الجامعة المستنصريـــــــــــــة / كليـــــــــة العلوم

قســــــم الفيزيـــــــــاء

Mustansiriyah University

College of Science

Physics Department

Digital Image Processing

Lecture (2) for PhD

Intensity Transformations and Spatial filters

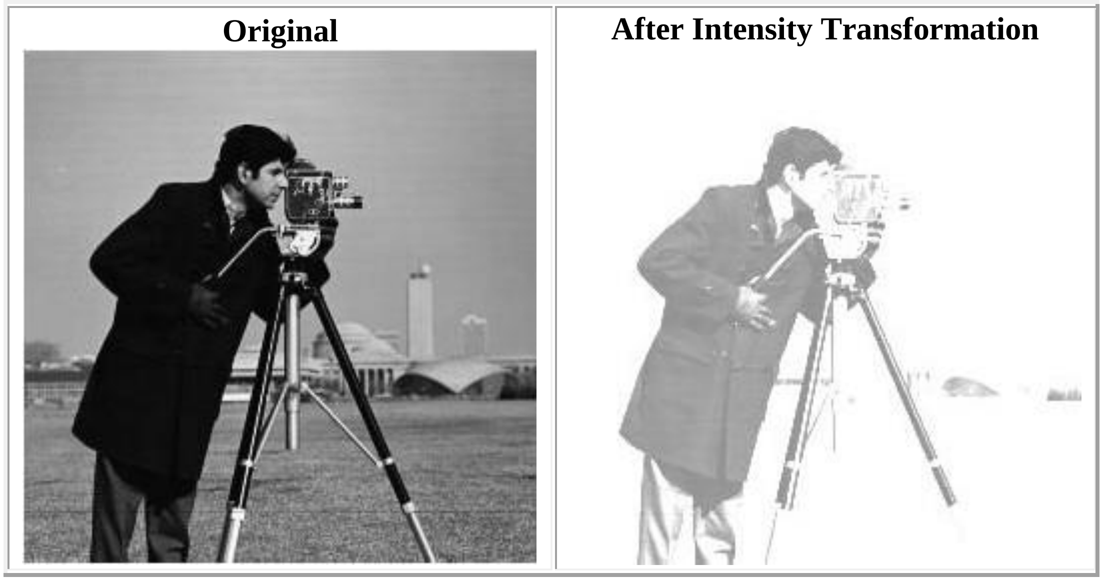
2020-2021

Edited by: Prof. Dr. Ali .A. Al-Zuky

Intensity Transformations

When you are working with gray-scale images, sometimes you want to modify the intensity values. For instance, you may want to reverse black and the white intensities or you may want to make the darks darker and the lights lighter. An application of intensity transformations is to increase the contrast between certain intensity values so that you can pick out things in an image. For instance, the following two images show an image before and after an intensity transformation.

Originally, the camera man's jacket looked black, but with an intensity transformation, the difference between the black intensity values, which were too close before, was increased so that the buttons and pockets became viewable.


Original

After Intensity Transformation

Generally, making changes in the intensity is done through *Intensity Transformation Functions*. The next sections talk about four main intensity transformation functions:

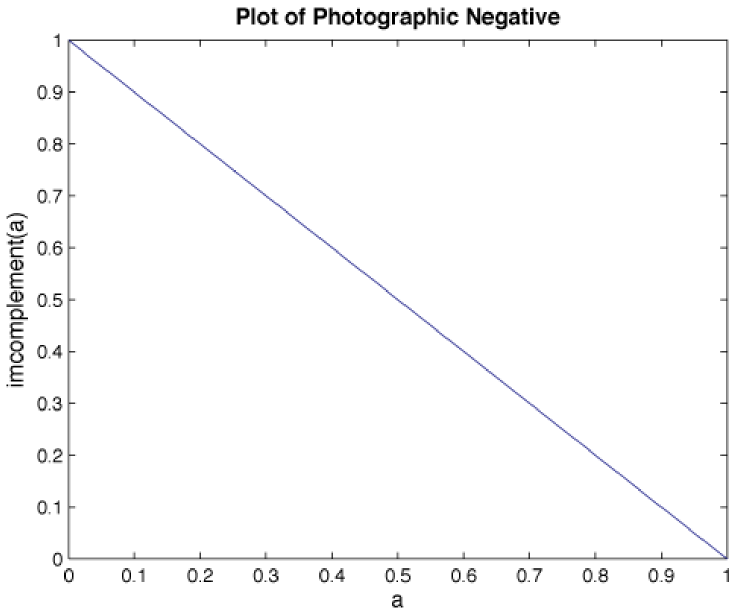
1. photographic negative using (**new_image(J)=1-old_image(I)** or **J(x,y)=(1- I(x,y))**
2. gamma transformation using($\boldsymbol{J(x,y) = A * I(x,y)^{\gamma}}$) where the non-negative real input image pixel value (I) is raised to the power

γ and multiplied by the constant ***A***, to get the output image pixel value **J(x,y).** In the common case of ***A* = 1**.

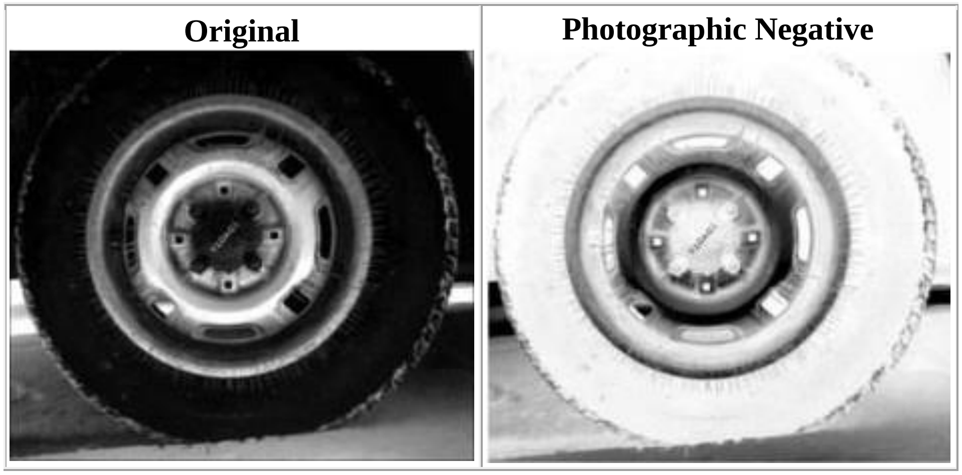
3. logarithmic transformations using (**J(x,y)=c*log(1+I(x,y))**)
4. contrast-stretching transformations like using histogram equalization.

- ## Photographic Negative

The Photographic Negative is probably the easiest of the intensity transformations to describe. Assume that we are working with grayscale image arrays where black is 0 and white is 1. The idea is that 0's become 1's, 1's become 0's, and any gradients in between are also reversed. In intensity, this means that the true black becomes true white and vise versa.. Below shows a graph of the mapping between the original values (x-axis) and the Negative function.



The following is an example of a photographic negative. Notice how you can now see the writing in the middle of the tire better than before:

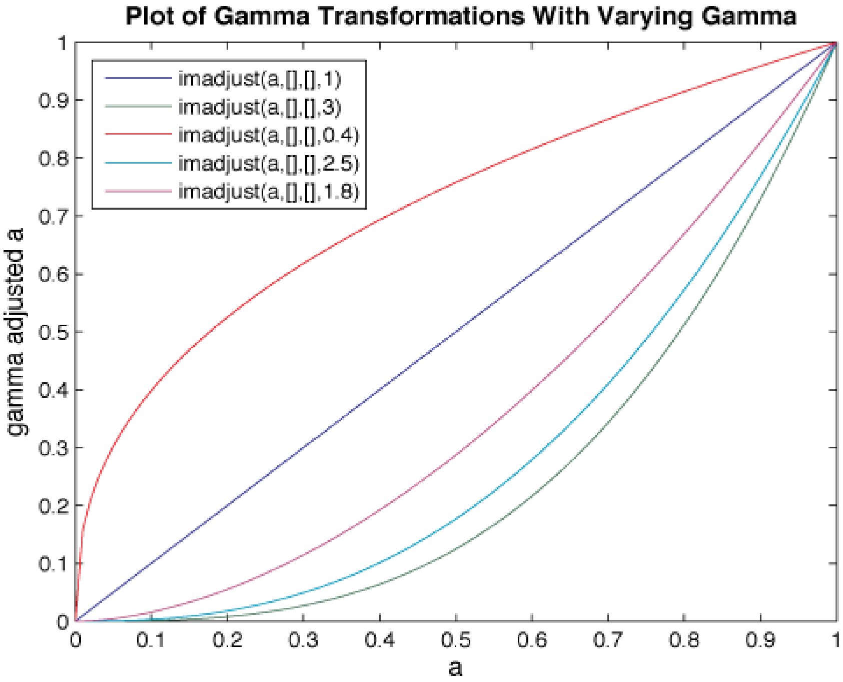


Original

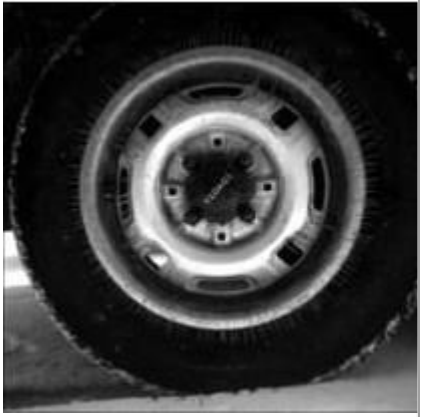
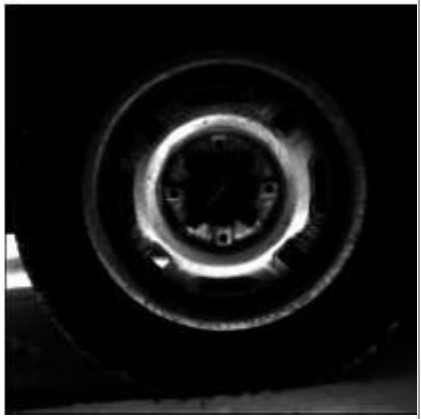
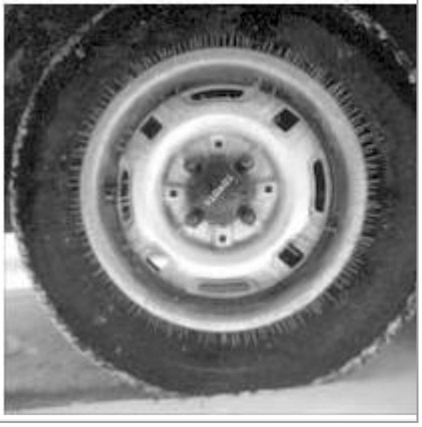
Photographic Negative

- **Gamma Transformations**

With Gamma Transformations, you can curve the grayscale components either to brighten the intensity (when gamma is less than one) or darken the intensity (when gamma (γ) is greater than one). Following plots show the effect of the gamma transformation with varying gamma (γ). Notice that the red line has γ =0.4, which creates an upward curve and will brighten the image.



The following shows the results of three of the gamma transformations shown in the plot above. Notice how the values greater than 1 one create a darker image, whereas values between 0 and 1 create a brighter image with more contrast in dark areas so that you can see the details of the tire.

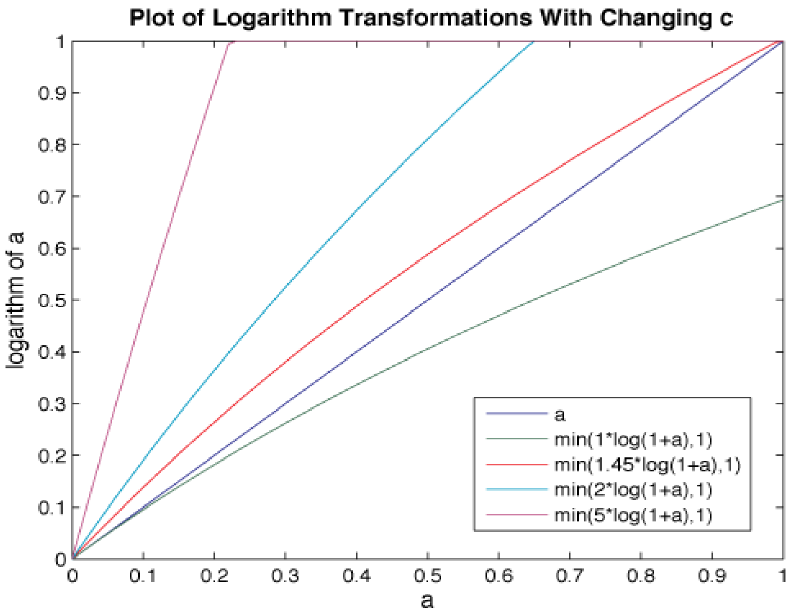
Original (and γ =1)	**γ =3**	**γ =0.4**
		

- **Logarithmic Transformations**

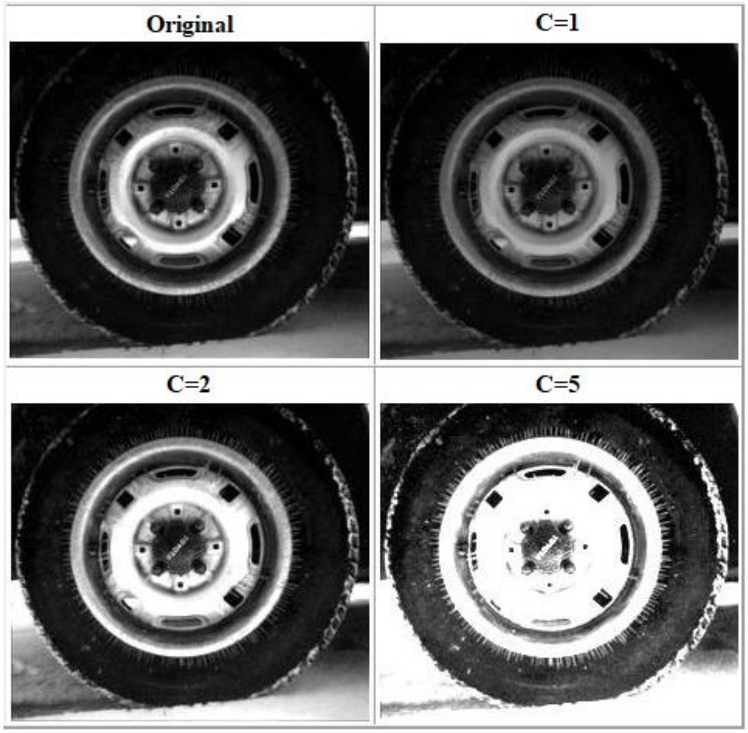
Logarithmic Transformations can be used to brighten the intensities of an image (like the Gamma Transformation, where **γ < 1**). More often, it is used to increase the detail (or contrast) of lower intensity values. The equation used to get the Logarithmic transform of image (**I**) is:

$$\mathbf{J = c*log(1 + (I)}$$

The constant c is usually used to scale the range of the log function to match the input domain. It can also be used to further increase contrast—the higher the c, the brighter the image will appear. Used this way, the log function can produce values too bright to be displayed. The plot below shows the result for various values of **c**. The y-values are clamped at 1 by the min function for the plot of **c=2 and c=5** (teal and purple lines, respectively).

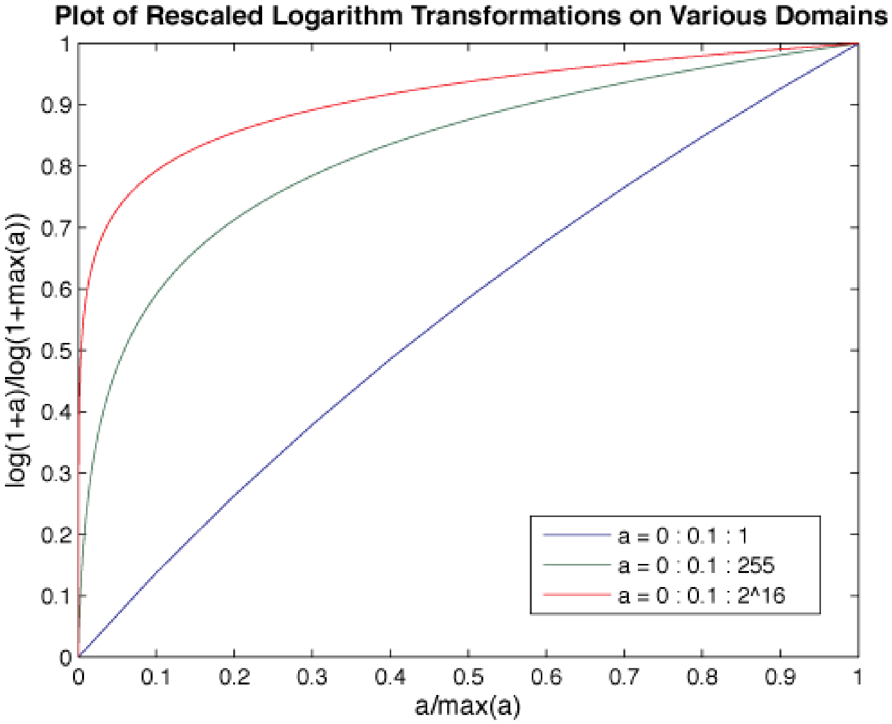


The following shows the original image and the results of applying three of the transformations from above. Notice that when **c=5,** the image is the brightest and you can see the radial lines on the inside of the tire (these lines are barely viewable in the original because there is not enough contrast in the lower intensities).

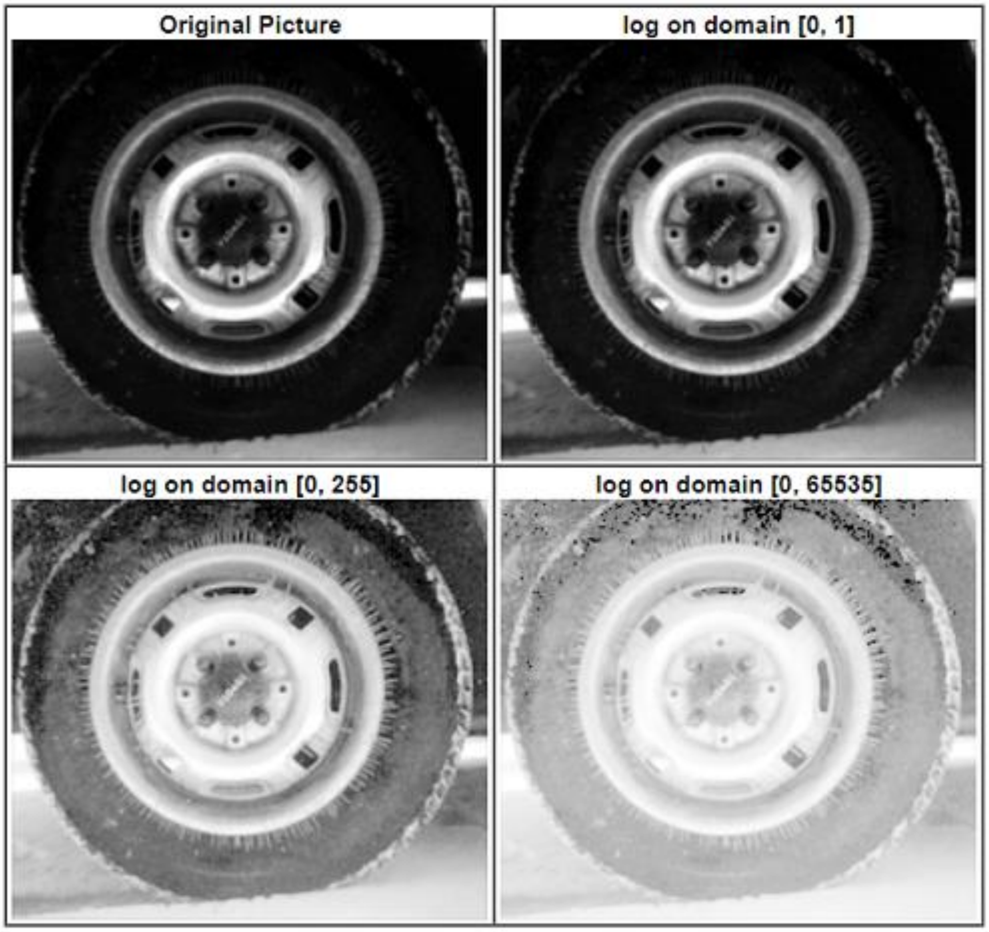


Notice the loss of detail in the bright regions where intensity values are clamped. Any values greater than one, produced from the scaling, are displayed as having a value of 1 (full intensity) and should be clamped.

Although logarithms may be calculated in different bases such as **(log10, log2 and ln (natural log),** the resulting curve, when the range is scaled to match the domain, is the same for all bases. The shape of the curve is dependent instead on the range of values it is applied to. Here are examples of the **log** curve for multiple ranges of input values:



It is important to be aware of this effect if you plan to use logarithm transformations successfully, so here is the result of scaling an image's values to those ranges before applying the logarithm transform:



Note that for domain **[0, 1]** the effects of the logarithm transform are barely noticeable, while for domain **[0, 65535]** the effect is extremely exaggerated. Also note that, unlike with linear scaling and clamping, gross detail is still visible in light areas.

- **Contrast-Stretching Transformations**

Contrast-stretching transformations increase the contrast between the darks and the lights. That transformation kept everything at relatively similar intensities and merely stretched the histogram to fill the image's intensity domain. Sometimes you want to stretch the intensity around a certain level. You end up with everything darker darks being a lot darker and everything lighter being a lot lighter, with only a few levels of gray around the level of interest. Following plots of the original gray images and their contrast enhancement images with histograms using histogram equalization method

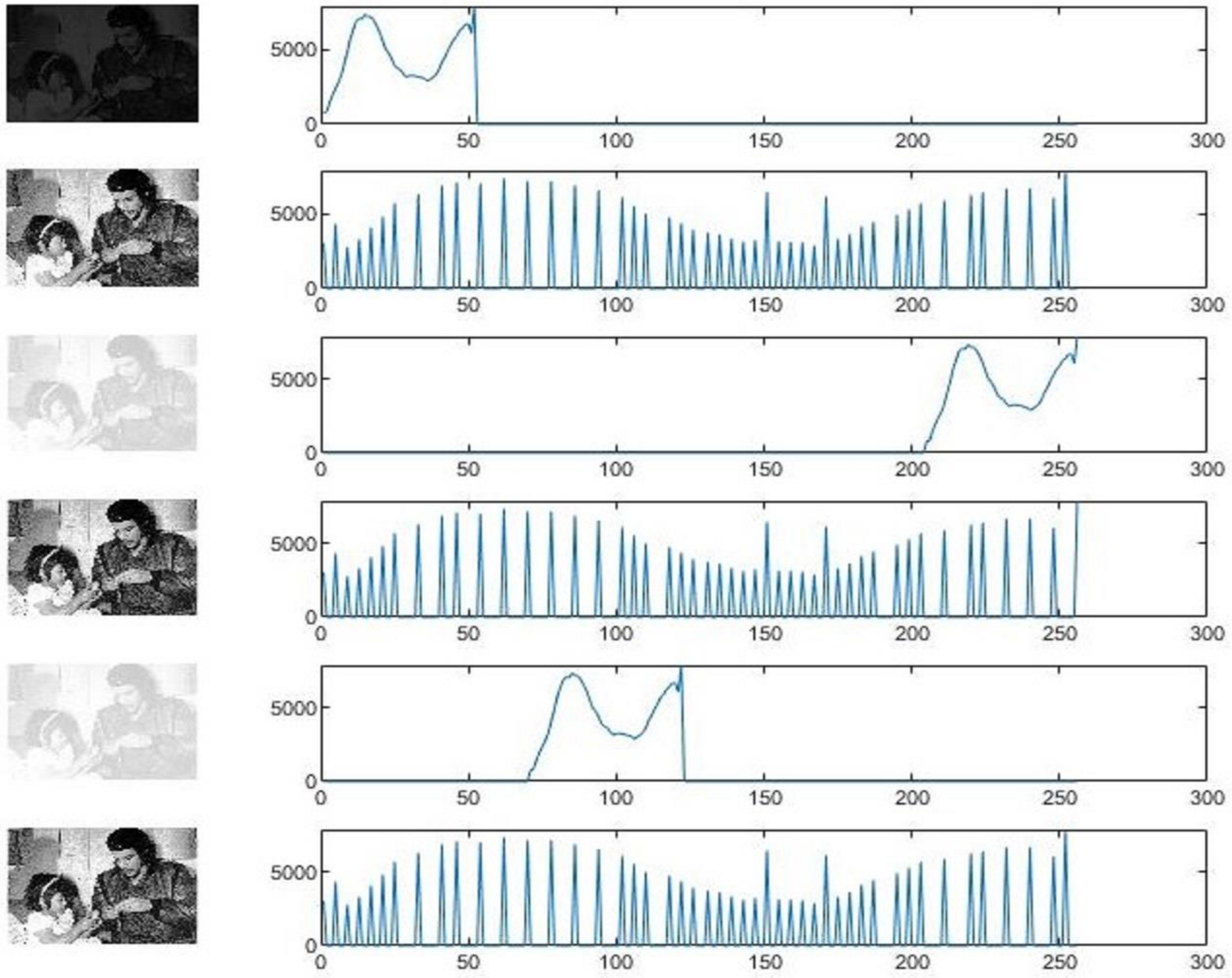


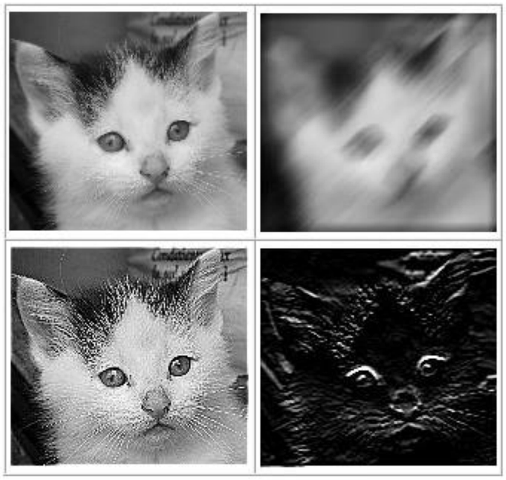
Image Filtering

There are two main types of filtering applied to images:

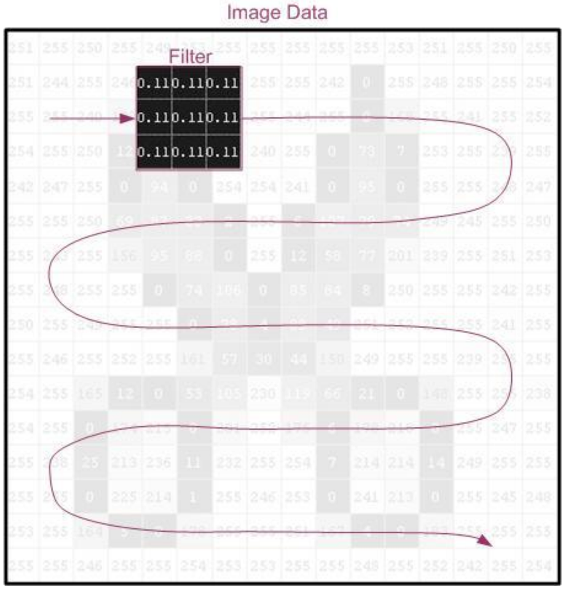
- spatial domain filtering: we are performing filtering operations directly on the pixels of an image.
- Frequency domain filtering: mean using of the Fourier Transform.

Spatial filtering

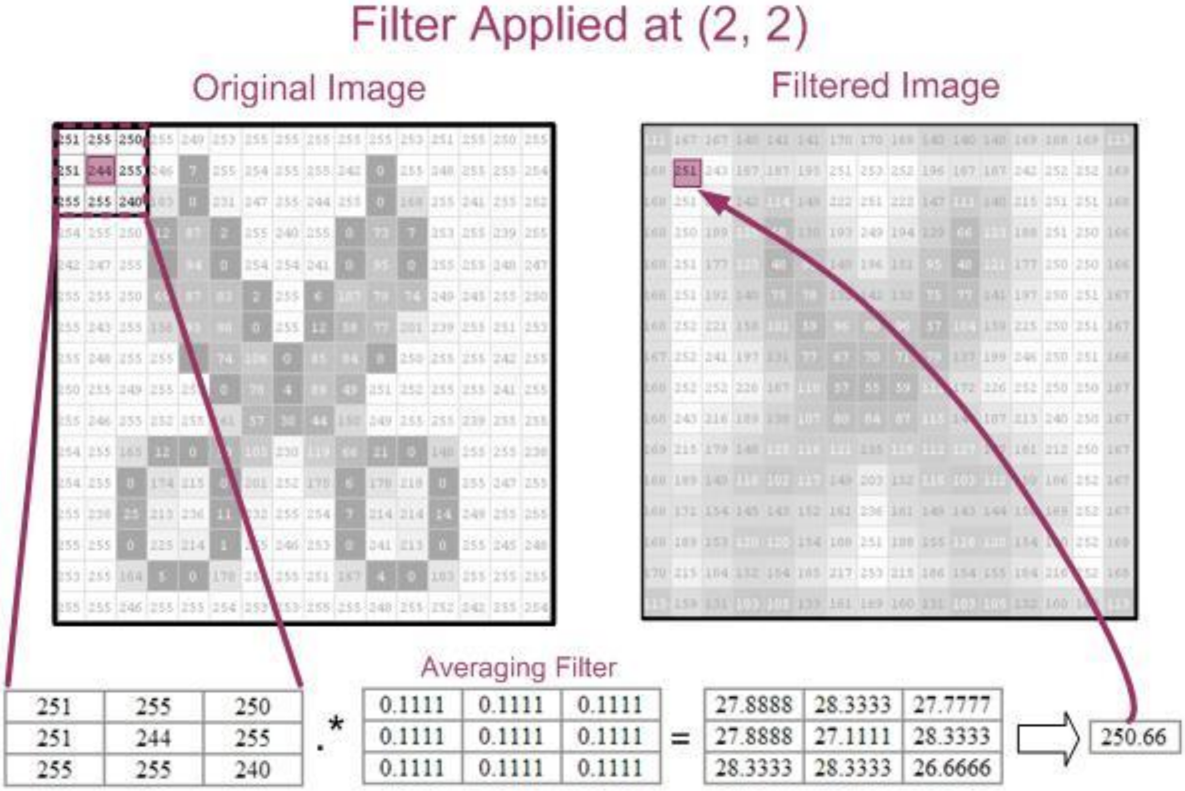
Spatial filtering is a technique that uses a pixel and its neighbors to select a new value for the pixel. The simplest type of spatial filtering is called linear filtering. It attaches a weight to the pixels in the neighborhood of the pixel of interest, and these weights are used to blend those pixels together to provide a new value for the pixel of interest. Linear filtering can be uses to ***smooth, blur, sharpen, or find the edges*** of an image. The following four images are meant to demonstrate what spatial filtering can do. The original image is shown in the upper left-hand corner.



Sometimes a linear filter is not enough to solve a particular problem. Non-linear filters are useful for smoothing only smooth areas, enhancing only strong edges or removing speckles from images. Spatial Filtering is sometimes also known as neighborhood processing. Neighborhood processing is an appropriate name because you define a center point and perform an operation (or apply a filter) to only those pixels in predetermined neighborhood of that center point. The result of the operation is one value, which becomes the value at the center point's location in the modified image. Each point in the image is processed with its neighbors. The general idea is shown below as a "sliding filter" that moves throughout the image to calculate the value at the center location.



Mean Filter: The following diagram is meant to illustrate in further details how the filter is applied. The filter (an averaging filter) is applied to location 2,2.



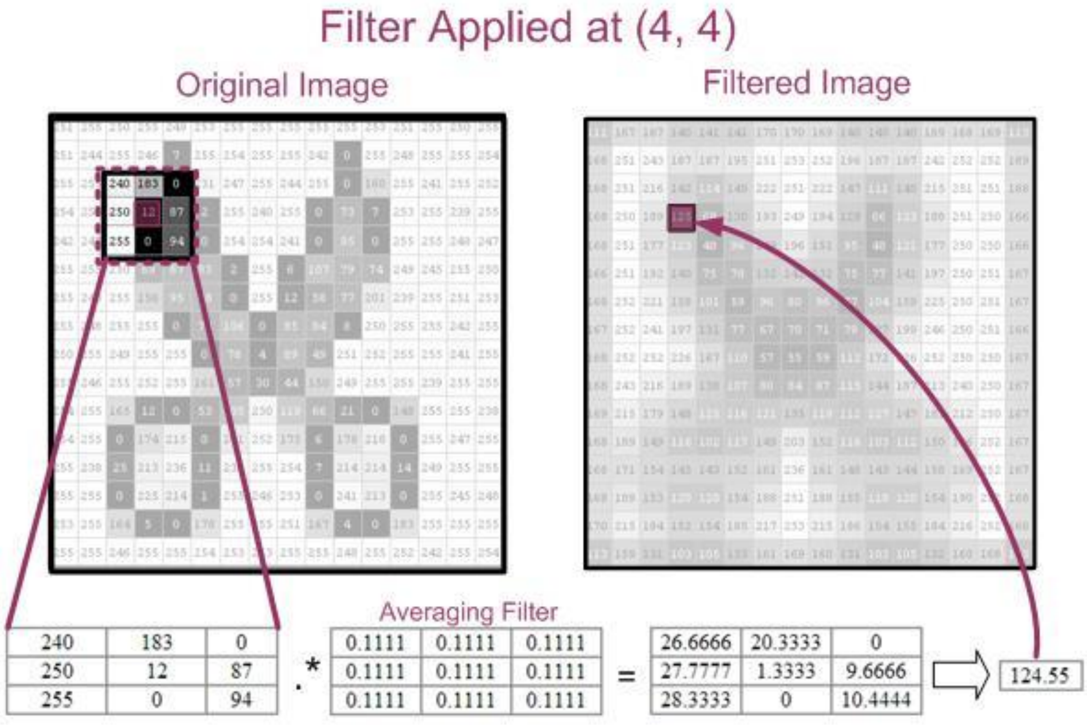
$$\begin{bmatrix} 251 & 255 & 250 \\ 251 & 244 & 255 \\ 255 & 255 & 240 \end{bmatrix} .* \begin{bmatrix} 0.1111 & 0.1111 & 0.1111 \\ 0.1111 & 0.1111 & 0.1111 \\ 0.1111 & 0.1111 & 0.1111 \end{bmatrix} = \begin{bmatrix} 27.8888 & 28.3333 & 27.7777 \\ 27.8888 & 27.1111 & 28.3333 \\ 28.3333 & 28.3333 & 26.6666 \end{bmatrix} \Rightarrow \boxed{250.66}$$

Notice how the resulting value is placed at location 2,2 in the filtered image. The breakdown of how the resulting value of 251 (rounded up from 250.66) was calculated mathematically is:

= 251*0.1111 + 255*0.1111 + 250*0.1111 + 251*0.1111 + 244*0.1111 + 255*0.1111 + 255*0.1111 + 255*0.1111 + 240*0.1111

= 27.88888 + 28.33333 + 27.77777 + 27.88888 + 27.11111 + 28.33333 + 28.33333 + 28.33333 + 26.66666 = 250.66

The following illustrates the averaging filter applied to location 4,4.



240	183	0
250	12	87
255	0	94

0.1111	0.1111	0.1111
0.1111	0.1111	0.1111
0.1111	0.1111	0.1111

26.6666	20.3333	0
27.7777	1.3333	9.6666
28.3333	0	10.4444

Once again, the mathematical breakdown of how 125 (rounded up from 124.55) was calculated is below:

= 240*0.1111 + 183*0.1111 + 0*0.1111 + 250*0.1111 + 12*0.1111 + 87*0.1111 + 255*0.1111 + 0*0.1111 + 94*0.1111 = 26.6666 + 20.3333 + 0 + 27.7777 + 1.3333 + 9.6666 + 28.3333 + 0 + 10.4444= 124.55

To apply the filter to the example above, the following w-mask convolved with the siding mask on image plane to produced smooth image,

mean filter mask

$$W=\begin{bmatrix}\frac{1}{9}&\frac{1}{9}&\frac{1}{9}\\\frac{1}{9}&\frac{1}{9}&\frac{1}{9}\\\frac{1}{9}&\frac{1}{9}&\frac{1}{9}\end{bmatrix}$$

sometime using this mask ⇒

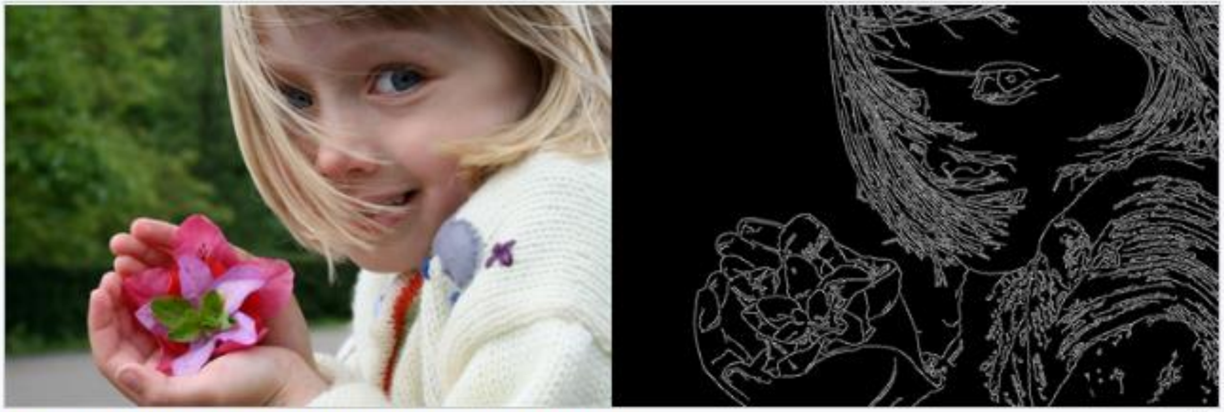
mean filter mask (weighted)

$$W=\begin{bmatrix}1&2&1\\2&4&2\\1&2&1\end{bmatrix}1/16$$

Image Edge detection

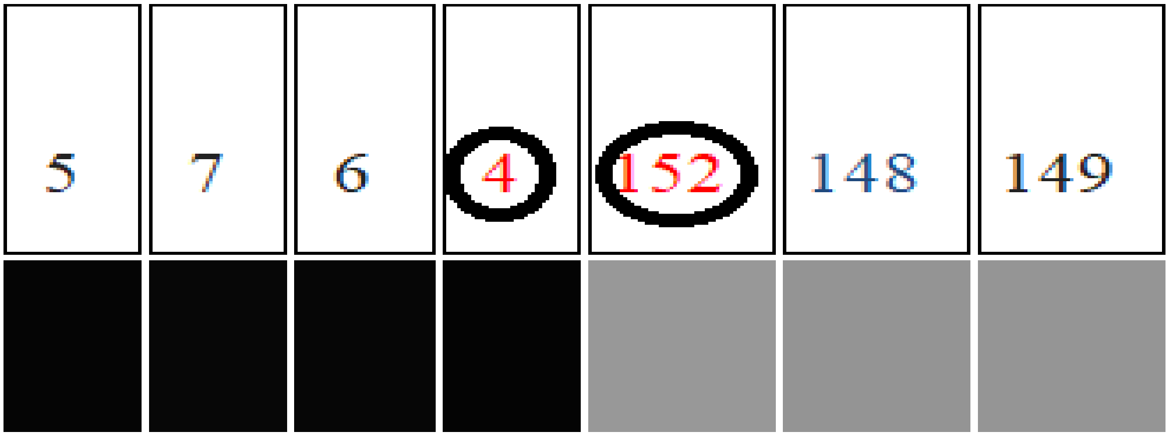
Edge detection includes a variety of mathematical methods that aim at identifying points in a digital image at which the image brightness changes sharply or, more formally, has discontinuities. The points at which image brightness changes sharply are typically organized into a set of curved line segments termed *edges*. Edge detection is a fundamental tool in image processing, machine vision and computer vision, particularly in the areas of feature detection and feature extraction. The purpose of detecting sharp changes in image brightness is to capture important events and changes in properties of the world. It can be shown that under rather general assumptions for an image formation model, discontinuities in image brightness are likely to correspond to:

- Discontinuities in depth,
- Discontinuities in surface orientation,
- Changes in material properties and
- Variations in scene illumination.



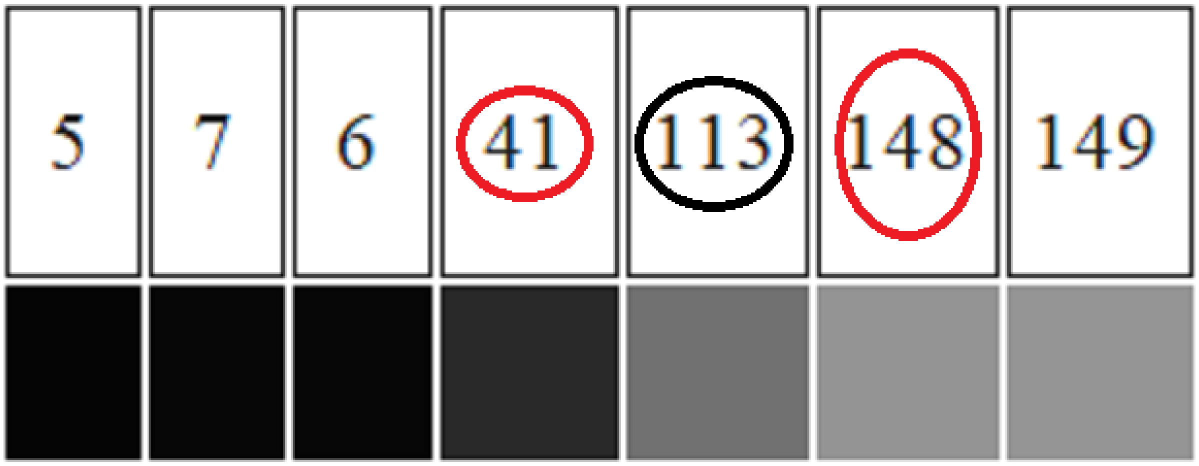
Canny edge detection applied to a photograph

To illustrate why edge detection is not a trivial task, consider the problem of detecting edges in the following one-dimensional signal. Here, we may intuitively say that there should be an edge between the 4th and 5th pixels.



5	7	6	4	152	148	149
⬛	⬛	⬛	⬛	▨	▨	▨

If the intensity difference were smaller between the 4th and the 5th pixels and if the intensity differences between the adjacent neighboring pixels were higher, it would not be as easy to say that there should be an edge in the corresponding region. Moreover, one could argue that this case is one in which there are several edges.



Hence, to firmly state a specific ***threshold*** on how large the intensity change between two neighbouring pixels must be for us to say that there should be an edge between these pixels is not always simple. Indeed, this is one of the reasons why edge detection may be a non-trivial problem unless the objects in the scene are particularly simple and the illumination conditions can be well controlled.

Robert gradient edge detector

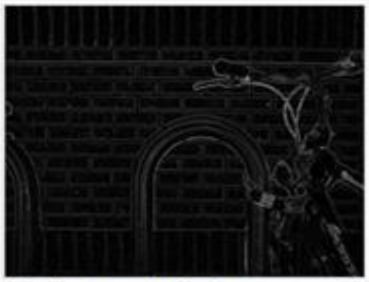
Robert’s gradient operator: is used in image processing and computer vision for edge detection. It was one of the first edge detectors and was initially proposed by Lawrence Roberts in 1963. As a differential operator, the idea behind the Roberts cross operator is to approximate the gradient of an image through discrete differentiation which is achieved by computing the sum of the squares of the differences between diagonally adjacent pixels. In order to perform edge detection with the Roberts operator we first convolve the original image, with the following two kernels:

$$\begin{bmatrix} +1 & 0 \\ 0 & -1 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 0 & +1 \\ -1 & 0 \end{bmatrix}.$$

Let I(x,y) be a point in the original image and Gx(x,y) be a point in an image formed by convolving with the first kernel and Gy(x,y) be a point in an image formed by convolving with the second kernel. The gradient can then be defined as:

$$\nabla I(x,y) = G(x,y) = \sqrt{G_x^2 + G_y^2}.$$

Grayscale test image of brick wall and bike rack



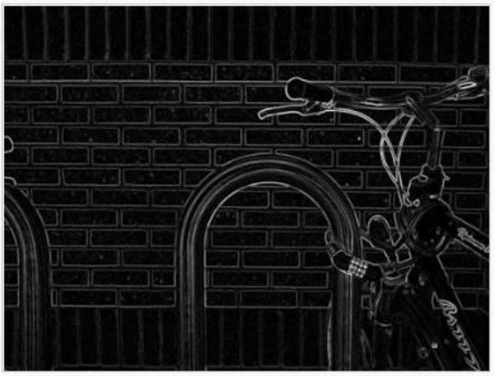
Gradient magnitude from Roberts cross operator

Prewitt operator

Mathematically, the operator uses two 3×3 kernels which are convolved with the original image to calculate approximations of the derivatives - one for horizontal changes, and one for vertical. If we define I(x.y) as the source image, and Gx and Gy are two images which at each point contain the horizontal and vertical derivative approximations, the latter are computed as:

$$\mathbf{G_x} = \begin{bmatrix} +1 & 0 & -1 \\ +1 & 0 & -1 \\ +1 & 0 & -1 \end{bmatrix} * \mathbf{I_{(x,y)}} \quad \text{and} \quad \mathbf{G_y} = \begin{bmatrix} +1 & +1 & +1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} * \mathbf{I_{(x,y)}}$$

Grayscale image of a brick wall & a bike rack



Gradient with Prewitt operator of grayscale image of a brick wall & a bike rack

Sobel operator(edge detector)

Sobel's edge detector or Sobel operator: is used in image processing and computer vision, particularly within edge detection algorithms where it creates an image emphasizing edges. It is a discrete differentiation operator, computing an approximation of the gradient of the image intensity function. At each point in the image, the result of the Sobel operator is either the corresponding gradient vector or the norm of this vector. The Sobel operator is based on convolving the image with a small, separable, and integer-valued filter(3*3 mask) in the horizontal and vertical directions and is therefore relatively inexpensive in terms of computations. On the other hand, the gradient approximation that it produces is relatively crude, in particular for high-frequency variations in the image.

The operator uses two 3×3 kernels which are convolved with the original image to calculate approximations of the derivatives one for horizontal changes, and one for vertical. If we define **(I)** as the source image, and $\mathbf{G}_x$ and $\mathbf{G}_y$ are two images which at each point contain the vertical and horizontal derivative approximations respectively, the computations are as follows:

$$\mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * \,| \quad \text{and} \quad \mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \,|$$

where (*) here denotes the 2-dimensional signal processing convolution operation.

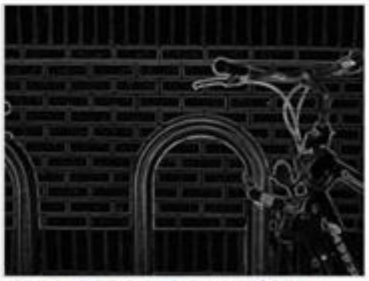
The *x*-coordinate is defined here as increasing in the "right"-direction, and the *y*-coordinate is defined as increasing in the "down"-direction. At each point in the image, the resulting gradient approximations can be combined to give the gradient magnitude, using:

$$\mathbf{G} = \sqrt{{\mathbf{G}_x}^2 + {\mathbf{G}_y}^2}$$

Can also calculate the gradient's direction:

$$\mathbf{\Theta} = \operatorname{atan}\left(\frac{\mathbf{G}_y}{\mathbf{G}_x}\right)$$

Grayscale test image of brick wall and bike rack

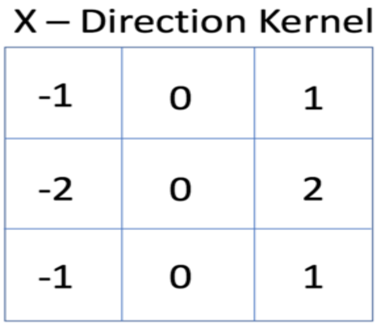


Gradient magnitude from Sobel operator

When using a Sobel Edge Detector, it is first best to convert the image from an RGB scale to a Grayscale image. Then from there, we will use what is called kernel convolution. A kernel is a 3 x 3 matrix consisting of differently (or symmetrically) weighted indexes. This will represent the filter that we will be implementing for an edge detection.

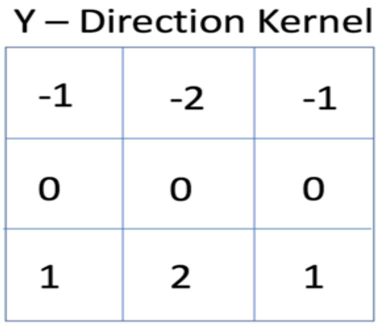
Examples:(Soble)

When we want to scan across the X direction of an image for example, we will want to use the following X Direction Kernel to scan for large changes in the gradient. Similarly, when we want to scan across the Y direction of an image, we could also use the following Y Direction Kernel to scan for large gradients as well.



X – Direction Kernel

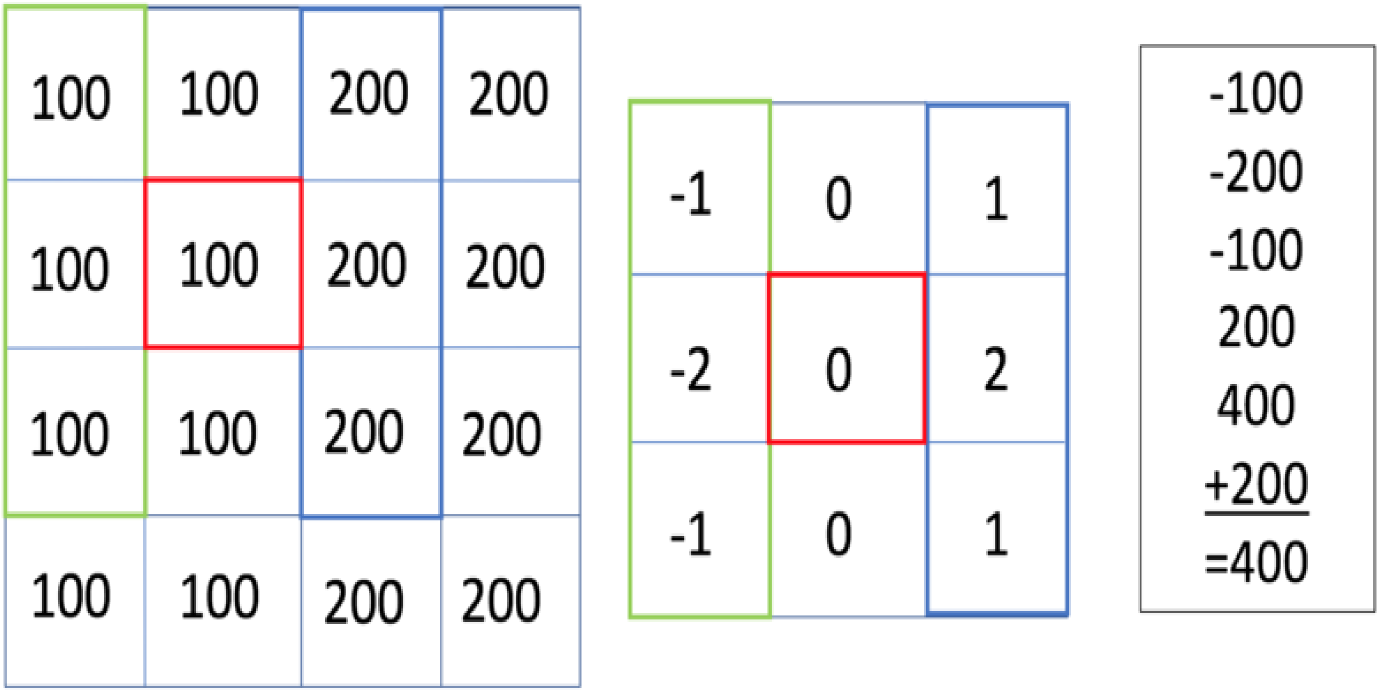
-1	0	1
-2	0	2
-1	0	1



Y – Direction Kernel

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

By using Kernel Convolution, we can see in the example image below there is an edge between the column of 100 and 200 values.



100	100	200	200
100	100	200	200
100	100	200	200
100	100	200	200

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

-100
-200
-100
200
400
+200
=400

Kernel Convolution: The bigger the value at the end, the more noticeable the edge will be.

This Kernel Convolution is an example of an X Direction Kernel usage. If an image were scanning from left to write, we can see that if the filter was set at **(2,2)** in the image above, it would have a value of **400** and therefore would have a fairly prominent edge at that point.

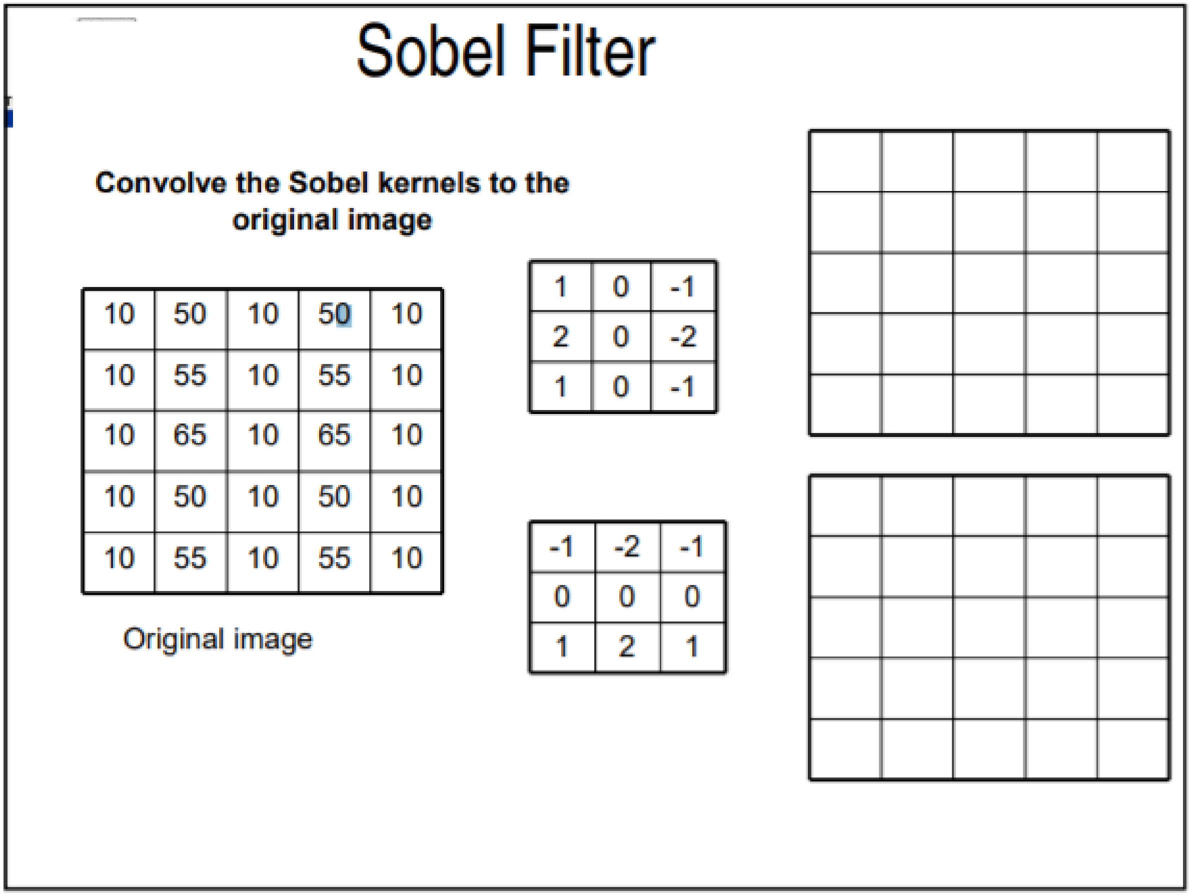
Here in order to determine whether the point (2,2) is an edge, we must compare the value of the gradient 400 with a specific threshold (th), so if it is greater than the threshold, then it is an edge, and we place a value of 255 for it, otherwise it is not an edge, we place a value of zero for it i.e

$$\text{Point } (2,2) = \begin{cases} 0 & \text{not edge if gradient } G<th \\ 255 & \text{represnet edge when } G>=th \end{cases}$$

If th=300 then point (2,2) represent edge point where(G=400>th=300).

If a user wanted to exaggerate the edge, then the user would need to change the filter values of -2 and 2 to higher magnitude. Perhaps -5 and 5. This would make the gradient of the edge larger and therefore, more noticeable. Once the image is processed in the X direction, we can then process the image in the Y direction. Magnitudes of both the X and Y kernels will then be added together to produce a final image showing all edges in the image.

H.W.



Sobel Filter

Convolve the Sobel kernels to the original image

10	50	10	50	10
10	55	10	55	10
10	65	10	65	10
10	50	10	50	10
10	55	10	55	10

Original image

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

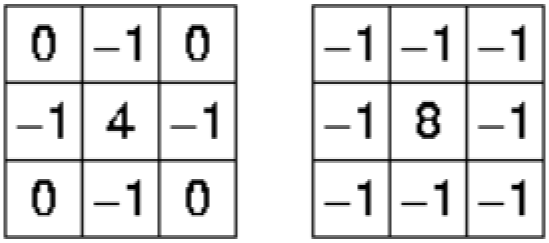
-1	-2	-1
0	0	0
1	2	1

Laplacian filter

The Laplacian *L(x,y)* of an image with pixel intensity values *I(x,y)* is given by:

$$L(x,y) = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$$

This can be calculated using a convolution filter. Since the input image is represented as a set of discrete pixels, we have to find a discrete convolution kernel that can approximate the second derivatives in the definition of the Laplacian. Two commonly used small kernels are shown follow.



0	−1	0
−1	4	−1
0	−1	0

−1	−1	−1
−1	8	−1
−1	−1	−1

Two commonly used discrete approximations to the Laplacian filter. (Note, we have defined the Laplacian using a negative peak because this is more common; however, it is equally valid to use the opposite sign convention.)

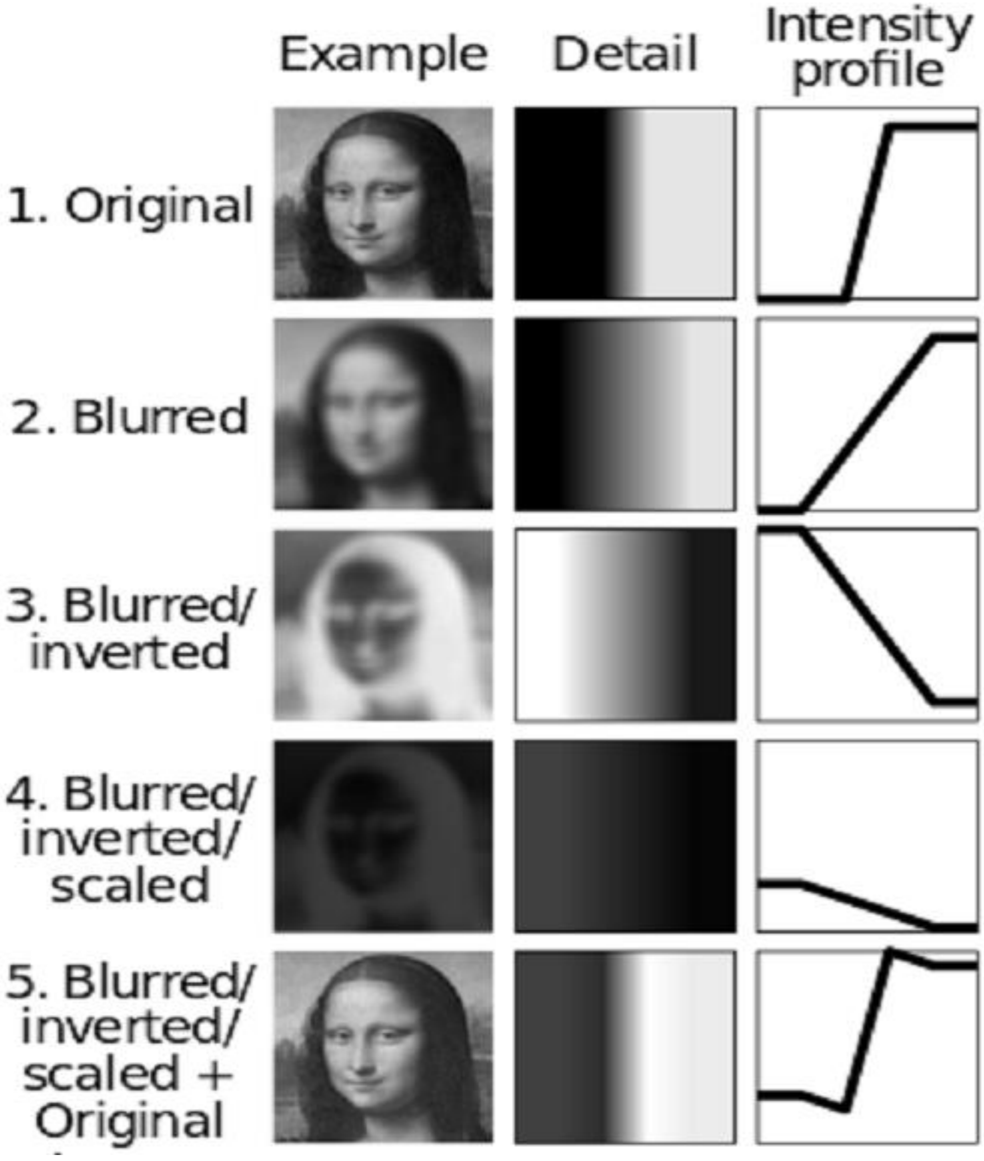
Laplacian Operator is also a derivative operator which is used to find edges in an image. The major difference between Laplacian and other operators like Robert’s , Sobel, is that these all are first order derivative masks but Laplacian is a second order derivative mask.

In this mask we have two further classifications one is Positive Laplacian Operator and other is Negative Laplacian Operator.

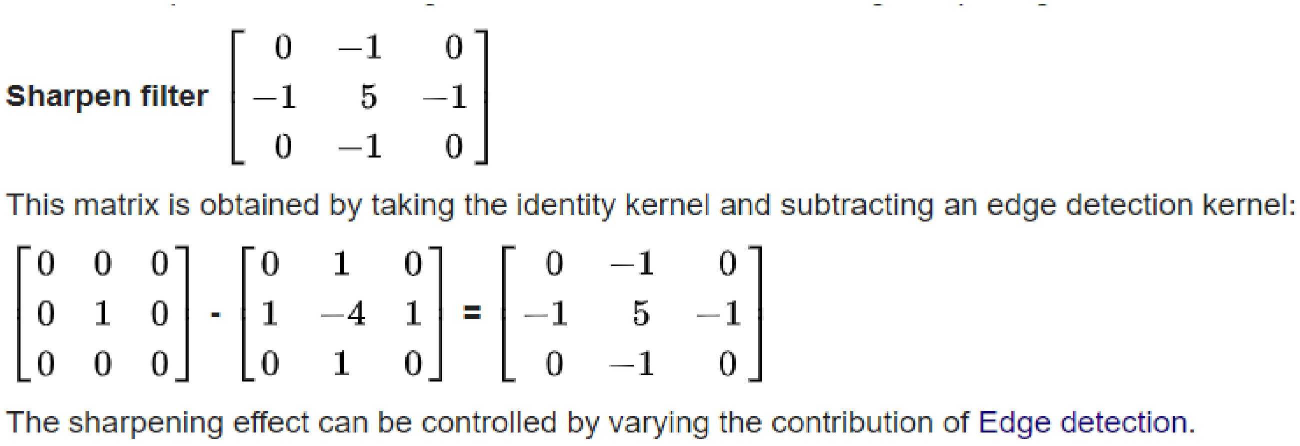
Another difference between Laplacian and other operators is that unlike other operators Laplacian didn’t take out edges in any particular direction but it take out edges in following classification.

Sharpen filters(Enhancement filters)

Image sharpening process is emphasize the texture and fine details in the image and increase its focus. Any imaging system always blurs an image to some extent. That is why sharpening image enhancement is a very useful process that can sharpen blurry image and make it clearer. As a result, a image appears to be a bit too soft and fuzzy, so it needs correction. When used image sharpening filters will enhance image edges and makes it look brighter and more precise.



In the example below, the image is convolved with the following sharpening filter:

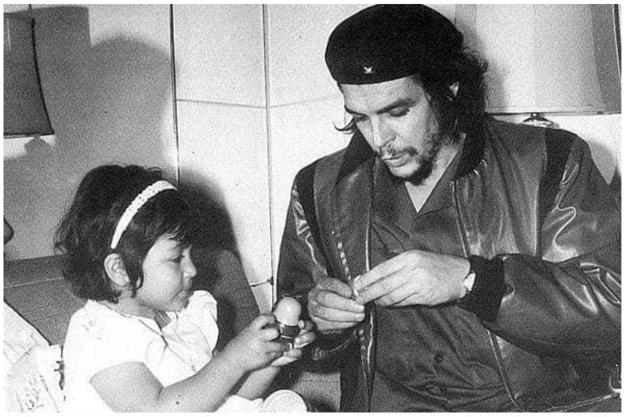


Sharpen filter $\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$

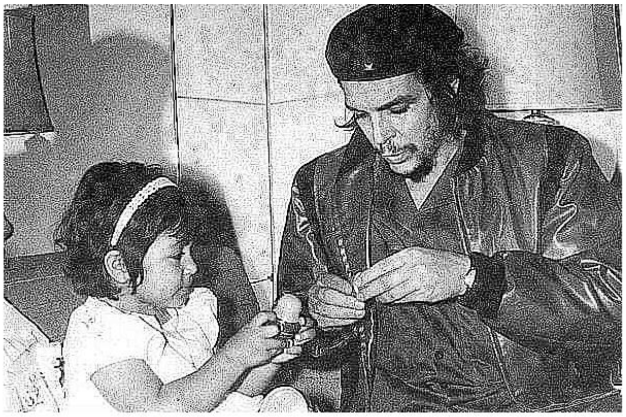
This matrix is obtained by taking the identity kernel and subtracting an edge detection kernel:

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

The sharpening effect can be controlled by varying the contribution of Edge detection.



Original image



Sharpening image