Image Processing [3EIB2]

1. Introduction to image processing

Alain Lebret (alain.lebret@ensicaen.fr)

About the course

Two parts

1. Image processing techniques
   - Practical work: 22 h
   - Alain Lebret (alain.lebret@ensicaen.fr)
2. Introduction to deep learning for image processing
   - Lecture: 2h / Practical work: 6h
   - Loïc Simon (loic.simon@ensicaen.fr)

About the course

Prerequisites

- Familiarity with Python (we use JupyterLab for practical works)
- Basic knowledge of linear algebra and Fourier transforms

About the course

Plan of the first part

1. Introduction
2. Colors and color spaces
3. Histograms & point operations
4. Spatial filters
5. Frequency domain filters
6. Detecting simple curves
7. Morphology
8. Regions
9. Geometric operations, comparing images

1. Introduction

What is an image?

We will also introduce 3D images

2-dimensional matrix of Intensity (gray or color) values

Set of Intensity values $I(u,v) \in \mathbb{P}$ and $u, v \in \mathbb{N}$ *Image Coordinates*

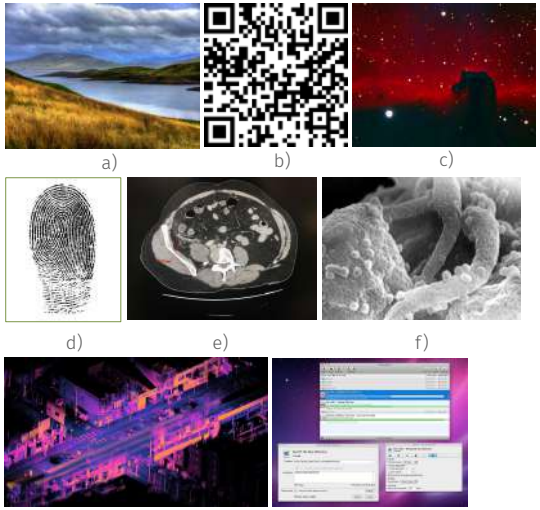
$F(x,y)$

```
104 104 104 103 103 103 103 102 103 103
105 105 102 104 104 103 103 104 104 103
106 106 106 106 106 106 105 105 105 105
106 107 107 106 106 107 108 107 107 107
108 108 109 109 109 109 110 111 110 111
110 109 107 107 108 109 109 109 108 109
107 107 107 106 109 109 109 107 106 107
106 105 103 103 103 104 103 103 103 104
```

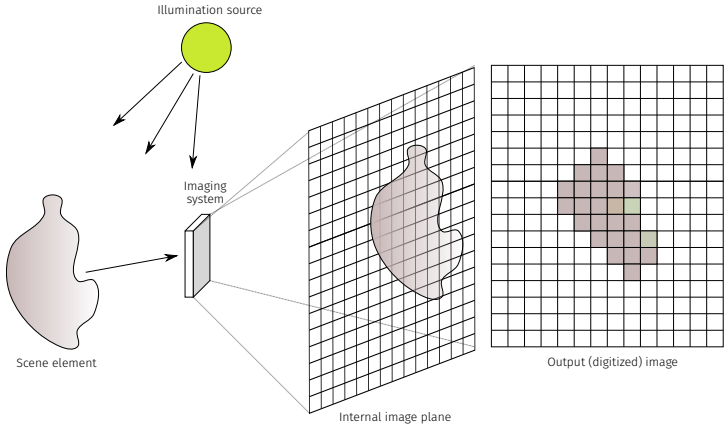
$I(u,v)$

Example of digital images

a) Landscape
b) QR code
c) Astronomical object
d) Fingerprint
e) Scanner image
f) Microscope image
g) Lidar image
h) Computer screenshot



Imaging system



Digital image?

Digitization causes a digital image to become an **approximation** of a real scene

Digital image formats

Common image formats

1 value per point/pixel (B&W or Grayscale)

3 values per point/pixel (Red, Green, and Blue)

4 values per point/pixel (Red, Green, Blue, + “Alpha” or Opacity)

Grayscale

RGB

RGBA

What is image processing?

Algorithms that alter an input image to create a new image

Algorithm to extract edges

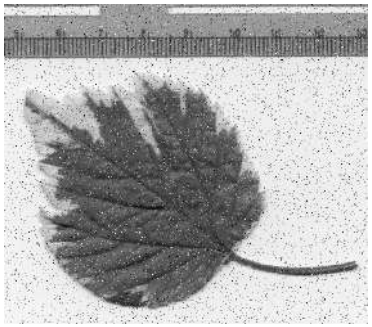
Goals may be to:

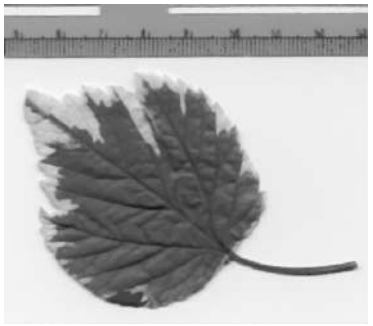
- Edit image
- Display and print image
- Enhance image
- Compress image

Examples: noise removal

Noisy images

Denoised images



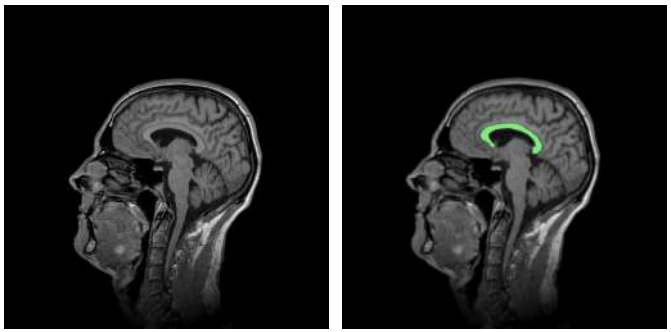


Examples: contrast adjustment

Low contrast | Original image | High contrast

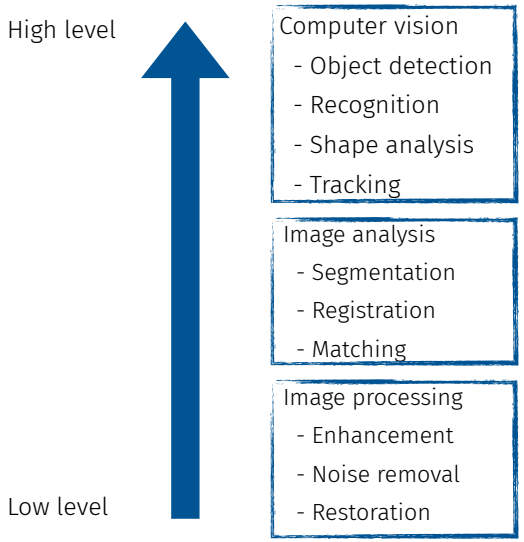
Examples: edge detection

Examples: region extraction and segmentation

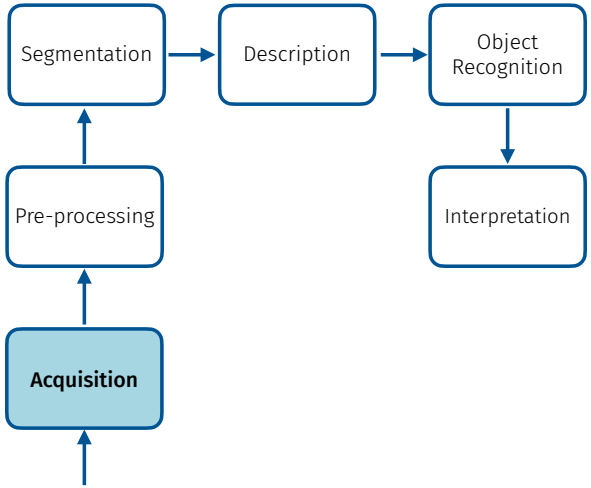


Examples: restoration (in-painting)

Relationship between fields

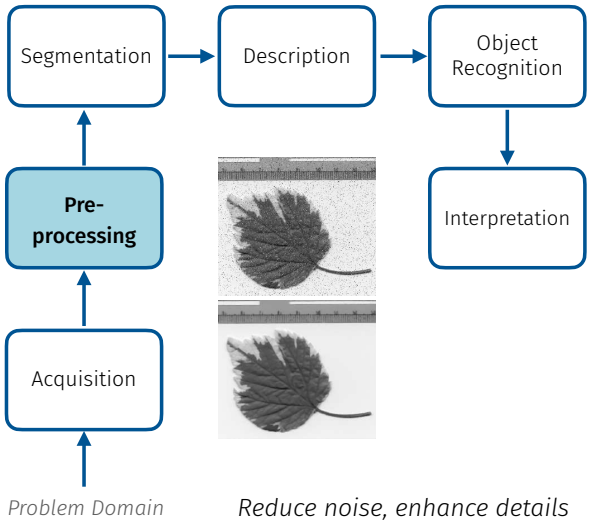


Key stages of a machine vision system



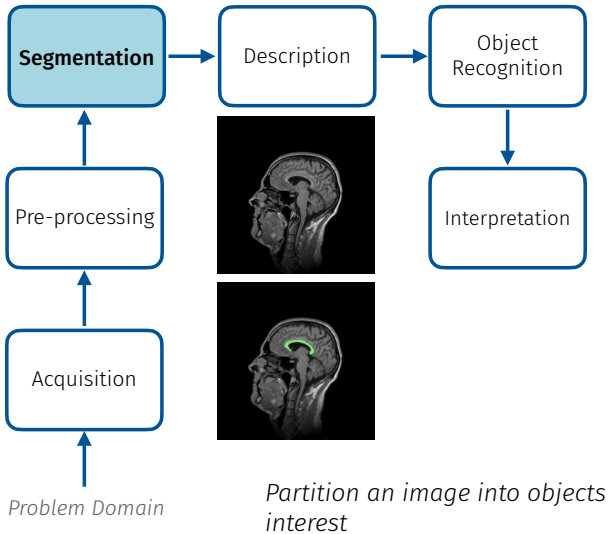
Yield a visual image

Key stages of a machine vision system



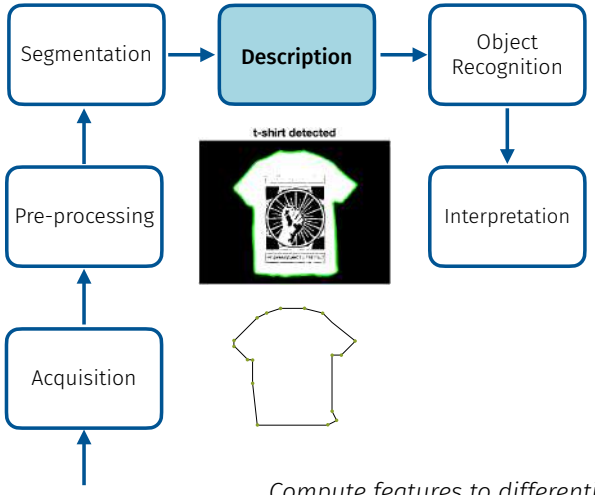
Reduce noise, enhance details

Key stages of a machine vision system



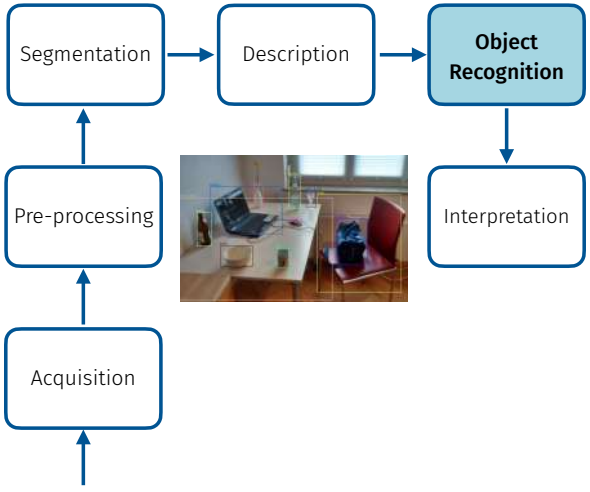
Partition an image into objects of interest

Key stages of a machine vision system



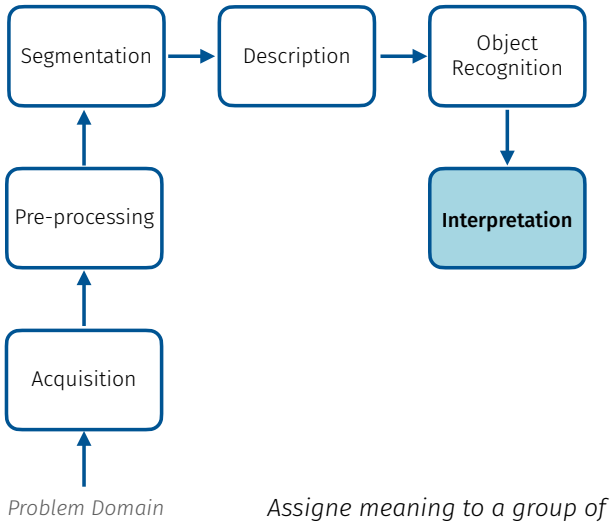
Compute features to differentiate objects

Key stages of a machine vision system



Identify objects

Key stages of a machine vision system



Assigne meaning to a group of objects

Image file formats

Format examples

- Portable Network Graphics (PNG)
- Graphics Interchange Format (GIF)
- Tagged Image File Format (TIFF)
- Portable Bitmap Format (PBM), JPEG, BMP

Pixel values can be

- **Binary**: 0 or 1 (displayed as 0 or 255)
- **Grayscale**: 0-255 range
- **Color**: RGB colors in 0-255 range
- **Application specific** (e.g. floating point values in medicine, astronomy)

How many bits per image element?

Grayscale/intensity image

Chanels	Bits/pixel	Range	Use
1	1	0...1	Binary: document, fax
1	8	0...255	Universal: photo, scan, print
1	12	0...4095	High quality: photo, scan, print
1	14	0...16383	Professional: photo, scan, print
1	16	0...65535	Highest quality: medicine, astronomy

Color image

Chanels	Bits/pixel	Range	Use
3	24	$[0...255]^3$	RGB universal: photo, scan, print
3	36	$[0...4095]^3$	RGB high quality: photo, scan, print
3	42	$[0...16383]^3$	RGB professional: photo, scan, print
3	32	$[0...255]^4$	CMYK, digital prepress

References

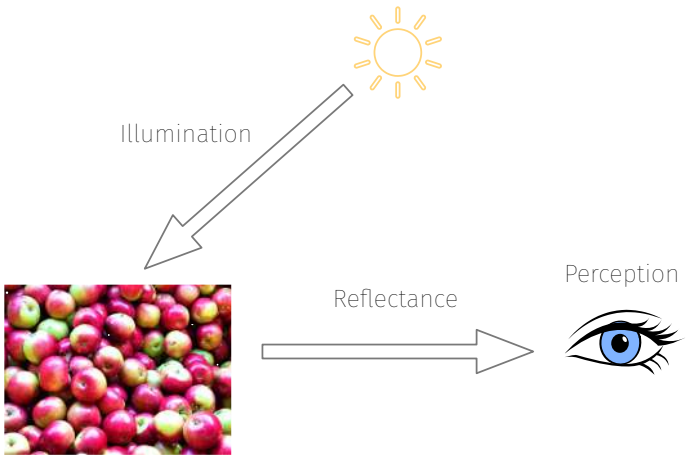
Python and imaging libraries

Libraries used under Jupyter-lab (or Google Colab)

- NumPy: array manipulation library used for its linear algebra, Fourier transform, and random number capabilities
- Pandas: data manipulation and data analysis
- CV2: computer vision tasks
- Scikit-image: image processing
- PIL: another image processing library
- Matplotlib: generates figures and provides a graphical user interface toolkit

2. Colors and color spaces

Basics of color



What is color?

Color is defined in many ways

Physicist's point of view

Electromagnetic spectrum from infrared to ultraviolet, etc.

Physician's point of view

Electrical impulses through optical nerves, excitation of photosensitive molecules in eye, interpretation by brain, etc.

What is color?

Computer scientist defines color as:

- **Hue**: dominant wavelength, color we see

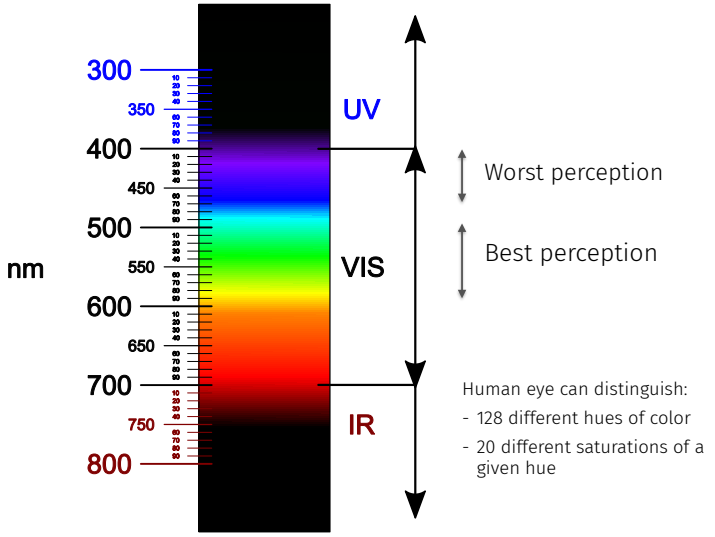
- **Saturation**:
  - How pure the mixture of wavelength is
  - How far is the color from gray

pink is less saturated than red and sky blue is less saturated than royal blue

- **Lightness/brightness**: how intense/bright is the light

- Hue, luminance (in cd/m^2) and saturation are useful for describing colors

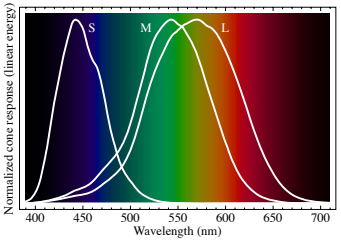
What is color?



Color spaces

Color spaces

Three types of cones in human eyes suggest color is a 3D quantity

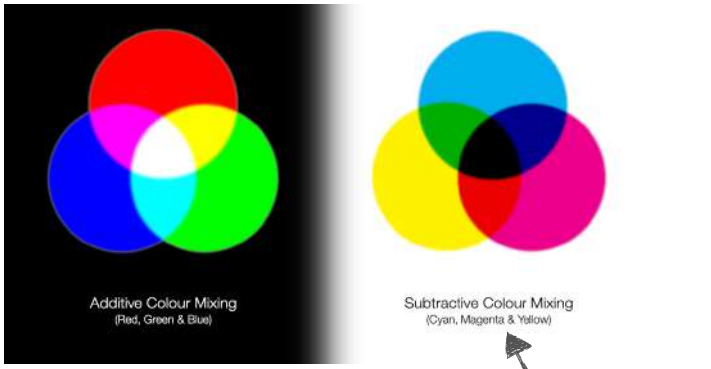


Many different colour spaces

RGB, CMY, HSL, HSV, and more...

Additive vs. subtractive color spaces

Color spaces are additive or subtractive

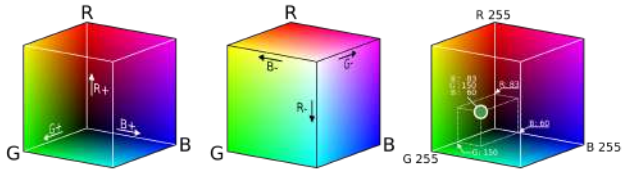


Remove components from white

RGB color space

Define colors with (r, g, b) amounts of red, green, blue

- Most popular additive color space
- Maximum value = 255 or 1.0 if normalized
- (0, 0, 0) = black and (1, 1, 1) = white
- Equal amounts of (r, g, b) = gray (white-black diagonal)

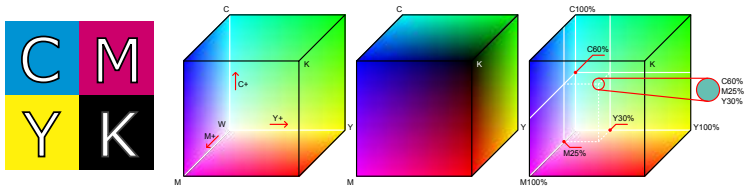


Most operations we will use in grayscale will work on color images by performing operation on each component

CMYK color space

Define colors with (c, m, y) amounts of cyan, magenta, yellow

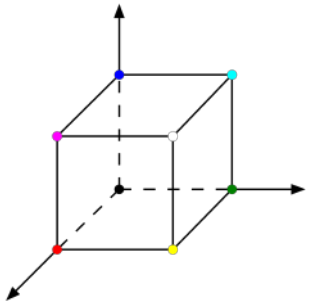
- Subtractive color space is used for printing
- Sometimes black (K) is also used for richer black
- (c, m, y) means subtract the complements of C (red), M (green), and Y (blue)



RGB - CMYK relationship

Interesting to put RGB and CMY in the same cube

- R, G, B and C, M, Y lie at vertices
- Perception of RGB may be non-linear



RGB vs CMYK

RGB CMYK

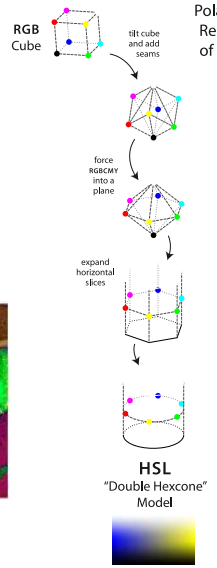
HSL color space

HSL stands for hue, saturation and lightness

- Based on warped RGB cube
- Look from (1, 1, 1) to (0, 0, 0) or RGB cube
  - All hues then lie on the hexagon
  - Hue corresponds to an angle in degrees
  - 0 degree: red

RGB

HSL

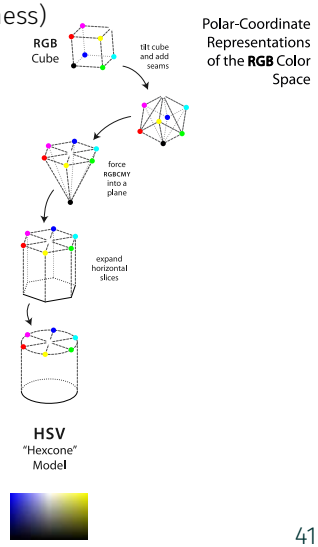


HSV color space

HSV stands for hue, saturation and value (brightness)

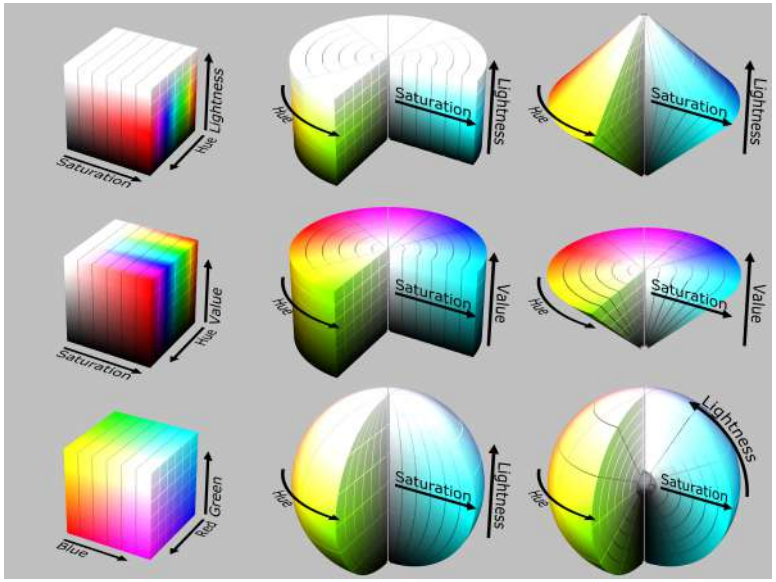
- More intuitive color space
- Based on artist tint, shade, tone
- Similar to HLS in concept

RGB HSV



Polar-Coordinate Representations of the **RGB** Color Space

RGB, HSV and HSL representations



TV color spaces: YUV, YIQ, YCbCr

- YUV and YIQ: color encoding for analog NTSC and PAL
- YC_bC_r: digital TV encoding

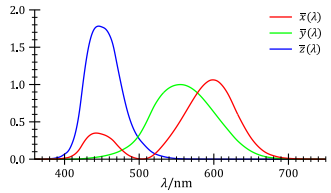
Principle

- Separate luminance channel Y from 2 chroma channels (H, S)
- Instead of encoding colors, encode color differences between channels

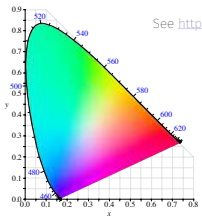
RGB YUV YIQ YCbCr

CIE color space

CIE (Commission Internationale d'Éclairage) has proposed 3 hypothetical lights X, Y, and Z with these spectra:



CIE created table of XYZ values for all visible colors



See https://fr.wikipedia.org/wiki/CIE_XYZ for more information

More about RGB color images

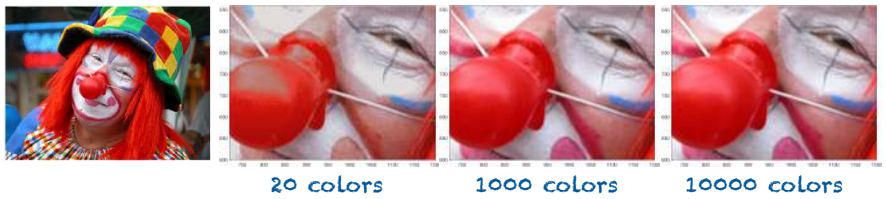
Organization of color images

True color

- Uses all colors in color space
- Used in applications that contain many colors with subtle differences (digital photography, photorealistic rendering, etc.)

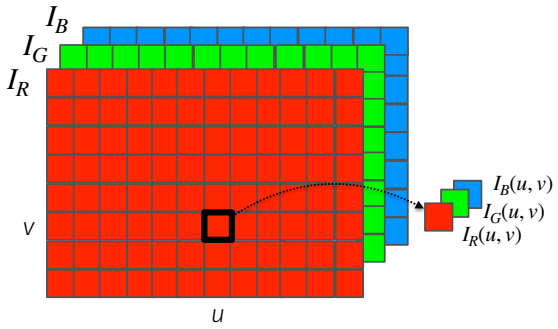
Indexed color

- Uses only some colors
- Subset of colors depends on application



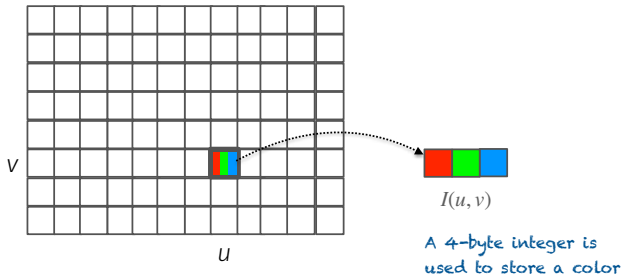
True color: component ordering

- Colors in 3 separate arrays of the same length
- Retrieve the same location (u, v) in each R, G and B array



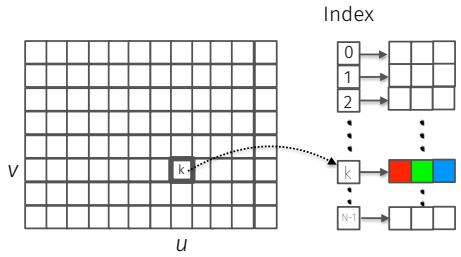
True color: packed ordening

Component (RGB) values representing a pixel's color is packed together into a single element



Indexed images

- Limited number of distinct colors (N = 2 to 256)
- Used in graphics containing large regions with the same color
- Instead of intensity values, image contains indices into color table or palette
- Palette is saved as part of the image
- Converting from true color to indexed color requires quantization



Strategies for processing color images

Strategy 1

Process each RGB channel separately

Strategy 2

Compute luminance and then process intensity matrix

$$Y = w_R R + w_G G + w_B B$$

Converting between color models

Converting color spaces

Converting between color models can be seen as a matrix transform

$$[X' \quad Y' \quad Z'] = [X \quad Y \quad Z] \begin{bmatrix} c_{11} & c_{21} & c_{31} \\ c_{12} & c_{22} & c_{32} \\ c_{13} & c_{23} & c_{33} \end{bmatrix}$$

Converting to grayscale

The simplest way to convert color to grayscale:

Luminance $Y = \frac{R+G+B}{3}$

The resulting image may be too dark in red and green areas

Better approach use a weighted sum of RGB:

$$Y = w_R R + w_G G + w_B B$$

Weights used for digital color encoding:

$$w_R = 0.2125, \quad w_G = 0.7154, \quad w_B = 0.072$$

3 channels

1 channel

Hueless (gray) color images

An RGB image is hueless or gray when all components are equal:

$$R = G = B$$

To remove color from an image:

1. Use previous weighted sum equation to calculate luminance value Y
2. Replace R, G and B with Y value

$$\begin{pmatrix} R' \\ G' \\ B' \end{pmatrix} = \begin{pmatrix} Y \\ Y \\ Y \end{pmatrix}$$

3 channels

3 channels

Desaturating color images

Desaturation is a uniform reduction in the amount of RGB

- Calculate the desaturated color by linearly interpolating RGB color and the corresponding (Y, Y, Y) gray point in RGB space

$$\begin{pmatrix} R_d \\ G_d \\ B_d \end{pmatrix} = \begin{pmatrix} Y \\ Y \\ Y \end{pmatrix} + s_{\text{color}} \cdot \begin{pmatrix} R-Y \\ G-Y \\ B-Y \end{pmatrix}$$

value in range [0, 1]

s_{color}=0.2 s_{color}=0.6

RGB to HSL conversion

Compute hue the same way as for HSV model:

$$H_{HSL} = H_{HSV}$$

Then compute

$$L_{HSL} = \frac{(C_{\text{high}} + C_{\text{low}})/255}{2}$$

$$S_{HSL} = \begin{cases} 0 & \text{for } L_{HSL} = 0 \\ 0.5.\frac{C_{\text{range}}/255}{L_{HSL}} & \text{for } 0 < L_{HSL} \leq 0.5 \\ 0.5.\frac{C_{\text{range}}/255}{1 - L_{HSL}} & \text{for } 0.5 < L_{HSL} < 1 \\ 0 & \text{for } L_{HSL} = 1 \end{cases}$$

RGB to HSL conversion


Original RGB


HSL


H


S


L

HSL to RGB conversion

Assuming H, L and S in range [0, 1]

$$(R', G', B') = \begin{cases} (0,0,0) & \text{for } L_{HSL} = 0 \\ (1,1,1) & \text{for } L_{HSL} = 1 \end{cases}$$

Otherwise calculate

$$H' = (6H_{HSL}) \bmod 6$$

Then compute the following values:

$$c_1 = H'$$
$$c_2 = H' - c_1$$
$$x = L_{HSL} - d$$
$$w = L_{HSL} + d$$
$$y = w - (w - x) \cdot c_2$$
$$z = w + (w - x) \cdot c_2$$

$$d = \begin{cases} S_{HSL} \cdot L_{HSL} & \text{for } L_{HSL} \leq 0.5 \\ S_{HSL} \cdot (1 - L_{HSL}) & \text{for } L_{HSL} > 0.5 \end{cases}$$

HSL to RGB conversion

Then, the assignment of RGB values is done as:

$$(R', G', B') = \begin{cases} (w, z, x) & \text{if } c_1 = 0 \\ (y, w, x) & \text{if } c_1 = 1 \\ (x, w, z) & \text{if } c_1 = 2 \\ (x, y, w) & \text{if } c_1 = 3 \\ (z, x, w) & \text{if } c_1 = 4 \\ (w, x, y) & \text{if } c_1 = 5. \end{cases}$$

RGB to HSV conversion

Define the following values:

$$C_{\text{high}} = \max(R, G, B)$$
$$C_{\text{low}} = \min(R, G, B)$$
$$C_{\text{range}} = C_{\text{high}} - C_{\text{low}}$$

Find saturation of RGB color components ($C_{\text{max}} = 255$):

$$S_{HSV} = \begin{cases} \dfrac{c_{\text{range}}}{c_{\text{high}}} & \text{for } C_{\text{high}} > 0 \\ 0 & \text{otherwise} \end{cases}$$

And luminance value:

$$V_{HSV} = \frac{C_{\text{high}}}{C_{\text{max}}}$$

RGB to HSV conversion

Normalize each component using:

$$R' = \frac{C_{\text{high}} - R}{C_{\text{range}}} \qquad G' = \frac{C_{\text{high}} - G}{C_{\text{range}}} \qquad B' = \frac{C_{\text{high}} - B}{C_{\text{range}}}$$

Calculate preliminary hue value H' as:

$$H' = \begin{cases} B' - G' & \text{if } R = C_{\text{high}} \\ R' - B' + 2 & \text{if } G = C_{\text{high}} \\ G' - R' + 4 & \text{if } B = C_{\text{high}} \end{cases}$$

Finally hue value is obtained by normalizing to range [0, 1]:

$$H_{HSV} = \frac{1}{6} \cdot \begin{cases} H' + 6 & \text{for } H' < 0 \\ H' & \text{otherwise} \end{cases}$$

RGB to HSV conversion

Original RGB

HSV

H

S

V

HSV to RGB conversion

First, calculate:

$$H' = (6H_{HSV}) \bmod 6$$

Then compute the following intermediate values:

$$c_1 = H' \qquad c_2 = H' - c_1$$

$$x = (1 - S_{HSV}) \cdot V_{HSV}$$
$$y = (1 - (S_{HSV} \cdot c_2)) \cdot V_{HSV}$$
$$z = (1 - (S_{HSV} \cdot (1 - c_2))) \cdot V_{HSV}$$

Normalized RGB values are calculated as (range [0, 1]):

$$(R', G', B') = \begin{cases} (V_{HSV}, z, x) & \text{if } c_1 = 0 \\ (y, V_{HSV}, x) & \text{if } c_1 = 1 \\ (x, V_{HSV}, z) & \text{if } c_1 = 2 \\ (x, y, V_{HSV}) & \text{if } c_1 = 3 \\ (z, x, V_{HSV}) & \text{if } c_1 = 4 \\ (V_{HSV}, x, y) & \text{if } c_1 = 5. \end{cases}$$

HSV to RGB conversion

RGB Components can be scaled to whole numbers in range [0, 255] as:

$$R = min(round(255.R'), 255),$$
$$G = min(round(255.G'), 255),$$
$$B = min(round(255.B'), 255),$$

YUV color space

Basis for color encoding in analog TV in North America (NTSC) and Europe (PAL)

- Y channel computed from RGB channels as:

$$Y = 0.299.R + 0.587.G + 0.114.B$$

- UV channels from RGB channels as:

$$U = 0.492.(B - Y) \qquad V = 0.877.(R - Y)$$

YUV color space

Original RGB

YUV

Y

U

V

YUV color space

Transformation from RGB to YUV

$$\begin{pmatrix} Y \\ U \\ V \end{pmatrix} = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ -0.147 & -0.289 & 0.436 \\ 0.615 & -0.515 & -0.100 \end{pmatrix} \cdot \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

Transformation from YUV to RGB

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1.140 \\ 1 & -0.395 & -0.581 \\ 1 & 2.032 & 0 \end{pmatrix} \cdot \begin{pmatrix} Y \\ U \\ V \end{pmatrix}$$

YIQ color space

Original NTSC used variant of YUV is called **YIQ**

- Y channel is the same as in YUV
- Both U and V color vectors rotated and mirrored so that:

$$\begin{pmatrix} I \\ Q \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} cos(\beta) & sin(\beta) \\ -sin(\beta) & cos(\beta) \end{pmatrix} \cdot \begin{pmatrix} U \\ V \end{pmatrix}$$

$\beta = 0.576$ (33 degrees)

YIQ color space


Original RGB


YIQ


Y


I


Q

YCbCr color space

Internationally standardized variant of YUV
- Used for digital TV and image compression (e.g. JPEG)

Y, C_b, C_r components are calculated as:

$$Y = w_R \,.\, R + (1 - w_B - w_R) \,.\, G + w_B \,.\, B$$

$$C_b = \frac{0.5}{1 - w_B} \,.\, (B - Y)$$

$$C_r = \frac{0.5}{1 - w_R} \,.\, (R - Y)$$

ITU preconizes

$$w_R = 0.299, \quad w_B = 0.114, \quad w_G = 1 - w_B - w_R = 0.587$$

YCbCr color space


Original RGB


YCbCr


Y


Cb


Cr

YCbCr color space

Inverse transform from YCbCr to RGB

$$R = Y + \frac{1 - w_R}{0.5}$$

$$G = Y - \frac{w_B \,.\, (1 - w_B) \,.\, C_b - w_R \,.\, (1 - w_R) \,.\, C_r}{0.5.(1 - w_B - w_R)}$$

$$B = Y + \frac{1 - w_B}{0.5} \,.\, C_b$$

YCbCr color space

Transformation from RGB to YCbCr

$$\begin{pmatrix} Y \\ C_b \\ C_r \end{pmatrix} = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.331 & 0.5 \\ 0.5 & -0.419 & -0.081 \end{pmatrix} \cdot \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

Transformation from YCbCr to RGB

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1.403 \\ 1 & -0.344 & -0.714 \\ 1 & 1.773 & 0 \end{pmatrix} \cdot \begin{pmatrix} Y \\ C_b \\ C_r \end{pmatrix}$$

RGB, HSV, HSL, YUV, YIQ and YCbCr comparison

Colors, color spaces and Python

Scikit-image

The Python library scikit-image supports 2 types of color images

- RGB full-color images (24-bit "RGB color")
  - packed order
  - Supports TIFF, BMP, JPEG, PNG and RAW file formats
- Indexed images ("8-bit color")
  - Up to 256 colors max (8 bits)
  - Supports GIF, PNG, BMP and TIFF (uncompressed) file formats

io and **color** modules in scikit-image contain methods for converting between different types of color and grayscale images

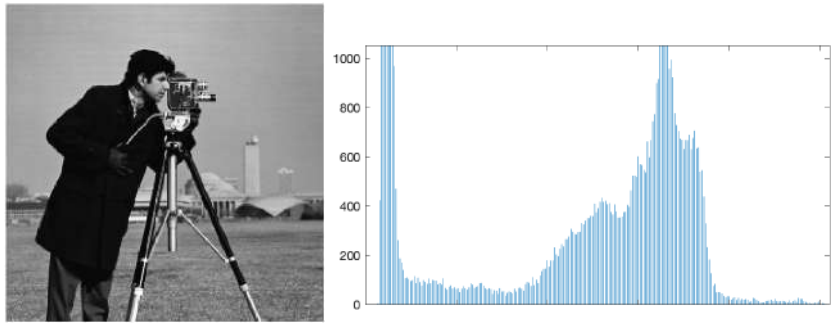
Note: **cv2** library (OpenCV) has also useful color functions

3. Histograms and point operations

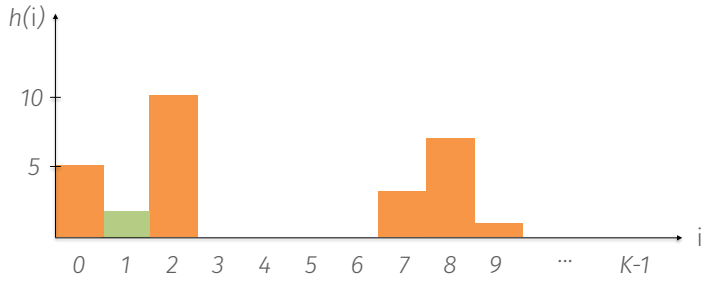
Histograms

Histogram

An histogram plots **how many times** each intensity value occurs



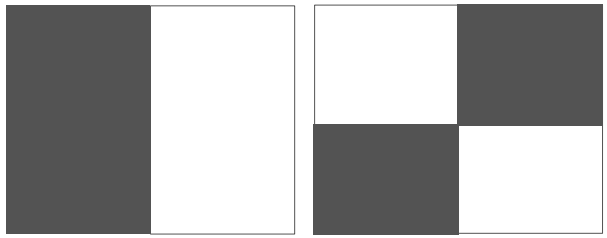
Histogram



h(i)	5	2	10	0	0	0	0	3	7	1	...	...	...
i	0	1	2	3	4	5	6	7	8	9	...		K-1

Histogram

Different images can have same histograms



Histograms

Some problems within an image can be identified on its histogram:

- Over and under exposure
- Contrast
- Dynamic range

Brightness

Brightness of a grayscale image is the **average intensity** of all pixels in it

$$B(I) = \frac{1}{wh} \sum_{v=1}^{h} \sum_{u=1}^{w} I(u, v)$$

Number of pixels

Sum of all pixel intensities

Detecting bad exposure using histograms

underexposed | properly exposed | overexposed

Contrast

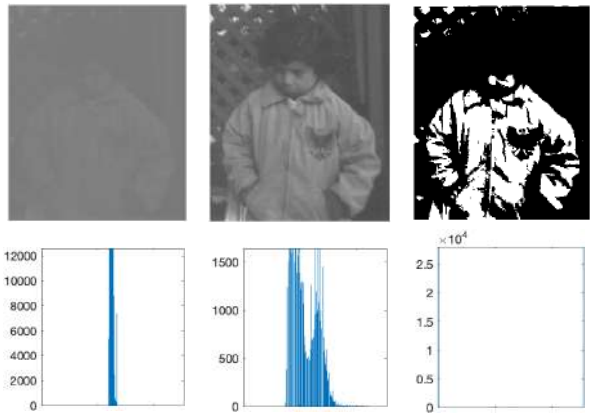
The contrast of a grayscale image indicates how easily objects within the image can be distinguished

High contrast: many distinct intensity values

Low contrast: few intensity values

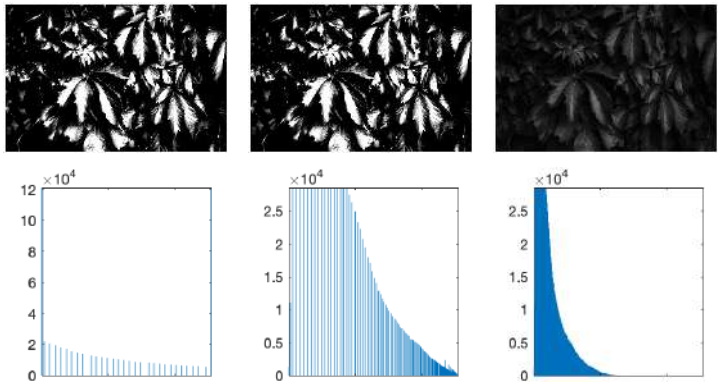
A good contrast has widely spread intensity values and large difference between min and max intensity values

Histograms and contrast



very low contrast normal contrast too high contrast

Histograms and dynamic range



very low dynamic range low dynamic range high dynamic range

Histograms and dynamic range

High dynamic range means very bright and very dark parts in an image (many distinct values)

The dynamic range may exceed number of available bits to represent pixels

Solution:

1) Capture multiple images at different exposures

2) Combine them using image processing

Large histograms: binning

An high resolution image can yield a very large histogram

Example: 32-bit image = 2^{32} = 4,294,967,296 columns

A large histogram is difficult to display

Solution:

Combine ranges of intensity values into histogram columns

How to calculate bin size?

$$\text{bin size} = \frac{\text{number of distinct values in image}}{\text{number of bins}}$$

Example: create 256 bins from 14-bit image

$$\text{bin size} = \frac{2^{14}}{256} = 2^{(14-8)} = 2^6 = 64$$

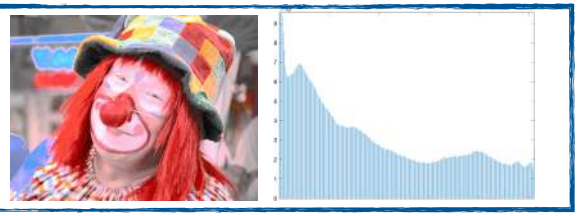
$$\begin{aligned} h(0) &\leftarrow \quad 0 \le I(u,v) < 64 \\ h(1) &\leftarrow \quad 64 \le I(u,v) < 128 \\ &\vdots \qquad\qquad \vdots \\ h(255) &\leftarrow 16320 \le I(u,v) < 16384 \end{aligned}$$

Color image histogram

1

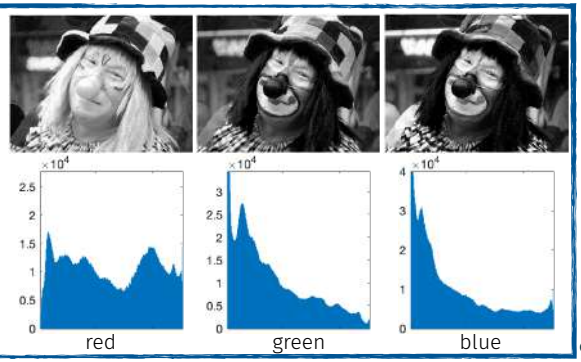
Intensity histogram (*)

(*) image is first convert to gray



2

Individual color component histograms



Point operations

Point operations

Change the intensity value of a pixel according to some function

$$I'(u,v) = f(I(u,v))$$

Examples:

- Addition (changes brightness)
- Multiplication (stretches or shrinks image contrast range)
- Real-valued functions ($exp(x), log(x), x^k, (1/x)$, etc.)
- Gamma correction
- Global thresholding

Necessity to clamp all values to fall within the original range

Point operations: common transformations

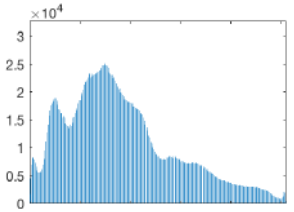
Basic gray level transformations

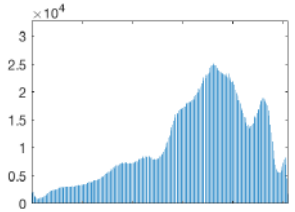
Common gray level transformations

- Linear (identity/negative)
- Logarithmic (Log / Inverse Log)
- Power (n^{th} power / n^{th} root)

Inverting images (negatives)

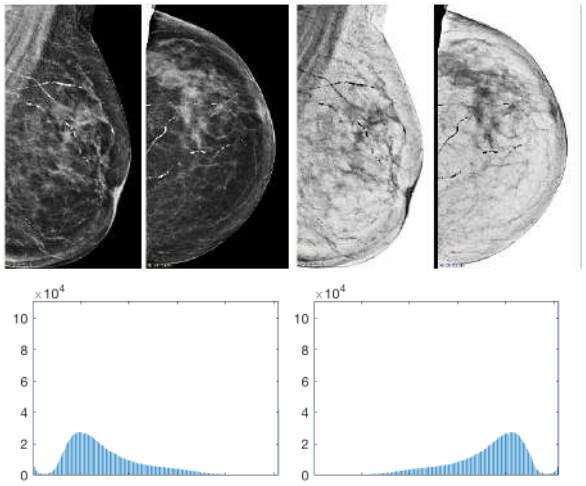
$$f_{invert}(I) = -I + Cste$$





Inverting images (negatives)

Enhancement of white or gray details in dark regions of an image



Logarithmic transformations

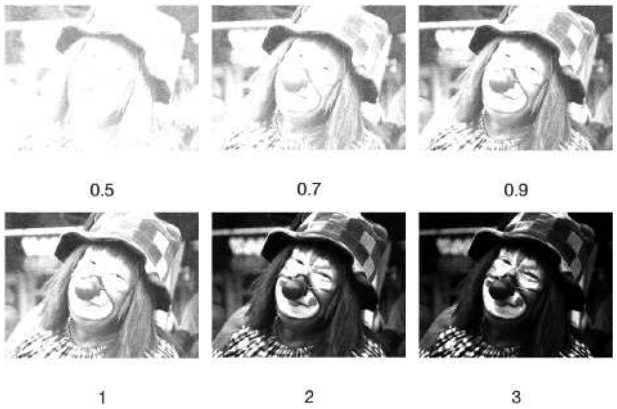
Transforms narrow range input values to wider range output values

$$I'(u,v) = C \times \log(1 + I(u,v))$$

Power law transformations

Map narrow range of dark input values into wider range of output values or vice versa

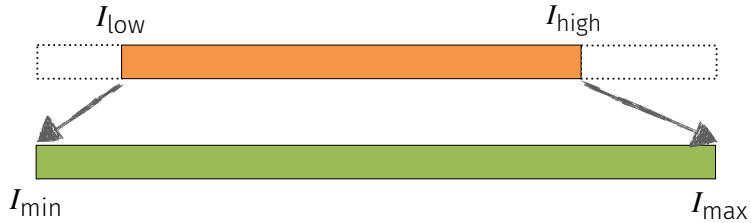
$$I'(u,v) = Cste \times I^{\gamma}(u,v)$$



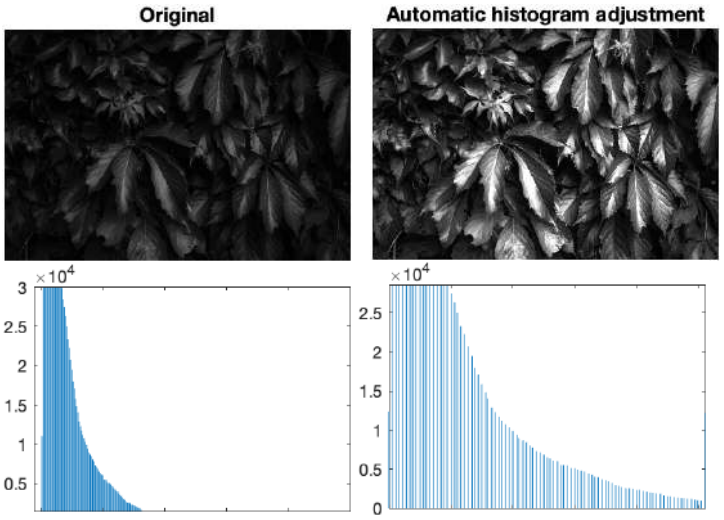
Automatic contrast adjustment

Modifies intensities such that available range of values is fully covered

$$f_{aca}(I) = I_{\min} + (I - I_{\text{low}}) \times \frac{I_{\max} - I_{\min}}{I_{\text{high}} - I_{\text{low}}}$$



Automatic contrast adjustment



Point operations: global thresholding

Thresholding

$$f_{threshold}(I) = \begin{cases} I_0 & \text{for } I < I_{th} \\ I_1 & \text{for } I \geq I_{th} \end{cases}$$

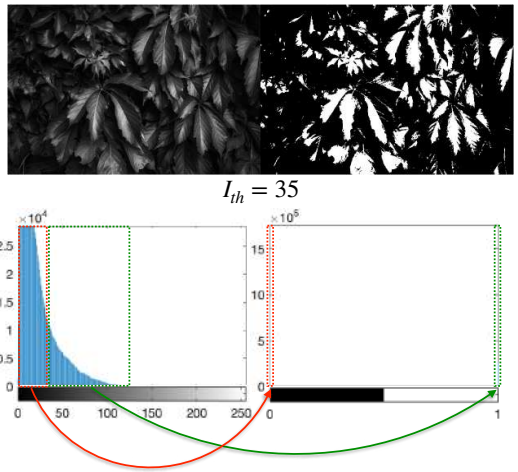
E.g. : Converts grayscale to binary image if:

- $I_0 = 0$
- $I_1 = 1$

Thresholding: example

$I_{th} = 35$

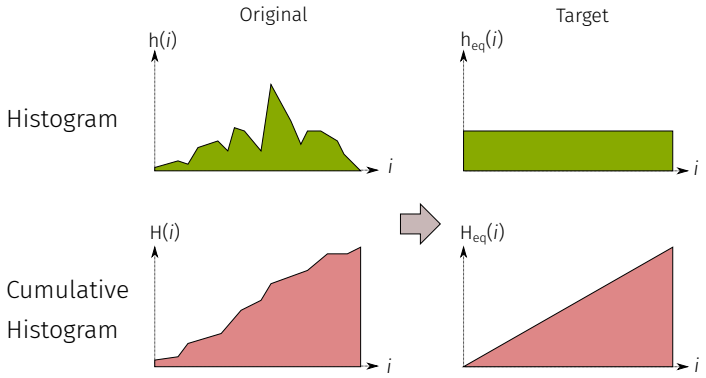
Thresholding and histograms



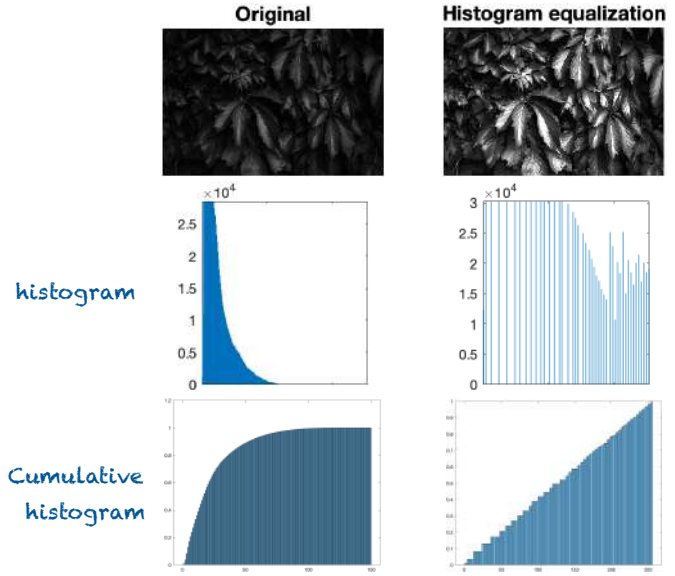
Point operations: histogram equalization

Histogram equalization

- Adjust 2 different images to make their histograms similar
- Changes histogram of modified image into uniform distribution



Histogram equalization



Histogram equalization

Histogram function

```
def get_histogram(image, bins):
    # array with size of bins, set to zeros
    histogram = np.zeros(bins)

    # loop through pixels and sum up counts
    # of pixels
    for pixel in image:
        histogram[pixel] += 1

    # return our final result
    return histogram
```

Cumulative sum function

```
def cumsum(a):
    a = iter(a)
    b = [next(a)]
    for i in a:
        b.append(b[-1] + i)
    return np.array(b)
```

```
# execute histogram function
hist = get_histogram(img, 256)

# execute the cumulative sum fn
cs = cumsum(hist)

# display the result
plt.plot(cs)
```

Normalization

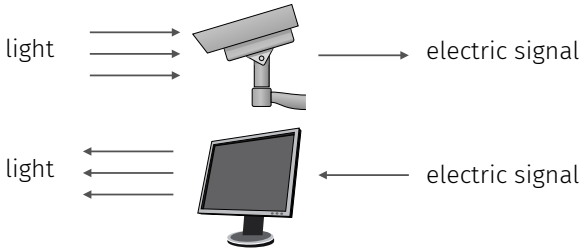
```
# numerator & denominator
nj = (cs - cs.min()) * 255
N = cs.max() - cs.min()

# re-normalize the cumsum
cs = nj / N

# cast it back to uint8 since we can't use
# floating point values in images
cs = cs.astype('uint8')

plt.plot(cs)
```


Histogram equalization

```
# get the value from cumulative sum for
# every index, and set that as img_new
img_new = cs[img]

# put array back into original shape since
# we flattened it
img_new = np.reshape(img_new, img.shape)

# set up side-by-side image display
fig = plt.figure()
fig.set_figheight(15)
fig.set_figwidth(15)
fig.add_subplot(1,2,1)
plt.imshow(img, cmap='gray')

# display the new image
fig.add_subplot(1,2,2)
plt.imshow(img_new, cmap='gray')
plt.show(block=True)
```


Point operations: gamma correction

Gamma correction



Different camera sensors

- Have different responses to light intensity
- Produce different electrical signals for same input

How do we ensure there is consistency in:

- Images recorded by different cameras for given light input?
- Light emitted by different display devices for same image?

Gamma correction

What is the relation between:

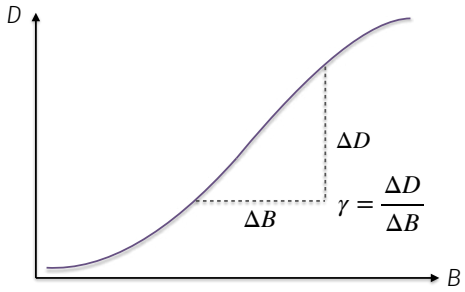
- Camera: light on sensor vs. "intensity" of corresponding pixel
- Display: pixel intensity vs. light from that pixel

Relation between pixel value and corresponding physical quantity is usually complex and nonlinear

Gamma correction

Exposure function: relationship between logarithmic light intensity (B) vs. resulting film density (D)

Gamma: slope of linear range of the curve



Gamma correction

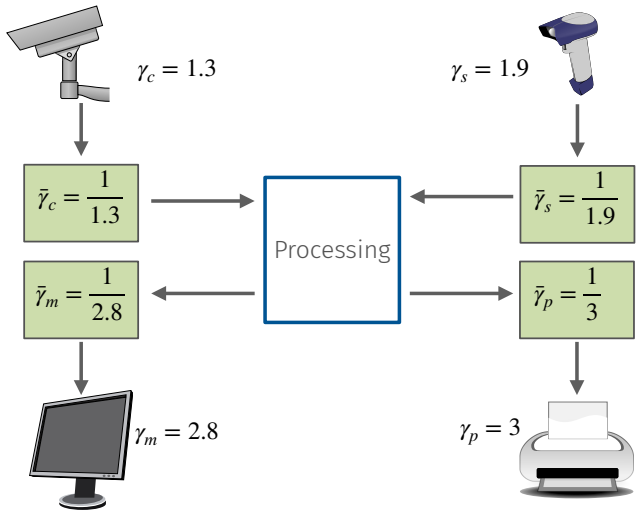
Gamma function: a good approximation of exposure curve

- Inverse of a Gamma function is another gamma function with

$$\bar{\gamma} = \frac{1}{\gamma}$$

- Gamma of CRT and LCD monitors: range 1.8 - 2.8 (typically 2.4)

Gamma correction



Gamma correction in Python

```
def load_image(filename, width, gamma):
   img = imageio.imread(filename)

   if img.shape[-1] == 4:
     # Blend the alpha channel
     img = color.rgba2rgb(img)

   # Convert to grayscale
   img = color.rgb2gray(img)

   # Resample and adjust the aspect ratio
   width_px = (3 * width) * 16

   img_width = 1.0 * width_px
   img_height = int(img.shape[0] * 3 * (img_width / (4 * img.shape[1])))
   img = transform.resize(img, (img_height, img_width), anti_aliasing=True,
   mode='constant')

   # Adjust the exposure
   img = exposure.adjust_gamma(img, gamma)

   return img
```


Histogram and point operations using Python

Histogram and point operations in Python

exposure module from scikit-image supports gamma correction and histogram manipulation

What is a filter?

Capabilities of point operations are limited

Filters combine pixel's value + values of neighbors

4. Spatial filters

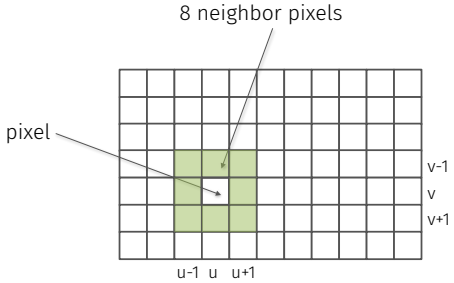
Spatial filter

Definition

Operation that combines each pixel intensity $I(u, v)$ with that of its neighborhood

E.g. smoothing computes **average** or **weighted average** of group of pixels

Example: smoothing by averaging with 3x3 neighborhood



Smoothing replaces each pixel with average intensity of itself + neighbors

Smoothing by averaging

Many possible filter parameters (size, weights, function, etc):

- Size of neighborhood: 3x3, 5x5, 7x7, ..., 21x21, ...
- Shape: square, rectangle, circle, etc.
- Weights: may apply unequal weighting to different pixels
- Function: can be linear (a weighted summation) or nonlinear

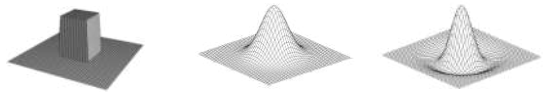
Example using different mask sizes, with square shape, equal weighting and weighted summation

Original 3x3 41x41 83x83

Linear filters

2 main classes of linear filters:

- **Smoothing filters**: positive weights (e.g. box, gaussian)
- **Difference filters**: positive and negative weights (e.g. Laplacian)



0	0	0	0	0
0	1	1	1	0
0	1	1	1	0
0	1	1	1	0
0	0	0	0	0

Box

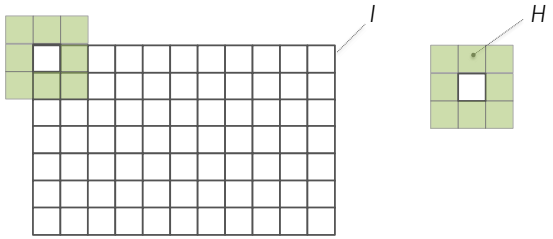
0	1	2	1	0
1	3	5	3	1
2	5	9	5	2
1	3	5	3	1
0	1	2	1	0

Gaussian

0	0	-1	0	0
0	-1	-2	-1	0
-1	-2	16	-2	-1
0	-1	-2	-1	0
0	0	-1	0	0

Laplacian

Apply linear filters using convolution



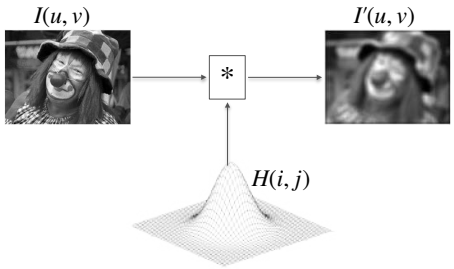
$$I'(u,v) = \sum_{i=-1}^{i=1} \sum_{j=-1}^{j=1} I(u+i, v+j) \,.\, H(i,j)$$

Mathematical properties of convolution

- Applying a filter as described is called **linear convolution**
- For discrete 2D signal, convolution is defined as:

$$I'(u,v) = \sum_{i=-\infty}^{+\infty} \sum_{j=-\infty}^{+\infty} I(u+i, v+j) \,.\, H(i,j)$$

$$I' = I * H$$



Mathematical properties of convolution

- Commutativity

$$I * H = H * I$$

Same result if we convolve image with filter or vice versa

- Associativity

$$A * (B * C) = (A * B) * C$$

Any order of filters give same result

- Linearity

$$(s \,.\, I) * H = I * (s \,.\, H) = s \,.\, (I * H)$$

Same result for an image multiplied with a scalar, or the convolved one

$$(I_1 + I_2) * H = (I_1 * H) + (I_2 * H)$$

Same result if 2 images are added and the result is convolved with a kernel H, or if each image is convolved individually + added

- Separability

$$H = H_1 * H_2 * \cdots * H_n$$

Small kernels are computationally cheaper

$$I * H = I * (H_1 * H_2 * \cdots * H_n) = (\cdots((I * H_1) * H_2) * \cdots * H_n)$$

Separability in x and y

A kernel may sometimes be separated into its "vertical" and "horizontal" components

Let $H_x = [1 \quad 1 \quad 1 \quad 1 \quad 1]$, and $H_y = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$

Then $H = H_x * Hy = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$

Complexity : 15 x width x height

Complexity : (3 + 5) x width x height

Gaussian kernel

2D gaussian is product of 1D gaussians

$$G_\sigma(x,y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$$

$$= \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{x^2}{2\sigma^2}\right) . \exp\left(-\frac{y^2}{2\sigma^2}\right)$$

$$= g_\sigma(x) \, . \, g_\sigma(y)$$

→ Convolution with a gaussian is separable

Impulse function (Dirac) kernel

Impulse function has: white pixel in (0, 0) on a black background

$$I * \delta = I$$

Reverse case:

$$H * \delta = \delta * H = H$$

Noise

Types of noise

Salt and pepper noise: randomly scattered black + white pixels

- Also called impulse noise, shot noise or binary noise
- Caused by sudden sharp disturbance

Original

Salt and pepper noise

Types of noise

Gaussian noise: idealized form of white noise added to image, normally distributed

$$I = I + \text{noise}$$

Original

Gaussian noise

Types of noise

Speckle Noise: pixel values multiplied by random noise

$$I = I . (1 + \text{noise})$$

Original

Speckle noise

Types of noise

Periodic noise is caused by disturbances of a periodic nature

Types of noise

Type of noise determines best types of filters for removing it!!

Salt and pepper, gaussian and speckle noises can be cleaned using spatial filters

Periodic noise can be cleaned using frequency domain filtering

Non-linear filters

Non-linear filters

Linear filters blurs all image structures such as edges and lines

→ They reduce image quality

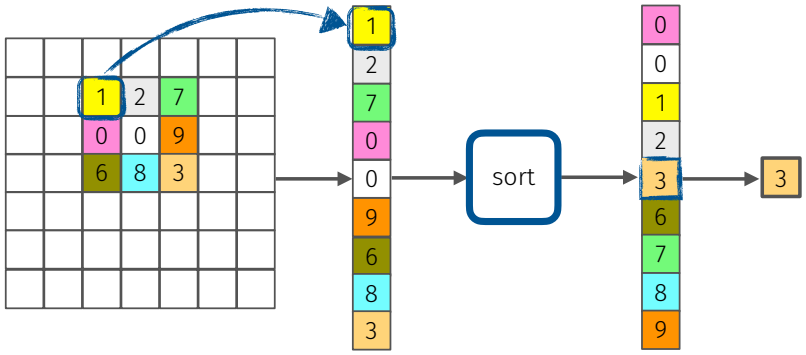
→ They should not be used a lot to remove noise

Linear filter (15x15 kernel)

Median filter

Median filter removes noise and keep structures

$$I'(u,v) = median\{\, I(u+i, v+j) \mid (i,j) \in \mathbb{R} \}$$



Effects of median filters

Linear filter (15x15 kernel)

Median filter

Weighted median filter

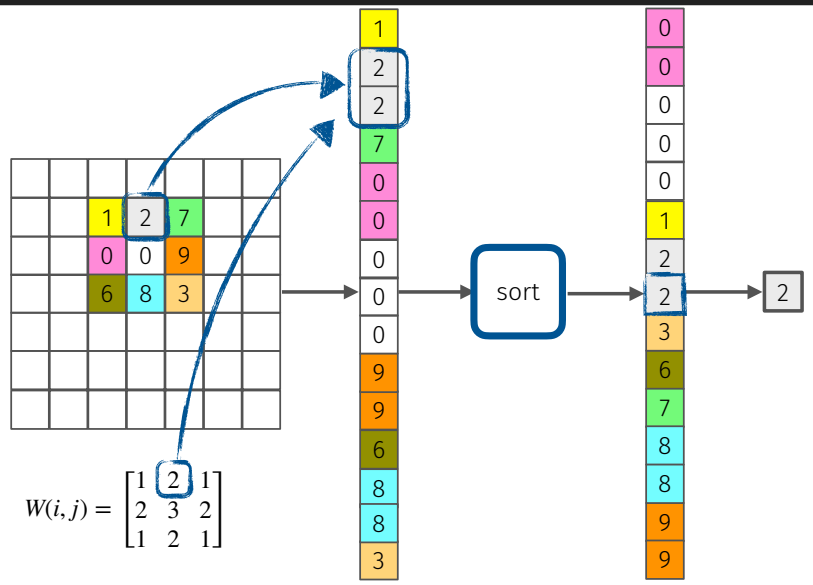
Median filter determines result by using "the majority" of pixels within the filter region

- It is considered robust since single high or low value cannot influence result (unlike linear average)
- Median filter assigns weights (number of "votes") to filter positions

$$W(i,j) = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 3 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

- To compute result, each pixel value within filter region is inserted $W(i,j)$ times to create extended pixel vector
- Extended pixel vector then sorted and median returned

Weighted median filter



Other non-linear filters

- Min and max filters
- Corner detection filters
- Morphological filters
- Also, filtering shall be discussed in frequency domain

Edge detection

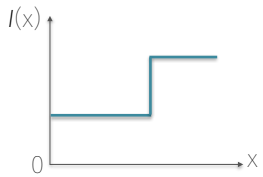
What is an edge?

An edge is a sharp change in brightness

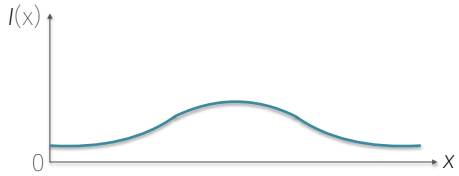
Examples:
- Boundaries between objects
- Changes in surface orientation
- Illumination changes

Characteristics of an edge

- Ideal edge is a step function in some direction

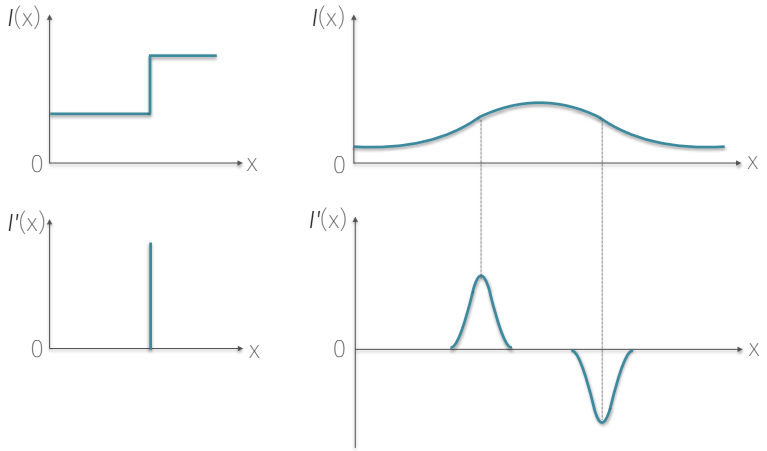


- Real (non-ideal) edge is a slightly blurred step function



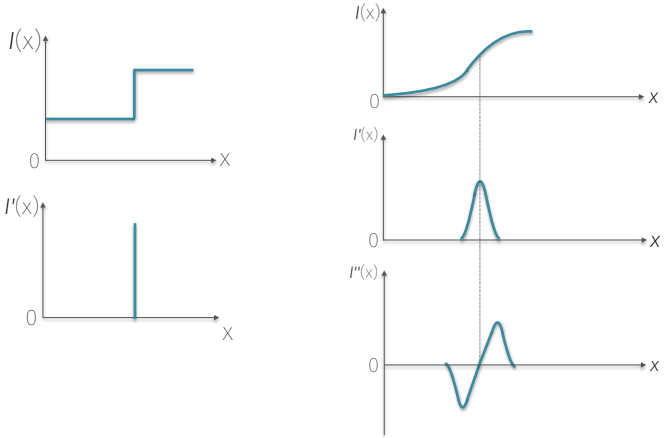
Characteristics of an edge

- Edges can be characterized by high value first derivative



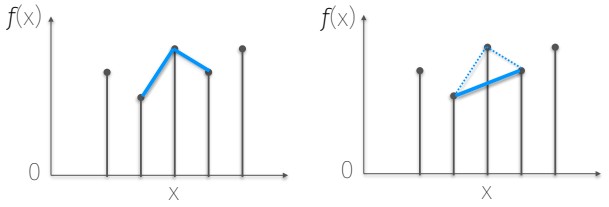
Characteristics of an edge

- First derivative of $I(x)$ has a peak at the edge
- Second derivative of $I(x)$ has a zero crossing at edge



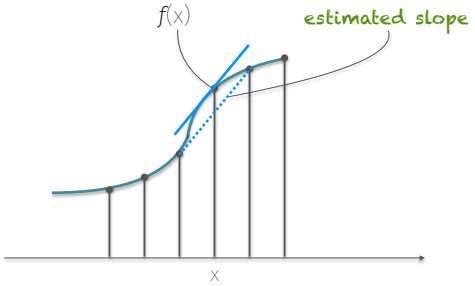
Slope of discrete functions

- Left and right slope may not be the same
- Solution? Take the average of left and right slopes



Computing derivative of discrete function

$$\frac{df}{dx}(x) \sim \frac{f(x+1) - f(x-1)}{2} = 0.5.(f(x+1) - f(x-1))$$



Finite differences

Forward difference (right slope)

$$\Delta_+ f(x) = f(x+1) - f(x)$$

Backward difference (left slope)

$$\Delta_- f(x) = f(x) - f(x-1)$$

Central difference (average slope)

$$\Delta f(x) = \frac{1}{2}(f(x+1) - f(x-1))$$

Gradient function

Let $f(x, y)$ be a 2D function

- **Gradient**: vector whose direction is in direction of maximum rate of change of f and whose magnitude is maximum rate of change of f
- Gradient is **perpendicular** to edge contour

$$\nabla f = \left[\frac{\partial f}{\partial x} \quad \frac{\partial f}{\partial y}\right]^T$$

Magnitude : $\left[(\frac{\partial f}{\partial x})^2 + (\frac{\partial f}{\partial y})^2\right]^{1/2}$

Direction : $\tan^{-1}\left(\frac{\frac{\partial f}{\partial y}}{\frac{\partial f}{\partial x}}\right)$

Image gradient

- An image is a 2D discrete function
- Image derivatives are in horizontal and vertical directions

$$\frac{\partial I}{\partial u}(u, v) \quad \text{and} \quad \frac{\partial I}{\partial v}(u, v)$$

- Image gradient at (u, v) is:

$$\nabla I(u, v) = \left[\frac{\partial I}{\partial u}(u, v) \quad \frac{\partial I}{\partial v}(u, v)\right]^T$$

- Gradient magnitude is:

$$|\nabla I|(u, v) = \left[(\frac{\partial I}{\partial u}(u, v))^2 + (\frac{\partial I}{\partial v}(u, v))^2\right]^{1/2}$$

Magnitude is invariant under image rotation!

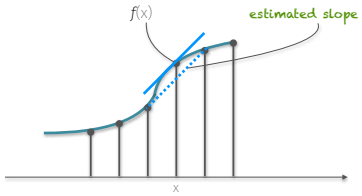
Derivative filters

- Recall that we compute derivative of a discrete function as:

$$\frac{df}{dx}(x) \sim \frac{f(x+1) - f(x-1)}{2} = 0.5.(f(x+1) - f(x-1))$$

- Can we define a linear filter to compute central differences using kernel?

$$H_x^D = [-0.5 \quad 0 \quad 0.5] = 0.5.[-1 \quad 0 \quad 1]$$



Finite differences as convolutions

- Forward differences

$$\Delta_+ f(x) = f(x+1) - f(x)$$

- Convolution kernel is: $H = [0 \quad -1 \quad 1]$

$$\Delta_+ f = f * H$$

Finite differences as convolutions

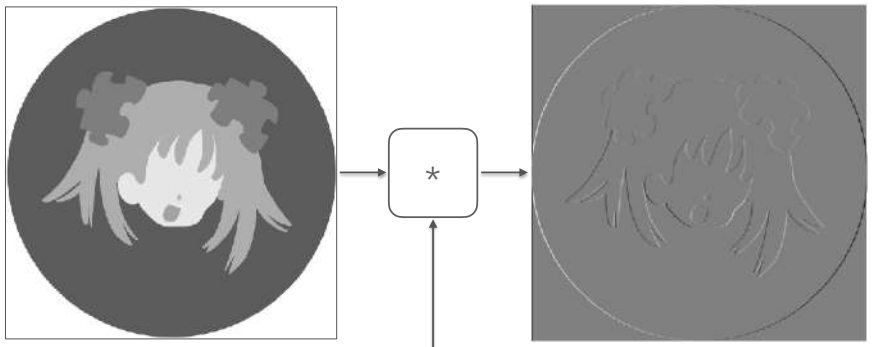
- Central differences

$$\Delta f(x) = \frac{1}{2}(f(x+1) - f(x-1))$$

- Convolution kernel is: $H = [-0.5 \quad 0 \quad 0.5]$

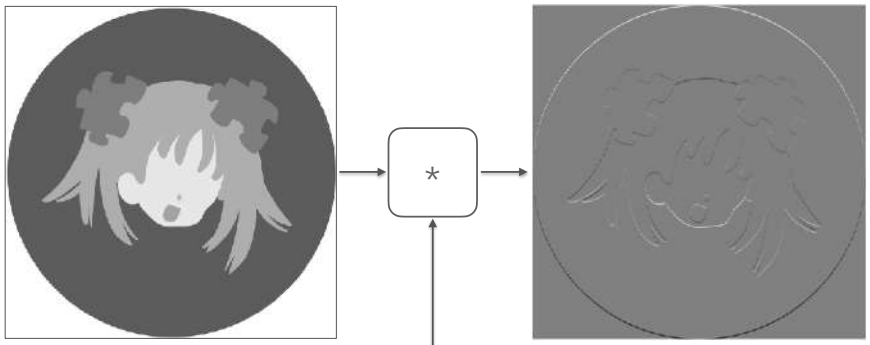
$$\Delta f = f * H$$

x-derivative of image using central difference



$H = [-0.5 \quad 0 \quad 0.5]$

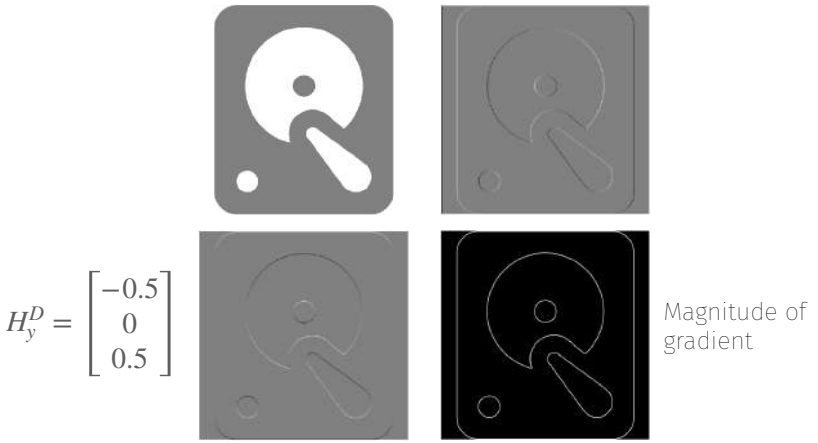
y-derivative of image using central difference



$H = [-0.5 \quad 0 \quad 0.5]^T$

Derivative filters

$H_x^D = [-0.5 \quad 0 \quad 0.5]$



$H_y^D = \begin{bmatrix} -0.5 \\ 0 \\ 0.5 \end{bmatrix}$

Magnitude of gradient

Partial image derivatives

- Partial derivatives of images replaced by finite differences

$$\Delta_x f = f(x,y) - f(x-1,y) \quad [-1 \quad 1]$$
$$\Delta_y f = f(x,y) - f(x,y-1) \quad \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

- Alternatives

$$\Delta_{2x} f = f(x+1,y) - f(x-1,y) \quad [-1 \quad 0 \quad 1]$$
$$\Delta_{2y} f = f(x,y+1) - f(x,y-1) \quad \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$$

- Robert's gradient

$$\Delta_+ f = f(x+1,y+1) - f(x,y) \quad \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$
$$\Delta_- f = f(x,y+1) - f(x+1,y) \quad \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

Prewitt

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

Sobel

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

Combining averaging with derivatives

- Finite difference operator is sensitive to noise
- Derivates is more robust if derivative computations are averaged in a neighborhood
- Prewitt operator: derivative in x, then average in y

$$H_x^{\text{Prewitt}} = \frac{1}{3} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} * [0.5 \quad 0 \quad -0.5] = \frac{1}{6} \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

Average in y direction

Derivative in x direction

H_y^{Prewitt} is defined similarly

Sobel operator

- Similar to Prewitt, but averaging kernel is higher in middle

$$H_x^{\text{Sobel}} = \frac{1}{4} \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} * [0.5 \quad 0 \quad -0.5] = \frac{1}{8} \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

$$H_y^{\text{Sobel}} = \frac{1}{4} [1 \quad 2 \quad 1] * \begin{bmatrix} 0.5 \\ 0 \\ -0.5 \end{bmatrix} = \frac{1}{8} \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Average in x direction

Derivative in y direction

Prewitt and Sobel edge operators

Prewitt operator

$$H_x^{\text{Prewitt}} = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad \text{and} \quad H_y^{\text{Prewitt}} = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Sobel operator

$$H_x^{\text{Sobel}} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad \text{and} \quad H_y^{\text{Sobel}} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

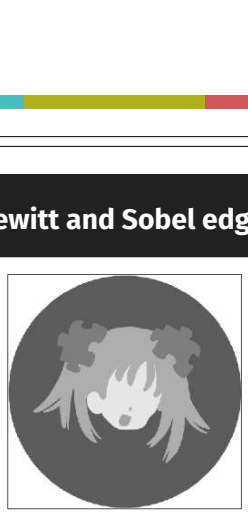
Improved Sobel operator

- Improved Sobel operator proposed by Jähne

$$H_x^{\text{betterSobel}} = \frac{1}{32}\begin{bmatrix} -3 & 0 & 3 \\ -10 & 0 & 10 \\ -3 & 0 & 3 \end{bmatrix}$$

$$H_y^{\text{betterSobel}} = \frac{1}{32}\begin{bmatrix} -3 & -10 & -3 \\ 0 & 0 & 0 \\ 3 & 10 & 3 \end{bmatrix}$$

Roberts, Prewitt and Sobel edge operators



Original

Roberts

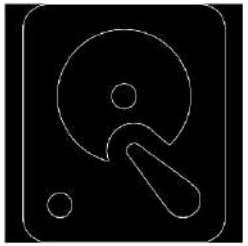
Prewitt

Sobel

Roberts, Prewitt and Sobel edge operators

Original

Roberts



Prewitt

Sobel

Roberts, Prewitt and Sobel edge operators

Original

Roberts

Prewitt

Sobel

Another edge operator

With edge operators based on first derivative:

- Edge is proportional to underlying intensity transition and may be difficult to localize precisely

Solution

Use second derivative with a pre-smoothing filter as it amplify noise

Another edge operator: Canny filter

Canny operator minimizes number of false edge points and achieves good localization of edges

Process

1. Smooth image using Gaussian filter
2. Find intensity gradients
3. Apply gradient magnitude thresholding or lower bound cut-off suppression
4. Apply double threshold to determine potential edges
5. Remove of weak edges that are not connected to strong edges

Image sharpening

Image sharpening

Blurring may occur during image scanning or scaling

- Sharpening reduces effects of blurring
- How? Amplify high frequency components
- High frequencies occur at edges

Two approaches to sharpen edges:

- Laplacian filter
- Unsharp masking

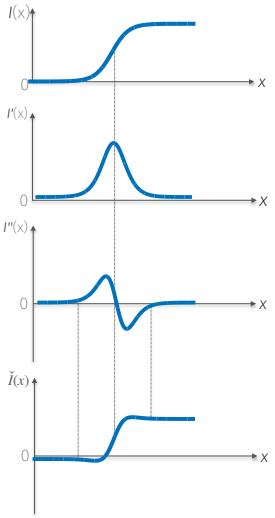
Edge sharpening using Laplace filter

Intensity of image

$$\check{I} = I(x) - w \,.\, I''(x)$$

Weight

2nd derivative of intensity



Laplacian operator

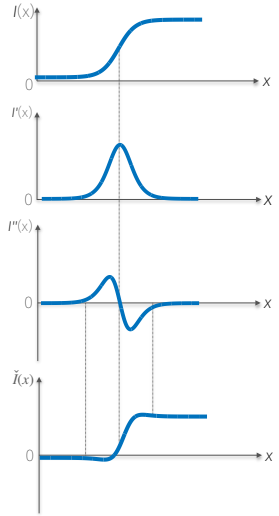
- 2D Laplacian operator combines 2nd derivatives in horizontal and vertical directions

- Laplacian operator is defined as:

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

2nd derivative of intensity in x direction

2nd derivative of intensity in y direction



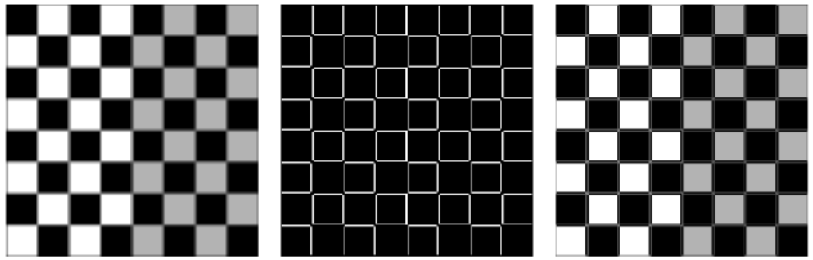
Laplacian operator

- Approximation of Laplacian:

$$\nabla^2 I(x,y) = [I(x+1,y) - I(x,y)] - [I(x,y) - I(x-1,y)] +$$
$$[I(x,y+1) - I(x,y)] - [I(x,y) - I(x,y-1)]$$
$$= [I(x+1,y) + I(x-1,y) + I(x,y+1) - I(x,y-1)] - 4I(x,y)$$

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

Laplacian operator



Original blurred image | Laplacian | Sharpen image

Sharpen image is obtained by subtracting Laplacian result from original image

Unsharp masking

1) Subtract smooth version (Gaussian smoothing) from image to obtain an enhanced edge mask

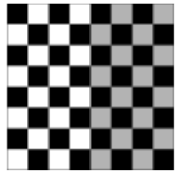
$$M = I - (I * \tilde{H})$$

2) Add the mask to image with a weight

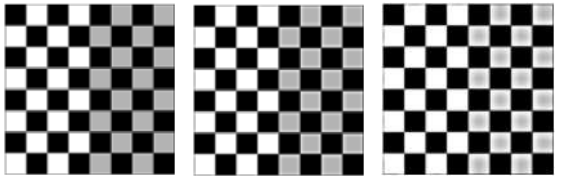
$$\check{I} = I + aM$$

Advantages of unsharp masking over Laplacian operator
- Reduced noise sensitivity due to smoothing
- Improved control through parameters σ and a (amount)

Unsharp masking



Original blurred image



Sharpen images

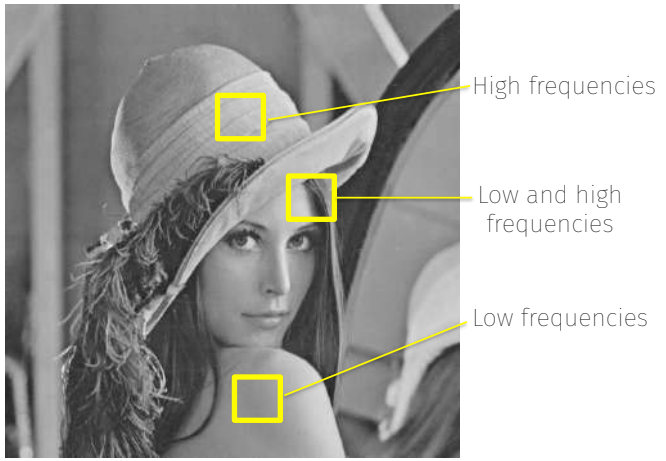
Spatial filters and Python

Spatial filters in Python

`filters` module from **scikit-image** supports most of all spatial kernels

5. Frequency domain filter

Frequencies in image



Filtering in frequency domain

One reason for using Fourier transform in image processing is due to convolution theorem

Spatial convolution can be performed by element-wise multiplication of the Fourier transform by a "filter matrix"

Discrete Fourier Transform (DFT)

1D discrete Fourier transform (1D DFT)

Let $f = [f_0, f_1, f_2, \cdots, f_{N-1}]$

Then $F = [F_0, F_1, F_2, \cdots, F_{N-1}]$

Where $F_u = \frac{1}{N} \sum_{x=0}^{N-1} f_x \exp\left[-2\pi i \frac{xu}{N}\right]$

Inverse: $x_u = \sum_{x=0}^{N-1} F_u \exp\left[2\pi i \frac{xu}{N}\right]$

2D Discrete Fourier Transform (2D DFT)

Let F the matrix of the Fourier transform of f:

$$F = \mathcal{F}(f)$$

Then the original matrix f is the inverse Fourier transform of F:

$$f = \mathcal{F}^{-1}(F)$$

For $M \times N$ matrix, forward and inverse Fourier transforms can be written:

$$F(u,v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) \exp\left[-2\pi i \left(\frac{xu}{M} + \frac{yv}{N}\right)\right],$$

$$f(x,y) = \frac{1}{MN} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u,v) \exp\left[2\pi i \left(\frac{xu}{M} + \frac{yv}{N}\right)\right].$$

Fast Fourier Transform (FFT)

- Many ways to compute DFT quickly
- Fast Fourier Transform (FFT) algorithm is one of those ways
- It divides original vector into 2
- And calculates FFT of each half recursively
- Then merges results

- Direct computation takes 2^{2n} multiplications in time
- FFT method takes $n2^n$ multiplications in time
- Time saving: $2^n/n$

Properties of 2D DFT: separability

- Notice that:

$$\underbrace{\exp\left[\pm 2\pi i \left(\frac{xu}{M} + \frac{yv}{N}\right)\right]}_{\text{2D DFT}} = \underbrace{\exp\left[\pm 2\pi i \frac{xu}{M}\right]}_{\text{1D DFT (row)}} \underbrace{\exp\left[\pm 2\pi i \frac{yv}{N}\right]}_{\text{1D DFT (column)}}$$

- Using the separability property, we can use 1D DFTs to calculate rows then columns of 2D Fourier Transform

Properties of 2D DFT: linearity

- DFT of a sum is equal to sum (or multiplication) of the individual DFT's

$$\mathcal{F}(f+g) = \mathcal{F}(f) + \mathcal{F}(g)$$

$$\mathcal{F}(kf) = k\mathcal{F}(f) \quad \text{where } k \text{ is a scalar}$$

- Useful property when processing degradations expressed as a sum (e.g. noise)

$$f' = f + n$$

$$\mathcal{F}(f') = \mathcal{F}(f) + \mathcal{F}(n)$$

⇒ The noise can be removed or reduced by modifying the transform of n

Convolution using DFT

- DFT provides alternate method to do **convolution** of image I with spatial filter S

$$I * S = \mathcal{F}^{-1}(\mathcal{F}(I) \,.\, \mathcal{F}(S'))$$

where S' is such that: S *has been padded* to make it same size as I

Convolution using DFT

Example: I = 512x512, S = 32x32

Direct computation

- 32^2 = 1024 multiplications for each pixel
- Total multiplications for entire image = 512 x 512 x 1024 = 268,435,456

Using DFT

- Each row requires 4608 multiplications
- Multiplications for rows = 4608 x 512 = 2,359,296 multiplications
- Repeat for columns, DFT of image = 4,718,592 multiplications
- Need same for DFT of filter and for inverse DFT
- Also need 512 x 512 multiplications for product of 2 transforms
- Total multiplications = 4,718,592 x 3 + 262144 = 14,417,920

DC component of the DFT

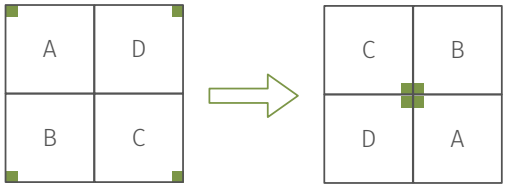
The value $F(0,0)$ of the DFT is called the DC coefficient

$$F(0,0) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)\exp(0) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)$$

It is the sum of all terms in the original matrix

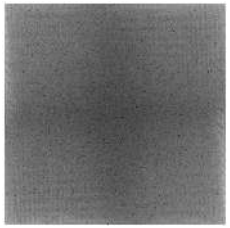
Centering DFT spectrum

- $F(0,0)$ is at top left corner
- To display DFT, it is more convenient to have DC component in center
- We just swap four quadrants of the DFT

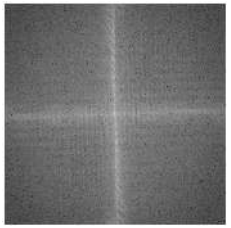


Centering DFT spectrum

Original image



Non-centered
$|F(u,v)|$

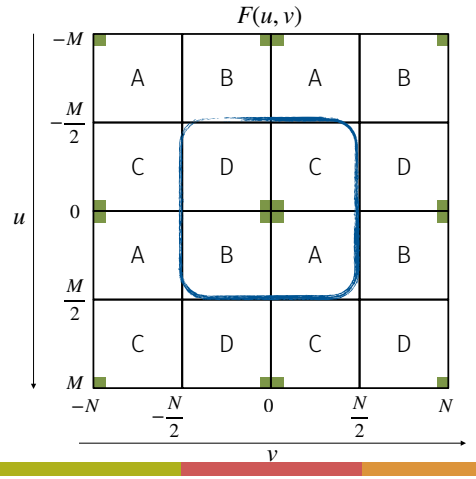


Centered $|F(u,v)|$

DC component is much larger than other values, so we stretch transform values by displaying log of transform: $log(1+|F(u,v)|)$

Conjugate symmetry

- DFT shows conjugate symmetry
- Half of the transform is mirror image of conjugate of other half
- Information is contained in only half of a transform, other half is redundant



Some examples of DFT

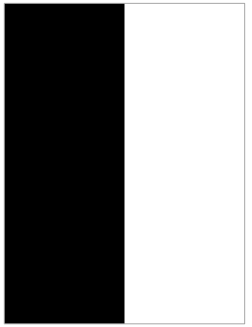
DFT of constant image

- Suppose input is a 13x8 constant image of all 1's: $I(x, y) = 1$
- The DFT yields just a DC component, 0 everywhere else
- The DC component is sum of all elements = 104

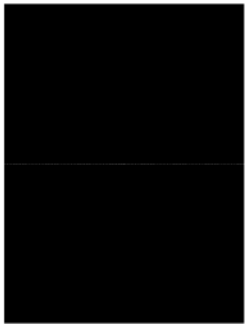
0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	104	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0

DFT of constant image

- Consider DFT of image with single edge:



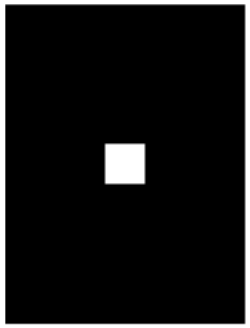
Original image



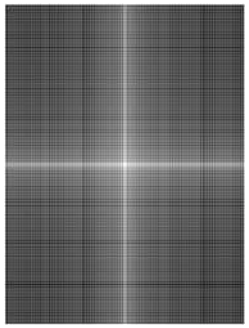
log(1+|F|)

DFT of square image

- Consider DFT of image with a synthetic square:



Original image

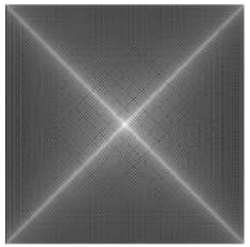


log(1+|F|)

DFT of rotated square image

- Consider DFT of image with a synthetic rotated square:

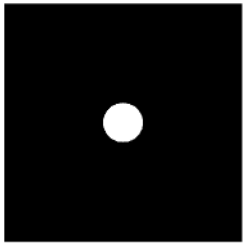
Original image



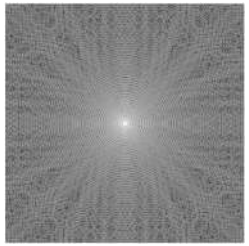
log(1+|F|)

DFT of circle image

- Consider DFT of image with a synthetic circle:



Original image



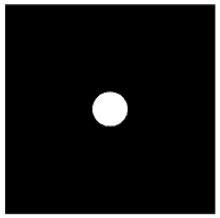
log(1+|F|)

Filtering in frequency domain

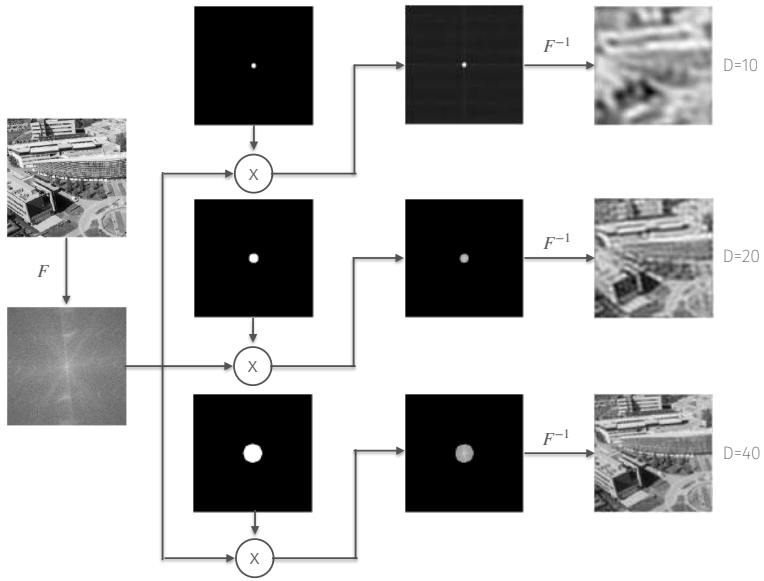
Ideal lowpass filter

- Low pass filter keeps frequencies below a certain frequency
- Low pass filtering causes blurring
- After DFT, DC components and low frequency components are towards center
- May specify a frequency cutoff using a disk

$$m(x,y) = \begin{cases} 1 & \text{if } (x,y) \text{ is closer to the center than some value } R \\ 0 & \text{if } (x,y) \text{ is further from the center than } R \end{cases}$$



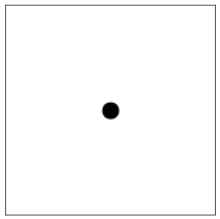
Ideal lowpass filter



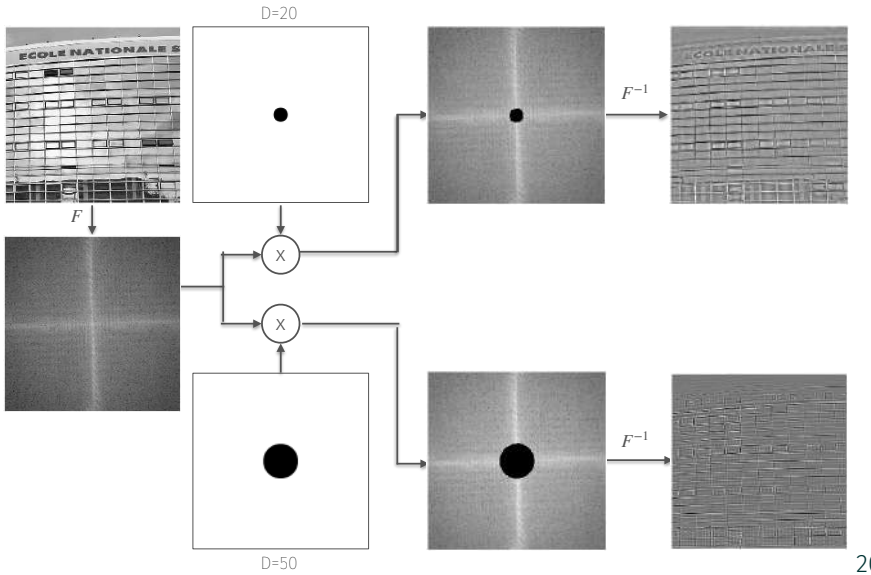
Ideal highpass filter

Opposite of low pass filtering: eliminate center (low frequency values) and keep others

- High pass filtering causes image sharpening
- If we use circle as cutoff again, size affects results
- Large cutoff ⇒ more information removed



Ideal highpass filter

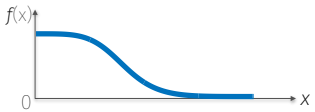


Butterworth filtering

- Sharp cutoff leads to ringing (oscillation artifacts)
- Circle with more gentle cutoff slope can be used to avoid ringing
- Butterworth filters have a more gentle cutoff slope

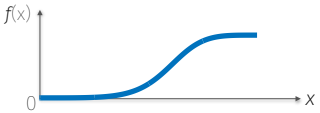
Butterworth low pass filter

$$f(x) = \frac{1}{1 + (x/D)^{2n}}$$



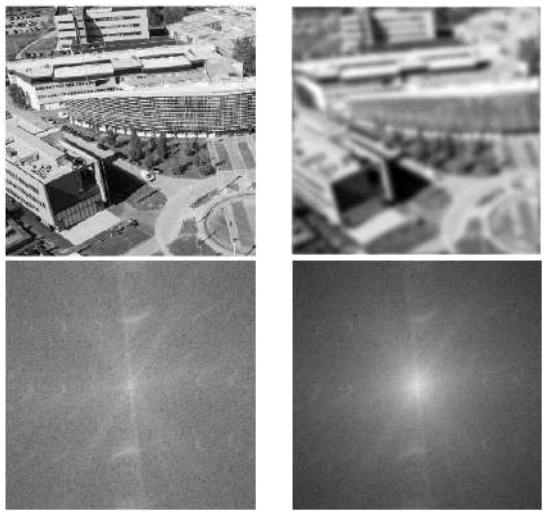
Butterworth high pass filter

$$f(x) = \frac{1}{1 + (D/x)^{2n}}$$



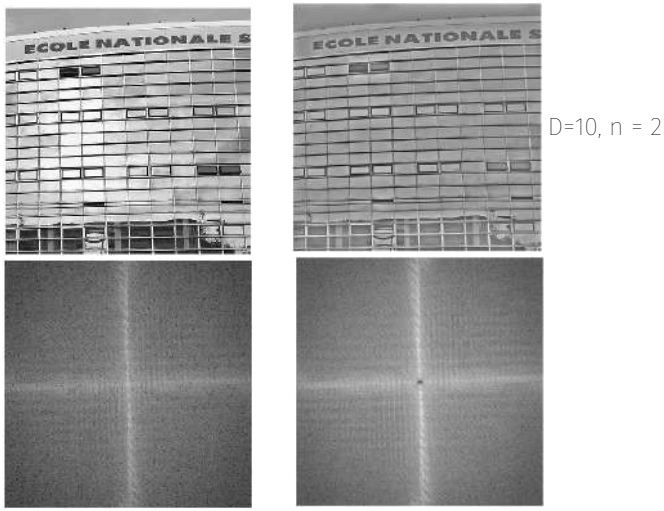
n is called order of the filter and controls sharpness of cutoff

Butterworth lowpass filter



D=20, n = 2

Butterworth highpass filter



D=10, n = 2

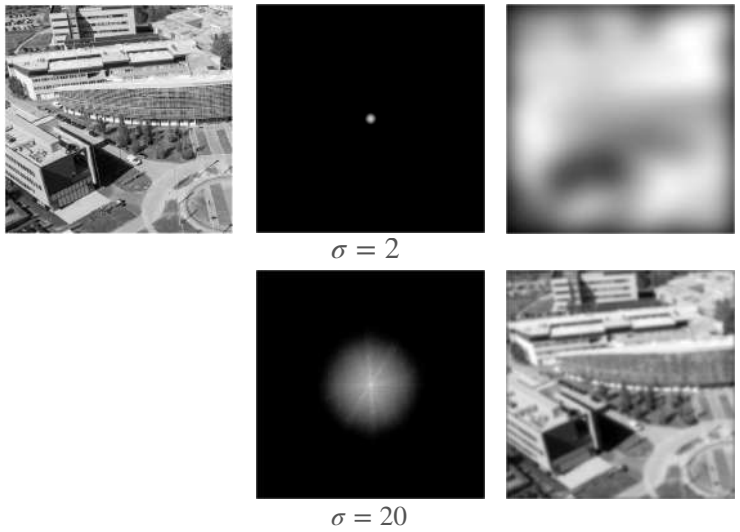
Gaussian filtering

Gaussian filters can be applied in frequency domain

- Create gaussian filter
- Multiply (DFT of image) by (gaussian filter)
- Invert result

- As Fourier transform of a gaussian is also a gaussian, just multiply with gaussian directly (no need to find its Fourier transform)

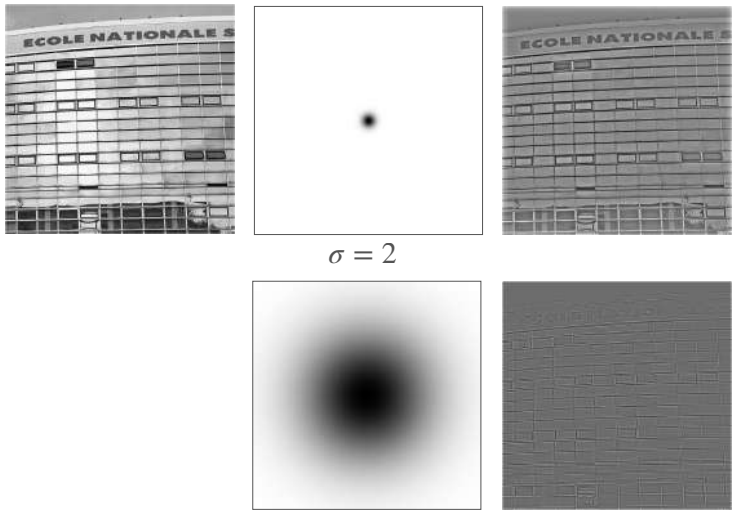
Gaussian lowpass filter



$\sigma = 2$

$\sigma = 20$

Gaussian highpass filter

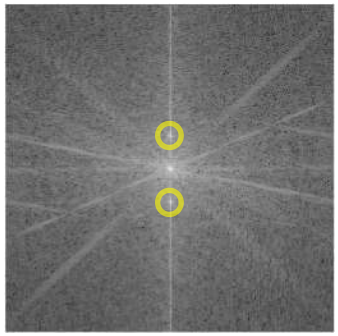


$\sigma = 2$

$\sigma = 20$

Removal of periodic noise

- Periodic noise could not be removed in spatial domain
- Periodic noise can be removed in frequency domain
- Periodic noise shows up as spikes away from origin in DFT

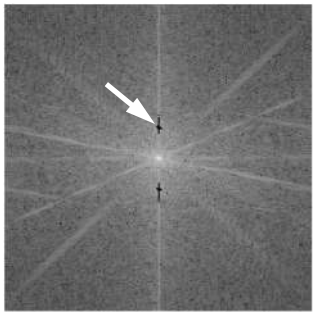
Removal of periodic noise

2 ways to remove periodic noise in frequency domain:

- Notch Filter
- Band reject filter

Notch filters set to 0 rows and columns of DFT corresponding to noise

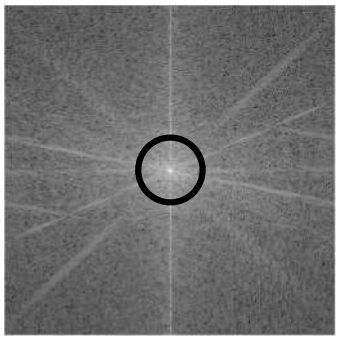
They removes much of the periodic noise




Removal of periodic noise

Band Reject Filters

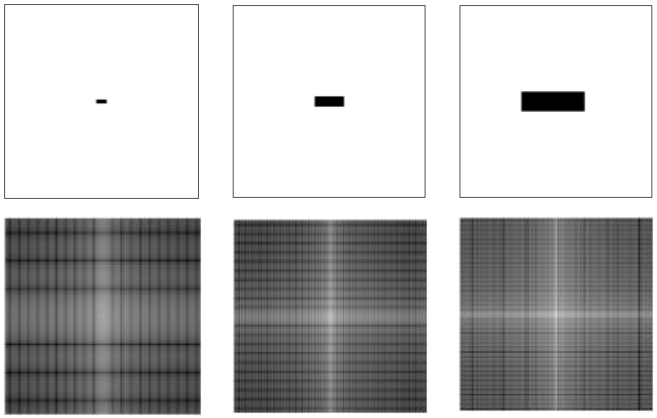
- Create filter with 0's at radius of noise from center, 1 elsewhere
- Apply filter to DFT




More examples of DFT

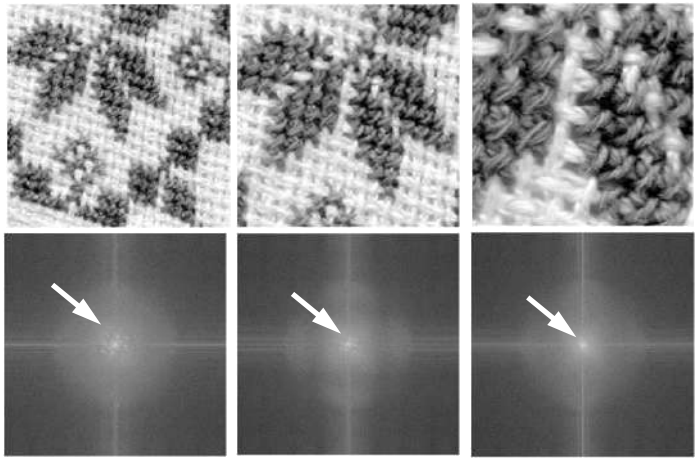
2D Fourier transform examples: scaling

Stretching image corresponds to spectrum contracts and vice versa



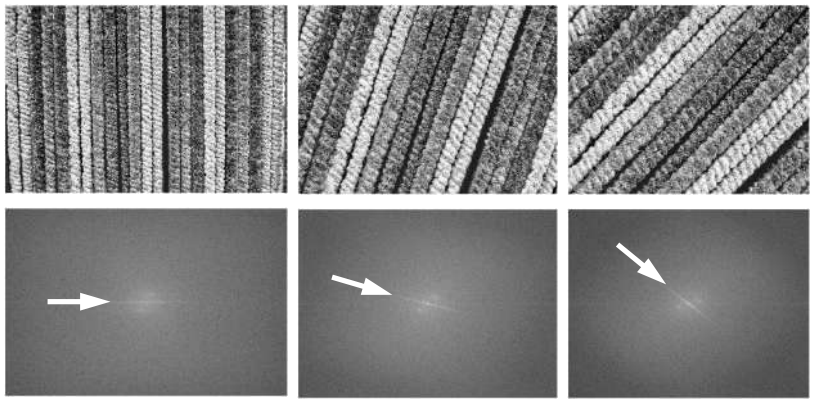
2D Fourier transform examples: periodic patterns

Repetitive periodic patterns appear as distinct peaks at corresponding positions in spectrum



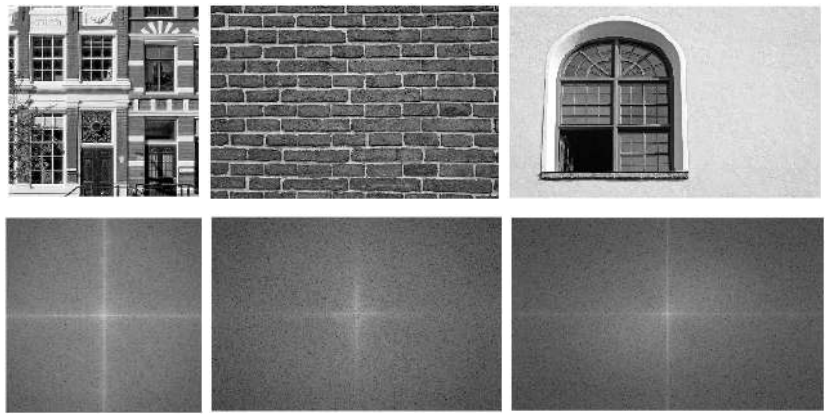
2D Fourier transform examples: rotation

Rotation corresponds to spectrum rotation by same angle/amount



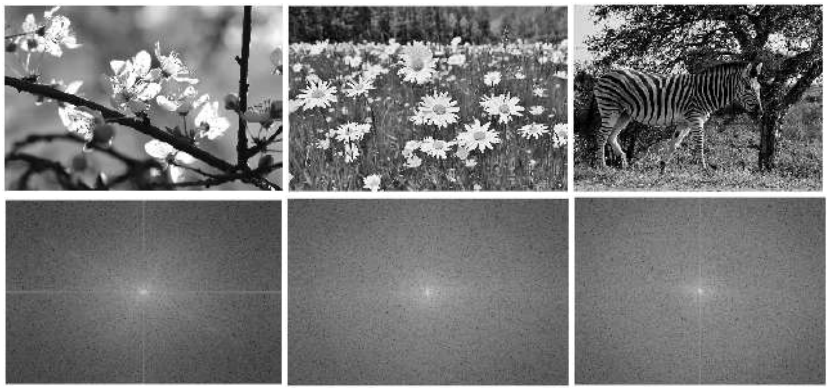
2D Fourier transform examples: oriented, elongated structures

Man-made elongated regular patterns appear dominant in spectrum



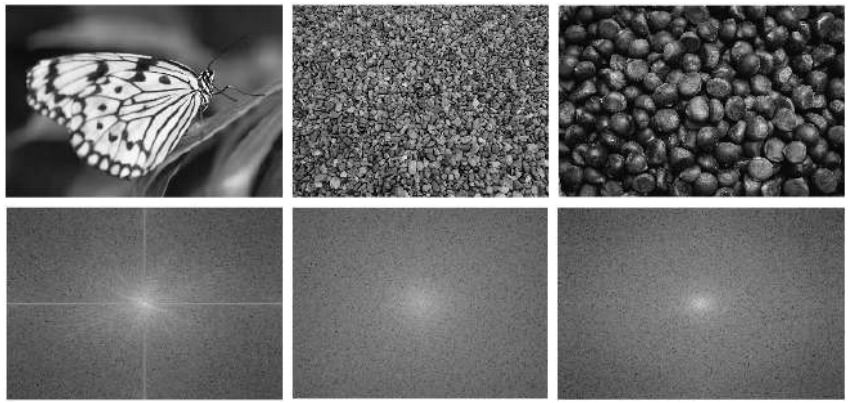
2D Fourier transform examples: natural scenes

Repetitions in natural scenes are less dominant than man-made ones



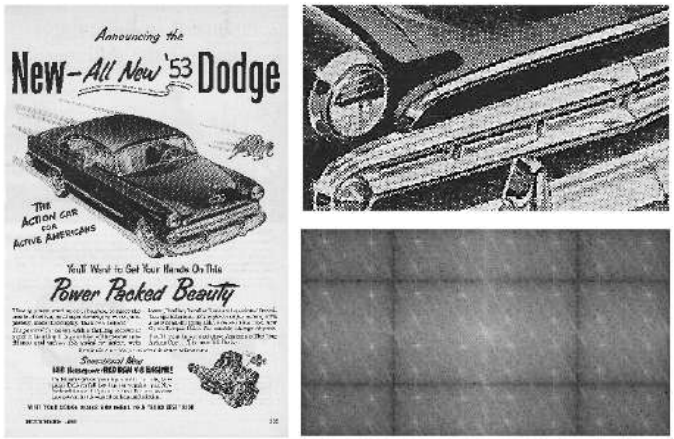
2D Fourier transform examples: natural scenes

Repetitive patterns in natural images have no dominant orientation



2D Fourier transform examples: printed patterns

Regular diagonal patterns caused by printing are clearly visible/ removable in spectrum



Frequency domain filters and Python

Frequency domain filters in Python

NumPy and SciPy both support FFT. We will use NumPy's package.

6. Mathematical morphology

Introduction

- Introduced in 1964 by Georges Matheron and Jean Serra (École des Mines de Paris)
- Originally operated on binary images (black and white) for mining applications

What kind of binary image?
- Faxes, digitally printed images
- Thresholded grayscale images

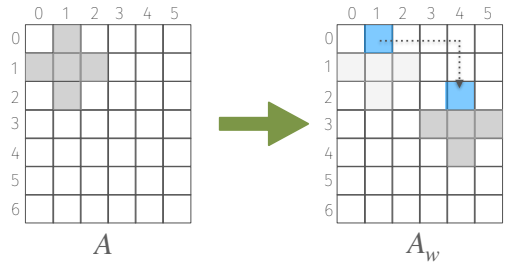
Morphological filters alter local structures in an image

Translation

Let A a set of pixels in a binary image
- $w = (x, y)$ is a particular coordinate point
- A_w is set A "translated" in direction (x, y). i.e

$$A_w = \{(a, b) + (x, y) : (a, b) \in A\}$$

Example: If A is the cross-shaped set, and $w = (3,2)$



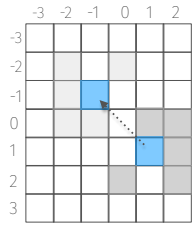
Reflection

Let A a set of pixels

- Reflection of A is given by:

$$\hat{A} = \{(-x, -y) : (x, y) \in A\}$$

Example:



Dilation and erosion

Dilation and erosion

2 basic operations (built from *translations* and *reflections*)

- Dilation
- Erosion

Several composite operations

- Opening
- Closing
- etc.

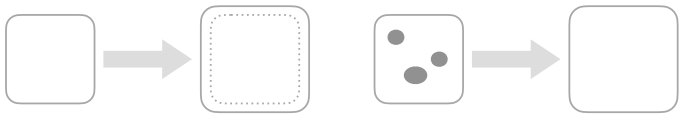
Dilation

Dilation **expands connected sets of 1** in a binary image.

Used for:

Growing features

Filling holes



Erosion

Erosion shrinks connected sets of 1 in a binary image.

Used for:

- Shrinking features
- Removing branches and bridges

Neighborhood

4-Neighborhood ($\mathcal{N}_4$): 4 pixels adjacent to given pixel in horizontal and vertical directions

8-Neighborhood ($\mathcal{N}_8$): 4 pixels in $\mathcal{N}_4$+ 4 pixels adjacent along diagonals

$\mathcal{N}_4$ $\mathcal{N}_8$

Formal specification as point sets

Morphological operations can be expressed by describing images as 2D point sets

For example, let I an image such that $I(u,v) \in \{0,1\}$, then:

$$\mathcal{Q}_I = \{\ p \mid\ I(p) = 1\ \}, \text{ and}$$

OR operation is union of individual sets:

$$\mathcal{Q}_{I_1 \vee I_2} = \mathcal{Q}_{I_1} \cup \mathcal{Q}_{I_2}$$

AND operation is intersection of individual sets:

$$\mathcal{Q}_{I_1 \wedge I_2} = \mathcal{Q}_{I_1} \cap \mathcal{Q}_{I_2}$$

Dilation

Suppose A and B are sets of pixels, dilation of A by B is:

$$A \oplus B = \cup_{x \in B} A_x$$

- This is called Minkowski addition
- For every pixel x in B:

1) Translate A by x
2) Take union of all these translations

$$A \oplus B = \{\ (x,y) + (u,v) \mid (x,y) \in A, (u,v) \in B\ \}$$

Properties of dilation

Dilation is commutative

$$I \oplus H = H \oplus I$$

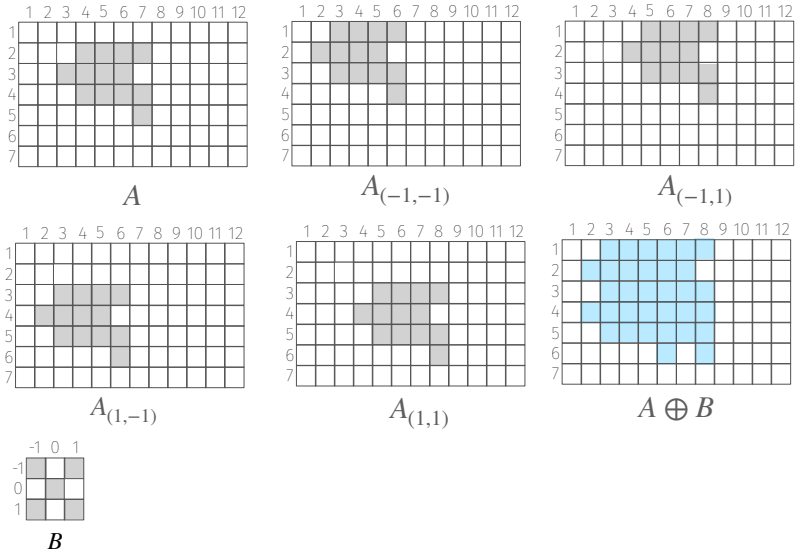
Dilation is associative (ordering of applying it not important)

$$(I_1 \oplus I_2) \oplus I_3 = I_1 \oplus (I_2 \oplus I_3)$$

It is more efficient to apply large structuring element as sequence of smaller structuring elements

$$I \oplus H = (\cdots((I \oplus H_1) \oplus H_2) \oplus \cdots \oplus H_n)$$

Dilation example

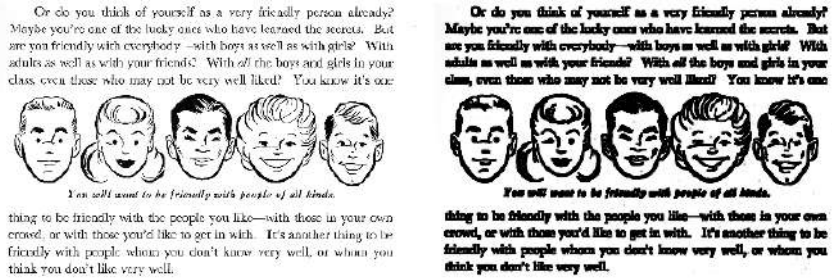


Structuring element

- We assume B is a smaller set of pixels than A
- B is called the structuring element
- A structuring element is a shape mask used in morphological operations
- It contains only 0 and 1 values
- It can be any shape and size (digitally representable), and it has an origin
- *The structuring element does not have to contain the origin!*

Dilation example on binary image

Dilation using a disk of radius 3 as the structuring element



Erosion

Given sets A and B, the erosion of A by B is:

$$A \ominus B = \cap_{b \in B} A_b$$

- This is called Minkowski subtraction
- It finds all occurrences of B in A

$$A \ominus B = \{ (x, y) \mid B(x, y) \subseteq A \}$$

Properties of erosion

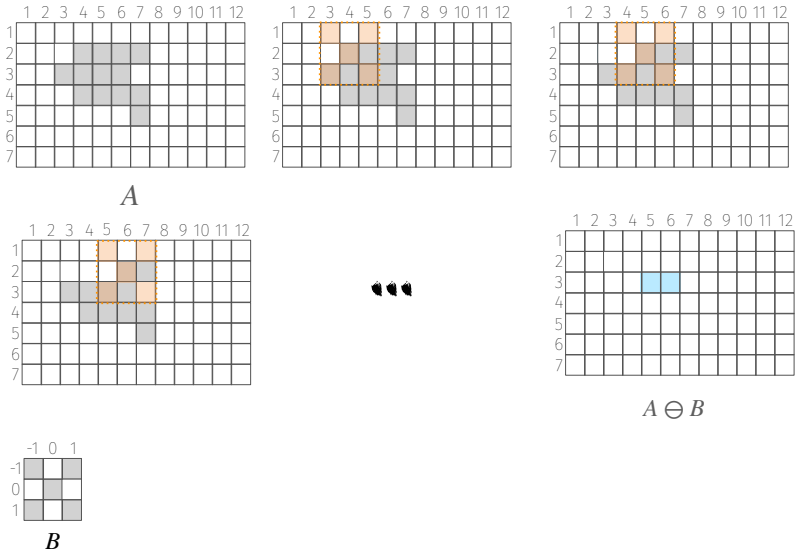
Erosion is not commutative

$$I \ominus H \neq H \ominus I$$

Dilation of foreground = erosion of background

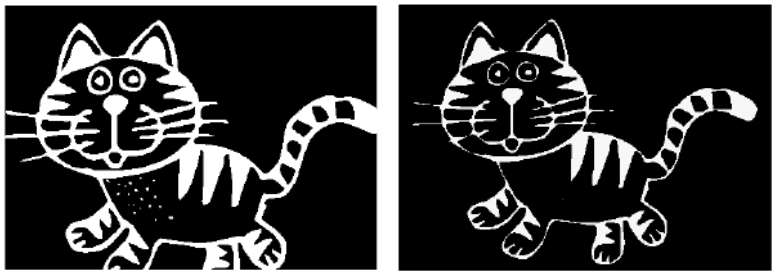
$$I \oplus H = \overline{\bar{I} \ominus H^*}$$

Erosion example



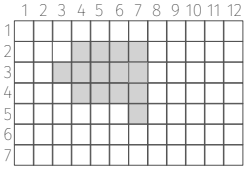
Erosion example on binary image

Erosion using a disk of radius 5 as the structuring element

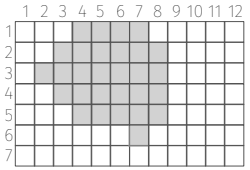


Boundary detection

Application: boundary detection

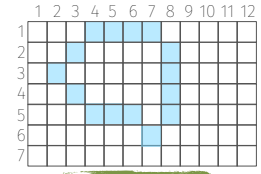


A



$A \oplus B$

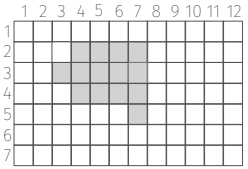
B



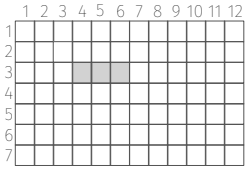
$(A \oplus B) - A$

External boundary

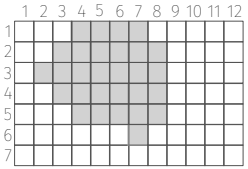
Application: boundary detection



A

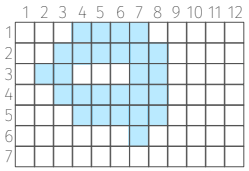


$A \ominus B$



$A \oplus B$

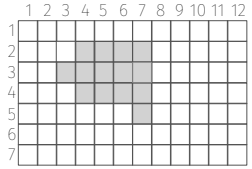
B



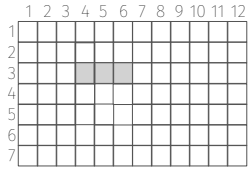
$(A \oplus B) - (A \ominus B)$

Morphological gradient

Application: boundary detection

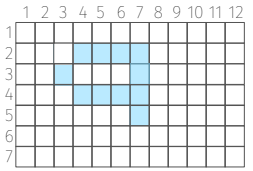


A



$A \ominus B$

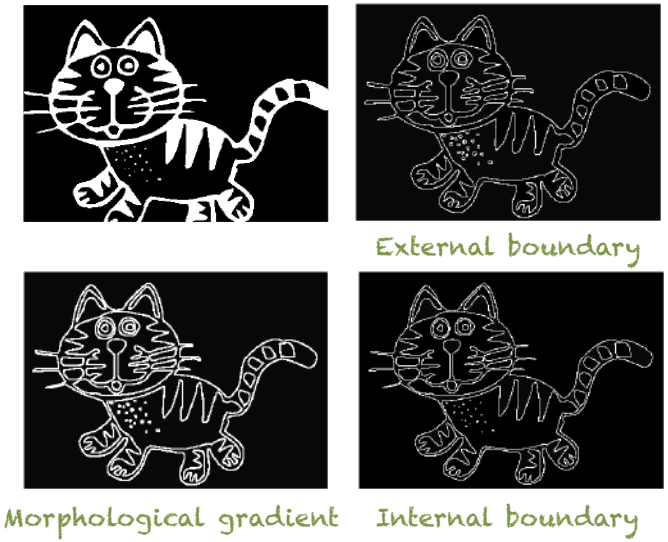
B



$A - (A \ominus B)$

Internal boundary

Application: boundary detection

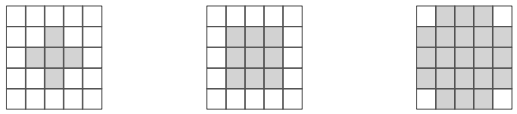


Designing morphological filters

Designing morphological filters

A morphological filter is specified by:

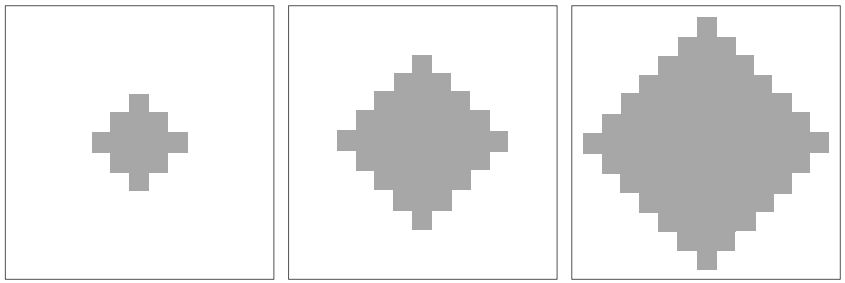
- Type of operation (e.g. dilation, erosion)
- Type of structuring element



- "Circular" structuring elements are often used in practice
- Dilation with circular structuring of radius R adds thickness R
- Erosion with circular structuring of radius R removes thickness R

Designing morphological filters

Composing large filters by repeatedly applying smaller filters



H_A $H_A \oplus H_A$ $H_A \oplus H_A \oplus H_A$

And why not $H_A \oplus H_B \oplus H_A \oplus H_B \oplus H_A$ (LoL)

Opening and closing

Opening

Opening is built on dilation and erosion

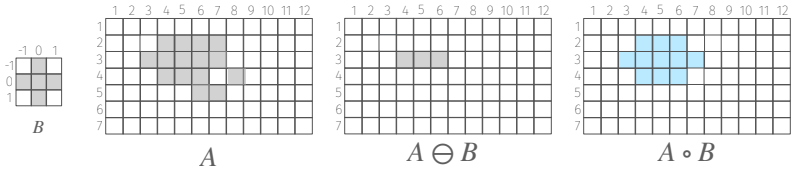
- Opening of A by structuring element B is:

$$A \circ B = (A \ominus B) \oplus B$$

- Opening is an erosion followed by a dilation. Alternatively:

$$A \circ B = \{\ B(x,y) \mid B(x,y) \subseteq A\ \}$$

- Opening is the union of all translations of B that fit in A



Opening

- Foreground structures smaller than structuring element are eliminated by first step (erosion)
- Remaining structures are smoothed by next step (dilation) then grown back to their original size

disk (R=1) disk (R=5) disk (R=11)

Properties of opening

Opening is subset of A:

$$A \circ B \subseteq A$$

Idempotence (opening can be applied only once)

$$(A \circ B) \circ B = A \circ B$$

Subsets:

$$\text{If } A \subseteq C, \text{ then } (A \circ B) \subseteq (C \circ B)$$

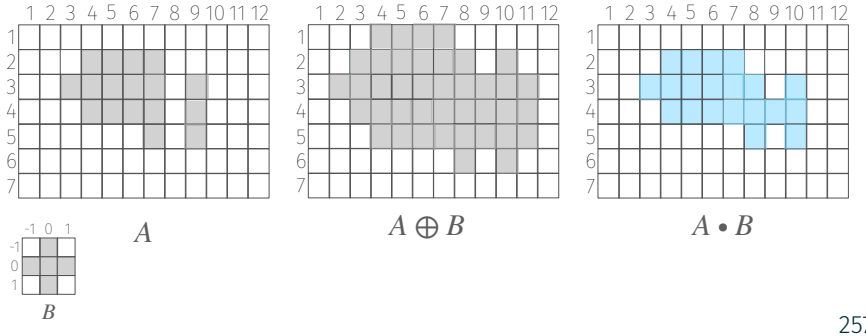
Closing

Closing is built on dilation and erosion

- Closing of A by structuring element B is:

$$A \bullet B = (A \oplus B) \ominus B$$

- Closing is a dilation followed by an erosion



Properties of closing

A is subset of closing:

$$A \subseteq (A \bullet B)$$

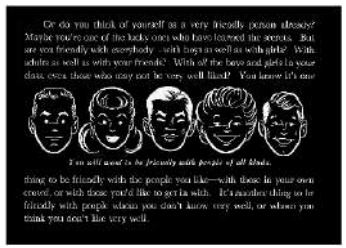
Idempotence (closing can be applied only once)

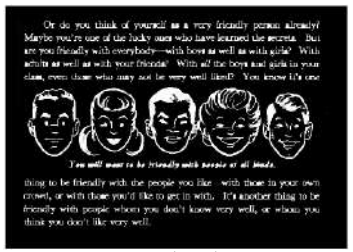
$$(A \bullet B) \bullet B = A \bullet B$$

Subsets:

If $A \subseteq C$, then $(A \bullet B) \subseteq (C \bullet B)$

Example of closing





disk (R=3)

disk (R=5)

Relationships between opening and closing

- Opening and closing are duals (opening foreground = closing background, and vice versa)

- Complement of a closing = the opening of the complement

$$\overline{A \bullet B} = \overline{A} \circ \hat{B}$$

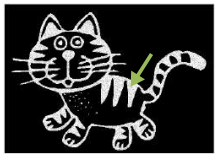
- Complement of an opening = the closing of the complement

$$\overline{A \circ B} = \overline{A} \bullet \hat{B}$$

Morphological filtering

Noise removal

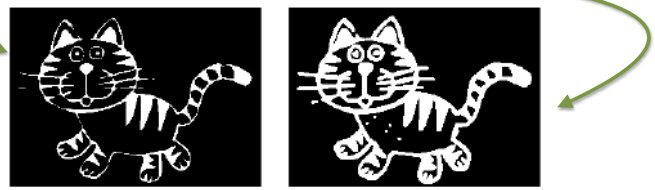
Suppose A, an image corrupted by speckle noise



- $A \ominus B$ removes single black pixels, but enlarge holes

-We can fill holes by dilating twice

$$((A \ominus B) \oplus B) \oplus B)$$



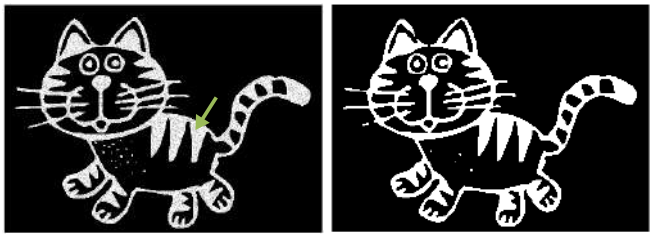
$B =$ disk(R=3)

Noise removal

- First dilation returns the holes to their original size
- Second dilation removes the holes, but enlarges objects in image
- To reduce them to their correct size, we perform a final erosion:

$$((A \ominus B) \oplus B) \oplus B) \ominus B$$

- This noise removal method corresponds to opening followed by closing: $(A \circ B) \bullet B$



$B =$ disk(R=3)

Grayscale morphology

Grayscale morphology

Morphology operations can also be applied to grayscale images

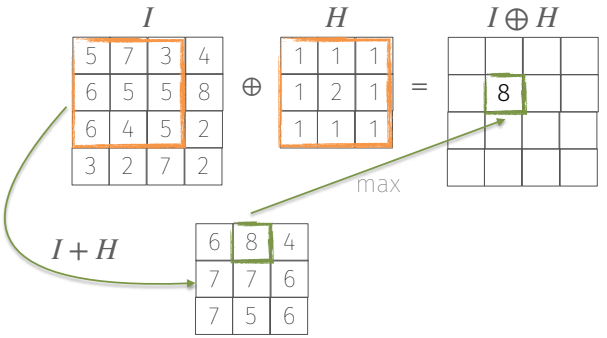
- Just replace (OR, AND) with (max, min)
- Grayscale morphology can also operate on binary images
- For color images, perform grayscale morphology operations on each color channel (RGB)
- For grayscale images, structuring element may contain real values (values may also be negative)

Grayscale dilation

Let a grayscale image I and a filter H

Dilation of A by H is:

$$(I \oplus H)(u,v) = \max_{(i,j)\in H}\{I(u+i, v+j) + H(i,j)\}$$

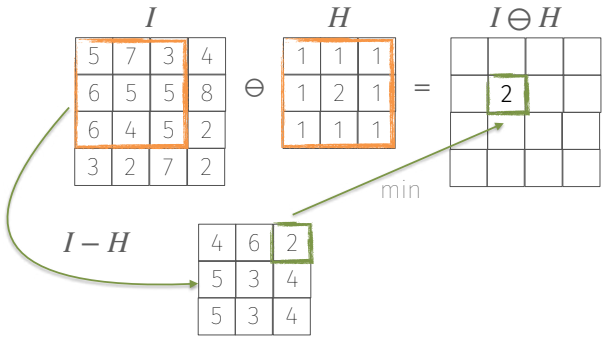


Grayscale erosion

Let a grayscale image I and a filter H

Erosion of A by H is:

$$(I \ominus H)(u,v) = \min_{(i,j)\in H}\{I(u+i, v+j) - H(i,j)\}$$



Grayscale dilation and erosion examples

Grayscale dilation with disk-shaped structuring elements of radius R = 1 and 2

Original R=1 R=2

Grayscale dilation and erosion examples

Grayscale erosion with disk-shaped structuring elements of radius R = 1 and 2

Original R=1 R=2

Grayscale opening and closing

Recall:

- opening = erosion then dilation
- closing = dilation then erosion

So we can implement grayscale opening as grayscale erosion then grayscale dilation

And grayscale closing as grayscale dilation then grayscale erosion

Grayscale opening and closing examples

Grayscale opening with disk-shaped structuring elements of radius R = 1 and 2

Original R=1 R=2

Grayscale opening and closing examples

Grayscale closing with disk-shaped structuring elements of radius R = 1 and 2

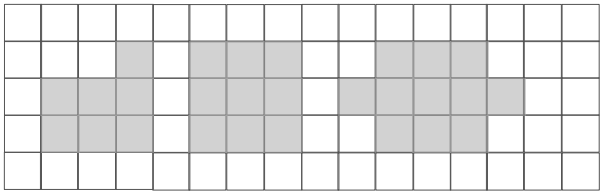
Original R=1 R=2

Other useful operations

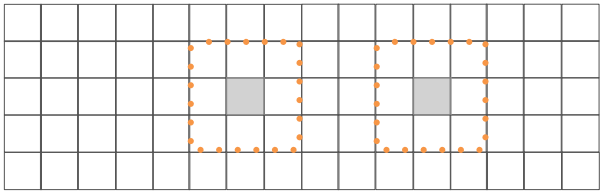
Hit-or-Miss transform

Powerful method for **finding shapes** in images

- Can be defined in terms of erosion
- Suppose we want to locate 3x3 square shapes in image below

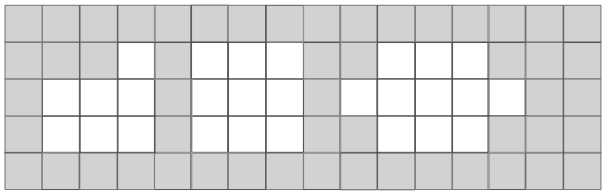


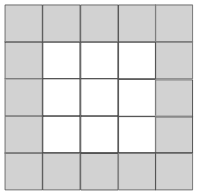
If we perform an erosion $A \ominus B$ with B being the square element, result is:



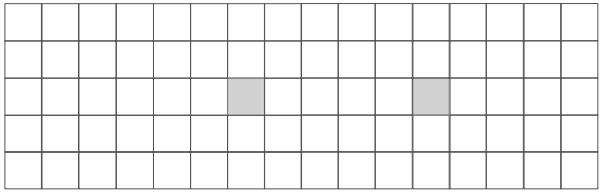
Hit-or-Miss transform

If we erode the complement of A, with a structuring element C that fits around 3x3 square





Result of $\overline{A} \ominus C$ is:



Intersection of 2 erosion operations produces 1 pixel at center of 3x3 square

Generalized Hit-or-Miss transform

If we are looking for a particular shape in an image, design two structuring elements:

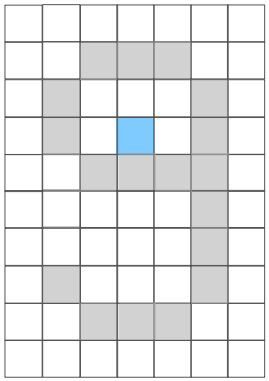
- B_1 that is same as the shape we are looking for, and
- B_2 that fits around the shape
- We can write $B = (B_1, B_2)$

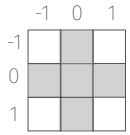
The hit-or-miss transform can be written as:

$$A \circledast B = (A \ominus B_1) \cap (\overline{A} \ominus B_2)$$

Region filling algorithm

Suppose an image has an 8-connected boundary





B

Given a pixel p within the region, we want to fill region

1. Start with p and dilate as necessary with the cross-shaped structuring element B

Region filling algorithm

2. After each dilation, intersect with $\overline{A}$ before continuing

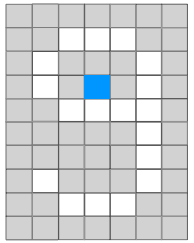
3. Then, we create the sequence:

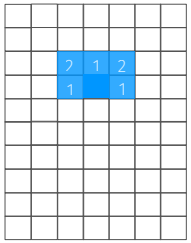
$$\{p\} = X_0, X_1, \cdots, X_k = X_{k+1}$$

for which

$$X_n = (X_{n-1} \oplus B) \cap \overline{A}$$

Finally, $X_k \cup A$ is the filled region





Connected component

Similar algorithm is used to find connected components in images
- Cross-shaped structuring element for 4-connected components
- Square-shaped structuring element for 8-connected components
- To fill rest of component by creating sequence of sets

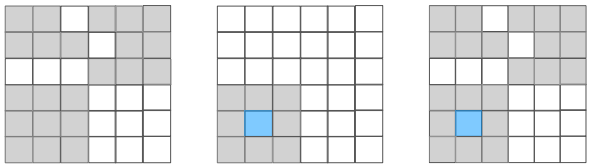
$$X_0 = \{p\}, X_1, X_2, \cdots$$

such that

$$X_n = (X_{n-1} \oplus B) \cap A$$

until $X_k = X_{k-1}$

Using Cross Using Square



Skeletonization

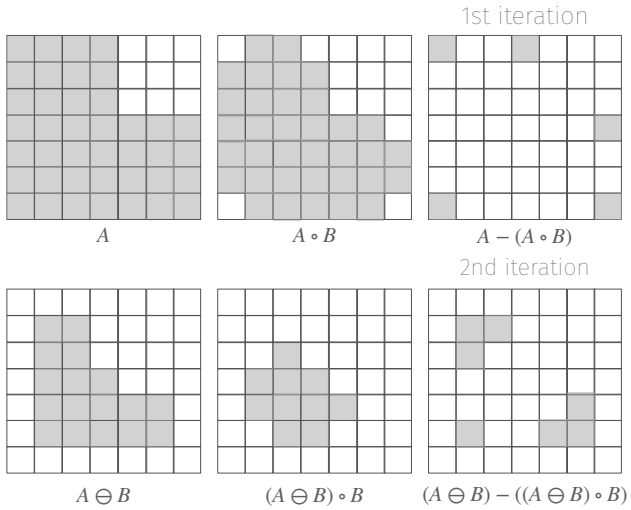
Table of operations used to construct a skeleton

Erosions	Openings	Set differences
A	$A \circ B$	$A - (A \circ B)$
$A \ominus B$	$(A \ominus B) \circ B$	$(A \ominus B) - ((A \ominus B) \circ B)$
$A \ominus 2B$	$(A \ominus 2B) \circ B$	$(A \ominus 2B) - ((A \ominus 2B) \circ B)$
$A \ominus kB$	$(A \ominus kB) \circ B$	$(A \ominus kB) - ((A \ominus kB) \circ B)$

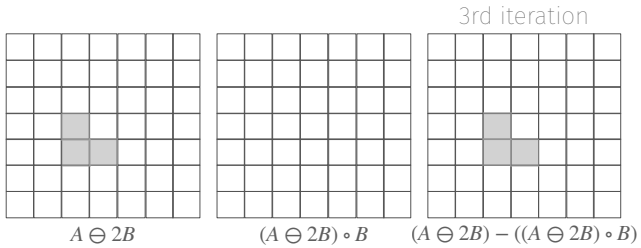
Sequence of k erosions with same structuring element: $A \ominus kB$

- Continue table until $(A \ominus kB) \circ B$ is empty
- Skeleton is union of all set differences

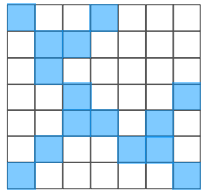
Example of skeletonization



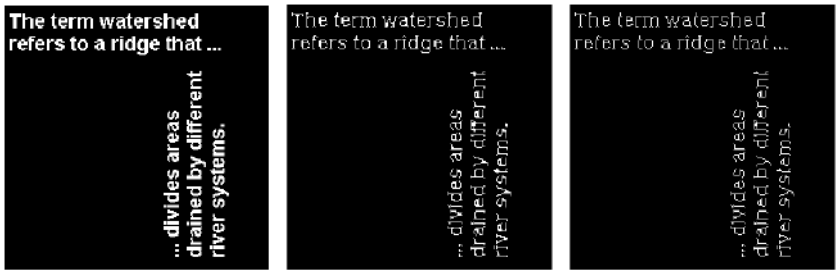
Example of skeletonization



Skeleton is union of three iterations



Examples of skeletonization



Mathematical morphology in Python

`morphology` module from **scikit-image** supports binary and grayscale operators

7. Regions in binary images

Introduction

- Vision task needs to recognize objects in black and white images:
  - Text on page
  - Objects in a picture
  - Microscope images
- Image may be grayscale (need conversion to black and white)

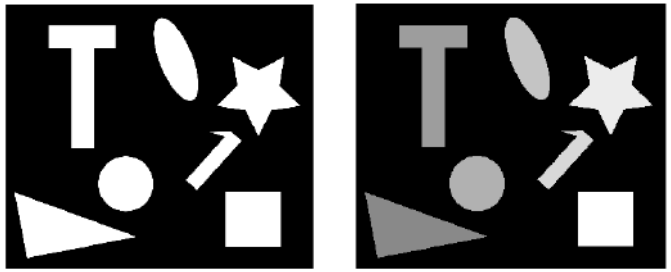
- Objects are found by grouping together connected groups of pixels that belong to it
- Each object define a binary region called connected component
- Then we can determine type of object by comparing them to models

Finding regions

Find image regions

- Tasks when searching for binary regions
  - Which pixels belong to which regions?
  - How many regions are in image?
  - Where are regions located?
- These tasks usually performed during region labeling
  - Find regions step by step and assign a label to identify it
- 3 methods:
  - Flood filling
  - Sequential region labeling
  - Combination of region labeling + contour finding

Example

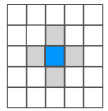


Original image

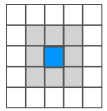
Labeled image

Find image regions

1. Consider 4-connected ($\mathcal{N}_4$) or 8-connected ($\mathcal{N}_8$) pixels as neighborhood



$\mathcal{N}_4$



$\mathcal{N}_8$

2. Adopt following convention in binary images:

$$I(u,v) = \begin{cases} 0 & \text{background pixel} \\ 1 & \text{foreground pixel} \\ 2,\ 3,\ \cdots & \text{region labels} \end{cases}$$

Finding regions: flood filling

Region labeling with the flood fill method

- Search for unmarked foreground pixel, then fill it
- 3 versions:
  - *Recursive* (limited to small images)
  - *Depth-First*
  - *Breadth-First*
- All are called by the region labeling algorithm

```
RegionLabeling(I: binary image)
label = 2
Iterate over all image coordinates (u, v)
If I(u,v) = 1 Then
    FloodFill(I, u, v, label)
    label = label + 1
Return I
```

Depth-First flood filling algorithm

- Records unvisited elements in a stack
- Traverses tree of pixels using depth first method

```
FloodFill(I, u, v, label)
Create an empty stack S
push(S, (u,v))
While S is not empty Do
   (x, y) = pop(S)
   If (x, y) is within image boundary and I(x, y) = 1 Then
      I(x, y) = label
      push(S, (x+1, y))
      push(S, (x, y+1))
      push(S, (x, y-1))
      push(S, (x-1, y))
Return
```


Breadth-First flood filling algorithm

- Similar to depth-first version
- Uses a queue to store unvisited elements instead of a stack

```
FloodFill(I, u, v, label)
Create an empty queue Q
enqueue(Q, (u,v))
While Q is not empty Do
   (x, y) = dequeue(Q)
   If (x, y) is within image boundary and I(x, y) = 1 Then
      I(x, y) = label
      enqueue(Q, (x+1, y))
      enqueue(Q, (x, y+1))
      enqueue(Q, (x, y-1))
      enqueue(Q, (x-1, y))
Return
```


Finding regions: sequential region labeling

Sequential region labeling

2 steps:

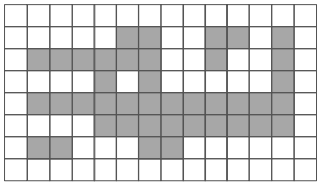
1. Preliminary labeling of image regions
2. Resolving cases where more than one label occurs

- Check following pixels depending on if we consider 4-connected or 8-connected neighbors

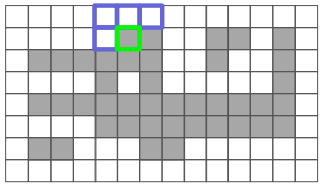
$\mathcal{N}_4(u,v)$ 

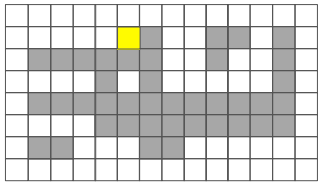
$\mathcal{N}_8(u,v)$ 

Sequential region labeling: propagating labels



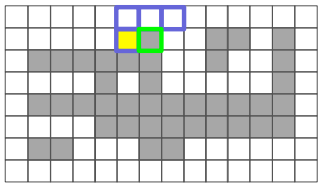
- First foreground pixel is found considering $\mathcal{N}_8(u,v)$
- All neighbors in $\mathcal{N}_8(u,v)$ are background pixels
- Then, assign pixel the first label (2)

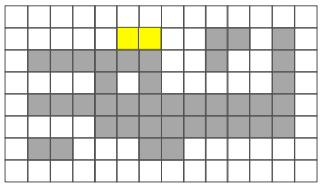




Sequential region labeling: propagating labels

- In next step, we propagate value 2 from neighborhood in $\mathcal{N}_8(u,v)$

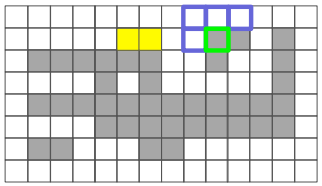


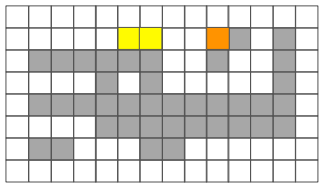


For only 1 different label, we use the same previous label

Sequential region labeling: propagating labels

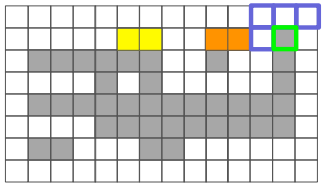
- Continue checking pixels as above
- At step below, there is no previous label in the neighborhood, so we increment the label (value 3 here)

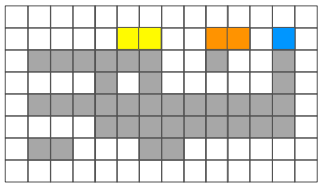




Sequential region labeling: propagating labels

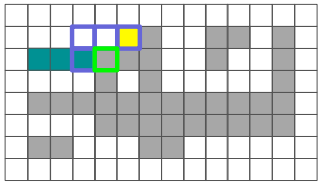
- Continue checking pixels as above
- At step below, there is no previous label in the neighborhood, so we increment the label again (value 4 here)

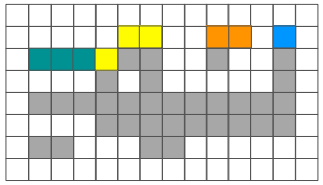




Sequential region labeling: propagating labels

- Continue checking pixels as above
- At step below, there are two neighboring pixels and they have different labels (2 and 5)
- One of these values is propagated, and collision is registered

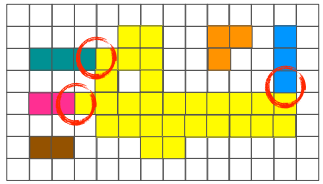


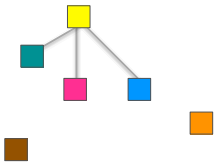


Sequential region labeling: label collisions

At the end of labeling step:

- All foreground pixels have been marked
- All collisions between labels have been registered
- Labels and collisions correspond to edges of undirected graph

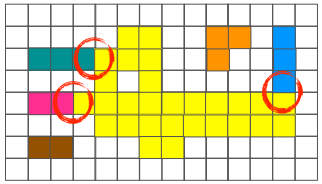


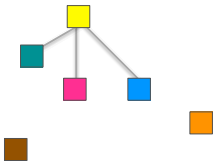


Undirected graph of collisions

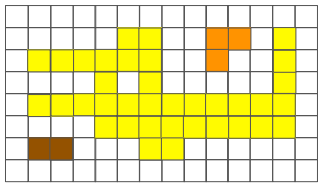
Sequential region labeling: resolving collisions

- Once all distinct labels within single region have been collected, assign labels of all pixels in region to be the same: e.g. assign all labels to have the smallest original label (label 2 here)





Undirected graph of collisions

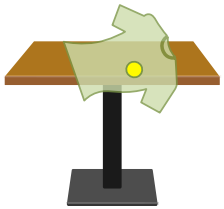


Properties of binary regions

Properties of binary regions

Human description of regions based on their properties:

- "An extra large khaki T-shirt with a yellow badge on an oak table"



- Not yet possible for computers to generate such descriptors
- Alternatively, computers can calculate mathematical properties of image or region to use for classification

Using features to classify images is fundamental part of pattern recognition

Types of features

- Shape features
- Geometric features
- Statistical shape properties
- Moment-based geometrical properties
- Topological Properties

Shape features

- Feature: numerical or qualitative value computable from values and coordinates of pixels in a given region (e.g. size which is the total number of pixels in region)

- Feature vector:
  - Combination of different features
  - Used as a "signature" of the region for classification or comparison

- Features should be:
  - Simple to calculate
  - Not affected by translation, rotations and scaling

Geometric features

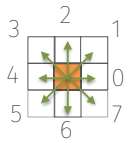
Region R of binary image = 2D distribution of foreground points within discrete plane

Geometric features: perimeter

- Perimeter: length of region's outer contour (R must be connected!)
- For 4-neighborhood, measured length of contour is larger than it's actual length
- Approximation for 8-connected chain code $C'_R = [C'_0, C'_1, \cdots, C'_{M-1}]$ is:

$$\text{Perimeter}(R) = \sum_{i=0}^{M-1} \text{length}(C'_i)$$

$$\text{with length}(C) = \begin{cases} 1 & \text{for } c = 0, 2, 4, 6 \\ \sqrt{2} & \text{for } c = 1, 3, 5, 7 \end{cases}$$



- Previous formula leads to overestimation (multiply by 0.95)

Geometric features: area

- Area: count image pixels that are within region

$$\text{Area}(R) = |R| = N$$

- Area of connected region (without holes) defined by M coordinate points can be estimated using the Gaussian area formula for polygons as:

$$\text{Area}(R) \simeq \frac{1}{2} \left| \sum_{i=0}^{M-1} (u_i \cdot v_{(i+1) mod M} - u_{(i+1) mod M} \cdot v_i \right|$$

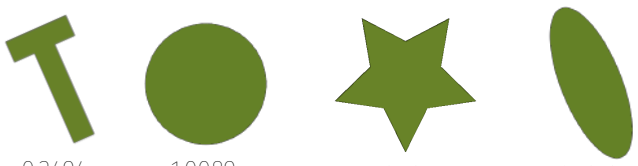
Geometric features: compactness and circularity

- Compactness is the relationship between a region's area and its perimeter.

$$\text{Compactness}(R) = \frac{\text{Area}(R)}{\text{Perimeter}^2(R)}$$

- It is invariant to translation, rotation and scaling
- Ratio has value $1/4\pi$ when applied to circular region

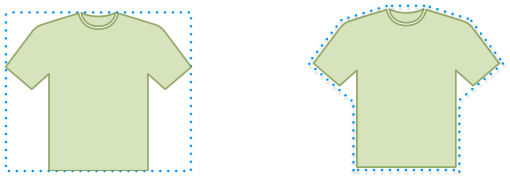
$$\text{Circularity}(R) = 4\pi \cdot \frac{\text{Area}(R)}{\text{Perimeter}^2(R)}$$



Geometric features: bounding box and convex hull

- Bounding box: mininimal axis-parallel rectangle that encloses all points in R

- Convex hull: smallest polygon that fits all points in R
  - Convexity: relationship between length of convex hull and perimeter of R
  - Density: ratio between area of the region and area of the convex hull



Statistical shape properties

- Points may be viewed as being *statistically distributed* in 2D space
- Can be applied to regions that are not connected
- E.g. centroid of a binary region is the arithmetic mean of all (x, y) coordinates in the region

$$\bar{x} = \frac{1}{|R|} \sum_{(u,v)\in R} u \quad \text{and} \quad \bar{y} = \frac{1}{|R|} \sum_{(u,v)\in R} v$$

Statistical shape properties

- Centroid is a specific case of more general concept of moments
- Ordinary moment of the order p, q for a discrete function $I(u, v)$ is:

$$m_{pq} = \sum_{(u,v)\in R} I(u, v) \cdot u^p v^q$$

- Area of a binary region is the zero-order moment

$$\text{Area}(R) = m_{00}(R) = \sum_{(u,v)\in R} 1 = \sum_{(u,v)\in R} u^0 v^0 = |R|$$

Statistical shape properties

- Similarly, centroid can be expressed as:

$$\bar{x} = \frac{1}{|R|} \cdot \sum_{(u,v)\in R} u^1 v^0 = \frac{m_{10}(R)}{m_{00}(R)}$$
$$\bar{y} = \frac{1}{|R|} \cdot \sum_{(u,v)\in R} u^0 v^1 = \frac{m_{01}(R)}{m_{00}(R)}$$

Statistical shape properties

- Central moments:
  - They use the region's centroid as reference to calculate translation-invariant region features
  - The origin is shifted to the region's centroid
- Order p, q central moments are calculated as:

$$\mu_{pq} = \sum_{(u,v)\in R} I(u, v) \cdot (u - \bar{x})^p (v - \bar{y})^q$$

- For a binary image $(I(u, v) = 1)$:

$$\mu_{pq} = \sum_{(u,v)\in R} (u - \bar{x})^p (v - \bar{y})^q$$

Statistical shape properties

- Values of central moments depends on:
  - Distances of all region points to centroid
  - Absolute size of the region
- Size-invariant features can be obtained by scaling central moments uniformly by some factor k

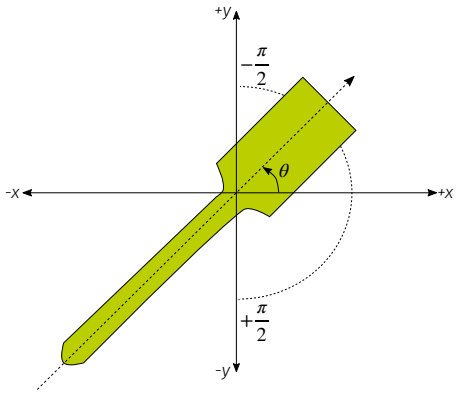
$$k^{(p+q+2)}$$

- Normalized central moments:

$$\bar{\mu}_{pq}(R) = \mu_{pq} \cdot \left(\frac{1}{\mu_{00}(R)}\right)^{(p+q+2)/2} \quad \text{for } (p+q) \geq 2$$

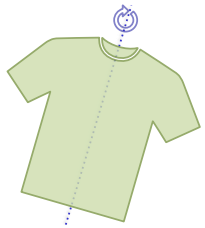
Moment-based geometrical properties

- Several interesting features can be derived from moments
- Orientation: describes direction of major axis that runs through centroid and along the widest part of the region



Moment-based geometrical properties

- Orientation is also called the major axis of rotation since rotating the region around the major axis requires the least effort



- Direction of major axis can be calculated from central moments:

$$\theta_R = \frac{1}{2}\tan^{-1}\left(\frac{2\mu_{11}(R)}{\mu_{20}(R) - \mu_{02}(R)}\right)$$

range [-90, 90]

Moment-based geometrical properties

- Plot orientation by using the parametric equation of a line:

$$\mathbf{x} = \bar{\mathbf{x}} + \lambda \cdot \mathbf{x_d} = \begin{pmatrix}\bar{x}\\ \bar{y}\end{pmatrix} + \lambda \cdot \begin{pmatrix}\cos(\theta_R)\\ \sin(\theta_R)\end{pmatrix}$$

start point — direction vector

-Orientation vector $\mathbf{x_d}$ can be computed as:

$$\mathbf{x_d} = \cos(\theta_\mathbf{R}) = \begin{cases} 0 & \text{for } a = b = 0 \\ \left[\frac{1}{2}\left(1 + \frac{b}{\sqrt{a^2+b^2}}\right)\right]^{\frac{1}{2}} & \text{otherwise,} \end{cases}$$

$$\mathbf{y_d} = \sin(\theta_\mathbf{R}) = \begin{cases} 0 & \text{for } a = b = 0 \\ \left[\frac{1}{2}\left(1 - \frac{b}{\sqrt{a^2+b^2}}\right)\right]^{\frac{1}{2}} & \text{for } a \geq 0 \\ -\left[\frac{1}{2}\left(1 - \frac{b}{\sqrt{a^2+b^2}}\right)\right]^{\frac{1}{2}} & \text{for } a < 0 \end{cases}$$

where $a = 2\mu_{11}(R)$ and $b = \mu_{20}(R) - \mu_{02}(R)$

Moment-based geometrical properties

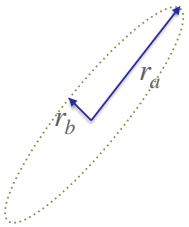
- Eccentricity: ratio of lengths of major axis and minor axis
- It expresses how elongated the region is

$$\text{Eccentricity}(R) = \frac{\mu_{20} + \mu_{02} + \sqrt{(\mu_{20} + \mu_{02})^2 + 4\mu_{11}^2}}{\mu_{20} + \mu_{02} - \sqrt{(\mu_{20} - \mu_{02})^2 + 4\mu_{11}^2}} = \frac{a_1}{a_2}$$

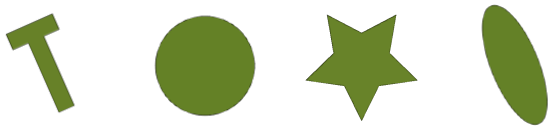
- Lengths of major and minor axis are:

$$r_a = 2 \cdot \left(\frac{2a_1}{|R|} \right)^{\frac{1}{2}}$$

$$r_b = 2 \cdot \left(\frac{2a_2}{|R|} \right)^{\frac{1}{2}}$$



Example



Area	54903	34822	41743	33433
Orientation	90°	14.84°	-60.92°	-68°
Circularity	0.3494	1.0089	0.43	0.7218
Minor axis length	201	210	250	126
Major axis length	571	210	250	336
Eccentricity	0.9356	0.0265	0.0354	0.9269

Moment-based geometrical properties

- Normalized central moments are invariant to translation or uniform scaling, but are modified by rotation
- Hu's moments (7 combinations of normalized central moments) are invariant to translation, scaling and rotation

$$H_1 = \bar{\mu}_{20} + \bar{\mu}_{02}$$

$$H_2 = (\bar{\mu}_{20} - \bar{\mu}_{02})^2 + 4\bar{\mu}_{11}^2$$

$$H_3 = (\bar{\mu}_{30} - 3\bar{\mu}_{12})^2 + (3\bar{\mu}_{21} - \bar{\mu}_{03})^2$$

$$H_4 = (\bar{\mu}_{30} + \bar{\mu}_{12})^2 + (\bar{\mu}_{21} + \bar{\mu}_{03})^2$$

$$H_5 = (\bar{\mu}_{30} - 3\bar{\mu}_{12}) \cdot (\bar{\mu}_{30} + \bar{\mu}_{12}) \cdot [(\bar{\mu}_{30} + \bar{\mu}_{12})^2 - 3(\bar{\mu}_{21} + \bar{\mu}_{03})^2] + (3\bar{\mu}_{21} - \bar{\mu}_{03}) \cdot (\bar{\mu}_{21} + \bar{\mu}_{03}) \cdot [3(\bar{\mu}_{30} + \bar{\mu}_{12})^2 - (\bar{\mu}_{21} + \bar{\mu}_{03})^2]$$

$$H_6 = (\bar{\mu}_{20} - \bar{\mu}_{02}) \cdot [(\bar{\mu}_{30} + \bar{\mu}_{12})^2 - (\bar{\mu}_{21} + \bar{\mu}_{03})^2] + 4\bar{\mu}_{11} \cdot (\bar{\mu}_{30} + \bar{\mu}_{12}) \cdot (\bar{\mu}_{21} + \bar{\mu}_{03})$$

$$H_7 = (3\bar{\mu}_{21} - \bar{\mu}_{03}) \cdot (\bar{\mu}_{30} + \bar{\mu}_{12}) \cdot [(\bar{\mu}_{30} + \bar{\mu}_{12})^2 - 3(\bar{\mu}_{21} + \bar{\mu}_{03})^2] + (3\bar{\mu}_{12} - \bar{\mu}_{30}) \cdot (\bar{\mu}_{21} + \bar{\mu}_{03}) \cdot [3(\bar{\mu}_{30} + \bar{\mu}_{12})^2 - (\bar{\mu}_{21} + \bar{\mu}_{03})^2]$$

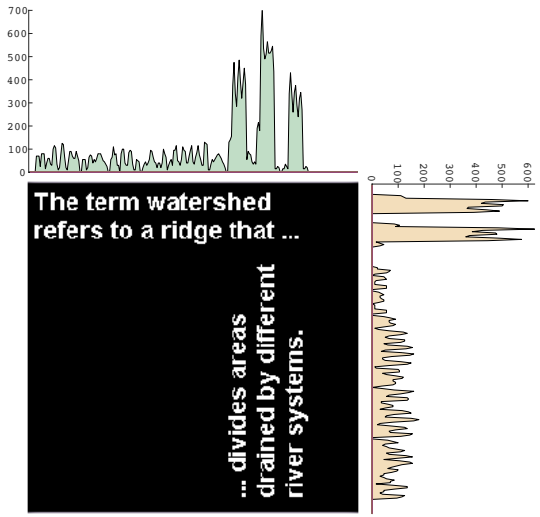
Projections

- Image projections are 1D representations of image contents
- Vertical and horizontal projections of image $I(u, v)$ are defined as

$$P_{\text{horizontal}}(v_0) = \sum_{u=0}^{M-1} I(u, v_0) \quad \text{for} \quad 0 < v_0 < N$$

$$P_{\text{vertical}}(u_0) = \sum_{v=0}^{N-1} I(u_0, v) \quad \text{for} \quad 0 < u_0 < M$$

Example of projections



Topological properties

- Define the structure of a region
- Invariant under strong image transformations
- Number of holes is simple, robust feature
- Euler number: number of connected regions – number of holes

$$N_E(R) = N_{CC}(R) - N_H(R)$$

- Topological features often combined with numerical features

Binary regions in Python

- `measure` module from **scikit-image** supports most of methods we used here.
- `filters`, `segmentation` and `morphology` modules have also some interesting functions.