EDGE DETECTION
(DIGITAL IMAGE PROCESSING)

Faculty of Information Technology
Ton Duc Thang University

August 2023

Edge Detection

- Extracting The Edges From An Image

- Objective
  - What is edge detection and how it can be helpful in image classification.
  - Learn how kernels are used to identify the edges in a given image.

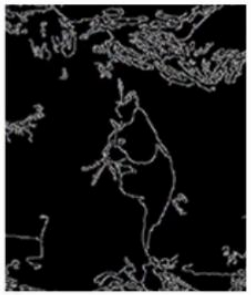
https://www.analyticsvidhya.com/blog/2021/03/edge-detection-extracting-the-edges-from-an-image/

Edge Detection: image classification

Can you differentiate between the objects?

Analytics Vidhya
Learn everything about analytics

Edge Detection: image classification



Can you still easily classify the images?

Analytics Vidhya

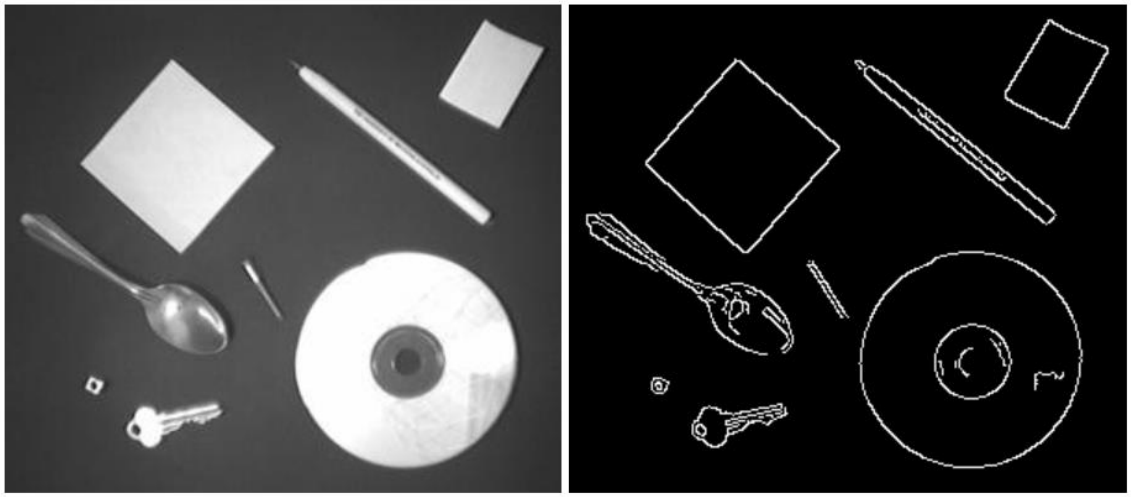
Edge Detection: image classification

- removed the color, the background, and the other minute details from the pictures

$\Rightarrow$ extract only the edges, we would still be able to classify the image.

Edges

- Edges are significant local changes of intensity in an image.
- Edges typically occur on the boundary between two different regions in an image (Trucco, Chapt 4 AND Jain et al., Chapt 5)



Goal of edge detection

- Produce a line drawing of a scene from an image of that scene
- Important features can be extracted from the edges of an image (e.g., corners, lines, curves)
- These features are used by higher-level computer vision algorithms (e.g., recognition)

What causes intensity changes?

- Geometric events
  - object boundary (discontinuity in depth and/or surface color and texture)
  - surface boundary (discontinuity in surface orientation and/or surface color and texture)

- Non-geometric events
  - specularity (direct reflection of light, such as a mirror)
  - shadows (from other objects or from the same object)
  - inter-reflections

Edge Detection

- Identify the edges by looking at the numbers or the pixel values

0	2	15	0	0	11	10	0	0	0	0	9	9	0	0	0
0	0	0	4	60	157	236	255	255	177	95	61	32	0	0	29
0	10	16	119	238	255	244	245	243	250	249	255	222	103	10	0
0	14	170	255	255	244	254	255	253	245	255	249	253	251	124	1
2	98	255	228	255	251	254	211	141	116	122	215	251	238	255	49
13	217	243	255	155	33	226	52	2	0	10	13	232	255	255	36
16	229	252	254	49	12	0	0	7	7	0	70	237	252	235	62
6	141	245	255	212	25	11	9	3	0	115	236	243	255	137	0
0	87	252	250	248	215	60	0	1	121	252	255	248	144	6	0
0	13	113	255	255	245	255	182	181	248	252	242	208	36	0	19
1	0	5	117	251	255	241	255	247	255	241	162	17	0	7	0
0	0	0	4	58	251	255	246	254	253	255	120	11	0	1	0
0	0	4	97	255	255	255	248	252	255	244	255	182	10	0	4
0	22	206	252	246	251	241	100	24	113	255	245	255	194	9	0
0	111	255	242	255	158	24	0	0	6	39	255	232	230	56	0
0	218	251	250	137	7	11	0	0	0	2	62	255	250	125	3
0	173	255	255	101	9	20	0	13	3	13	182	251	245	61	0
0	107	251	241	255	230	98	55	19	118	217	248	253	255	52	4
0	18	146	250	255	247	255	255	255	249	255	240	255	129	0	5
0	0	23	113	215	255	250	248	255	255	248	248	118	14	12	0
0	0	6	1	0	52	153	233	255	252	147	37	0	0	4	1
0	0	5	5	0	0	0	0	0	14	1	0	6	6	0	0

there is a significant difference between the pixel values around the edge

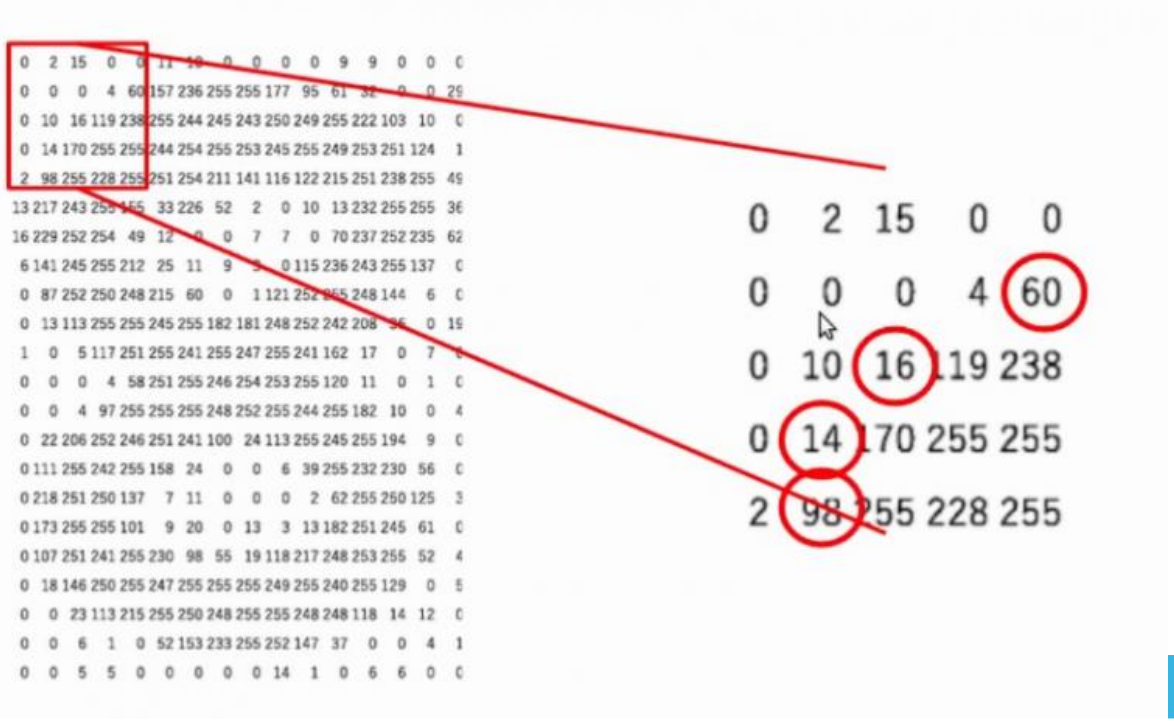
Edge Detection (ct)

- Edge detection is an image processing technique for finding the boundaries of an object in the given image
- Edges are the part of the image that represents the boundary or the shape of the object in the image
  - the pixel values around the edge show a significant difference or a sudden change in the pixel values
- Based on this fact we can identify which pixels represent the edge or which pixel lie on the edge.

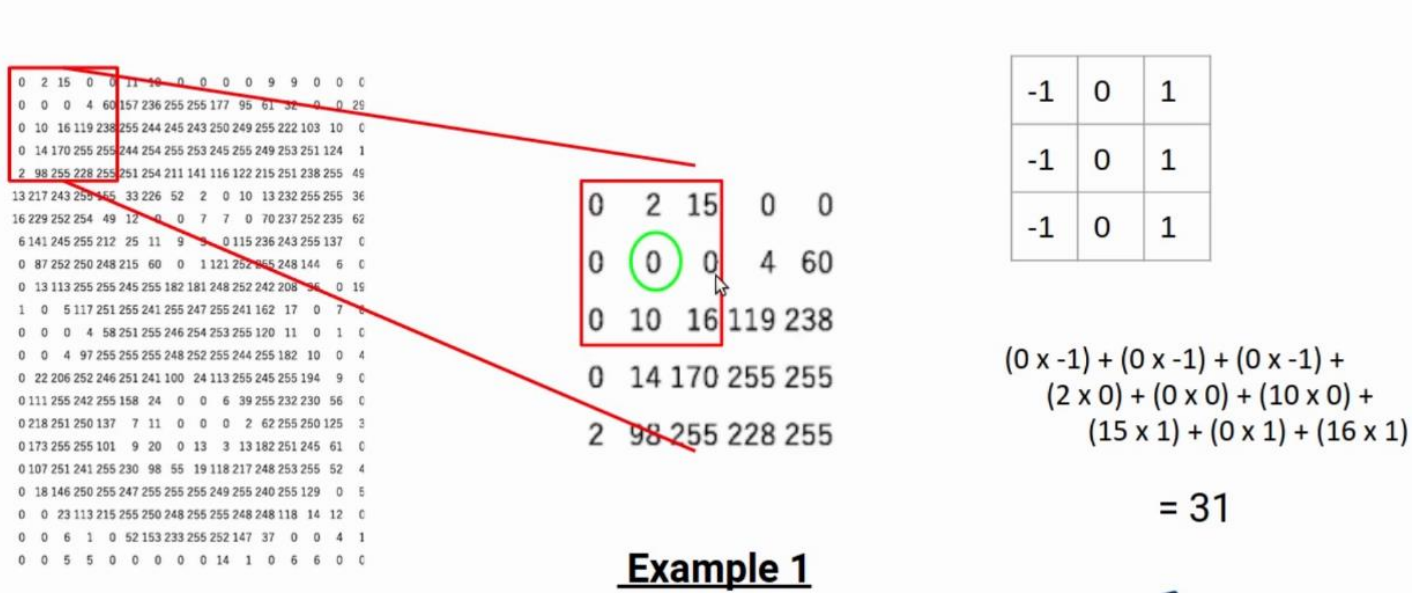
How to Extract the Edges

- compare the pixel values with its surrounding pixels, to find out if a particular pixel lies on the edge
- use a matrix known as the kernel and perform the element-wise multiplication

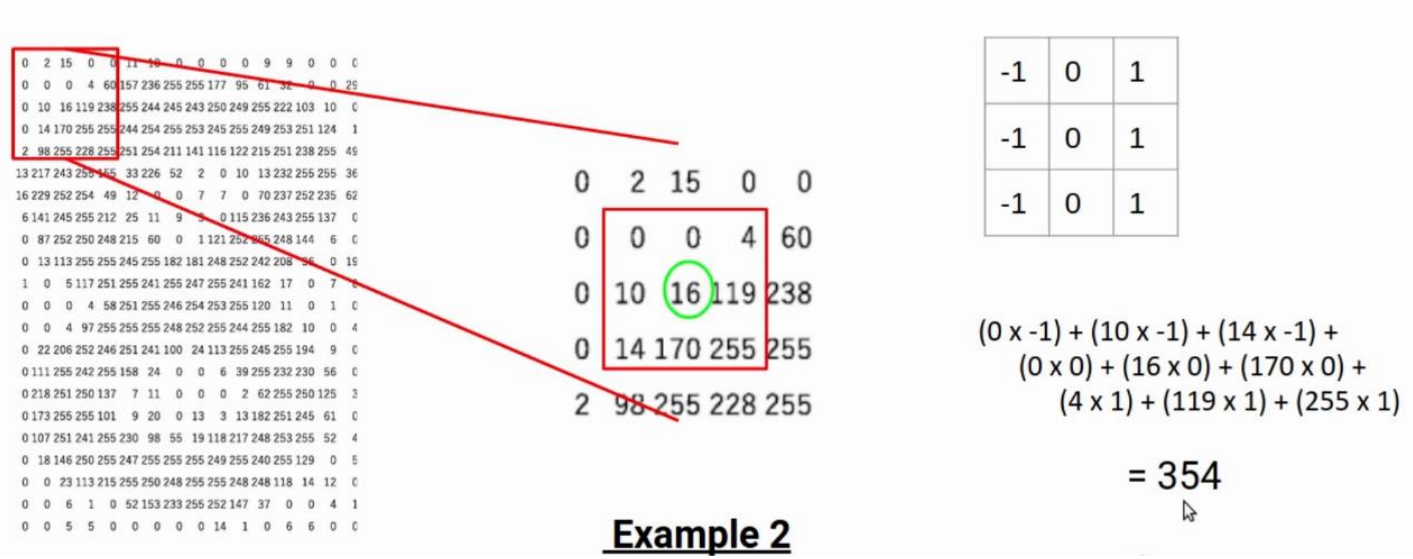
How to Extract the Edges (ct)



How to Extract the Edges (ct)



How to Extract the Edges (ct)



Ho

0	2	15	0	0	11	10	0	0	0	0	9	9	0	0	0
0	0	0	4	60	157	236	255	255	177	95	61	32	0	0	29
0	10	16	119	238	255	244	245	243	250	249	255	222	103	10	0
0	14	170	255	255	244	254	255	253	245	255	249	253	251	124	1
2	98	255	228	255	251	254	211	141	116	122	215	251	238	255	49
13	217	243	255	155	33	226	52	2	0	10	13	232	255	255	36
16	229	252	254	49	12	0	0	7	7	0	70	237	252	235	62
6	141	245	255	212	25	11	9	3	0	115	236	243	255	137	0
0	87	252	250	248	215	60	0	1	121	252	255	248	144	6	0
0	13	113	255	255	245	255	182	181	248	252	242	208	36	0	19
1	0	5	117	251	255	241	255	247	255	241	162	17	0	7	0
0	0	0	4	58	251	255	246	254	253	255	120	11	0	1	0
0	0	4	97	255	255	255	248	252	255	244	255	182	10	0	4
0	22	206	252	246	251	241	100	24	113	255	245	255	194	9	0
0	111	255	242	255	158	24	0	0	6	39	255	232	230	56	0
0	218	251	250	137	7	11	0	0	0	2	62	255	250	125	3
0	173	255	255	101	9	20	0	13	3	13	182	251	245	61	0
0	107	251	241	255	230	98	55	19	118	217	248	253	255	52	4
0	18	146	250	255	247	255	255	255	249	255	240	255	129	0	5
0	0	23	113	215	255	250	248	255	255	248	248	118	14	12	0
0	0	6	1	0	52	153	233	255	252	147	37	0	0	4	1
0	0	5	5	0	0	0	0	0	14	1	0	6	6	0	0

31	111	267	300

-1	0	1
-1	0	1
-1	0	1

(-1x0 + -1x4 + -1x119 + 0x0 + 0x60 + 0x238 + 1x11 + 1x157 + 1x255)

Prewitt & Sobel kernels

Sobel kernels, higher importance is given to the pixel values right next to the target pixel

-1	0	1
-1	0	1
-1	0	1

Prewitt Kernel
X Direction

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

Sobel Kernel
X Direction

-1	-1	-1
0	0	0
1	1	1

Prewitt Kernel
Y Direction

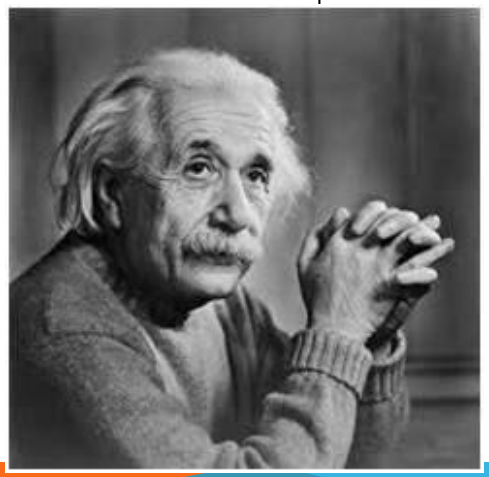
-1	-2	-1
0	0	0
1	2	1

Sobel Kernel
Y Direction

Prewitt kernels

- Vertical direction

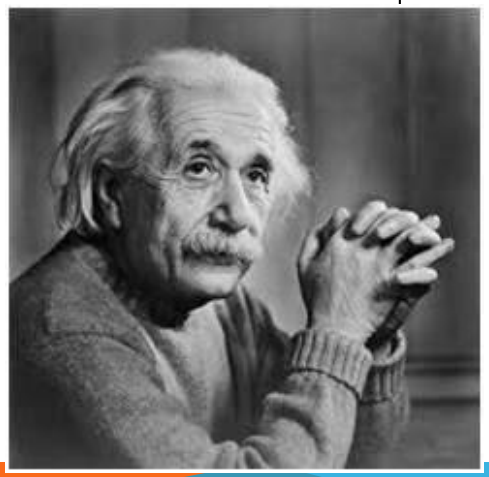
$$\begin{vmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{vmatrix}$$



Prewitt kernels (ct)

- Horizontal direction

-1	-1	-1
0	0	0
1	1	1



Prewitt kernel – both directions

$$\mathbf{G} = \sqrt{\mathbf{G}_x{}^2 + \mathbf{G}_y{}^2} \qquad \mathbf{\Theta} = \operatorname{atan}\left(\frac{\mathbf{G}_y}{\mathbf{G}_x}\right)$$

Magnitude **Direction**

Ex. 1

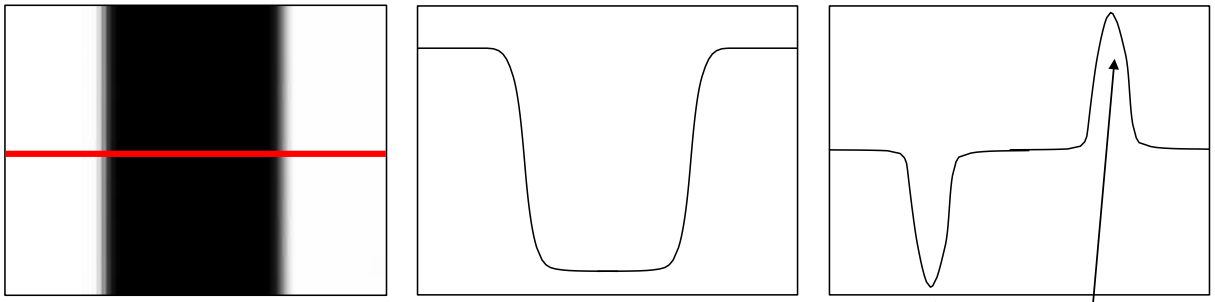
- Apply Prewitt/Sobel operators in X direction, Y direction, and both directions into the following image:

Original Source Image in Grayscale with Intensity Values

150	150	150	255	255
150	150	255	255	1
150	255	255	1	1
255	255	1	1	1
255	1	1	1	1

Edges again

- An edge is a place of rapid change in the image intensity function



Points which lie on an edge can be detected by

- detecting local maxima or minima of the first derivative

edges correspond to extrema of derivative

Edges and derivate

- derivatives only exists for continuous functions but the image is a discrete 2D intensity function
- approximated the image gradients using finite approximation as

Forward $$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

Backward $$f'(x) \approx \frac{f(x) - f(x-h)}{h}$$

Central $$f'(x) \approx \frac{f(x+0.5h) - f(x-0.5h)}{h}$$

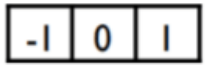
https://theailearner.com/tag/prewitt-operator/

prefer the central difference as shown above

Edges and derivate (ct)

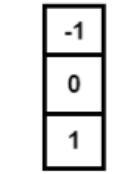
- obtain the derivative filter in x and y directions as shown below

$$f(x,y) = \frac{f(x+h,y) - f(x-h,y)}{2h}$$



x-derivative

$$f(x,y) = \frac{f(x,y+h) - f(x,y-h)}{2h}$$

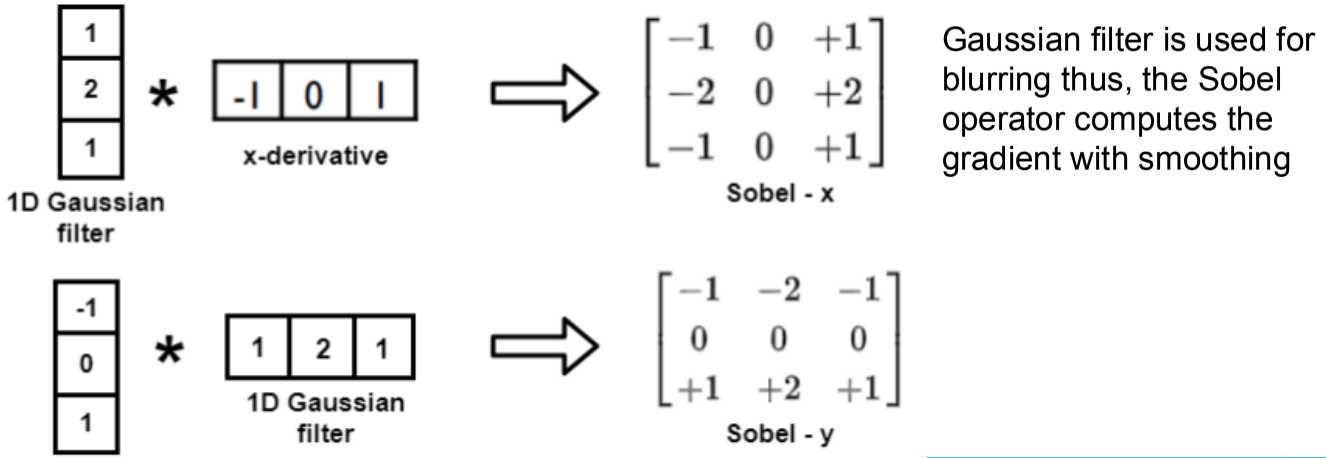


y-derivative

- assumed that the x-coordinate is increasing in the “right”-direction, and y-coordinate in the “down”-direction
- By weighting these x and y derivatives, we can obtain different edge detection filters

Sobel Operator

- multiplying the x, and y-derivative filters obtained above with some smoothing filter(1D) in the other direction
  - For example, a 3×3 Sobel-x and Sobel-y filter can be obtained as



Gaussian filter is used for blurring thus, the Sobel operator computes the gradient with smoothing

Sobel operators (ct)

- Convolve these Sobel operators with the image, they estimate the gradients in the x, and y-directions(say Gx and Gy). For each point, we can calculate the gradient magnitude and direction as

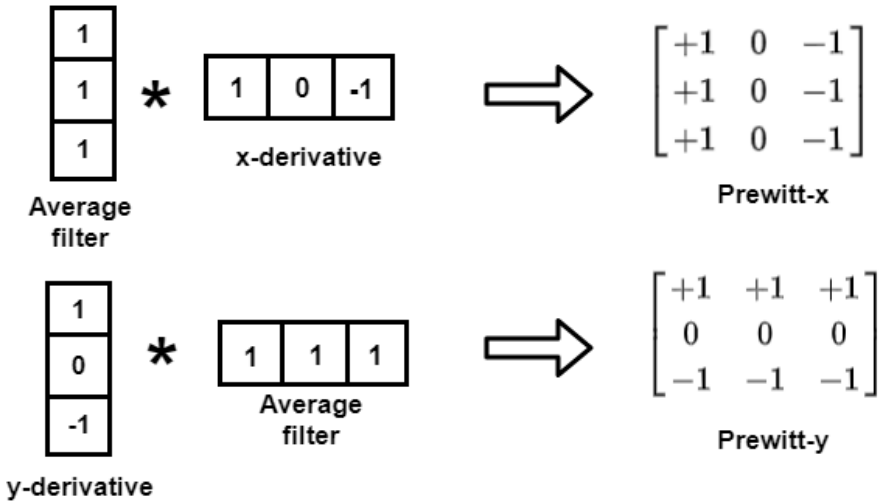
$$\mathbf{G} = \sqrt{\mathbf{G}_x{}^2 + \mathbf{G}_y{}^2} \qquad \mathbf{\Theta} = \operatorname{atan}\left(\frac{\mathbf{G}_y}{\mathbf{G}_x}\right)$$

Magnitude **Direction**

https://docs.opencv.org/3.4/d2/d2c/tutorial_sobel_derivatives.html

Prewitt Operator

- x, and y-derivative filters are weighted with the standard averaging filter as shown below



Prewitt operator example

Original
1024x710

$$\begin{pmatrix} -1 & 0 & 1 \\ -1 & [0] & 1 \\ -1 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} -1 & -1 & -1 \\ 0 & [0] & 0 \\ 1 & 1 & 1 \end{pmatrix}$$

Digital Image Processing: Bernd Girod, © 2013 Stanford University – Edge Detection 27

Prewitt operator example (cont.)

Sum of squared horizontal and vertical gradients (log display)

threshold = 900

threshold = 4500

threshold = 7200

Digital Image Processing: Bernd Girod, © 2013 Stanford University – Edge Detection 28

Sobel operator example

Sum of squared horizontal and vertical gradients (log display)

threshold = 1600

threshold = 8000

threshold = 12800

Digital Image Processing: Bernd Girod, © 2013 Stanford University – Edge Detection 29

Canny Edge Detection

- Canny Edge Detection is a popular edge detection algorithm. It was developed by John F. Canny
- It is a multi-stage algorithm:

- Noise Reduction
  - Edge detection is susceptible to noise in the image, first step is to remove the noise in the image with a Gaussian filter

https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html

Canny Edge Detection (ct)

- Finding Intensity Gradient of the Image
  - Smoothened image is then filtered with a Sobel kernel in both horizontal and vertical direction
    - to get first derivative in horizontal direction (G_x) and vertical direction (G_y)
    - find edge gradient and direction for each pixel (gradient magnitude and angle):

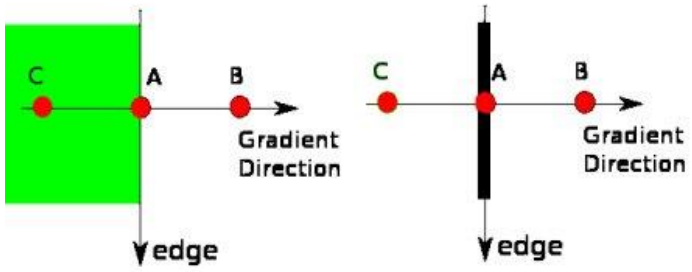
$$Edge\_Gradient\ (G) = \sqrt{G_x^2 + G_y^2}$$

$$Angle\ (\theta) = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

    - Gradient direction is always perpendicular to edges. It is rounded to one of four angles representing vertical, horizontal and two diagonal directions.

Canny Edge Detection (ct)

- Non-maximum Suppression
  - remove any unwanted pixels which may not constitute the edge
  - at every pixel, pixel is checked if it is a local maximum in its neighborhood in the direction of gradient
    - Its gradient magnitude is smaller than either of its neighbors?



Point A is on the edge (in vertical direction). Point B and C are in **gradient directions**. So point A is checked with point B and C to see if it forms a local maximum. If so, it is considered for next stage, otherwise, it is suppressed (put to zero).

Canny Edge Detection (ct)

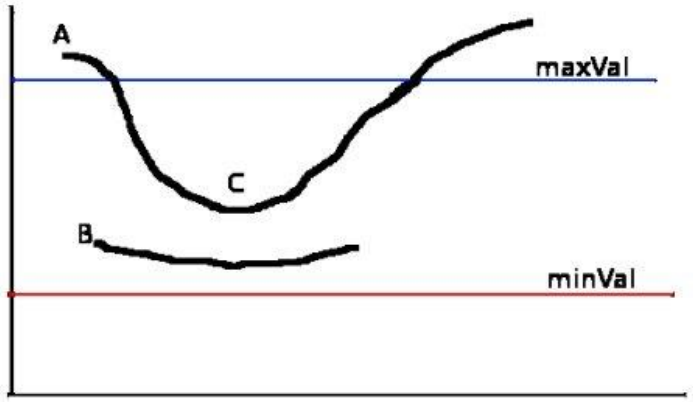
- Hysteresis Thresholding
  - This stage decides which are all edges are really edges and which are not
  - Use two threshold values, minVal and maxVal
    - Any edges with intensity gradient **more than maxVal** are sure to be edges and those **below minVal** are sure to be non-edges
    - Those who **lie between** these two thresholds are classified edges or non-edges based on their connectivity. If they are **connected to "sure-edge«(strong edge)** pixels, they are considered to be part of edges. Otherwise, they are also discarded.

$$\text{Strong edge:} \quad M\left[x, y\right] \geq \theta_{high}$$

$$\text{Weak edge:} \quad \theta_{high} > M\left[x, y\right] \geq \theta_{low}$$

Canny Edge Detection (ct)

- The edge A is above the maxVal, so considered as "sure-edge". Although edge C is below maxVal, it is connected to edge A, so that also considered as valid edge and we get that full curve. But edge B, although it is above minVal and is in same region as that of edge C, it is **not connected to any "sure-edge"**, so that is discarded.



Canny Edge Detection (ct)

- Hysteresis Thresholding
  - This stage also removes small pixels noises on the assumption that edges are long lines

https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html

Ex. 2

- Apply Canny operator into the following image:

Original Source Image in Grayscale with Intensity Values

150	150	150	255	255
150	150	255	255	1
150	255	255	1	1
255	255	1	1	1
255	1	1	1	1

Template Matching

- a technique for finding areas of an image that match (are similar) to a template image (**patch**)
  - compare the template image against the source image by sliding it

https://docs.opencv.org/3.4/de/da9/tutorial_template_matching.html

Template Matching (ct)

- moving the patch one pixel at a time (left to right, up to down)
- At each location, a metric is calculated so it represents how "good" or "bad" the match at that location is (or how similar the patch is to that particular area of the source image)

https://docs.opencv.org/3.4/d4/dc6/tutorial_py_template_matching.html

OpenCV & Machine Learning

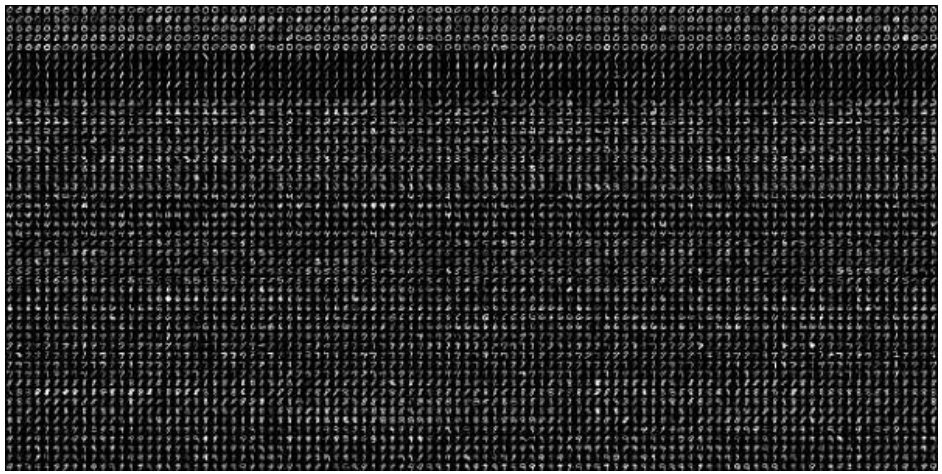
- Machine Learning Library (MLL) is a set of classes and functions for statistical classification, regression, and clustering of data

https://docs.opencv.org/4.x/dd/ded/group__ml.html

https://docs.opencv.org/4.x/d8/d4b/tutorial_py_knn_opencv.html

OpenCV & Machine Learning (ct)

- OCR of Hand-written Data using kNN



References

1. https://docs.opencv.org/4.x/d4/d86/group__imgproc__filter.html
2. https://www.youtube.com/watch?app=desktop&v=kGHz-cEyjiE
3. https://docs.opencv.org/4.x/d4/dc6/tutorial_py_template_matching.html
4. https://www.youtube.com/watch?app=desktop&v=kGHz-cEyjiE
5. https://vincmazet.github.io/bip/filtering/convolution.html