Topics on High-Dimensional Data Analytics

Image Analysis

Kamran Paynabar, Ph.D.

Associate Professor

School of Industrial and Systems Engineering

Introduction to Image Processing

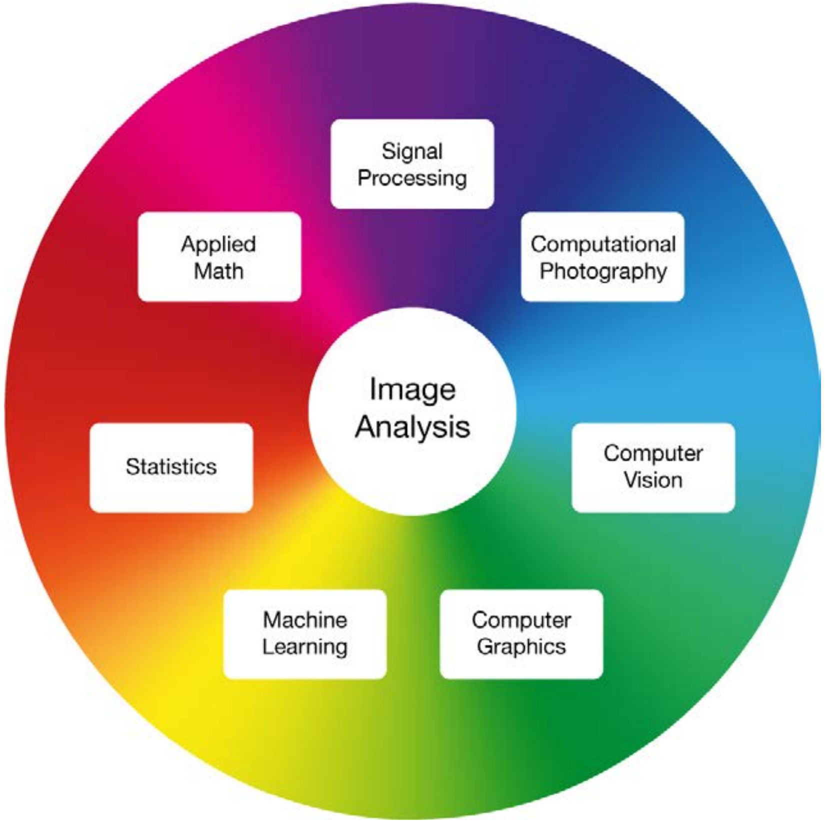
Learning Objectives

- How to quantify an image
- Define difference between, RGB, Gray scale and BW and their conversions
- How to control size and resolution
- Use basic functions in Matlab

Image Analysis Levels

- Image analysis is the process of processing raw images and extracting useful information for decision making.
- Level 0: Image representation (acquisition, sampling, quantization, compression)
- Level 1: Image to Image transformations (enhancement, filtering, restoration, smoothing, segmentation)
- Level 2: Image to vector transformation (feature extraction and dimension reduction)
- Level 3: Feature to decision mapping

Image Analysis



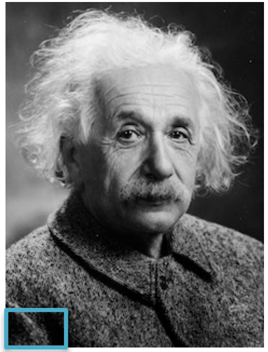
What is an Image?

- A gray (color-RGB) image is a 2-D (3-D) light intensity function, f(x1,x2), where f measures brightness at position f(x1,x2).

- A digital gray (color) image is a representation of an image by a 2-D (3-D) array of discrete samples.

- Pixel is referred to an element of the array.

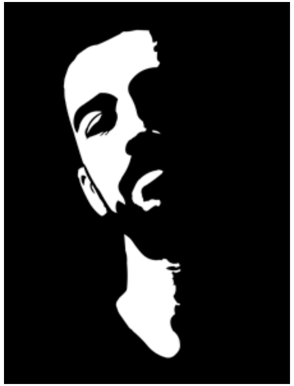
10	8	12	14	9	17	29	30	6	17	18
8	8	12	10	9	21	30	26	9	20	21
13	12	13	14	12	12	11	7	8	13	13
11	10	10	15	19	16	12	12	15	16	11
13	14	10	10	18	19	18	24	22	23	16
14	19	14	9	17	26	32	38	24	25	21
10	16	15	10	15	25	32	34	31	30	33
16	15	16	14	12	14	14	10	32	34	40
24	18	21	27	24	23	23	17	22	29	36
39	22	19	28	24	34	21	8	10	25	41
38	14	21	26	27	21	27	30	17	22	36
29	19	31	30	37	43	37	15	36	30	42
10	28	42	37	25	47	37	12	38	57	46
19	19	25	26	13	30	33	30	31	83	48
29	13	24	24	26	28	34	29	34	61	54
17	21	46	42	33	29	31	23	31	18	49
21	23	35	44	31	37	28	18	24	8	41



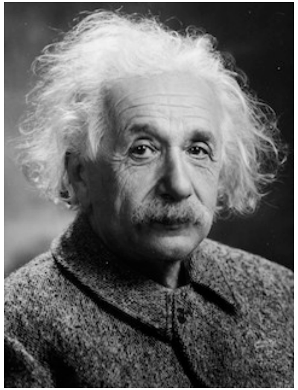
Example:
Gray image: 240*334 pixels

Examples

Black and White image



Gray image



Color image

Basic functions in MATLAB

- Read an Image
  ```
  I = imread('rice.png');
  ```

- Show an Image
  ```
  imshow(I)
  ```

- Save an Image
  ```
  imwrite(I,'rice.jpg','jpg')
  ```

- Images are stored in uint8 format, we may need to convert to other formats for further analysis
  ```
  I = double(I)
  ```

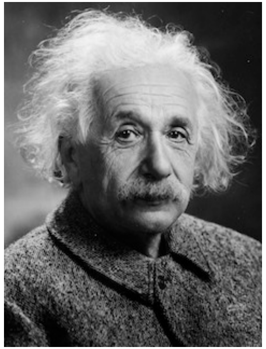
Image color conversion

- RGB to Gray

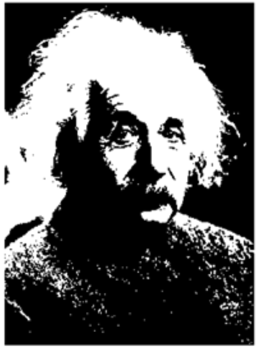
```
X= imread('peppers.jpg');
I = rgb2gray(X);
figure; imshow(I)
```

- RGB/Gray to BW

```
I= imread('Einstein.jpg');
BW = im2bw(I,level)
```



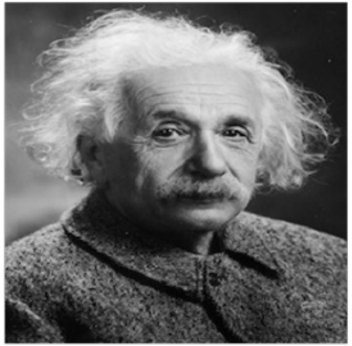
Level = 0.7



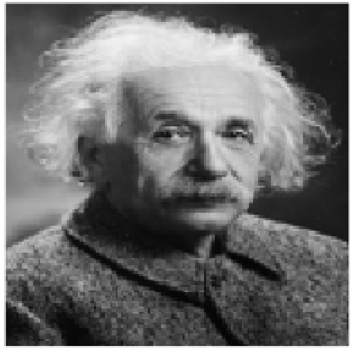
Level = 0.5

Size and Resolution of Image

- Size and Resolution:

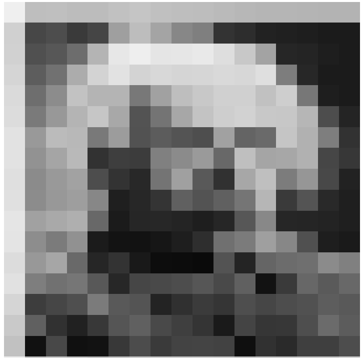


256×256



128×128

64×64



32×32

- Size and resolution refers to the number of pixels in the image horizontally and vertically
- MATLAB examples:

```
I = imread('Einstein.jpg');
J = imresize(I, 0.5);
figure, imshow(I), figure, imshow(J)
```

Topics on High-Dimensional Data Analytics

Image Analysis

Kamran Paynabar, Ph.D.

Associate Professor

School of Industrial and Systems Engineering

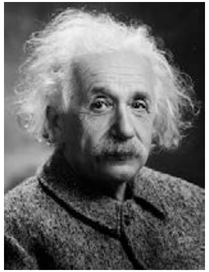
Image Transformation

Learning Objectives

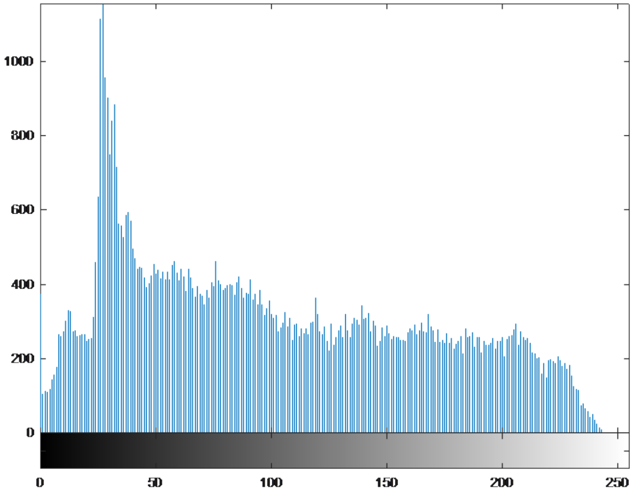
- Use image transformation techniques
- Define histogram of an image and how to shift and stretch it
- How to enhance an image using transformation functions

Image Histogram

- Histogram represents the distribution of gray levels
  - For all pixels x[m,n], count all x[m,n] = I
  - The x axis of the histogram shows the range of pixel values and the y axis is the frequency.



- The histogram is an estimate of the probability density function (pdf) of the underlying random process



- Matlab: imhist(I,N)
  - Display a histogram for the image I using N bins
  - Default: N = 256

MATLAB Example

```
I = imread('coins.png');
imhist(I)
```

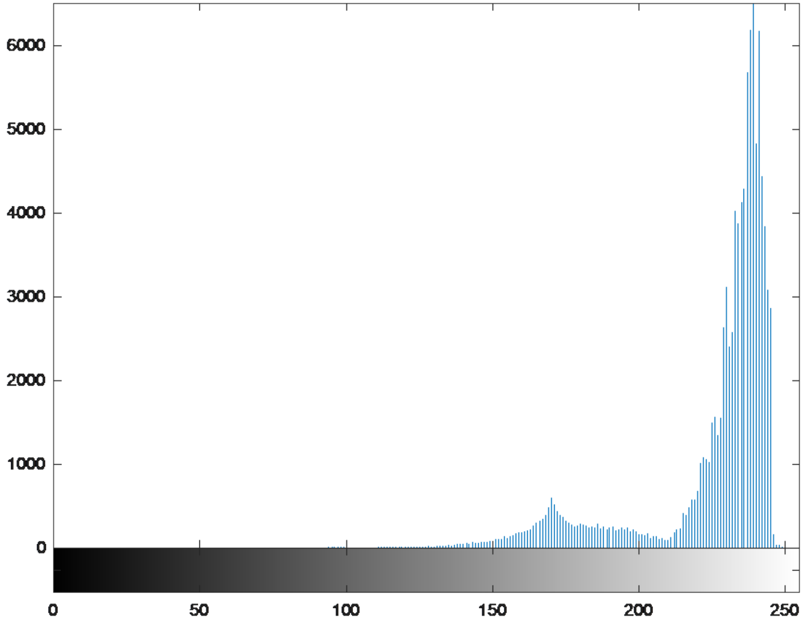
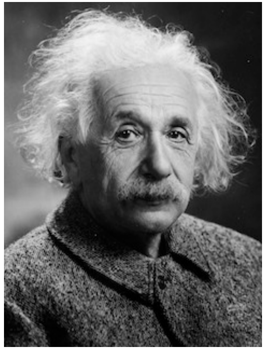


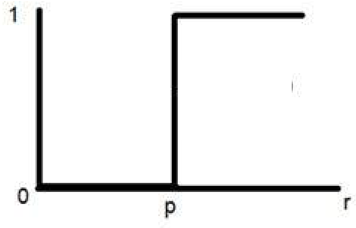
Image Transformation

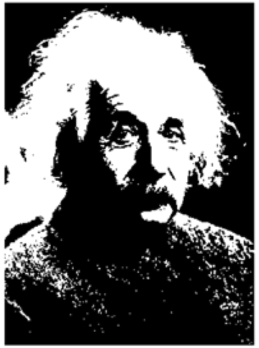
- Images can be transformed by applying a function on the image matrix,

$$g(x, y) = T(f(x, y))$$

- For example if a thresholding function is sued as the transformation function a gray-scale image can be converted to a BW image.







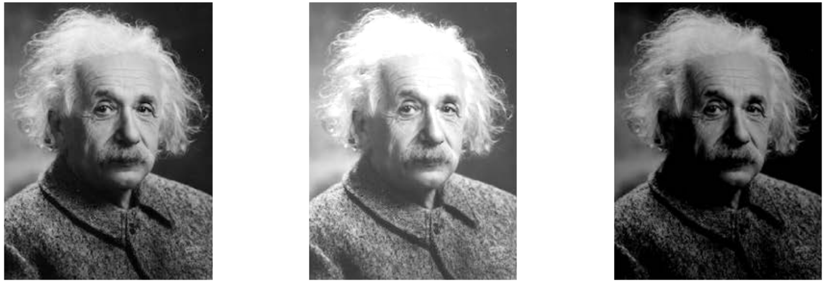
$$g(x, y) = T\big(f(x, y)\big) = \begin{cases} 1 & f(x, y) > p \\ 0 & f(x, y) \leq p \end{cases}$$

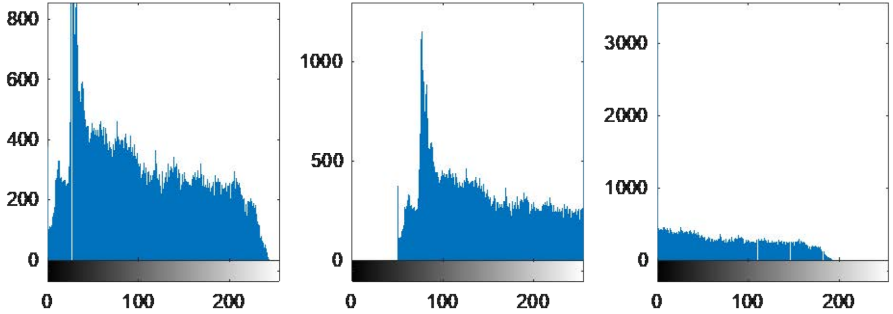
Transformation – Shifting Histogram

- The brightness of an image can be changed by shifting its histogram.
- Suppose the pixel values are ranged between L and U (0 and 255 for gray-scale). The transformation function is defined by

$$g(x,y) = T\big(f(x,y)\big) =$$

$$\begin{cases} U & f(x,y) > U - s \\ f(x,y) + s & \text{otherwise} \\ L & f(x,y) \le L - s \end{cases}$$





$s = 0$ $s = 50$ $s = -50$

Transformation – Stretching Histogram

- The contrast of an image is defined by the difference between maximum and minimum pixel intensity.

- It can be changed by stretching the histogram. Suppose the pixel values are ranged between L and U. The transformation function is defined by

$$g(x,y) = T\big(f(x,y)\big) = \left(\frac{f(x,y) - \min\big(f(x,y)\big)}{\max\big(f(x,y)\big) - \min\big(f(x,y)\big)}\right)\lambda$$

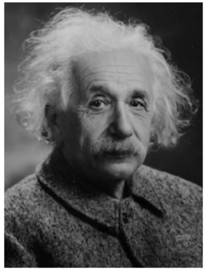
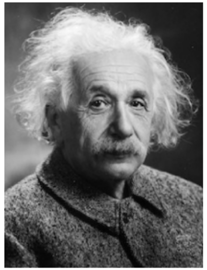
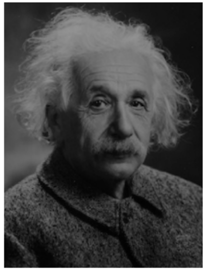
Transformation – Stretching Histogram

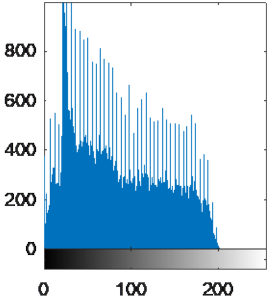
$$g(x,y) = T\big(f(x,y)\big) = $$

$$\left(\frac{f(x,y) - \min\big(f(x,y)\big)}{\max\big(f(x,y)\big) - \min\big(f(x,y)\big)}\right)\lambda$$

```
D=double(I);
imshow(I1=uint8((D-min(min(D)))/(max(max(D))-min(min(D)))*205))
```

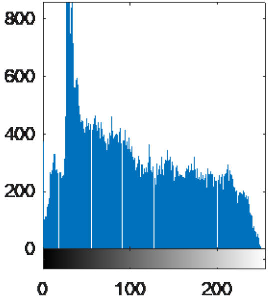
Original contrast = 248



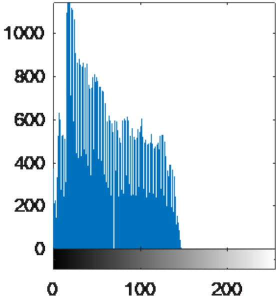
$\lambda = 205$

Contrast= 205



$\lambda = 255$

Contrast= 255



$\lambda = 150$

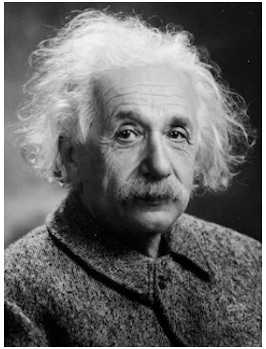
Contrast= 150

Gray Level Resolution (Bit Depth)

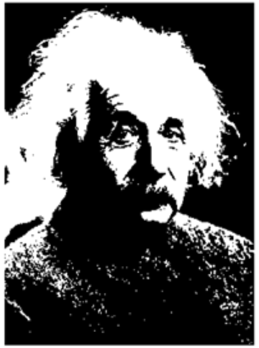
- Gray level resolution refers to change in the shades or levels of gray in an image.

- The number of different colors in an image is depends on bits per pixel (bpp).

$$L = 2^{bpp}$$



$L = 2^8$



$L = 2^1$

Gray Level Transformation

- Gray level transformation is often used for image enchantment.
- Three typical transformation functions are
  - Linear (negative image)

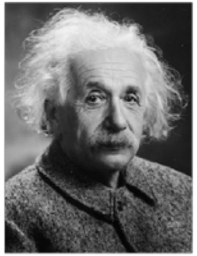
$$g(x,y) = T\big(f(x,y)\big) = (L-1) - f(x,y)$$

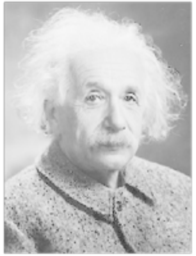
  - Log

$$g(x,y) = T\big(f(x,y)\big) = c\log(f(x,y)+1)$$

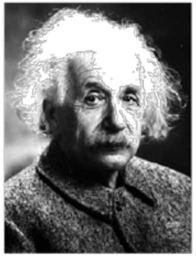
  - Power-Law

$$g(x,y) = T\big(f(x,y)\big) = cf(x,y)^{\gamma}$$





$c = 45$



$c = 0.1$
$\gamma = 1.5$

Topics on High-Dimensional Data Analytics

Image Analysis

Kamran Paynabar, Ph.D.

Associate Professor

School of Industrial and Systems Engineering

Convolution and Image Filtering

Learning Objectives

- Define convolution concept
- Apply image filtering using image convolution with masks
- How to denoise and sharpen an image using convolution.

Convolution

- The convolution of functions *f* and *g* is defined by

$$(f * g)(t) \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} f(\tau) g(t - \tau)\, d\tau$$
$$= \int_{-\infty}^{\infty} f(t - \tau) g(\tau)\, d\tau.$$

Continuous

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m] g[n - m]$$
$$= \sum_{m=-\infty}^{\infty} f[n - m] g[m].$$

Discrete

- Convolution is widely used in image processing for denoising, blurring, sharpening, embossing, and edge detection.

Image Filtering

- Image filtering is a convolution of a mask (aka kernel, and convolution matrix) with an image that can be used for blurring, sharpening, edge detection, etc.
- A mask is a matrix convolved with an image.

Edge Detection

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

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

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

. . .

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

Sharpening

. . .

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

Blurring

Image Convolution with a Mask

- Flip the mask (kernel) both horizontally and vertically.

- Put the center element of the mask at every pixel of the image. Multiply the corresponding elements and then add them up. Replace the pixel value corresponding to the center of the mask with the resulting sum.

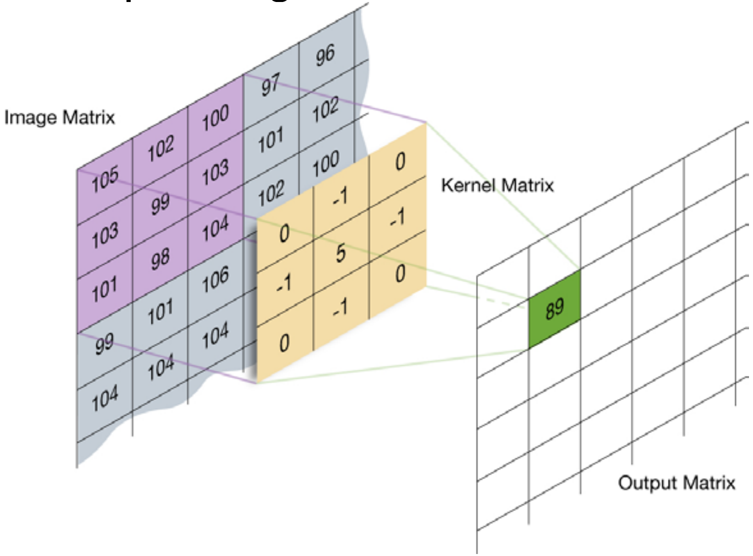
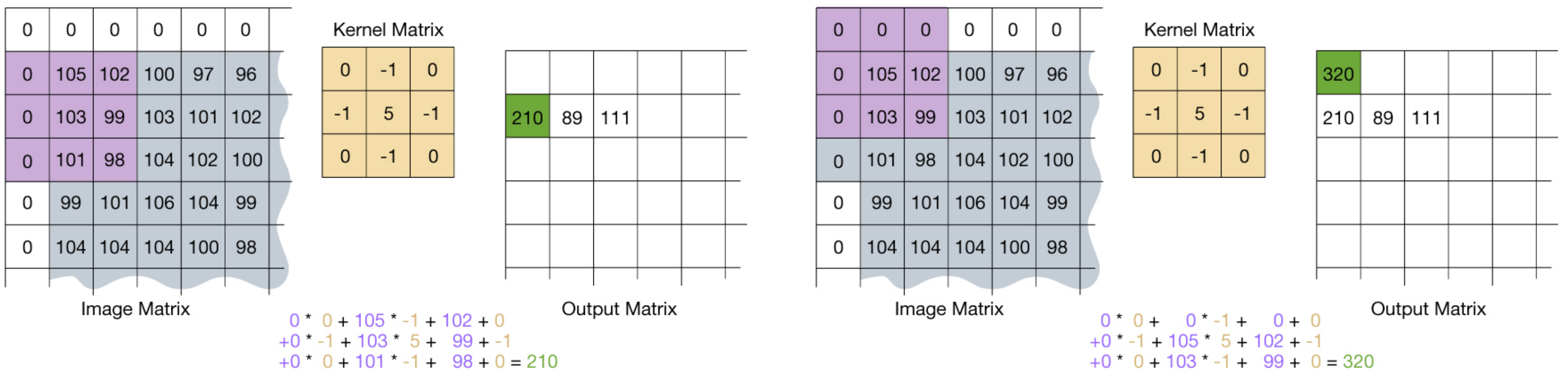


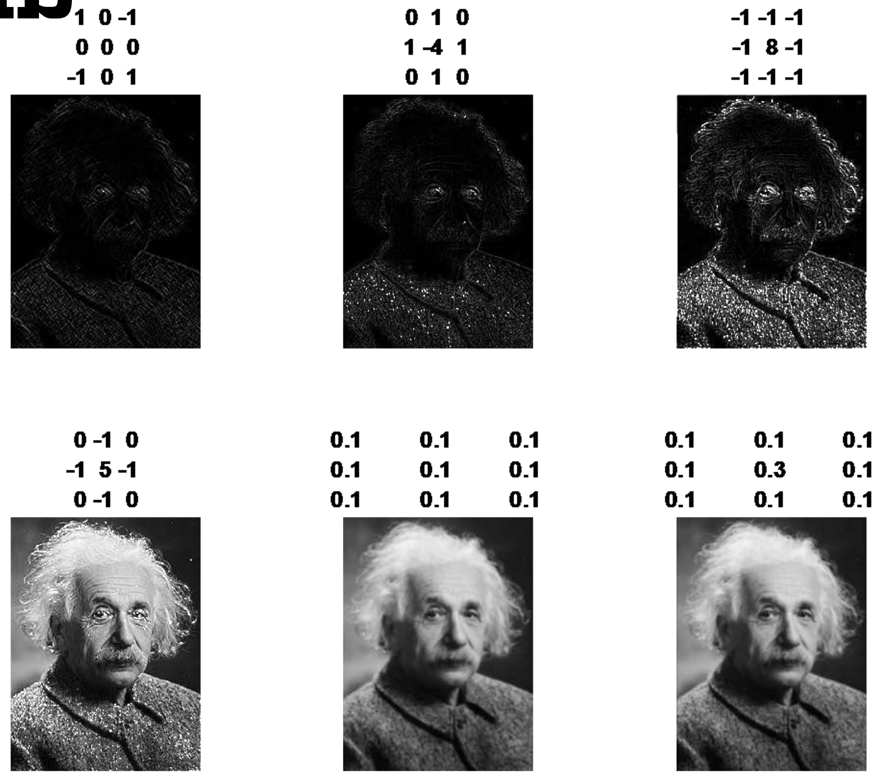
Image Convolution with a Mask

- For the pixels on the border of image matrix, some elements of the mask might fall out of the image matrix. In this case, we can extend the image by adding zeros. This is known as padding.



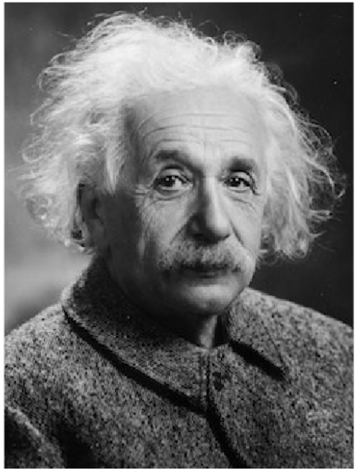
Example in Matlab

```
Y = imread('einstein.jpg');
K = cell(6,1);

K{1} = [1 0 -1; 0 0 0; -1 0 1];       %Edge Detection
K{2} = [0 1 0;1 -4 1;0 1 0];          %Edge Detection
K{3} = [-1 -1 -1;-1 8 -1;-1 -1 -1];   %Edge Detection
K{4} = [ 0 -1 0; -1 5 -1;0 -1 0];     %Sharpening
K{5} = ones(3,3)/9;                   %Blurring
K{6} = [ 1 2 1;2 4 2;1 2 1]/16;       %Blurring

Yk = cell(6,1);

for i = 1:6
    Yk{i} = imfilter(Y,K{i});
    subplot(2,3,i)
    imshow(Yk{i})
    title(num2str(round(K{i},1)))
end
```

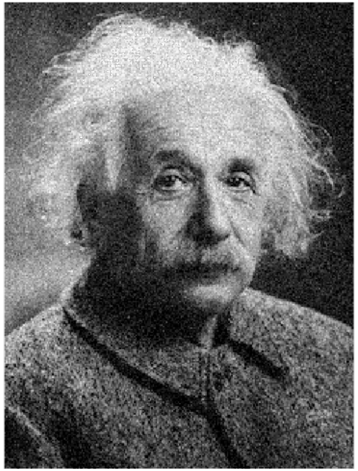


More types of masks can be found in: https://lodev.org/cgtutor/filtering.html

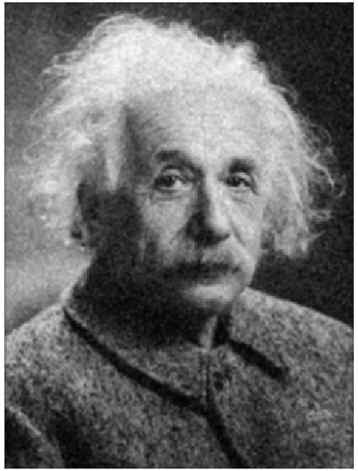
Denoising using Blurring Mask



Original Albert



Noisy Albert



Denoised Albert

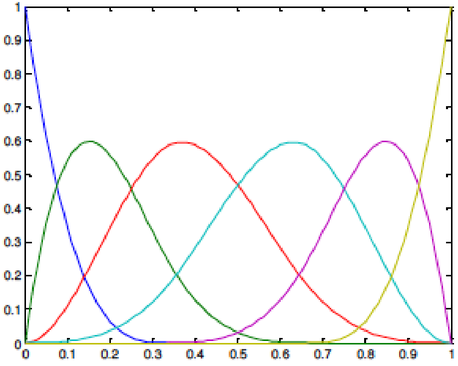
```
Y = imread('einstein.jpg');
K = [ 1 2 1;2 4 2;1 2 1]/16
I1 = uint8(double(Y)+normrnd(0,20,320,240));
I2 = imfilter(I1,K);
```

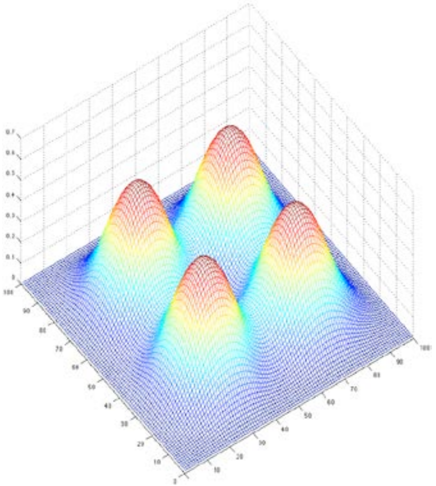
Denoising of Smooth Images using Splines

- Another approach for denoising smooth images is to use local regression with smooth basis (e.g., splines)
- Using Kronecker product, a 2D-spline basis can be generated from 1D basis matrices:

$$B = B_2 \otimes B_1$$
$$\hat{H} = B(B'B)^{-1}B'$$

$$\hat{H}_i = B_i(B_i'B_i)^{-1}B_i'$$
$$\hat{H} = \hat{H}_2 \otimes \hat{H}_1$$
$$\hat{y} = (\hat{H}_2 \otimes \hat{H}_1)y \text{ or } \hat{Y} = \hat{H}_1 Y \hat{H}_2$$





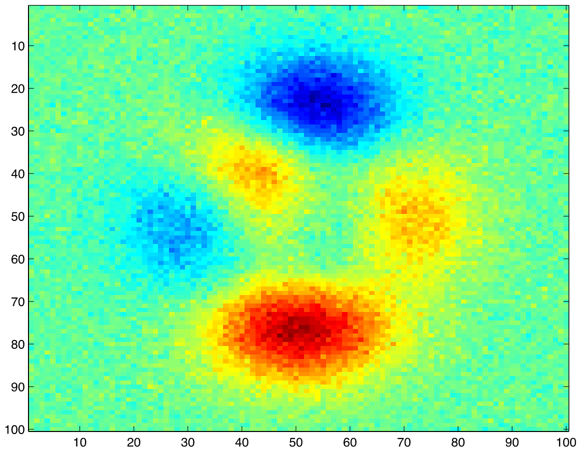
Example

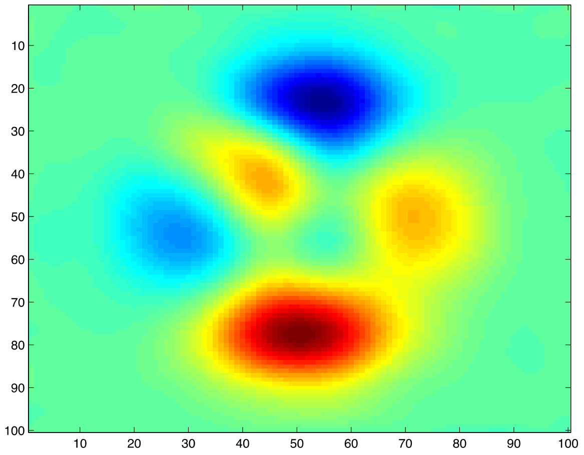
- 2D example: Generate data

```
n = 100;
sigma = 0.5;
Y = peaks(n) + randn(n)*sigma;
imagesc(Y)
```

- 2D Spline

```
sd = 10;
knots = [ones(1,sd-1)...
    linspace(1,n,10) n * ones(1,sd-1)];
nKnots = length(knots) - sd;
kspline = spmak(knots,eye(nKnots));
H = cell(2,1); B=cell(2,1);
for i = 1:2
    B{i}=spval(kspline,1:n)';
    H{i} = B{i}/(B{i}'*B{i})*B{i}';
end
Yhat = H{2}*Y*H{1};
imagesc(Yhat)
```





Topics on High-Dimensional Data Analytics

Image Analysis

Kamran Paynabar, Ph.D.

Associate Professor

School of Industrial and Systems Engineering

Image Segmentation

Learning Objectives

- Define image segmentation
- Apply the Otsu’s model for image segmentation
- Use K-means for partitioning images

Image Segmentation

- The main goal of image segmentation is to partition an image into multiple sets of pixels (segments).

- Image segmentation has been widely used for object detection, face and fingerprint recognition, medical imaging, video surveillance, etc.

- Various methods exist for image segmentation including
  - Local and global thresholding
  - Otsu’s method
  - K-means clustering

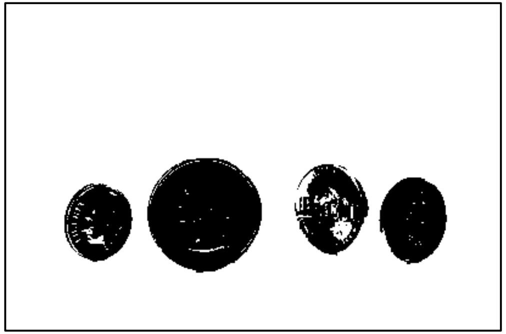
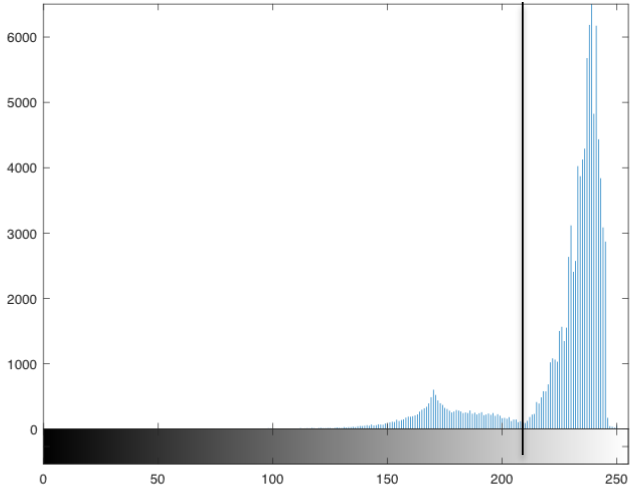


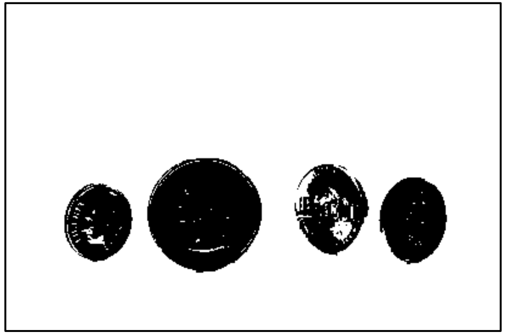
Image Segmentation – Thresholding

- Thresholding is a simple segmentation approach that converts grayscale image to binary image by applying the thresholding function on histogram.

$$p(x,y) = \begin{cases} \text{white} & p(x,y) \geq t \\ \text{black} & p(x,y) < t \end{cases}$$

t is the threshold





Otsu's Method

- The goal is to automatically determine the threshold *t* given an image histogram.
- Formulation:
  - Determine *t* by minimizing (maximizing) the intra-class variance (inter-class variance), defined by

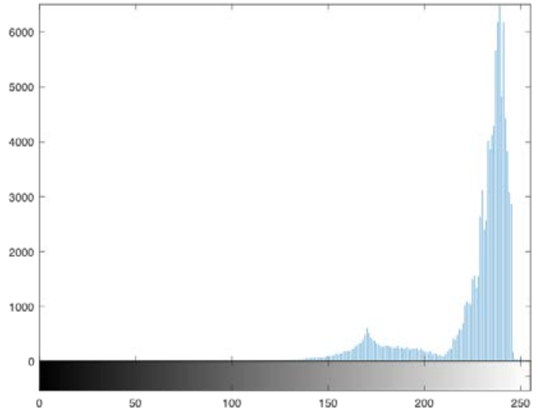
$$\sigma_\omega^2(t) = \omega_1(t)\,\sigma_1^2(t) + \omega_2(t)\,\sigma_2^2(t)$$

  - Weight $\omega_i(t)$ are the probabilities of the two classes separated by threshold t and $\sigma_i^2(t)$ variance of these classes.
  - Class probability: $\omega_1(t) = \sum_{i=0}^{t} p(i)\,, \omega_2(t) = 1 - \omega_1(t)$
  - Class mean: $\mu_1(t) = \frac{\left(\sum_{i=0}^{t} ip(i)\right)}{\omega_1}; \mu_2(t) = \frac{\left(\sum_{i=t}^{255} ip(i)\right)}{\omega_2}$
  - Inter class variance: $\sigma_b^2(t) = \sigma^2 - \sigma_\omega^2(t) = \omega_1(t)\,\omega_2(t)\big(\mu_1(t) - \mu_2(t)\big)^2$

Otsu’s Method in MATLAB

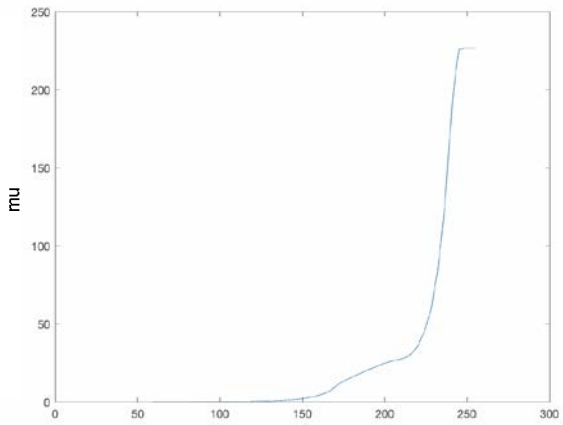
- Step 1: get the histogram of image

```
Y = imread('coins.png');
I = im2uint8(Y(:));
num_bins = 256;
counts = imhist(I,num_bins);
```



- Step 2: Calculate group mean

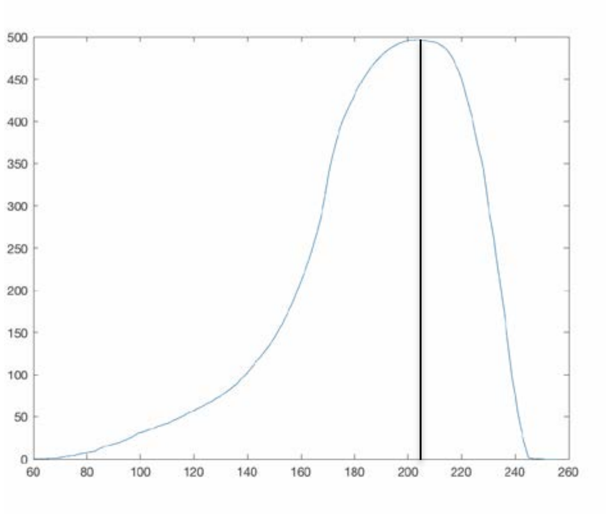
```
p = counts / sum(counts);
omega1 = cumsum(p); omega2 = 1-cumsum(p);
mu = cumsum(p .* (1:num_bins)’); mu_T = mu(end);
mu1 = mu./omega1; mu2 = (mu_T-mu1)./omega2;
```



Otsu's Method in MATLAB—Continued

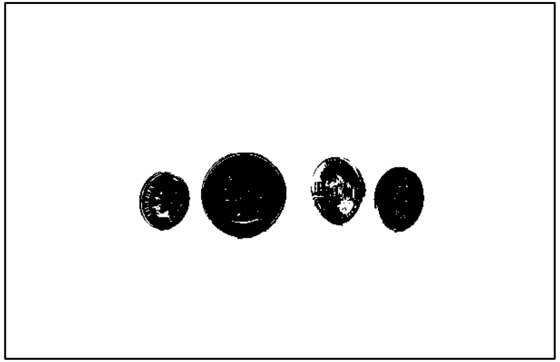
- Step 3: find the maximum value of $\sigma_b^2(t)$

```
sigma_b_squared = (mu1- mu2).^2 (omega1.* omega2);
maxval = max(sigma_b_squared);
idx = mean(find(sigma_b_squared == maxval));
```



- Step 4: Thresholding and get final image

```
level = (idx - 1) / (num_bins - 1);
BW = Y > level*256
figure, imshow(BW)
```



Otsu’s Method in MATLAB—Continued

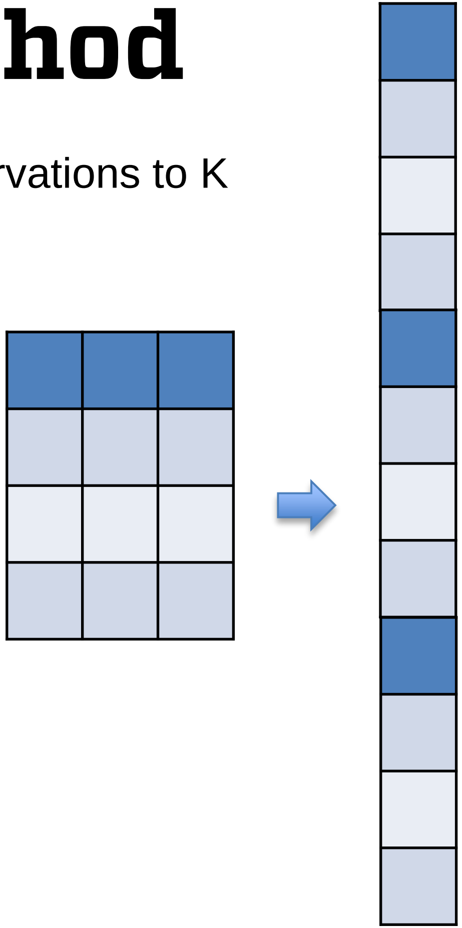
Y = imread('coins.png’);
Thresh = multithresh(Y,3);
Imshow(imquantize(Y,thresh)),[])

K-Means Clustering Method

- K-means clustering is a method for partitioning a set of observations to K clusters, such that the within-cluster variation is minimized.

$$\sigma_\omega^2 = \sum_{j=1}^{K}\sum_{i=0}^{t}\left(x_i - \mu_j\right)^2$$

- Algorithm (Inputs: K, image pixels or features):
  a) Rearrange the image pixels such that the number of rows in the resulting matrix is equal to the number of pixels and the number of columns is the same as the number of color channels.
  e.g., a 10 by 10 RGB image becomes a matrix of 100 by 3.
  b) Randomly select K centers.
  e.g., in gray-scale images K numbers between 0 and 255



K-Means Clustering Method

- Algorithm (Inputs: K, image pixels or features):
  c) Assign each pixel to the closet cluster (based on proximity to the center).
  d) Update the cluster mean (center).
  e) Repeat step c and d until convergence.

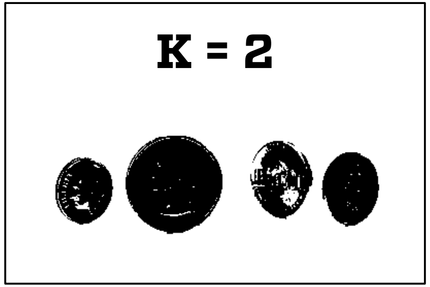
Original Image

Clustered Image with K=3

MATLAB Example - K-Means Clustering

```
I = imread('coin.png');
imshow(I)
X=reshape(I,size(I,1)*size(I,2),size(I,3));
X=double(X);
K=2;
max_iter = 100;
%Clustering
[N, d] = size(X);
L = zeros(N, 1);
C = zeros(K, d); % centers matrix
```

```
for i = 1:max_iter
    % step 1: optimize the labels
    dist = zeros(N,K);
    for j = 1:N
        for k = 1:K
            dist(j,k) = norm(X(j,:)-C(k,:))^2;
        end
    end
    [disto, index] = sort(dist,2);
    L = index(:,1);
   % step 2: optimize the centers
    for k = 1:K
        if sum(L == k)~= 0
            C(k,:) = sum(X(L == k, :),1)/sum(L == k);
        end
    end
end
Y = reshape(L,size(I,1),size(I,2));
BW = Y == 1;
figure, imshow(BW)
```



MATLAB Example using built-in function

```
% input image
I = imread('CS.png');
imshow(I)
X=reshape(I,size(I,1)*size(I,2),size(I,3));

% segmentation with different K values
K=[2 3 4 5]
for i = 1:4
[L,Centers] = kmeans(double(X),K(i));
Y = reshape(L,size(I,1),size(I,2));
B = labeloverlay(I,Y);
subplot (2,2,i);
imshow(B)
end
```

Topics on High-Dimensional Data Analytics

Image Analysis

Kamran Paynabar, Ph.D.

Associate Professor

School of Industrial and Systems Engineering

Edge Detection

Learning Objectives

- How to perform edge detection methods using image derivatives
- Use Sobel edge detection method
- Use Matlab for edge detection

Edge Detection Using Derivatives

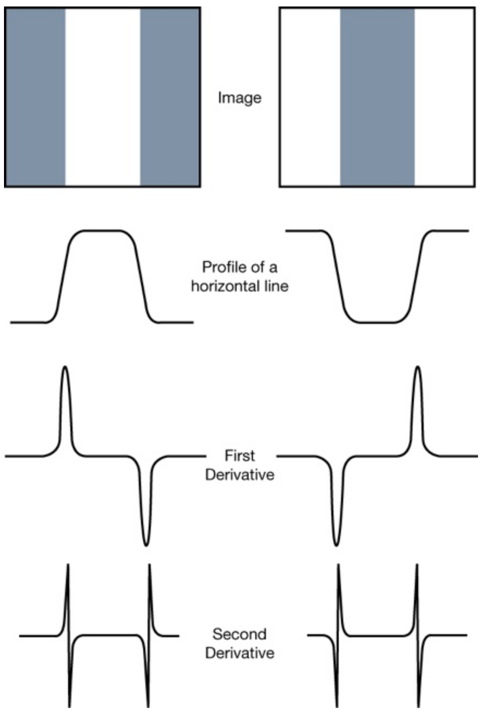
- Edges are significant local changes of intensity in an image.

- Edge Detection: detect pixel with sudden intensity change

- Often, points that lie on an edge are detected by:

  (1) Detecting the local **maxima** or **minima** of the first derivative.

  (2) Detecting the **zero-crossings** of the second derivative.



Approximate Gradient

- Approximate gradient using finite differences:

$$\frac{\partial f}{\partial x} = \lim_{h->0} \frac{f(x+h,y)-f(x,y)}{h} \qquad \frac{\partial f}{\partial y} = \lim_{h->0} \frac{f(x,y+h)-f(x,y)}{h}$$

$$\frac{\partial f}{\partial x} = \frac{f(x+h_x,y)-f(x,y)}{h_y} = f(x+1,y)-f(x,y), \ (h_x=1)$$

$$\frac{\partial f}{\partial y} = \frac{f(x,y+h_y)-f(x,y)}{h_y} = f(x,y+1)-f(x,y), \ (h_y=1)$$

Another Approximation

- Consider the arrangement of pixels around the pixel (*i*, *j*):

3 x 3 neighborhood:
$$\begin{array}{ccc} a_0 & a_1 & a_2 \\ a_7 & [i, j] & a_3 \\ a_6 & a_5 & a_4 \end{array}$$

- The partial derivatives $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial y}$ can be computed by:

$$\frac{\partial f}{\partial x} = (a_2 + ca_3 + a_4) - (a_0 + ca_7 + a_6)$$
$$\frac{\partial f}{\partial y} = (a_6 + ca_5 + a_4) - (a_0 + ca_1 + a_2)$$

- The constant *c* is the weight given to pixels closer to the center of the mask.

Sobel Operator

- Setting $c = 2$, we get the Sobel operator:

$$M_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad M_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

- Sobel kernels can be decomposed as the products of an averaging and a differentiation kernel, they compute the gradient with smoothing.

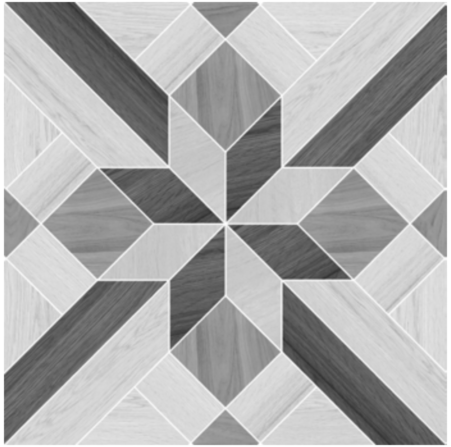
$$M_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$

- Examples of other edge detection operators include Prewitt, Krisch, and Laplacian.

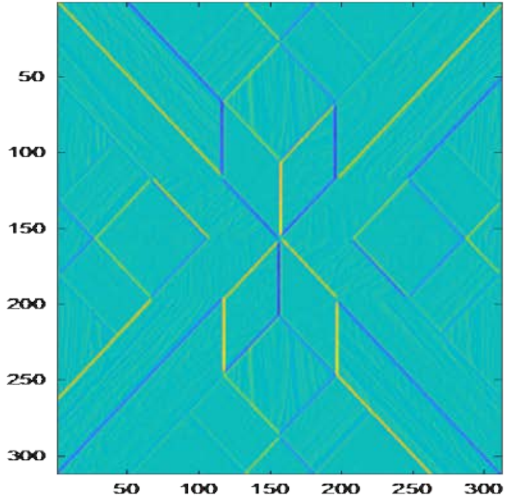
Sobel Operator

- Setting c = 2, we get the Sobel operator:

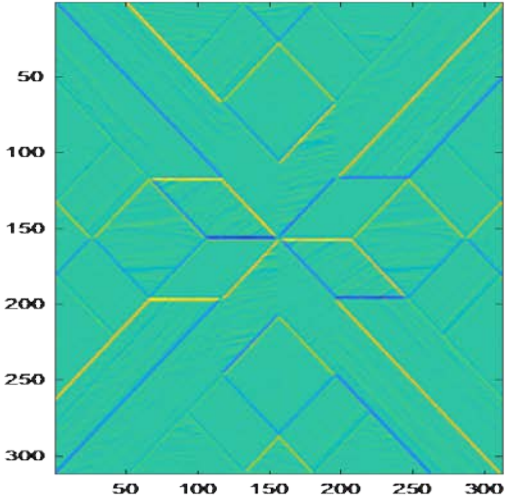
$$M_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad M_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$



f_x



f_y



Sobel Operator

- Setting c = 2, we get the Sobel operator:

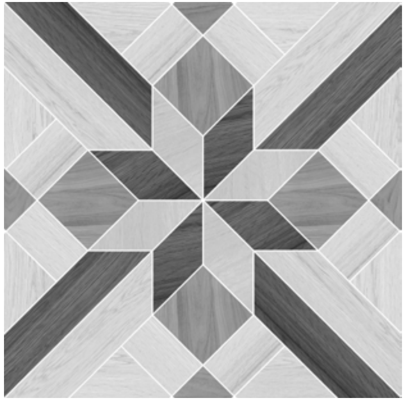
$$M_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad M_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

- Find the convolution of these masks with image to identify edges in horizontal and vertical directions.

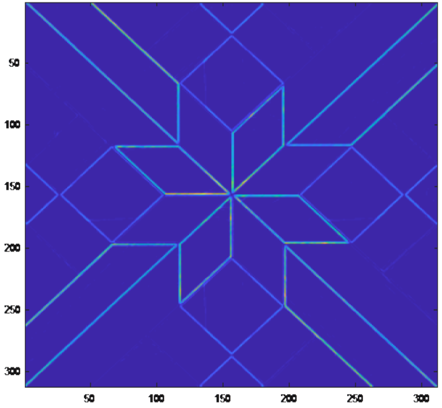
- Get $f = fx.*fx + fy.*fy$

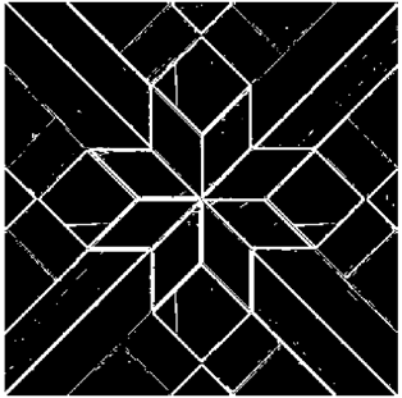
- Determine edge by threshold:

  f = b > cutoff;

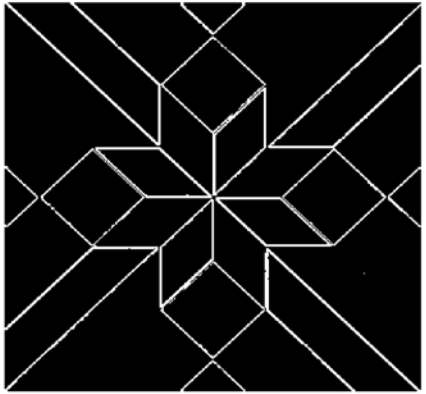


$f = fx.*fx + fy.*fy$



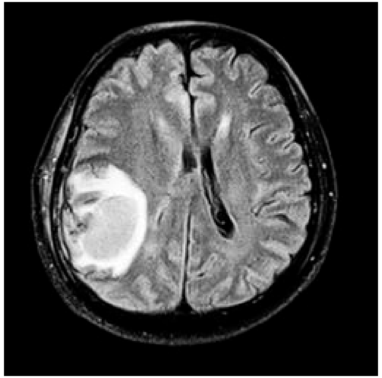


Low threshold

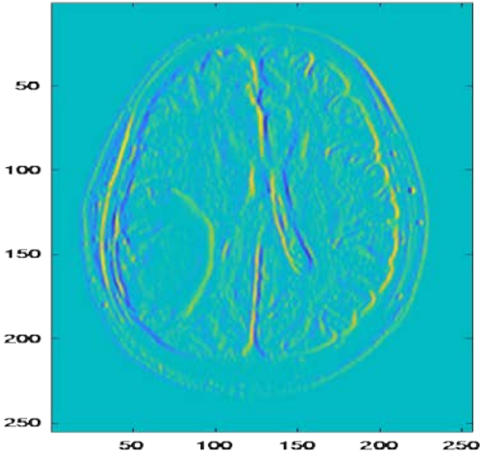


High threshold

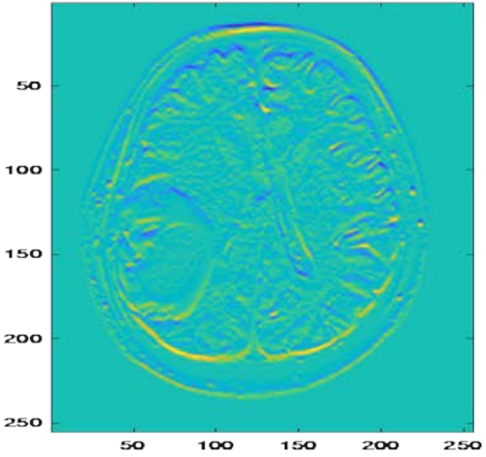
Example in MATLAB



```
a = double(imread('MRI.png'));
op = [1 2 1; 0 0 0;-1 -2 -1]; x_mask = op';  y_mask = op;
fx = imfilter(a,x_mask,'replicate');
fy = imfilter(a,y_mask,'replicate');
subplot(1,2,1);imagesc(fx);subplot(1,2,2);imagesc(fy);
```



fx

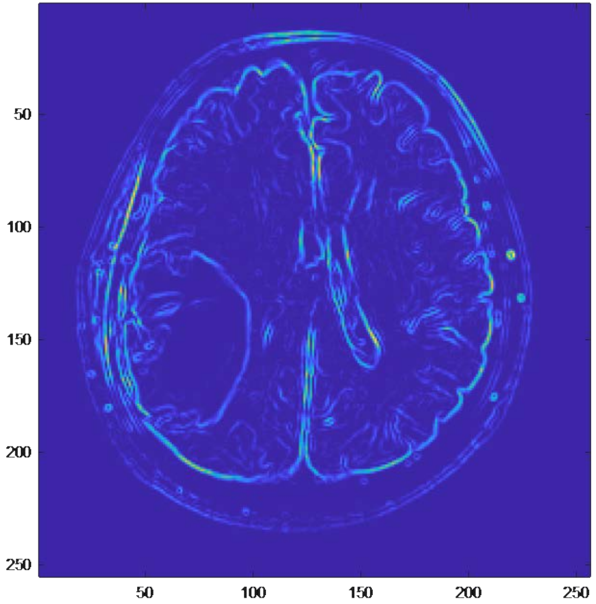


fy

Example in MATLAB

f = fx.*fx + fy.*fy;

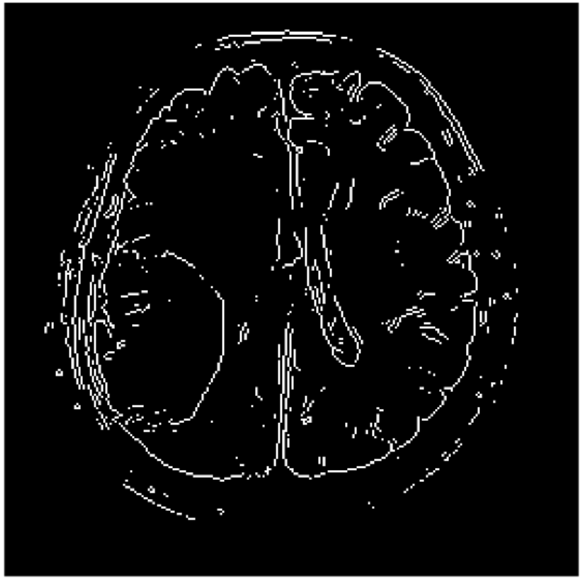
Imagesc(f)



f = b > cutoff;

imshow((fx.*fx+fy.*fy)>cutoff)

edge(a,'sobel')



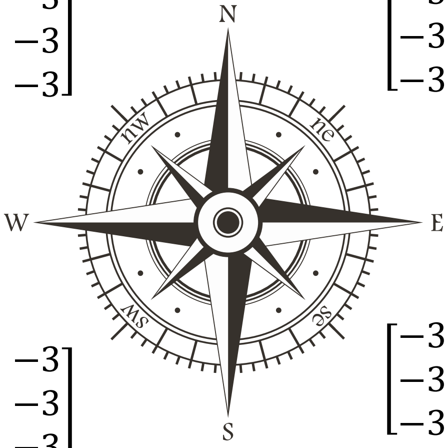
Krisch Operator

- Krisch is a another derivative mask that finds the maximum edge strength in eight directions of a compass.

$$\begin{bmatrix} 5 & 5 & 5 \\ -3 & 0 & -3 \\ -3 & -3 & -3 \end{bmatrix}$$

$$\begin{bmatrix} 5 & 5 & -3 \\ 5 & 0 & -3 \\ -3 & -3 & -3 \end{bmatrix} \quad \begin{bmatrix} -3 & 5 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & -3 \end{bmatrix}$$

$$\begin{bmatrix} 5 & -3 & -3 \\ 5 & 0 & -3 \\ 5 & -3 & -3 \end{bmatrix} \quad \begin{bmatrix} -3 & -3 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & 5 \end{bmatrix}$$



$$\begin{bmatrix} -3 & -3 & -3 \\ 5 & 0 & -3 \\ 5 & 5 & -3 \end{bmatrix} \quad \begin{bmatrix} -3 & -3 & -3 \\ -3 & 0 & 5 \\ -3 & 5 & 5 \end{bmatrix}$$

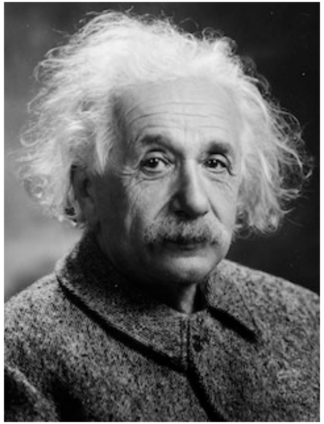
$$\begin{bmatrix} -3 & -3 & -3 \\ -3 & 0 & -3 \\ 5 & 5 & 5 \end{bmatrix}$$

Prewitt Mask

- Prewitt is very similar to Sobel but with different masks.

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

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



edge(a, 'prewitt')

Laplacian and Laplacian of Gaussian Mask

- Laplacian mask is a second order derivative mask.
- For noisy images, is combined with a Gaussian mask to reduce the noise.

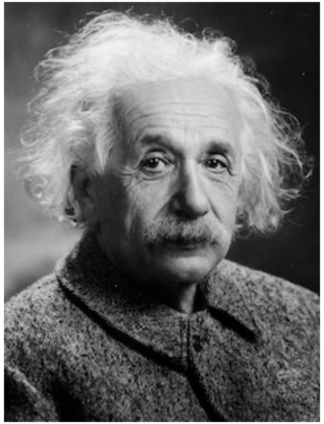
$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Laplacian

0	0	-1	-1	-1	0	0
0	-2	-3	-3	-3	-2	0
-1	-3	5	5	5	-3	-1
-1	-3	5	16	5	-3	-1
-1	-3	5	5	5	-3	-1
0	-2	-3	-3	-3	-2	0
0	0	-1	-1	-1	0	0

Laplacian of Gaussian



edge(a,'log')