSV, please take a moment to read about it. Also, set the `wsize` (windowsize) to the same size as the object-image.

```
% Object Image:
imobj = imread('data/sign1_64.jpg');
```

EXERCISE 5 Inspect the histograms of the HSV channels, and explain what you see. How does the channels provide information of the appearance of the object?

EXERCISE 6 Store the histograms as vectors of bin counts (256 bins); `h_hist = histcounts(hsvimage(:,:,1),256)`. Remember to normalize the histograms; `h_hist = h_hist/sum(h_hist)`.

EXERCISE 7 Expand your code, so that the H,S and V histogram-vector are calulated for a window in the for-loop (same procedure as above). Select 1-3 of the channels/histograms you wish to compare. Calculate the distance between the selected channels/histograms of the object-image image and the window in the for-loop. Hint: use `pdist2`, beware of the orientation of the histrogram data.

EXERCISE 8 Run your code, and inspect the distances as the window is sliding across the image. How do they change when the window reaches the object you are searching for? Experiment with the channels you chose to compare.

We use the distance scores to determine how well the object we are searching for is matching a given window.

EXERCISE 9 Plot the window with the lowest distance score (best match) on top of the image. Additionally you can try plotting the 2nd, 3rd, 4th, 5th... lowest score to see what other windows are good matches. Comment on the results.
Optinal: Try to select a threshold and apply the sliding window detector on another image.

EXERCISE 10 Try changing the step size. What is the result of changing the stepsize?

You have now implemented a simple sliding-window object detector. Obviously, there are many things you could do to improve the code performance. In our case, we are only able to detect one type of object using a HSV histogram comparrison. It is possible to use other methods to classify images dependent on the type of objects we wish to detect.

EXERCISE 12 Explain when the HSV colorspace can be a better way than RGB for comparing. Think of other methods to classify objects for the sliding-window detector. How could you detect more than one type of object?

3 Object Detection Using HOG

We have now compared images using their appearance in the colorspace. Another, and oftentimes, more robust approach would be to compare objects on their shape appearence. The HOG (Histogram of Oriented Gradients) features uses gradients or edges in the image. If you visualize the HOG features, you may even be able to recognize objects with your own eyes.

EXERCISE 13 In a clean Matlab script-file, try to extract and visualize HOG features of any image using matlabs function `extractHOGFeatures()`.

The function returns the feature vector, and a plot-ready object. Explain what you see in the HOG plot.

```
im = imread('...');

[featureVector, visualization] = extractHOGFeatures(im);
figure;
imagesc(im); hold on;
plot(visulization);
```

EXERCISE 14 Load the following data, and try plotting HOG features on the images. Can you recognize the object in the query image by inspecting the HOG features? How large are the HOG cells, and how can you see that in the HOG plot?

```
% Query Image:
im = imread('data/dturoad1.jpg');
im = imresize(im,[240 320]); % set size of image

% Object Image:
imobj = imread('data/sign1_64.jpg');
```

EXERCISE 15 Create a new sliding-window scripts just like in the previous exercise. This time, use the HOG feature-vector (histogram) to compare the window and the object image. It is recommended that you resize the image, so that your window is approximately the same size as the sign in the image. Again, use `im = imresize(im,[240 320]);`.
Your new object detector should be more precise than the previous (HSV detector).

EXERCISE 15 Explain the strenghts of using HOG for detecting the type of object we are working with. In which ways are the HOG more robust compared to HSV?

4 Multi-Scale Detection

As you may have realised, the detector is most likely to detect objects that have the same size as the sliding-window. In other words, we have only searched for objects in one scale. Open the provided script; `slidingWindow.m`. Download VL_Feat [0] and extract it into your MATLAB directory (you may already have used VL_Feat earlier to find SIFT features).
The `slidingWindow.m` script and `compareHog.m` is doing exactly the same

[0] http://www.vlfeat.org/download.html

as you have done previously. It extracts HOG (now using VL_Feat), and slides a window across the image (now using `blockproc()`). Have a look at what the script is doing and understand what is happening.

EXERCISE 16 Try to detect objects on other scales. Change the `scalesize` parameter to find objects on other scales. Comment on what is happening when you tune this parameter.

Exercise 17 Modify the code so that you can detect as many objects as possible in the image.

EXERCISE 18 Load another query image and see if your detector is able to detect objects in this image.

EXERCISE 19 (Optional) Improve the robustness of your detector by adding more object images. You can load multiple object-images for training, by adding them in a stack, and calculate the mean HOG. See example below:

```
%Load images:
im_objs(:,:,:,n) = %...
% Calculate HOG on image stack:
for i = 1:size(im_objs,4)
        hog_objs(:,:,:,i) =...
         vl_hog(single(im_objs(:,:,:,i)), cellSize) ;
end
% Find mean of HOG stack:
hog_obj = mean(hog_objs, 4) ;
```