Exercises in Tracking & Detection

In this exercise we will implement a random forest for Hough voting. Luckily the training of the forest has been done already and you will only need to implement the testing. The forest has been trained to locate the center of airplanes from the Pascal VOC 2012 Dataset.

Exercise 1 Loading the forest

The forest consists of 10 trees, each conveniently saved in a text-based format. One file per tree. Download them on the website.

- The first line of the file tells you how many internal nodes there are. We will call this number n. Then, the next n lines have to following format: $i\ c_L\ c_R\ t\ x_0\ y_0\ z_0\ x_1\ y_1\ z_1\ s$.
- i is the id of the node. c_L and c_R denote the id of the node's children. If $c_R < 1$ (or c_L respectively) then a leaf node with id $|c_R|$ is reached. t is the threshold. x_i, y_i, z_i with $i \in \{0, 1\}$ denotes the first $i = 0$ or second $i = 1$ box feature center location offset, with z_i selecting the color channel. Finally s is the size of the two boxes.
- after n nodes the next line contains m the number of leaves. Again the next m lines have the form $j\ p_x\ p_y$. j is the ID of the leaf node and $p_x\ p_y$ is the mean vote offset stored in that leaf.
- write a Matlab script that reads and stores a tree from such a file.

Exercise 2 Integral Images

To evaluate the Haar-Like features, we will need integral images. Write a function that takes a standard RGB image and computes three integral images, one for each color channel. Do not normalize the color values in any way, otherwise they do not fit the thresholds stored in the trees!

Exercise 3 Testing

Finally, it is time to run the forest on an image. The nodes inside the tree perform feature tests $f(x, y, t, x_0, y_0, z_0, x_1, y_1, z_1) = b(x + x_0, y + y_0, z_0, s) - b(x + x_1, y + y_1, z_1, s) < t$. Here (x, y) is the current pixel's location in the image. Comparing two box evaluations with a threshold. The box evaluation runs on the integral image and needs to access four values to compute the sum of all intensities in the box centered at $(x + x_i, y + y_i)$ with size s and on channel z_i. That mean, that you compute the average of all intensities in the square defined by $(x + x_i - s, y + y_i - s)$ and $(x + x_i + s, y + y_i + s)$. If $f < 0$, proceed to the left child c_L, otherwise to the right child c_R.

Once you reach a leaf the you get a prediction in the form of an offset vector (p_x, p_y). Compute the mean of the predictions from all the trees to have a final vote for the current pixel (x, y).

Create a heat-map of all predictions from all the pixels in the image. If everything is working correctly most votes should accumulate in the center of the object. Visualize the heat map and the maximum. Once you find the maximum also visualize all pixels that voted for this location.

Frequently Asked Questions

- **The color channel indiced range from 0 to 2, what do we do?** The Mapping from the index to RGB is as follows: 0:B 1:G 2:R. Is you load an image in MATLAB, the order of channels is RGB not BGR.
- **Should we compute the average or just the sum of the pixels inside the boxes?** Compute the average.
- **How do we compute the sum with integral images?** Suppose your integral image is called I. Compute the feature as follows: $b(x+x_i, y+y_i, z_i, s) = \frac{1}{(2s+1)^2}[I(x+x_i+s, y+y_i+s) - I(x+x_i-s, y+y_i+s) - I(x+x_i+s, y+y_i-s) + I(x+x_i-s, y+y_i-s)]$
- **What do we do if a box extends over the image boundaries?** Just assume the image is padded with zeros. That is the same as truncating the box at image boundaries. Since you do note compute the average just sum the pixels in the smaller box as well.
- **How exactly do we choose to go left or right?** If $b(x+x_0, y+y_0, z_0, s) - b(x+x_1, y+y_1, z_1, s) < t$ go left otherwise right.
- **Hint**: Be careful how you index images in MATLAB. The first index is for the rows (y) and the second for columns (x). In the trees however, x is to the right and y downwards.
- **Hint**: Simple function calling in MATLAB creates a copy of all variables that are passed to the function. This results in an extreme slow down (for some people days instead of minutes) for example when you copy the integral image every time you evaluate a feature. A moderately efficient implementation (in MATLAB) should not take longer than 2 minutes to compute the heatmap for the image.