DD2423 Image Analysis and Computer Vision

Lab 2: Edge Detection & Hough Transform

Dario Del Gaizo, Giulia Rebay, Isabella Rositi

1 December, 2022

Question 1: What do you expect the results to look like and why? Compare the size of dxtools with the size of tools. Why are these sizes different?

When computing the first order derivative of an image, whatever the direction, discontinuities will be captured and will appear like the only relevant features in the final convolved image. This is clearly shown in the figures above, where only edges appear clearly in both x and y cases. It is interesting to note that edges are actually shaded differently in the two derivative images. This can be easily explained. In fact, vertical edges appear more clearly in the x derivative image since a vertical edge will result in bigger discontinuities when moving horizontally. The opposite will happen when looking at a horizontal edge, which is why shading is almost opposite in the y derivative image. Being the derivative calculated on every pixel apart from the edges, the size of the derivative is going to be (N-2)x(N-2) w.r.t. the original image.

Question 2: Is it easy to find a threshold that results in thin edges? Explain why or why not!

By comparing the plots above, it is quite clear that finding the best fitting threshold for thin edges isn't always that straight-forward. This is particularly true when smoothing is applied to the image. If we just restrict to the original image, however, we can see that different threshold values will lead us to considering edges or not some parts of the image. Setting an adequate threshold is very important, since it allows us to discard unwanted noise which the Laplacian might be picking up on and to purely focus on actual edges. This is due to the fact that zero-crossings of the Laplacian can respond to false edges. This situation is depicted quite well when setting t = 10 in the original image: here the analysis of the image picks up on several image details which don't really correspond to an edge in practice (take a look, for example, at the facade of the house, which appears to be spotted despite there not being any edges there). This indicates that $t = 10$ is too low. On the contrary, when the threshold value is too high, we lose information on real edges, which will not appear on screen. This can already be seen when looking at the top border of the roof for $t = 30$.

Question 3: Does smoothing the image help to find edges?

Smoothing results in more confused features and less defined edges. However, it also allows us to (pretty much) completely remove noise from the image and only pick up on features that actually correspond to real edges. Therefore, while smoothing does result in more approximated and unrefined conclusions, it also allows us to bypass the "false edges" problem. In practice, when smoothing isn't excessive (cfr. $\sigma = 10$, too high!). It does help us detect the most prominent edges and features in the image, without losing too much time in small detail depiction. The best results are achieved when picking a smaller variance value for the Gaussian kernel ($\sigma = 2$) and setting the threshold to $t = 10$. However, smoothing will generally result in thicker edges.

Question 4: What can you observe? Provide an explanation based on the generated images.

When increasing the scale value, we are increasing the smoothing effect the gaussian kernel has on the image. This will result in blurrier features, less defined edges and therefore more unrefined results. Such a trend can be clearly seen in the set of images above. Starting from the far left, where the scale value is very low and set to 0.0001, we can see how details in the image get lost as the scale value increases, until very little information of the original representation is left once scale gets up to 64. We are clearly looking at a trade-off situation, between detailed (yet probably noisy) representation of edges on the left and noise-free yet incomplete depiction on the right. The optimal scale value is therefore between 1 and 4, where accurate depiction of features and edges is still visible yet we are not excessively tuning to noise and non-existent edges.

Question 5: Assemble the results of the experiment above into an illustrative collage with the subplot command. Which are your observations and conclusions?

Several considerations can be made by looking at the plot above. We can first of all appreciate the effect the scale parameter has on the features of the image, which we have already discussed in some cells above. What is even more interesting to point out is that plotting both $L_{vv} = 0$ and $L_{vvv} < 0$ conditions really does help us in clearly distinguishing generic features - and their noisy counterparts - and actual thin edges. The best results are achieved with a scale value of 1, where noise is suppressed sufficiently but a good level of granularity is left, enabling us to clearly distinguish details in the image. It is also interesting to see how smoothing affects the third derivative more than it does the second. This can be seen by looking at how thicker edges get in the second row of the plot. As the scale parameter increases, features get blurred together and edges become thicker and more blunt. We therefore lose specificity and precision. We can also see that in both rows of the plot, thus in both order derivative cases, the areas which stand out are indeed the same, which shows us once again how both conditions work toward the same goal of providing strong and consistent mathematical tools for edge detection. The first row carries more detail since the second derivative is zero also in the case of minima, which are completely disregarded and filtered out by the condition on the sign of the third order derivative.

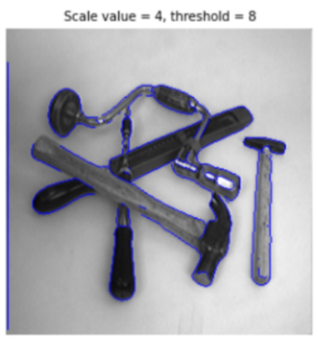
Question 6: How can you use the response from L_{vv} to detect edges, and how can you improve the result by using L_{vvv}?

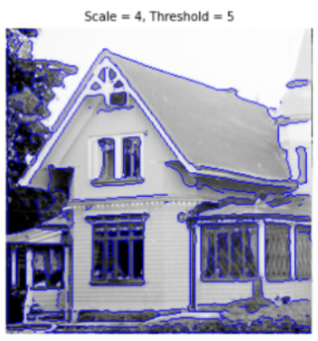
As we have briefly explained in the previous answer, by zooming into the points that satisfy the condition $\tilde{L}_{vv} = 0$ we are able to find both maxima and minima of the gradient magnitude. Since edges always correspond to the most abrupt variations in pixel values, we look for edges where the gradient magnitude is maximal. This is done by filtering out the minima, thus by adding to the previous condition also a sign condition on the third order derivative, $\tilde{L}_{vvv} < 0$. This will improve the result as it will allow us to only focus on points that satisfy all conditions which have to be met in order for a point to belong to an edge. Clearly, $\tilde{L}_{vv}$ on its own also gives us very insightful information, but the mathematical conditions that being an edge involves are only fully described when both $\tilde{L}_{vv}$ and $\tilde{L}_{vvv}$ are used.

Question 7: Present your best results obtained with *extractedge* for house and tools.

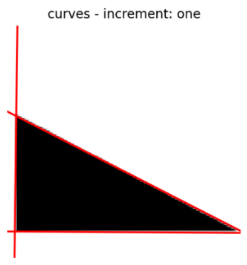
scale = 4 with threshold = 5 for the house image

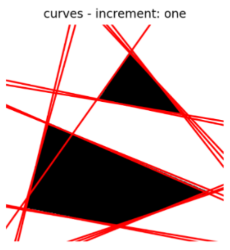
scale = 4 with threshold = 8 for the tools image

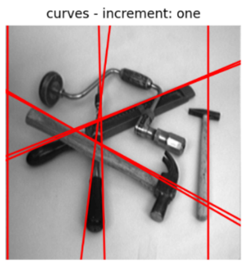




Question 8: Identify the correspondences between the strongest peaks in the accumulator and line segments in the output image. Doing so, convince yourself that the implementation is correct. Summarize the results in one or more figures.







We tested the *houghedgeline()* function on the two test images and indeed, got satisfactory results. In the first image we can see that 3 lines are enough to describe the triangle perfectly. In the second image, we need more *nlines* (15) in order to be able to detect edges for both shapes. The smaller shape has lines that receive less votes, so we need to have a high value for *nlines* in order to be able to extract them.

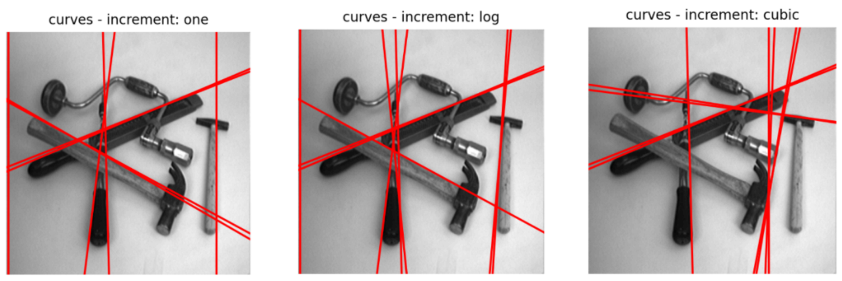
Question 9: How do the results and computational time depend on the number of cells in the accumulator?

By decreasing the number of cells in both the theta and rho accumulators we will end up with a grainier texture when printing the "acc" matrix (we will have a smaller set of available values of rho and theta to compute the Hough transform of the image). This equates to overall less precise results, which will however require less CPU time. If instead we increase the size of the accumulators, we will have better results which will on average require more time to be obtained. Clearly, increasing the parameters will increase the matrix dimensions, which makes it very clear why more computational time is required, since more computations have to be performed. We have also observed that increasing theta impacts computations more since the algorithm can explore more directions.

Question 10: How do you propose to do this? Try out a function that you would suggest and see if it improves the results. Does it?

We tried two different methodologies to define the accumulator increment according to the gradient magnitude:

- log: instead of increasing the accumulator by one, we increment it by taking its natural logarithm
- cubic: instead of increasing the accumulator by one, we increment it by taking its the cube



These approaches will result in different importance attributed to the detected edges. When taking the cube, lines associated with more evident edges (i.e. higher gradient magnitudes) will increase in importance with respect to the base case (increment = 1).

By looking at the structure of the function used for the increment, we can give more importance to higher gradient magnitudes and, for instance, take more dark lines with increment = "cubic", as their gradient will be increased by power = 3. In practice, this will result in a different definition of importance weights, with respect to the uniform weights we had in the case of increment = 1.