Australian National University

Assignment 1 Report

ENGN**4528** Computer Vision

by Abhinav Pandey

(U6724645)

1. Consider the filter f = [1,2,1] and the 1D image I = [0,1,2,4,10,3,4]. What is the result of $f * I$? Pad the image with zeros at the boundaries if necessary.

The first task was to pass a filter (***f* = [1,2,1]**) over a 1-dimensional image (***I* = [0,1,2,4,10,3,4]**) to obtain the filtered image (***f* * *I***). We can visualise the image and filters as shown below (Note: the image appearances are just an approximation to help us understand the effect of the filtering process):

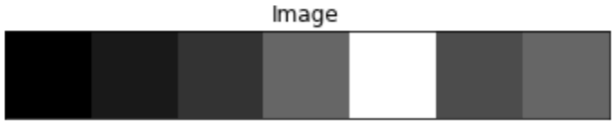

Figure 1. Grayscale representation of image, I

Figure 2. Grayscale representation of filter, f

To determine the result of ***f* * *I***, the filter was passed over the image I (with a stride of 1) such that each element in the image was multiplied with the filter element just above it.

For instance, if the filter **[1, 2, 1]** was over the first 3 elements of the 1D image i.e. [0, 1, 2], the resulting (filtered) image would be [0 x **1**, 1 x **2**, 2 x **1**] = [0, 2, 2].

Filter f, essentially doubles the pixel intensity of the image. Since, the 1st and 3rd elements of ***f*** are 1, they do not affect the image on multiplication. However, the 2nd element, 2, doubles the value of the pixel in ***I*** under it. As the filter passes over the image, the 2nd filter element doubles the image intensity of every pixel in the image, ***I***.

Since, only the 2nd filter element brings a change in the image it was necessary for it passed over every pixel of the image, to ensure complete processing of the image. For this reason, we padded the image with a zero on either side.

The result, ***f* * *I*** = [0,2,4,8,20,6,8] and can be visualised as shown below.

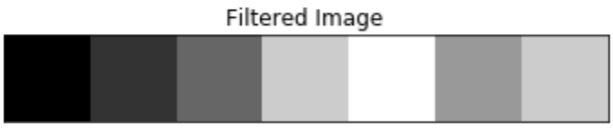

*Figure 3. Grayscale representation of result, **f * I***

2. The boundary of a set A, denoted by $\beta(A)$, can be obtained by first eroding A by B and then perform the set difference between A and its erosion. That is $\beta(A) = A -$

($A \ominus B$), where $\ominus$ denotes the erosion operation, B is a suitable structuring element.
Note that 'grey' represents binary 1s and white binary 0s in this question.
(1) Given A and B as follows, please plot $A \ominus B$.
(2) Please plot $\beta(A)$.

We have an image(A) and a structuring element (B) in this task, where we are expected to determine the boundary of the image. This can be done in two broad steps:

1. Performing erosion of image using the filter.
2. Subtracting the eroded image with the original image.

The image A and structuring element B looks like this:

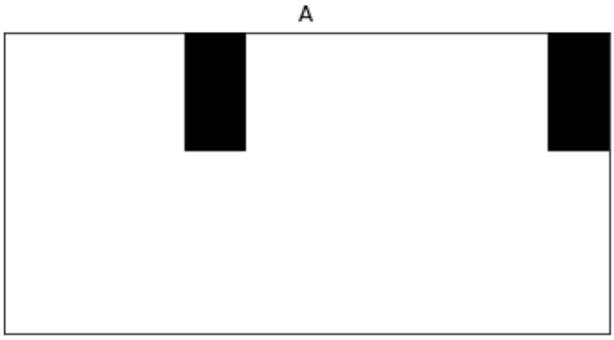

Figure 4. Image A (before erosion)

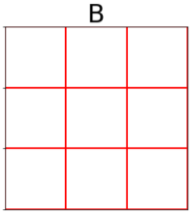

Figure 5. Structuring Element, B

1. **Performing erosion of image using the filter :**

Erosion is a basic morphological operation in image processing, that removes pixels on image boundaries. [1]

This is done by first padding the target image with 1s, and then convolving a structuring element over the padded image.

During convolution, the center pixel of the image region under B is set as the *minimum* value (in the case of binary images, like A) of all pixels in the neighbourhood. [1]

To put it simply, a pixel is set to 0 if any of the neighbouring pixels have the value 0.

Ans (1) Here is the erosion of A, ($A \ominus B$), where $\ominus$ denotes the erosion operation:

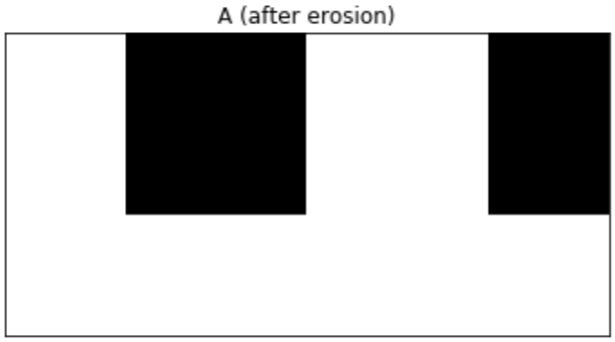

Figure 6. Image $A \ominus B$ (A eroded by B)

2. **Subtracting the eroded image with the original image :**

Since, the erosion operation removes the boundary pixels in an image, we can subtract the erosion result from the original to obtain the boundary.

Mathematically,

$$\beta(A) = A - (A \ominus B)$$

Ans (2) The result of the equation above is as shown below:

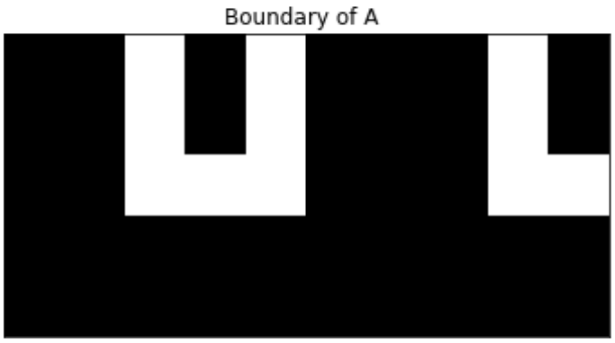

Figure 7. Boundary of A, $\beta(A)$

3. Contour Detection

Contour detection for any image can done by following a series of simple steps. These are:

- De-noising target image using Gaussian Blur (of suitable size and value for standard deviation).
- Computing gradient intensity of the image in the horizontal and vertical direction.
- Determining the norm of the horizontal and vertical gradients to find the overall gradient magnitude at every pixel.
- Applying a threshold to the edges to remove pixels representing weak edges.
- Determining the angle of the gradient at every pixel.
- Using the above calculated gradient magnitude and directions to perform non-maximum suppression to obtain single pixel width edges.

It is essential to smooth/de-noise an image before performing edge detection. This is because images can often contain noise/irregularities. Without removing these irregularities, an edge detection system is much likely to incorrectly identify certain pixels as edges. Therefore, uniformly blurring an image before edge detection is a must to perform reliable edge-detection.

Figure 8. Image before blurring

Figure 9. Image after blurring

The next step is to compute the gradient of the image in the horizontal and vertical direction. This is done using an edge detection filter (such as Sobel or Scharr) and convoluting it over the image. We can get the edges of the image by computing the norm of the vertical and horizontal gradients.

We can see the edge detection results, both before and after blurring, below :

Figure 10. Edges of image (without blurring)

Figure 11. Edges of image (with blurring)

On comparing these results, we can see the effect of blurring before the edge detection process. The edges in the left image, although sharp, contain a lot of noisy elements (*such as the **A** on the tail of the airplane*). On the other hand, the edges in the blurred image are much smoother. They are accurate and supress the noisy edges found in the left image.

It is also important to note the *artefacts* found on the boundaries of both these images. To understand why they occur, it is important to understand how convolution is being performed for gradient computation.

The convolution in the edge detection system is being done using the *convolve2d* function from the *signal* module of *scipy*. The convolution is such that the image boundaries are not being processed.

This can be simply corrected, by specifying an additional parameter in our call to the *convolve2d* function viz. boundary = '*symm'*, which ensures that the gradient operator passes over every pixel of the image, including those at the image boundaries. It does so by padding the target image with a suitable number of layers of 0s.

The edges after adjusting the convolution parameter look like this :

Figure 12. Edges of blurred image

Although, the system has correctly identified most of the edges, there are still a lot of noisy elements that remain. We can simply fix this by setting a threshold to determine the edges. Pixels with magnitude greater than the threshold will be accepted as edges, while the rest will not.

Figure 13. Image Edges (before thresholding)

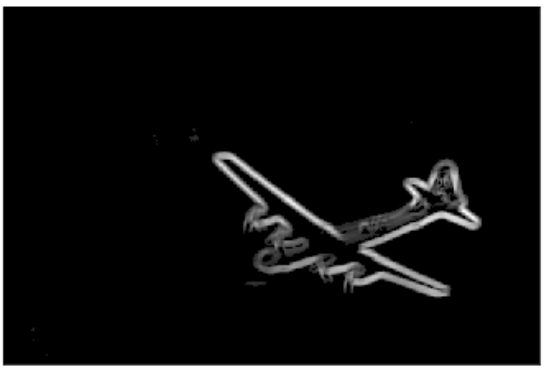

Figure 14. Image Edges (after thresholding)

The edges of our edge detection system look amazing. However, there is still scope for improvement by reducing the width of these edges to a single pixel. This can be done using the concept of non-maximum suppression.

This is how the we can implement non-maximum suppression :

1. Convolve a 3 x 3 filter over the image containing edges.
2. Compare the magnitude of the center element with the two neighbouring pixels corresponding to the direction of the gradient.
    a. For example, if the gradient direction at a pixel is 0°, we compare its gradient intensity to the pixels on its left and right.
    b. Similarly, if the gradient direction is 90°, we compare the pixel to the pixel above and below it.
    c. In cases, when the gradient direction corresponds to a direction between multiple neighbouring pixels, we use interpolation to find the intensity of these *intermediate* neighbouring pixels.
3. If the magnitude of the pixel is greater than that of its neighbouring pixels, then it is selected as an edge. Otherwise, it is removed.

Now, let us apply non-maximum suppression to our image and compare the result -

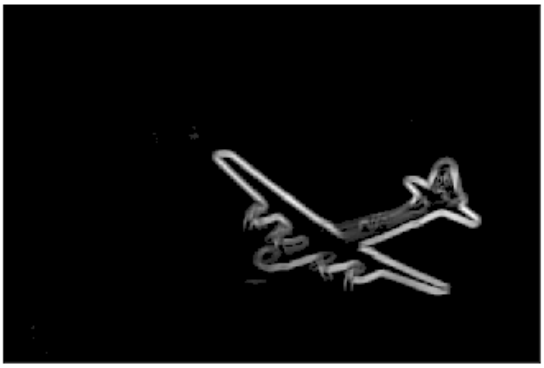

Figure 15. Image Edges (before non-maximum suppression)

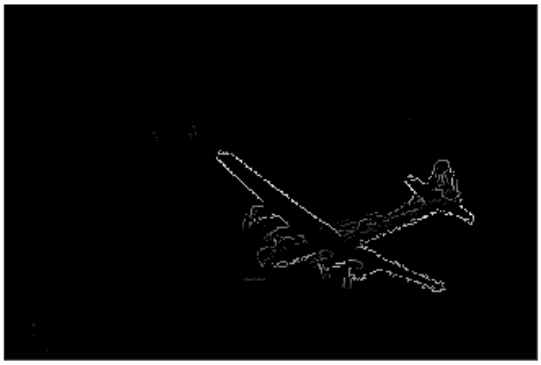

Figure 16. Image Edges (after non-maximum suppression)

Below, is the performance summary of our edge detection system, which is determined by calculating the edges using our system and evaluating those results against the ground-truth labels for each image.

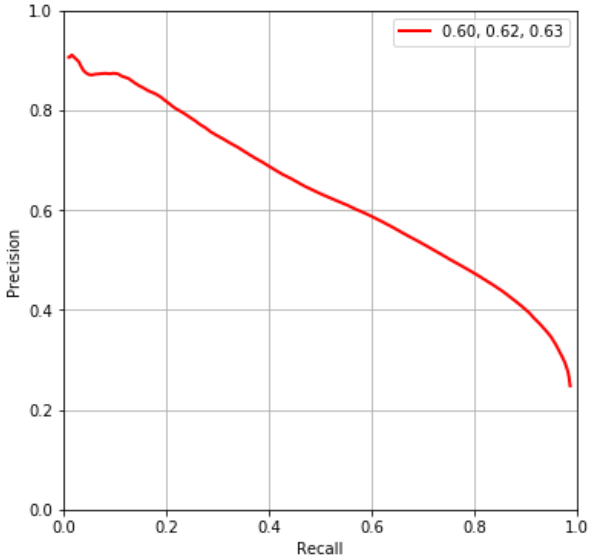

Figure 17. Precision-Recall curve of edge detection system (with non-maximum suppression)

Overall max F1 score	Mean max F1 score	Area under PR curve
0.604567	0.619714	0.625604

Table 1. ***Evaluation metrics for edge detection system with non-maximum suppression***

4. Superpixels and SLIC

1. **Please explain what superpixel is and its applications in computer vision applications.**
   Superpixels are groups of pixels with similar properties such as colour intensity, position in image, etc. They are extremely useful in different domains of computer vision such as colour image segmentation and object detection.

2. **Please describe the proposed algorithm (simple linear iterative clustering (SLIC)) generate superpixels.**

   The SLIC algorithm can be used for colour image segmentation by processing images in the CIELAB colour space, along with their x, y coordinates. The most significant point of the proposed algorithm is its novel distance measure, that incorporates distance between pixels based on both, colour and spatial information.

   The SLIC algorithm can be implemented by following these key steps :

1. Selecting a value for **K**, which is the total number of superpixels/clusters in the image.
2. Sampling **K** regularly spaced cluster centres $C_k = [l_k, a_k, b_k, x_k, y_k]^T$ with k = [1, K] throughout the image, such that each cluster center lies at the lowest gradient position in its 3 x 3-pixel neighbourhood. This is done to avoid placing the cluster center at an edge or a noisy pixel.
   i. The size, **S** of each superpixel/cluster is equal to $\sqrt{N/K}$. This **S x S (approx.)** grid centred at the cluster center becomes the search area for all pixels belonging to that superpixel.
3. Assign the best matching pixels in the **2S x 2S** search space to the closest cluster center. The limitation put on the search space is because after a certain distance from the center, the information from x,y coordinates begin to outweigh the colour information's importance. Limiting the search space helps in tackling this problem.
   i. The "best" matching pixels are determined by computing the distance, $\mathbf{D_s}$, where :

   ii. $$Ds = d_{lab} + m/S * d_{xy}$$

   iii. and d_{lab}, d_{xy} are the Euclidean distance between a pixels and cluster centres, LAB values and X,Y-coordinates respectively. The parameter, m emphasises the importance of spatial proximity i.e. x,y coordinates. Greater the value of m, the more compact the superpixel/cluster.
4. Computing new cluster centres for each cluster (and store residual errors).
5. Repeat steps 3. and 4. until convergence (when cluster centres come to a fixed position)
6. Ensure connectivity. In other words, we want all the clusters to be tightly packed without any space in between. This is done by relabelling the remaining disjoint segments to the largest neighbouring cluster.

References

1. 2020. [online] Available at: <https://au.mathworks.com/help/images/morphological-dilation-and-erosion.html> [Accessed 6 June 2020].
2. *SLIC Superpixels, Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aure- lien Lucchi, Pascal Fua, and Sabine Su ̈sstrunk, SLIC Superpixels, EPFL Technical Report 149300, June 2010.*