Image Processing Algorithms

By: Abdelrahman Moataz Noureldin

1. Sobel filter

Sobel filter is an edge detection algorithm that approximates the gradient of the image intensity function for grey scale images. It consists of 2 3x3 kernels that are convolved with the target image, Gx produces the gradient along the horizontal direction and Gy produces the gradient along the vertical direction.

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

Figure 1: Horizontal (left) and vertical (right) kernels of the Sobel filter

Gx and Gy produce the edge maps in the horizontal and vertical directions respectively. To produce the final image, the absolute magnitude of the gradient is calculated for each pixel using Pythagoras theorem.

$$|G| = \sqrt{{G_x}^2 + {G_y}^2}$$

The greater the magnitude of the gradient for a certain pixel, the higher the chance it lies on an edge. Therefore, the final step for the edge detector is to define a threshold for the magnitude to separate pixels belonging to edges and those that do not.

The Sobel filter is known to produce noisy edge estimates due to the small size kernel and simplicity of the method. So, it is common that the original image is smoothed first (usually gaussian blur due to its isotropic nature like Sobel filter) before convolving the image with the Sobel kernels.

Another use of Gx and Gy is to find the orientation of each edge pixel (angle at which the tangent at that pixel would be) using trigonometry:

$$|G| = \arctan\left(\frac{G_y}{G_x}\right)$$

Pseudo code:

```
For pixel in image:

Store surrounding pixels into variables (top left right centre, bottom left
right centre, middle left and right)
Gx = topleft * -1 + middleleft * -2 + bottomleft * -1 + topright * 1 +
middleright * 2 + bottomright * 1
Gy =  bottomleft * -1 + bottomcentre * -2 + bottomright * -1 + topleft * 1 +
topcentre * 2 + topright * 1
pixelGradient = sqrt(gx ** 2 + gy ** 2)
if pixelGradient > threshold: pixel = 255 (white)
else: pixel = 0 (black)

end
```

2. Laplacian Filter

The Laplacian filter is an isotropic image processing algorithm that calculates the 2nd derivative of the image intensity function of a grayscale image. Calculating the 2nd derivative makes the algorithm very sensitive to zero crossings (point where sudden change of intensity occurs) and therefore is very useful in edge detection. It does not require to separate kernels for the horizontal and vertical direction and therefore skips a step compared to the Sobel filter. The sensitivity of the Laplacian filter makes it vulnerable to noise and therefore it is a common practice to smoothen the image using Gaussian blur before applying Laplacian, resulting in Laplacian of Gaussian (LoG) process.

0	−1	0
−1	4	−1
0	−1	0

−1	−1	−1
−1	8	−1
−1	−1	−1

Figure 2: Two commonly used discrete approximations to the Laplacian filter

The Laplacian filter is commonly used in zero crossing detectors for edge detection. It also finds use in image sharpening (by subtracting the LoG from the original image) and in feature extraction (for example in convolutional neural networks).

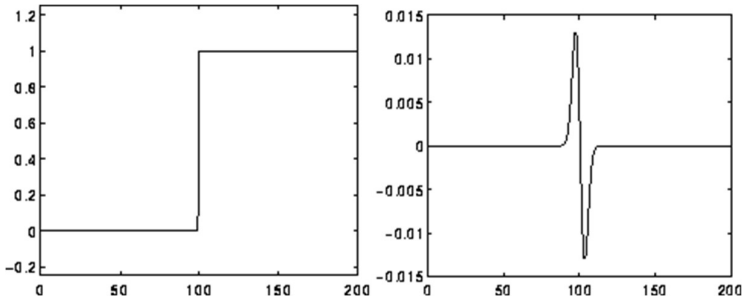

Figure 3: Left: 1-D image containing a step edge Right: 1-D LoG filter applied to left image

Example Python code from educative.io:

```
1  import numpy as np
2
3  def WindowMultiply(LaplacianFilter,Image,row,col):
4      Result=0
5      for i in range(3):
6          for c in range(3):
7              Result=Result+LaplacianFilter[i][c]*Image[i+row][col+c]
8      return Result
9
10 def ConvolutionWithLaplacian(fitler,Image):
11     Result = np.random.rand(3,4)
12     for i in range(3):
13         for c in range(4):
14             Result[i][c]=WindowMultiply(LaplacianFilter,Image,i,c)
15     return Result
16
17 if __name__=="__main__":
18
19   LaplacianFilter=np.array([[0,-1,0],
20               [-1,4,-1],
21               [0,-1,0]])
22   Image=np.array([[4,7,10,1,14,5],[8,13,8,10,18,20],[7,8,54,10,3,10],[9,6,9,7,
23 9,12],[54,9,1,9,5,9]])
24   Result=ConvolutionWithLaplacian(LaplacianFilter, Image)
25
26   print("Result after Convolution with Laplacian Filter : ")
27   print(Result)
```

The explanation of the code is given below.

- **Line 3–8:** The function WindowMultiply() multiplies the LaplacianFilter and window of the Image.
- **Line 10–15:** ConvolutionWithLaplacian() performs the convolution of the image with a filter. The nested for loops in this function indicates the sliding window, and for each sliding window, we call the function WindowMultiply().
- **Line 21–23:** Initializing the Laplacian filter.
- **Line 24:** Represents the Image in the form of an array.
- **Line 25:** Storing the convolution result in the variable Result.

3. Canny Edge Detector

The Canny Edge Detector is a multi-step edge detection algorithm that tries to fix the many issues that can come when detecting edges of different images. An optimal edge detector has:

- High detection rate while minimising false positives (noise appearing as edges) and false negatives (missing real edges).
- Good localisation of edge pixels (result edge as close as possible in shape to real edge).
- Single response constraint: a single point/pixel is returned for each true edge edge point; that is, minimize the number of local maxima around the true edge (created by noise).
- Low noise sensitivity.

The following are the steps the algorithm takes to achieve optimal edge detection:

1. Smooth target image using Gaussian blur.
2. Find the magnitude and orientation images (using Sobel or Prewitt filters)
3. Apply non-maximum suppression to thin multi-pixel edges.
4. Edge linking using hysteresis thresholding.

Non-maximum suppression: To find the edge points, we need to find the local maxima of the gradient magnitude. Broad ridges must be thinned so that only the magnitudes at the points of greatest local change remain. All values along the direction of the gradient that are not peak values of a ridge are suppressed.

The result of the non-maximum suppression will contain local maxima created from noise. Using a single threshold will be ineffective because:

- A low threshold will not remove all local maxima produced from noise.
- A high threshold might remove some of the true maxima, creating a fragmented edge.

The solution is to use both thresholds, called **hysteresis thresholding**.

- Anything above the high threshold is considered a true edge.
- Anything below the low threshold is considered noise and is removed.
- **Edge linking** is done in between the two thresholds. Edge pixels that are connected to an edge passing through the high threshold will be considered true edges. Otherwise, it will be considered noise and removed.

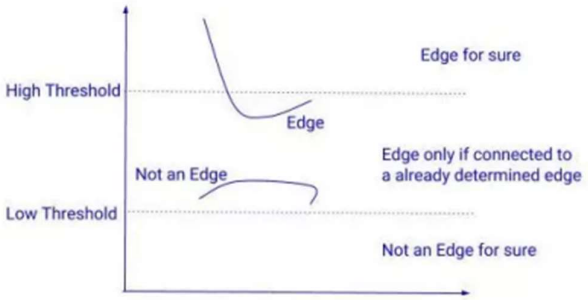

Figure 4: Hysteresis thresholding

Figure 5 is an example of canny edge detection.

a) Original Image
b) Result of horizontal Prewitt filter
c) Result of vertical Prewitt filter
d) Quantised orientation map
e) Gradient amplitude map
f) Amplitudes after non-maximum suppression
g) Double thresholding. White pixels are above the high threshold, red pixels are between the high and low thresholds
h) Final output after hysteresis thresholding. White pixels from (g) are kept with any red pixels connected to them.

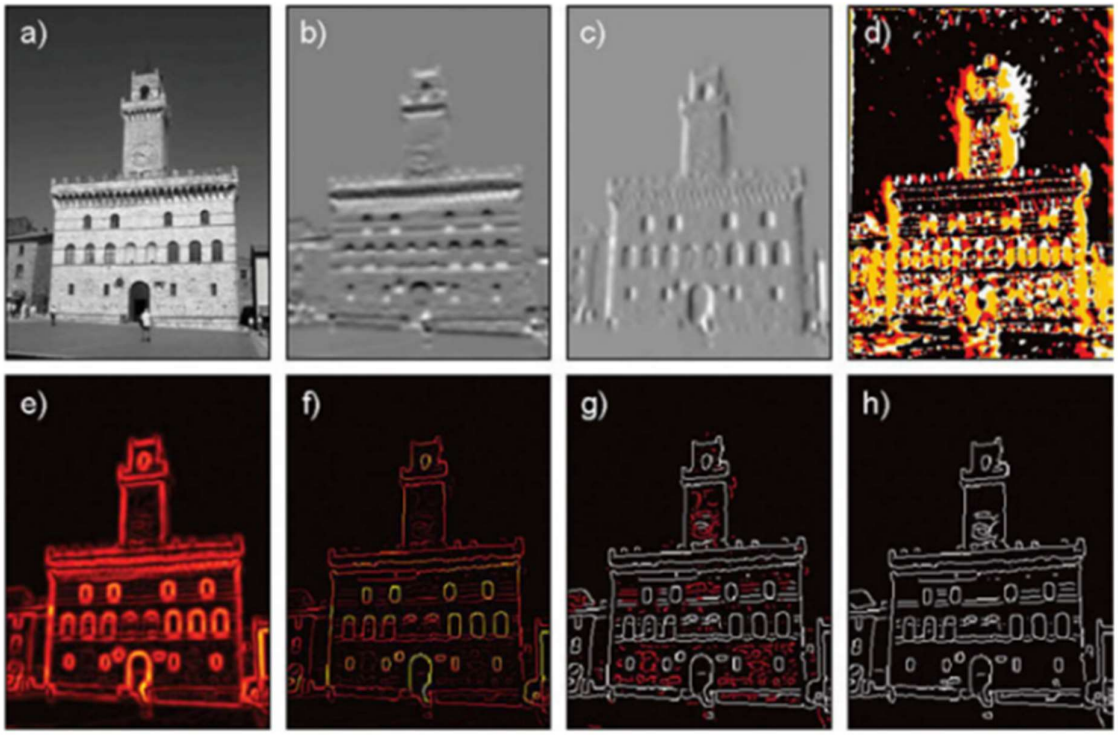

Figure 5: Canny edge detector example

Full python code for a Canny edge detector algorithm with comments will be uploaded to GitHub with this report.

4. Contours

In computer vision, a contour is a curve that connects a series of connected points that define the boundary of the object. These points have similar intensity values and therefore separate the object from its surroundings (hence a boundary). The contour outlines an object in the image and can therefore be considered shape detection. It is useful in shape analysis and object detection and recognition.

It is easiest to form contours for images that are in grayscale or binary, having the ability to easily differentiate a shape from its surroundings.

5. Reference

- https://homepages.inf.ed.ac.uk/rbf/HIPR2/sobel.htm
- How To Find Edges In Images: Sobel Operators & Full Implementation By: Aryaman Sharda- https://www.youtube.com/watch?v=VL8PuOPjVjY
- Finding the Edges (Sobel Operator) By: Computerphile- https://www.youtube.com/watch?v=uihBwtPIBxM
- https://en.wikipedia.org/wiki/Sobel_operator
- Exploring Laplacian Filter: Theory and Numerical Application By: Dr. Rashi Agarwal- https://www.youtube.com/watch?v=xsf7NfAAb6g
- https://www.educative.io/answers/what-is-the-laplacian-filter
- https://homepages.inf.ed.ac.uk/rbf/HIPR2/log.htm
- https://homepages.inf.ed.ac.uk/rbf/HIPR2/zeros.htm
- https://www.cse.unr.edu/~bebis/CS791E/Notes/EdgeDetection.pdf
- https://docs.opencv.org/3.4/d4/d73/tutorial_py_contours_begin.html
- https://www.baeldung.com/cs/computer-vision-contours
- https://www.geeksforgeeks.org/what-are-contours-in-computer-vision/